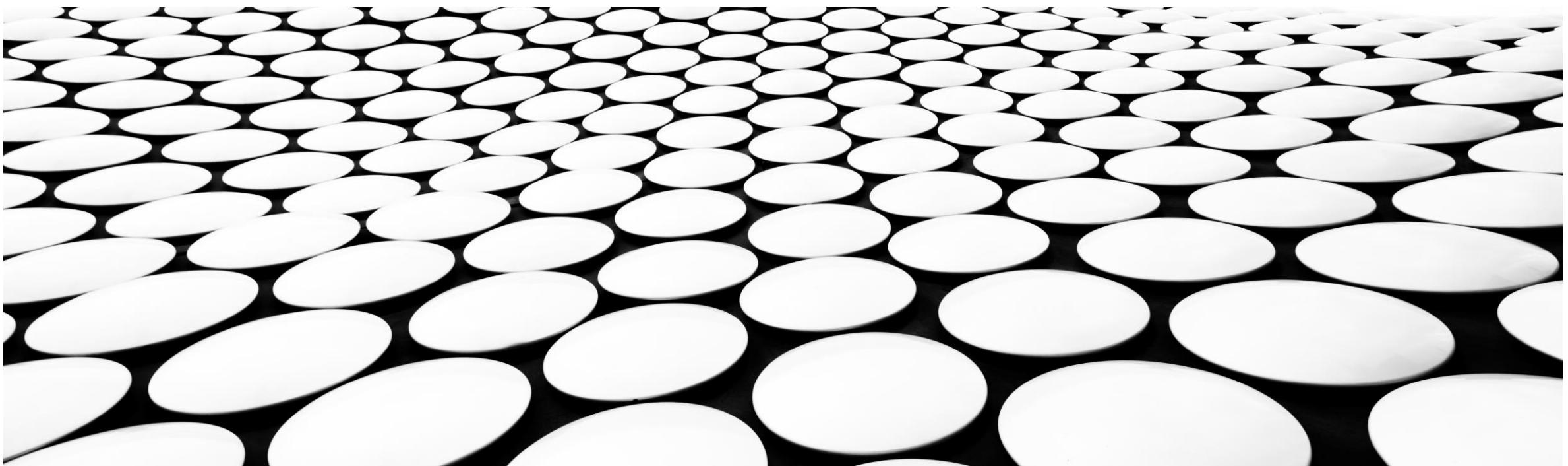

深度学习

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	棱	1
2172	禹	1
2173	据	1
1962	莽	1

建立字典

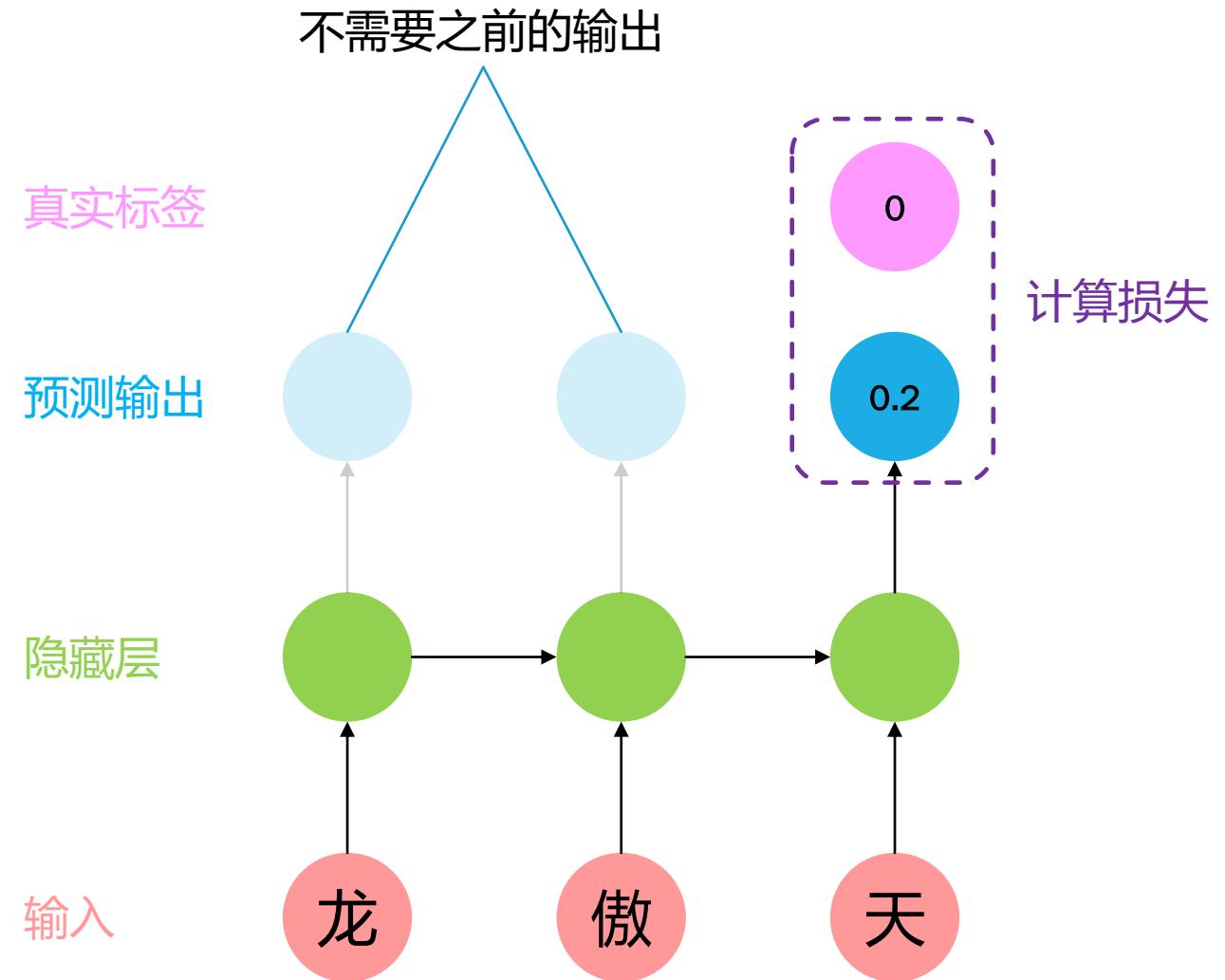
■ 将500个常用字作为字典

```
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

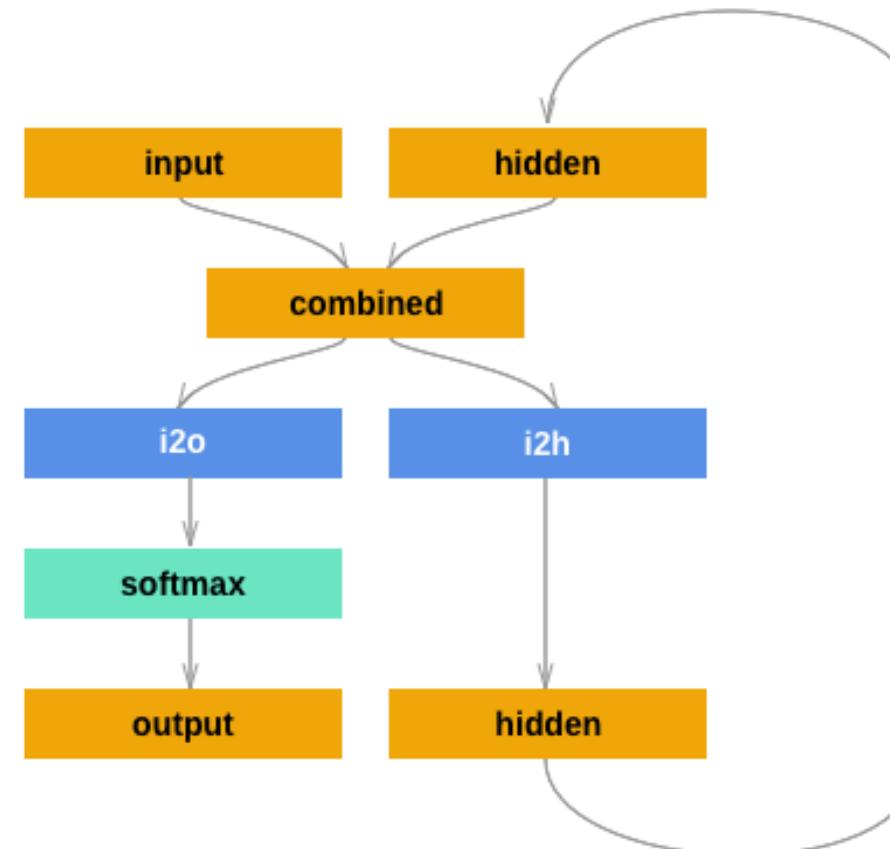
建模原理



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

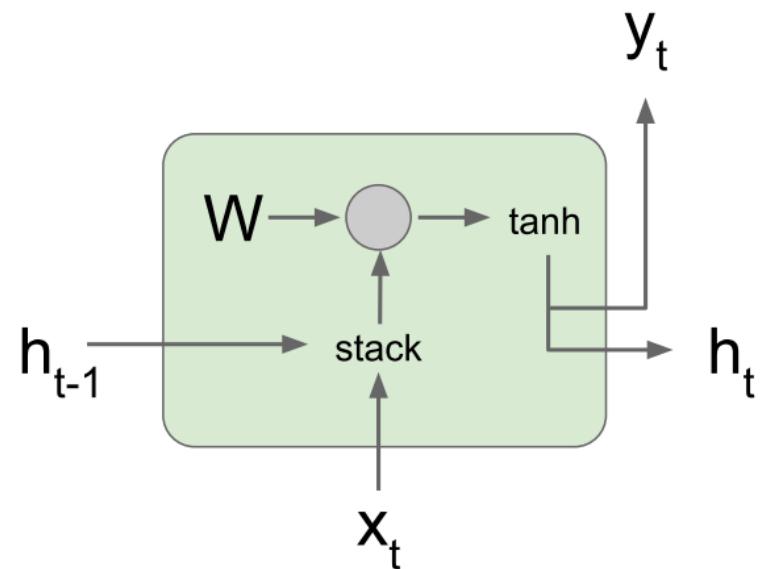
- 相比于前馈神经网络和 CNN, RNN 有其独特的性质
- 优点
 - 处理任意长的序列数据
 - 利用历史信息
 - 参数数量不随序列变长而增加
- 缺点
 - 序列很长时计算量非常大
 - 梯度消失/爆炸问题

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

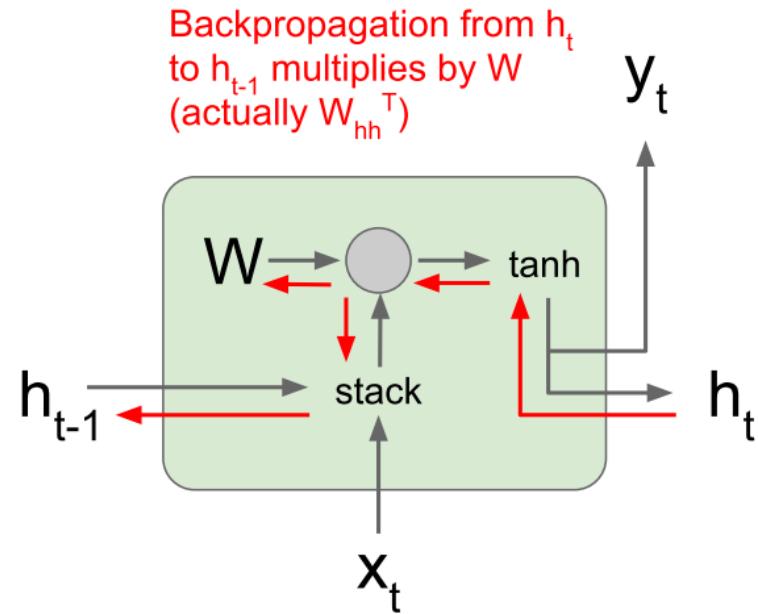
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

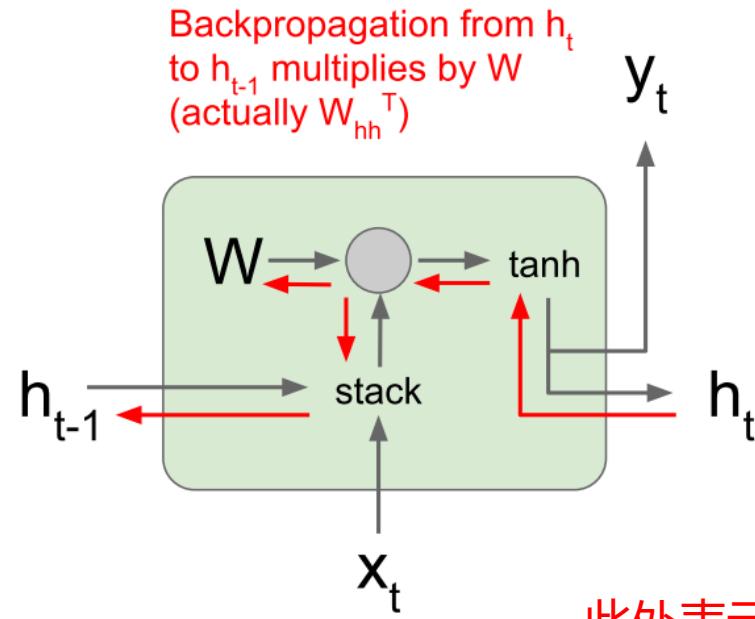
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

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



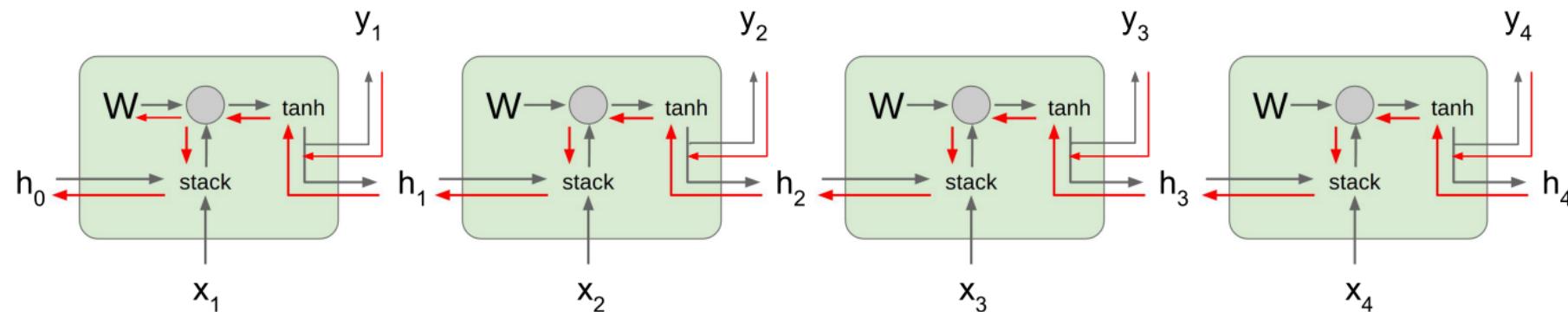
$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

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

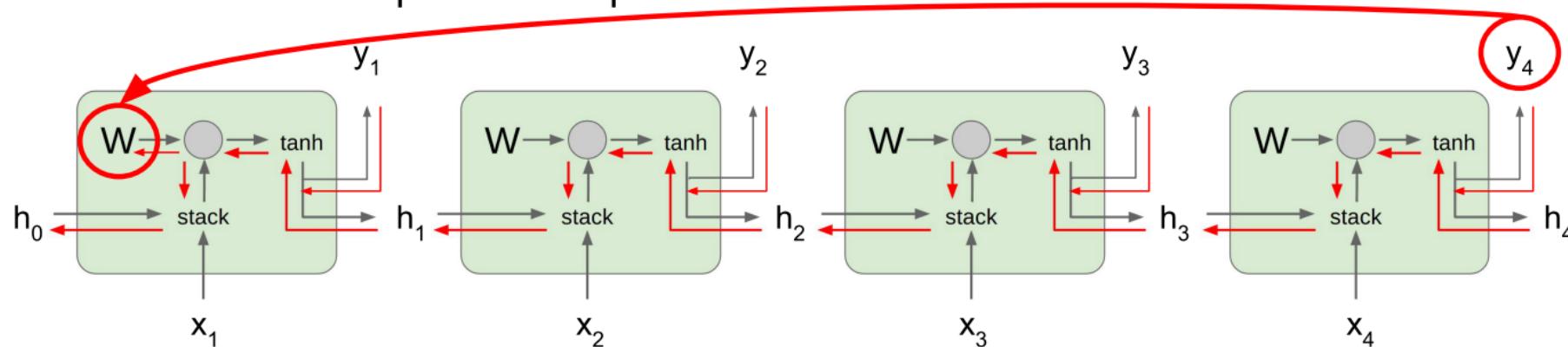


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



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

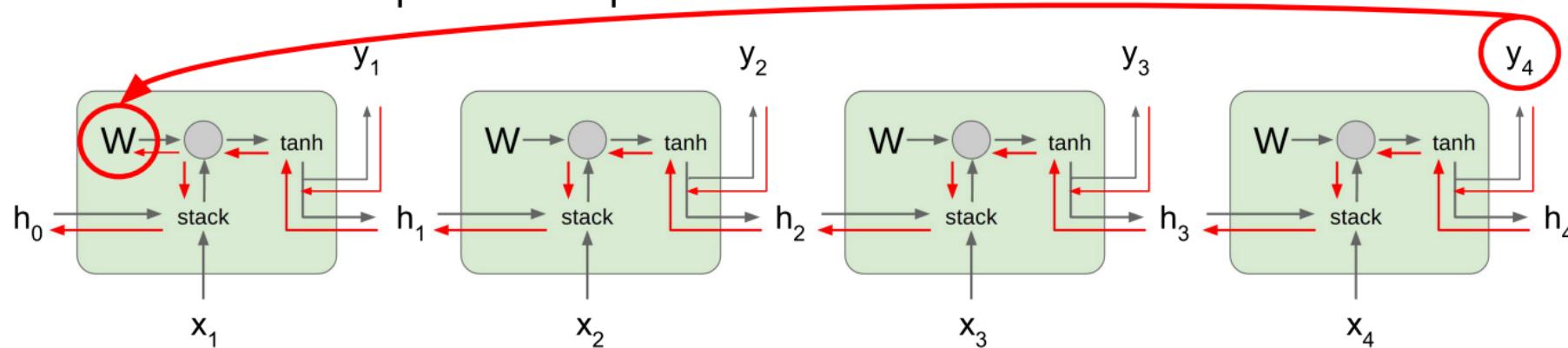
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_1}{\partial W}$$

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

Gradients over multiple time steps:

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



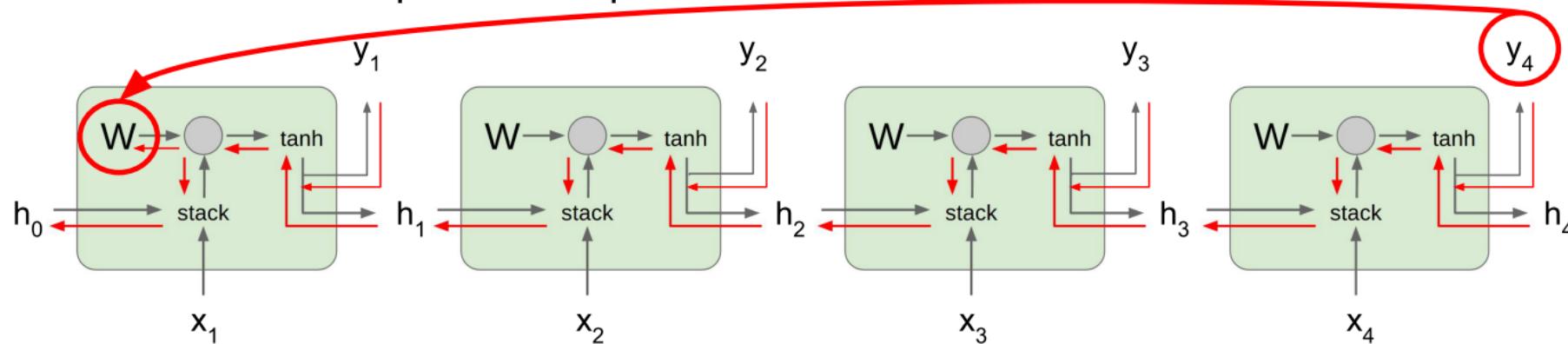
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

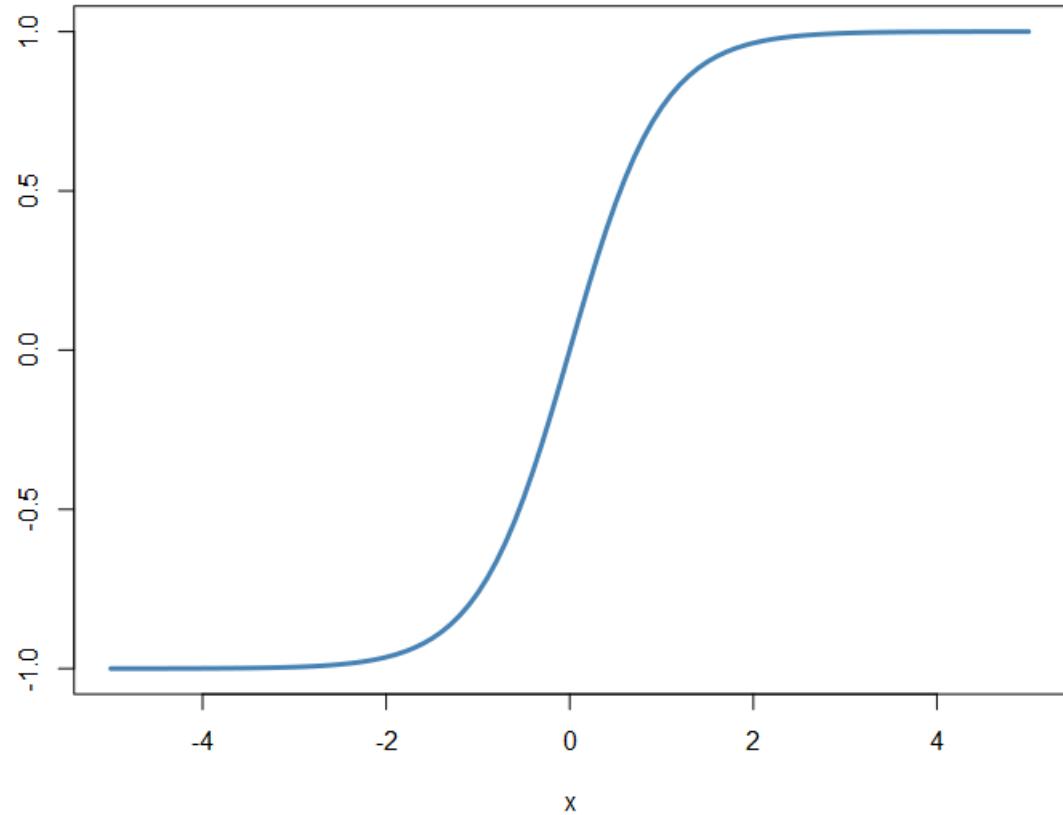
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



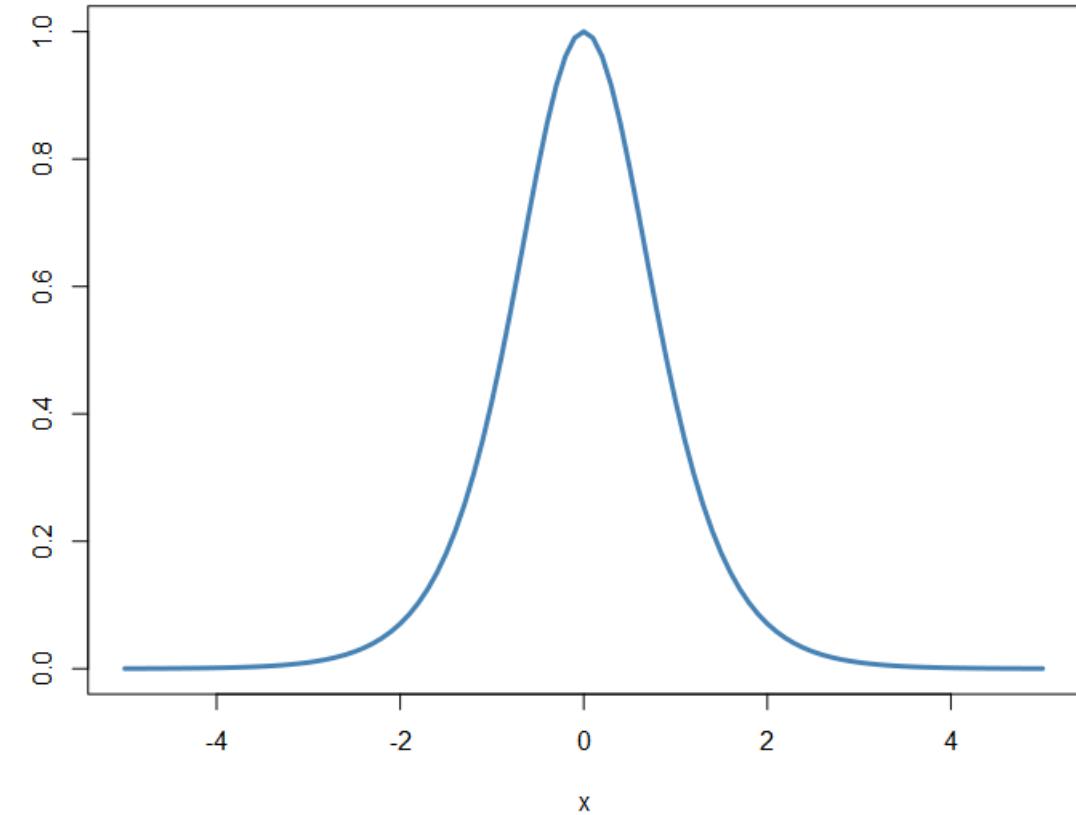
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \boxed{\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}}$$

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

tanh(x)



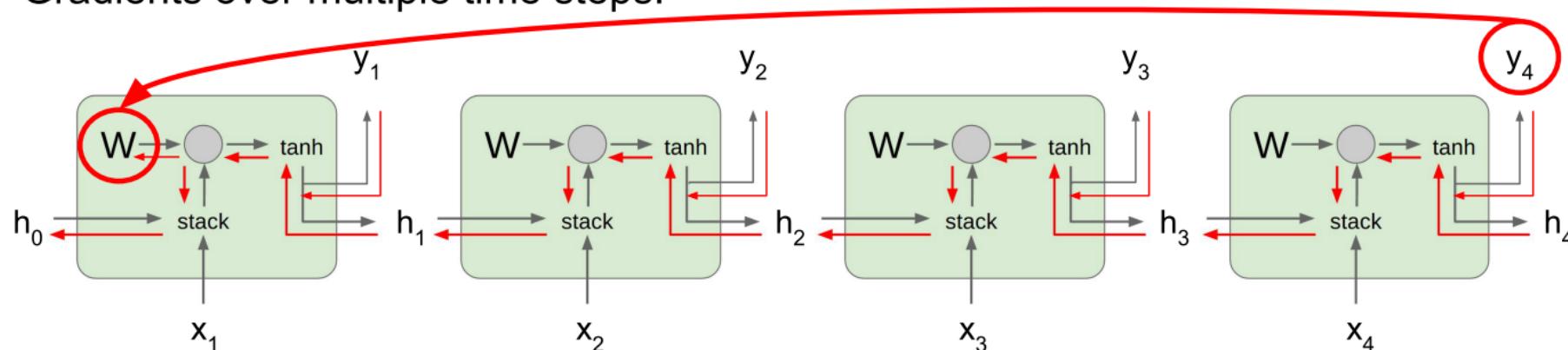
tanh'(x)



Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



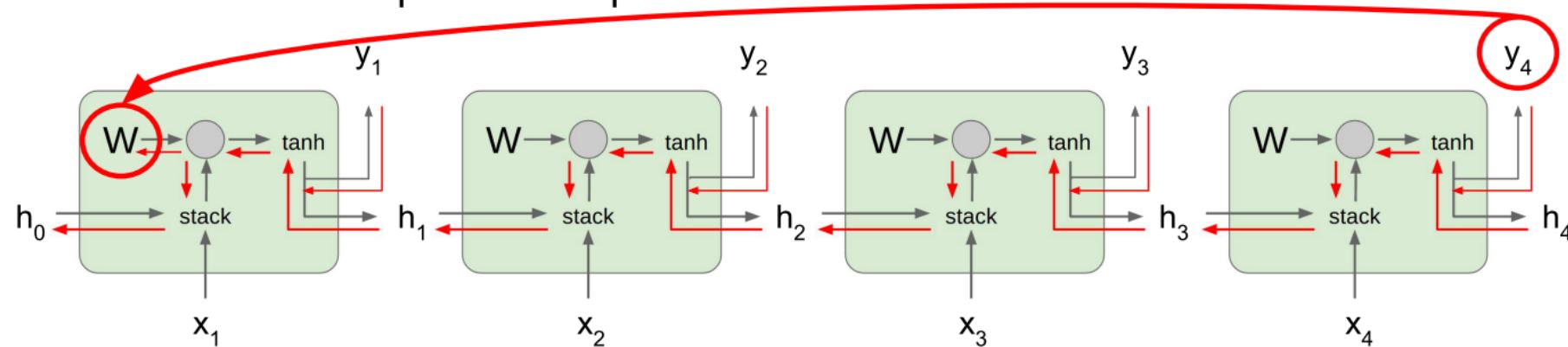
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \text{Almost always } < 1 \\ \text{Vanishing gradients}$$

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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

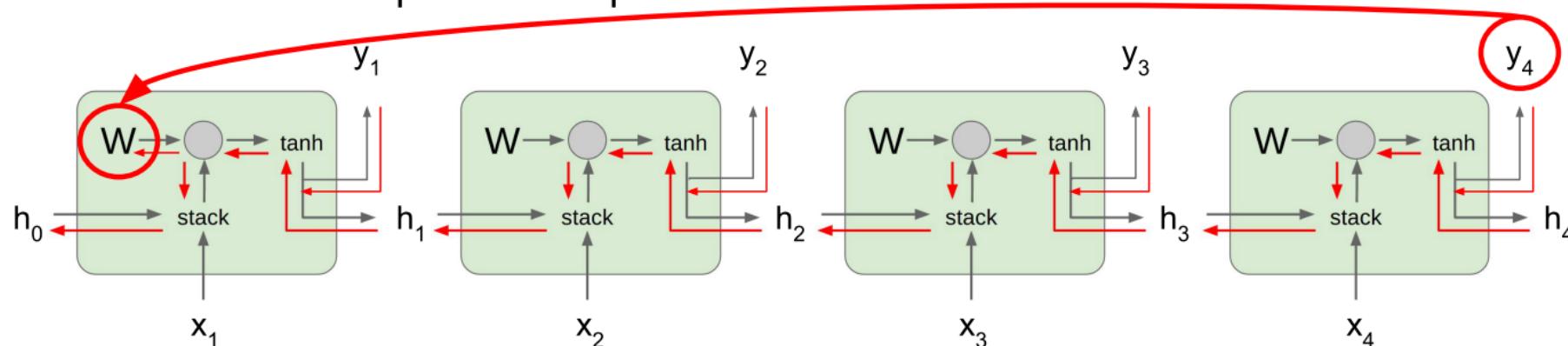


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \text{What if we assumed no non-linearity?}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

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



What if we assumed no non-linearity?

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

Largest singular value > 1:
Exploding gradients

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

Largest singular value < 1:
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

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

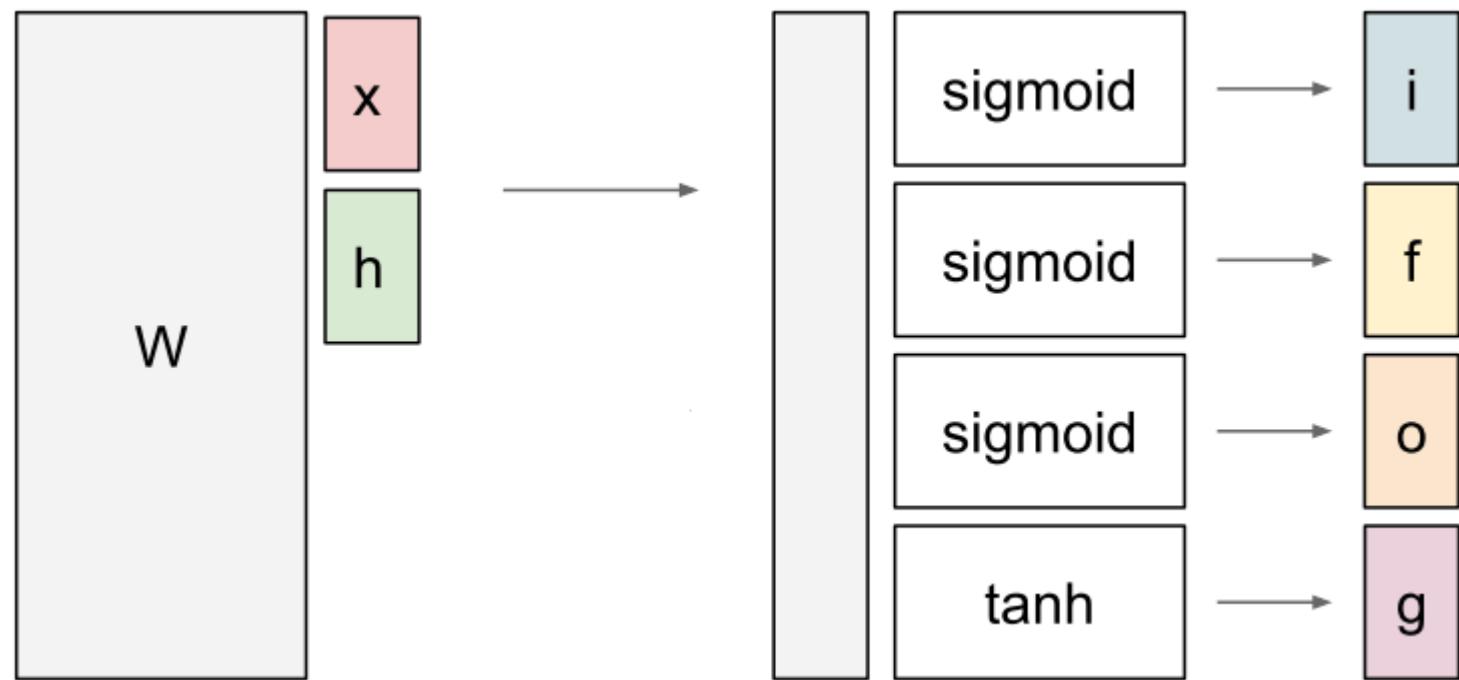
LSTM

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

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

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

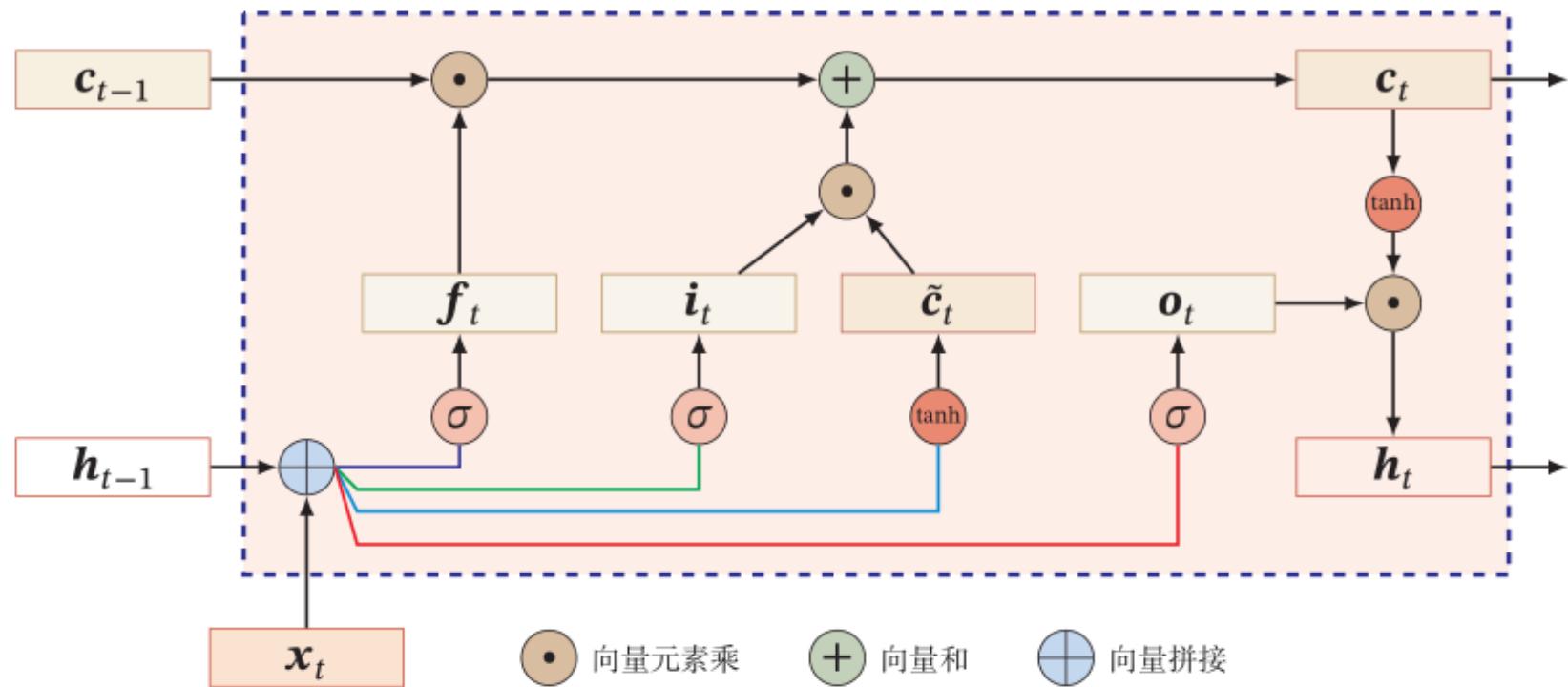
LSTM



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

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

LSTM



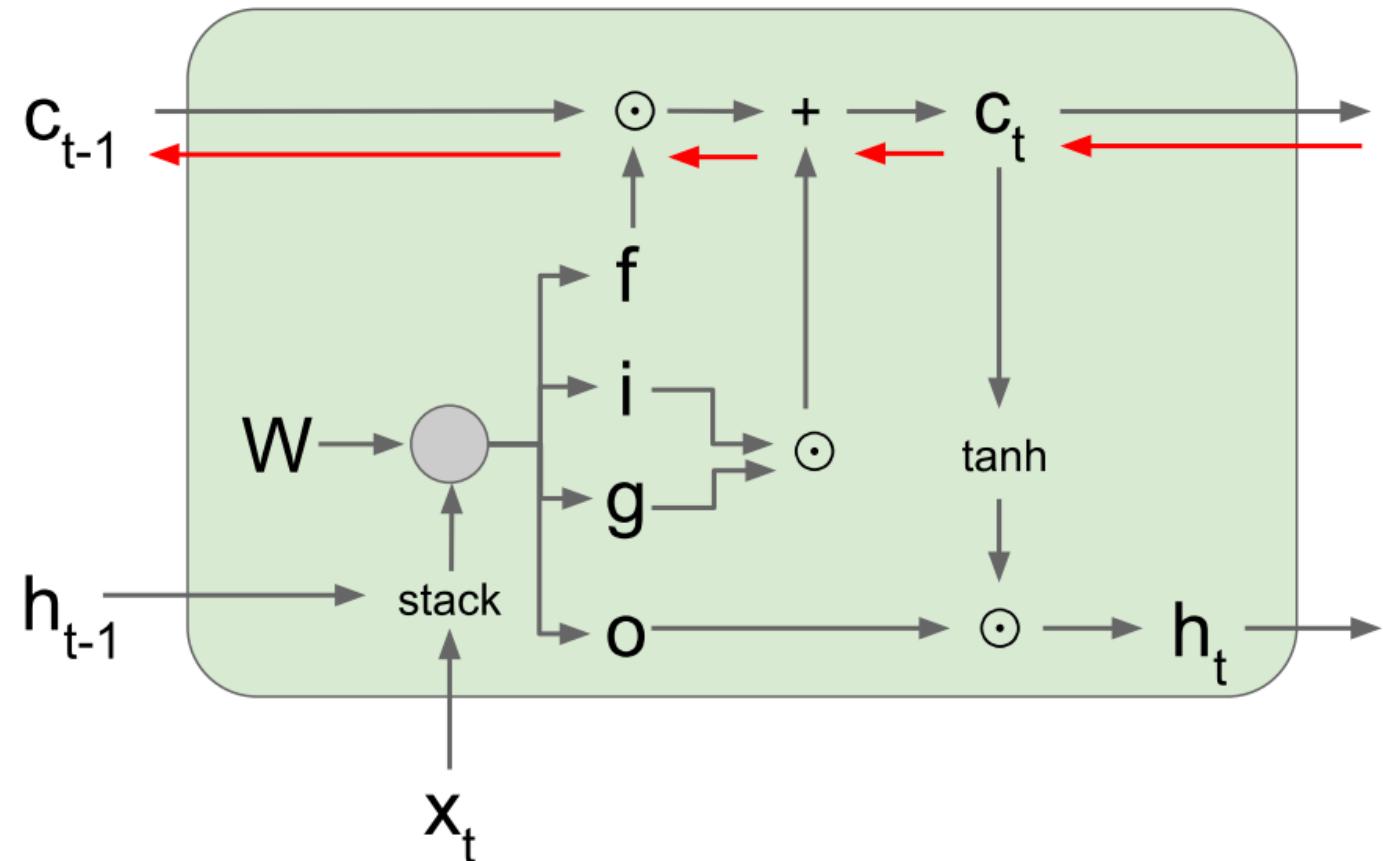
- i : 是否写入当前单元
- g/\tilde{c} : 多大程度上写入当前单元
- f : 是否遗忘上一个单元
- o : 多大程度上将单元转成隐藏状态

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

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

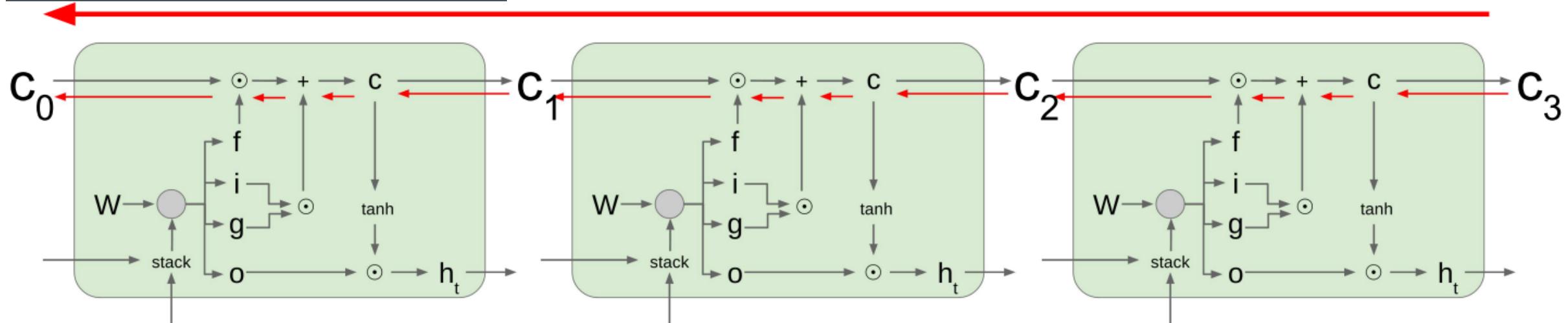
LSTM

- 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



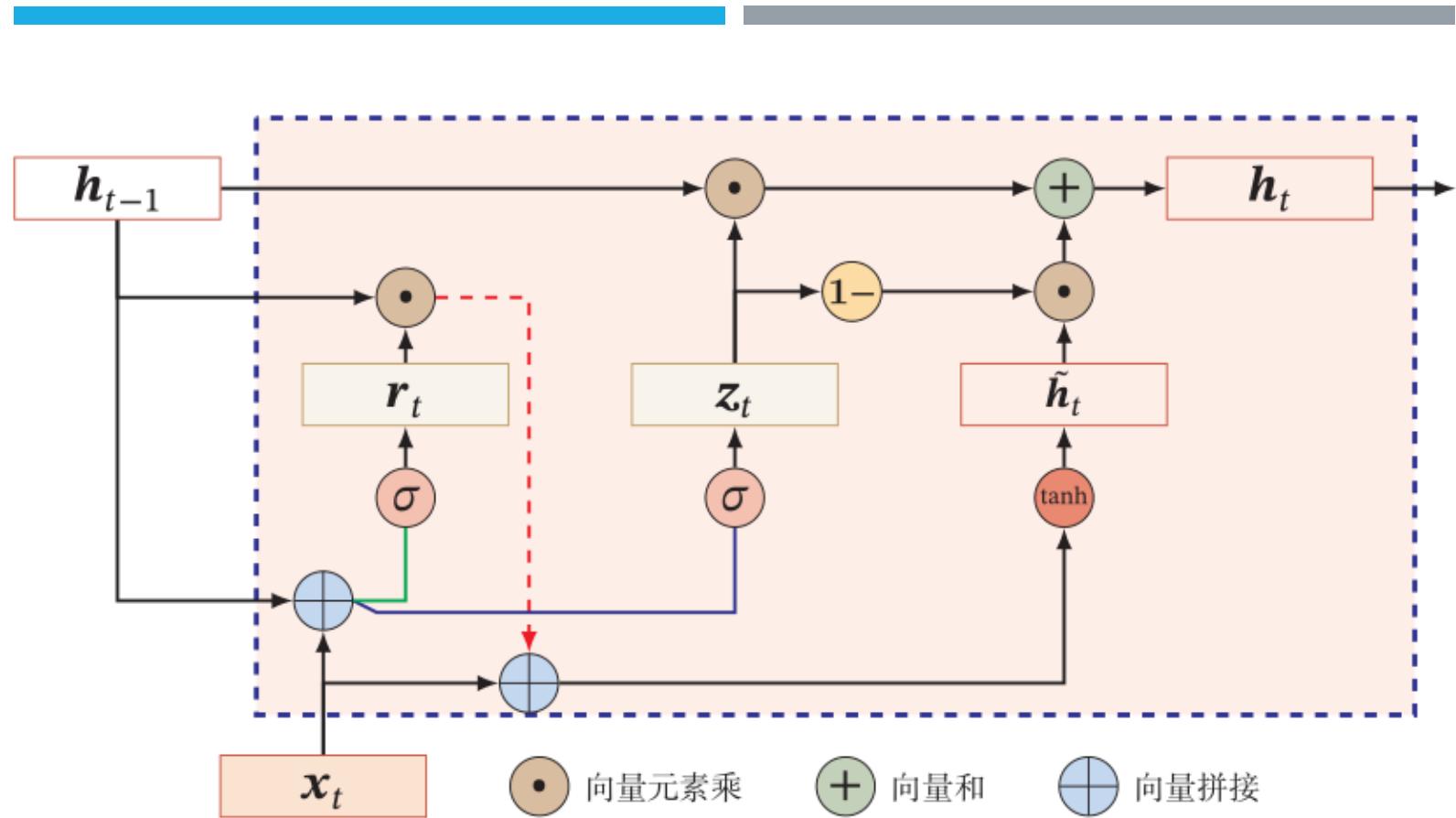
LSTM

- 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

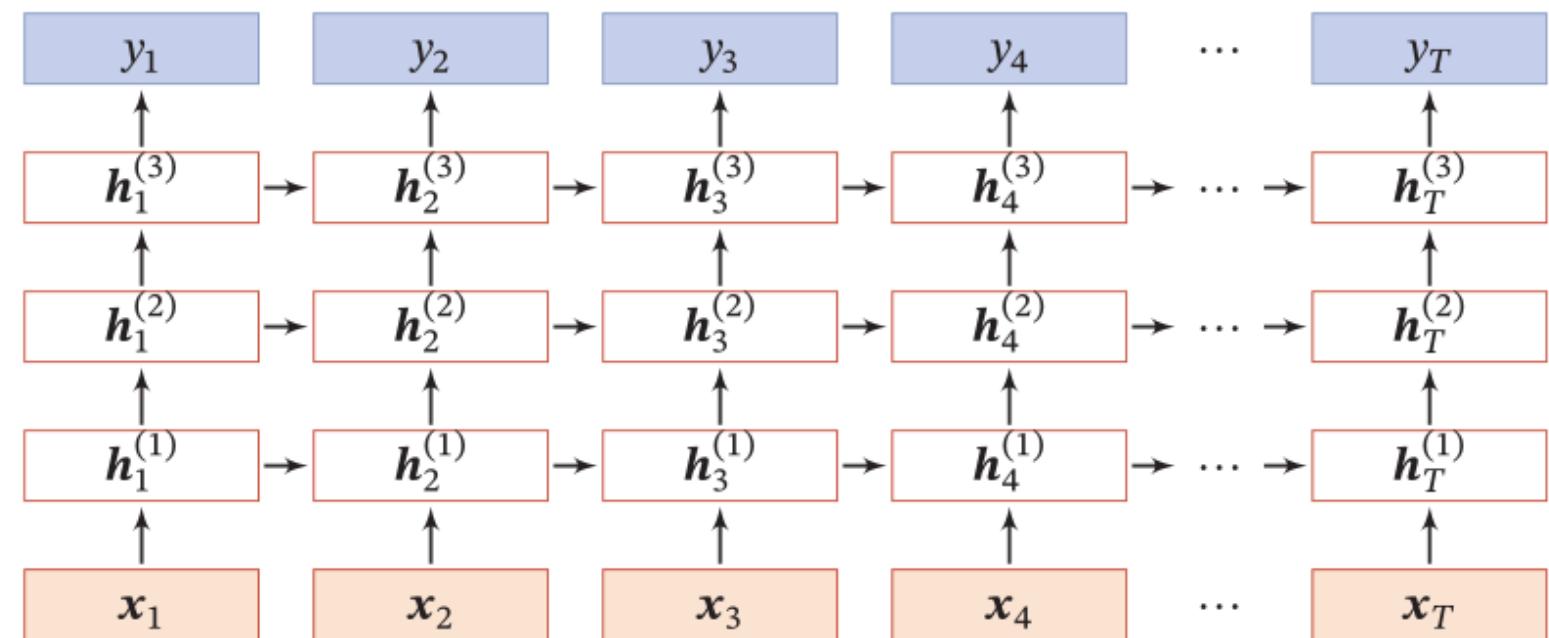
$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r), \quad \tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c (\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

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

更新门

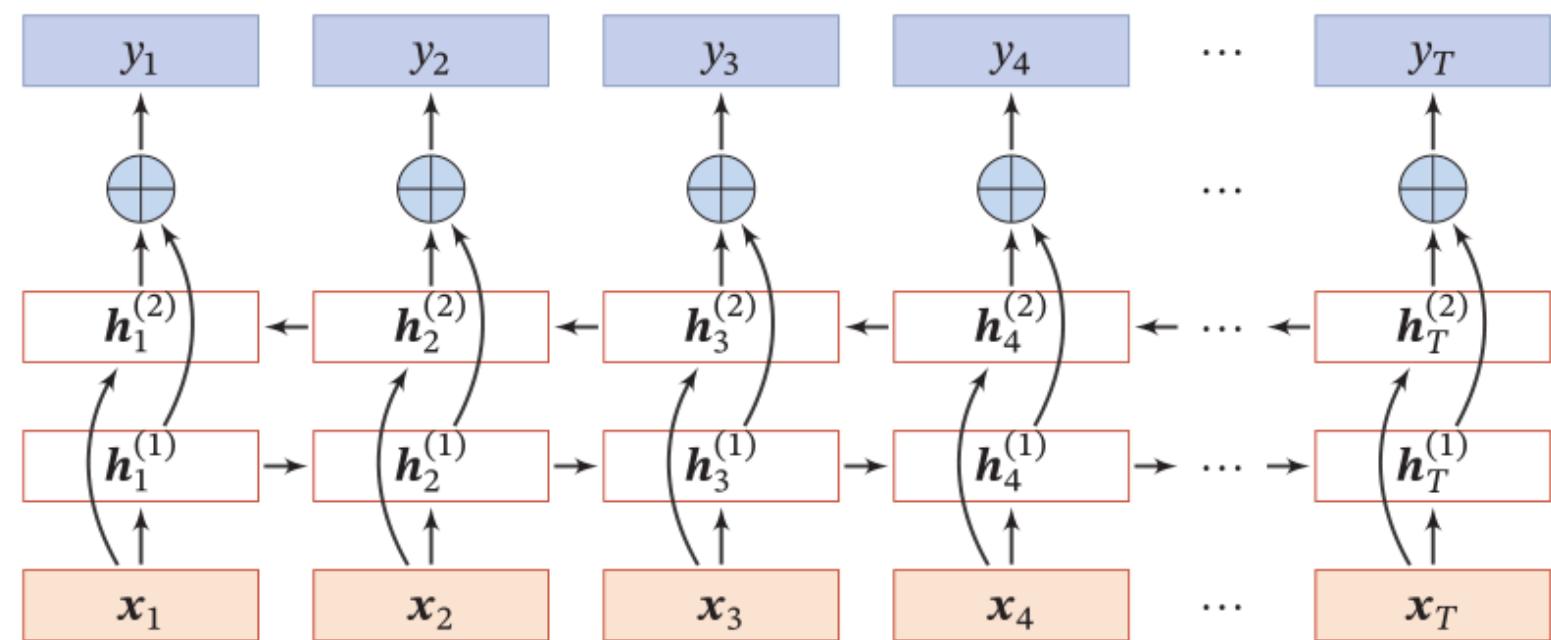
其他架构

■ 多层堆叠 RNN



其他架构

■ 双向 RNN



RNN生成模型

人名生成

- 数据来源
- <https://github.com/wainshine/Chinese-Names-Corpus>
- >42万条英文翻译人名

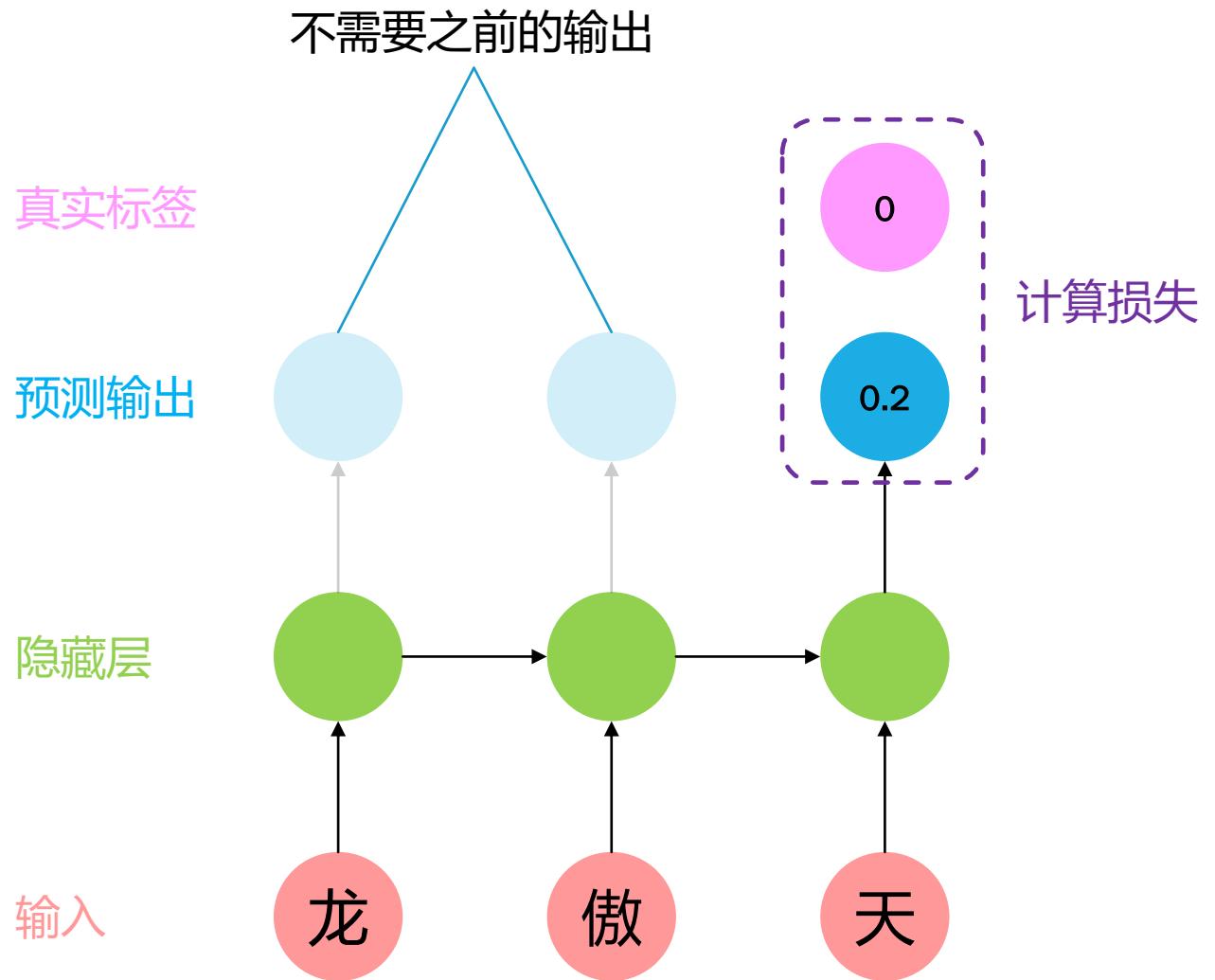
5 阿阿巴伊	92901 帕多奈小	144332 海尔波莱宁	221388 劳豆斯基	319237 萨布多维奇
6 阿阿贝伊奥卢	92962 德罗茨	144333 海尔博恩	221389 劳登斯卡	319238 萨蒂纳拉扬
7 阿阿比金	92963 德罗代	144334 海尔博伊	221390 劳迪	319239 萨蒂南
8 阿阿尔坦	92964 德罗德	144335 海尔布科	221391 劳迪安斯基	319240 萨蒂宁
9 阿阿吉尔	92965 德罗德洛	144336 海尔布龙	221392 劳迪奥	319241 萨蒂农德
10 阿阿卡伊	92966 德罗迪	144337 海尔布龙纳	221393 劳迪德	319242 萨蒂诺
11 阿阿拉尔	92967 德罗蒂	144338 海尔布伦	221394 劳迪尔	319243 萨蒂皮
12 阿阿兰坦	92968 德罗尔	144339 海尔布特	221395 劳迪弗	319244 萨蒂什
13 阿阿姆恰姆	92969 德罗尔克	144340 海尔采	221396 劳迪迈茨基	319245 萨蒂斯
14 阿阿内尔	92970 德罗尔斯哈根	144341 海尔采尔	221397 劳迪诺	319246 萨蒂亚
15 阿阿奇迪肯	92971 德罗法	144342 海尔采戈夫	221398 劳迪奇	319247 萨蒂亚查兰
16 阿阿奇汗	92972 德罗夫	144343 海尔采戈娃	221399 劳迪奇纳	319248 萨蒂亚尔
17 阿阿恰亚克	92973 德罗夫斯基	144344 海尔采戈瓦茨	221400 劳迪萨	319249 萨蒂亚吉特
18 阿阿韦尔迪	92974 德罗盖	144345 海尔采格	221401 劳迪斯	319250 萨蒂亚凯图
19 阿阿哲克	92975 德罗盖奥	144346 海尔采格菲	221402 劳迪西	319251 萨蒂亚纳拉亚纳
20 阿阿哲克奥卢	92976 德罗盖蒂	144347 海尔采格福尔维	221403 劳迪西奥	319252 萨蒂亚纳拉扬
21 阿埃蒂斯	92977 德罗盖伊	144348 海尔采格哈齐	221404 劳迪希奇	319253 萨蒂亚纳拉因
22 阿埃多	92978 德罗戈	144349 海尔采格豪尔米	221405 劳迪亚	319254 萨蒂亚南达
23 阿埃尔	92979 德罗戈菲	144350 海尔采格利亚	221406 劳迪耶罗	319255 萨蒂亚南德
24 阿埃尔曼	92980 德罗戈雷亚努	144351 海尔采吉	221407 劳蒂	319256 萨蒂亚帕尔
25 阿埃费菜	92981 德罗戈什	144352 海尔采尼克	221408 劳蒂艾宁	319257 萨蒂亚普里亚
26 阿埃季	92982 德罗格	144353 海尔察	221409 劳蒂奥	319258 萨蒂延德拉
27 阿埃科博耶	92983 德罗格雷	144354 海尔茨	221410 劳蒂奥阿霍	319259 萨蒂扬
28 阿埃拉特尔	92984 德罗格里	144355 海尔茨卡	221411 劳蒂奥拉	319260 萨丁
29 阿埃莱内伊	92985 德罗格里亚	144356 海尔茨库	221412 劳蒂卡	319261 萨丁马
30 阿埃里略	92986 德罗格塞特	144357 海尔达尔	221413 劳蒂拉	319262 萨丁尼亚
31 阿埃利萨贝泰	92987 德罗格特	144358 海尔代洛	221414 劳蒂莫	319263 萨丁奇
32 阿埃利翁	92988 德罗古	144359 海尔代伊	221415 劳蒂齐	319264 萨乐
	92989 德罗古尔	144360 海尔当	221416 劳蒂西	319265 萨杜



原理

回顾

■ 分类问题 (性别预测)



回顾

- 分类问题 (性别预测)
- 本质上是对 $p(y|x_1, \dots, x_T)$ 进行建模
- 这是一个条件分布
- Y 是标签, 视作服从 Bernoulli 分布
- $P(Y = 1)$ 的取值依赖于序列 x_1, \dots, x_T

生成模型

- 而序列生成模型是对 $p(x_1, \dots, x_T)$ 进行建模
- 这是一个联合分布
- 每个 X_t 代表一个字的 one-hot 向量
- 如何利用 RNN 对联合分布进行建模？

核心思想

- 核心思想依然是极大似然估计
- 条件概率分解公式
$$p(x_1, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_T|x_1, \dots, x_{T-1})$$
- 事实上，这也是当前许多大语言模型的根本出发点
- 它表明，联合分布的建模可以拆解成一系列的序列分类问题

生成模型

- 人名生成可以看作是序列预测序列的问题
- 例如如下观测数据

张三
小五郎
福尔摩斯

- 可以转换成序列预测问题

自变量

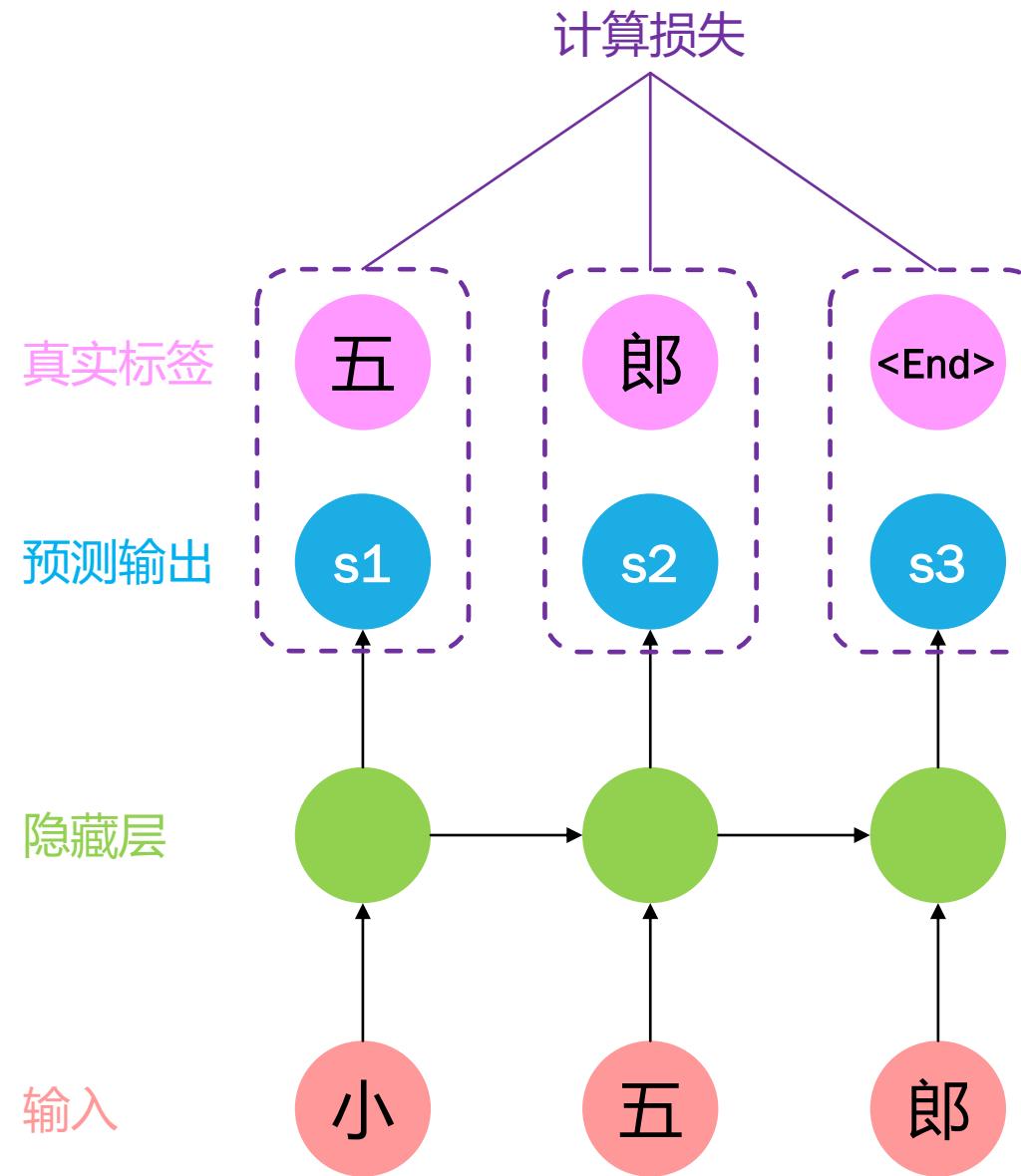
[张, 三]
[小, 五, 郎]
[福, 尔, 摩, 斯]

因变量

[三, <End>]
[五, 郎, <End>]
[尔, 摩, 斯, <End>]

- <End>是特殊字符，表示名字的结尾

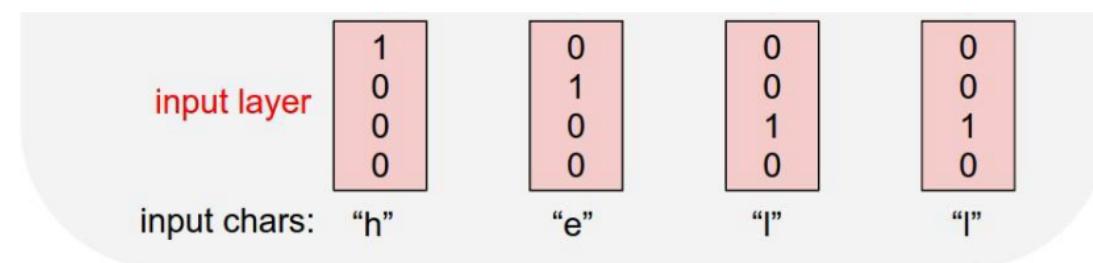
模型原理



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

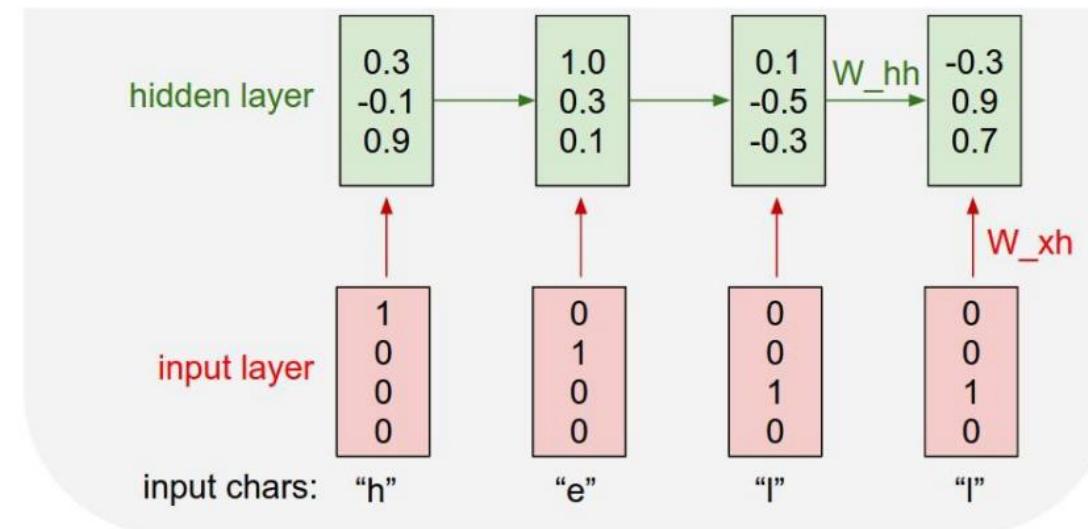


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

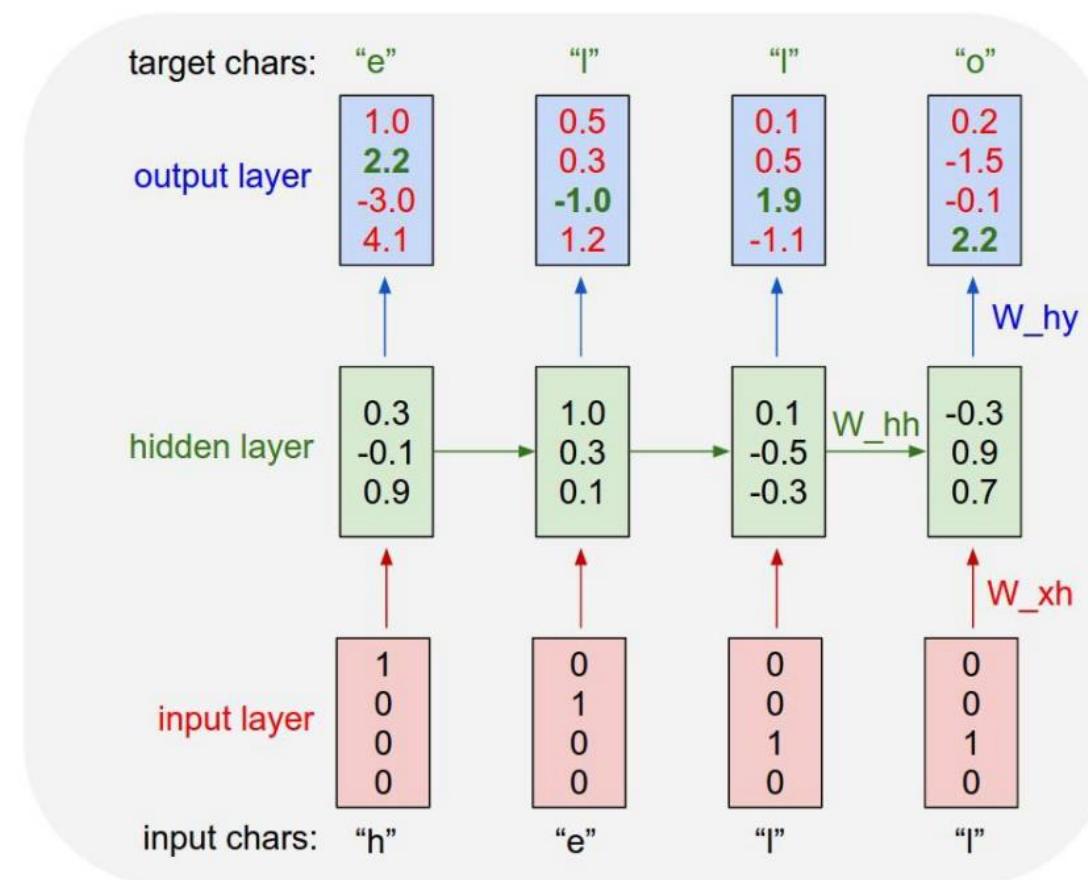
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

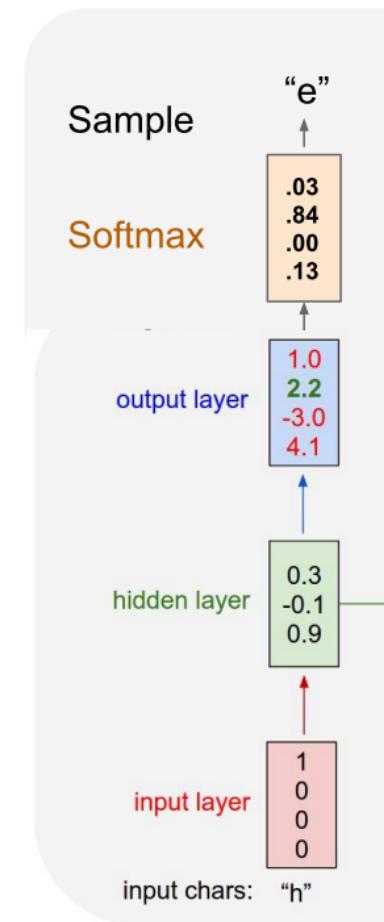
Example training
sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

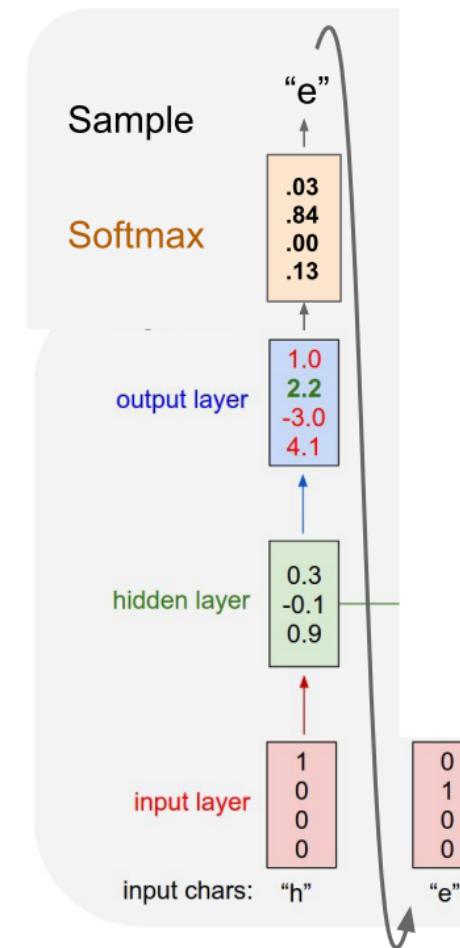
At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

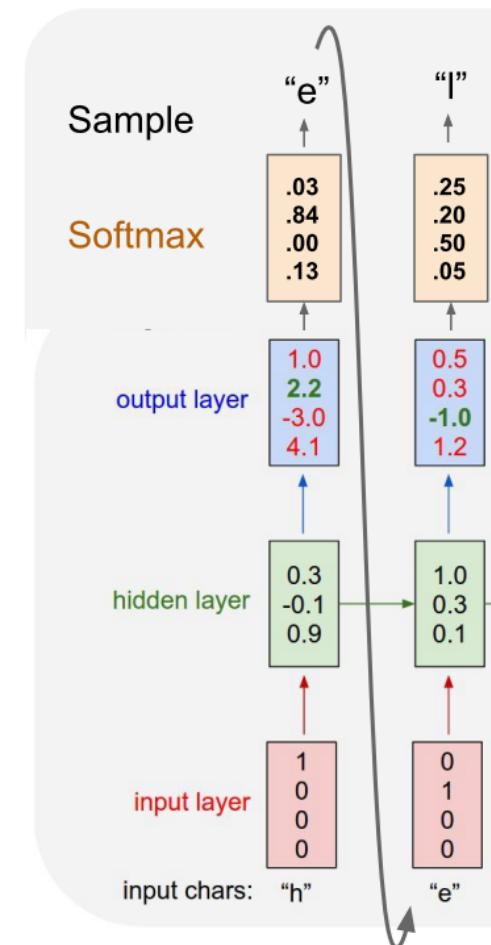
At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model

