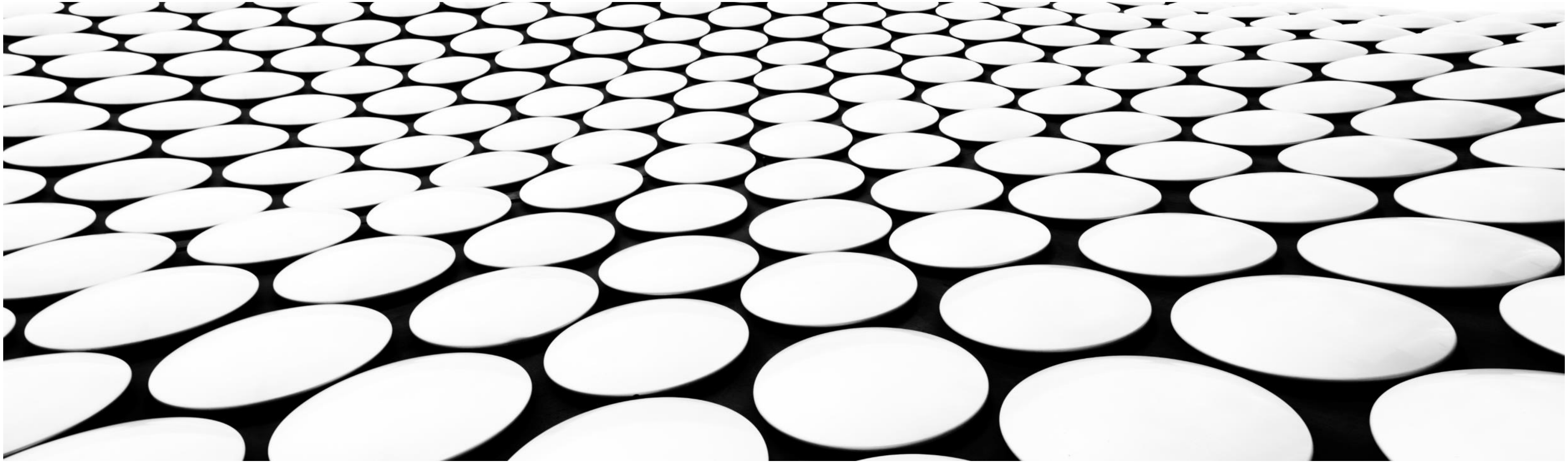
深度学习

邱怡轩



今天的主题

- 循环神经网络（续）

实例演示

性别预测

- 数据来源
- https://github.com/wainshine/Chinese-Names-Corpus
- >114万条已标注性别的人名

```
dict,sex
阿安,男
阿彬,未知
阿斌,男
阿滨,男
阿冰,女
阿冰冰,女
阿兵,男
阿婵,女
阿超,男
阿朝,男
阿琛,女
阿臣,男
阿辰,未知
阿晨,未知
```

```
石晓彦,女
石晓艳,女
石晓燕,女
石晓艺,女
石晓英,女
石晓莹,女
石晓颖,女
石晓影,女
石晓勇,男
石晓宇,男
石晓玉,女
石晓云,女
石晓泽,男
石晓珍,女
石筱,女
```

```
闫志慧,女
闫志坚,男
闫志江,男
闫志杰,男
闫志娟,女
闫志军,男
闫志君,未知
闫志丽,女
闫志利,男
闫志亮,男
闫志林,男
闫志玲,女
闫志龙,男
闫志梅,女
闫志民,男
```

```
佐非,未知
佐江,男
佐军,男
佐丽,女
佐隆,男
佐明,男
佐木,未知
佐娜,女
佐楠,女
佐山,男
佐腾,男
佐威,男
佐为,男
佐樱,女
佐子,男
```

预处理

- 删除未知类别
- 计算每个字出现的频率
- 选取前500个高频字
- 将范围限定在500个常用字中

	char	freq
636	王	50390
926	李	49078
1086	张	47089
1203	陈	41512
1394	刘	39986
...	...	...
2170	墙	1
2171	楼	1
2172	禺	1
2173	据	1
1962	莽	1

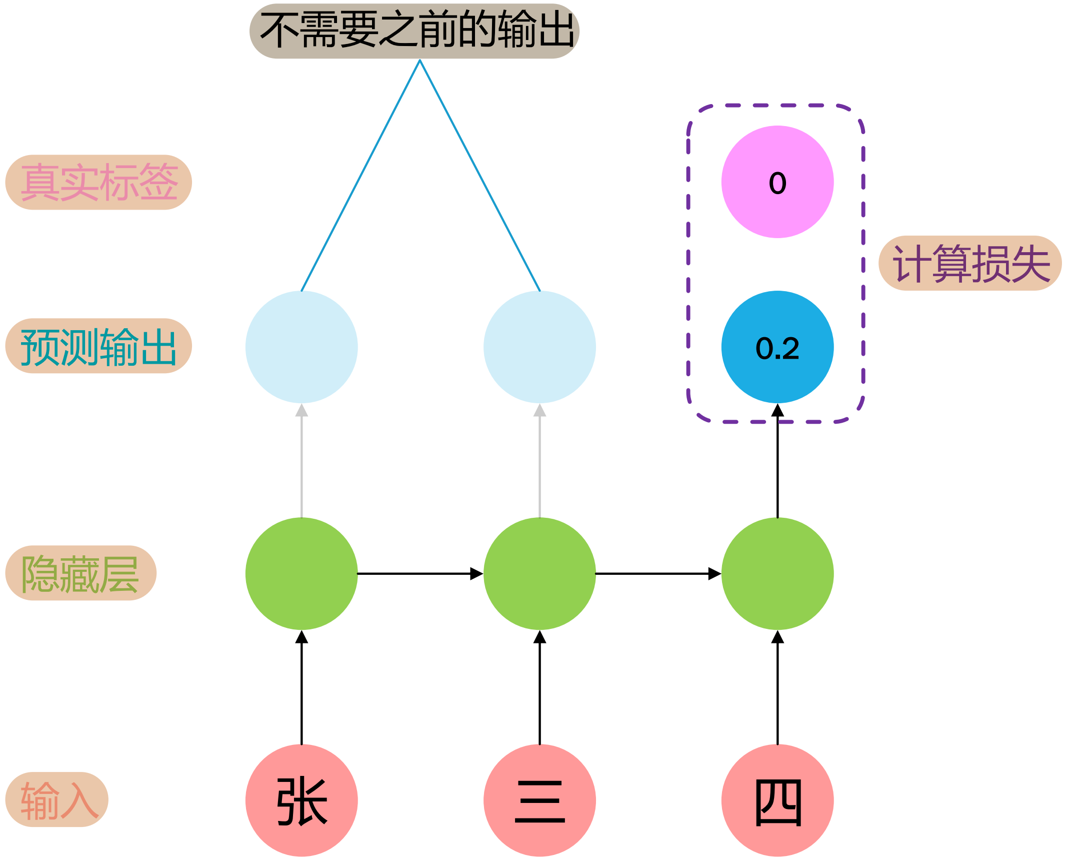
建立字典

- 将500个常用字作为字典

```
array(['王', '李', '张', '陈', '刘', '文', '林', '明', '杨', '华', '黄', '吴', '金',
       '周', '晓', '国', '赵', '玉', '伟', '海', '志', '徐', '丽', '红', '建', '朱',
       '孙', '平', '军', '英', '春', '龙', '胡', '永', '荣', '德', '云', '成', '郭',
       '东', '郑', '高', '芳', '马', '何', '梅', '新', '杰', '辉', '生', '秀', '玲',
       '江', '俊', '洪', '强', '世', '光', '罗', '艳', '燕', '兰', '子', '庆', '峰',
       '忠', '梁', '宇', '凤', '谢', '霞', '美', '宋', '祥', '清', '立', '兴', '萍',
       '许', '叶', '雪', '良', '安', '慧', '娟', '福', '宝', '佳', '方', '家', '唐',
```

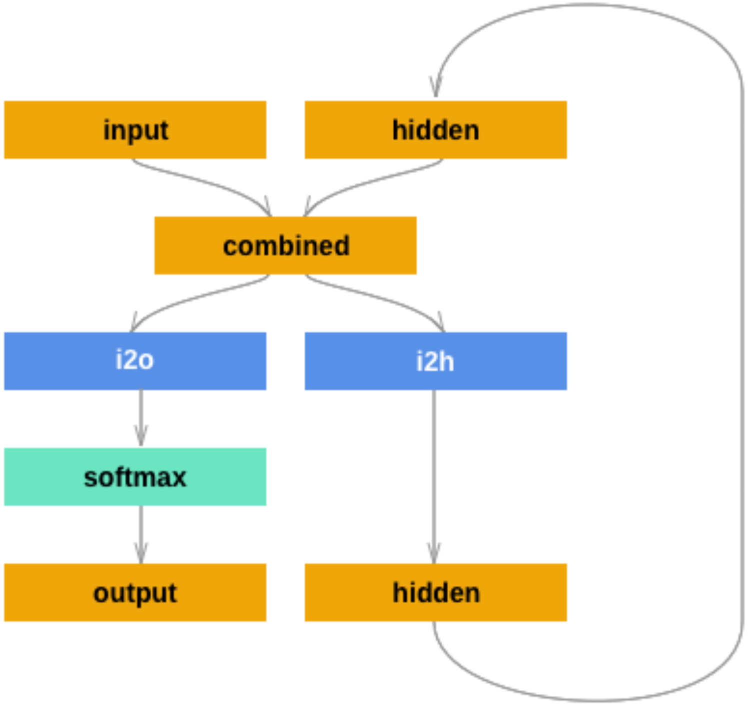
- 将汉字用 one-hot 向量编码
  - 王 = [1,0,0,…]
  - 李 = [0,1,0,…]
  - 张 = [0,0,1,…]
- 每个名字看作是一个序列 $x = (x_1, x_2, \dots)$
  - 每个 x_t 是一个 one-hot 向量

建模原理



建立模型

- 参考：https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html



测试结果

- 简单构建训练集（1万）和测试集（1千）
- CPU 上训练10秒钟
- 测试集准确率可达 97.9%

代码实现

- 参见 `name_classify.ipynb`

改进 RNN

优缺点

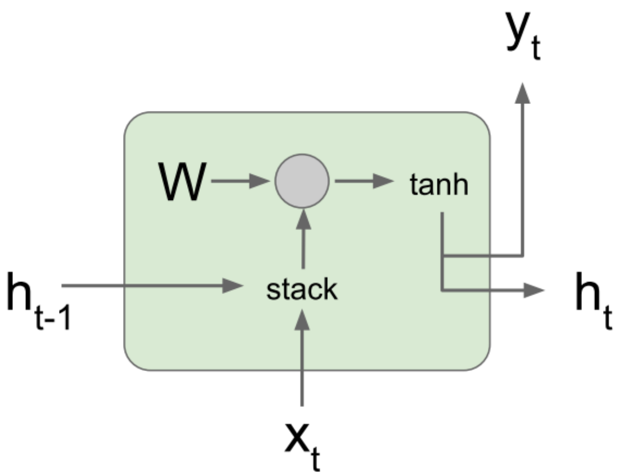
- 相比于前馈神经网络和 CNN，RNN 有其独特的性质
- 优点
  - 处理任意长的序列数据
  - 利用历史信息
  - 参数数量不随序列变长而增加
- 缺点
  - 序列很长时计算量非常大
  - 梯度消失/爆炸问题

反向传播

- 要理解 RNN 的局限，就需要明白它的反向传播原理

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

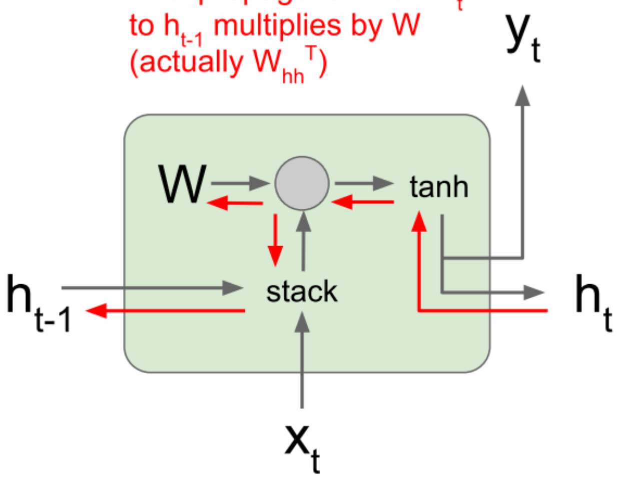


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix}W_{hh} & W_{hx}\end{pmatrix}\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}\right)$$

资料来自http://cs231n.stanford.edu/

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

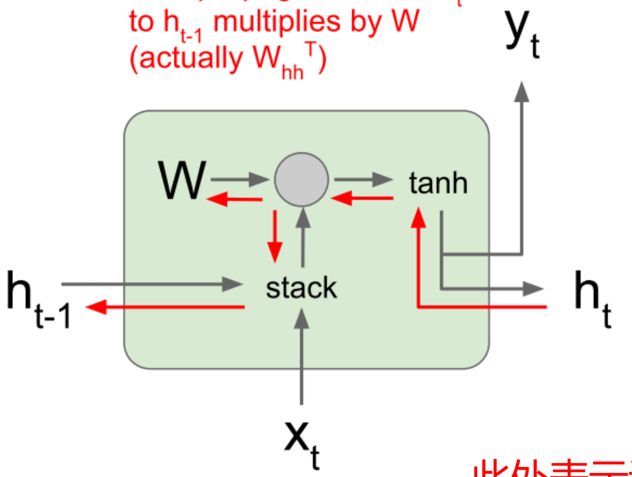


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



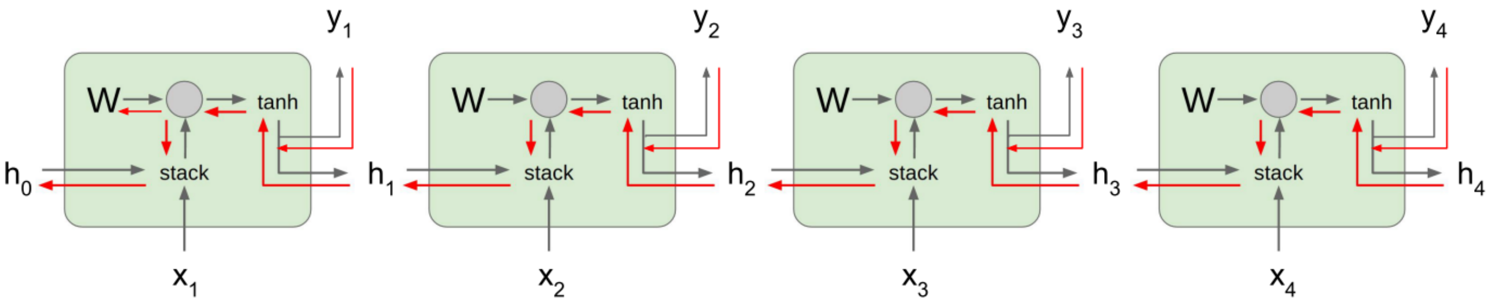
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

此处表示该向量形成的对角矩阵

$$\frac{\partial h_t}{\partial h_{t-1}} = \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}W_{hh}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

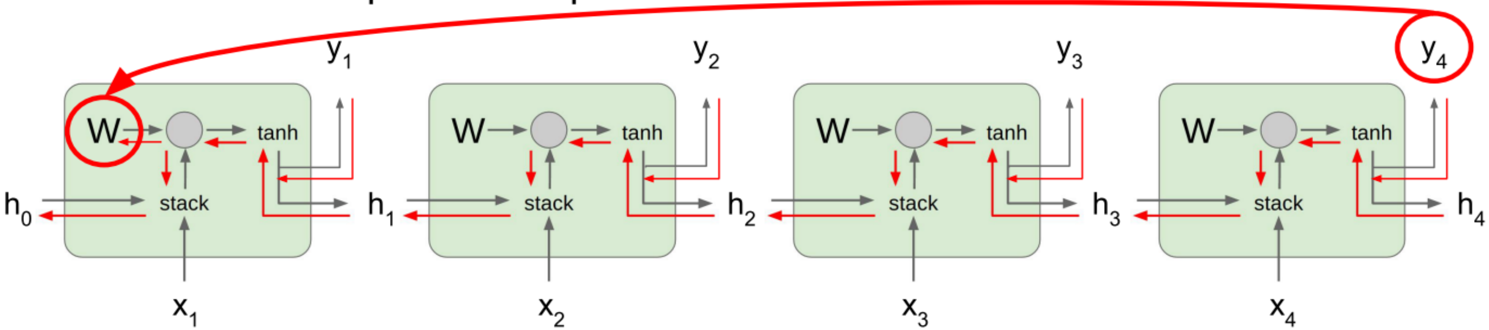


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



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

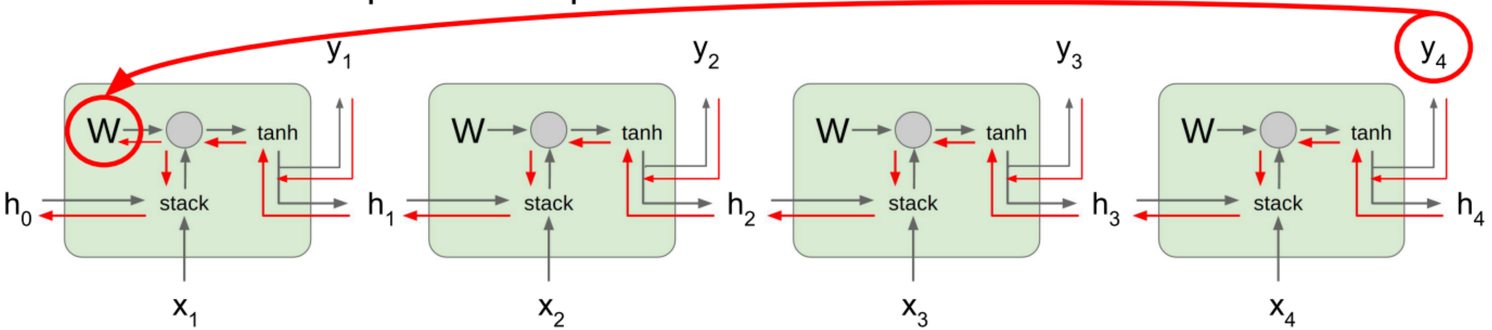
$$\boxed{\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}}$$

此处并不严谨，实际上等式右边只是导数的其中一项，但这一项能提供一些直观的认识。

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



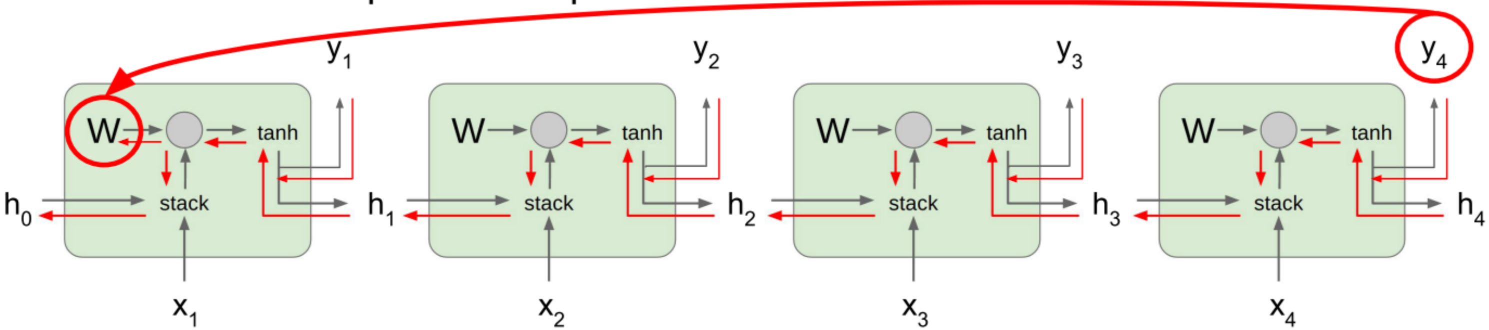
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

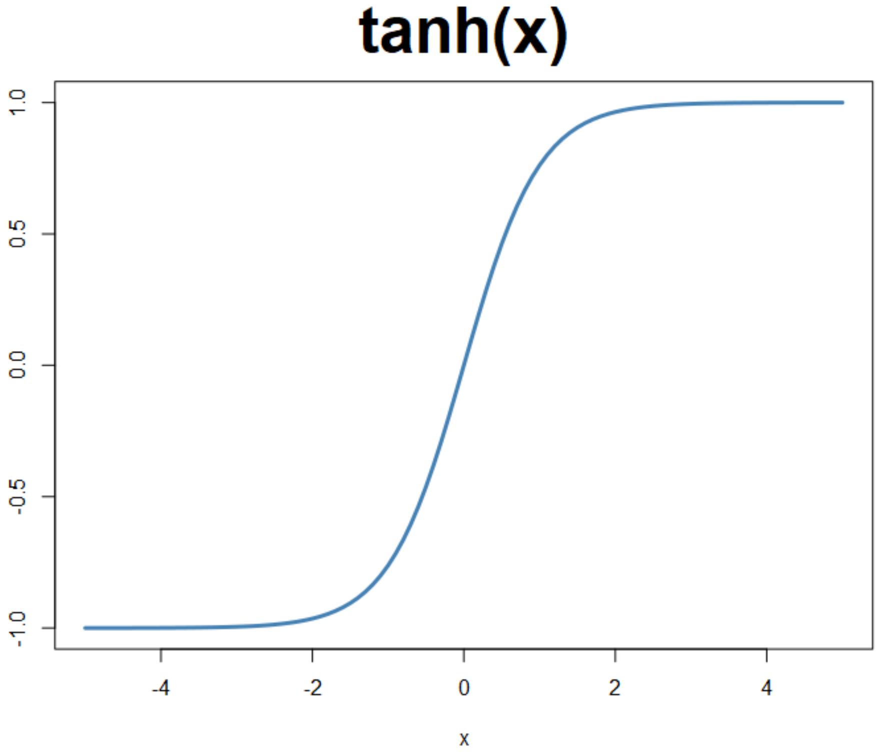
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

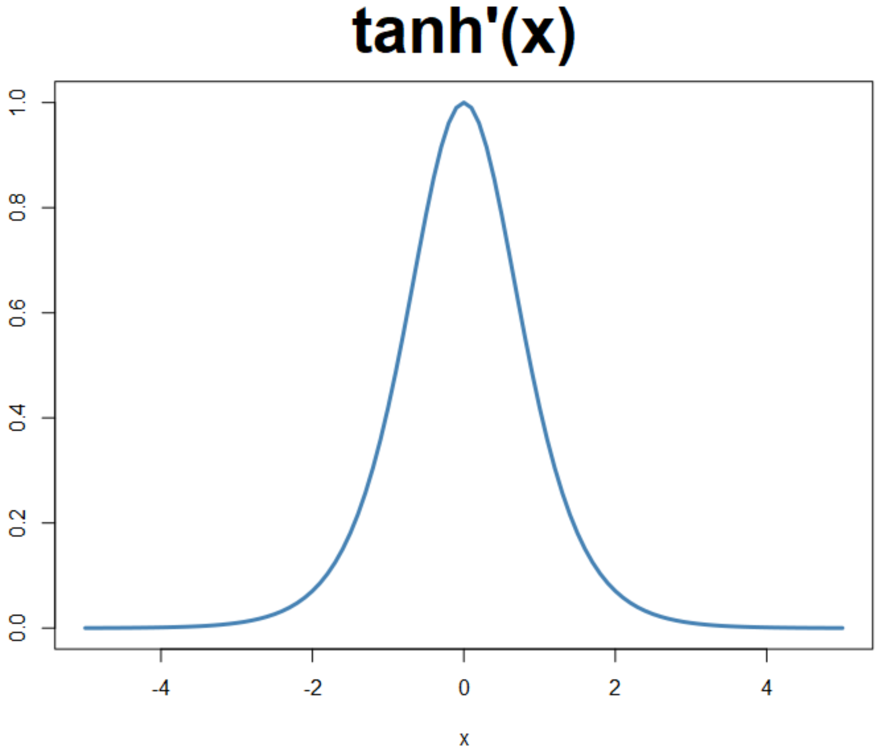
Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \qquad \boxed{\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \boxed{\frac{\partial h_t}{\partial h_{t-1}}}\right)\frac{\partial h_1}{\partial W}$$

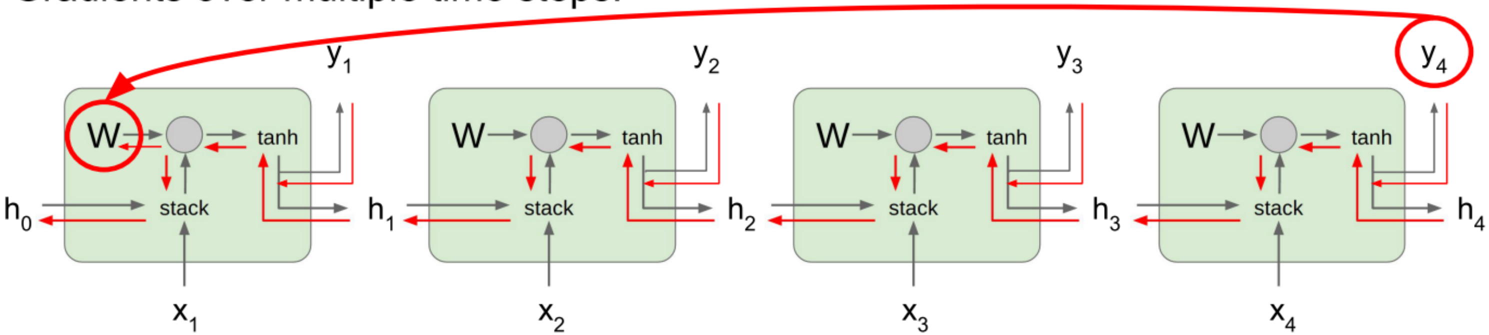

tanh(x)
1.0
0.5
0.0
-0.5
-1.0
-4
-2
0
2
4
x


tanh'(x)
1.0
0.8
0.6
0.4
0.2
0.0
-4
-2
0
2
4
x

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



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

Almost always < 1

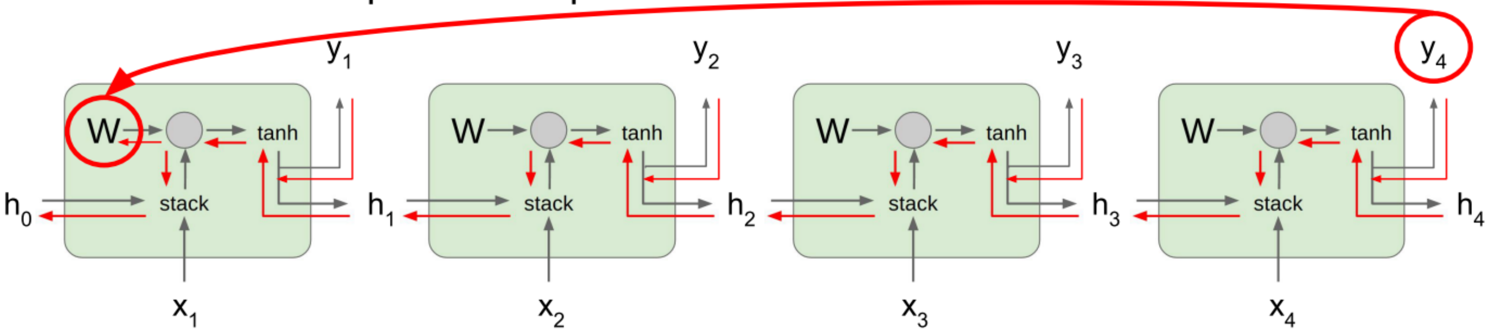
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



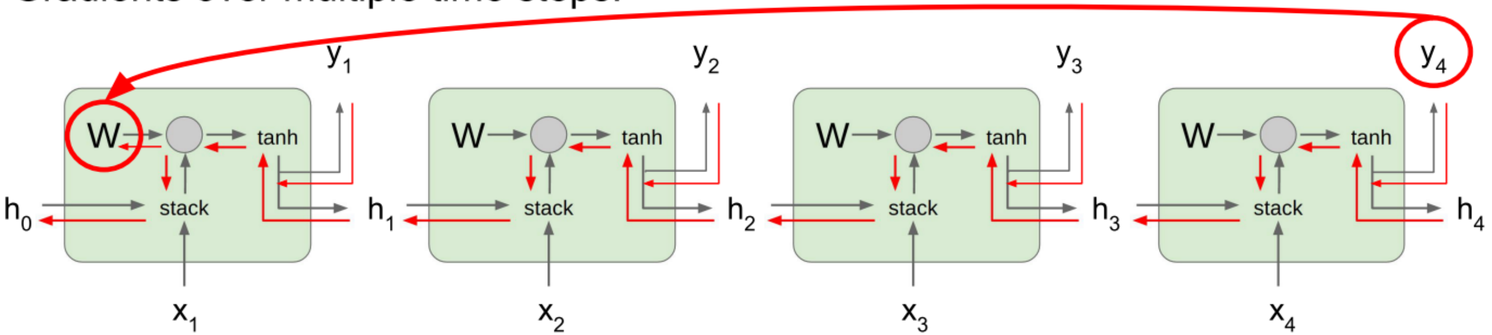
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



What if we assumed no non-linearity?

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

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{in}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

改进方法

- 梯度爆炸：对梯度的上限进行截断
- 梯度消失：更改 RNN 架构

LSTM

- 长短期记忆神经网络
- Long Short-Term Memory

- Hochreiter and Schmidhuber (1997). Long Short Term Memory, Neural Computation.

LSTM

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

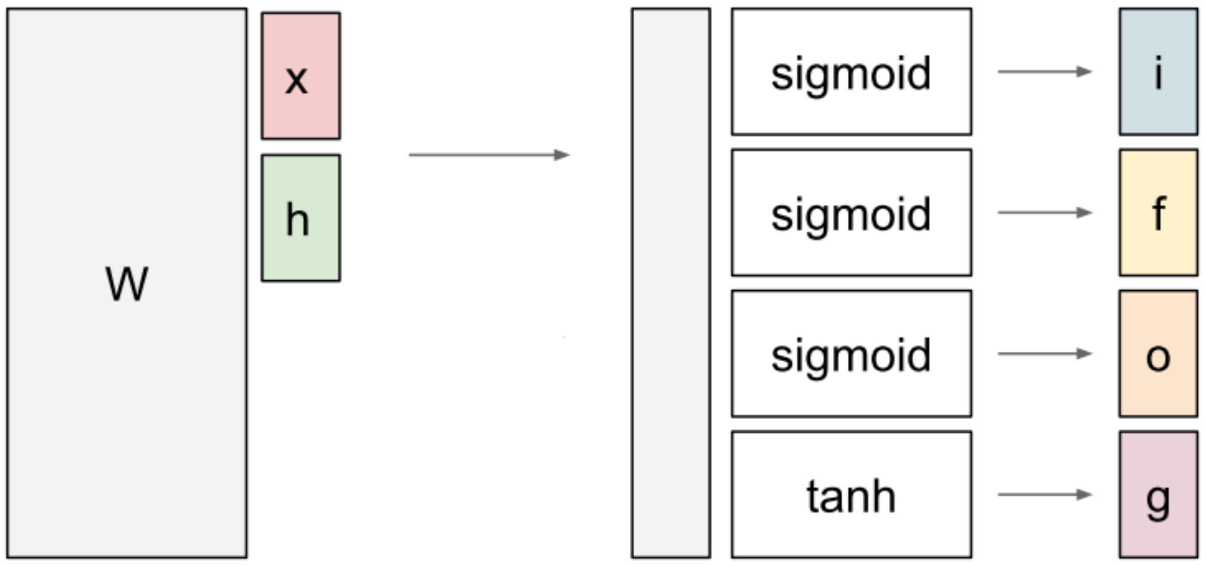
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

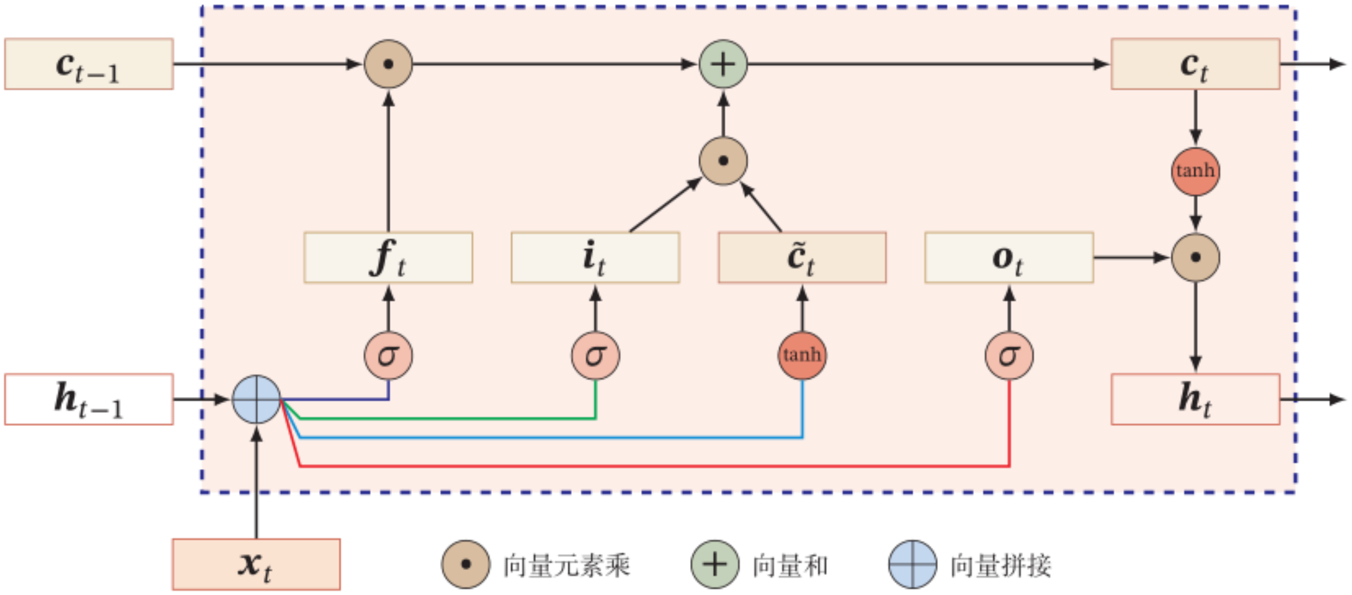
LSTM



$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

LSTM



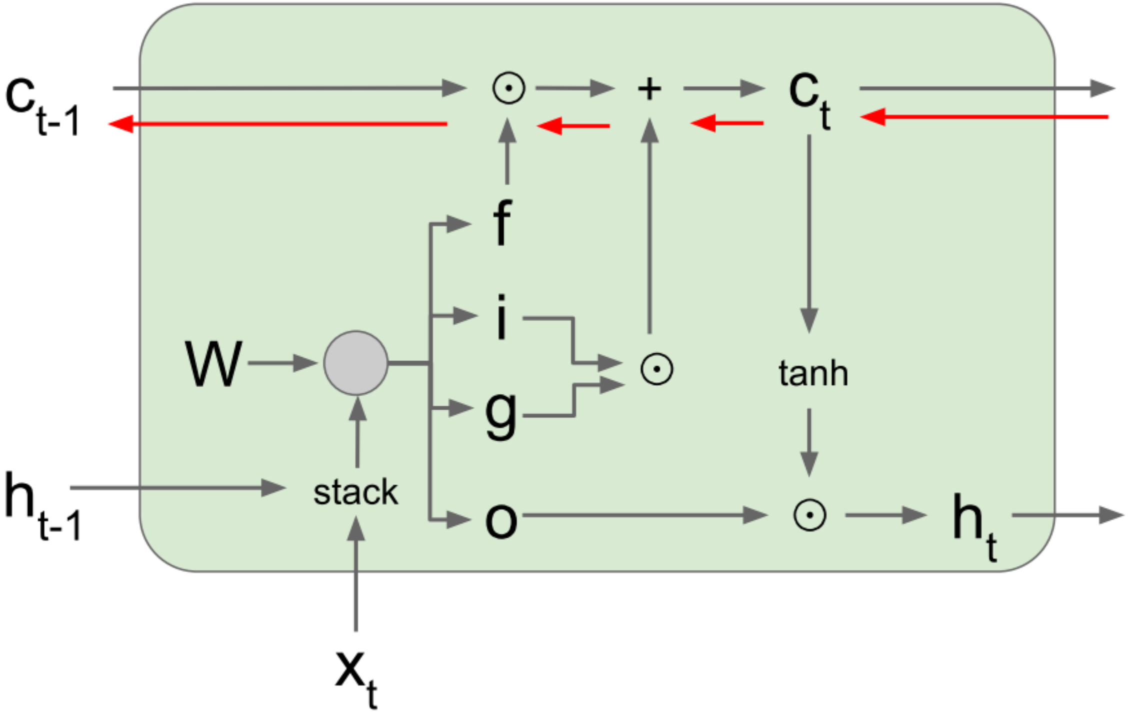
- i：是否写入当前单元
- $g/\tilde{c}$：多大程度上写入当前单元
- f：是否遗忘上一个单元
- o：多大程度上将单元转成隐藏状态

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

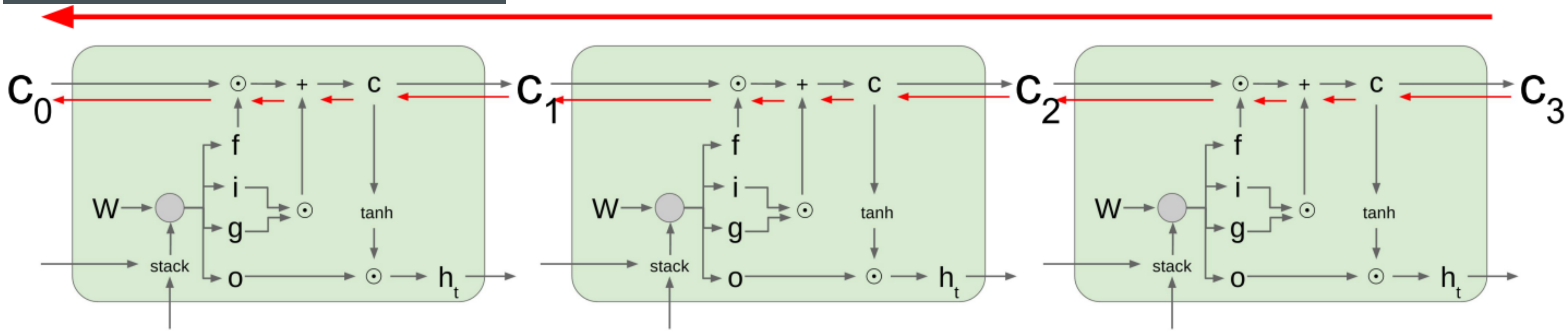
LSTM

- 梯度从 c_t 传向 c_{t-1} 时只牵涉到与 f 的逐元素相乘，没有直接的矩阵乘法



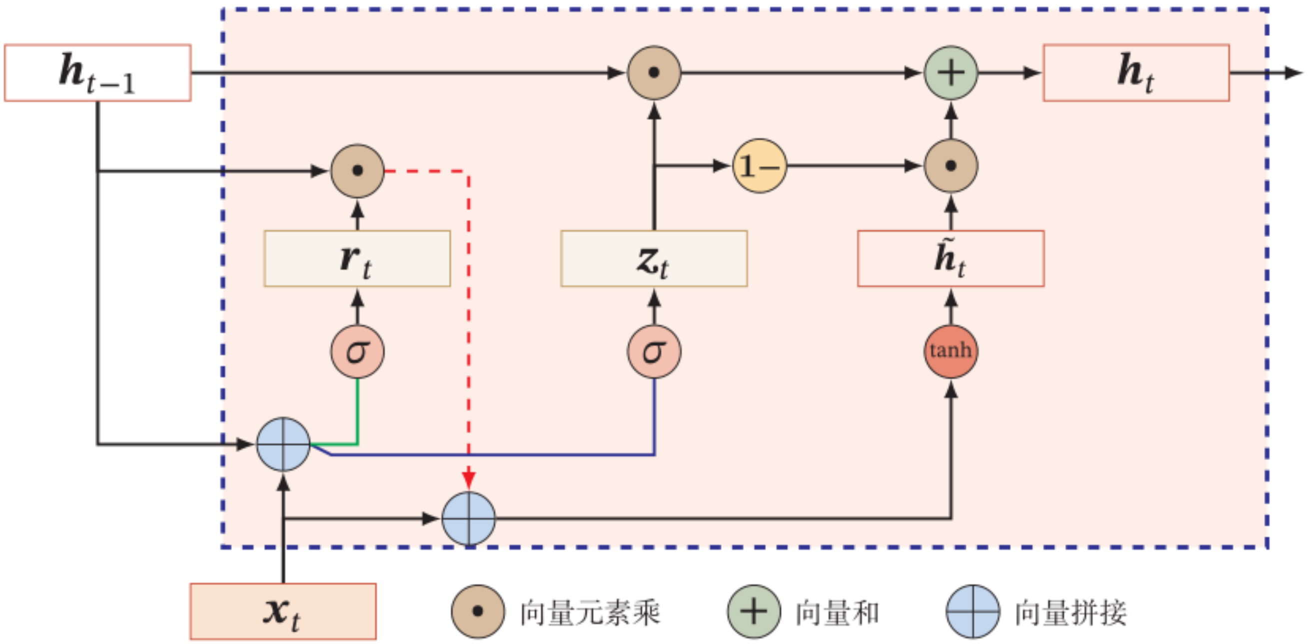
LSTM

- 可以更好地对梯度进行远距离传播
- “没有中间商赚差价”
- 可以类比于残差神经网络



GRU

本页内容取自邱锡鹏
《神经网络与深度学习》



重置门

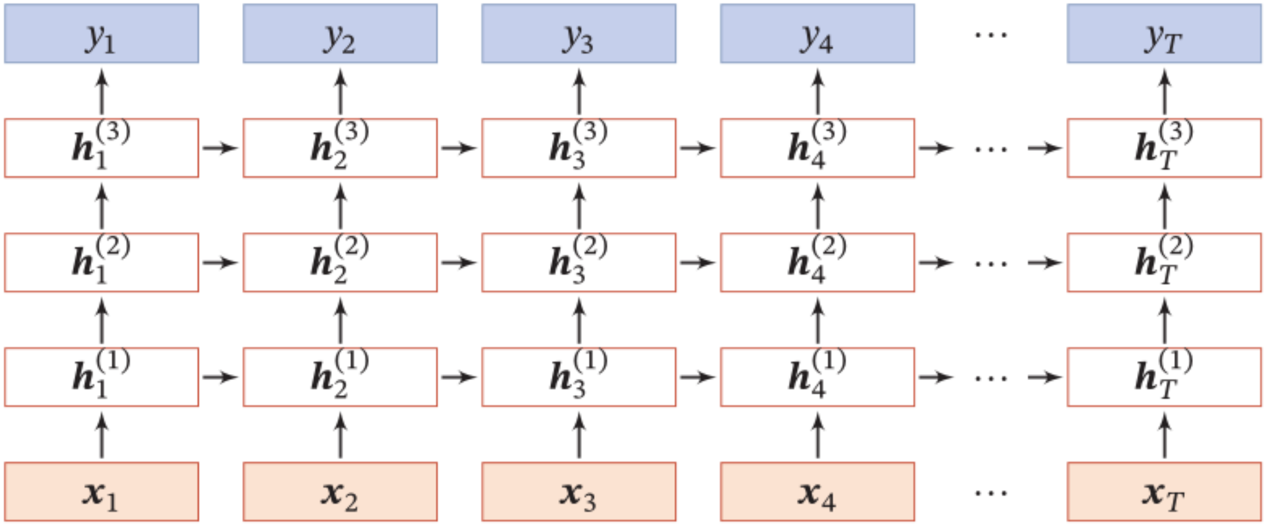
$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r), \quad \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z), \quad \mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,$$

更新门

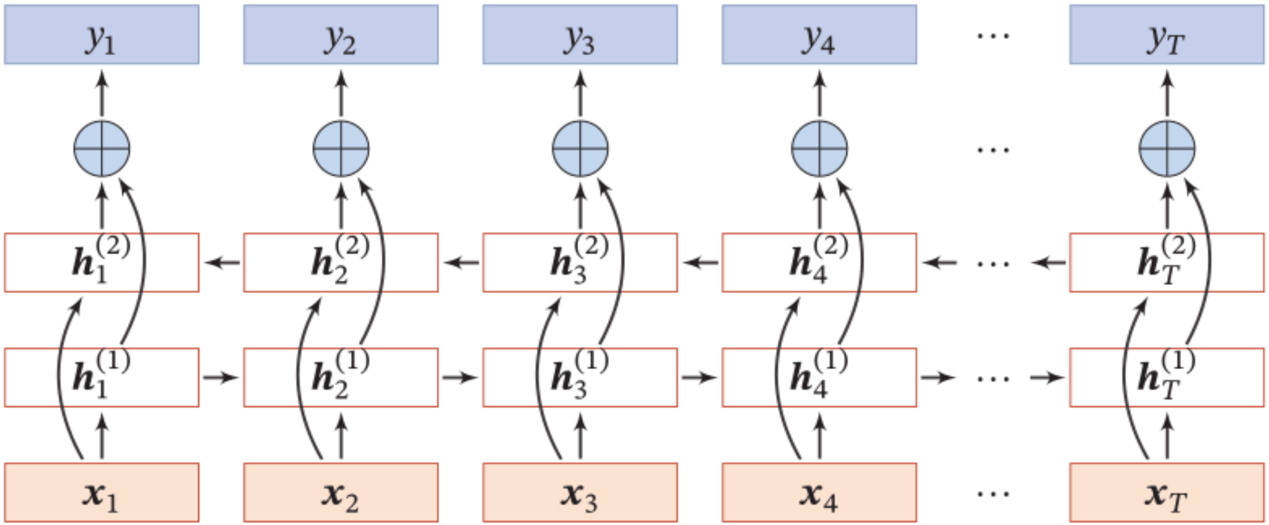
其他架构

- 多层堆叠 RNN



其他架构

- 双向 RNN



y_1
y_2
y_3
y_4
y_T
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$h_4^{(2)}$
$h_T^{(2)}$
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_4^{(1)}$
$h_T^{(1)}$
x_1
x_2
x_3
x_4
x_T