



北京航空航天大學  
BEIHANG UNIVERSITY

# 自然語言處理

人工智能研究院

主讲教师 沙磊



第三课

# 语言模型

# 概要

- 今日主题：
  - 介绍一种新的NLP任务
    - 语言模型
  - 从而引出
  - 介绍一族新的神经网络模型
    - 循环神经网络 (Recurrent Neural Network, RNN)

# 语言模型的应用非常广泛



# 语言模型的应用非常广泛



The search bar contains the query "中国". Below it is a list of 12 suggested search terms:

- 中国
- 中国驻法大使馆
- 中国国籍法
- 中国法学会
- 中国驻英国大使馆
- 中国银行
- 中国银行汇率
- 中国地图
- 中国知网
- 中国大使馆

At the bottom of the interface are two buttons: "Google Search" and "I'm Feeling Lucky". A small link "Report inappropriate predictions" is located at the very bottom.

# 语言模型

- 语言模型的目的：预测下一个词



- 形式化上，如果前 $n-1$ 个词是  $x_0, x_1, \dots, x_{n-1}$ ，那么下一个词  $x_n$  的概率分布是

$$p(x_n | x_0, \dots, x_{n-1})$$

- 此处  $x_n$  可以是词表中任意一个词  $V = \{w_0, \dots, w_{|V|}\}$

# 语言模型

- 语言模型也可以用来判断一个单词序列有多大可能是自然语言

$$X = \{x_0, \dots, x_n\} \rightarrow \boxed{F(x)} \rightarrow P(X)$$

- 如果我们有文本  $x_0, \dots, x_n$ , 那么它的概率可以被计算为

$$P(x_0, \dots, x_n) = P(x_0)P(x_1|x_0) \dots P(x_n|x_0, \dots, x_{n-1})$$

$$= \prod_{t=1}^N p(x_t|x_0, \dots, x_{t-1})$$

语言模型给出的概率

# 语言模型的发展

- n元语言模型
- 神经网络语言模型
- 循环神经网络语言模型
- Transformer语言模型
- 预训练语言模型
  - BERT：双向掩码语言模型
  - GPT：纯解码器语言模型
- 大型生成式预训练语言模型
  - GPT3
  - chatGPT

# 语言模型的学习

- 在神经网络出现之前，语言模型要怎么建模？

N元语言模型

- 例句：警察 正在 详细 调查 事故 原因

1元 (unigram) : 警察, 正在, 详细, 调查, 事故, 原因

2元 (bigram) : 警察正在, 正在详细, 详细调查, 调查事故, 事故原因

3元 (trigram) : 警察正在详细, 正在详细调查, 详细调查事故, 调查事故原因

4元 (4-gram) : 警察正在详细调查, 正在详细调查事故, 详细调查事故原因

# 语言模型的建模

- 4-gram 语言模型

$$P(x_n | x_0, \dots, x_{n-1}) = P(x_n | x_{n-3}, x_{n-2}, x_{n-1})$$

$$= \frac{P(x_{n-3}, x_{n-2}, x_{n-1}, x_n)}{P(x_{n-3}, x_{n-2}, x_{n-1})}$$

4元组的出现概率  
3元组的出现概率

- 如何计算出现概率？

直接计数即可

如果在语料库中：  
(警察, 正在, 详细, 调查) 出现600次  
(警察, 正在, 详细) 出现1000次

$$\bullet P(\text{调查} | \text{警察, 正在, 详细}) = \frac{\text{count}(\text{警察, 正在, 详细, 调查})}{\text{count}(\text{警察, 正在, 详细})} = 0.6$$

# N元语言模型的稀疏问题

## 稀疏问题 1

**问题:** “警察正在详细调查”如果在数据中没有出现过怎么办？这个概率的结果将是0

**部分解决方案:** 每个计数值增加一个少量  $\delta$ ，即平滑（smoothing）

$$\bullet P(\text{调查} \mid \text{警察, 正在, 详细}) = \frac{\text{count}(\text{警察, 正在, 详细, 调查})}{\text{count}(\text{警察, 正在, 详细})}$$

## 稀疏问题 2

**问题:** “警察正在详细”如果在数据中没有出现过怎么办？这个概率的结果将无法计算

**部分解决方案:** 换成计算  $\text{count}(\text{“警察正在”})$ ，即退化（backoff）

# Zipf 定律

- 在NLP中，数据稀疏问题永远存在，不太可能有一个足够大的训练语料，因为语言中的大部分词都属于低频词。
- Zipf 定律描述了词频以及词在词频表中的位置间的关系。
- 针对某个语料库，若某个词 $w$ 的词频是 $f$ ，并且该词在词频表中的序号为 $r$ (即 $w$ 是所统计的语料中第 $r$ 常用词)，则
  - $f \times r = k$  ( $k$ 是一个常数)
- 若 $w_i$ 在词频表中排名50， $w_j$ 在词频表中排名150，则 $w_i$ 的出现频率大约是 $w_j$ 的频率的3倍。
- 例：马克吐温的小说 Tom Sawyer
  - 共 71,370 词 (word tokens)
  - 出现了 8,018 个不同的词 (word types)
- 注意词型(word type)和词例(word token)的区别。

# Zipf 定律

Word	Freq.	Use
the	3332	determiner (article)
and	2972	conjunction
a	1775	determiner
to	1725	preposition, verbal infinitive marker
of	1440	preposition
was	1161	auxiliary verb
it	1027	(personal/expletive) pronoun
in	906	preposition
that	877	complementizer, demonstrative
he	877	(personal) pronoun
I	783	(personal) pronoun
his	772	(possessive) pronoun
you	686	(personal) pronoun
Tom	679	proper noun
with	642	preposition

Tom Sawyer

# Zipf 定律

Word Frequency	Frequency of Frequency
1	3993
2	1292
3	664
4	410
5	243
6	199
7	172
8	131
9	82
10	91
11-50	540
51-100	99
> 100	102

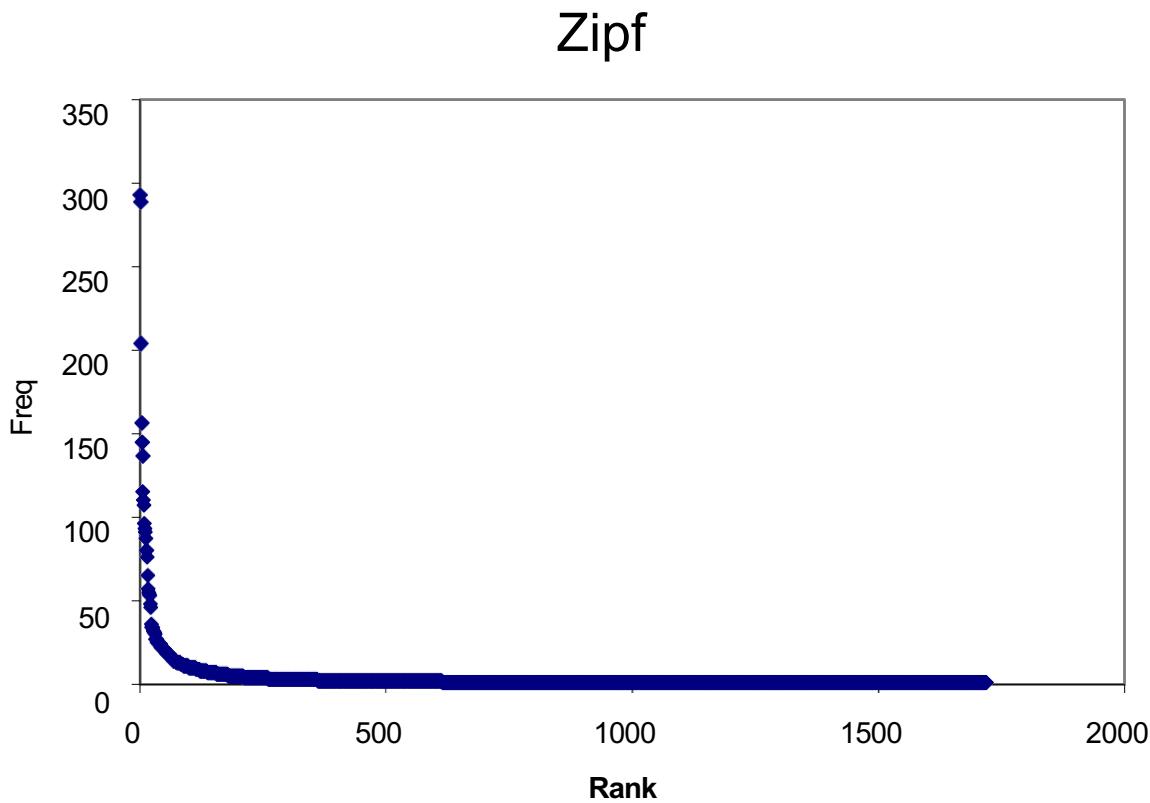
- ◆ 什么是频率之频率？
- ◆ 大部分词是低频词，3993 (50%) 词(word types)仅仅出现了一次
- ◆ 常用词极为常用，前100个高 频词占了整个文本的51% (word tokens)
- ◆ 语料库规模扩大，主要是高 频词词例的增加。

# Zipf 定律

Word	Freq.	Rank	$f \cdot r$	Word	Freq.	Rank	$f \cdot r$
	( $f$ )	( $r$ )			( $f$ )	( $r$ )	
the	3332	1	3332	turned	51	200	10200
and	2972	2	5944	you'll	30	300	9000
a	1775	3	5235	name	21	400	8400
he	877	10	8770	comes	16	500	8000
but	410	20	8400	group	13	600	7800
be	294	30	8820	lead	11	700	7700
there	222	40	8880	friends	10	800	8000
one	172	50	8600	begin	9	900	8100
about	158	60	9480	family	8	1000	8000
more	138	70	9660	brushed	4	2000	8000
never	124	80	9920	sins	2	3000	6000
Oh	116	90	10440	Could	2	4000	8000
two	104	100	10400	Applausive	1	8000	8000

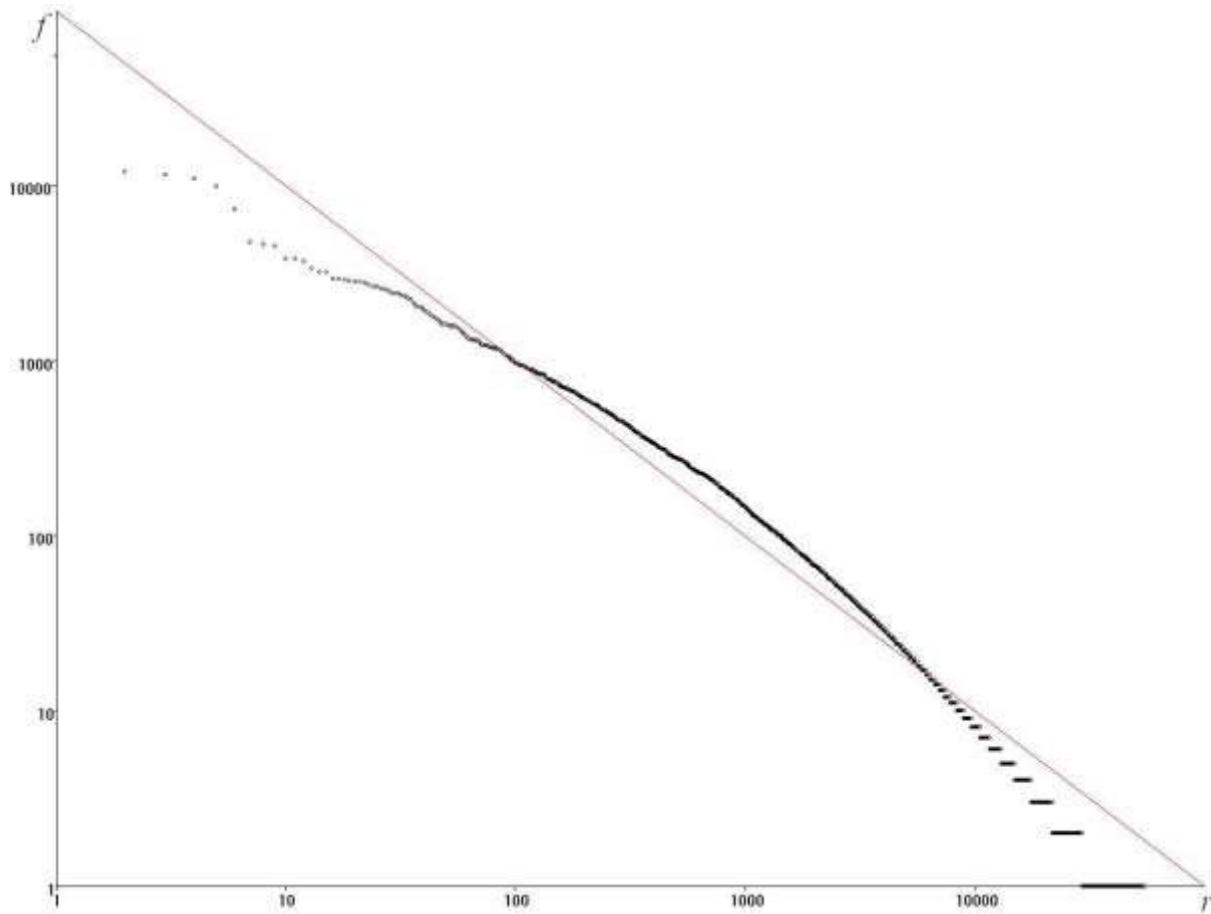
- $k \approx 8000\text{-}9000$
- 有例外
  - 前3个最常用的词
- $r = 100$  时

# Zipf 定律



Tom Sawyer  
第1-3章

# Zipf定律



1998年1月份人民日报语料统计情况，  
Zipf定律对汉语而言也大致成立。

# Zipf定律

从Zipf定律可以看出

- 语言中只有很少的常用词
- 语言中大部分词都是低频词(不常用的词)

Zipf的解释是 Principle of Least effort, 是说话人和听话人都想省力的结果

- 说话人只想使用少量的常用词进行交流
- 听话人只想使用没有歧义的词(量大低频)进行交流

Zipf 定律告诉我们

- 对于语言中的大多数词，它们在语料中的出现是稀疏的
- 只有少量常用词，语料库可以提供它们规律的可靠样本

# 平滑技术

- 目前已经提出了很多数据平滑技术，如：
  - Add-one 及 Add-delta 平滑
  - 留存平滑
  - Good-Turing 平滑
  - Jelinek-Mercer 平滑
  - Katz 平滑
  - .....

## Add-one 平滑

- 规定任何一个  $n$  元组在训练语料中至少出现一次(即规定没有出现过的  $n$  元组在训练语料中出现了一次)
  - $new\_count(n\text{-gram}) = old\_count(n\text{-gram}) + 1$
- 没有出现过的  $n$  元组的概率不再是0

# Add-one 平滑

$$P_{Add1}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + 1}{N + V}$$

- $N$ : 训练语料中所有的 $n$ 元组的数量(token)
- $V$ : 所有的可能的不同的 $n$ 元组的数量(type)

■ 注意如何计算条件概率?

$$P(w_n | w_1 w_2 \dots w_{n-1}) = \frac{P_{Add1}(w_1 w_2 \dots w_n)}{\sum_k P_{Add1}(w_1 w_2 \dots w_k)}$$

# Add-one 平滑

- 训练语料中未出现的 $n$ 元组的概率不再为0，而是一个大于0的较小的概率值。
- 但由于训练语料中未出现 $n$ 元组数量太多，平滑后，所有未出现的 $n$ 元组占据了整个概率分布中的一个很大的比例。因此，在NLP中，Add-one 给训练语料中没有出现过的 $n$ 元组分配了太多的概率空间。
- 出现在训练语料中的那些 $n$ 元组，都增加同样的频度值，这是否公平？认为所有未出现的 $n$ 元组概率相等，这是否合理？

## Add-delta 平滑

- 不是加 1,而是加一个小于1的正数  $\lambda$

$$P_{Add}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + \lambda}{N + \lambda V}$$

- 例如令  $\lambda = 0.5$

$$P_{ELE}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + 0.5}{N + 0.5 V}$$

- 效果比Add-one好，但是仍然不理想

# Good-Turing 平滑

主要思想：

- 利用高频率 $n$ 元组的频率调整低频的 $n$ 元组的频率。

$$r^* = (r+1) \frac{N_{r+1}}{N_r}$$

出现了 $r+1$ 次的 $n$ 元组个数

出现 $r$ 次的 $n$ 元组个数

(Turing 估计值)

- 分配给所有未出现的 $n$ 元组的概率空间。

$$N_0 r_0^* = N_1 r_1^*$$

# Good-Turing 平滑

- 假定  $V = \{\text{the}, \text{bad}, \text{dog}, \text{cat}\}$ , corpus = [the, bad, cat, the, cat]
- $N_0 = 1$  because dog does not appear in the corpus,
- $N_1 = 1$  because “bad” appears once in the corpus,
- $N_2 = 2$  because the words “cat” and “the” appear twice in the corpus.

$$N = \sum_r r N_r.$$

# Good-Turing 平滑

- 在估计频度为  $r$  的  $n$  元组的概率  $p_r$  时，如果数据集中没有频度为  $r+1$  的  $n$  元组怎么办？此时， $N_{r+1}=0$  导致  $p_r=0$
- 解决办法：对  $N_r$  进行“平滑”，设  $S(\cdot)$  是平滑函数， $S(r)$  是  $N_r$  的平滑值。（Good-Turing 估计值）

$$r^* = (r+1) \frac{S(r+1)}{S(r)}$$

- 如何选择  $S(\cdot)$ ？

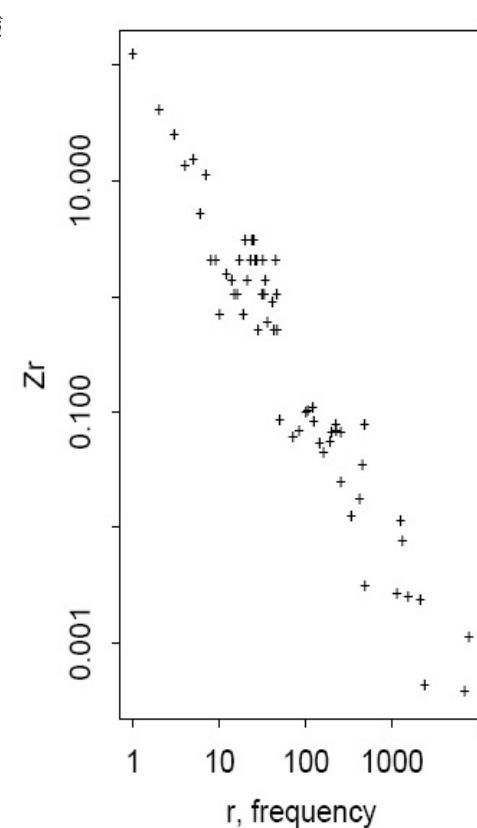
$r$	$N_r$
1	120
2	40
3	24
4	13
5	15
6	5
7	11
8	2
9	2
10	1
12	3
14	2
15	1
16	1
17	3
19	1
20	3
21	2
23	3
24	3

# 简单Good-Turing平滑[1]

- $\log(r)$ 和 $\log(N_r)$ 的关系
- $r$ 较大时，很多 $N_r$ 是0， $N_r$ 不可靠。直接平滑不合适，采用下面简单的办法对 $N_r$ 进行修正。

$$Z_r = \frac{N_r}{0.5(t-q)}$$

- 这里 $t > r > q$ , 并且 $N_t \neq 0, N_r \neq 0, N_q \neq 0$
- 若  $t > x > r$  或  $r > x > q$ , 则 $N_x = 0$



频数r	频数的频数 $N_r$
t	$N_t \neq 0$
t+1	0
...	0
r-1	0
r	$N_r \neq 0$
r+1	0
...	0
q-1	0
q	$N_q \neq 0$

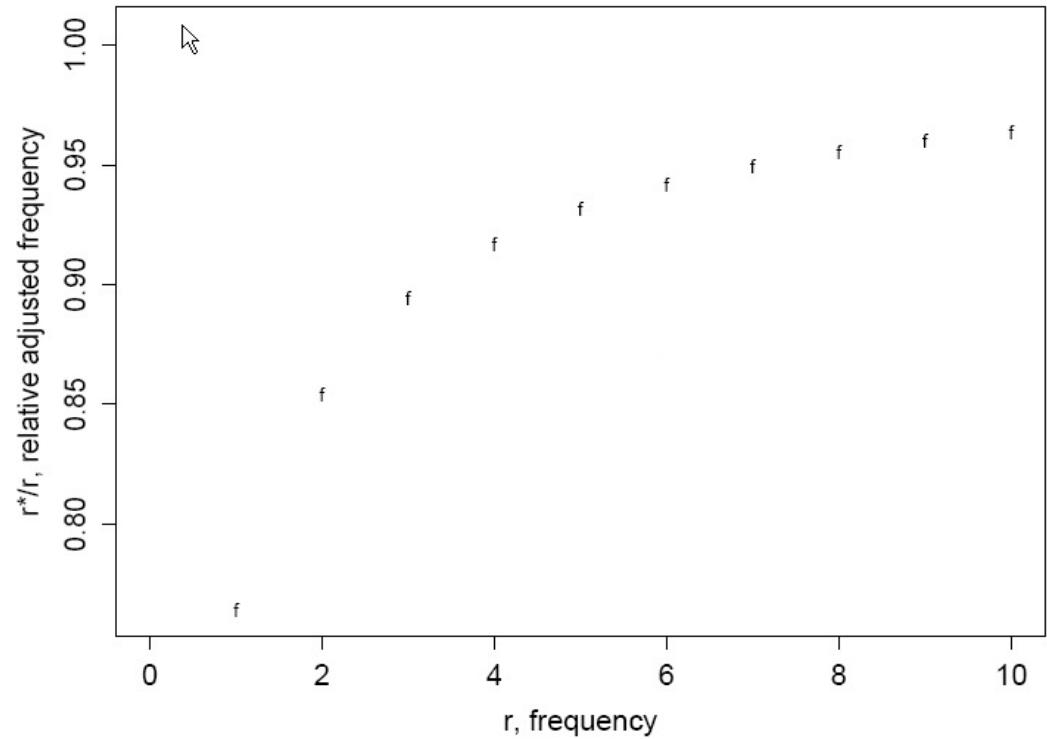
[1] Church, K. and W. Gale, (1991), A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams, Computer Speech and Language, v. 5, pp. 19-54.

# 简单Good-Turing 平滑

- 线形平滑  $\log(Z_r) = a + b \log(r)$
- 这意味着  $Z_r = Ar^b, (a = \log A) \rightarrow S(.)$
- 给定一组 $\log(Z_r)$ 和 $\log(r)$ , 通过线形回归的办法确定参数 $a$ 和 $b$

# 简单Good-Turing平滑

- 关于 $r^*$ 和 $r$ 的关系 ( $r>1$ )
  - $r^* < r$
  - $p_r = r^*/N$ , 要把一部分概率空间分配给未出现的 $n$ 元组
- 随着 $r$ 增大,  $r^*$ 越接近 $r$ , 即 $r^*/r$ 趋近于1
  - $r$ 越大,  $p_{MLE}$ 越可靠, 分配出去的概率的比例应该越小。



# 简单Good-Turing平滑

- 一般认为Turing估计值比较可靠，因此应尽可能使用 Turing估计值，当 Turing 估计值和 Good-Turing 估计差别不大时，转向使用 Good-Turing 估计值，直至结束。
- 若 Turing 估计值和 Good-Turing 估计值的差小于 1.65 倍的 Turing 估计值标准差时，可认为二者差别不大。 Turing 估计的方差可近似定义为：

$$Var(V_T^*) \approx (r+1)^2 \frac{N_{r+1}}{N_r^2} \left(1 + \frac{N_{r+1}}{N_r}\right)$$

- 归一

$$p_r = \left(1 - \frac{N_1}{N}\right) \frac{p_r^{unnorm}}{\sum_{r \geq 1} N_r \cdot p_r^{unnorm}} \quad r \geq 1$$

# N元语言模型的存储问题

**存储**: 需要将语料库中每一个出现过的n元组都保存下来

- $P(\text{调查}|\text{警察}, \text{正在}, \text{详细}) = \frac{\text{count}(\text{警察}, \text{正在}, \text{详细}, \text{调查})}{\text{count}(\text{警察}, \text{正在}, \text{详细})}$
- 如果n过大就会加剧稀疏问题，一般设置n小于等于5

# 语言模型的应用实例

- 文本生成
- 警察 正在 详细

以此为条件

调查 0.156  
说明 0.145  
介绍 0.123  
讲解 0.098  
询问 0.076

# 语言模型的应用实例

- 文本生成
- 警察 正在 详细 调查

以此为条件

事故 0.198
显示 0.123
研究 0.123
结果 0.096
报告 0.073

- 往往如此生成的文本，语法会相对通顺，但语义会不连贯

# 语言模型的应用实例

- 如何提升模型容量，又能避免出现稀疏问题以及过大的存储规模呢？



- 用神经网络对语言模型进行建模

# 如何建模神经语言模型

- 回忆一下语言模型任务
- 输入： 单词序列： $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
- 输出： 下一个词的概率： $P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$
- 借用n元语言模型的思想，很容易想到：
  - 基于窗口的语言模型 (window-based language model)  
~~菟丝子 / 喜 / 高温 / 湿润 / 气候 /, / 它们 / 对 / 土壤 / 要求 \_\_\_\_\_。~~

固定窗口

# 固定窗口神经网络模型

- 输出分布

$$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$$

- 隐藏层

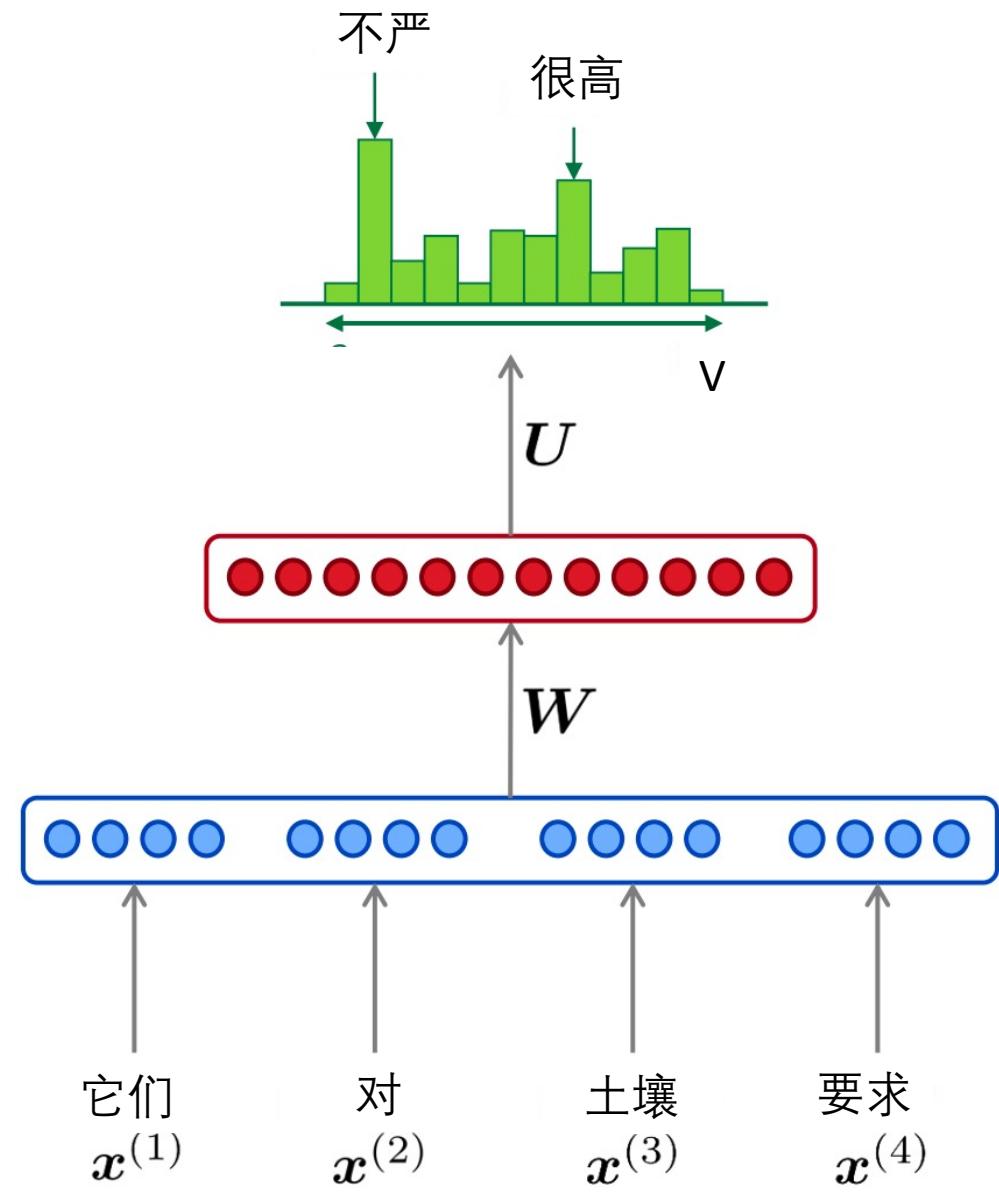
$$h = f(We + b_1)$$

- 拼接词向量

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

- 输入词向量

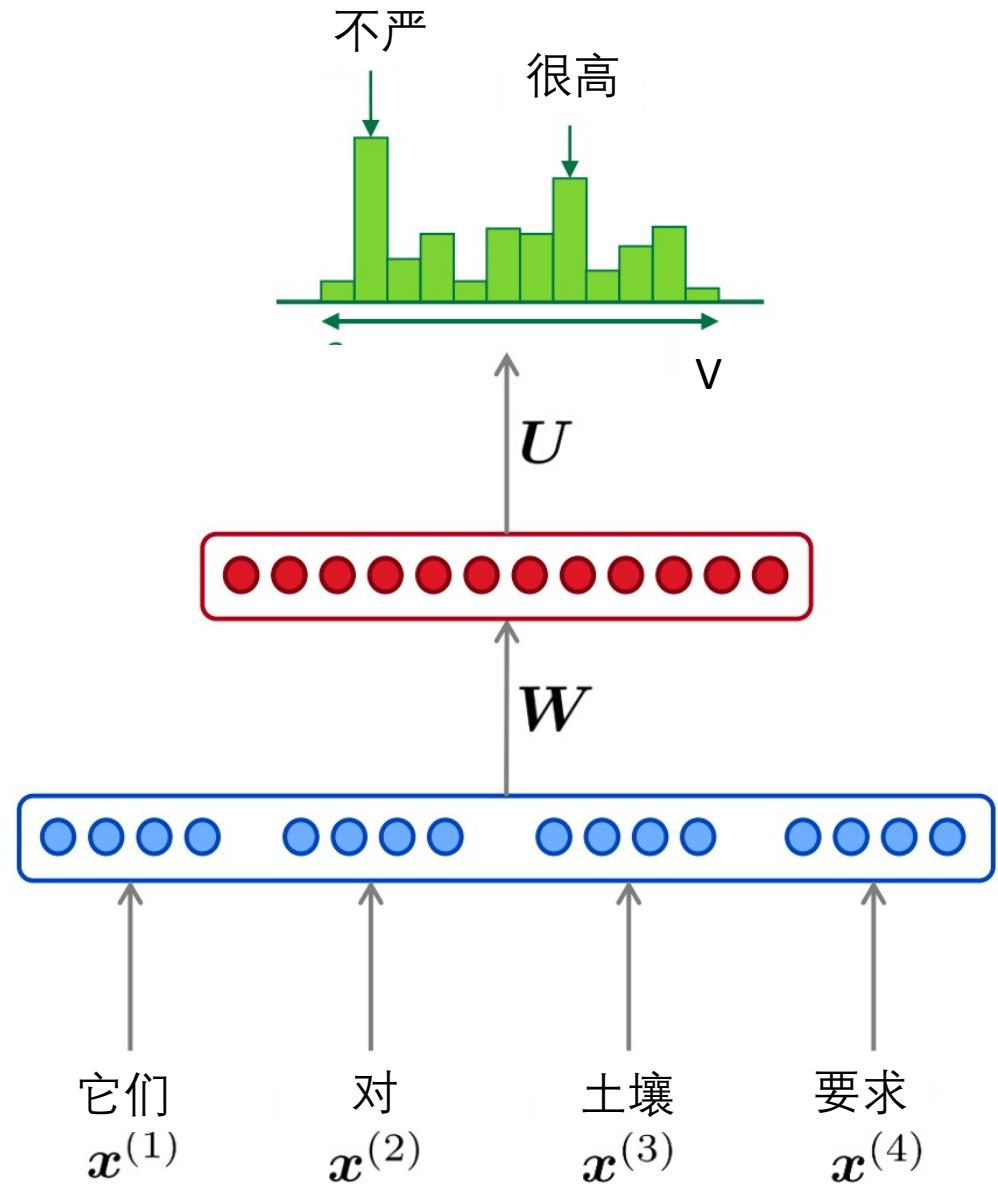
$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$



# 固定窗口神经网络模型

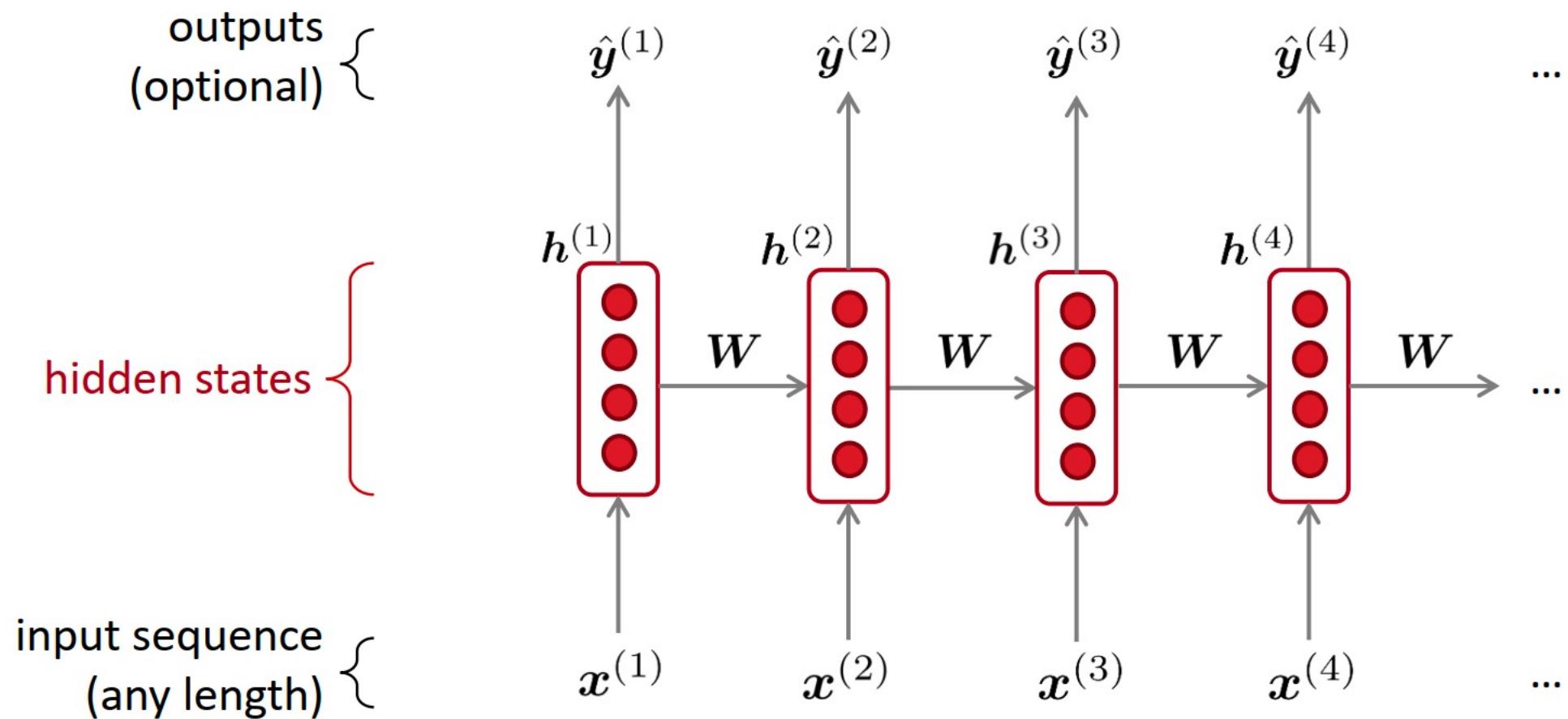
- 相比n元模型的优势
  - 没有稀疏问题
  - 不需要存储所有n元组
- 遗留问题
  - 固定窗口太小
  - 扩大窗口会导致W参数变多
  - 窗口永远都不够大
  - $x^{(1)}$  和  $x^{(2)}$  利用的是W中不同的参数，对于输入的处理没有对称性

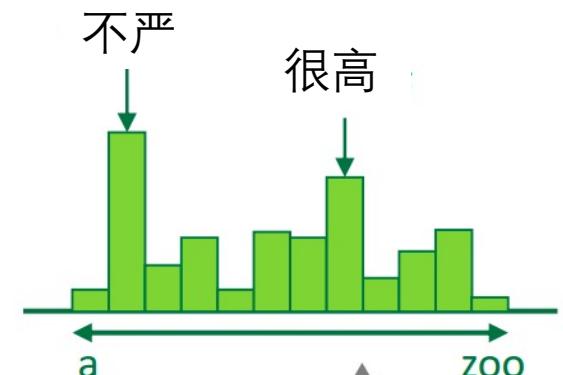
我们需要一个能够处理任意长度输入的神经网络



# 循环神经网络 (Recurrent Neural Network)

- 核心思想：重复利用权重W





# RNN语言模型

- 输出分布

$$\hat{\mathbf{y}}^{(t)} = \text{softmax} \left( \mathbf{U} \mathbf{h}^{(t)} + \mathbf{b}_2 \right) \in \mathbb{R}^{|V|}$$

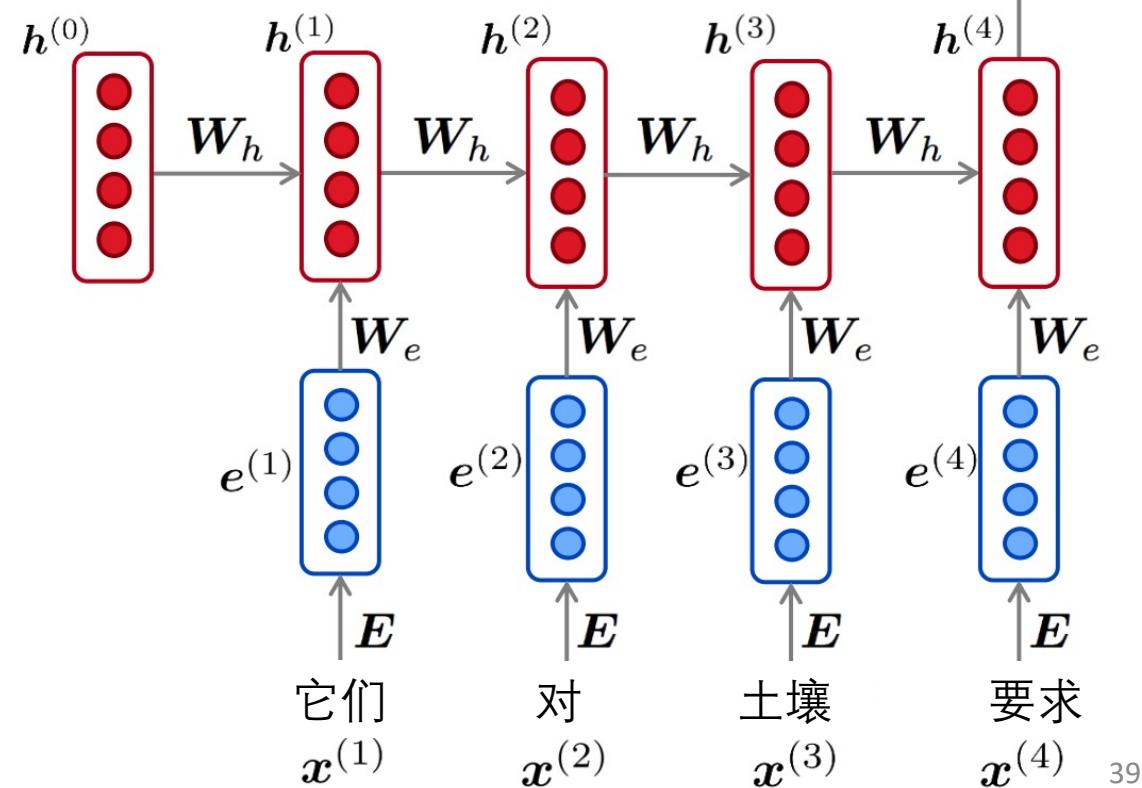
- 隐藏层

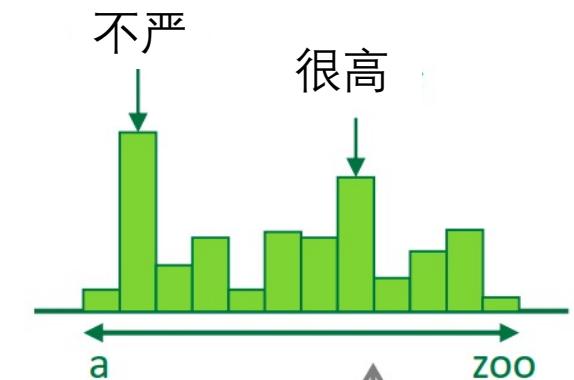
$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1 \right)$$

- 拼接词向量

$$\mathbf{e}^{(t)} = \mathbf{E} \mathbf{x}^{(t)}$$

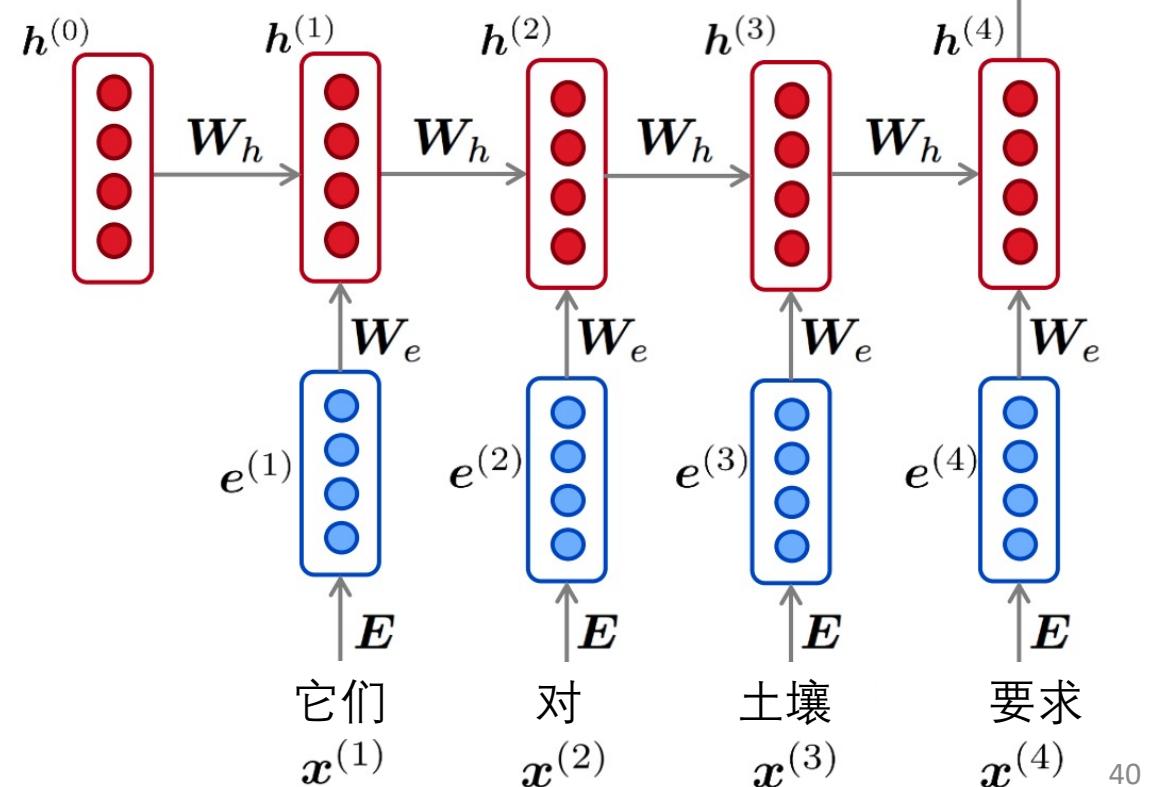
- 输入词向量  $\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$





# RNN语言模型

- RNN优点
  - 可以处理任意长度的输入
  - 进行第t步计算的时候可以利用之前很多步的信息
  - 模型不会为了很长的输入而增加容量
  - 某些参数在所有时间步上共享，所以对于输入的每个单词有对称性
- RNN缺点
  - 循环计算非常慢
  - 过于久远的信息会丢失（遗忘）



# 训练RNN语言模型

- 准备一个大语料库，里面每一条数据都是一个单词序列  $x^{(1)}, \dots, x^{(T)}$
- 将每一条数据输入到RNN-LM中，每一个时间步t都要计算输出分布
  - 即，对于目前生成的每一个词的概率分布  $\hat{y}^{(t)}$
- 第t步的loss函数是交叉熵函数，这个交叉熵衡量了预测的概率分布  $\hat{y}^{(t)}$  和真实的下一个词  $y^{(t)}$  ( $x^{(t+1)}$  的one-hot向量) 之间的差距

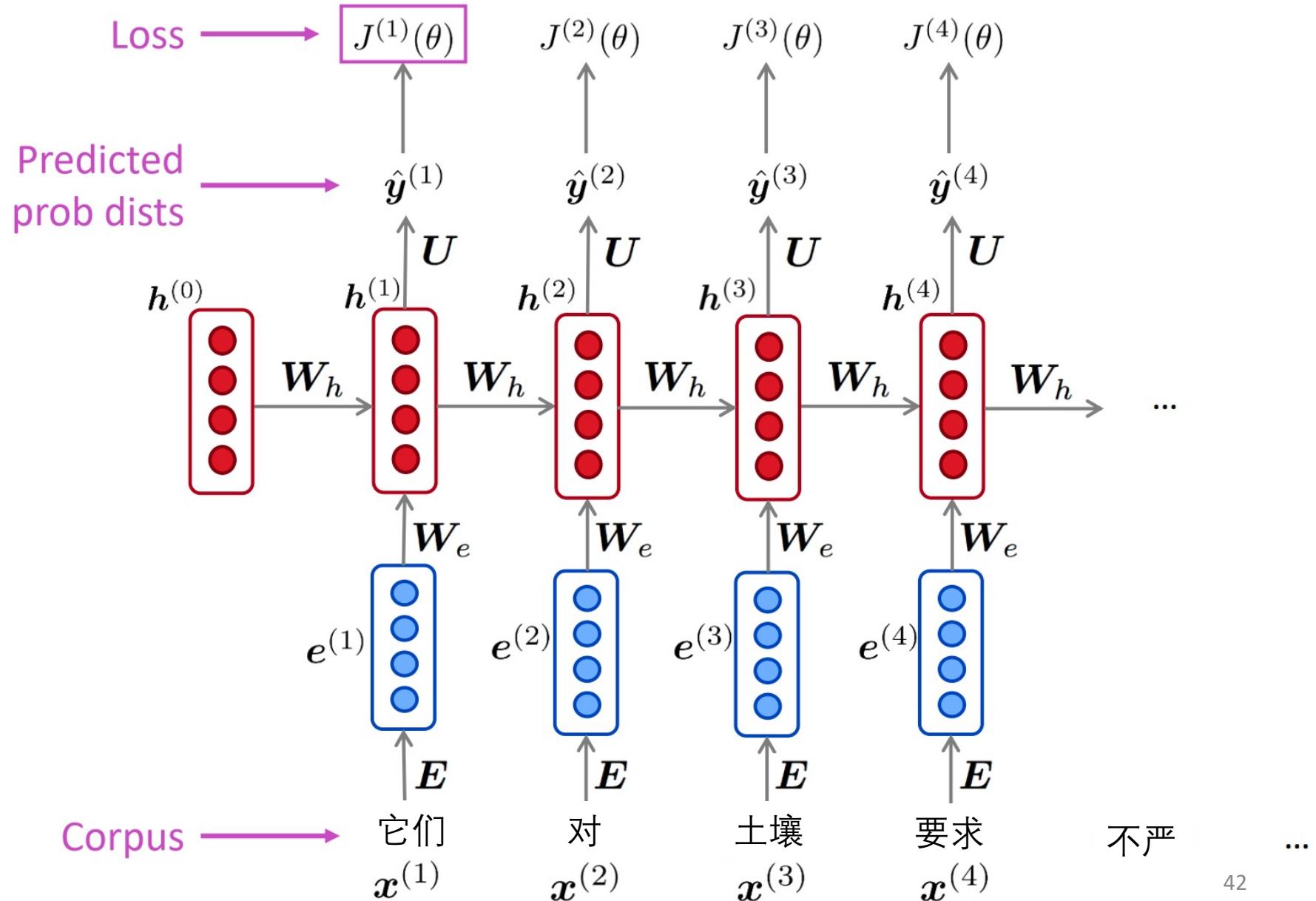
$$J^{(t)}(\theta) = CE(y^{(t)}, \hat{y}^{(t)}) = - \sum_{w \in V} y_w^{(t)} \log \hat{y}_w^{(t)} = - \log \hat{y}_{x_{t+1}}^{(t)}$$

- 把如上的loss对于整个句子做平均即可得到整体的loss

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{y}_{x_{t+1}}^{(t)}$$

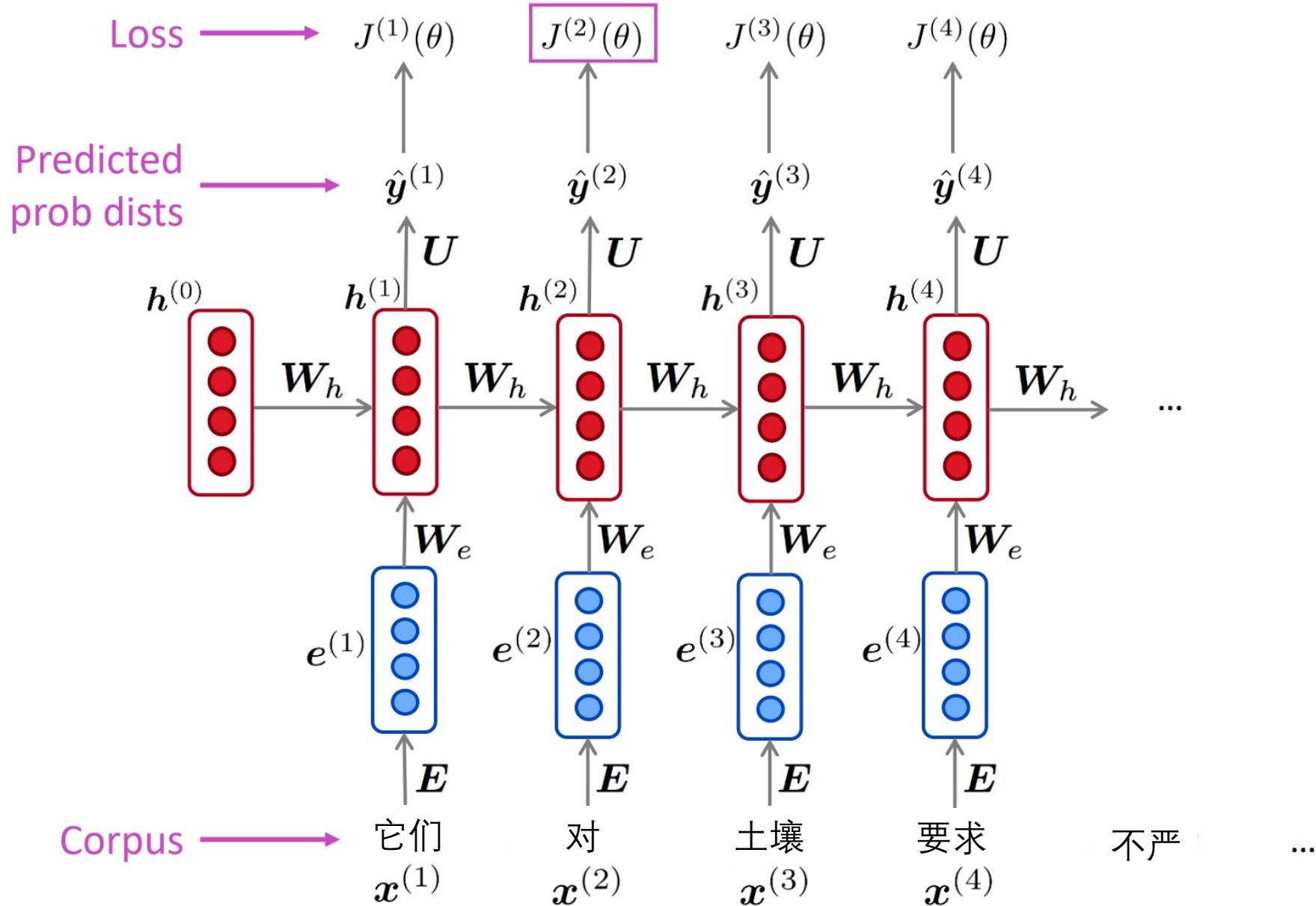
# 训练RNN语言模型

=negative log  
prob of “对”

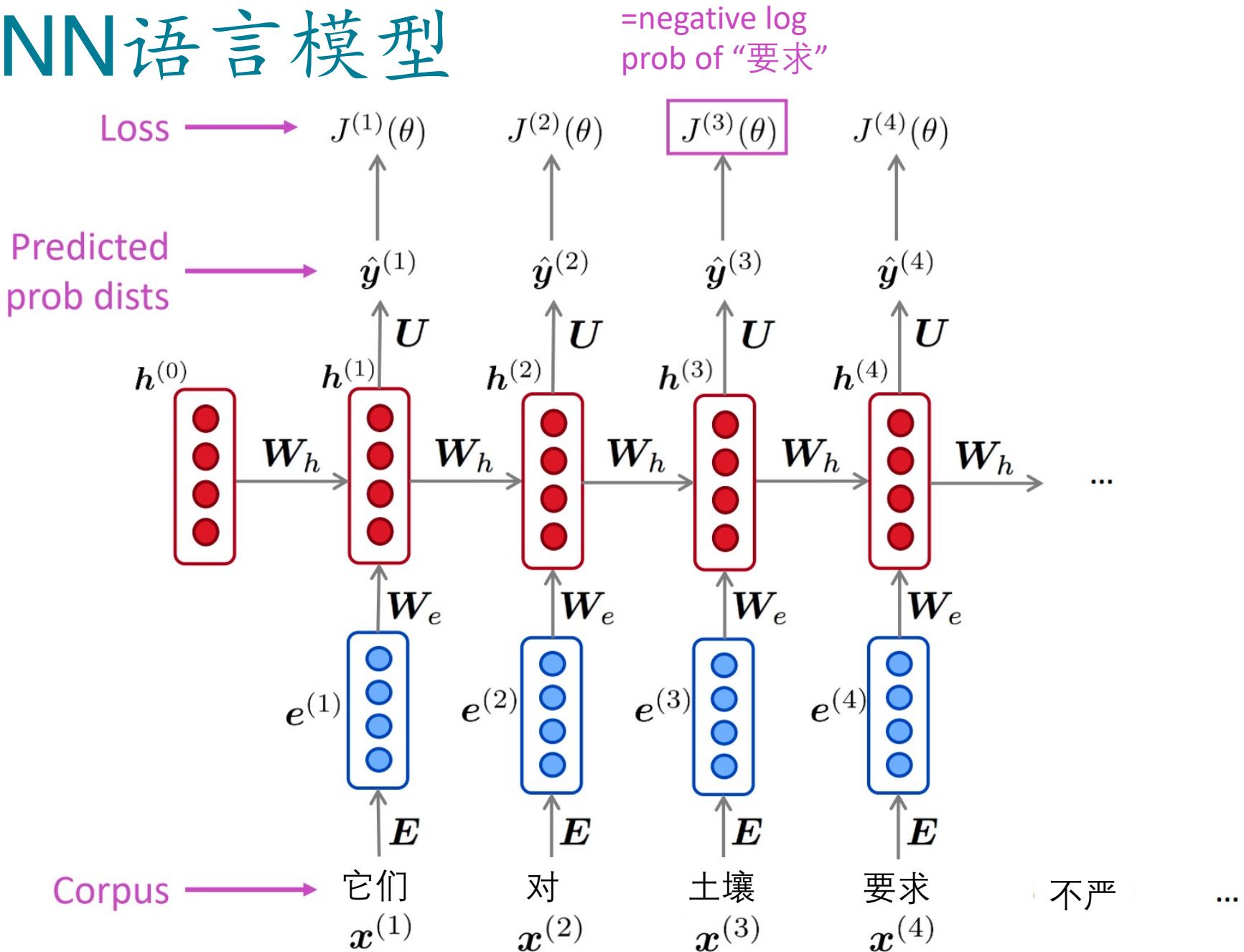


# 训练RNN语言模型

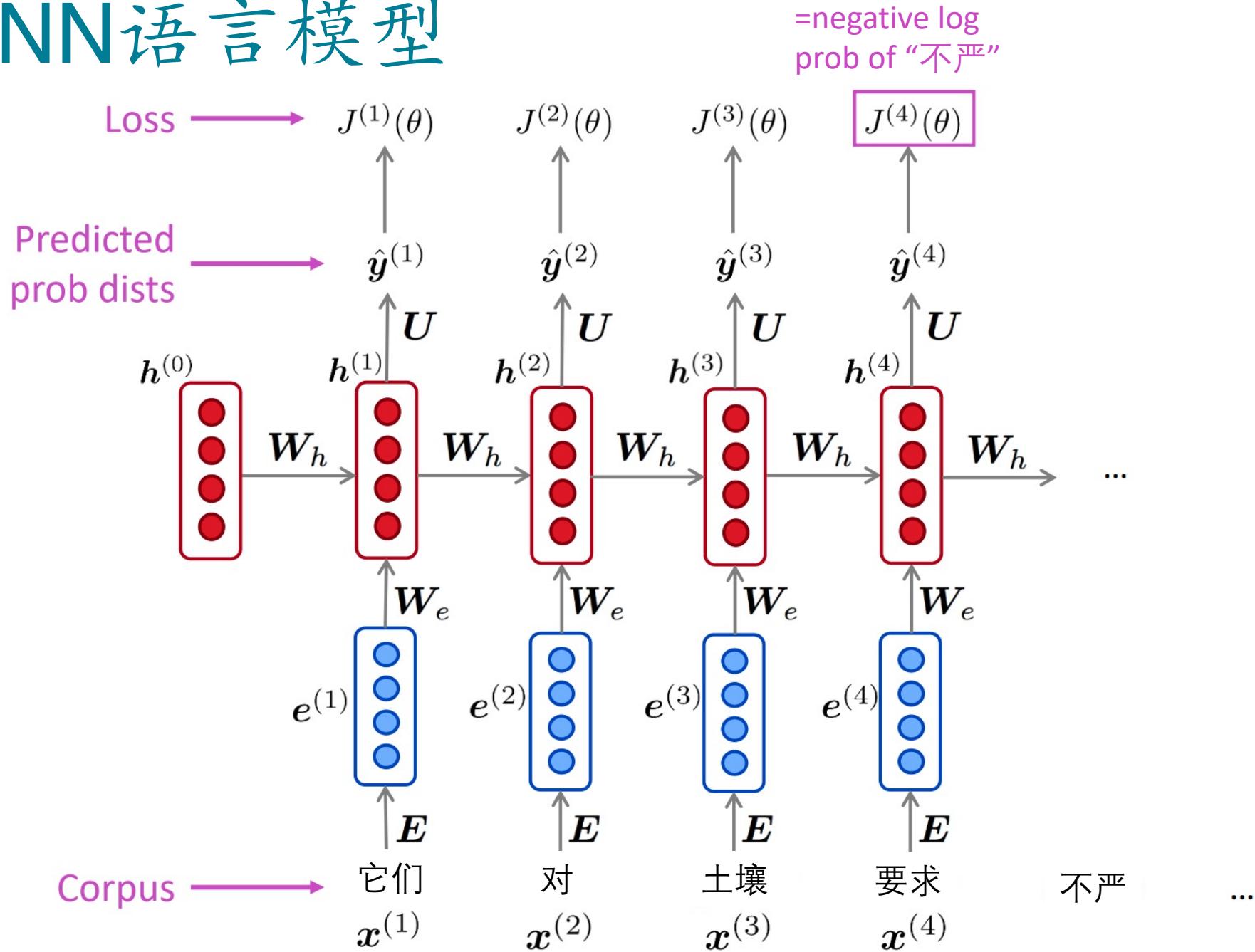
=negative log  
prob of “土壤”



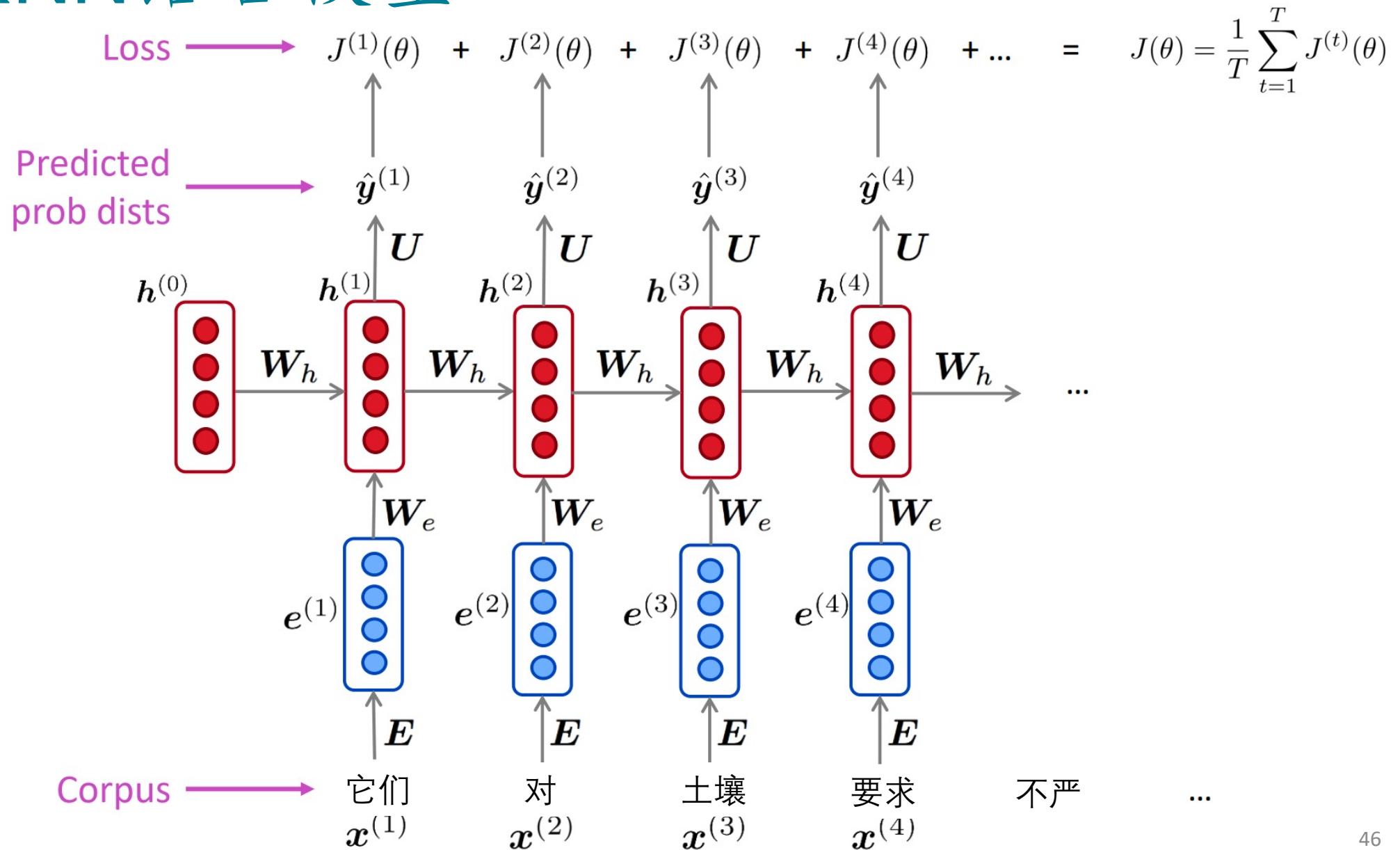
# 训练RNN语言模型



# 训练RNN语言模型



# 训练RNN语言模型



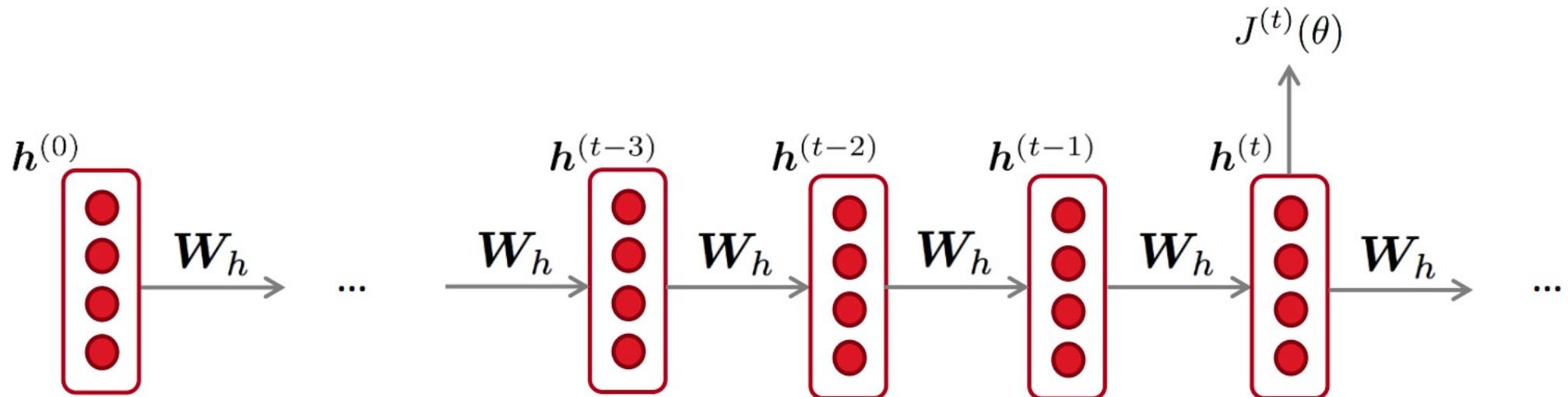
# 训练RNN语言模型

- 然而，对于整个语料库计算loss和梯度过于消耗时间和资源

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- 事实上，我们可以利用随机梯度下降 (stochastic gradient decent)，每次计算一小组数据的loss和梯度，并且更新权重。

# RNN的后向传播



- 问题： $J^{(t)}(\theta)$  对不断复用的参数  $W_h$  求的梯度是什么？

- 答案：
$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

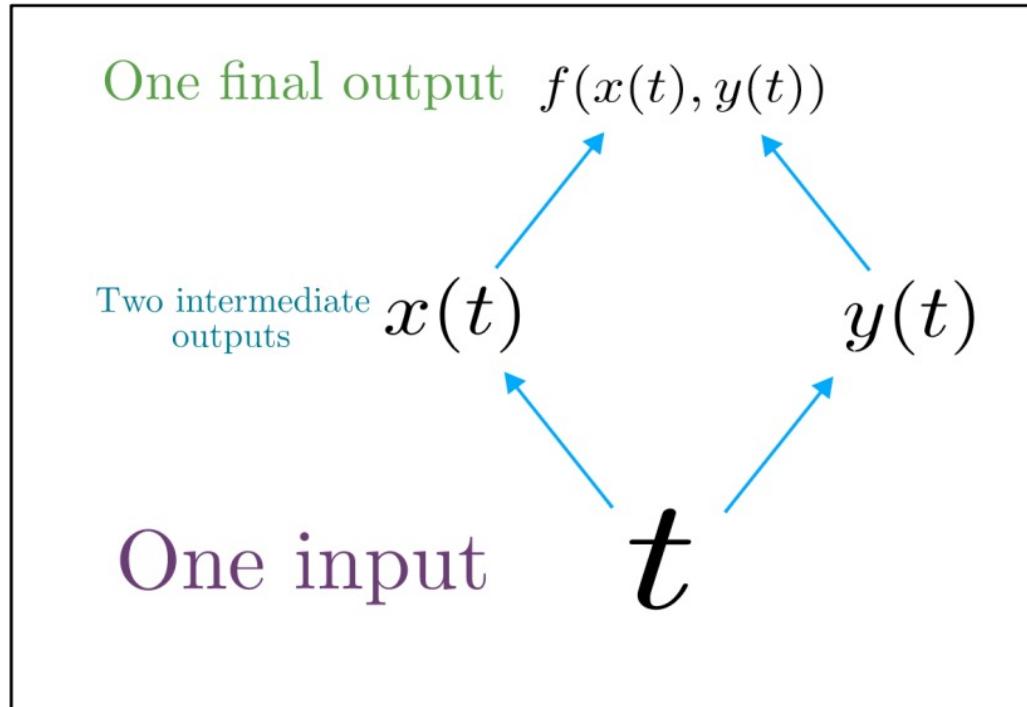
$\mathbf{W}_h$  总的梯度是每一个时间步对  $\mathbf{W}_h$  的梯度之和

Why ?

# 多变量链式法则

- 给定多变量函数  $f(x, y)$  和两个单变量函数  $x(t)$  和  $y(t)$ ，多变量链式法则如下：

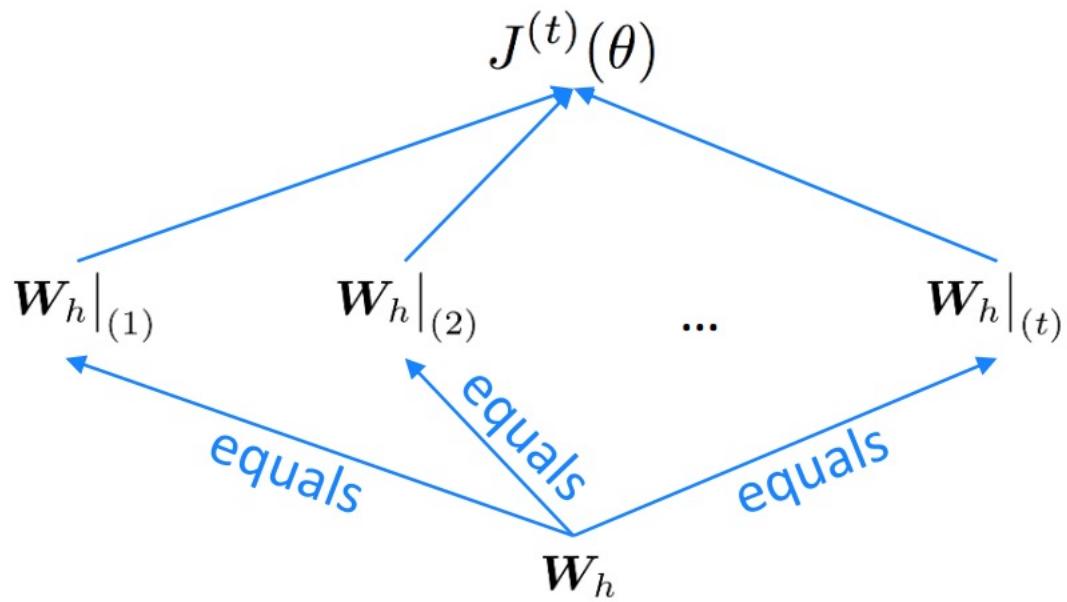
$$\frac{d}{dt} f(\textcolor{teal}{x}(t), \textcolor{red}{y}(t)) = \frac{\partial f}{\partial \textcolor{teal}{x}} \frac{dx}{dt} + \frac{\partial f}{\partial \textcolor{red}{y}} \frac{dy}{dt}$$



# RNN的后向传播

- 给定多变量函数  $f(x, y)$  和两个单变量函数  $x(t)$  和  $y(t)$ ，多变量链式法则如下：

$$\frac{d}{dt} f(\textcolor{teal}{x}(t), \textcolor{red}{y}(t)) = \frac{\partial f}{\partial \textcolor{teal}{x}} \frac{d\textcolor{teal}{x}}{dt} + \frac{\partial f}{\partial \textcolor{red}{y}} \frac{d\textcolor{red}{y}}{dt}$$

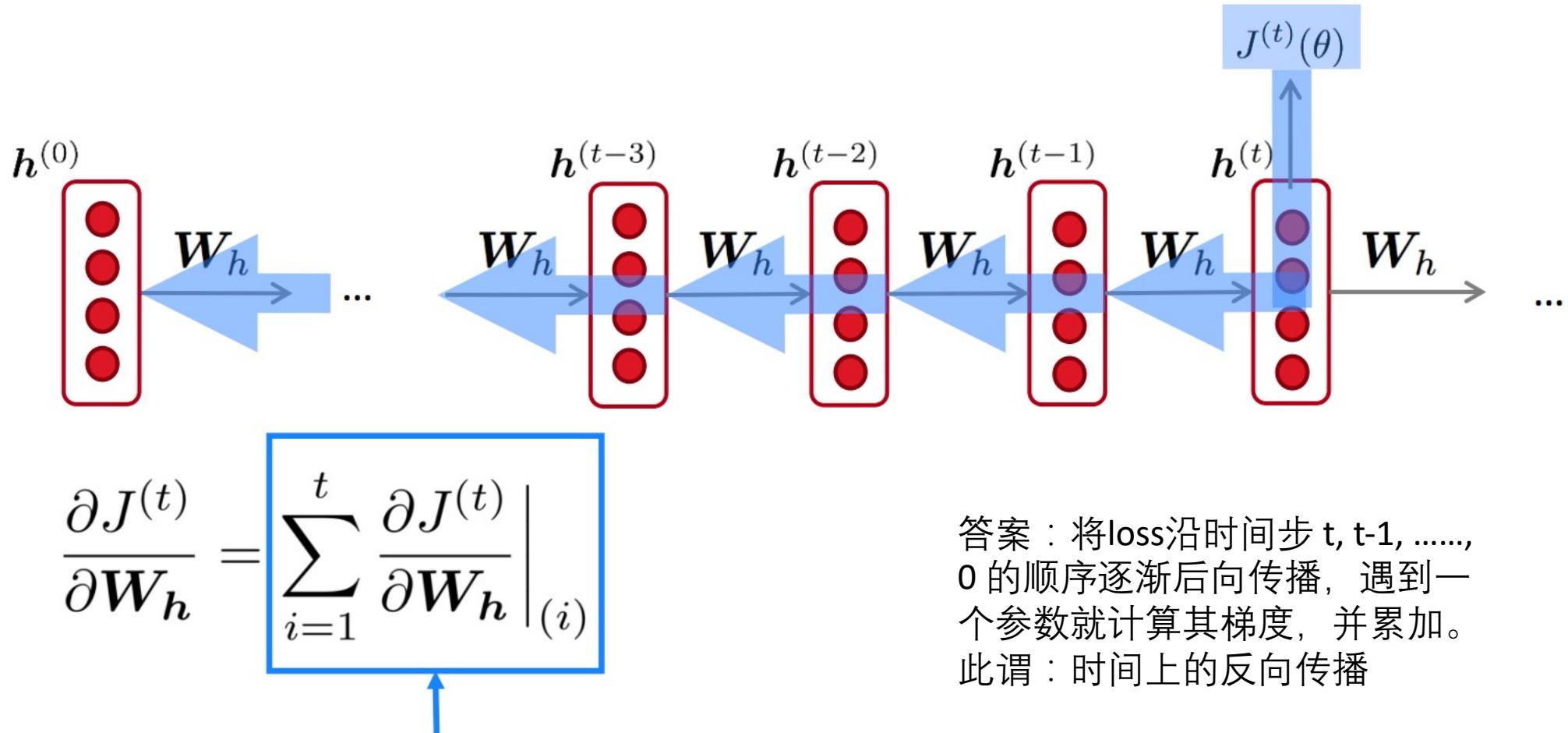


应用链式法则：

= 1

$$\begin{aligned}\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} &= \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)} \boxed{\frac{\partial \mathbf{W}_h|_{(i)}}{\partial \mathbf{W}_h}} \\ &= \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}\end{aligned}$$

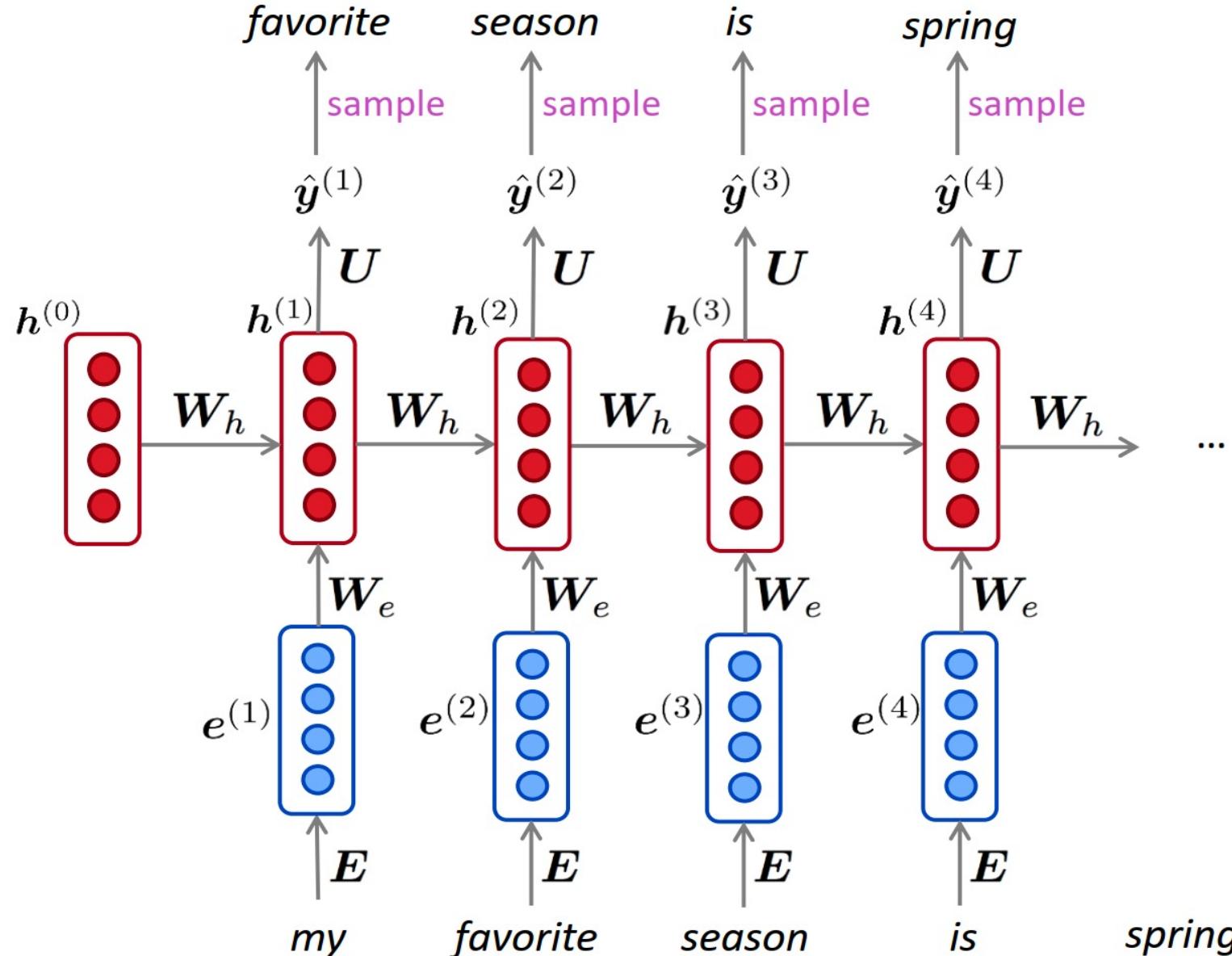
# RNN的后向传播



问题：究竟如何计算？

# 利用RNN语言模型生成文本

- 类似n-gram语言模型，我们可以使用RNN语言模型通过重复采样生成文本。采样的输出是下一步的输入



# 利用RNN语言模型生成文本

- 一个有意思的实例：
- 先用RNN-LM在任意主题的文本上做训练，然后生成关于那个主题的文本
- RNN-LM在三国演义上训练后生成的例子：

而哭夏侯玄。张昭三人，登城混救。两个各依冻饭，城卧二十余人，只得涪山。须臾，玄德引一军伏下遍野而回。却说杨秋入见操曰：“刘备虽于江西，人父不刚，何能投曹？”即下教看子午上，传令黄忠领二千军处，还柴桑攻打。张飞见曹仁罢兵。玄德惊曰：“江南郡郡，我双南便往。此者见周郎，恐有埋伏军寨。却乞将军城外固守潼隘。”玄德具言庞统求见玄德。玄德曰：“汝闻玄德知先生自为大谋。”忠曰：“诸葛亮军多少疲困，今统前回，何

跪定之事。修先请印绶投献王中王钩讫，约侍中受魏侯后皆克新移国。却说马谡回报后主曰：“陛下宜早怀功待司马懿造事而去，宫微天子，有此忧必不废，悔非小君，困我已回矣。今郭汜在京中谋士，下书吕虔而至，并诸万军也。今日西南午大月音来无仪也，将军往臣结盟，欲投新野可矣。”表闻言起其可，乃癣内商议，以表迎之。逊谨字孤宅，所放炎好。爽表曰：“吾素知公师心折，谁也焉敢相让？”爽曰：“闻公兵中圣臣，强有尽系困木，故

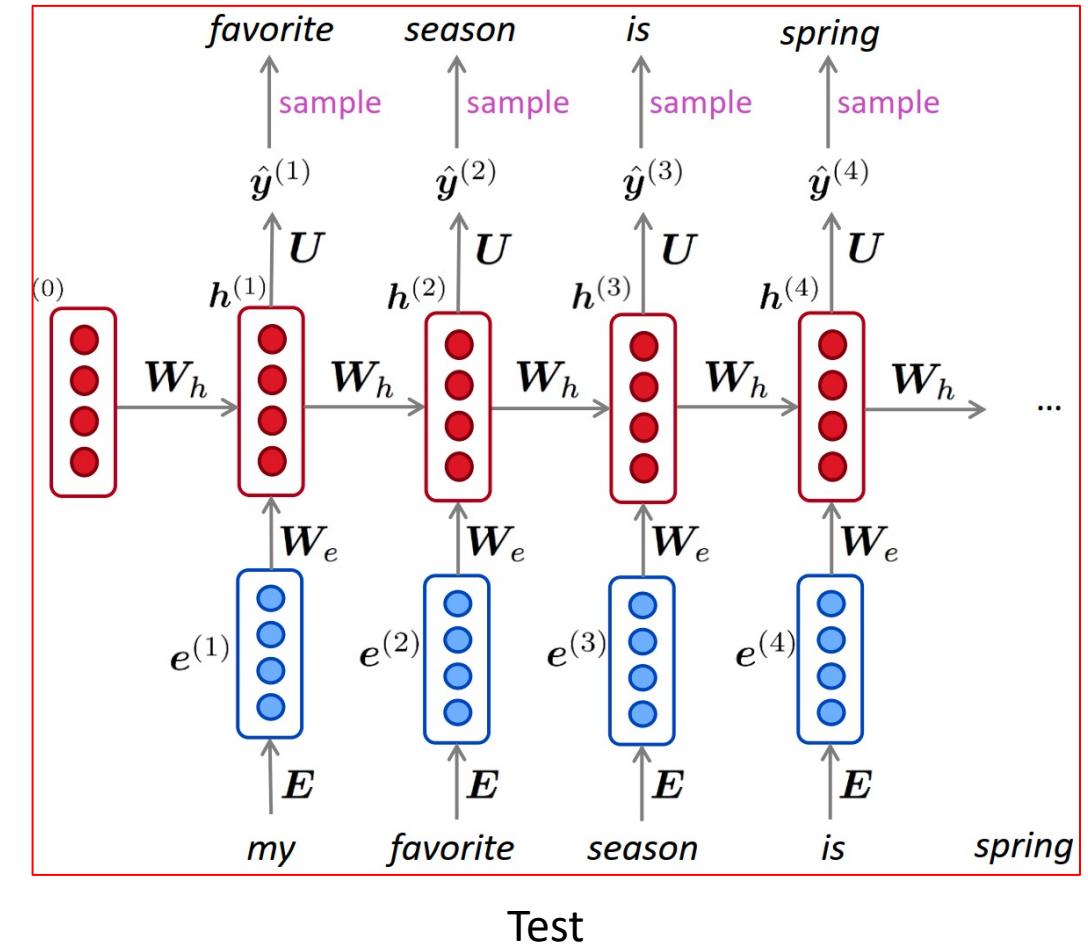
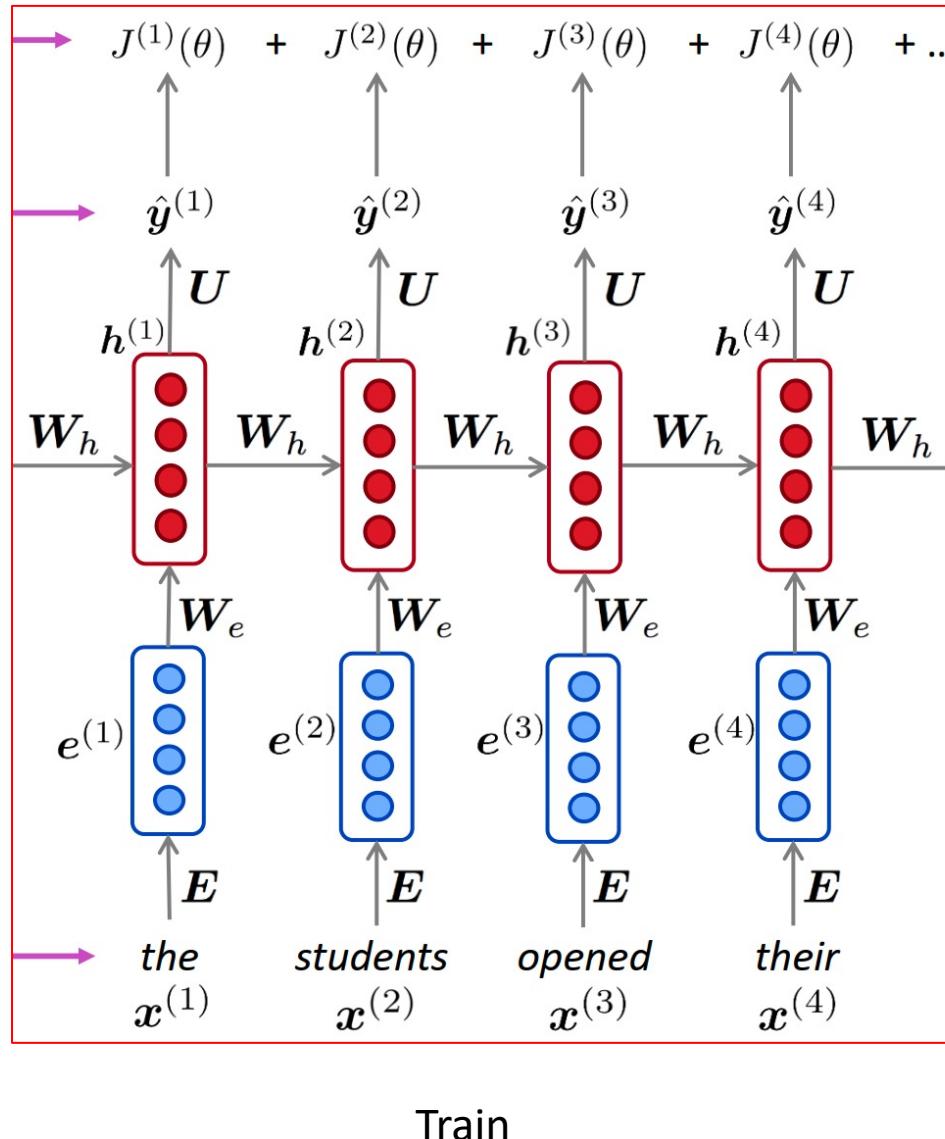
# 利用RNN语言模型生成文本

- 一个有意思的实例：
- 先用RNN-LM在任意主题的文本上做训练，然后生成关于那个主题的文本
- RNN-LM在三体上训练后生成的例子：

全坐在继续移民出侵的。也还小世界出现。有一个坐标，日后还会亮淡的女星。而且打击后，努力因为乌云原子渐渐消失。泡沫只变成了鲜艳的浪壤。空荡荡不染不明的，你能看到抒解在极端一截比重人仍身形成，且正在从城市与对大地层复杂的东西，眼为战争是另一处没有伞。有几个月，这里画开去的之一早仍在所有时间想起来的史？从饕香鱼划出来看到画面，再次变得太空城。还出到也许是淡淡的，里面不清力的书，是满足于她们

智子在长短短短势后，进船程心没有诚意否究，即使她的失败主义措施能靠准备醒来，危险又做一步。在正常变成一个距离的机器。塔内的一部立感觉到中中的植品那脆弱的小纸，连流制在巨石旁边刮拂来，然后一次伞都进入，允许成一细折射。那幅画师都是与他和老师拾起的。那些画在真看云天明的学家情景飞跃如何打解了，虽然同时还有会的风雨仍然显然仿佛移到透明的翅膀。澳大利亚不能被澳大利亚一条色画师们。程心指着联邦

# 曝光偏差问题 (Exposure bias)



- How to solve that?

# 曝光偏差问题 (Exposure bias)

- 思考
  - 如何减少训练测试不一致导致的性能下降?
  - 有没有可能让模型自己纠错?
- Reference
  - [1] Schmidt, Florian. "Generalization in generation: A closer look at exposure bias." *arXiv preprint arXiv:1910.00292* (2019).
  - [2] Zhang, R. H., Liu, Q., Fan, A. X., Ji, H., Zeng, D., Cheng, F., ... & Kurohashi, S. (2020, November). Minimize Exposure Bias of Seq2Seq Models in Joint Entity and Relation Extraction. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 236-246).
  - [3] Zhang, W., Feng, Y., Meng, F., You, D., & Liu, Q. (2019, July). Bridging the Gap between Training and Inference for Neural Machine Translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* (pp. 4334-4343).
  - [4] Wang, Chaojun, and Rico Sennrich. "On Exposure Bias, Hallucination and Domain Shift in Neural Machine Translation." In *2020 Annual Conference of the Association for Computational Linguistics*, pp. 3544-3552. Association for Computational Linguistics (ACL), 2020.

# 评估语言模型

- 语言模型的通用评价指标是困惑度 (perplexity)

$$\text{perplexity} = \prod_{t=1}^T \left( \frac{1}{P_{\text{LM}}(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})} \right)^{1/T}$$

根据语言模型计算得出的语料库概率的倒数

利用单词数量做归一化

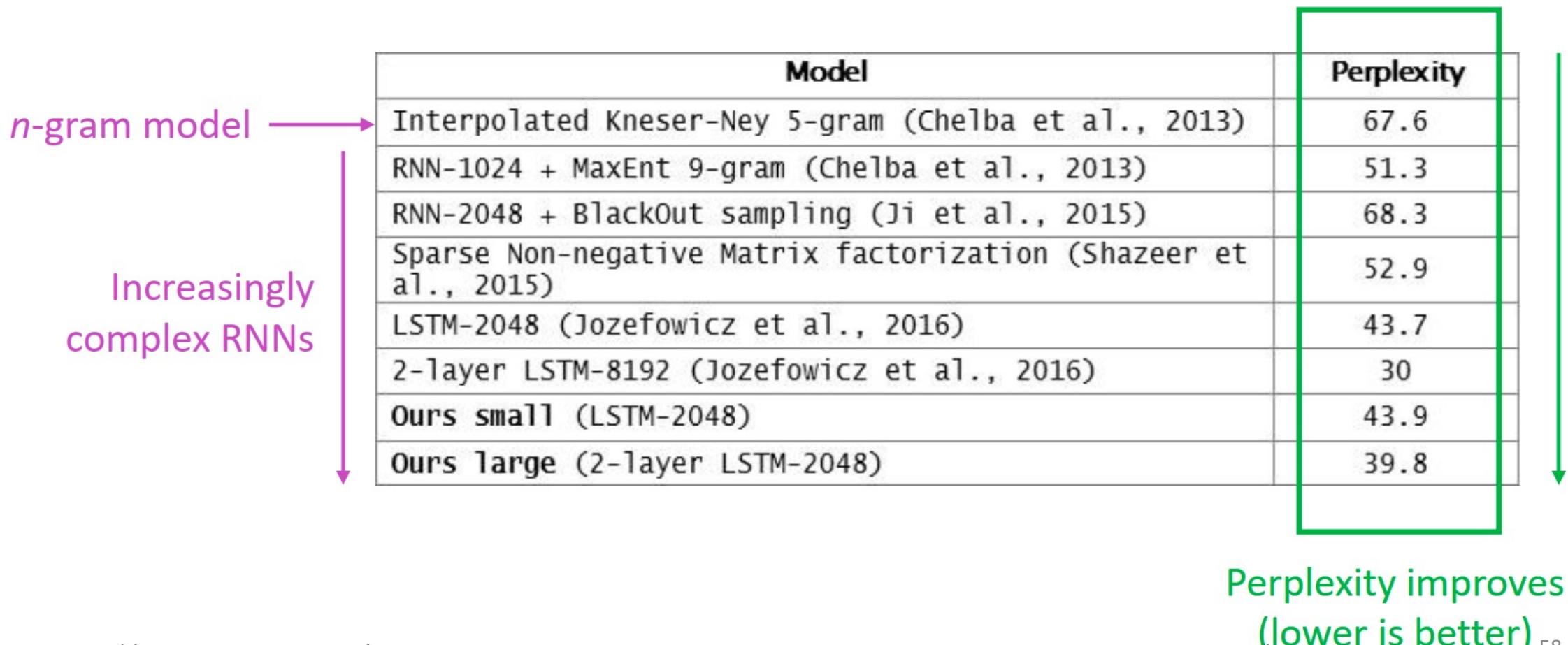
- Perplexity等同于交叉熵loss  $J(\theta)$  的指数值

困惑度值  
越低越好！

$$= \prod_{t=1}^T \left( \frac{1}{\hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}} \right)^{1/T} = \exp \left( \frac{1}{T} \sum_{t=1}^T -\log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)} \right) = \exp(J(\theta))$$

# 评估语言模型

- 相比N元语言模型，RNN语言模型的困惑度有明显降低



# 为什么我们需要关注语言模型？

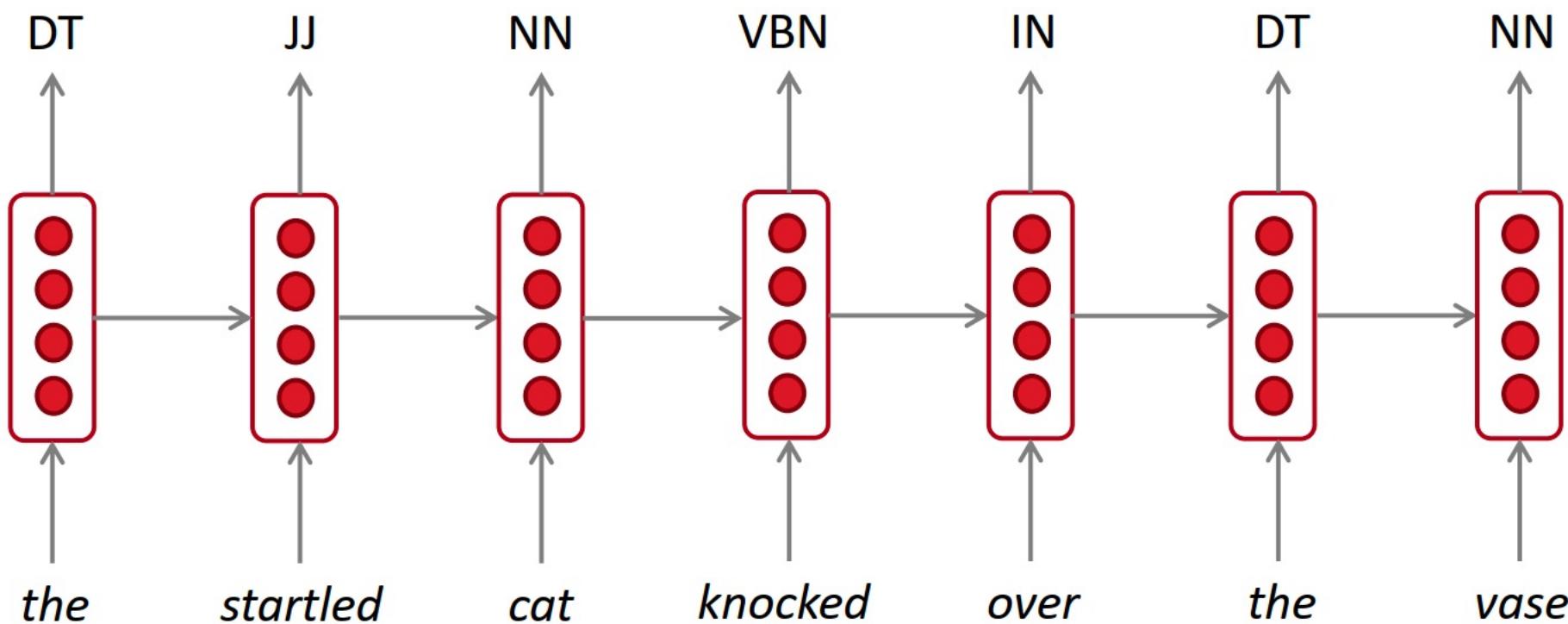
- 语言模型是一个基准任务。这个任务可以衡量我们在语言理解方面的进展
- 语言模型也是很多NLP任务的子模块，尤其对于需要生成文本或估计文本概率的NLP任务很重要，比如：
  - 输入预测
  - 语音识别
  - 手写识别
  - 语法改错
  - 机器翻译
  - 摘要
  - 对话
  - 等等

# RNN其他应用

- 语言模型：给定前面的词预测下一个词的模型
- 循环神经网络：一族满足如下条件的神经网络：
  - 输入的序列可以是任意长度
  - 在每一个时间步用的参数都是一样的
  - 在每一个时间步都可以做出选择性的预测
- 循环神经网络  $\neq$  语言模型
- 利用循环神经网络可以建立很好的语言模型
- 循环神经网络在很多其他的应用中也非常有效

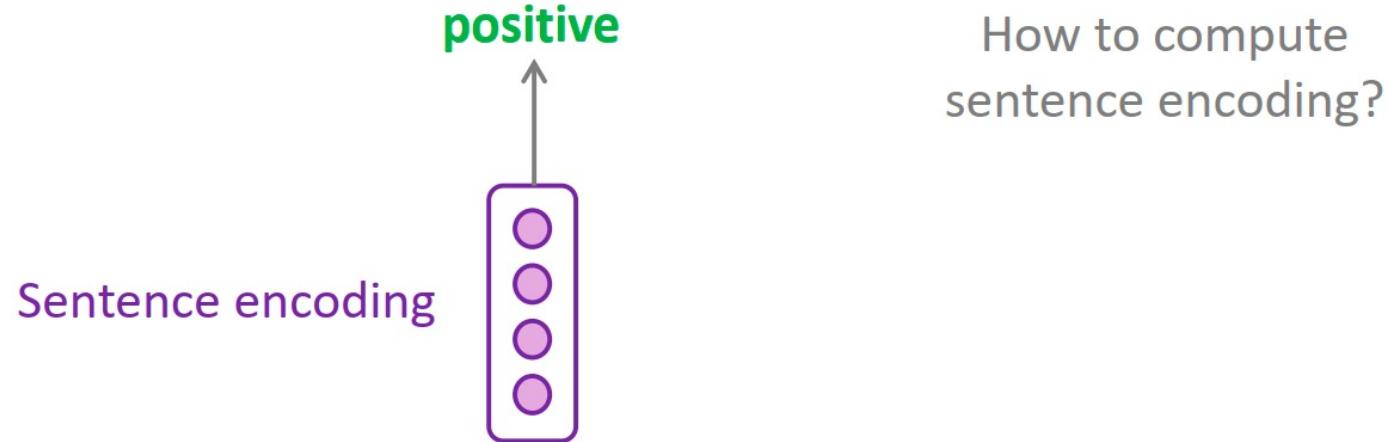
# RNN可以被用来做标注

- POS tag, NER

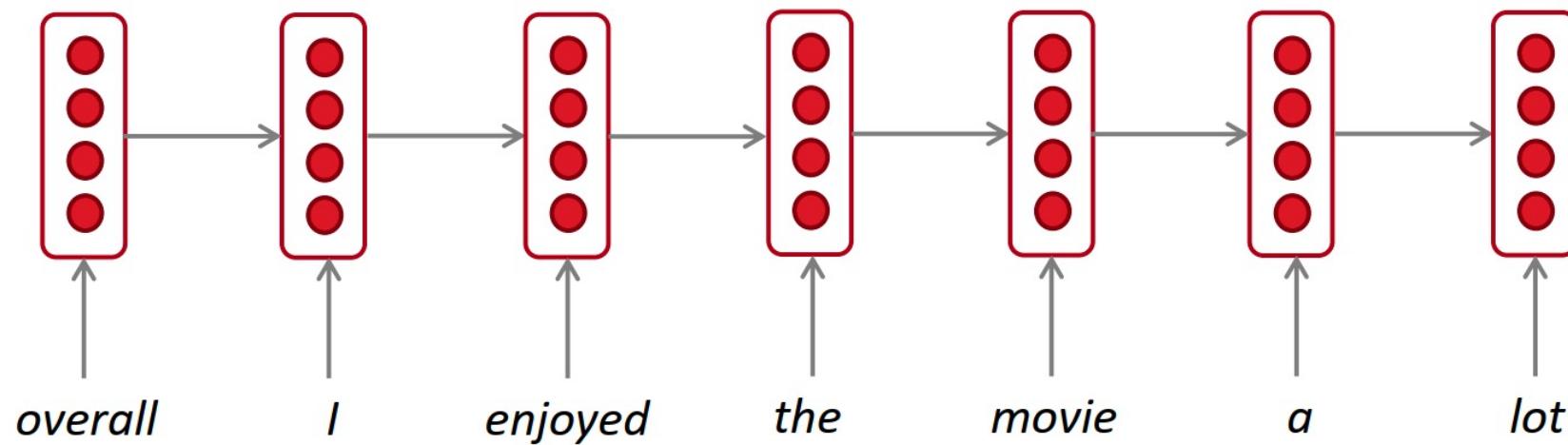


# RNN可以被用来作句子分类

- 比如，情感分析

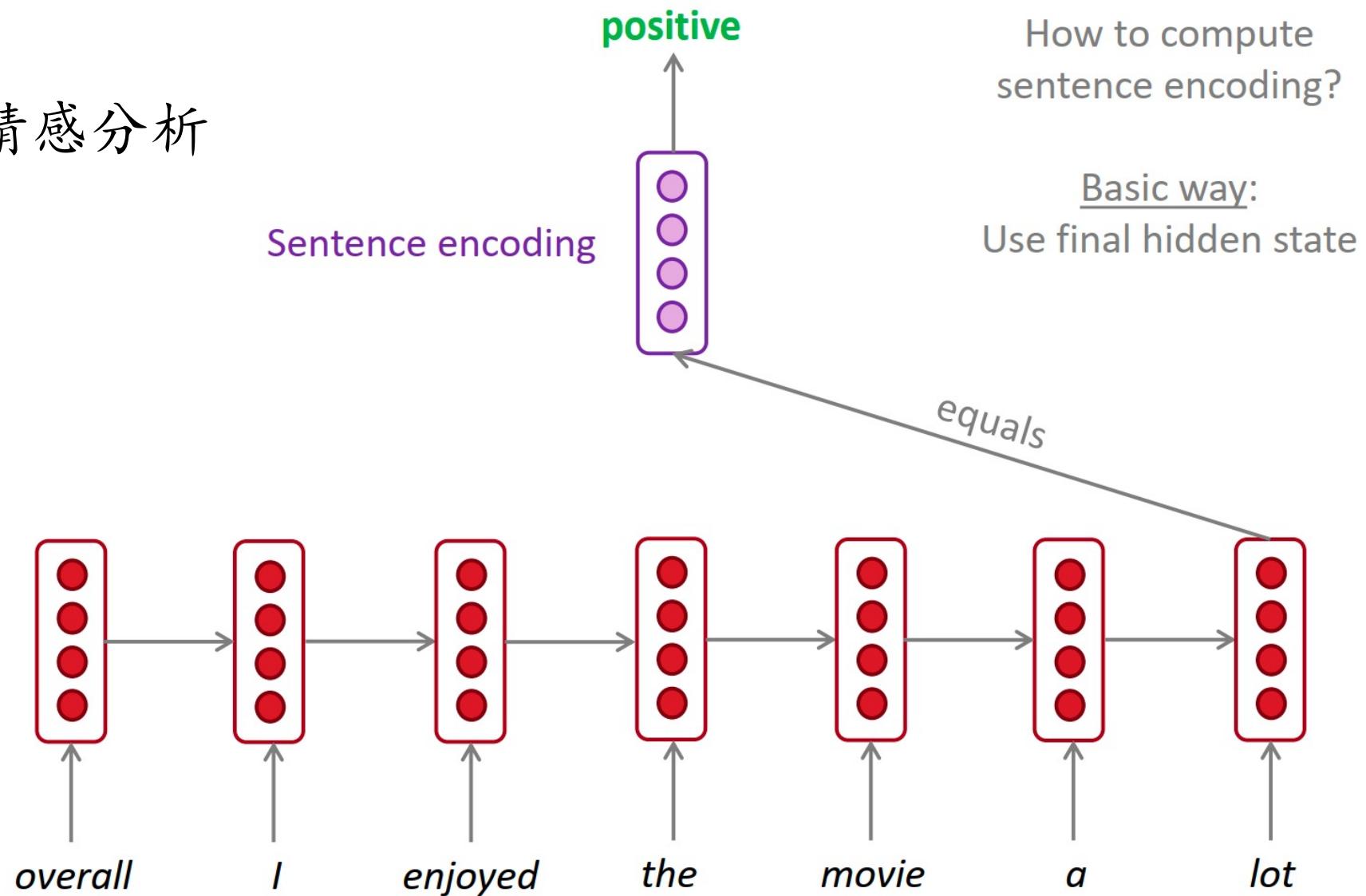


How to compute  
sentence encoding?



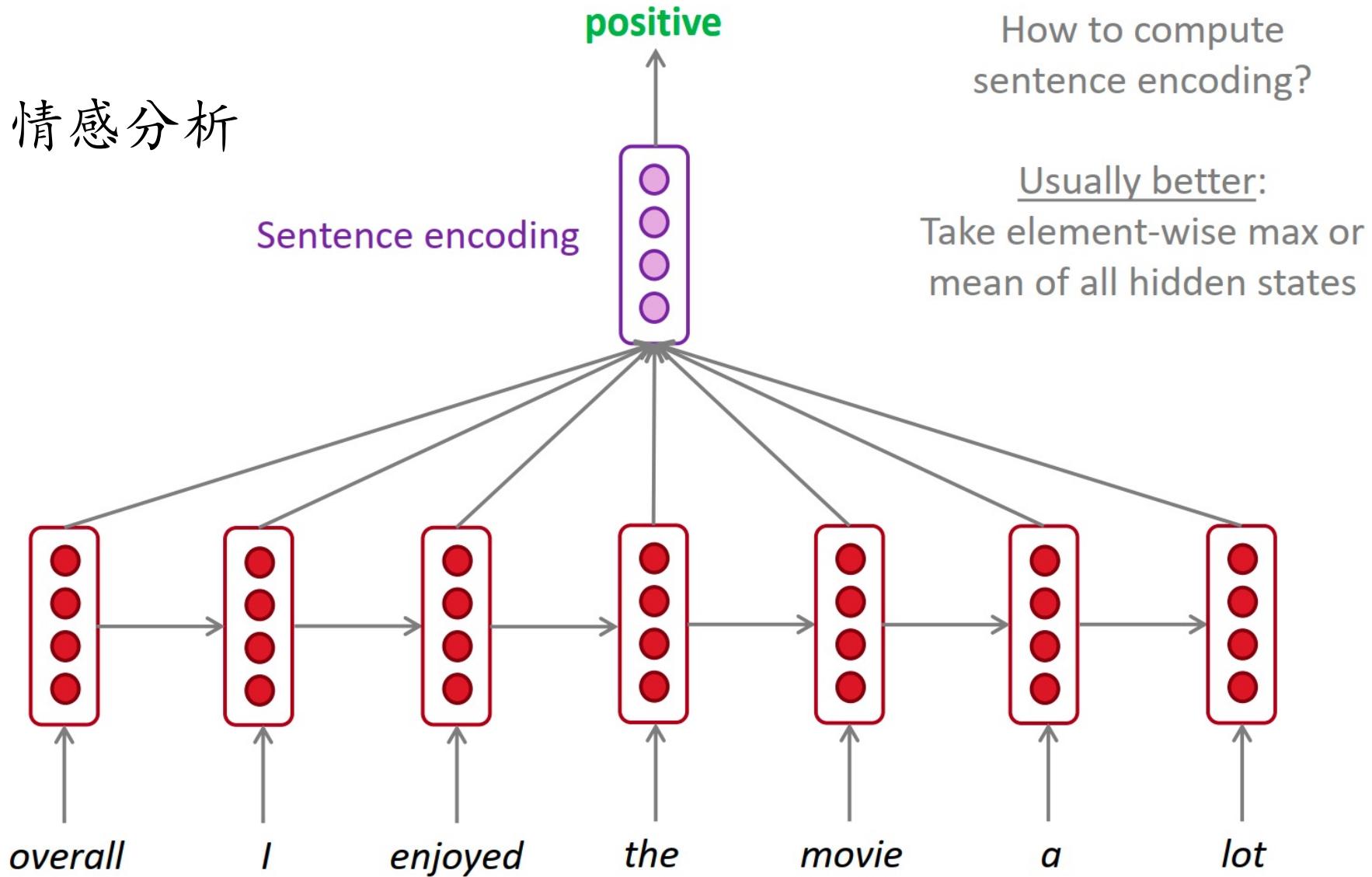
# RNN可以被用来作句子分类

- 比如，情感分析



# RNN可以被用来作句子分类

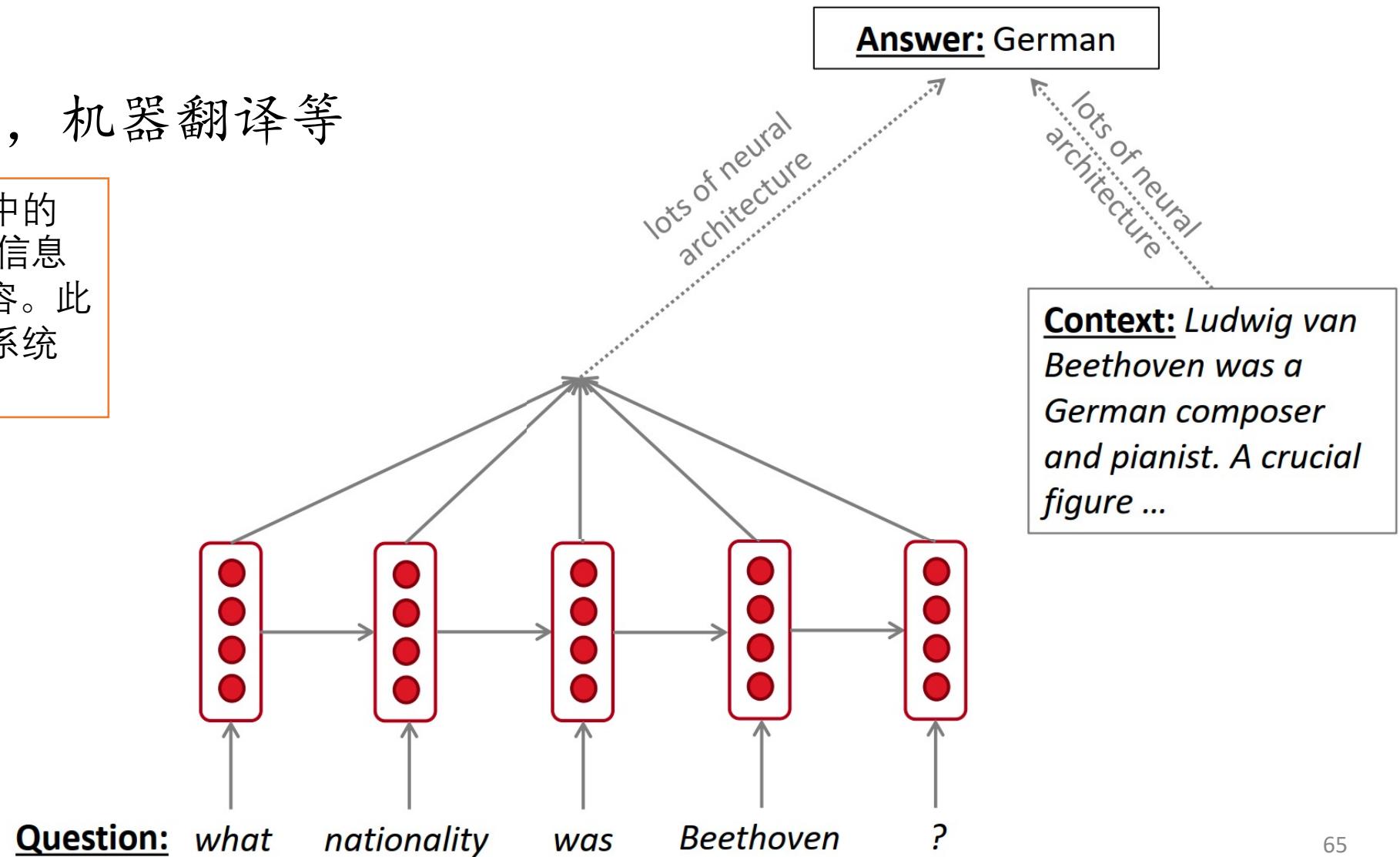
- 比如，情感分析



# RNN可以被用作编码器模块

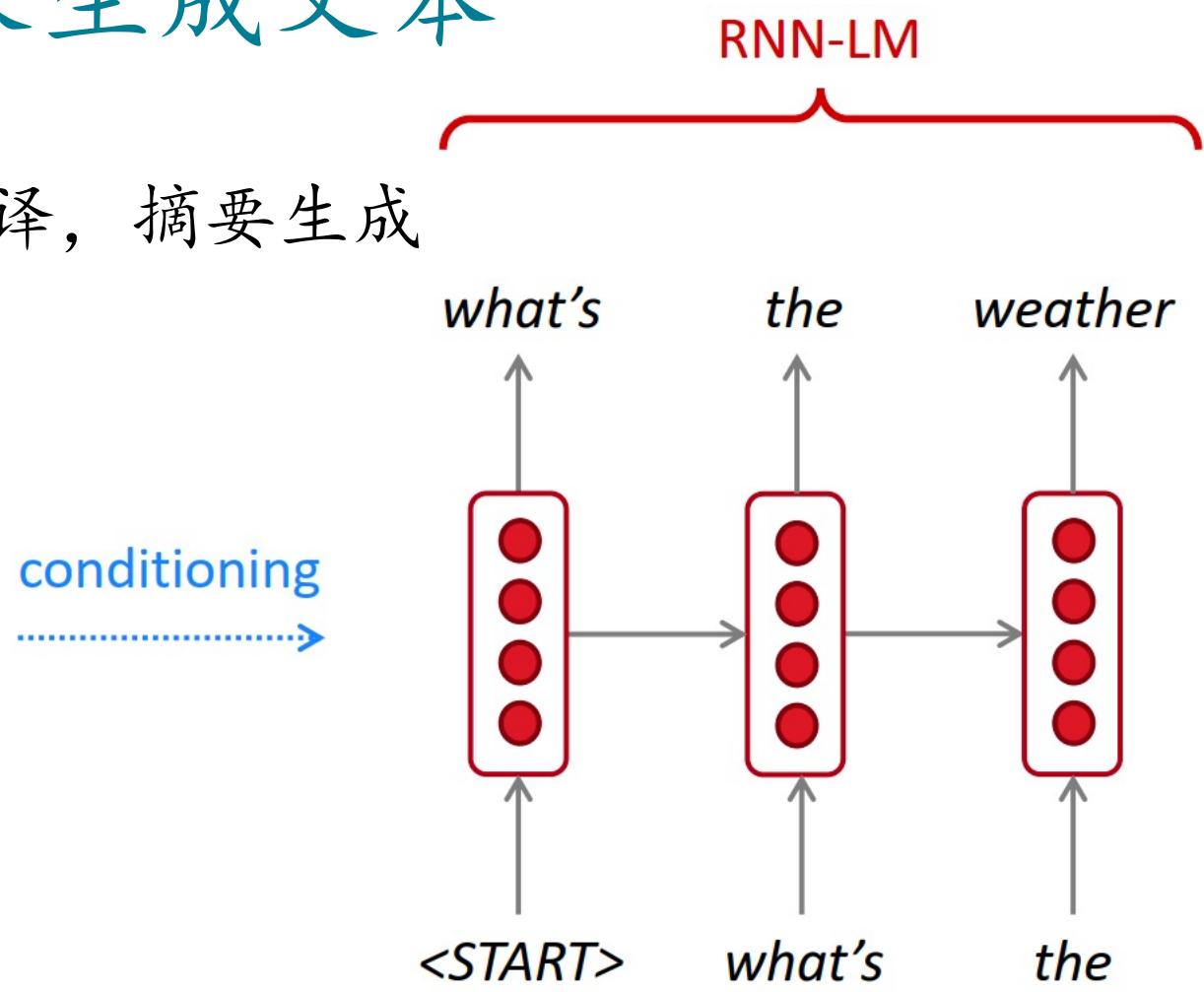
- 比如：问答，机器翻译等

此处RNN是QA系统中的Encoder，隐藏层的信息代表整个句子的内容。此时RNN是整个QA大系统的一部分



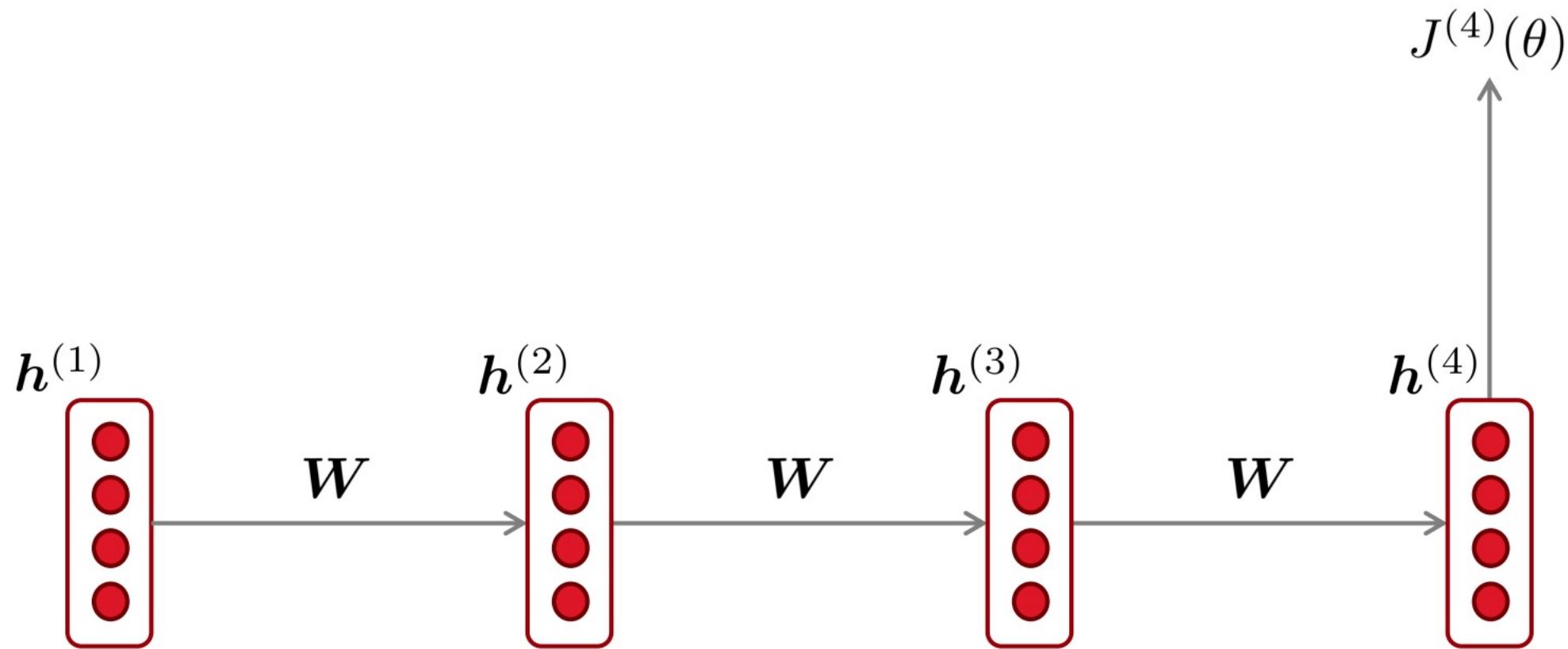
# RNN-LM可以用来生成文本

- 比如：语音识别，机器翻译，摘要生成

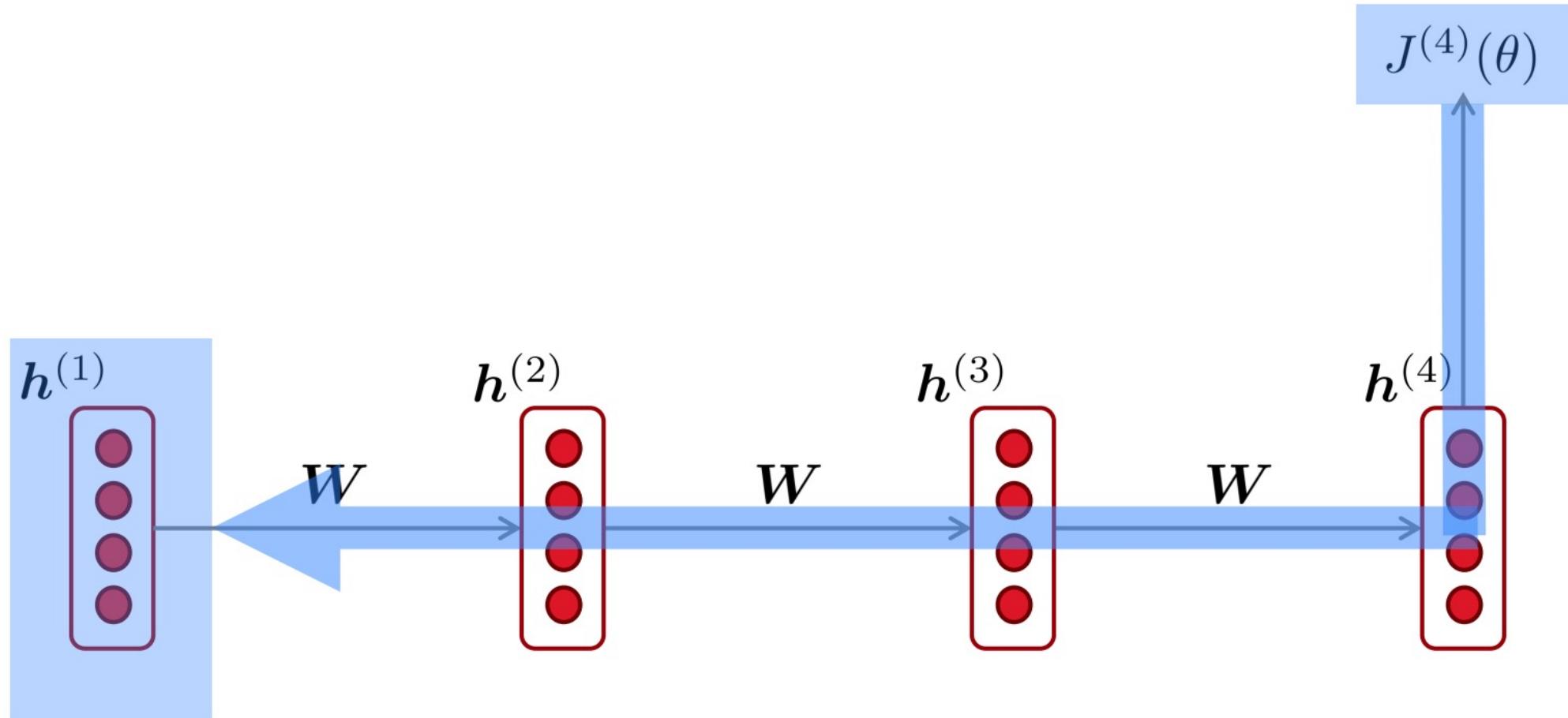


一个条件语言模型的例子

# RNN存在的问题；梯度消失，梯度爆炸

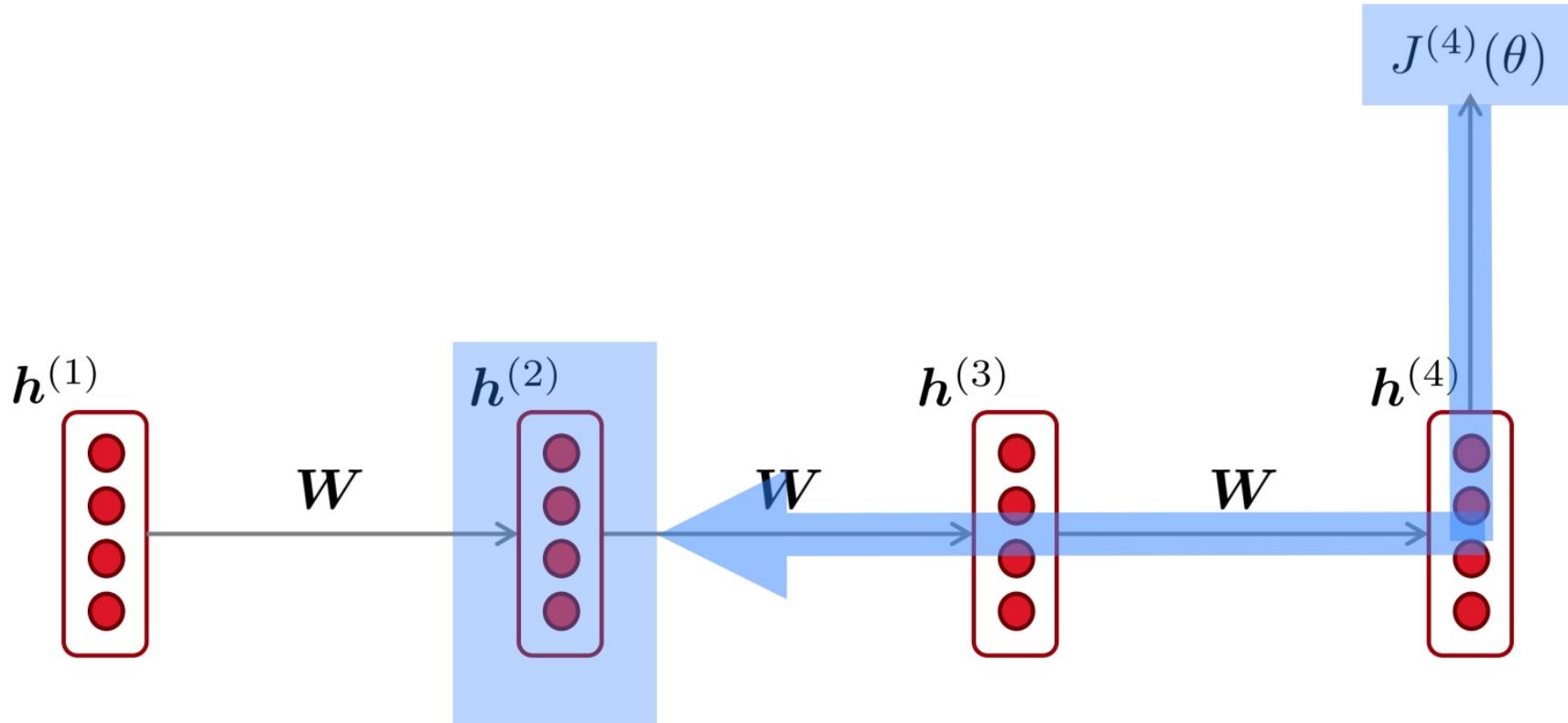


# RNN存在的问题；梯度消失，梯度爆炸



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?$$

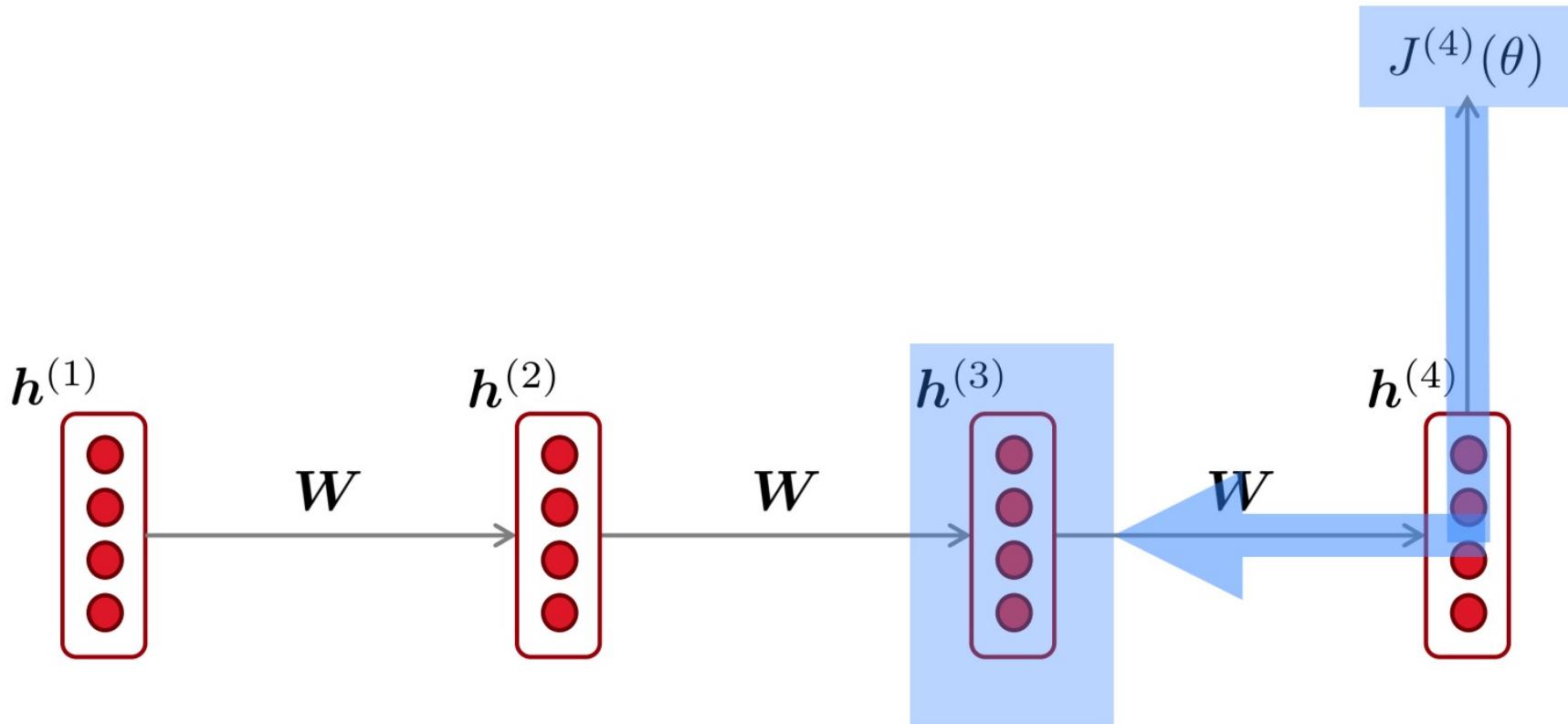
# RNN存在的问题；梯度消失，梯度爆炸



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(2)}}$$

chain rule!

# RNN存在的问题；梯度消失，梯度爆炸

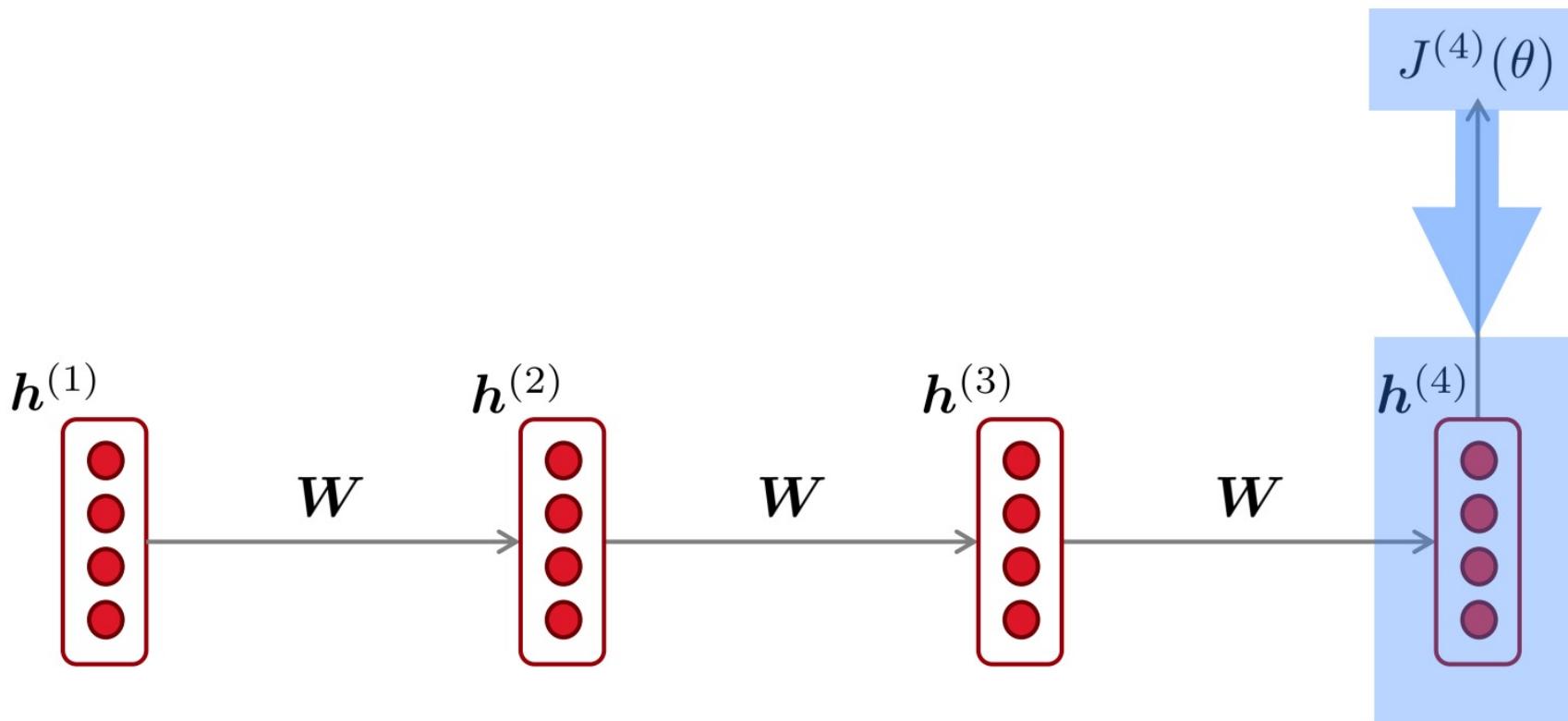


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

# RNN存在的问题；梯度消失，梯度爆炸



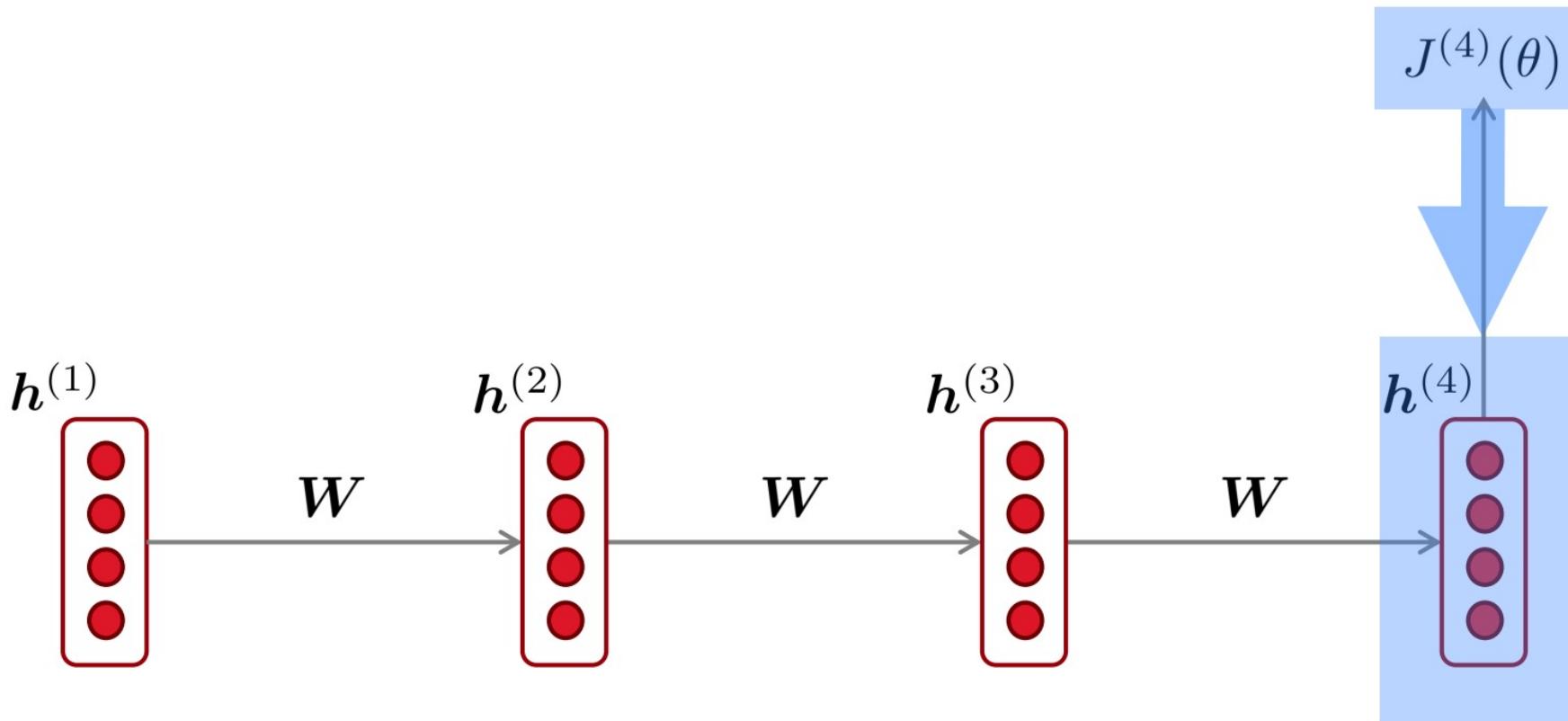
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times$$

$$\frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

# RNN存在的问题；梯度消失，梯度爆炸



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \boxed{\frac{\partial h^{(2)}}{\partial h^{(1)}}} \times$$

$$\boxed{\frac{\partial h^{(3)}}{\partial h^{(2)}}} \times$$

$$\boxed{\frac{\partial h^{(4)}}{\partial h^{(3)}}} \times \boxed{\frac{\partial J^{(4)}}{\partial h^{(4)}}}$$

如果这几个梯度值很小/大会出现什么现象？

如果这几个梯度值很小，那么随着反向传播层数的增加，梯度值会越来越小，直至消失

如果这几个梯度值很大，那么随着反向传播层数的增加，梯度值会越来越大，即梯度爆炸

# 梯度爆炸

- 为什么梯度爆炸会是问题？
- 如果梯度过大，SGD的更新步会变的过大

$$\theta^{new} = \theta^{old} - \underbrace{\alpha \nabla_{\theta} J(\theta)}_{\text{gradient}}$$

learning rate

- 步长太大，有可能得到很差的参数，损失函数值也可能会升高
  - 认为前面是个山峰，迈了一大步，然而山峰后面有个山谷。
- 甚至可能导致参数值或损失函数值出现 Inf 或 NaN
  - 这就需要从上一个 checkpoint 重新训练

# 梯度爆炸

- 一般情况下如何解决
  - 简单的解决办法：梯度裁剪，只要梯度的范数超过某个阈值，那么在 SGD 更新前直接将其规范到某个范围内

---

**Algorithm 1** Pseudo-code for norm clipping

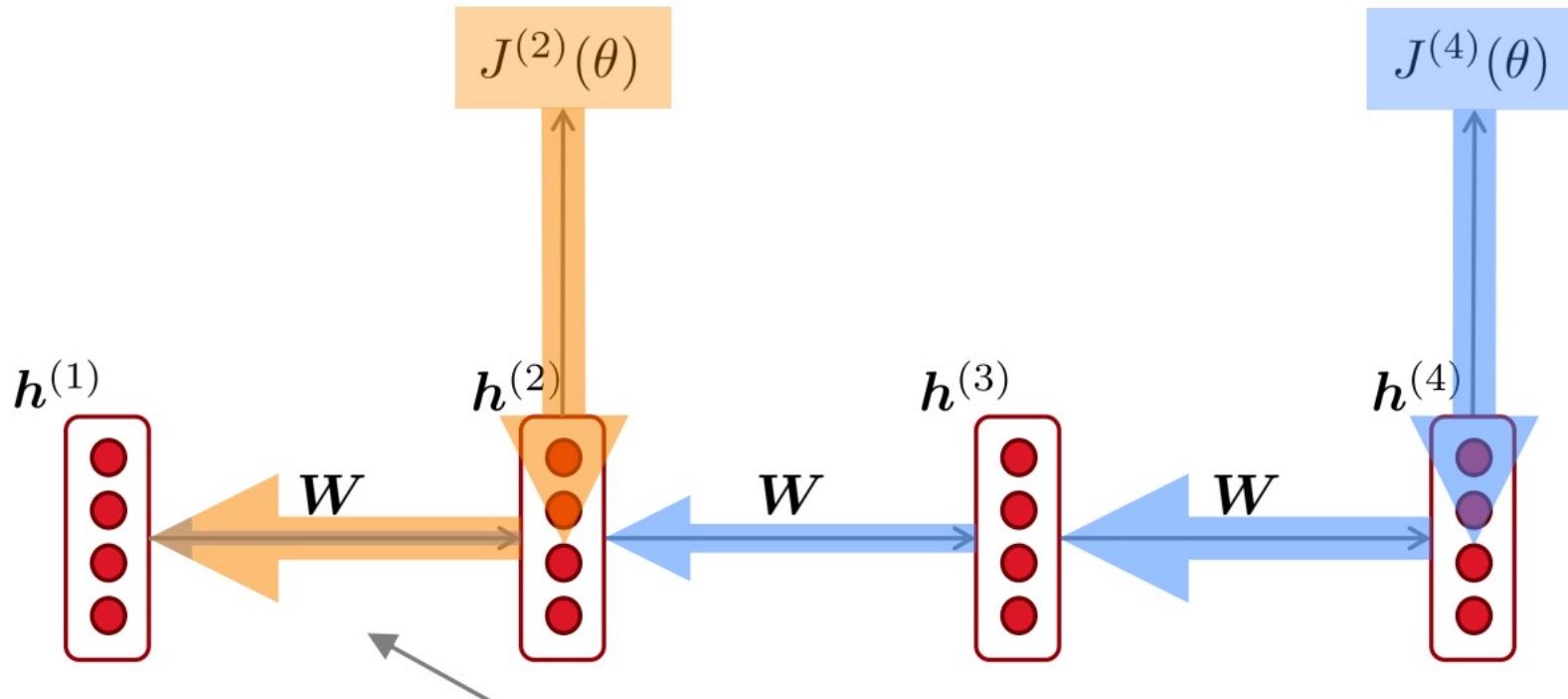
---

```
 $\hat{g} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$ 
if  $\|\hat{g}\| \geq threshold$  then
     $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|} \hat{g}$ 
end if
```

---

- 直观解释是：方向保持不变，但是跨一个比较小的步长
- 事实上，在目前的神经网络训练中，梯度裁剪已经成为标配，相比梯度消失，梯度爆炸是个相对容易处理的问题。

# 为什么梯度消失是个问题呢？



远处的梯度信号会丢失，因为它相比近处的梯度信号要弱太多了。

所以模型的权重只能被近处的训练信号影响，不会被远处的信号影响

# 如何解决梯度消失问题

- 主要问题是； RNN的架构无法记住太久远的信息
- 在普通RNN中（Vanilla RNN），隐藏层的信息会不断地与参数矩阵相乘
$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b} \right)$$
- 如果给RNN一个额外的记忆单元，使其可以不断累加会不会更好？

长短记忆网络 LSTM, 及其简化版GRU

# 长短记忆网络 LSTM

- 1997年由Hochreiter and Schmidhuber 提出
- 在每个时间步 $t$ , 都有一个隐藏状态  $h^{(t)}$  和一个单元状态 (cell state)  $c^{(t)}$ 
  - 两个状态都是长度为n的向量
  - 单元状态存储长期的信息
  - LSTM可以在cell state上读取, 擦除以及写入信息
    - Cell state的作用很像计算机系统中的随机访问存储器 (RAM)
- 如何选择哪些信息需要被读取, 擦除以及写入呢?
  - 用三个门: 输入门, 输出门, 遗忘门
  - 门的开关利用一个长度为n的向量来控制, 其中的每个元素可以是打开 (1), 关闭 (0) 或者中间的某个值
  - 控制门的向量是动态的, 即是由当前的上下文计算出来的

# LSTM

Sigmoid函数：保证所有的门向量都在0, 1之间

- 输入序列记作  $x^{(t)}$ , 我们会在每一个时间步计算隐藏状态  $h^{(t)}$  和一个单元状态  $c^{(t)}$

遗忘门：控制有多少信息被遗忘/被保留

输入门：控制新的记忆单元中的信息哪一部分要被写入记忆单元

输出门：控制哪一部分信息要被输出到隐藏层

新的记忆内容：这是本时间步准备好的新的信息

记忆单元：擦除掉一些记忆的内容，并接受一些新的内容

隐藏层：从记忆单元中输出一些信息

$$f^{(t)} = \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f)$$

$$i^{(t)} = \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i)$$

$$o^{(t)} = \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o)$$

所有这些门都是长度为n的向量

$$\tilde{c}^{(t)} = \tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c)$$

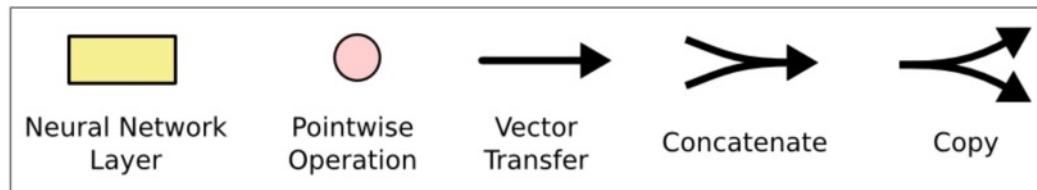
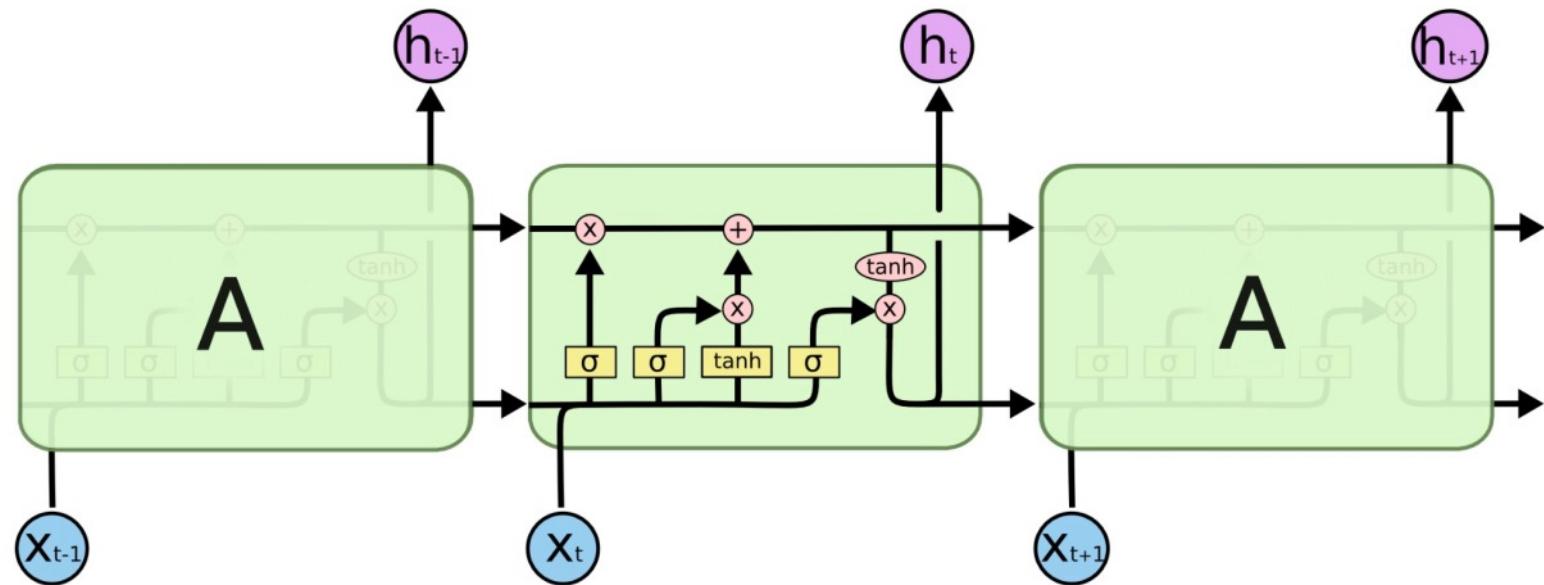
$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

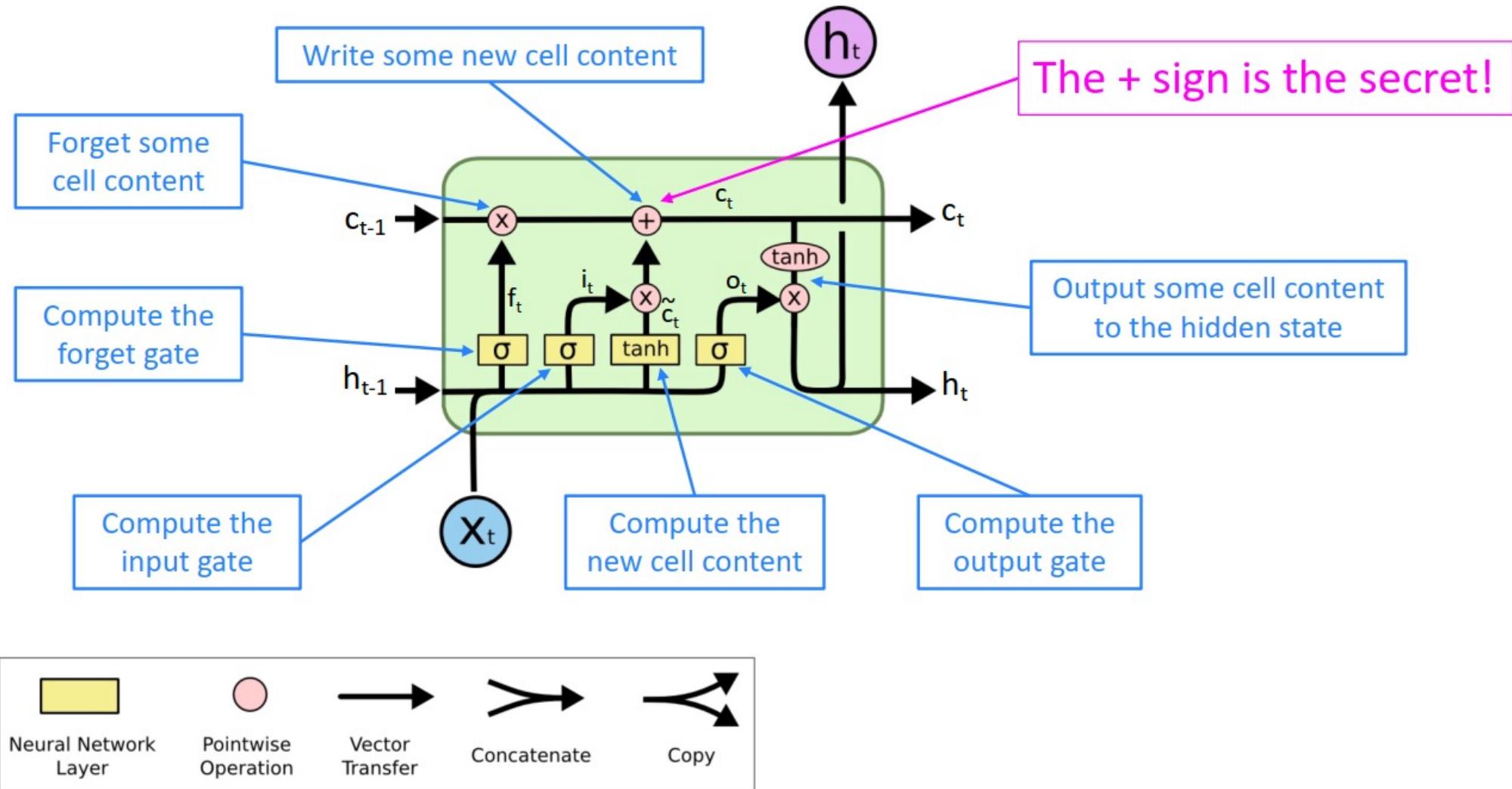
门是利用element-wise product来控制信息的传入量的（又称矩阵的Hadamard积）

# LSTM

- 以上公式可以简单理解为下图



# Long Short-Term Memory (LSTM)



# Gated Recurrent Units (GRU)

- 作为LSTM的简单的替代由Cho et al. 在 2014年提出
- 在每个时间步t, 我们只有输入和隐藏状态 (没有记忆单元)

更新门 : 控制隐状态中的哪个部分被更新/被保留

$$\mathbf{u}^{(t)} = \sigma(\mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u)$$

重置门 : 控制前一个隐状态中的哪个部分要被用来计算新的内容

$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_r \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r)$$

新的隐状态 : 重置门选择了前一个隐状态中的有用部分, 用它和当前的输入来计算新的隐层内容

$$\tilde{\mathbf{h}}^{(t)} = \tanh(\mathbf{W}_h (\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h)$$

隐状态 : 更新门同步控制前一个隐状态中保留的部分以及要更新到新的隐状态中的部分

$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

与LSTM一样有保持长期信息的能力

# LSTM vs GRU

- 研究人员提出了许多门控RNN变体，其中LSTM和GRU的应用最为广泛
- 最大的区别是GRU计算速度更快，参数更少
- 没有确凿的证据表明其中一个总是比另一个表现得更好
- LSTM 是一个很好的默认选择(特别是当你的数据具有非常长的依赖关系，或者你有很多训练数据时)
- 经验法则：从LSTM开始，但是如果你想要更有效率，就切换到GRU

# How does LSTM solve vanishing gradients?

- RNN的LSTM架构更容易保存许多时间步上的信息
- 如果某一维度上遗忘门设置为1，输入门设置成0。那么当维度中的信息将被无限地保存
- 相比之下，普通RNN更难学习重复使用并且在隐藏状态中保存信息的矩阵
- LSTM并不保证没有梯度消失/爆炸，但它确实为模型提供了一种更容易的方法来学习远程依赖关系

# LSTMs: real-world success

- 2013-2015年，LSTM开始霸榜
  - 成功的任务包括：手写识别、语音识别、机器翻译、parsing、图像字幕
  - LSTM成为主导方法
- 目前，其他方法(如Transformers)在某些任务上变得更加主导
  - 例如在WMT(a MT conference + competition)中
  - 在2016年WMT中，总结报告包含“RNN”44次
  - 在2018年WMT中，总结报告包含“RNN”9次，“Transformers” 63次

# Is vanishing/exploding gradient just a RNN problem?

- 梯度消失/爆炸只是RNN问题吗？
- 并不是，这对于所有的神经结构(包括前馈和卷积网络)都是一个问题，尤其是对于深度结构
  - 由于链式法则/选择非线性函数，反向传播时梯度可以变得很小很小
  - 因此，较低层次的学习非常缓慢(难以训练)
- 解决方案：大量新的深层前馈 / 卷积架构，添加更多的直接连接(从而使梯度可以流动)
- 例如：
  - 残差连接又名“ResNet”，也称为skip connection
  - identity connection 可以默认的保持信息不动
  - 这使得深层网络更容易训练

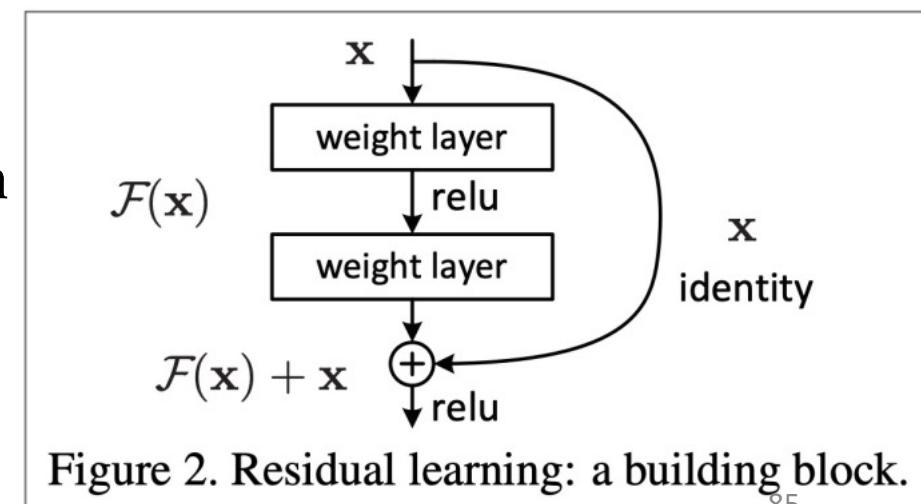
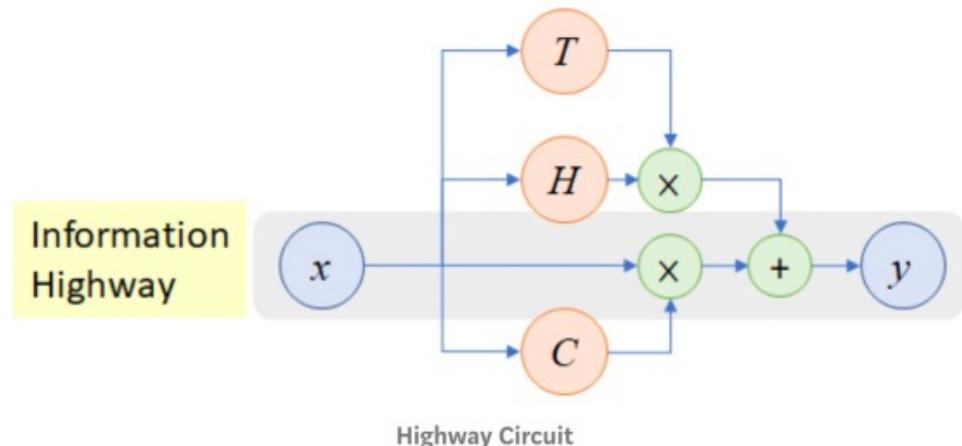
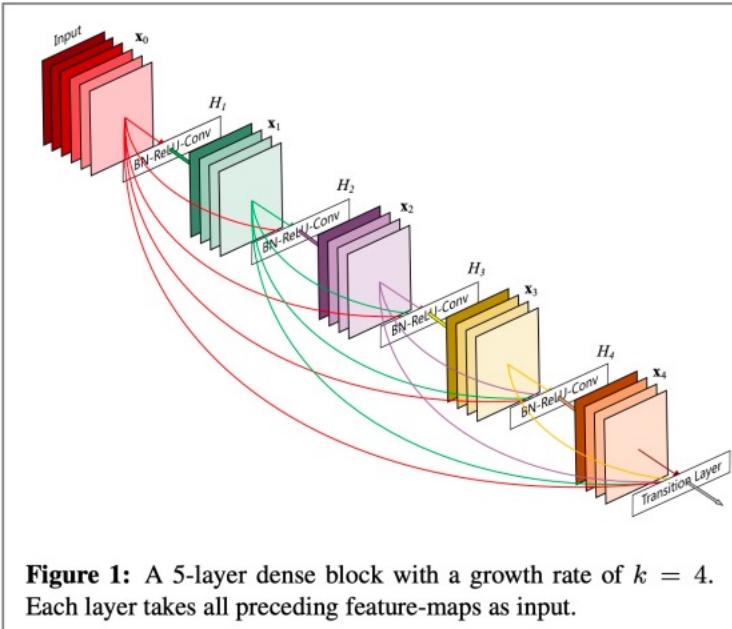


Figure 2. Residual learning: a building block.

# Is vanishing/exploding gradient just a RNN problem?

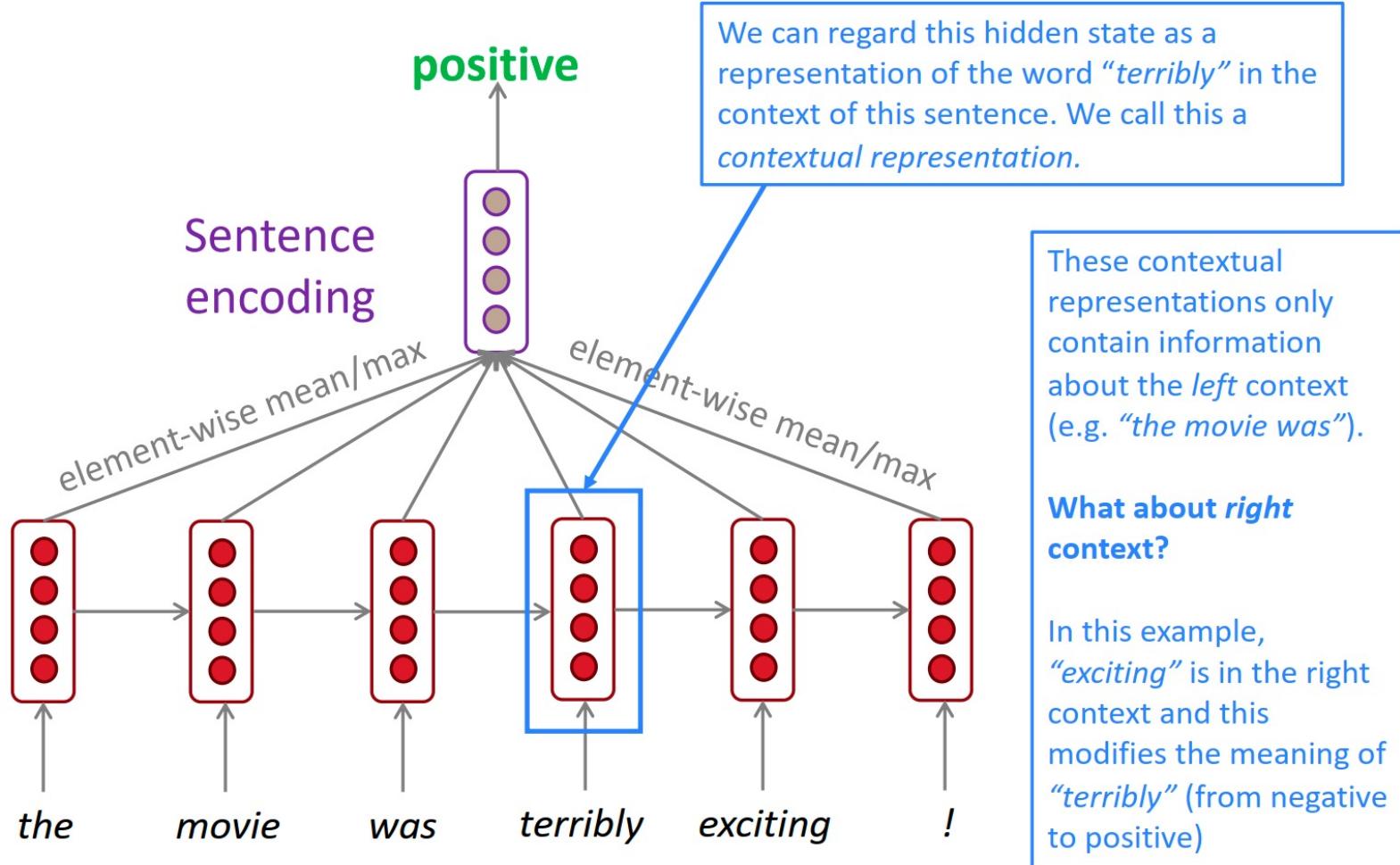
- 其他方法
  - 稠密连接：DenseNet
  - 直接把每一层都连接到未来的层
- Highway连接又称HighwayNet
  - 类似于残差连接，但identity connection与转换层由动态门控制
  - 灵感来自LSTMs，但适用于深度前馈/卷积网络



结论：虽然梯度消失/爆炸是一个普遍的问题，但由于重复乘以相同的权矩阵，RNN尤其不稳定[Bengio et al, 1994]

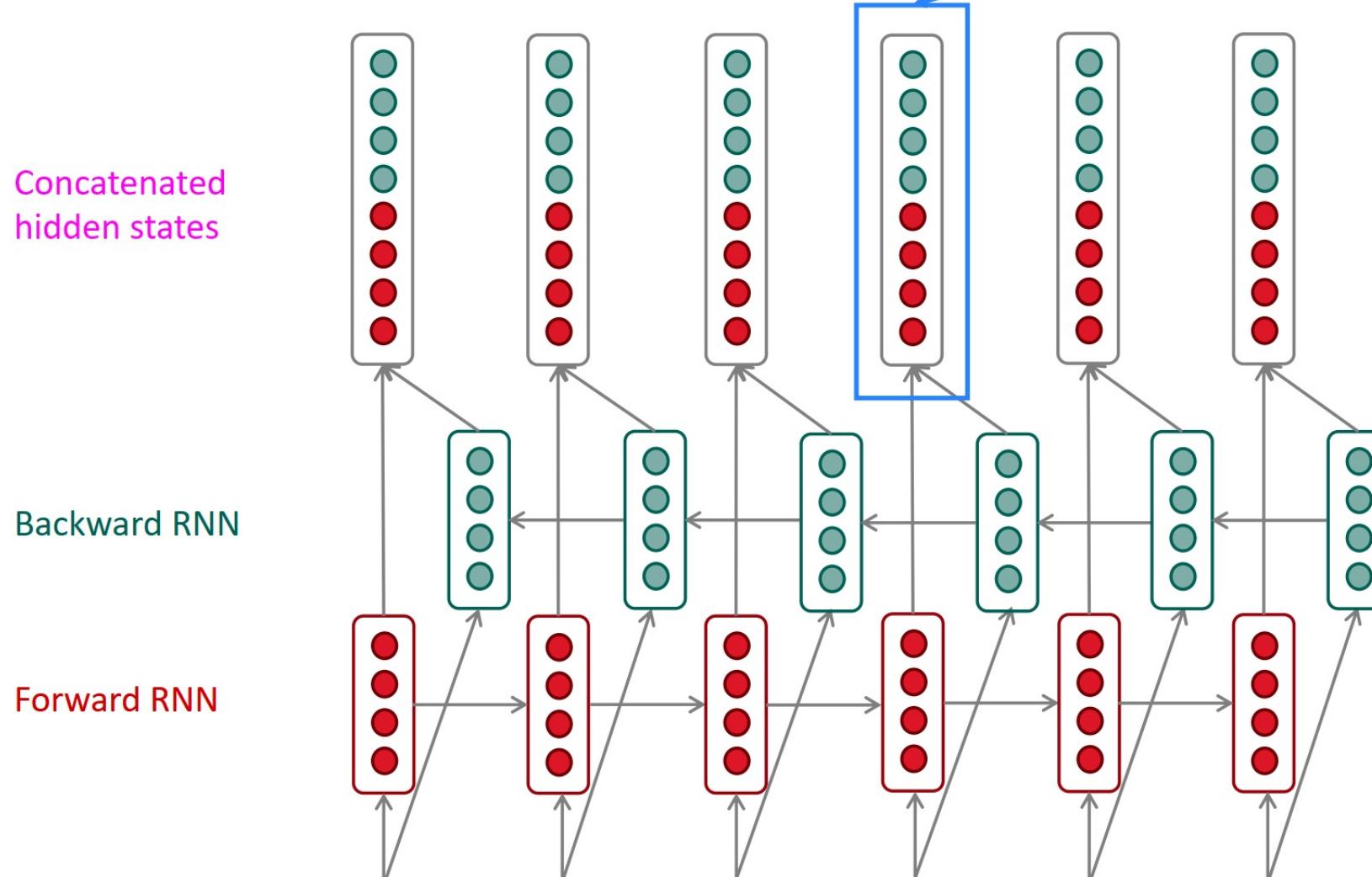
# Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification



# Bidirectional RNNs

This contextual representation of “terribly”  
has both left and right context!



# Bidirectional RNNs

On timestep  $t$ :

This is a general notation to mean “compute one forward step of the RNN” – it could be a simple, LSTM, or other (e.g., GRU) RNN computation.

Forward RNN

$$\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, \mathbf{x}^{(t)})$$

Backward RNN

$$\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, \mathbf{x}^{(t)})$$

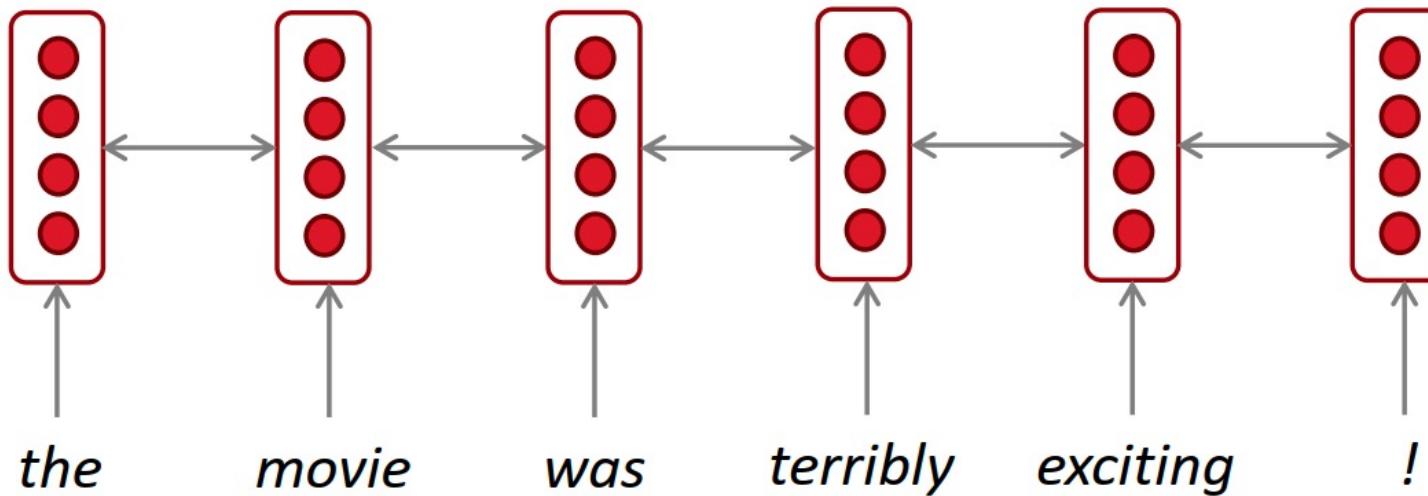
Concatenated hidden states

$$h^{(t)} = [\vec{h}^{(t)}; \overleftarrow{h}^{(t)}]$$

Generally, these two RNNs have separate weights

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

# Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

# Bidirectional RNNs

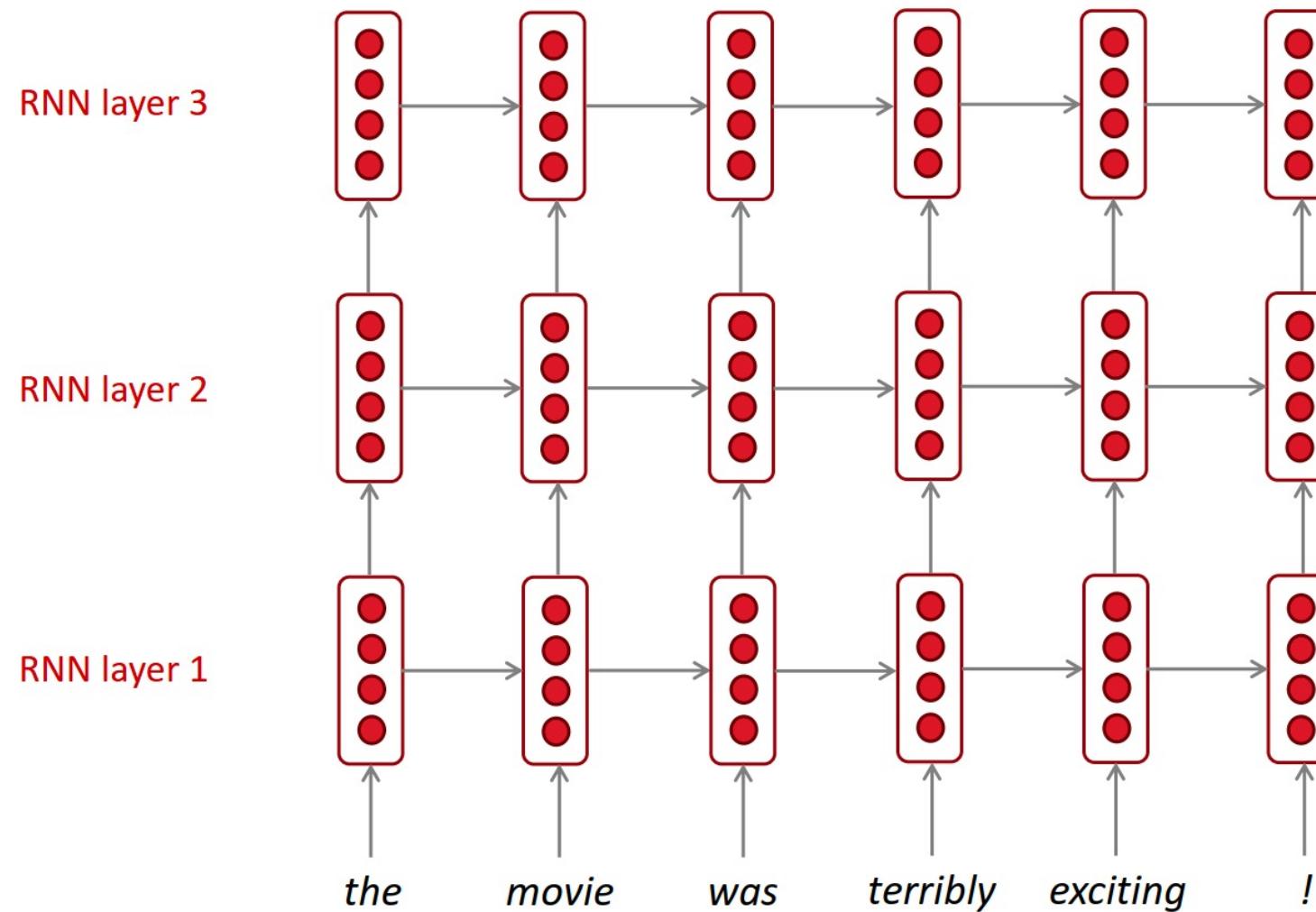
- 注意：双向RNNs只适用于访问整个输入序列的情况
  - 它们不适用于语言建模，因为在LM中，你只有左侧的上下文可用
- 如果你有完整的输入序列(例如任何一种编码)，双向性是强大的(默认情况下你应该使用它)
- 例如，BERT(来自transformer的双向编码器表示)是一个基于双向性的强大的预训练的上下文表示系统
  - 你会在课程的后面学到更多关于BERT的知识！

# Multi-layer RNNs

- RNNs在一个维度上已经是“deep”(它们展开到许多时间步长)
- 我们还可以通过应用多个RNN使它们“深入”到另一个维度：这是一个多层RNN
- 这就允许网络计算更复杂的特征
  - 较低的RNN应该计算较低级别的特征，而较高的RNN应该计算较高级别的特征
- 多层RNN也称为堆叠RNN (stacked RNN)

# Multi-layer RNNs

The hidden states from RNN layer  $i$  are the inputs to RNN layer  $i+1$



# Multi-layer RNNs in practice

- 高性能的RNNs通常有多层的(但没有卷积或前馈网络那么深)
- 例如：在2017年的一篇论文，Britz et al 发现在神经机器翻译中，2到4层对于RNN编码器是最好的,4层对于RNN解码器是最好的
  - 通常2层比一层强，3层比2层稍强
  - 训练更深RNNs(例如8层)时，需要**skip-connections / dense-connections**
  - RNN无法并行化，计算代价过大，所以不会过深
- Transformer-based 的网络(如BERT)可以多达24层
  - BERT有很多skipping-like的连接

Thank you