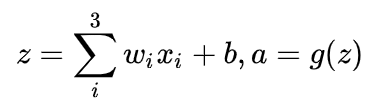
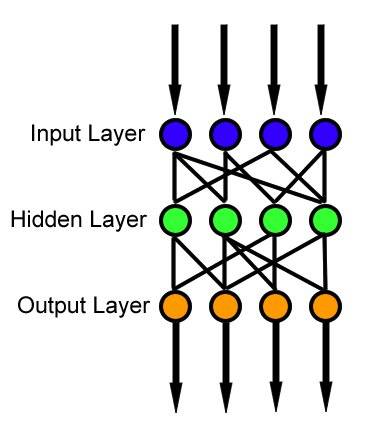
# RNN

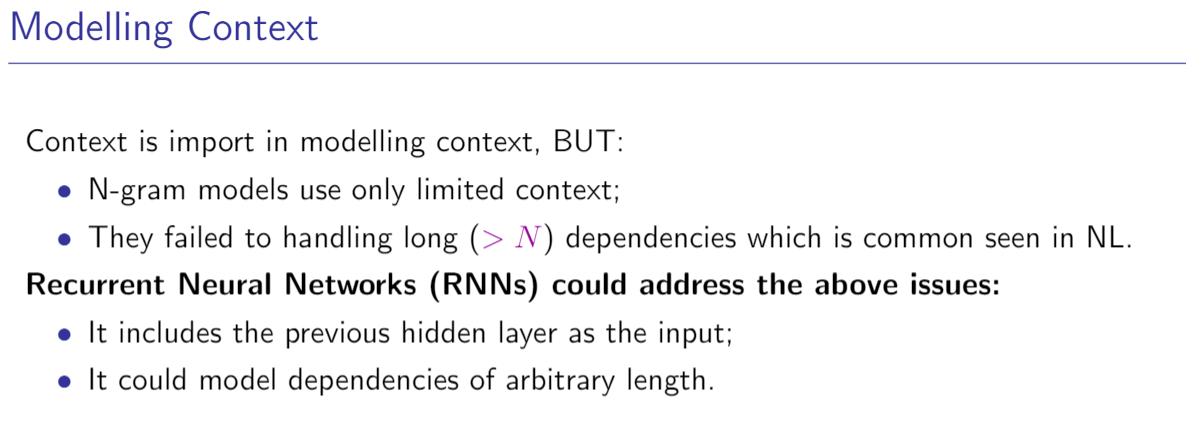
RNN适合应用于序列，变长的句子。

## 引入



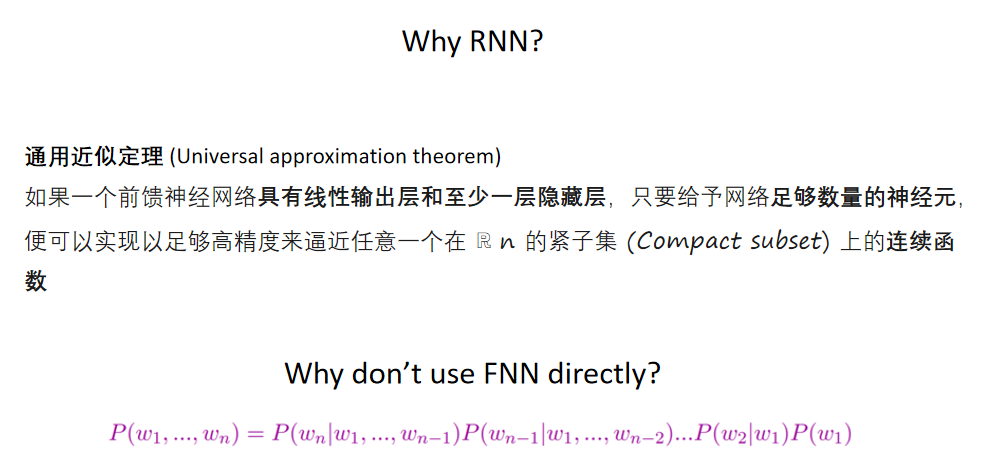
z为线性变化，a为激活函数

## 函数近似语言模型\*（近似定理）

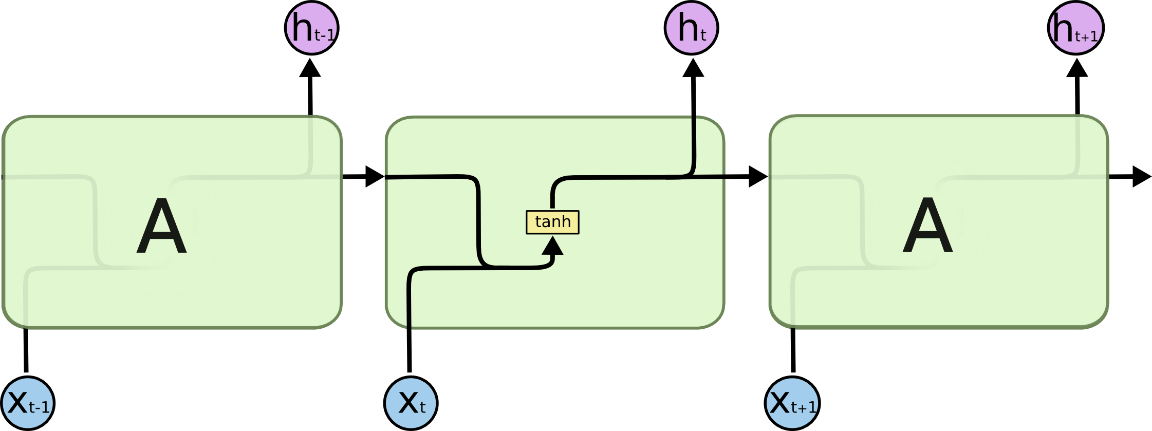


前馈的输入的长度是有限的依赖问题，FNN太宽（也就是0/1就可以构成一个windows，但不会这么做）

引入RNN能更好的学习有历史依赖关系的情况\*（序列有前后依赖关系）



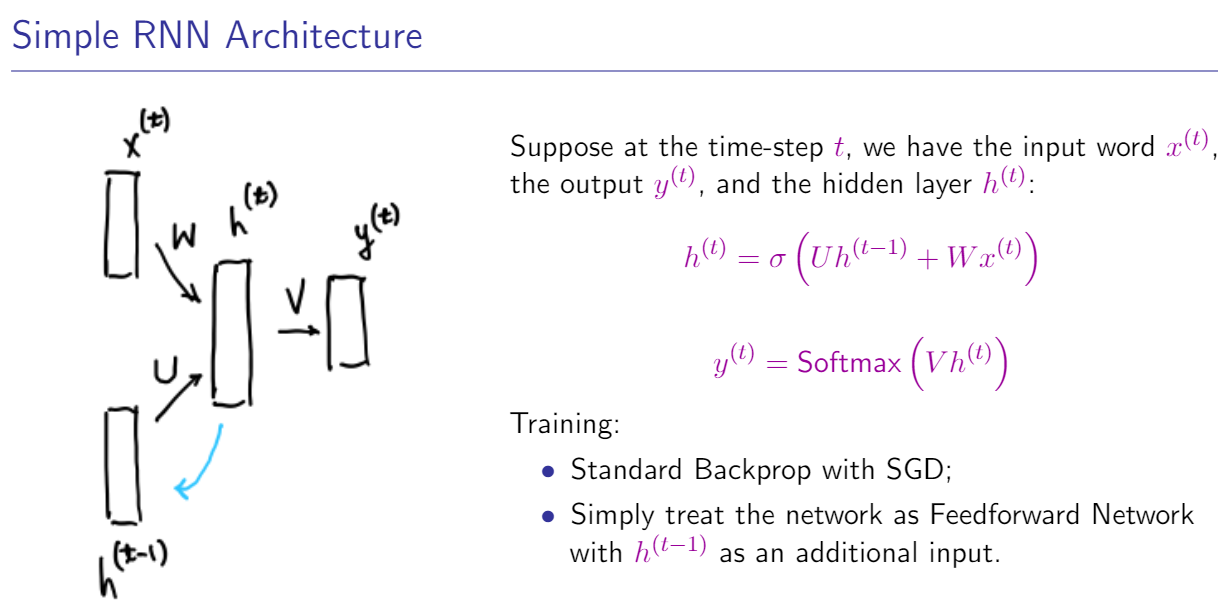
## RNN

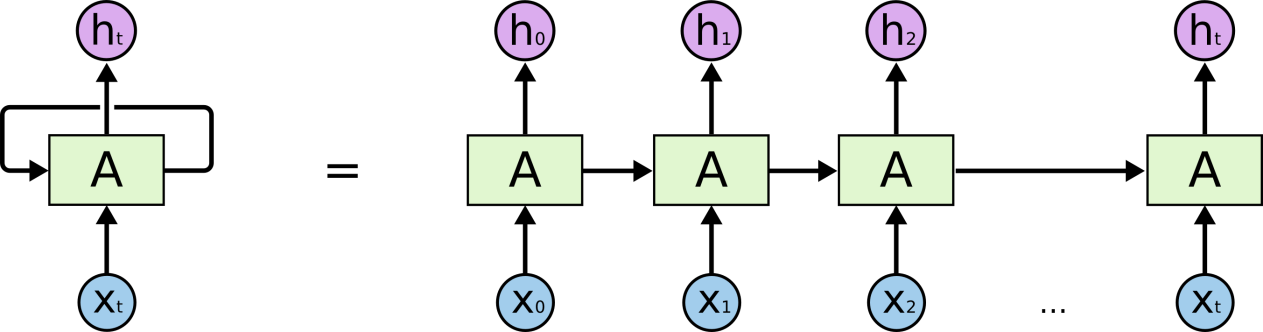


U和W是共享的，也就是在更新的时候是不变化的。

输出的时候也是用同一个V。（实际应用中可以调整，实际应用最好共享）

求 U，V，W参数：用反向传播算法（几乎处处连续可微，因为relu在0不可微）





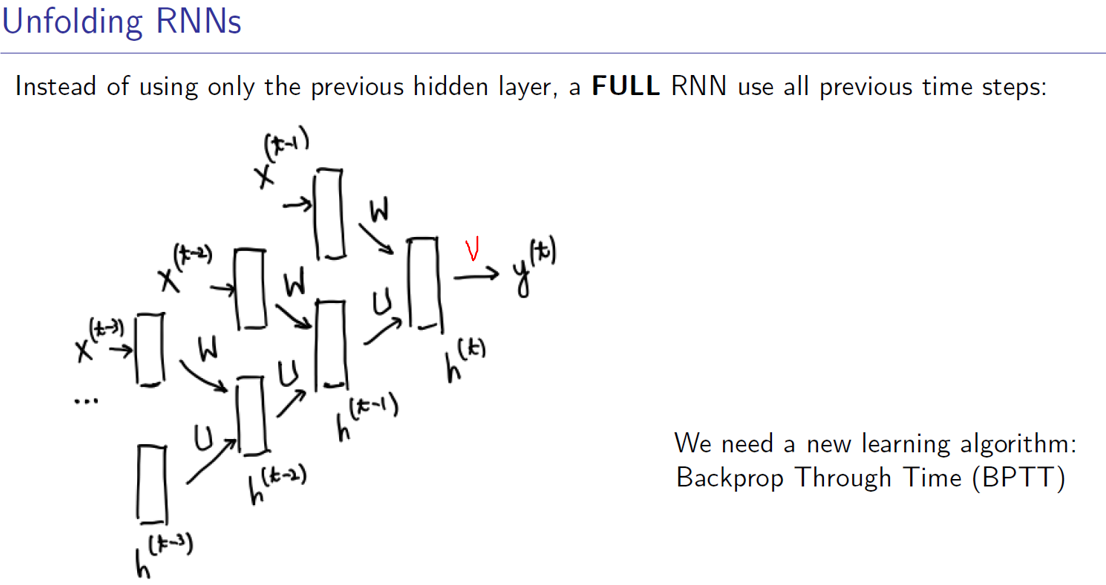
## Back-propagation For Recurrent Nets

在实际应用中，会线性RNN的长度。

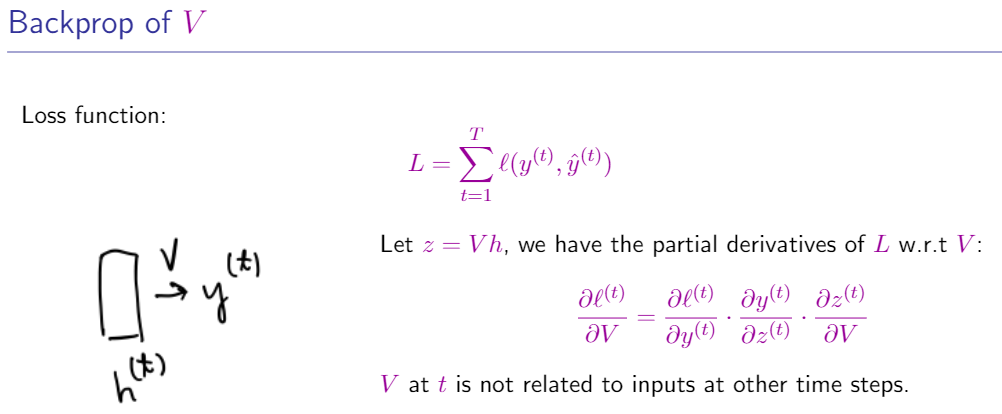
### BPTT计算梯度

把计算RNN的过程展开为一个前馈神经网络，再用前馈神经网络的方式求解参数。

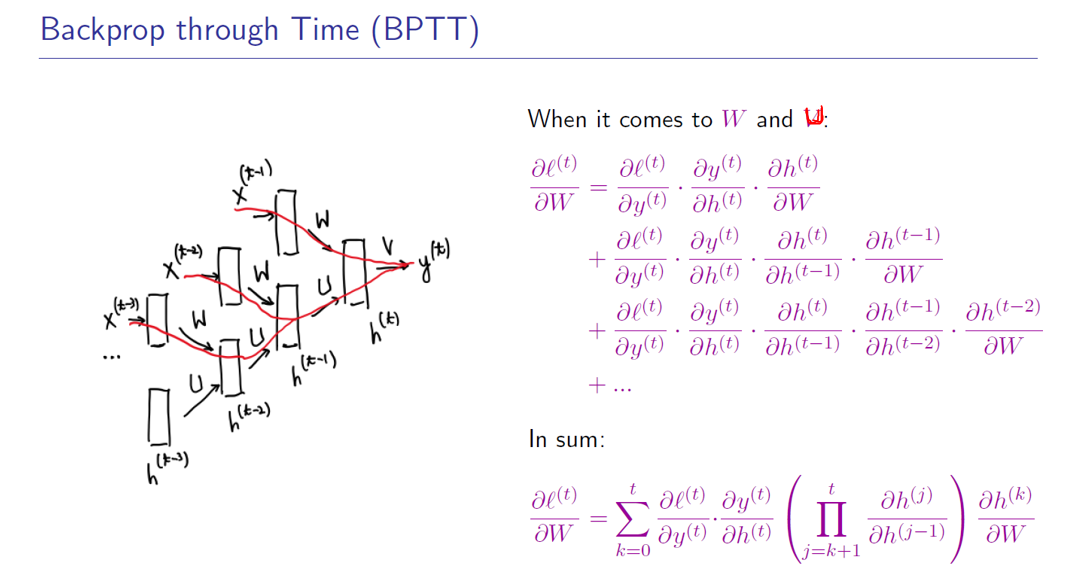
**所有的梯度都得保存。**



关于参数V（只用到了在计算h（t）的时候）



参数W和U

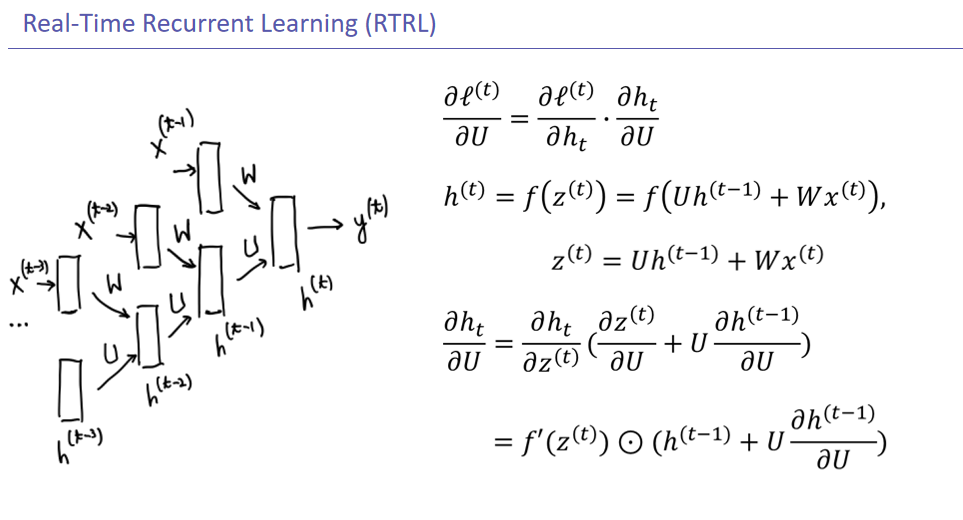


### RTRL计算梯度（NLP用的少）

**Real-time** Recurrent Learning（RTRL），最后求出来的梯度是一样的。

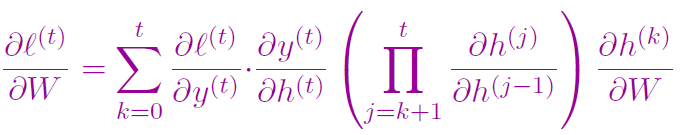
**在前向传播的时候，维护一个额外的变量，同时保存梯度。实时性.**

**没有人专门研究RTRL和BPTT谁好.**

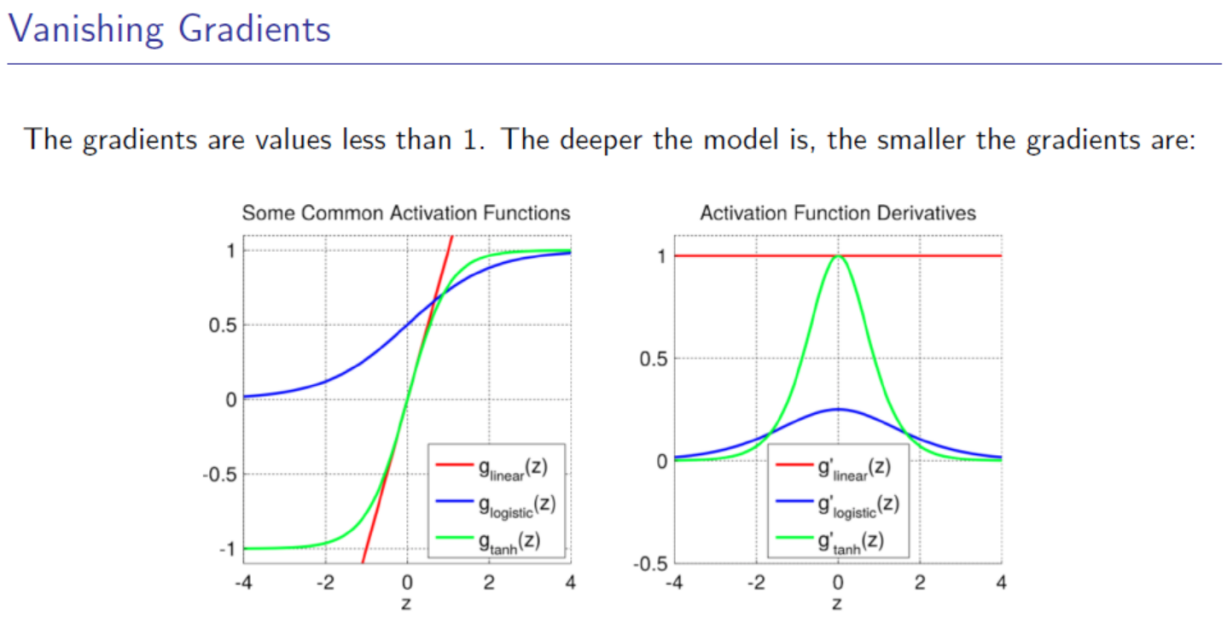


### 4.3 为什么不建立一个无限长的序列

RNN在应用的时候,会限制其长度.RNN长的时候很难优化,会有梯度为0或者梯度爆炸.(连乘)



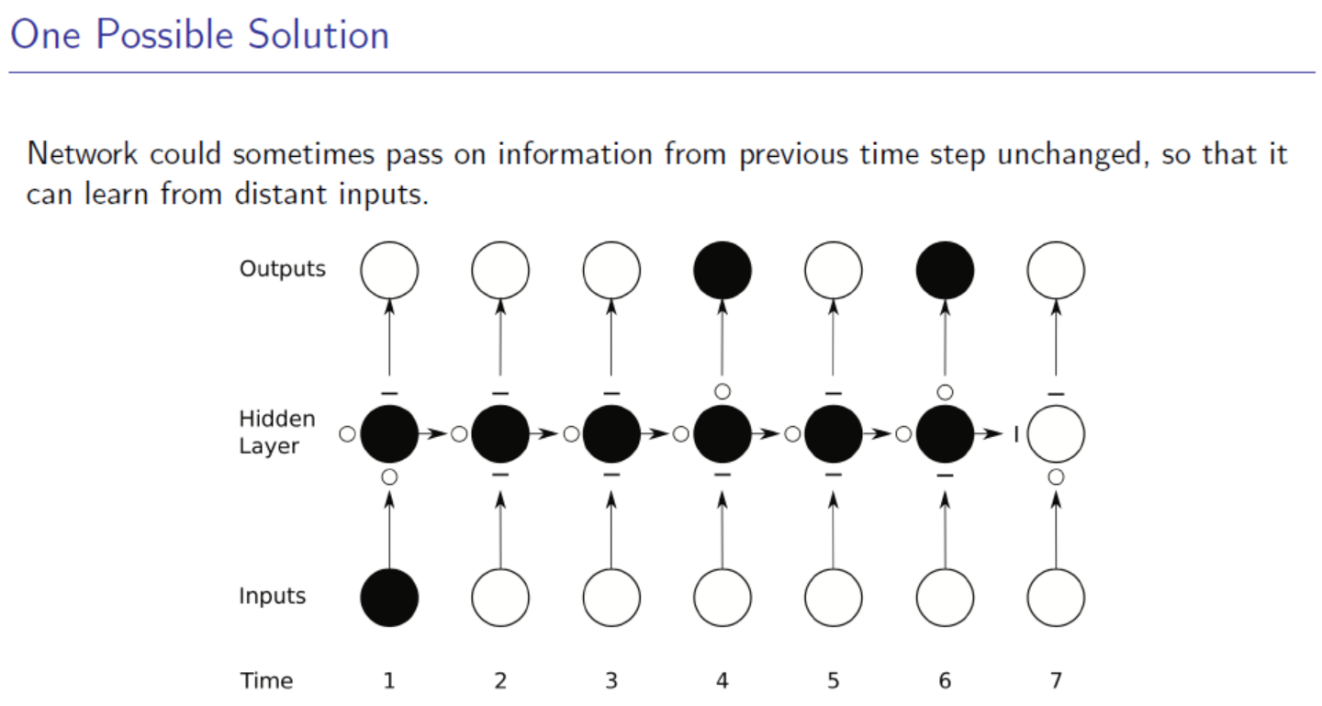
偏导数都是小于1的

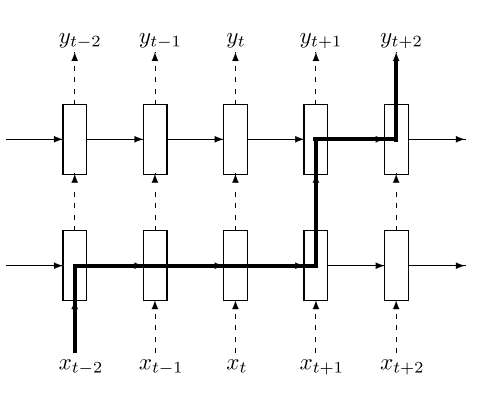


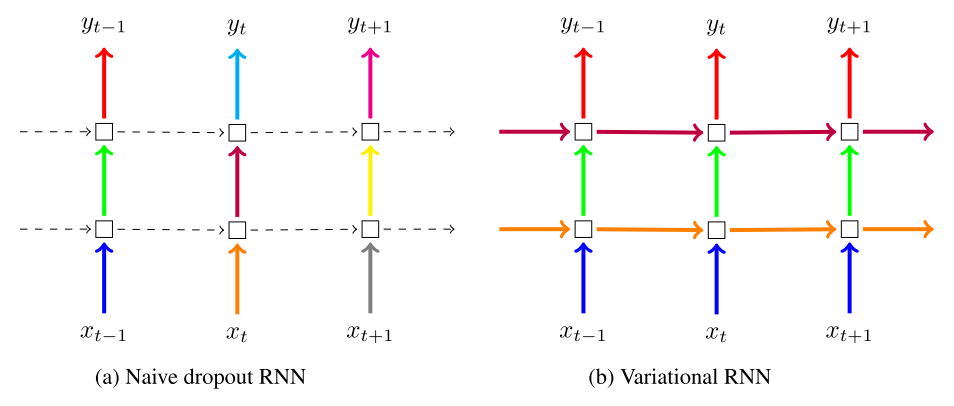
### 解决:Dropout

Mark一些h(t)

<https://medium.com/@bingobee01/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b>







## LSTM（Long Short-term Memory）

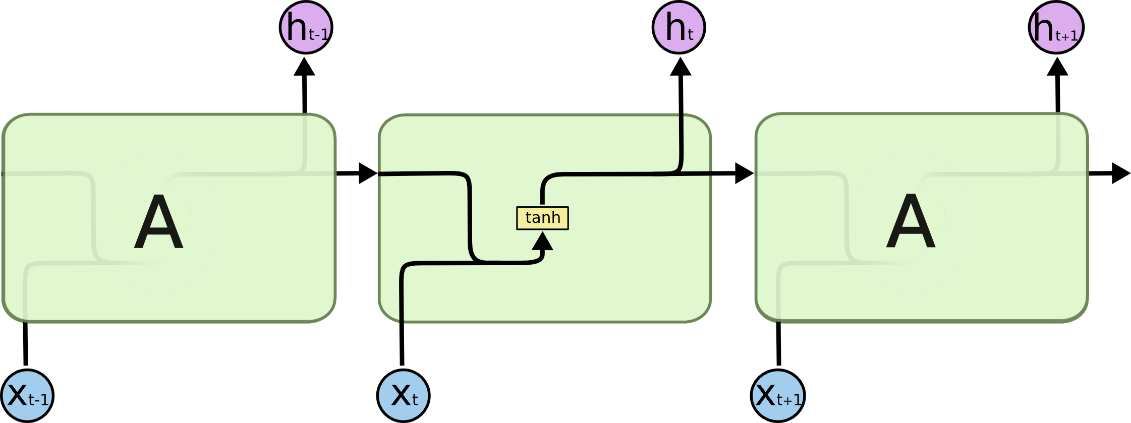
LSTM接着RNN，自动学习了Marks，在哪个位置更新h，哪个位置不更新（不参与梯度计算）。

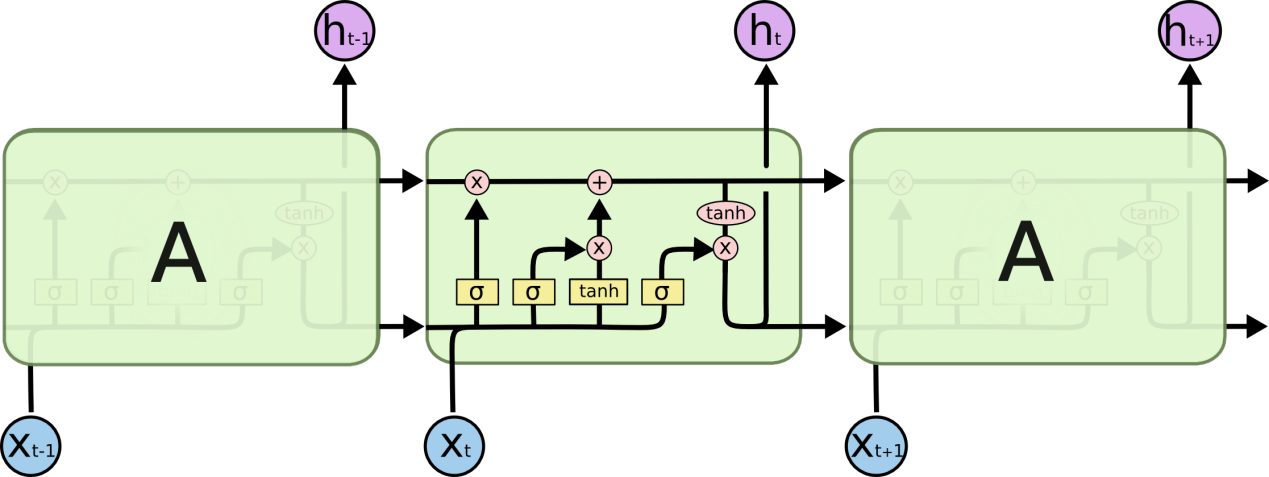
防止了梯度消失。

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



### Naïve RNN vs LSTM

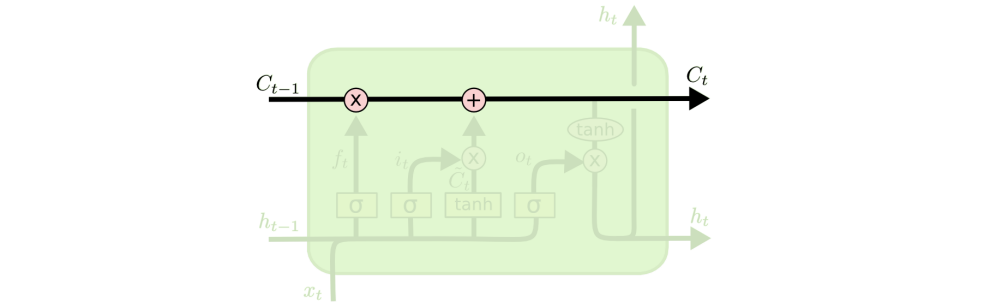




### LSTM具体介绍

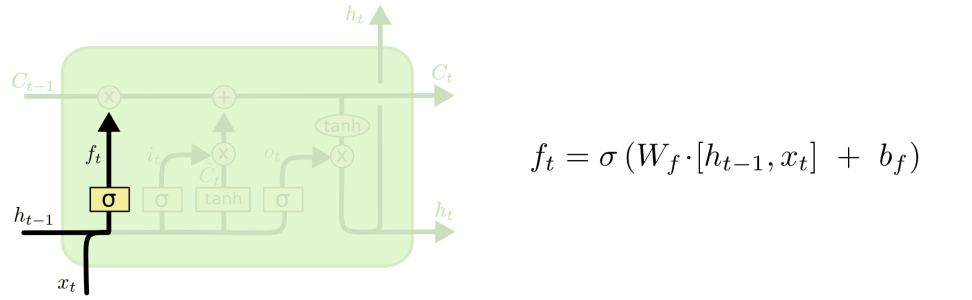
#### Core Idea of the LSTM: Gate

根据C去决定是否更新当前状态要不要使用t去更新当前的隐状态

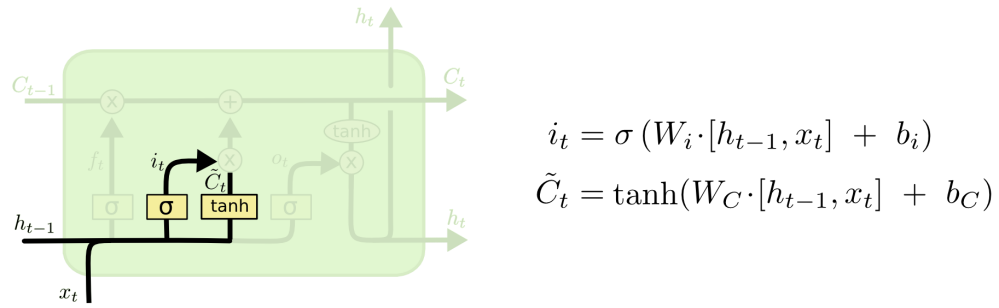


#### Forget Gate

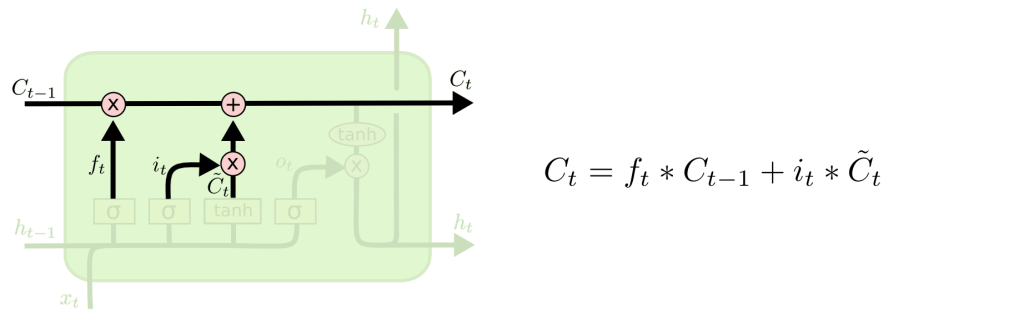
用遗忘门忘记一些东西，去为更新C\_t做准备（要不要更新C和h）。Sigmoid的激活函数（输出在0-1之间，偏向于0更想忘掉一些东西，偏向于1更像记住）



#### input Gate

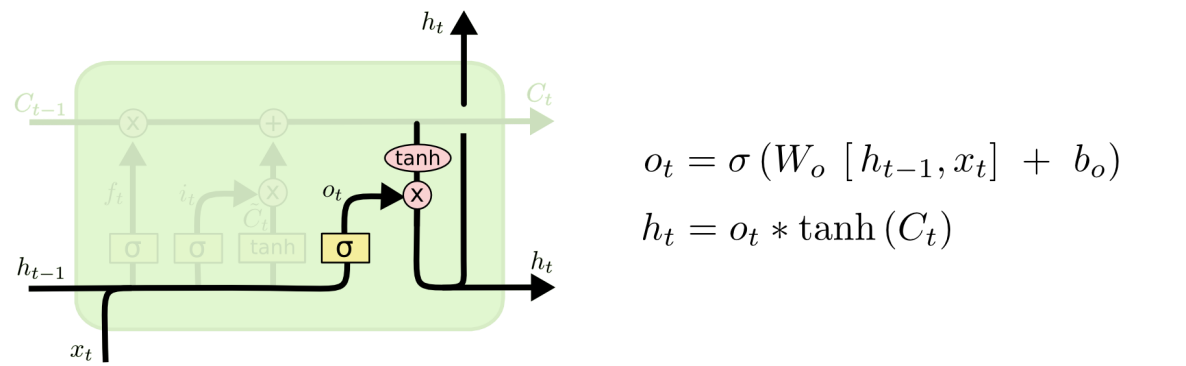


**更新C\_t**

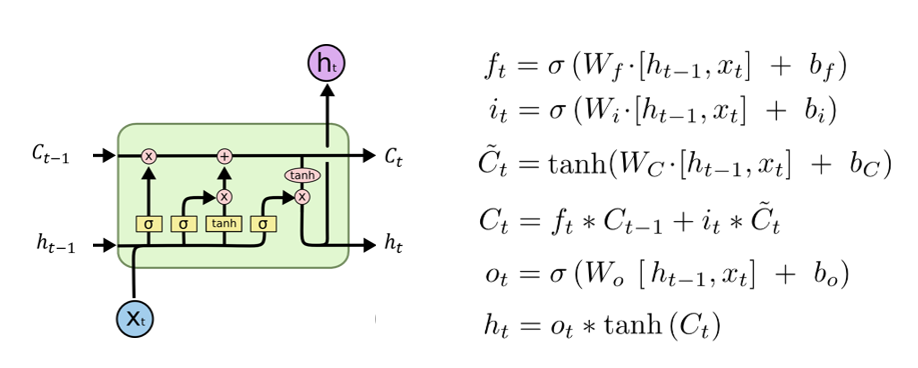


#### Output Gate

Output对c进行了选择

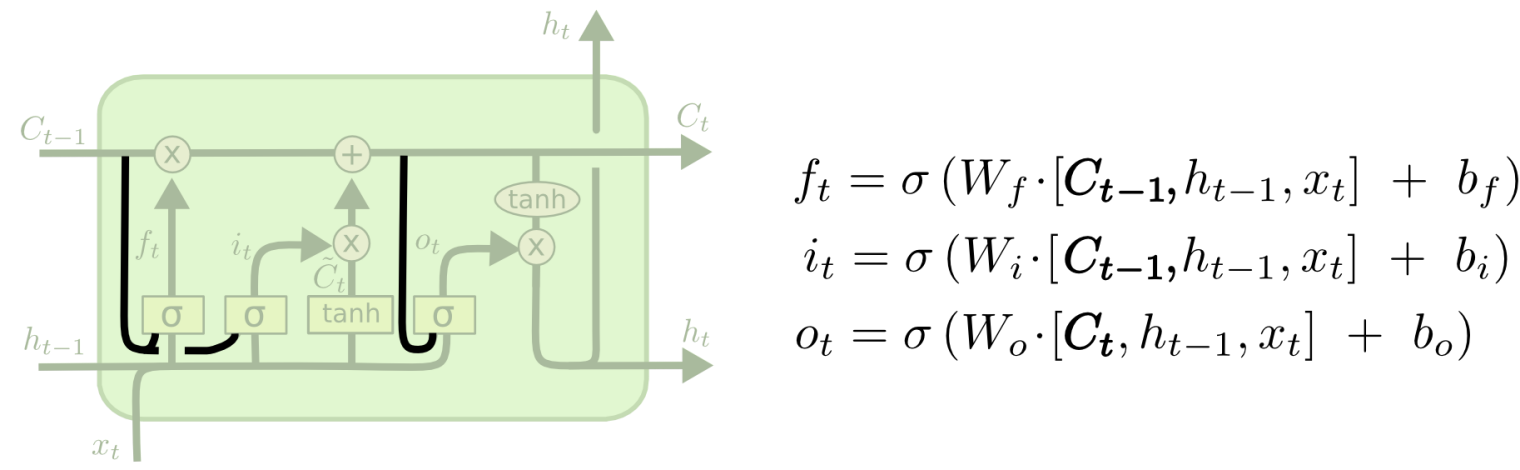


总的：最后我们会把h\_t和C\_t拼接起来继续输出。（C和h的维度是一样的）

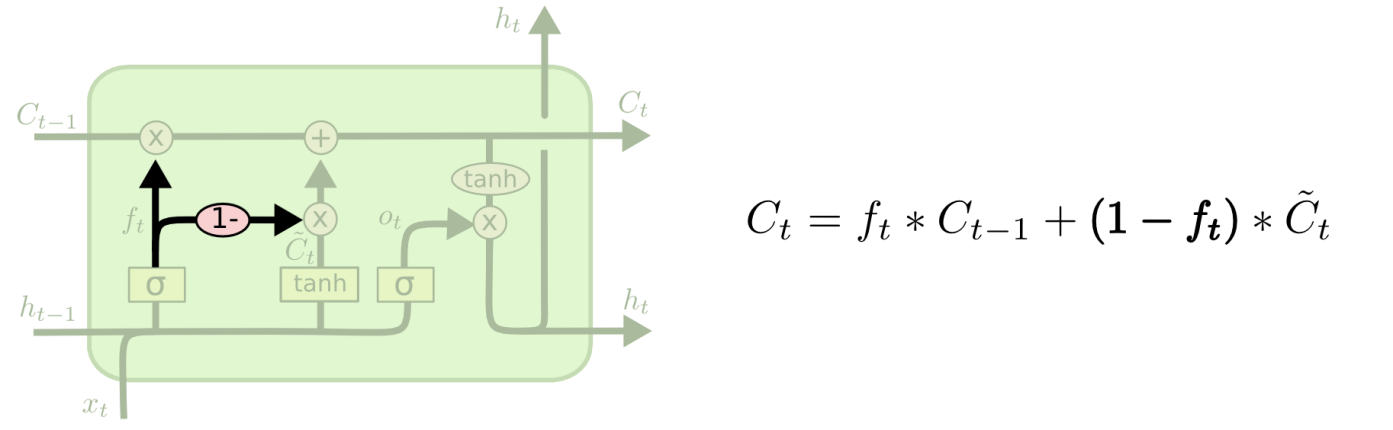


### LSTM变种

add peephole



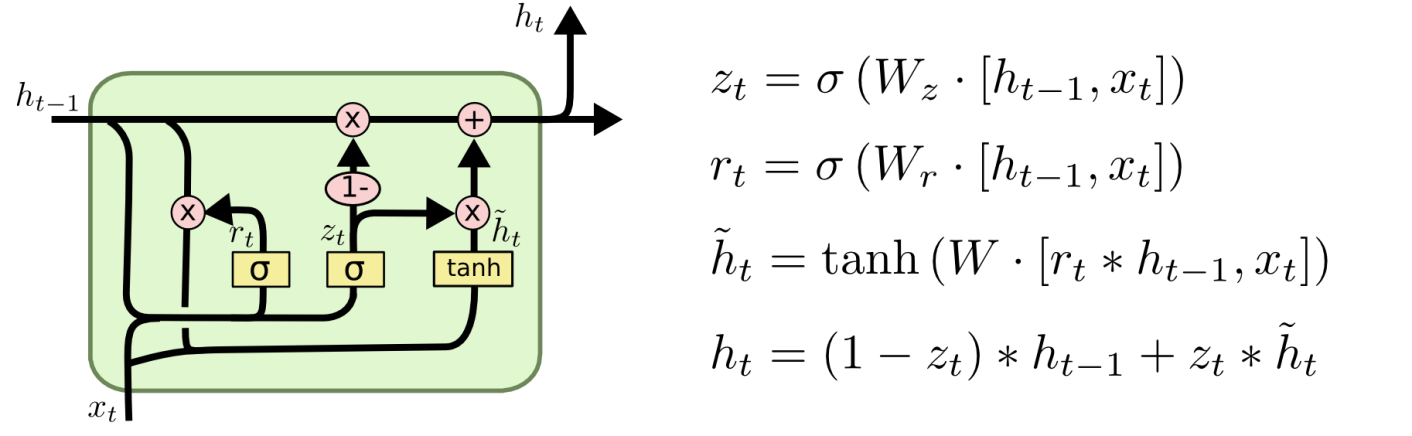
Forget while input：忘多少记多少



### GUR：LSTM最流行的变种

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

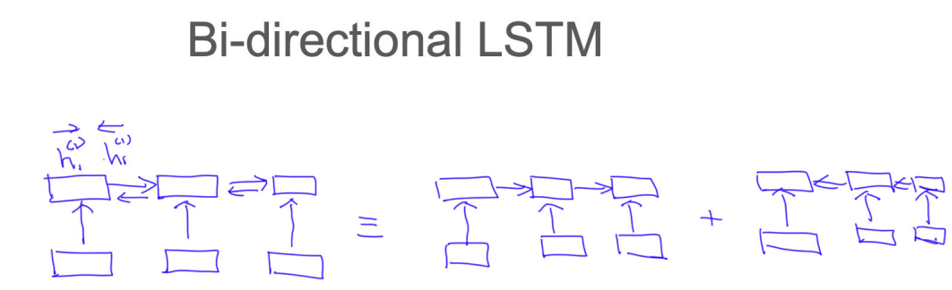
效果类似，速度更快



### Bi-LSTM

**LSTM可视化：Visualizing and Understanding Neural models in nlp**

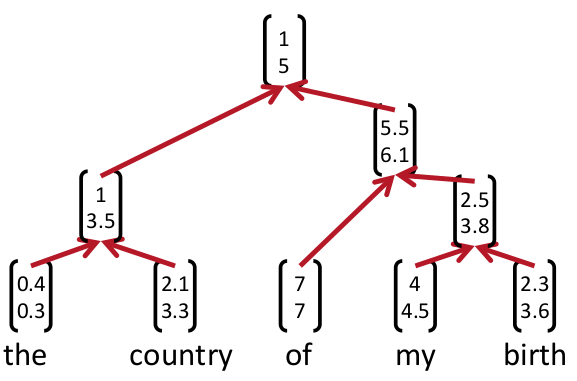
**LSTMVis：A Tools for Visual Analysis of hidden state Dynamics in Recurrent Neural Networks**



## Applications

### 3.1 递归神经网络（recursive neural networks）

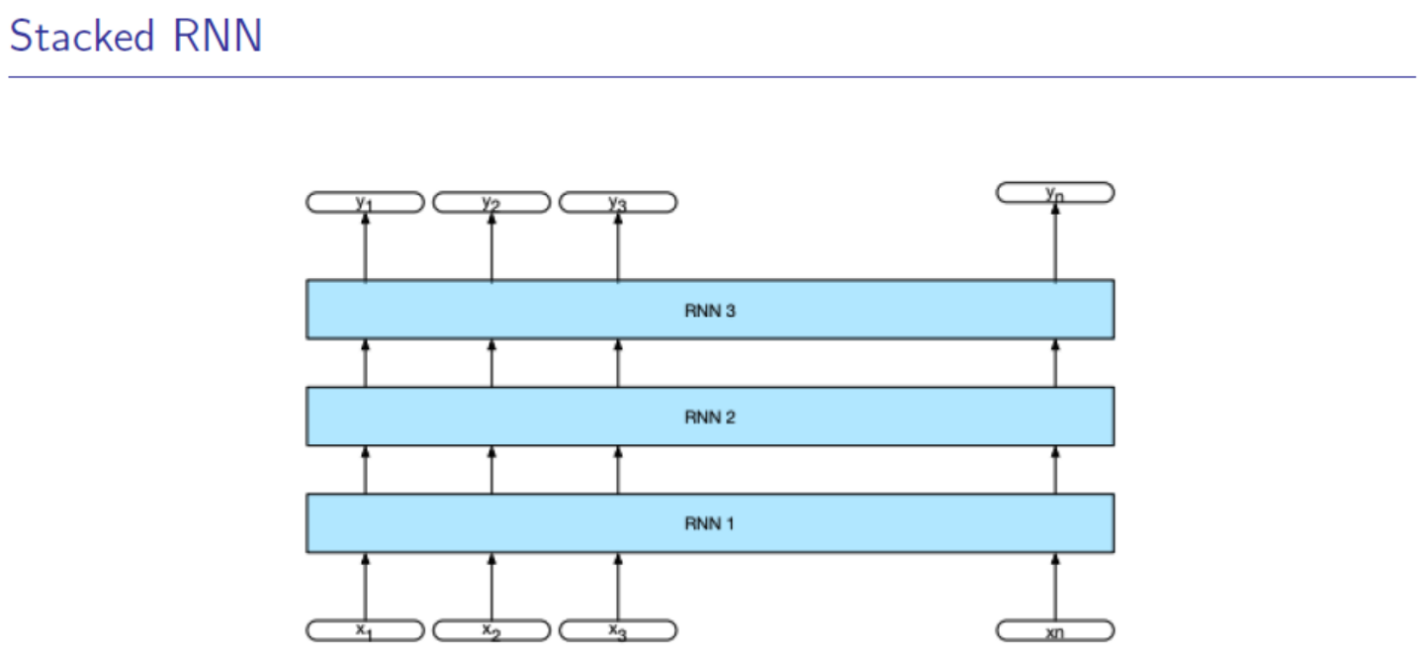
未解决问题：怎么决定融合的顺序？（贪心？）



### 3.2 Stacked RNN(堆叠式的RNN)

多个隐状态h\_t,m\_t,n\_t（参数多，计算慢，但能力更强）

1-5层



### 3.3 双向RNN

同时拥有左右的信息。可并行计算。

