

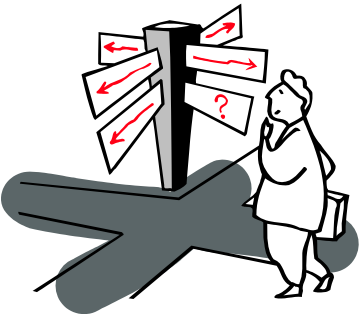


自然语言处理

Natural Language Processing

Chapter 4

循环神经网络

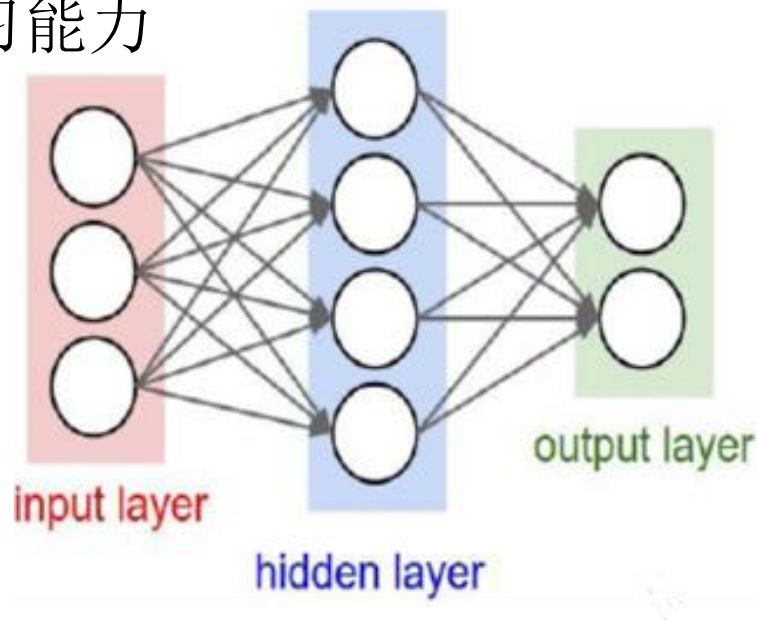


Outline

- 循环神经网络(RNN)
- RNN的反向传播算法
- RNN主流变体(LSTM, GRU)
- NLP中的案例分析

回顾：前馈神经网络

- 全连接
- 没有同层节点连接，没有反馈连接
- 每一层输出只由当前输入影响
- 减弱了模型学习能力

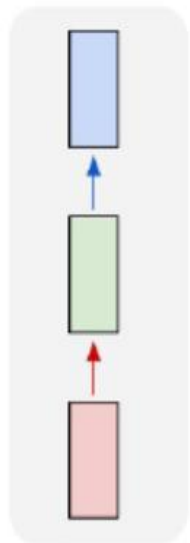


Vanilla NN

NLP中的序列任务

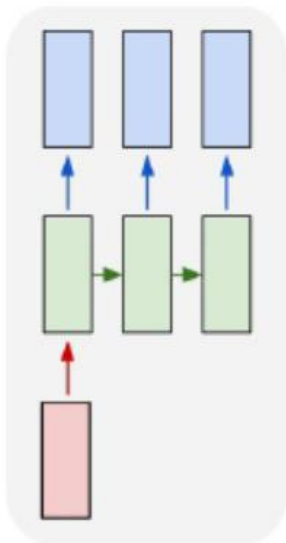
- 按照输入输出的token数量:

one to one



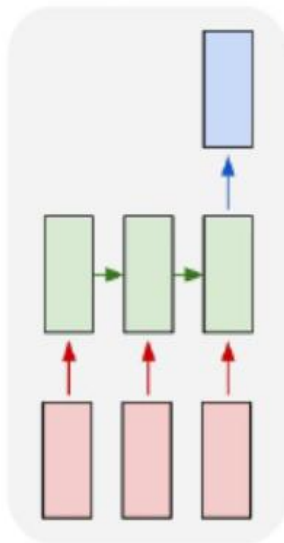
一对一

one to many



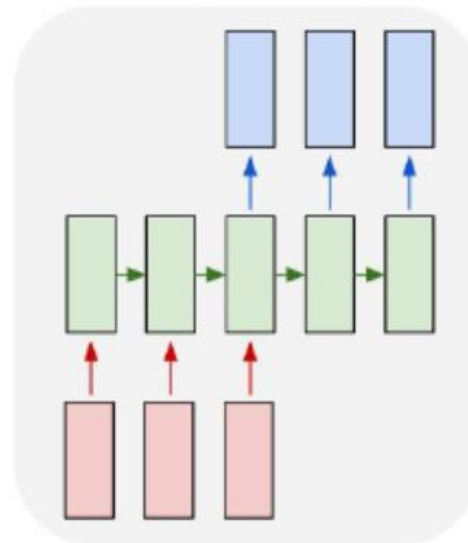
一对多

many to one



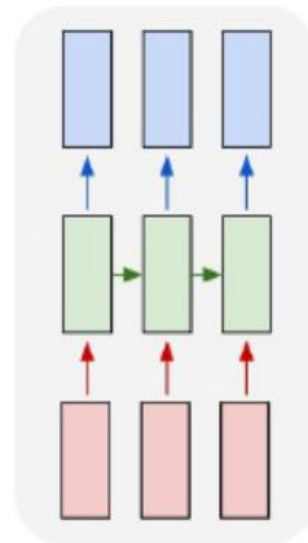
多对一

many to many



异步多对多

many to many



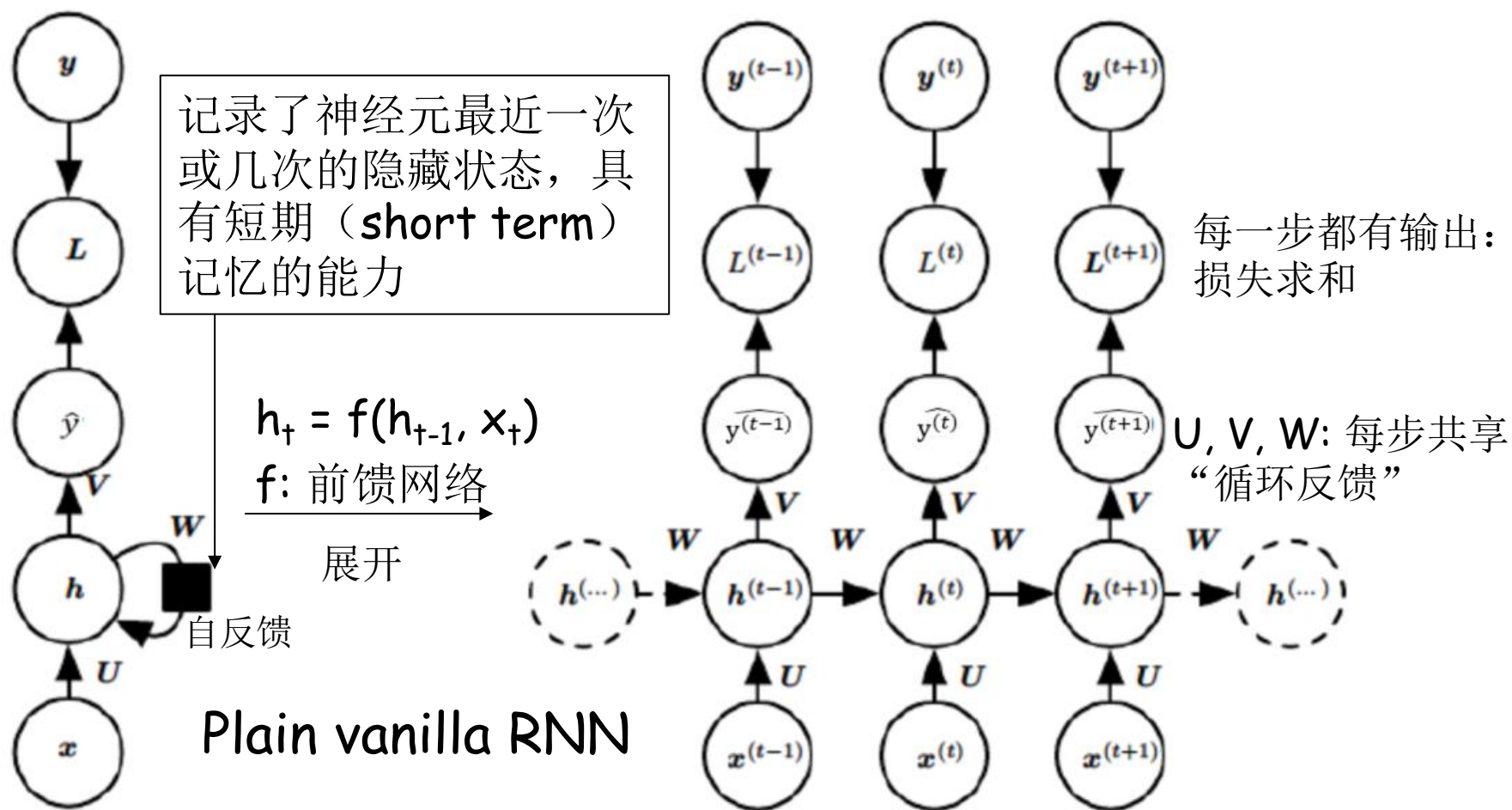
同步多对多

输入长度不固定，前馈神经网络难以处理

循环神经网络

- 循环神经网络(Recurrent Neural Network, RNN. 1986) 是一系列适合处理序列数据的神经网络模型。
- 循环：当前（处理第 t 个token）输出是先前时间步输出的函数，即前面输出的结果会加入到当前步骤的计算，且计算方式、参数相同
- 前馈网络的权重参数不共享 vs RNN共享权重参数
- 前馈网络接受固定长度数据 vs RNN可以处理变长数据
- 如果进行批处理 ($\text{batch_size} > 1$)，则需要固定batch内长度

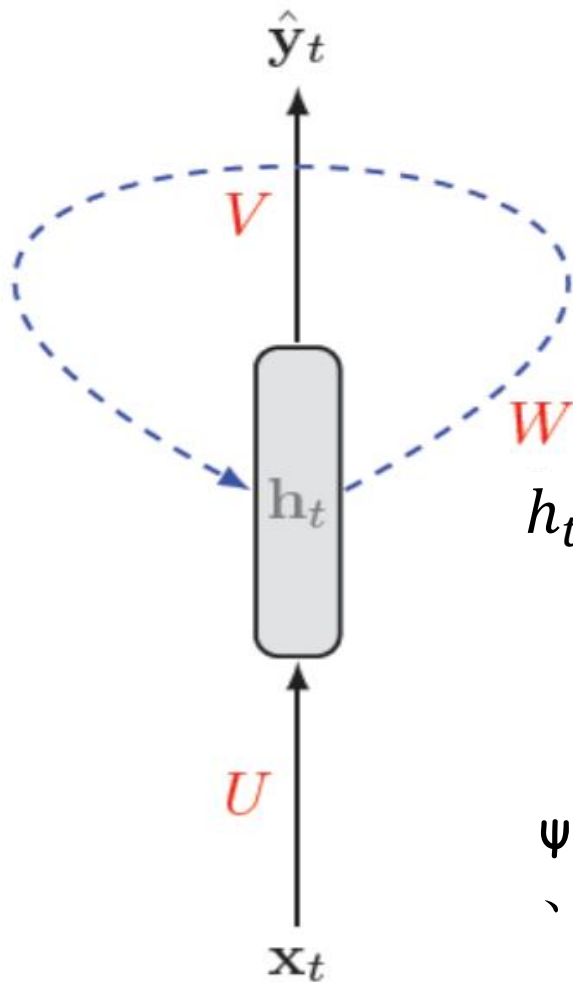
循环神经网络



RNN可以看作在时间维度上权值共享的神经网络，更加符合生物神经网络的结构

$x^{(t)}$: 文本中第 t 个词

反馈连接



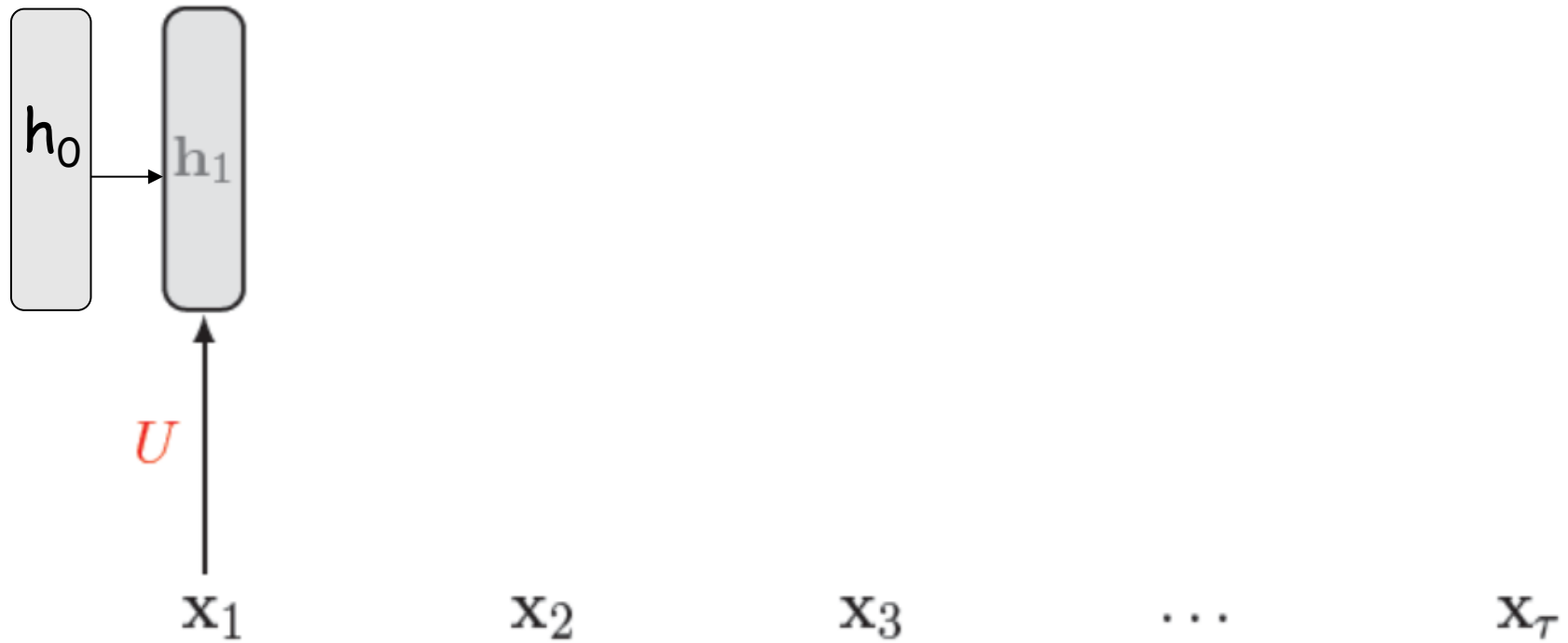
$$\hat{\mathbf{y}}_t = \Phi(\mathbf{V}h_t + c)$$

$$h_t = \psi(\mathbf{U}x_t + \mathbf{W}h_{t-1} + b)$$

ψ 和 Φ 是非线性函数，e.g. ψ 是tanh、relu函数等， Φ 是softmax函数

反馈的展开

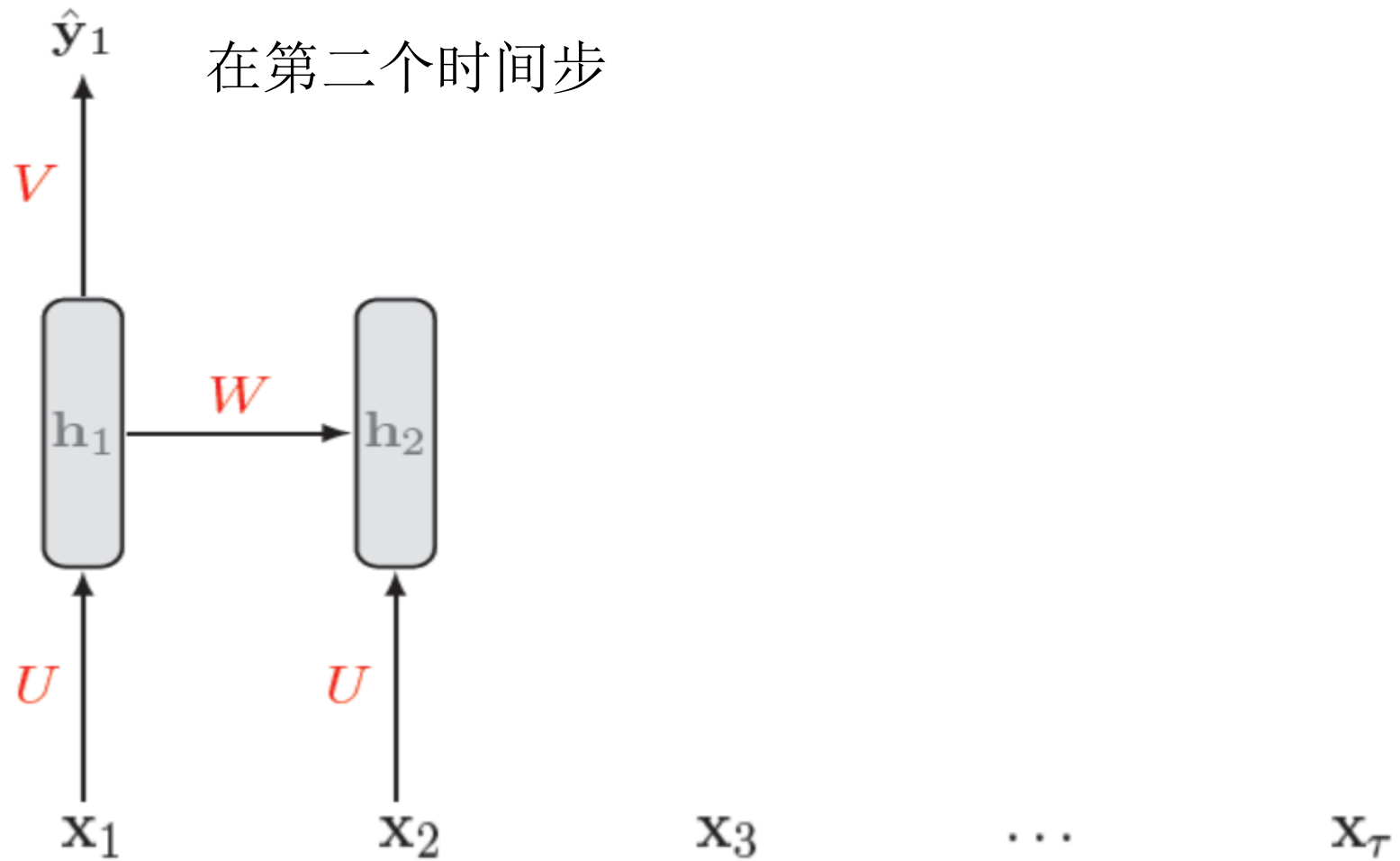
在第一个时间步



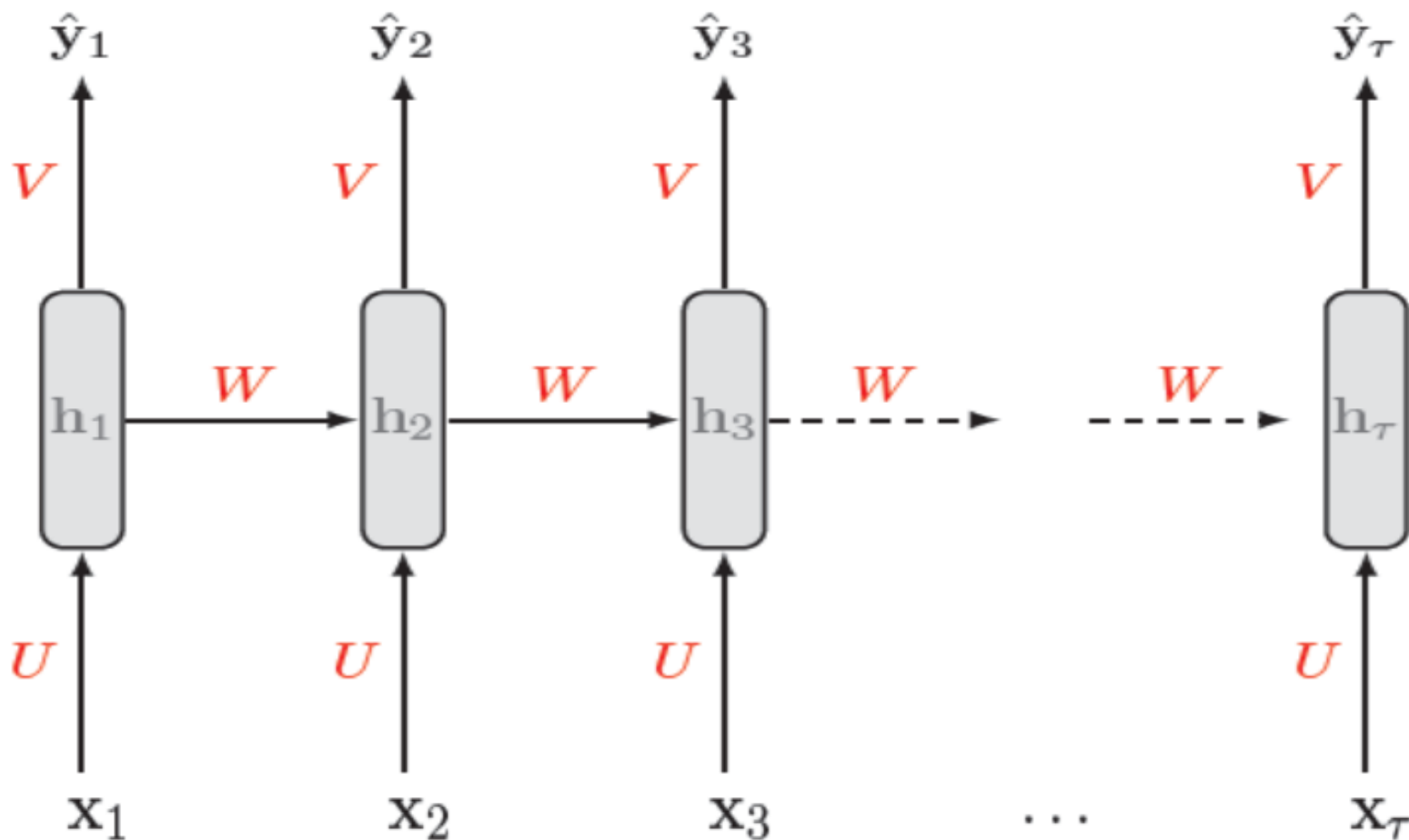
反馈的展开



反馈的展开



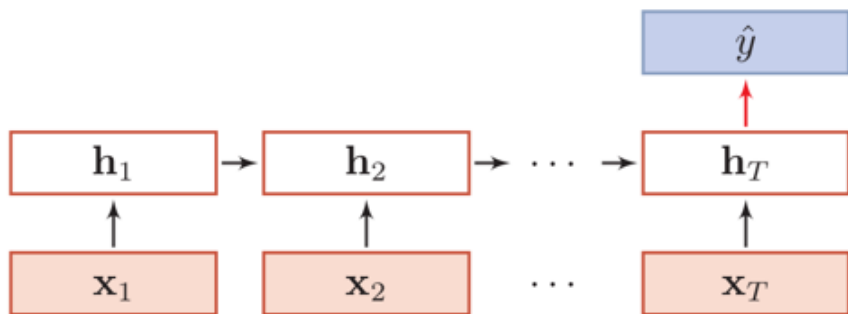
反馈的展开



每个时间步重用相同的权重矩阵：矩阵规模不随序列增长而变大

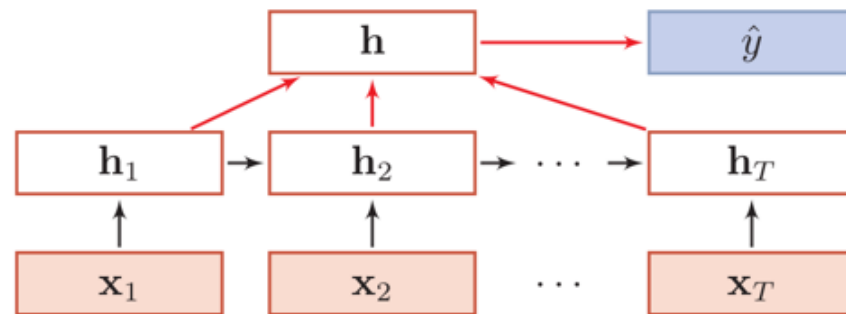
RNN的使用

- 多对一
- 场景：文本主题分类，自动作文评分...



(a) 正常模式

取最后一个时间步的隐藏状态
作为特征



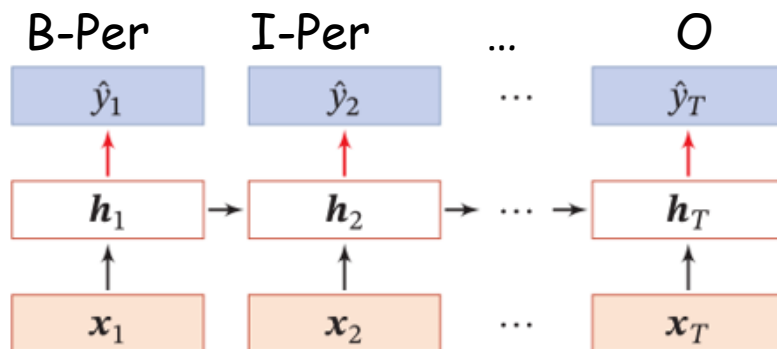
(b) 按时间进行平均采样模式

取所有时间步的隐藏状态的平
均作为特征

RNN的使用

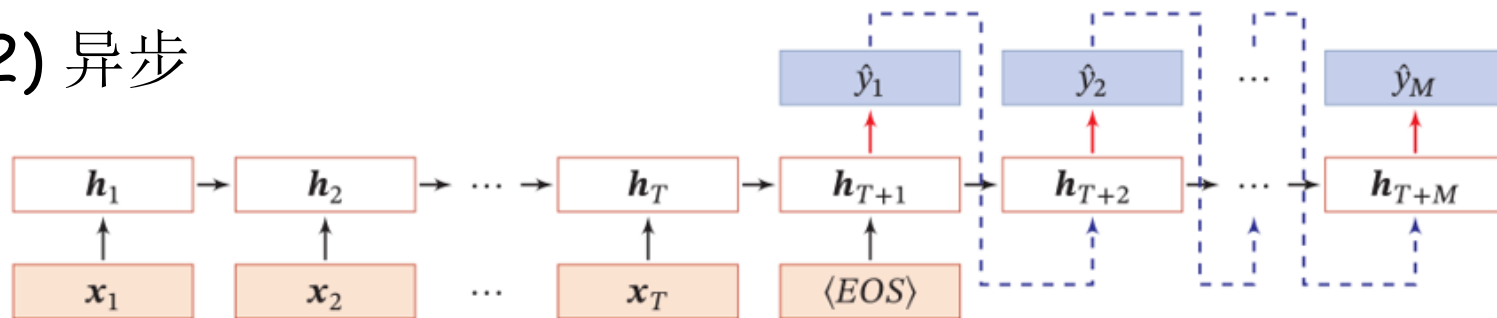
- 多对多

(1) 同步



场景：命名实体识别NER, 词性标注...

(2) 异步



场景：机器翻译，自动问答...（典型seq2seq任务）

RNN的前向传播

- RNN中的前向传播是从左往右(默认句首到句尾)进行的
- 公式:

$$z_t = b + Wh_{t-1} + Ux_t \quad \text{第} t \text{时刻隐藏层的净输入}$$

$$h_t = \tanh z_t$$

$$o_t = c + Vh_t$$

$$\hat{y}_t = \text{softmax}(o_t) \quad \text{第} t \text{时刻的输出}$$

RNN的前向传播

- RNN中的损失 L 是所有时间步损失的和 (具体看任务的输出)
- 假设时间步 t 的真实输出和预测输出是 y_t 和 \hat{y}_t , L_t 为交叉熵损失, 则对于某一个样本:

$$L = L(\{x_1, \dots, x_T\}, \{y_1, \dots, y_T\}) = - \sum_{t=1}^T y_t \log(\hat{y}_t)$$

- 如果 L_t 取负对数似然, 则: $L = -\log \prod_t \hat{y}_t^{y_t}$ \equiv

- 观测: RNN的前向传播过程时间复杂度是 $O(t)$, 和序列长度成正比。无法并行计算。

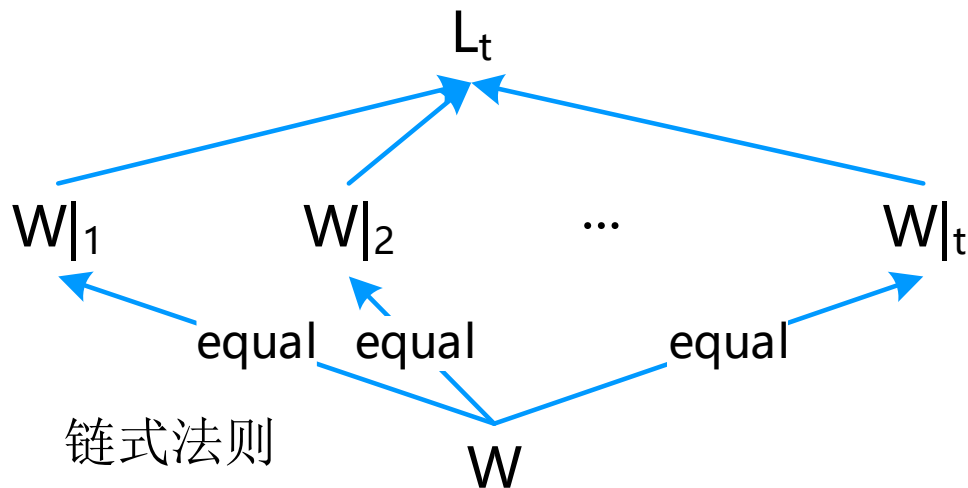


Outline

- 循环神经网络(RNN)
- 反向传播算法
- RNN主流变体(LSTM, GRU)
- 案例分析

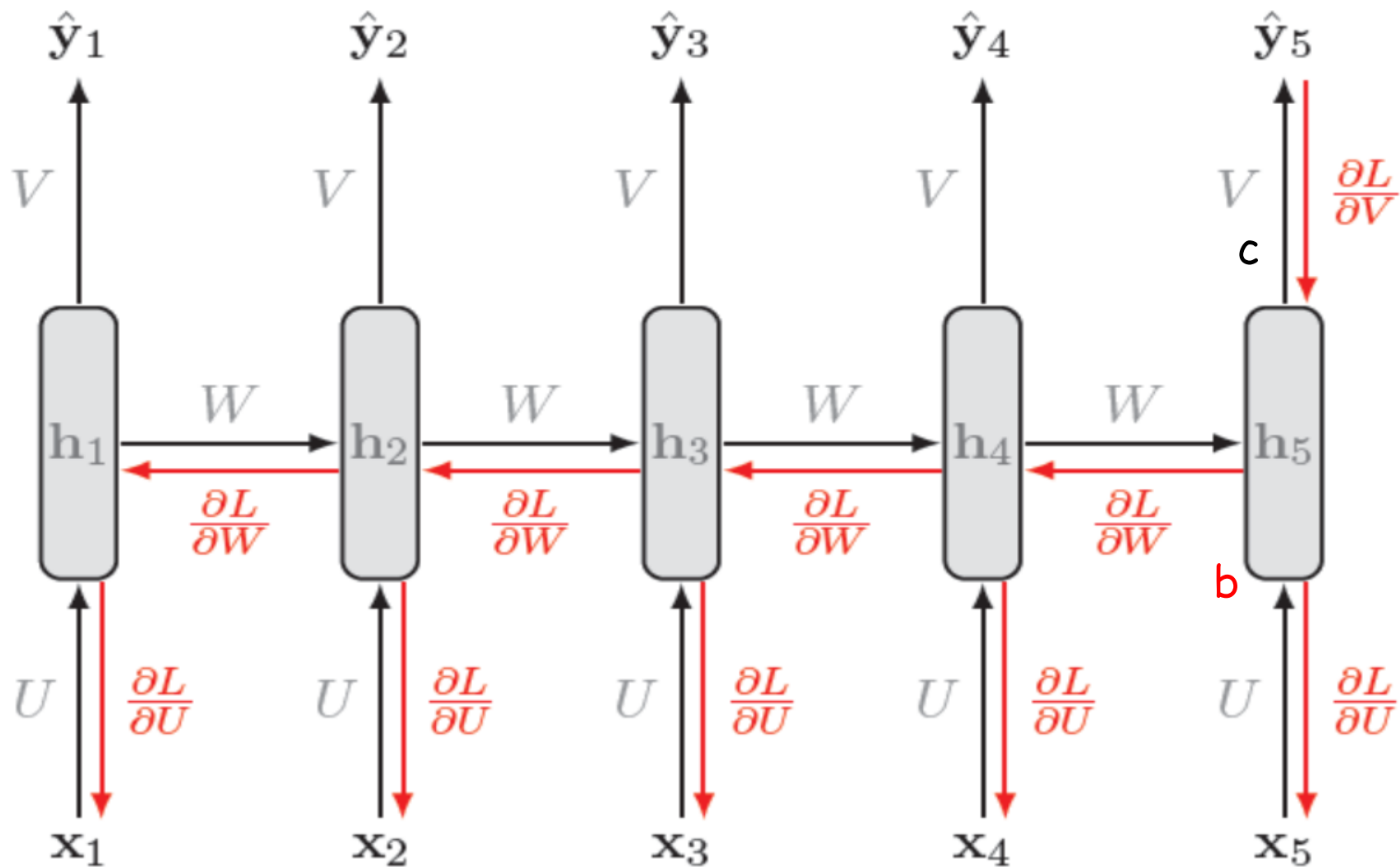
RNN的反向传播

- 对于所有参数，要计算： $\nabla_V L, \nabla_W L, \nabla_U L, \nabla_c L, \nabla_b L$
- 在展开的RNN网络上进行反向传播
- E.g. 对于权重矩阵 W



- 反向传播从右往左，基于时间步反向传播
- Backpropagation through time, 简称BPTT
- 时间复杂度为 $O(t)$

RNN的反向传播



梯度计算

$$L = - \sum_t y_t \log(\hat{y}_t)$$

$$z_t = b + W h_{t-1} + U x_t$$

$$h_t = \tanh z_t$$

$$o_t = c + V h_t$$

$$\hat{y}_t = \text{softmax}(o_t)$$

o_t 只参与 \hat{y}_t 的计算

交叉熵求导

$$\nabla_{o_t} L = \frac{\partial L_t}{\partial o_t} = \frac{\partial y_t \log(\hat{y}_t)}{\partial o_t} = \hat{y}_t - y_t$$

矩阵和向量求导

c, V 与 h_t 无关, 因此梯度不受 $t+1$ 影响

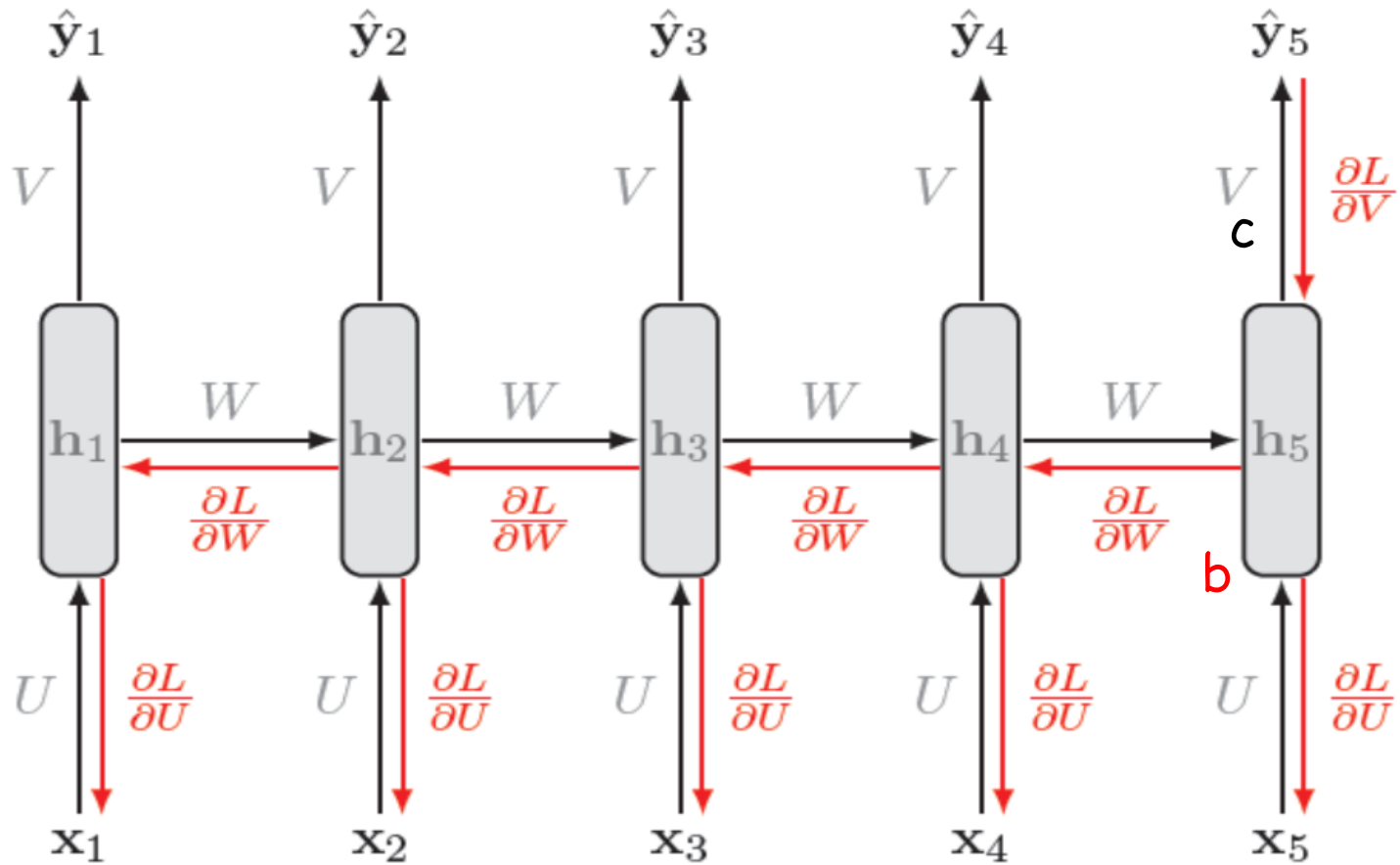
$$\nabla_c L = \frac{\partial L}{\partial o_t} \boxed{\frac{\partial o_t}{\partial c}} \xrightarrow{\text{I (单位矩阵)}}$$

$$\begin{aligned} \nabla_V L &= \frac{\partial L}{\partial o_t} \boxed{\frac{\partial o_t}{\partial V}} \xrightarrow{\quad} h_t^T \\ &= (\hat{y}_t - y_t) * h_t^T \end{aligned}$$

梯度计算

- V, c 的偏导数只与当前时间步 t 有关
- W, U, b 的偏导数由 t 和 $t+1$ 步的损失决定

$$\frac{\partial L}{\partial V} = (\hat{y}_t - y_t) * h_t^T$$



梯度计算

$$z_t = b + Wh_{t-1} + Ux_t$$

$$h_t = \tanh z_t$$

$$o_t = c + Vh_t$$

$$\hat{y}_t = \text{softmax}(o_t)$$

$$\nabla_W L = \sum_t \frac{\partial L}{\partial z_t} \frac{\partial z_t}{\partial W} \rightarrow (h_{t-1})^T$$

$$\nabla_{z_t} L = \sum_t \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial z_t}$$

$$\frac{\partial h_t}{\partial z_t} = \begin{bmatrix} \frac{\partial h_t^1}{\partial z_t^1} & \dots & \frac{\partial h_t^1}{\partial z_t^m} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_t^m}{\partial z_t^1} & \dots & \frac{\partial h_t^m}{\partial z_t^m} \end{bmatrix}$$

dim = m

$$\frac{\partial h_t}{\partial z_t} = \begin{bmatrix} \frac{\partial h_t^1}{\partial z_t^1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{\partial h_t^m}{\partial z_t^m} \end{bmatrix}$$

$$\frac{\partial h_t}{\partial z_t} = \text{diag}(1 - (\tanh z_t)^2)$$

梯度计算

$$z_t = b + Wh_{t-1} + Ux_t$$

$$h_t = \tanh z_t$$

$$o_t = c + Vh_t$$

$$\hat{y}_t = \text{softmax}(o_t)$$

$$\nabla_W L = \sum_t \frac{\partial L}{\partial z_t} \frac{\partial z_t}{\partial W} \quad \nabla_{z_t} L = \sum_t \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial z_t}$$

$$\nabla_{h_t} L = \sum_t \frac{\partial L}{\partial o_t} \frac{\partial o_t}{\partial h_t} \quad \text{No!} \quad + \underbrace{\left(\frac{\partial h_{t+1}}{\partial h_t} \right)^T (\nabla_{h_{t+1}} L)}_{(W)^T (\nabla_{h_{t+1}} L) \tanh'(z_{t+1})}$$

$$\nabla_W L = \sum_t \text{diag}(1 - (h_t)^2) \frac{\partial L}{\partial h_t} (h_{t-1})^T$$

$$\frac{\partial L}{\partial h_t} = \begin{cases} V^T (\hat{y}_t - y_t) & \text{最后一步} \\ V^T (\hat{y}_t - y_t) + (W)^T (\nabla_{h_{t+1}} L) (1 - h_{t+1}^2) & \text{其他} \end{cases}$$

梯度计算

$$z_t = b + Wh_{t-1} + Ux_t$$

已知: $\nabla_W L = \sum_t \text{diag}(1 - (h_t)^2) \frac{\partial L}{\partial h_t} (h_{t-1})^T$

则: $\nabla_U L = \sum_t \text{diag}(1 - (h_t)^2) \frac{\partial L}{\partial h_t} (x_t)^T$

$$\nabla_b L = \sum_t \text{diag}(1 - (h_t)^2) \frac{\partial L}{\partial h_t}$$

结论: **RNN**的**BPTT**与前馈神经网络中使用的标准反向传播算法是完全相同的。关键的区别是, 需要在每一个时间步展开, 对它们的梯度进行求和。



RNN反向传播整理

- BPTT算法将RNN看作一个展开的多层前馈网络

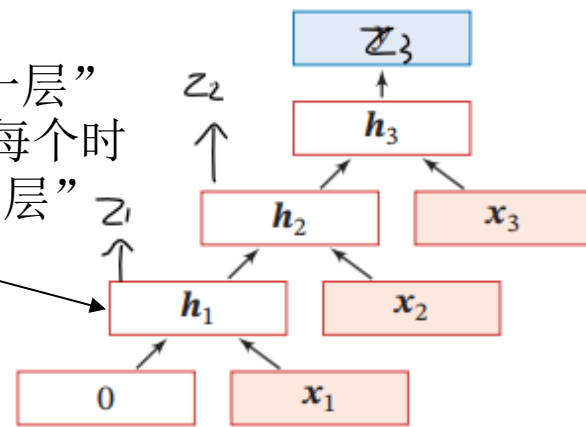
$$z_t = b + Wh_{t-1} + Ux_t$$

$$h_t = f(z_t)$$

$$\hat{y}_t = g(h_t)$$

$$L = \sum_{t=1}^T L_t \quad L_t = L(y_t, g(h_t))$$

前馈网络“每一层”
对应RNN的“每个时
刻”。“3个隐层”



RNN反向传播整理

- 定义 $\delta_{t,k} = \frac{\partial L_t}{\partial z_k}$ 为第 t 时刻的损失对第 k 时刻隐层的净输入 z_k 的导数

$$= \frac{\partial h_k}{\partial z_k} \frac{\partial z_{k+1}}{\partial h_k} \frac{\partial L_t}{\partial z_{k+1}} = \overset{\text{局部误差 local error}}{\text{diag}(f'(z_k)) W^T} \delta_{t,k+1}$$

- 参数梯度:

$$\frac{\partial L}{\partial U} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} x_k^T$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} h_{k-1}^T$$

$$\frac{\partial L}{\partial b} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k}$$

local error & local input

$$z_t = b + W h_{t-1} + U x_t$$

$$h_t = f(z_t)$$

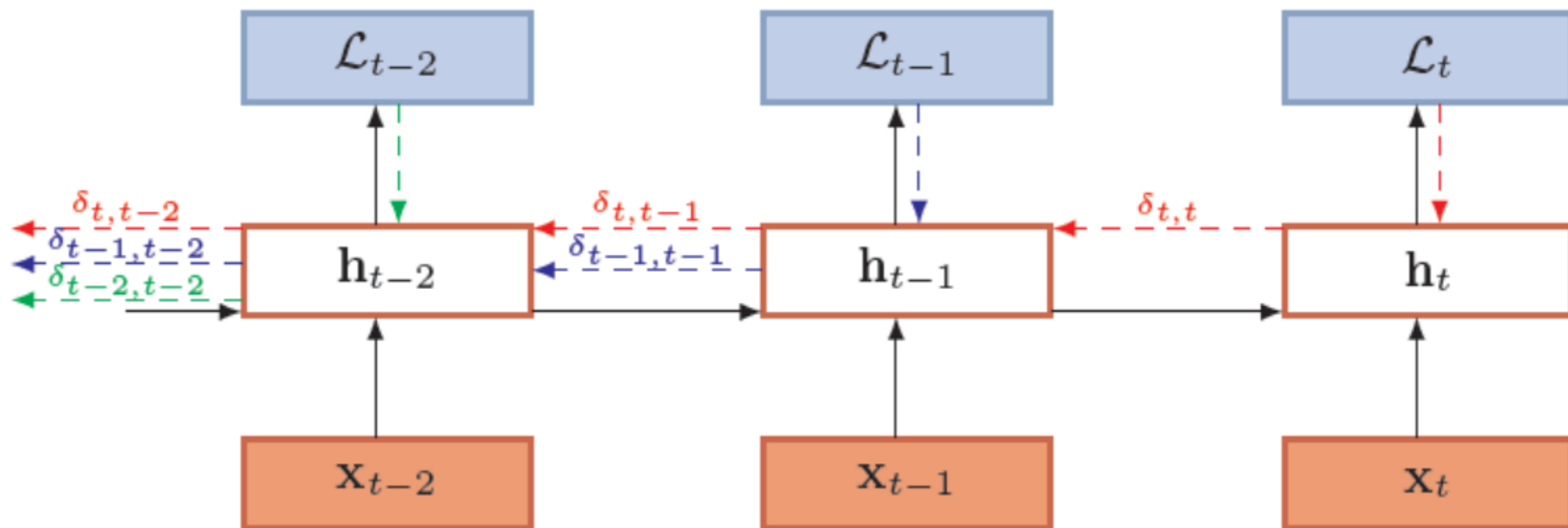
$$\hat{y}_t = g(h_t)$$

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

$$L_t = L(y_t, g(h_t))$$

$$\delta_{t,k} = \frac{\partial L_t}{\partial z_k}$$

RNN反向传播整理



$\delta_{t,t}$ = 第 t 时刻的损失对第 t 时刻的净输入 z_t 的导数

RNN参数更新

e.g. 梯度下降法

$$V := V - \lambda \frac{\partial L}{\partial V}$$

$$W := W - \lambda \frac{\partial L}{\partial W}$$

$$U := U - \lambda \frac{\partial L}{\partial U}$$

$$b := b - \lambda \frac{\partial L}{\partial b}$$

$$c := c - \lambda \frac{\partial L}{\partial c}$$

- 学习率. 通常是一个超参
- 过小: 模型训练缓慢
- 过大: 可能会使损失函数直接越过全局最优点, 容易发生梯度爆炸, 模型难以收敛

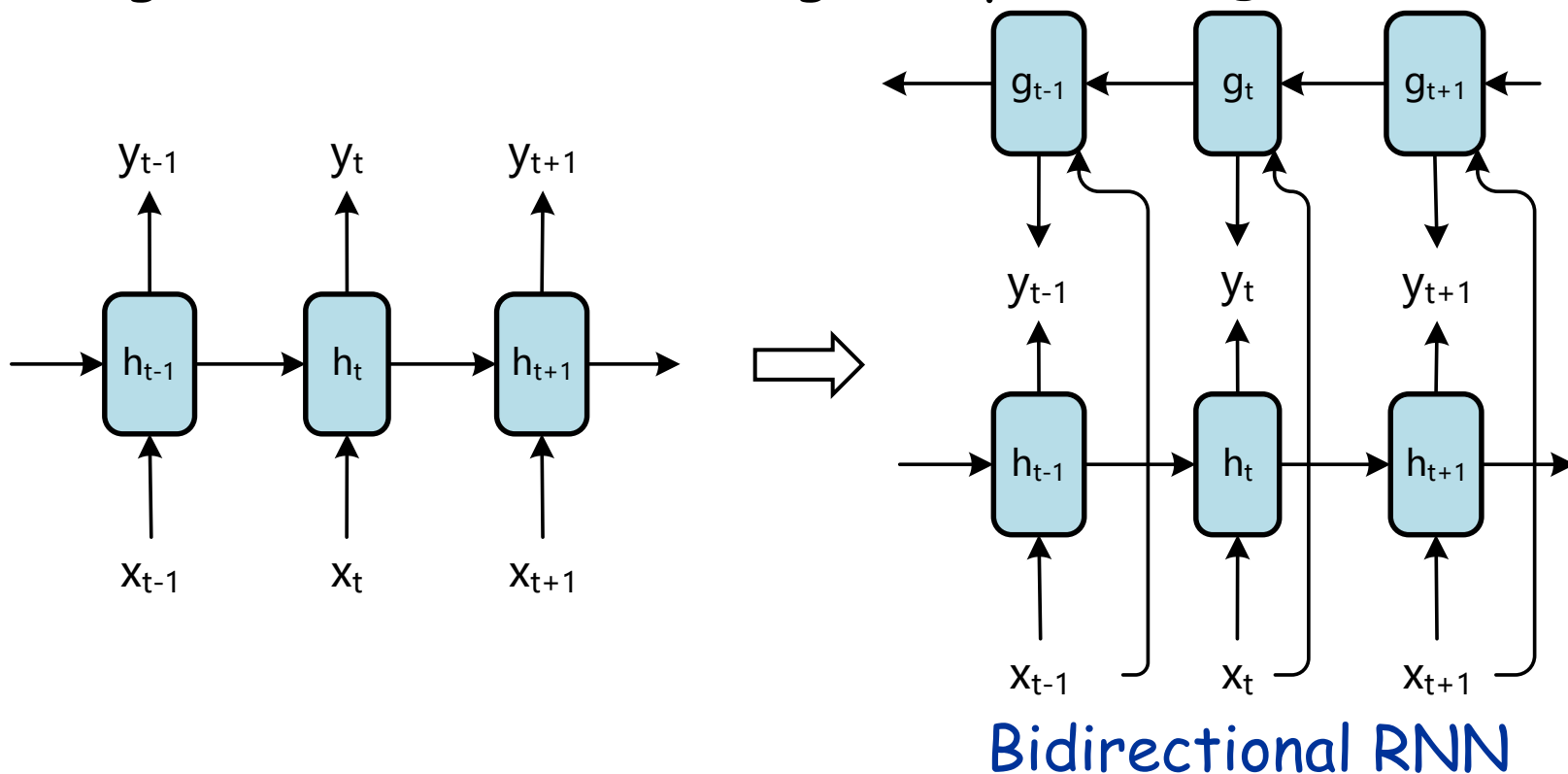
RNN参数更新

- 学习率可以设为常数
- 也可以选择动态学习率
 - 初始范围： $10^{-2} \sim 10^{-4}$ （与优化器选择有关）
 - 学习率 λ 调整方法：
 - 1) 离散下降**discrete staircase**. 每经过 k 个epoch, λ 减半
 - 2) 指数下降**exponential decay**. $\lambda = \alpha^{\text{epoch_num}} \lambda$
 - 3) 分数减缓 **1/t decay**. $\lambda = \frac{\lambda}{1 + \text{decay_rate} * \text{epoch_num}}$
 - 4) . . .

双向RNN

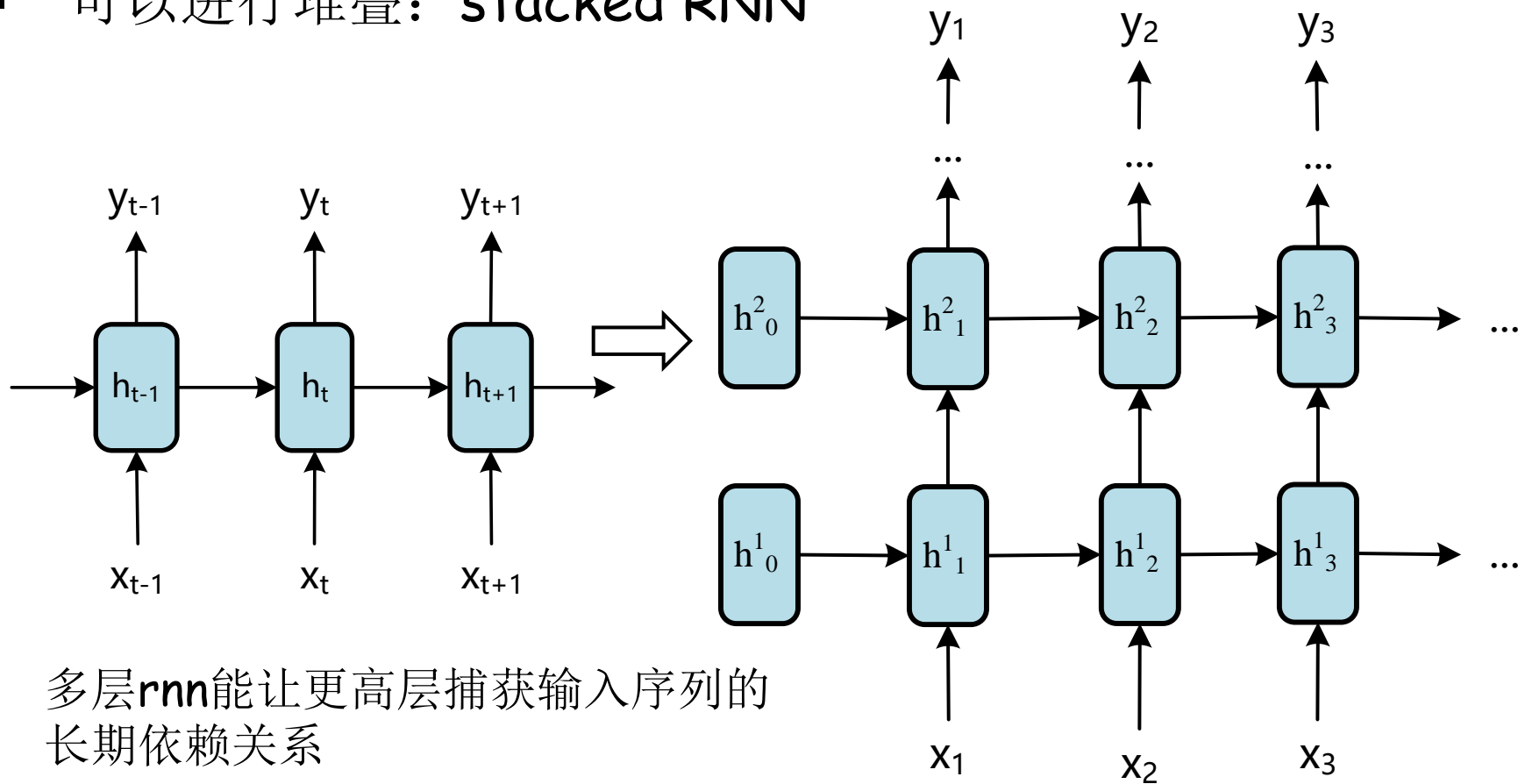
- 单向RNN只能获取过去时刻的信息
- 有时候 $y^{(t)}$ 的计算需要使用未来时刻的信息

E.g. 词性标注, $x = \text{Good night, my sweet girl}$



多层RNN

- 单层RNN只有一个隐藏层
- 可以进行堆叠: **stacked RNN**

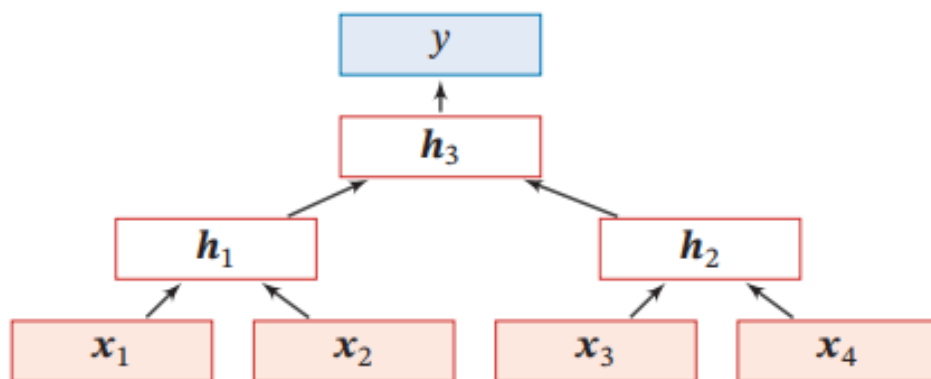


多层rnn能让更高层捕获输入序列的长期依赖关系

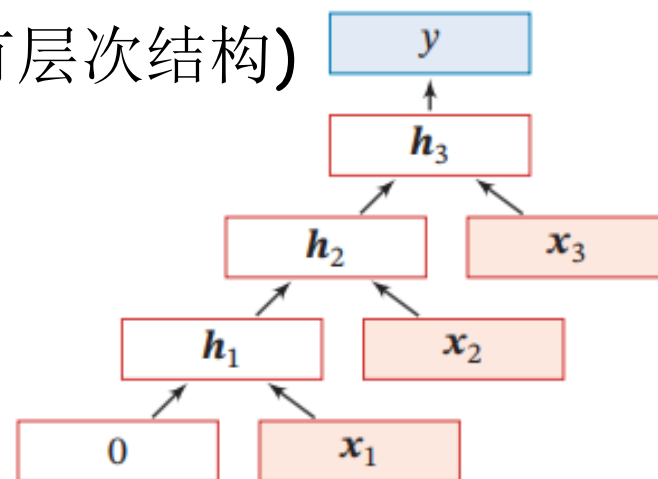
Stacked RNNs

RNN同名

- Recursive Neural Network递归神经网络
- 是一个树形的RNN
- 可用于建模句子语义(自然语言具有层次结构)



(a) 一般结构



(b) 退化结构

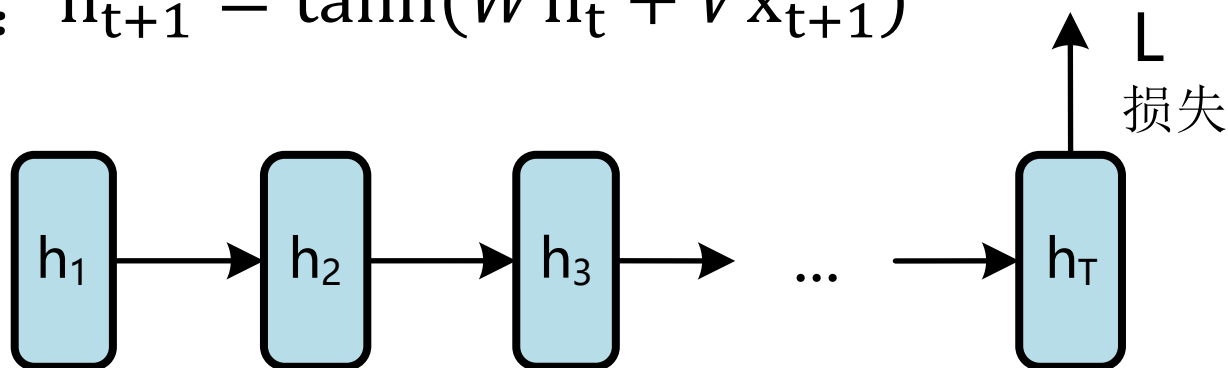
- 一般结构退化为线性序列结构时，递归神经网络就回到了vanilla rnn

RNN的问题

- 序列过长时，可能存在先前步骤信息的丢失
- E.g. 词语预测：She's from Italy and cannot speak Chinese. She moved here last month... I use my phone to translate what we say into ?
- 梯度消失/爆炸
- 梯度消失(vanish)比较常见，指梯度接近于0，模型不学习。也叫梯度弥散
- 梯度爆炸(explode, blow up)发生较少，指梯度过大，模型也不能学习，产生参数振荡

梯度消失/爆炸的原因

- 循环: $h_{t+1} = \tanh(W h_t + V x_{t+1})$



- 根据链式法则:

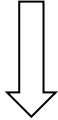
$$\frac{\partial L}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \frac{\partial L}{\partial h_2} = \frac{\partial h_2}{\partial h_1} \frac{\partial h_3}{\partial h_2} \frac{\partial L}{\partial h_3} = \dots$$

一般化

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=1}^{T-1} \frac{\partial h_{k+1}}{\partial h_k}$$
$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L}{\partial h_T} \prod_{k=t}^{T-1} \text{diag}(1 - (h_{k+1})^2) W_k^T \quad \uparrow$$

- 实际只能学习到短周期的依赖关系: 长程依赖/长距离依赖

如何避免

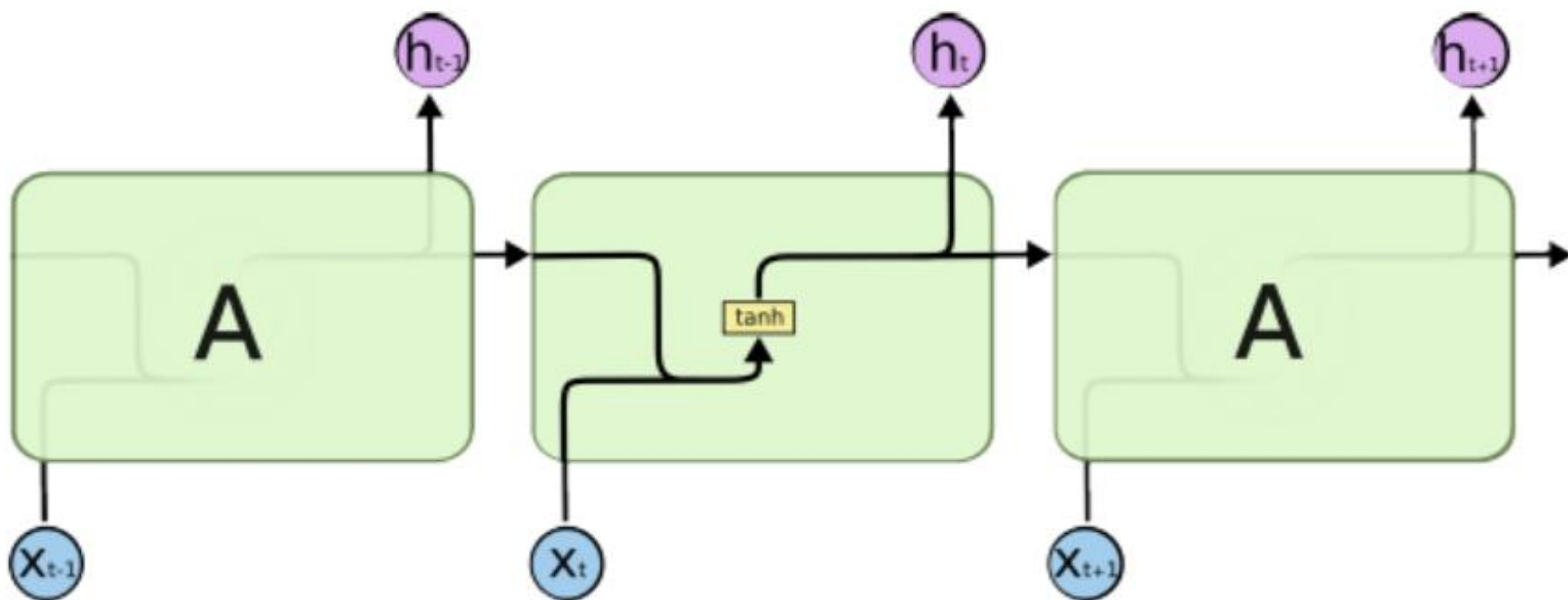
- 梯度爆炸：
权重衰减 (为参数值添加正则化项，一般是L2正则化)
- 梯度消失：
修改激活函数
修改模型 (引入门控单元)

主要解决手段
体现在RNN的主流变体中！



Outline

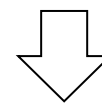
- 循环神经网络(RNN)
- 反向传播算法
- RNN主流变体(LSTM, GRU)
- 案例分析

回顾RNN



$$h_t = \tanh(Ux_t + Wh_{t-1} + b)$$

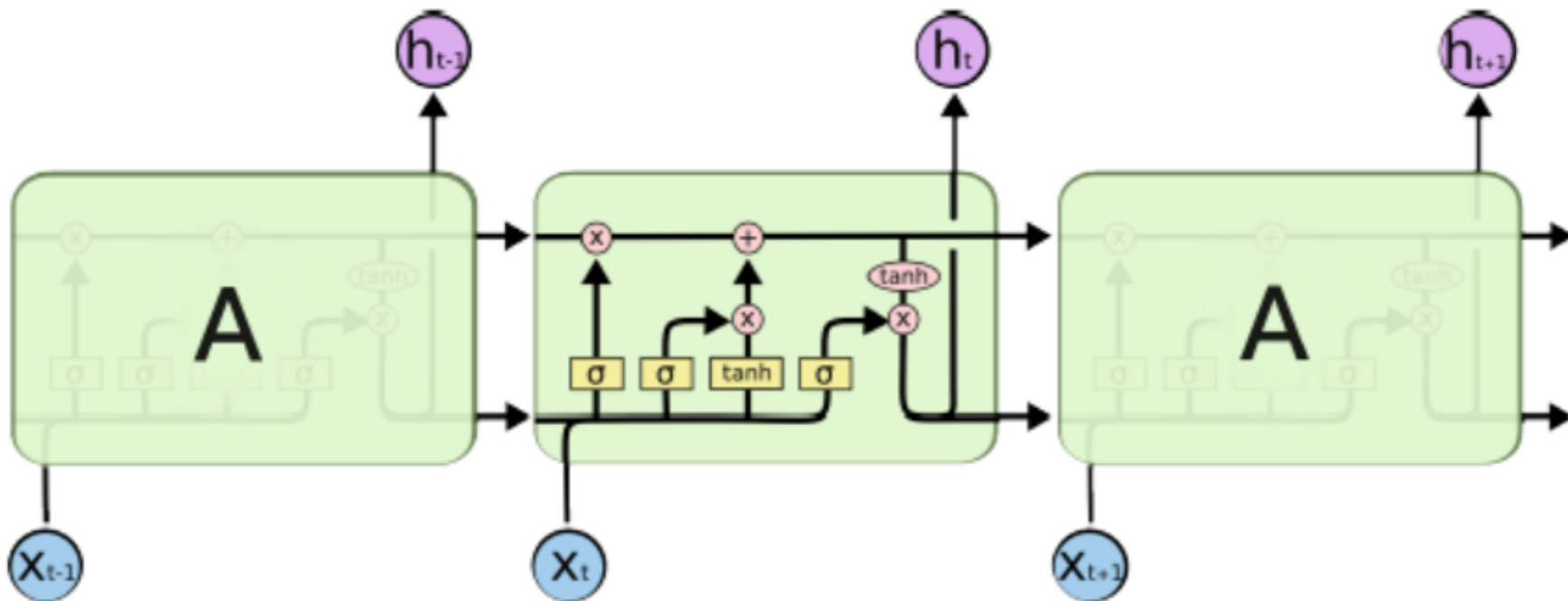
短期记忆
short term memory



如何设计带有独立内存的RNN?

Long Short Term Memory

- 简称**LSTM**，中文称长短时记忆
- 提出于1997年(Hochreiter and Schmidhuber)

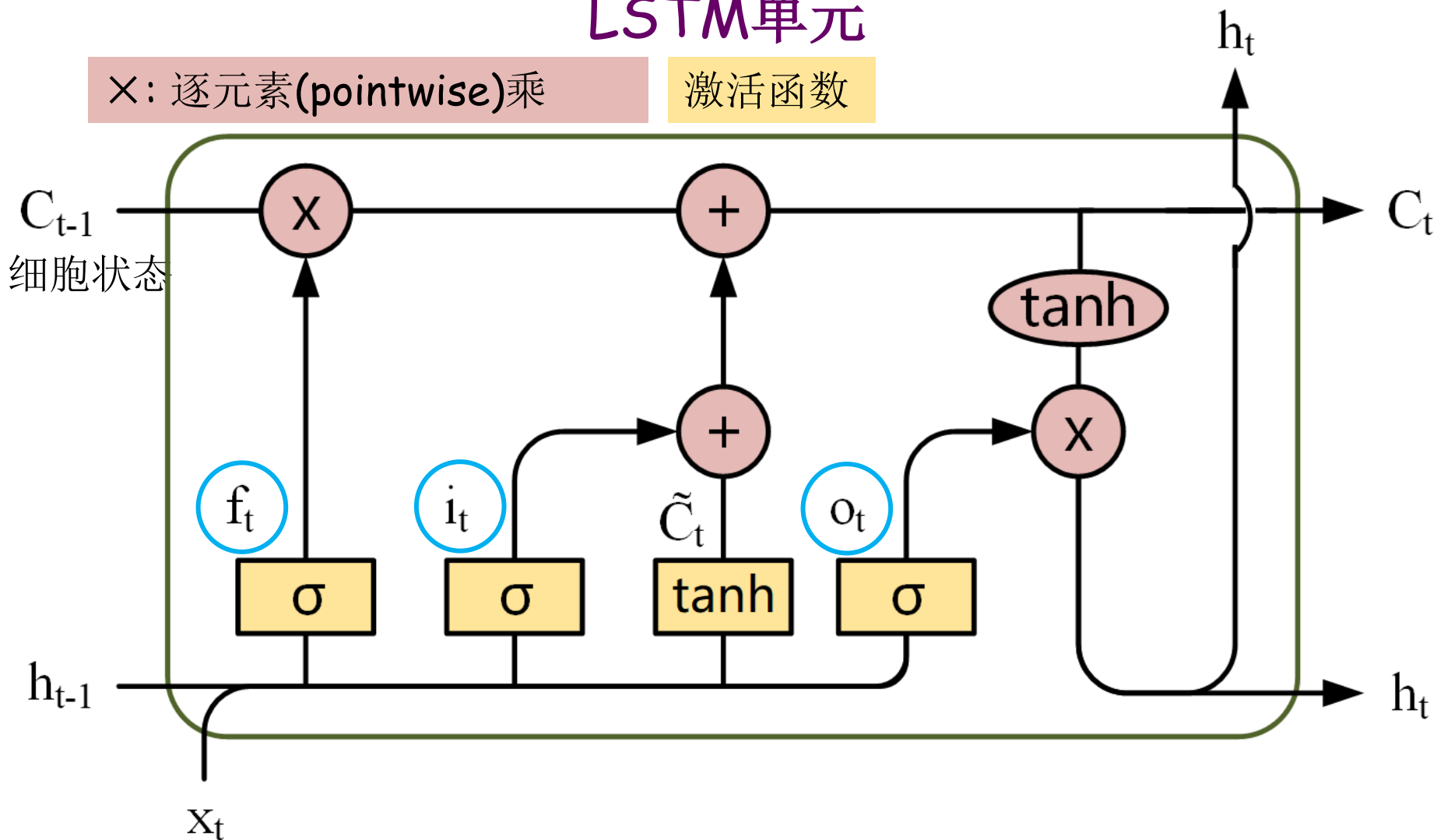


额外的传递信息单元：细胞状态cell
内部有4个网络层：3个sigmoid层，1个tanh层

LSTM单元

×: 逐元素(pointwise)乘

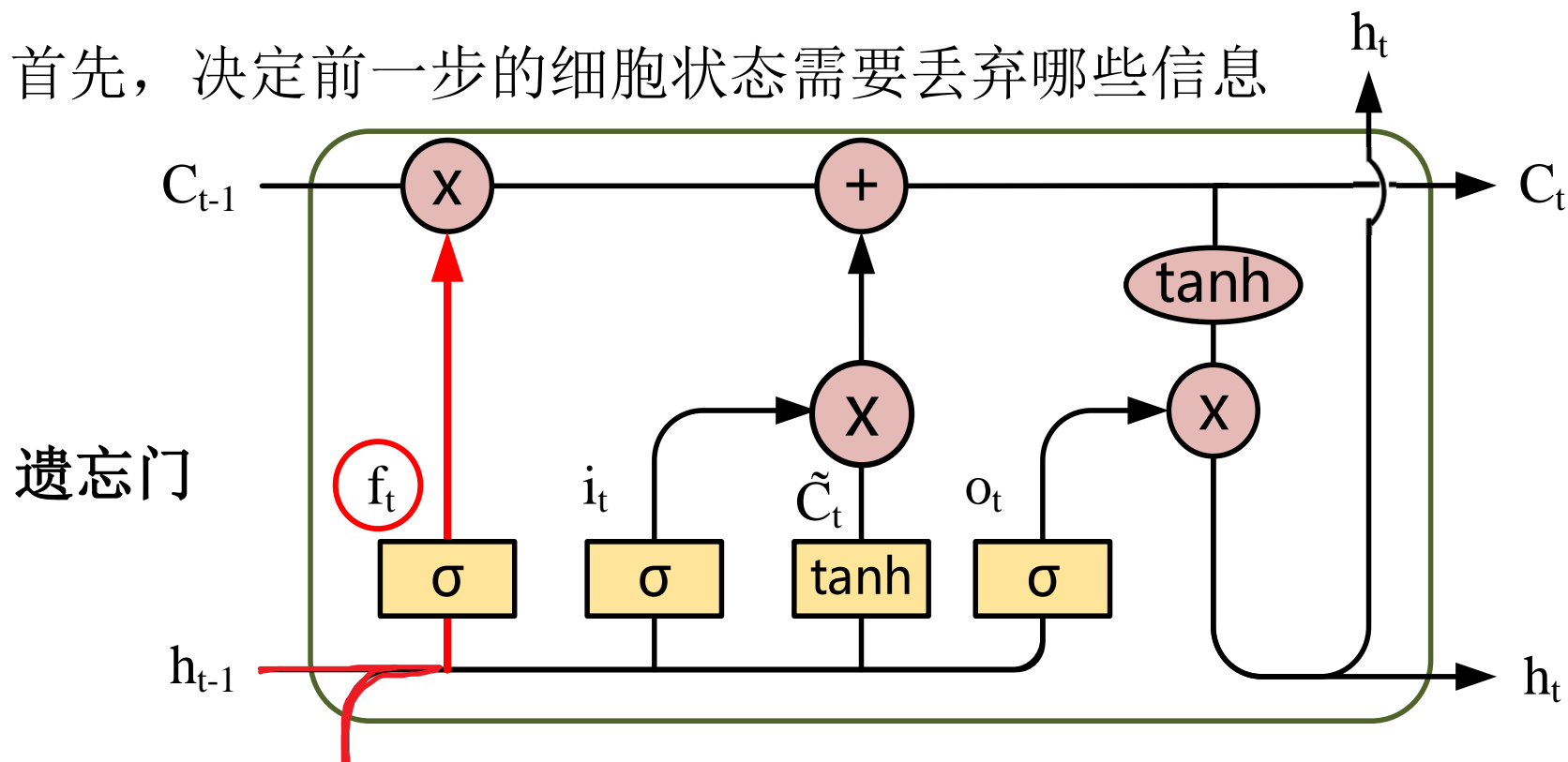
激活函数



LSTM通过门机制实现对细胞状态的读取、删除、添加信息等
门: **sigmoid**+哈达玛积(×)

LSTM单元

首先，决定前一步的细胞状态需要丢弃哪些信息



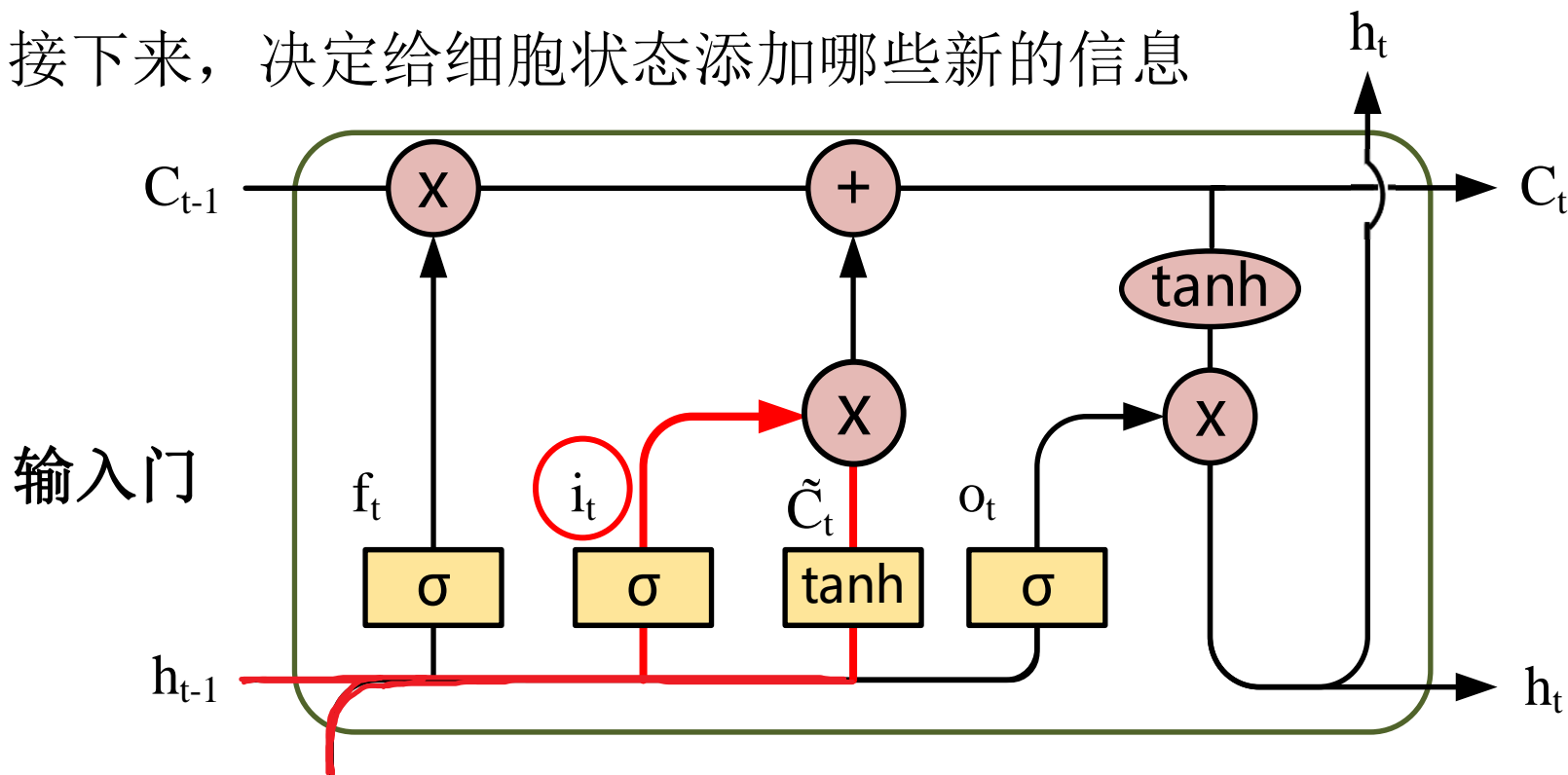
forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Sigmoid输出一个0-1之间的向量，0-1值表示旧细胞状态信息的保留或丢弃，0表示不保留，1表示全保留

其他版本: $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$

LSTM单元

接下来，决定给细胞状态添加哪些新的信息



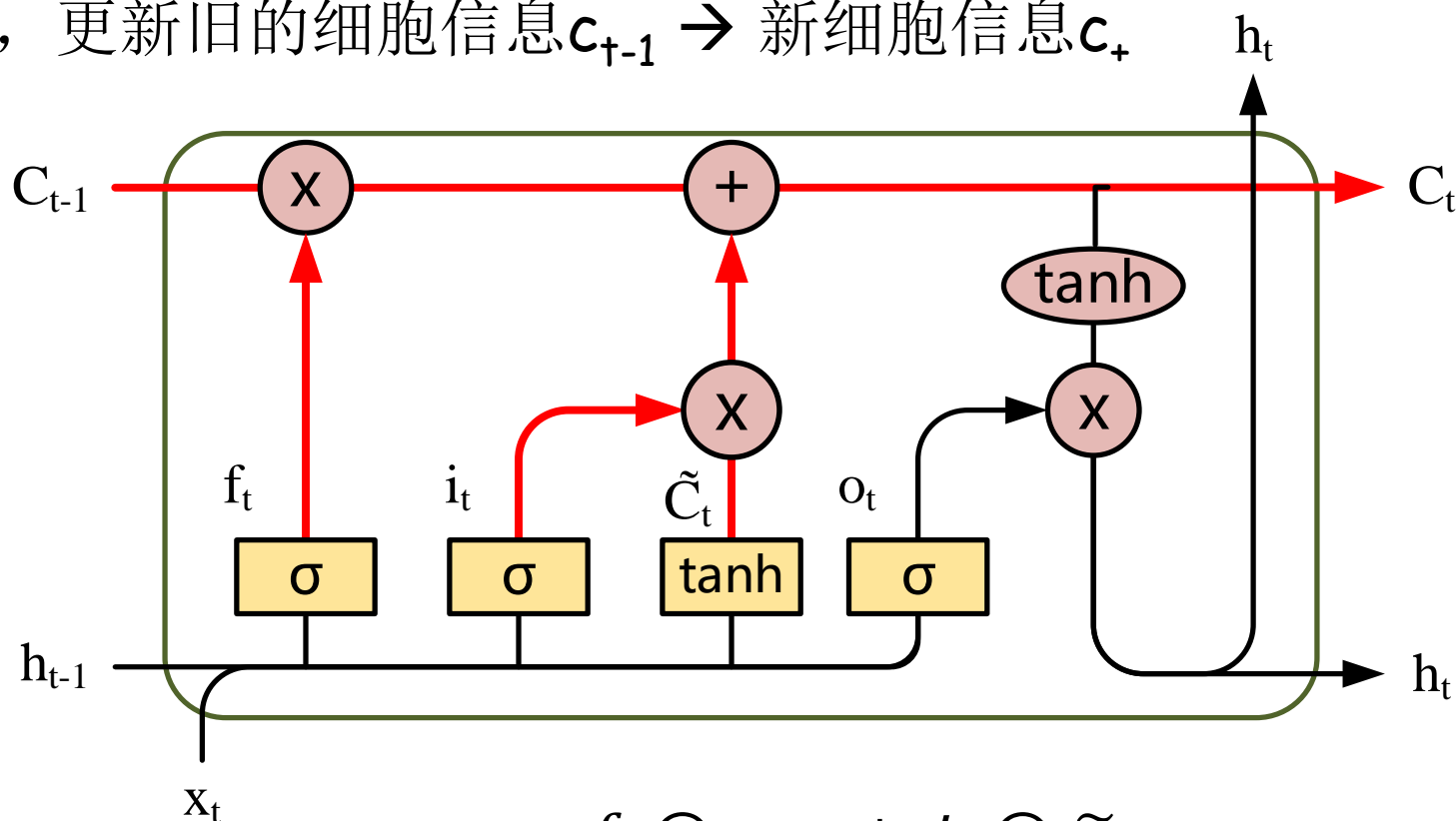
input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ 决定更新哪些信息

新的候选细胞信息: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

这些信息可能被更新到细胞状态中, 也叫new cell content

LSTM单元

然后，更新旧的细胞信息 $c_{t-1} \rightarrow$ 新细胞信息 c_t

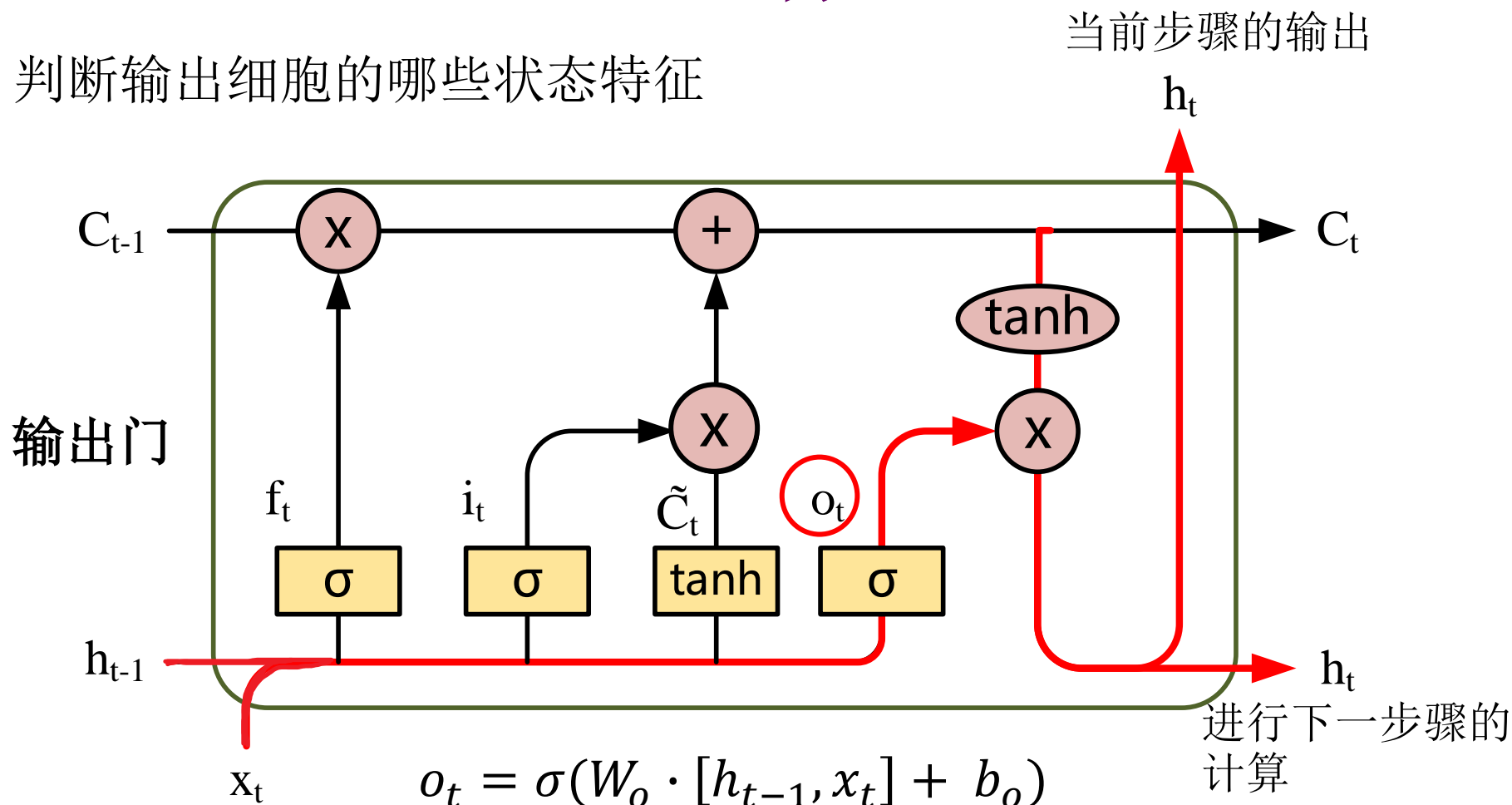


$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

通过遗忘门 f_t 选择忽略 c_{t-1} 的一部分(即传入0-1之间的部分旧信息)，通过输入门 i_t 选择添加候选细胞信息 \tilde{c}_t 的一部分

LSTM单元

判断输出细胞的哪些状态特征



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

$$h_t = o_t \odot \tanh(C_t) \quad \text{读取部分细胞状态}$$

注意 h_t 有两个流向