

# 神经网络语言模型

张家俊

[jjzhang@nlpr.ia.ac.cn](mailto:jjzhang@nlpr.ia.ac.cn)

模式识别国家重点实验室  
中国科学院自动化研究所




# 基于计数的N-元语言模型

$$\begin{aligned}
 & P(w_1 w_2 \cdots w_{t-1} w_t) \\
 = & \prod_{t=1}^n P(w_t | w_{t-1} \cdots w_1) \\
 \approx & \prod_{t=1}^n P(w_t | w_{t-1} \cdots w_{t-n+1}) \\
 & \quad \quad \quad \downarrow \\
 = & \frac{P(w_t | w_{t-1} \cdots w_{t-n+1})}{\text{count}(w_{t-1} \cdots w_{t-n+1} w_t)} \\
 & \quad \quad \quad \text{count}(w_{t-1} \cdots w_{t-n+1})
 \end{aligned}$$

# 基于计数的N-元语言模型

该课程很枯燥，大家觉得很无聊。

$$P(\text{无聊}|\text{很}) \\ = \frac{\text{count}(\text{很 无聊})}{\text{count}(\text{很})}$$


问题①：数据稀疏  
N-元组“很 无聊”未出现过，则回退

问题②：忽略语义相似性  
“无聊”与“枯燥”虽语义相似，但无法共享信息

# 基于计数的N-元语言模型

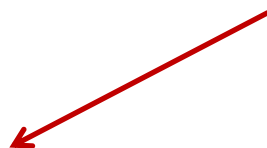
该课程很枯燥，大家觉得很无聊。

$$= \frac{P(\text{无聊}|\text{很}) \cdot \text{count}(\text{很 无聊})}{\text{count}(\text{很})}$$

$P(\text{无聊}|\text{很})$

**vs.**

$P(\text{枯燥}|\text{很})$



问题①：数据稀疏  
N-元组“很 无聊”未出现过，则回退

问题②：忽略语义相似性  
“无聊”与“枯燥”虽语义相似，但无法共享信息

# 词语表示

- 典型方法：抽象符号（字符串）

该 课程 很 枯燥 ， 大家 觉得 很 无聊 。

$w_0$ =该  $w_1$ =课程  $w_2$ =很  $w_3$ =枯燥  $w_4$ =,   
  $w_5$ =大家  $w_6$ =觉得  $w_7$ =很  $w_8$ =无聊  $w_9$ =。

- 等价表示方法：one-hot表示法

$|V|$

$$\begin{bmatrix} \vdots \end{bmatrix}$$


所有词按照出现的顺序排序



每个词语将对应唯一的下标

枯燥

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

无聊

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

# 词语表示

- 问题

枯燥

$$\begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

无聊

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \mathbf{1} \\ 0 \end{bmatrix}$$

枯燥  $\otimes$  无聊

$$\begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$\times$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \mathbf{1} \\ 0 \end{bmatrix}$$

$= 0$



任意两个词之间的相似度都为0!

# 现实世界 VS. 认知世界

- 现实世界：物体相互独立地存在





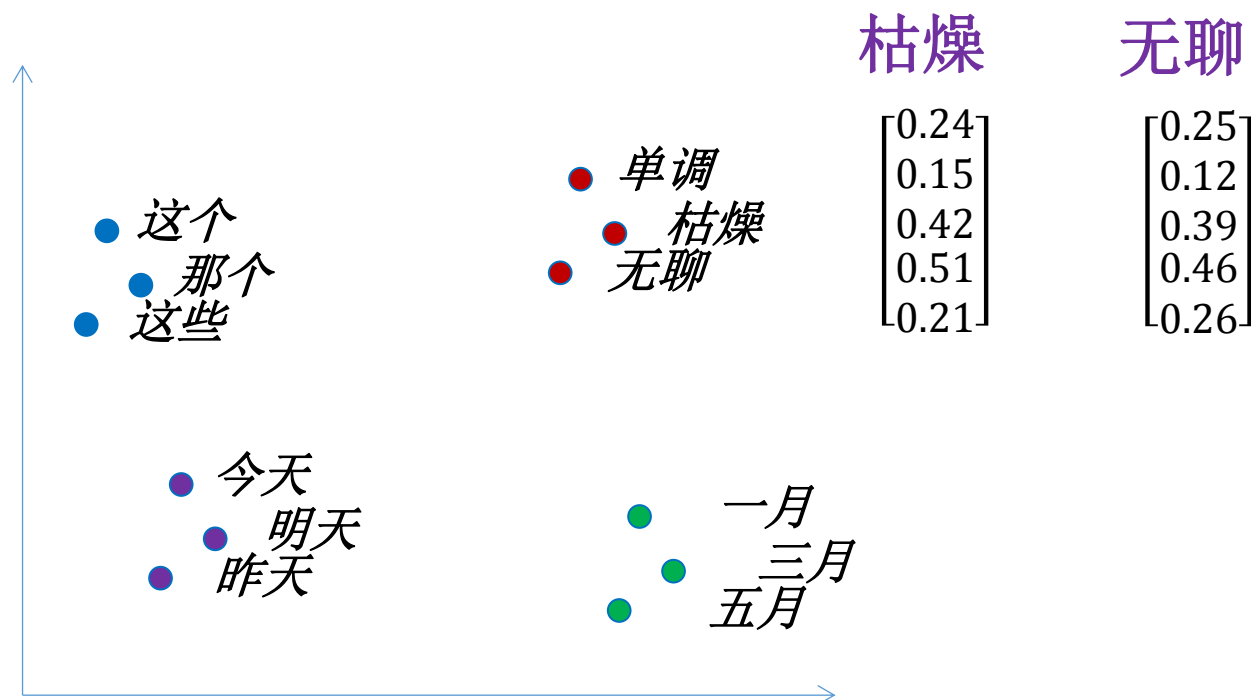
# 现实世界 VS. 认知世界

- 认知世界：概念互相联系、语义连续分布





# 基于连续语义空间的词语表示



低维、稠密的连续实数空间

# 基于分布式表示的N-元语言模型

很

$P(\text{无聊}|\text{很})$

**vs.**

$P(\text{枯燥}|\text{很})$

$\begin{bmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \end{bmatrix}$

$$P \left( \begin{bmatrix} 0.25 \\ 0.12 \\ 0.39 \\ 0.46 \\ 0.26 \end{bmatrix} \middle| \begin{bmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \end{bmatrix} \right)$$

**vs.**

$$P \left( \begin{bmatrix} 0.24 \\ 0.15 \\ 0.42 \\ 0.51 \\ 0.21 \end{bmatrix} \middle| \begin{bmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \end{bmatrix} \right)$$

$$f \left( \begin{bmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.25 \\ 0.12 \\ 0.39 \\ 0.46 \\ 0.26 \end{bmatrix} \right)$$

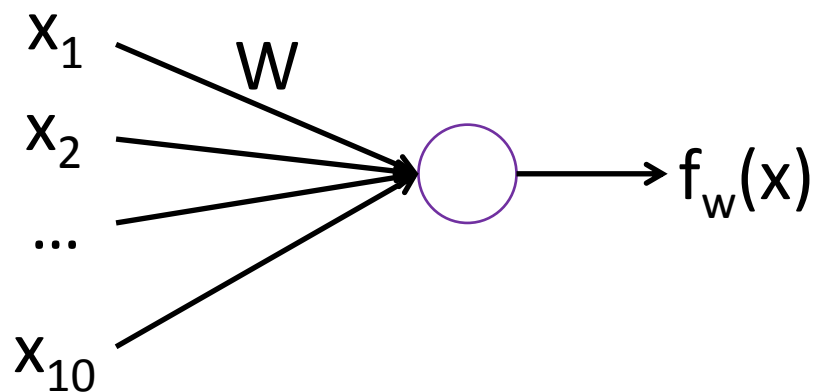
**vs.**

$$f \left( \begin{bmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.24 \\ 0.15 \\ 0.42 \\ 0.51 \\ 0.21 \end{bmatrix} \right)$$

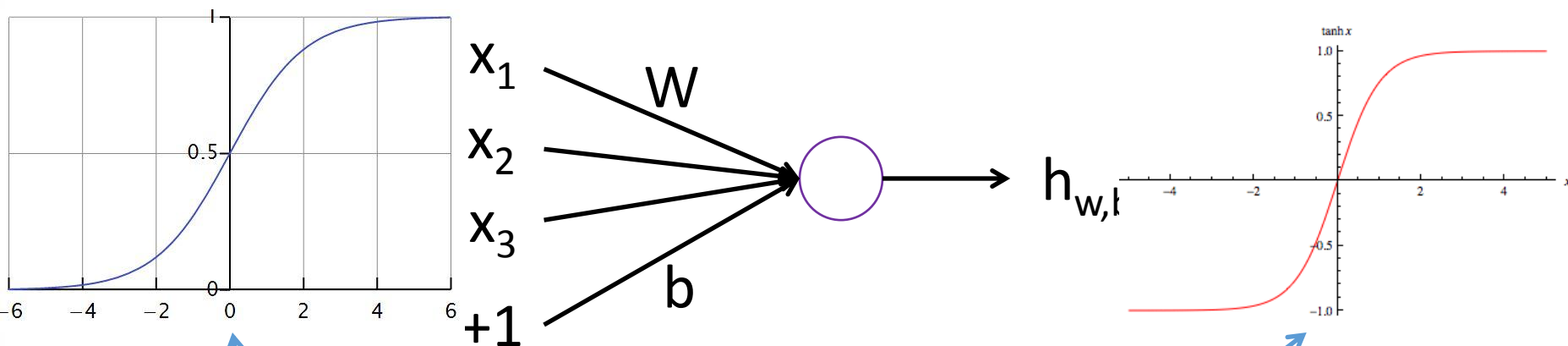
# 基于分布式表示的N-元语言模型

$P(\text{无聊}|\text{很})$

$$f \begin{pmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.25 \\ 0.12 \\ 0.39 \\ 0.46 \\ 0.26 \end{pmatrix} = f \begin{pmatrix} w_1 \times 0.01 \\ w_2 \times 0.59 \\ w_3 \times 0.18 \\ w_4 \times 0.05 \\ w_5 \times 0.47 \\ w_6 \times 0.25 \\ w_7 \times 0.12 \\ w_8 \times 0.39 \\ w_9 \times 0.46 \\ w_{10} \times 0.26 \end{pmatrix} = f(WX)$$



# 神经元



$$h_{W,b}(x) = f(W^T x + b)$$

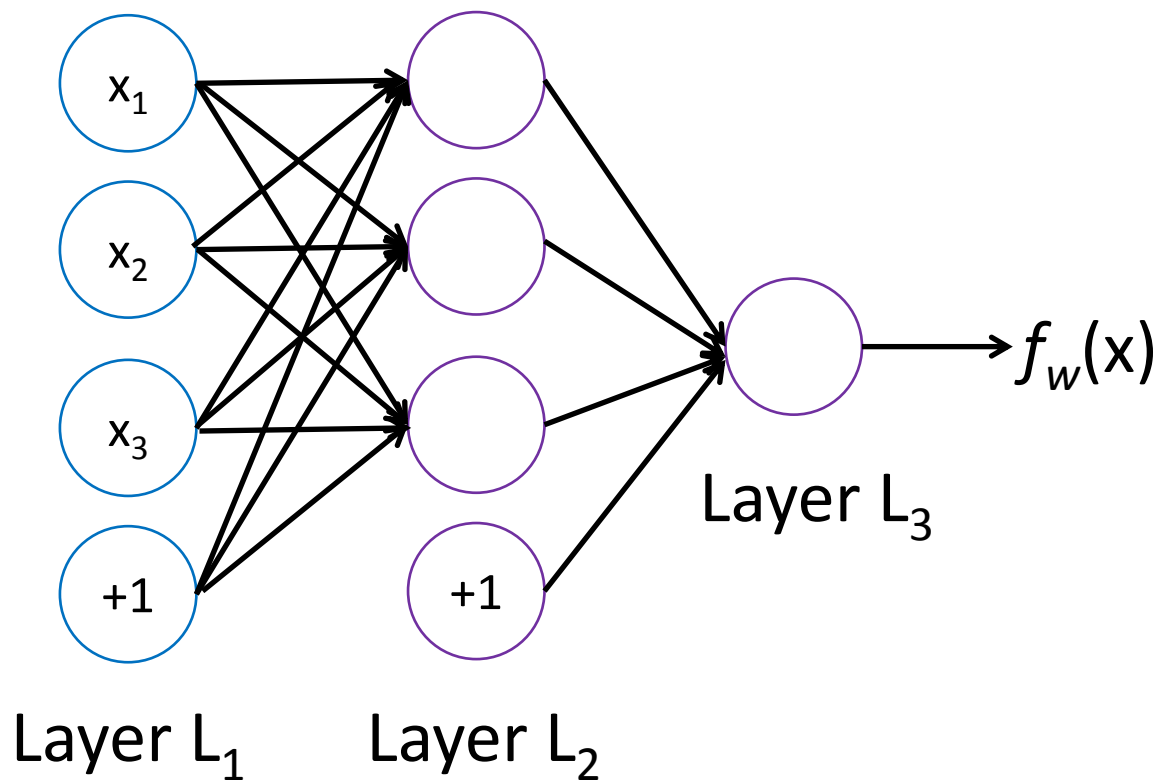
$f$ : 非线性激活函数

$$\begin{cases} f(z) = \frac{1}{1 + \exp(-z)} \\ f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \end{cases}$$

$$f'(z) = f(z)(1 - f(z))$$

$$f'(z) = 1 - f^2(z)$$

# 神经网络





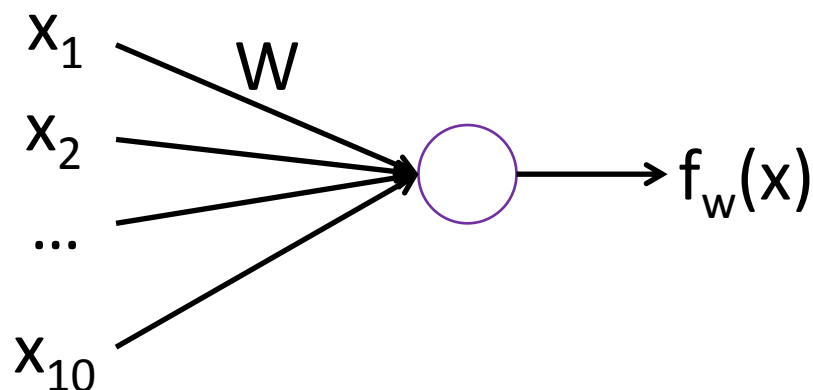
# 基于分布式表示的N-元语言模型

$P(\text{无聊}|\text{很})$

**vs.**

$P(\text{枯燥}|\text{很})$

$$f \begin{pmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.25 \\ 0.12 \\ 0.39 \\ 0.46 \\ 0.26 \end{pmatrix} = f \begin{pmatrix} w_1 \times 0.01 \\ w_2 \times 0.59 \\ w_3 \times 0.18 \\ w_4 \times 0.05 \\ w_5 \times 0.47 \\ w_6 \times 0.25 \\ w_7 \times 0.12 \\ w_8 \times 0.39 \\ w_9 \times 0.46 \\ w_{10} \times 0.26 \end{pmatrix} = f(WX)$$



问题①：词向量

如何将每个词映射到实数向量空间中的一个点

问题②： $f$ 函数的设计

设计什么样的神经网络结构模拟函数 $f$

# 词向量表示

$$L = \begin{bmatrix} \text{枯燥} & \dots & \text{单调} & \text{无聊} \end{bmatrix} \quad V$$

枯燥 ... 单调 无聊

$$D = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad D$$

枯燥  $L \in R^{D \times V}$

- 通常称为look-up table
  - 我们可以对 $L$ 右乘一个词的one-hot表示 $e$ 得到该词的低维、稠密的实数向量表达： $x = Le$

# 词向量表示

$$L = \begin{bmatrix} \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{matrix} & \dots & \begin{matrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{matrix} & \dots & \begin{matrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{matrix} \end{bmatrix} \begin{matrix} V \\ D \end{matrix} \quad L \in R^{D \times V}$$

枯燥 ... 单调 无聊

- 词表规模 $V$ 和词向量维度 $D$ 如何确定
  - $V$ 的确定：1, 训练数据中所有词；2, 频率高于某个阈值的所有词；3, 前 $V$ 个频率最高的词
  - $D$ 的确定：超参数，人工设定，一般从几十到几百

# 词向量表示

$$L = \begin{bmatrix} \text{枯燥} & \dots & \text{单调} & \text{无聊} \end{bmatrix} \quad L \in R^{D \times V}$$

Diagram illustrating the word vector matrix  $L$ . The matrix is shown as a collection of columns, each representing a word. The first column is labeled "枯燥" (boredom), the second "单调" (monotony), and the third "无聊" (boredom). The matrix is enclosed in large blue brackets, with the dimension  $D$  indicated on the right. The dimension  $V$  is indicated above the matrix. The matrix is labeled  $L \in R^{D \times V}$ .

- 如何学习  $L$ 
  - 通常先随机初始化，然后通过目标函数优化词的向量表达（e.g. 最大化语言模型似然度）

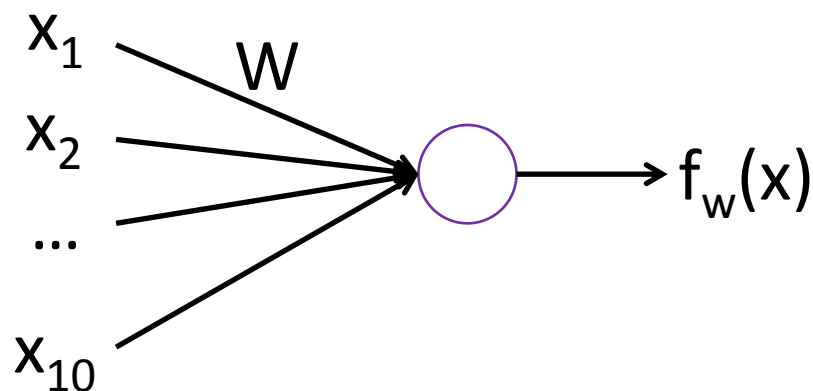
# 基于分布式表示的N-元语言模型

$P(\text{无聊}|\text{很})$

**vs.**

$P(\text{枯燥}|\text{很})$

$$f \begin{pmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.25 \\ 0.12 \\ 0.39 \\ 0.46 \\ 0.26 \end{pmatrix} = f \begin{pmatrix} w_1 \times 0.01 \\ w_2 \times 0.59 \\ w_3 \times 0.18 \\ w_4 \times 0.05 \\ w_5 \times 0.47 \\ w_6 \times 0.25 \\ w_7 \times 0.12 \\ w_8 \times 0.39 \\ w_9 \times 0.46 \\ w_{10} \times 0.26 \end{pmatrix} = f(WX)$$



问题①：词向量

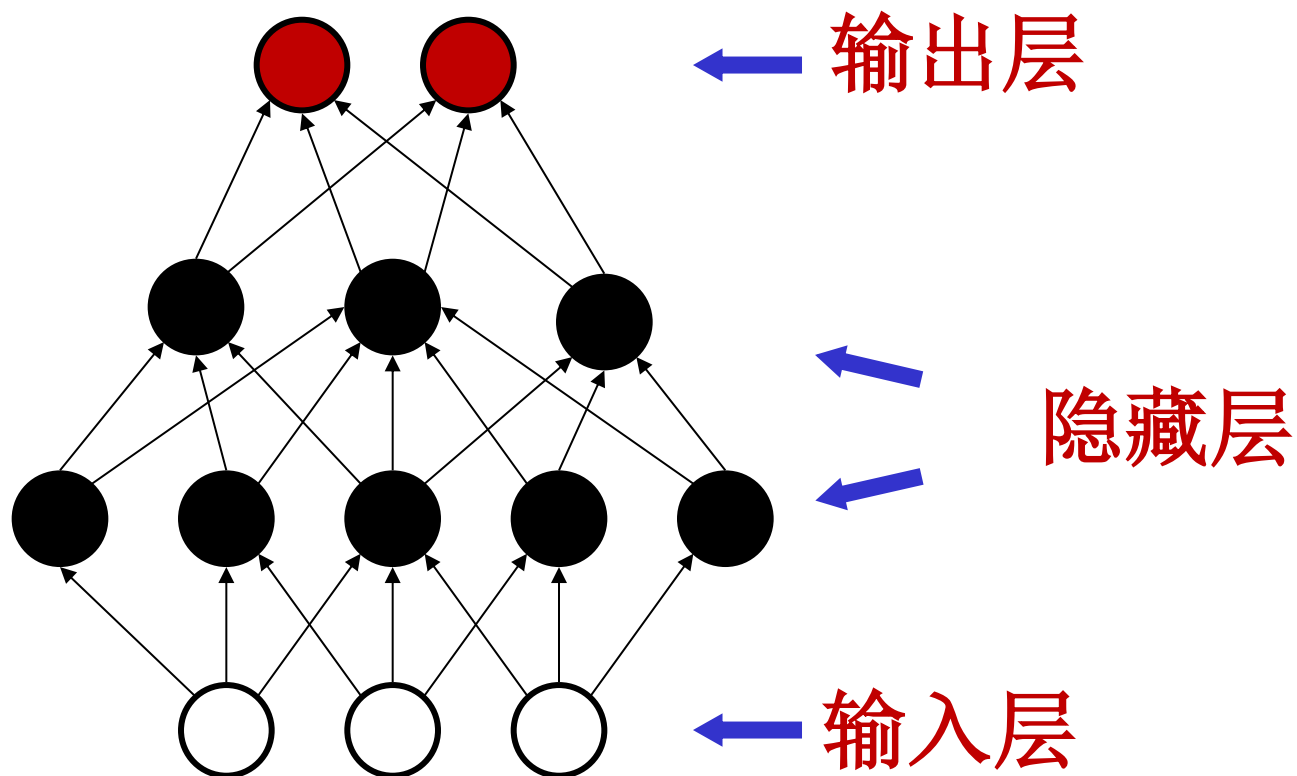
如何将每个词映射到实数向量空间中的一个点

问题②： $f$ 函数的设计

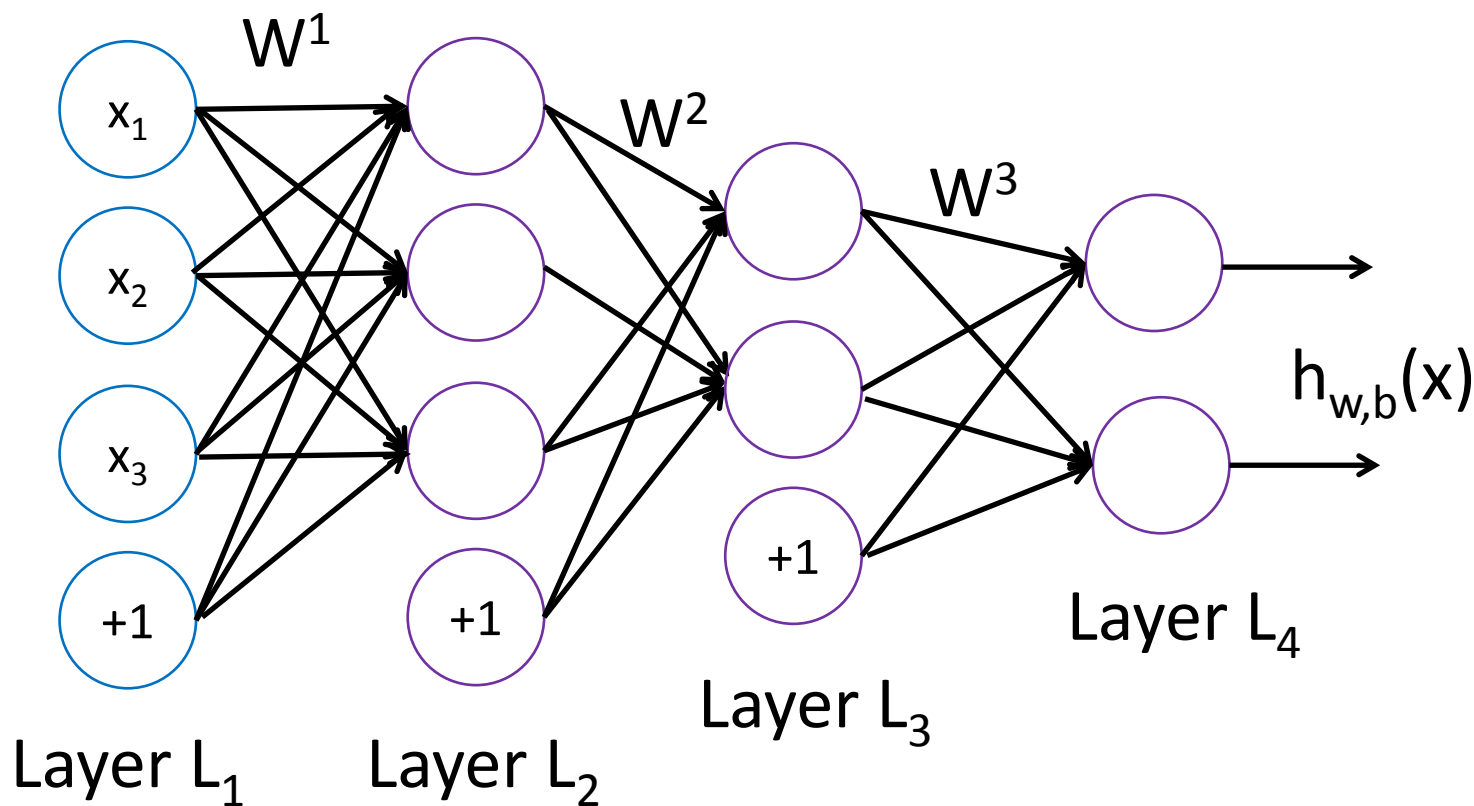
设计什么样的神经网络结构模拟函数 $f$



# 神经网络



# 前馈神经网络



# 语言模型-前馈神经网络

$i$ -th output =  $P(w_t = i | \text{context})$

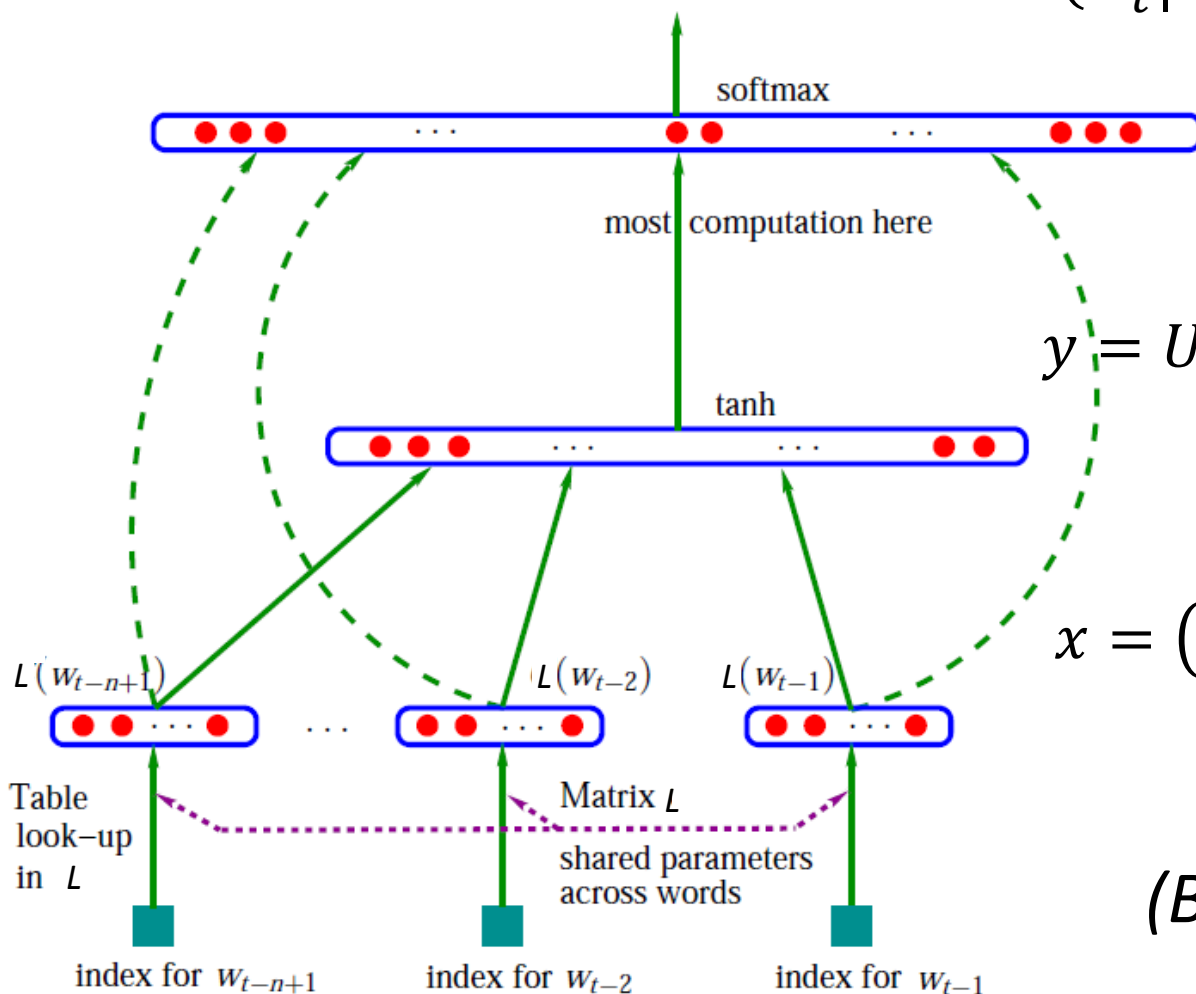
$$P(w_t | w_{t-1} \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

$$\theta \leftarrow \theta + \frac{\partial \log P}{\partial \theta}$$

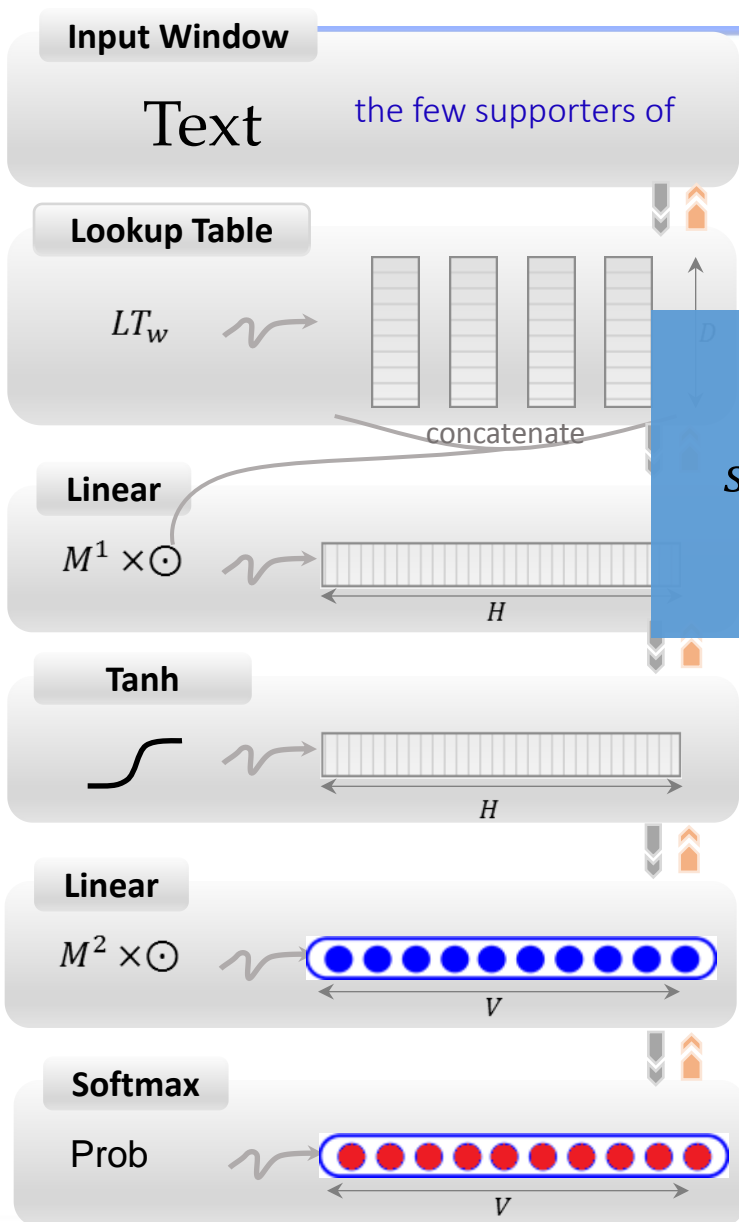
$$y = U \tanh(Hx + d) + Wx + b$$

$$x = (L(w_{t-1}), \cdots L(w_{t-n+1}))$$

(Bengio et al., 2003)



# 语言模型-前馈神经网络



$$V = \{the, few, supporters, of, this\}$$

$$P(\text{this} | the, few, supporters, of)$$

将每个词通过词向量矩阵L映射为低维实数向量

$$\text{softmax} \begin{pmatrix} h \cdot the \\ h \cdot few \\ h \cdot supporters \\ h \cdot of \\ h \cdot this \end{pmatrix} = \text{softmax} \begin{pmatrix} 0.5 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.2 \end{pmatrix}$$

线性映射+非线性变换

$$h \in R^H$$

Softmax 输出层:

$$P(\text{this} | the, few, supporters, of)$$

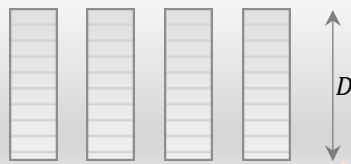
## Input Window

Text

the supporters of

## Lookup Table

$LT_w$



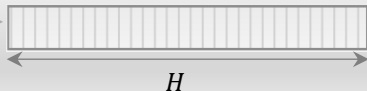
concatenate

## Linear

$M^1 \times \odot$

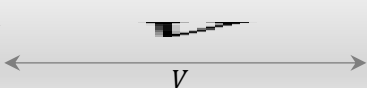


## Tanh



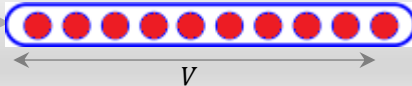
## Linear

$M^2 \times \odot$



## Softmax

Prob



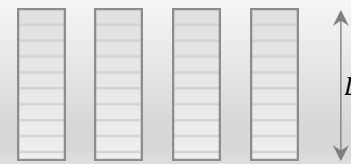
## Input Window

Text

the supporters of

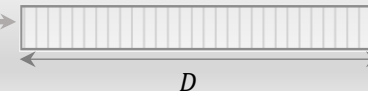
## Lookup Table

$LT_w$



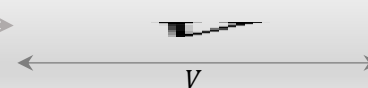
SUM

## Tanh



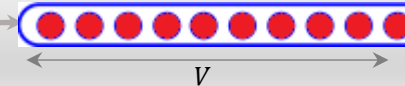
## Linear

$M^2 \times \odot$



## Softmax

Prob



VS.



# 语言模型-前馈神经网络

- 问题

仅对小窗口的历史信息建模

例如5-gram语言模型，仅考虑前面4个词的历史信息



能否对所有的历史信息进行建模

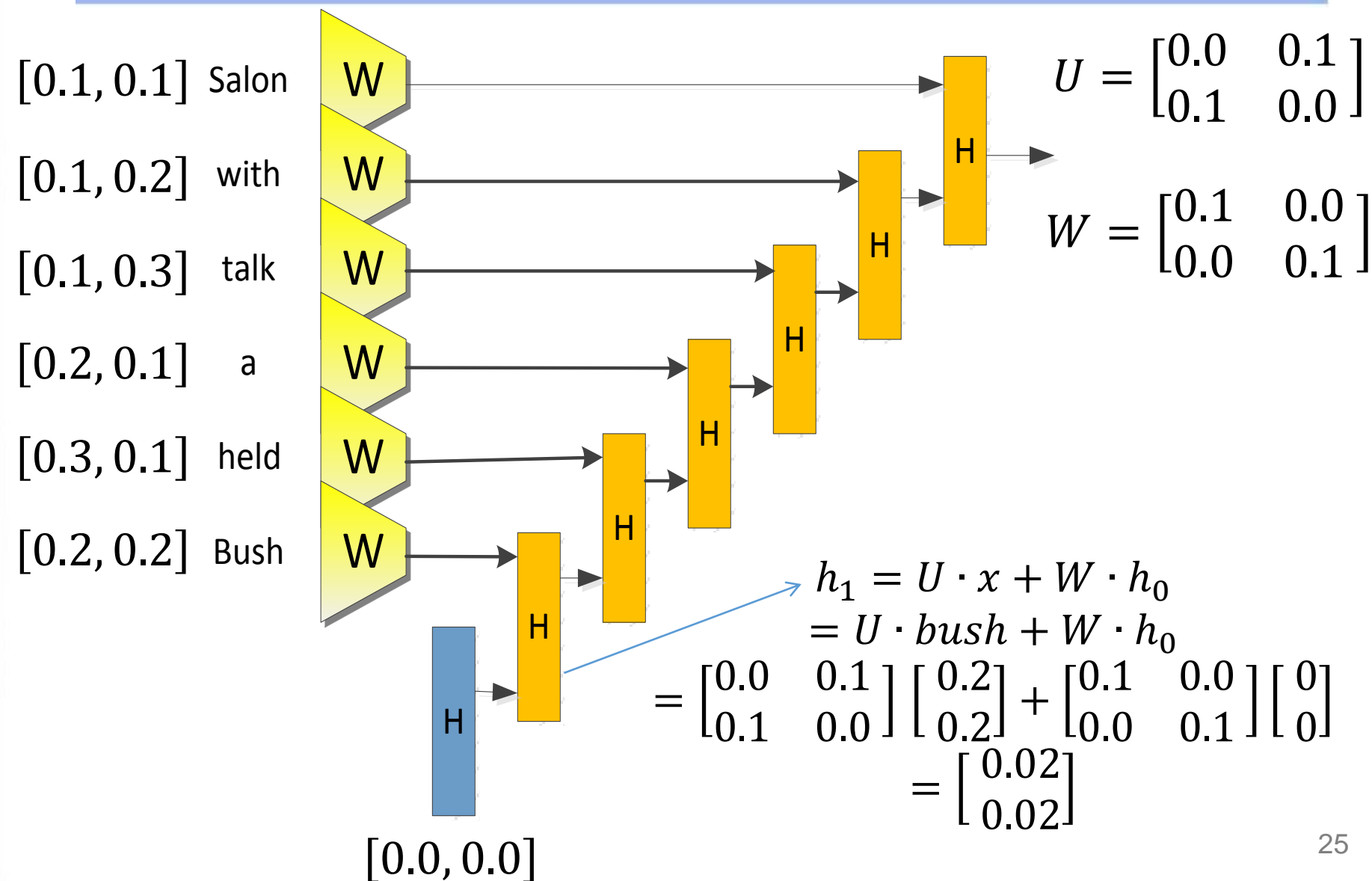
即第t个词的语言模型概率依赖于所有前t-1个词

$$P(w_t | w_{t-1} \cdots w_{t-n+1})$$

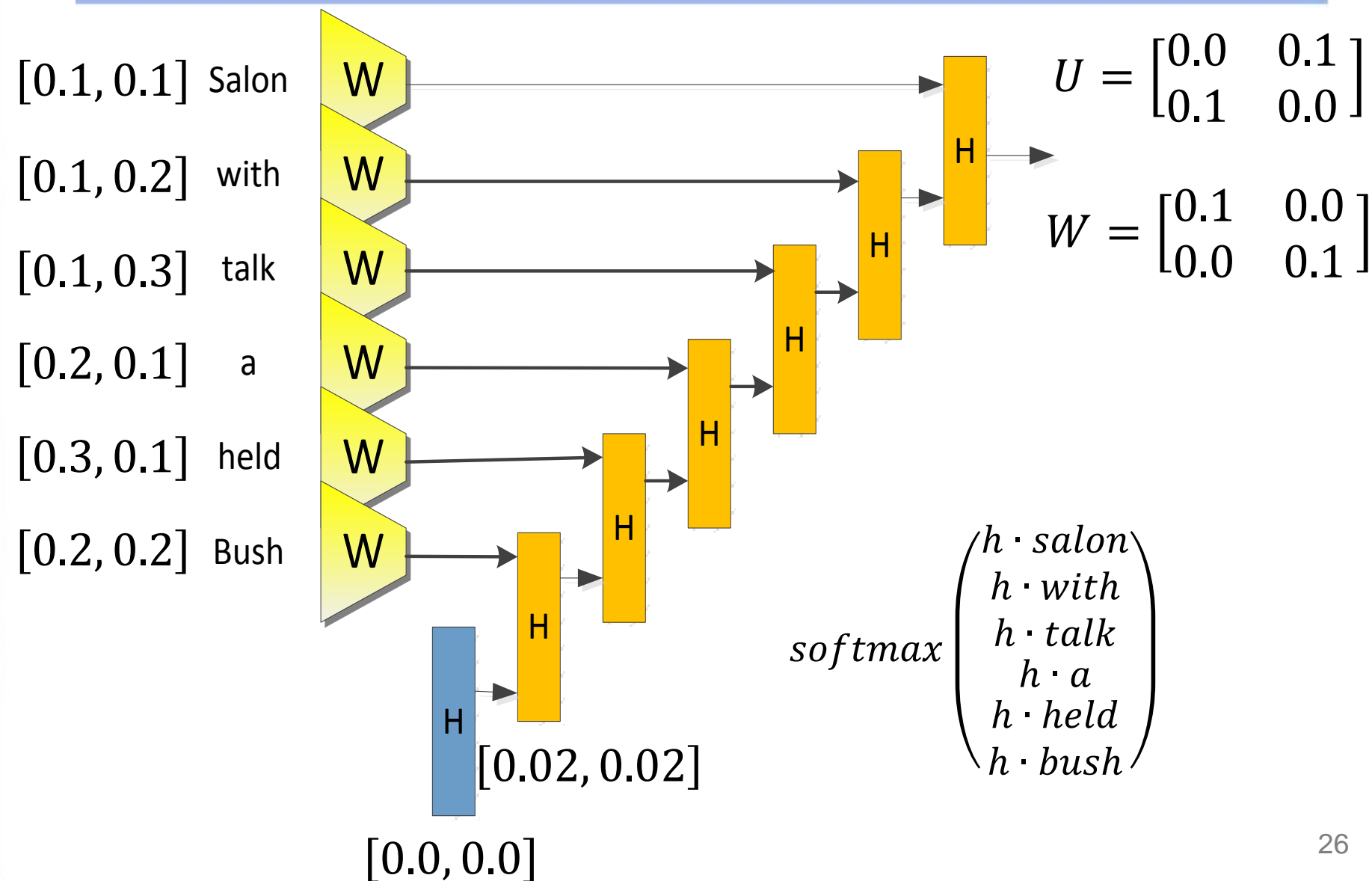


$$P(w_t | w_{t-1} \cdots w_2 w_1)$$

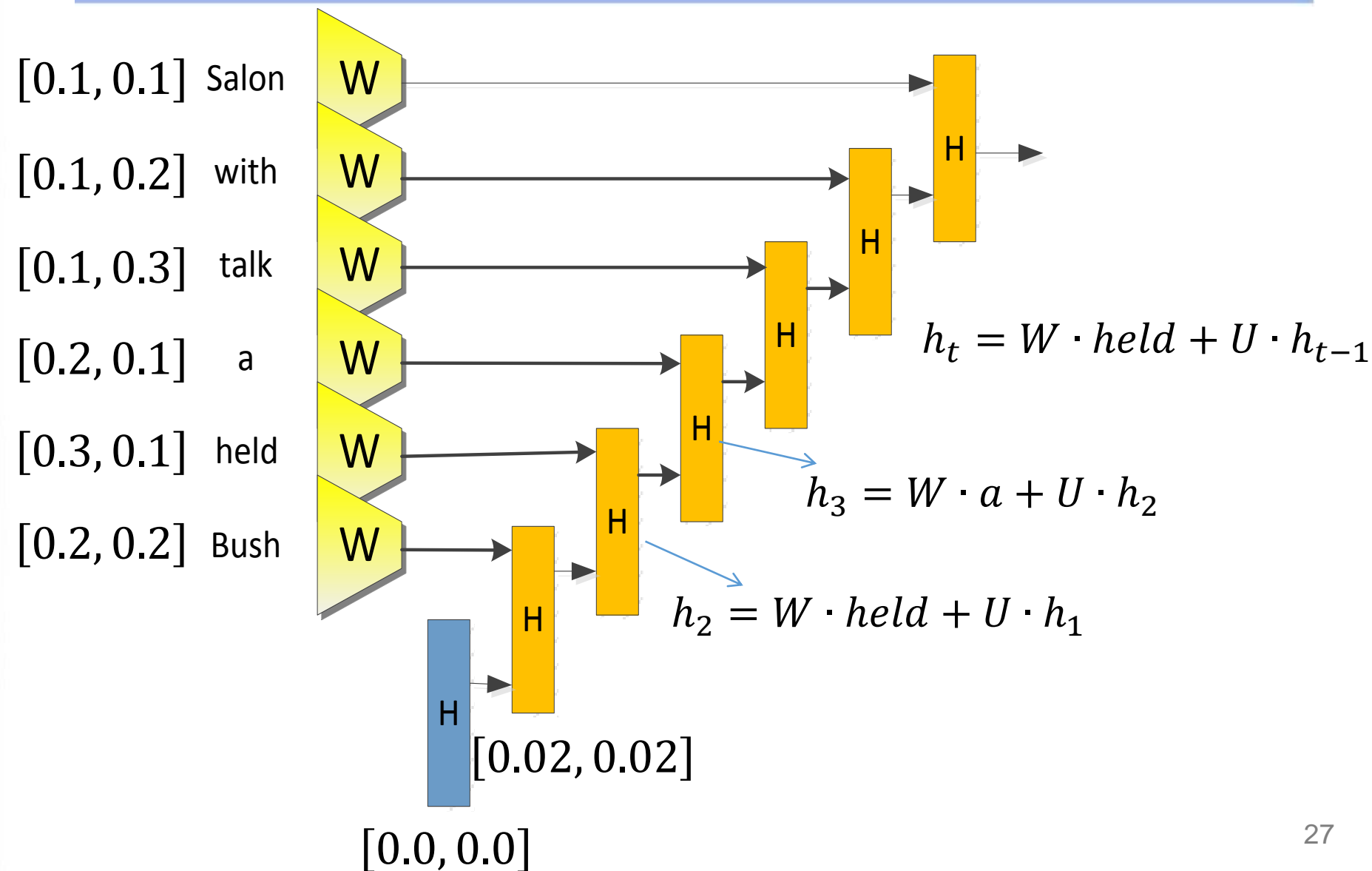
# 语言模型-循环神经网络



# 语言模型-循环神经网络

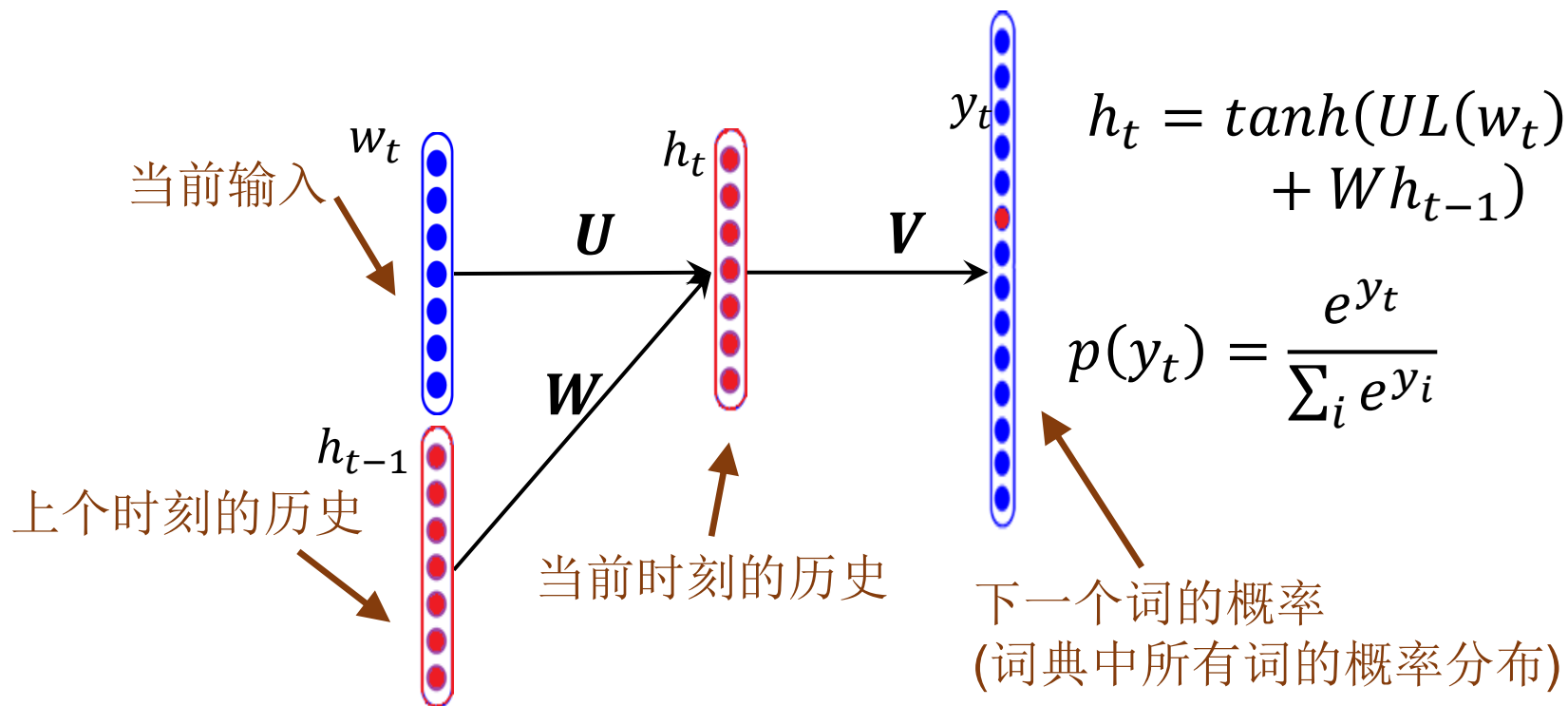


# 语言模型-循环神经网络



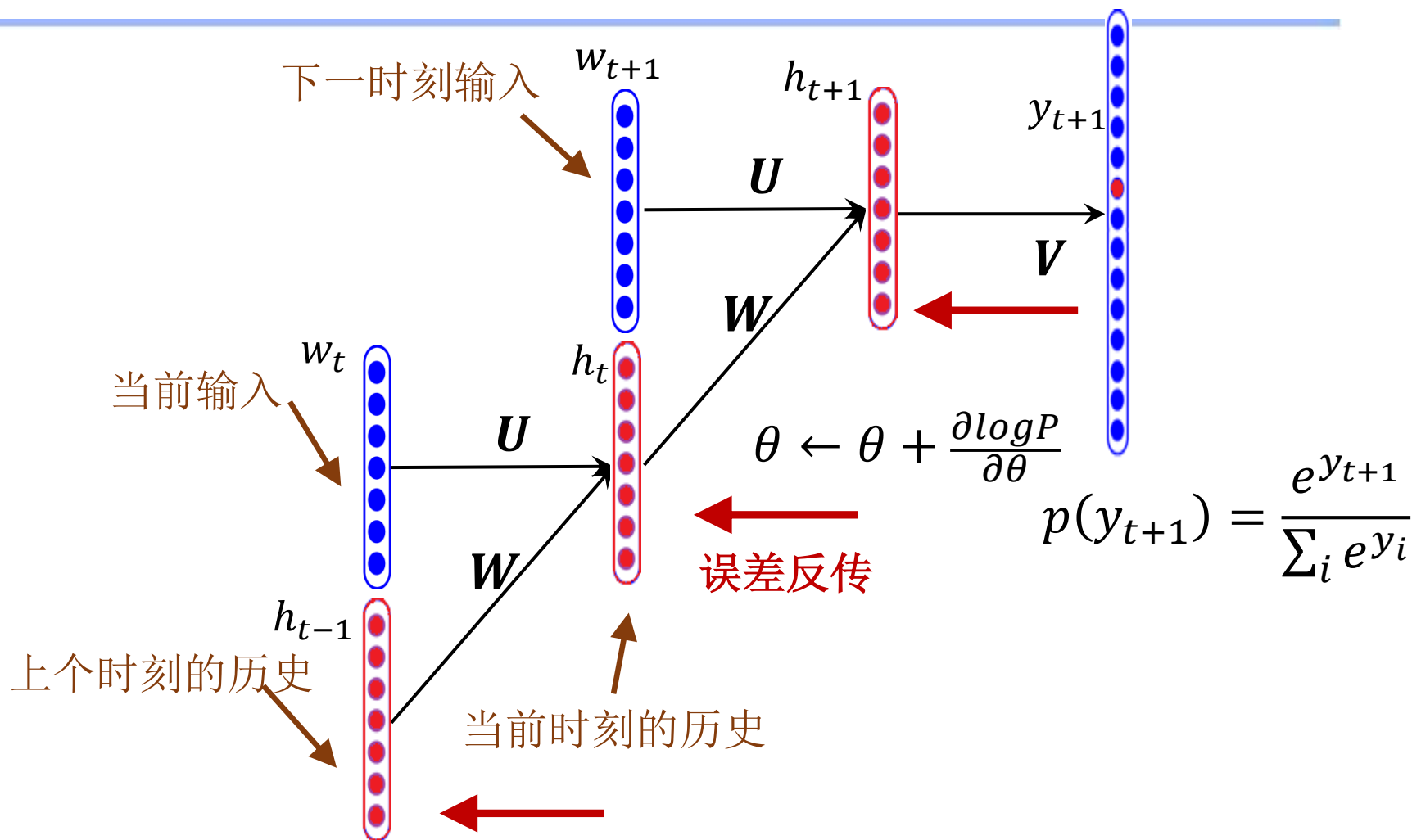
# 语言模型-循环神经网络

- 输入:  $t - 1$ 时刻历史  $h_{t-1}$  与  $t$ 时刻输入  $w_t$
- 输出:  $t$ 时刻历史  $h_t$  与 下个时刻  $t + 1$ 输入  $y_t$  的概率

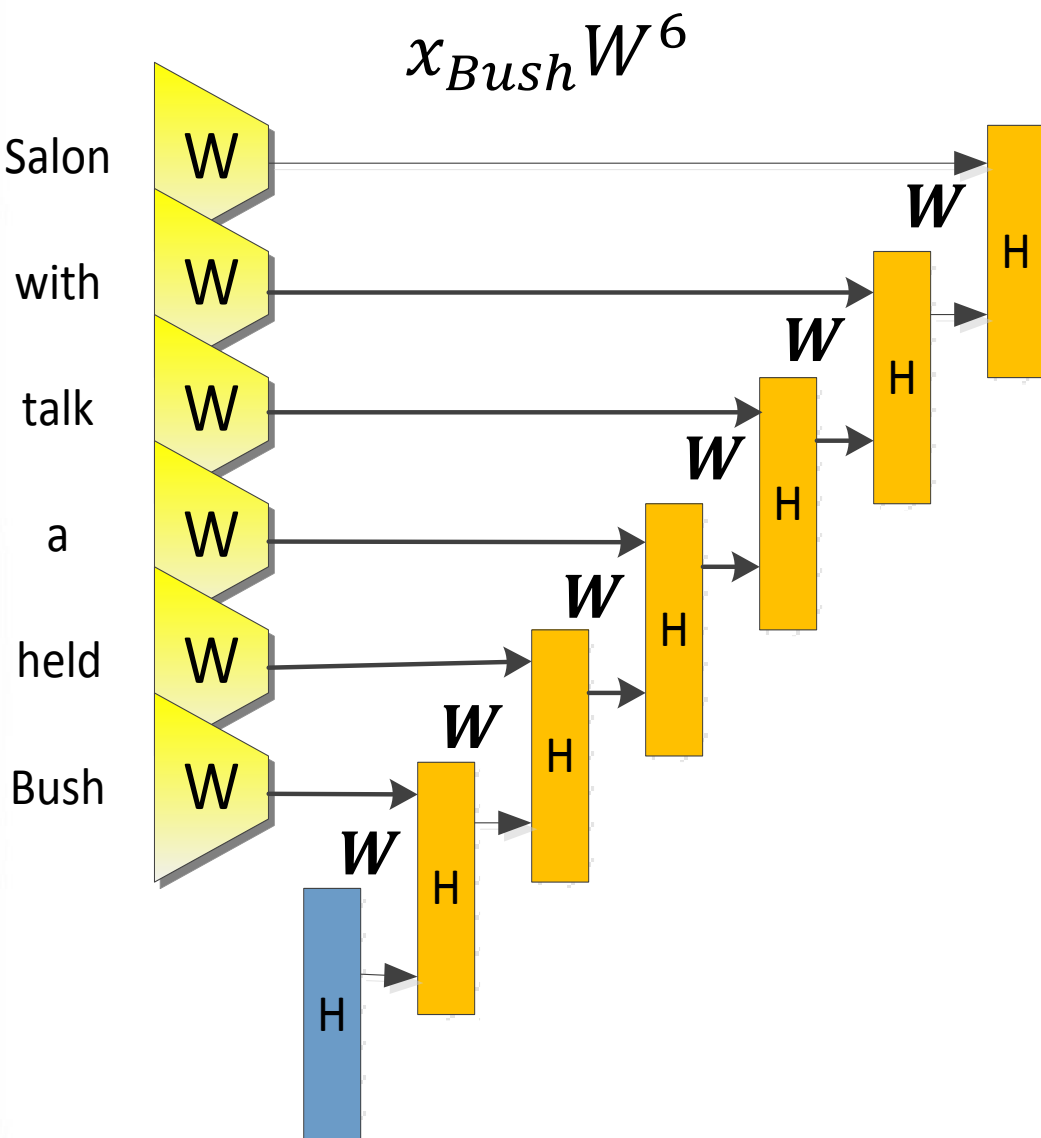




# 语言模型-循环神经网络



# 语言模型-循环神经网络

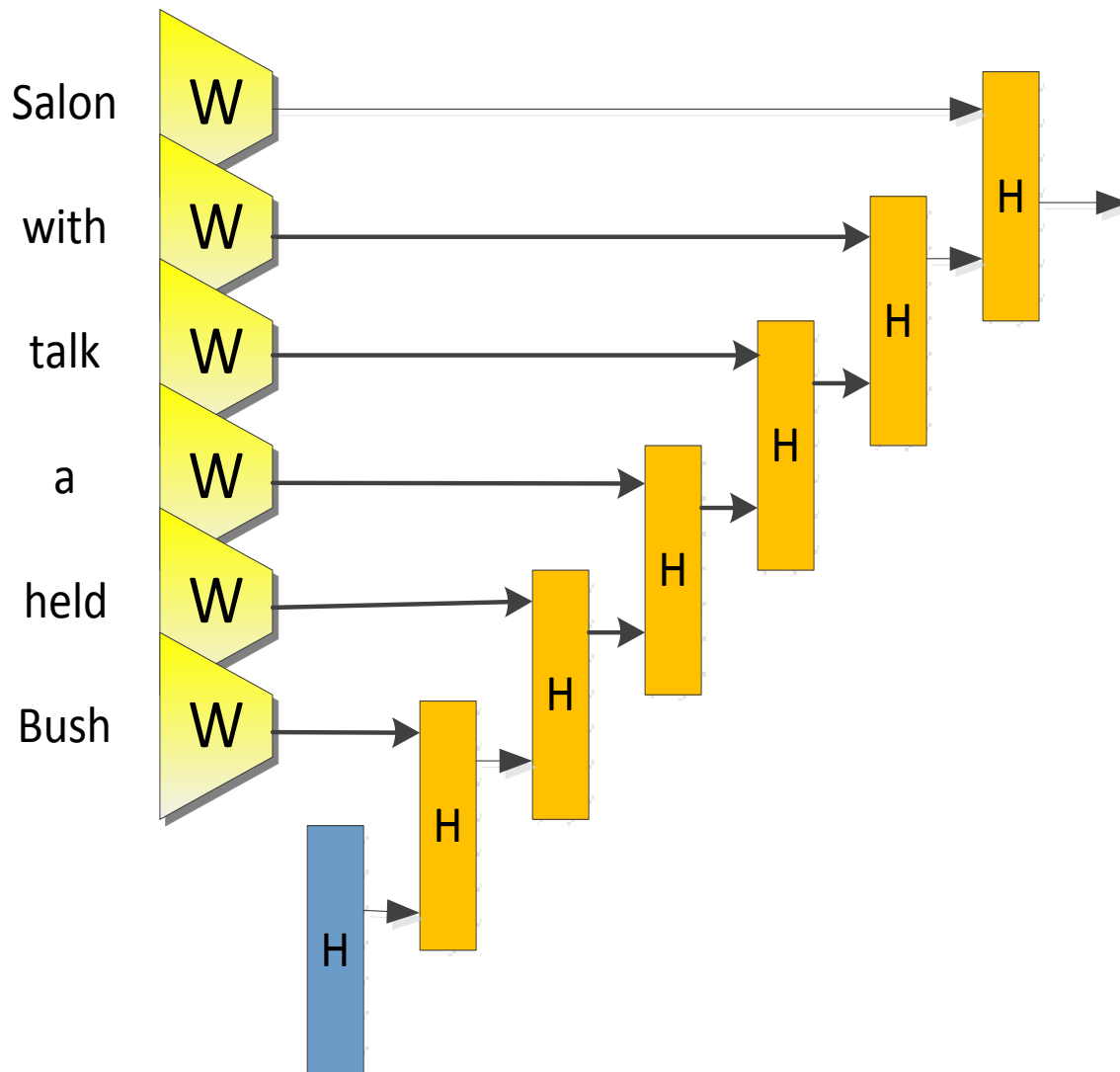


## • 问题

**梯度消失和爆炸**  
 参数  $W$  经过多次传递后，易发生梯度消失 ( $<1$ ) 或爆炸 ( $>1$ )

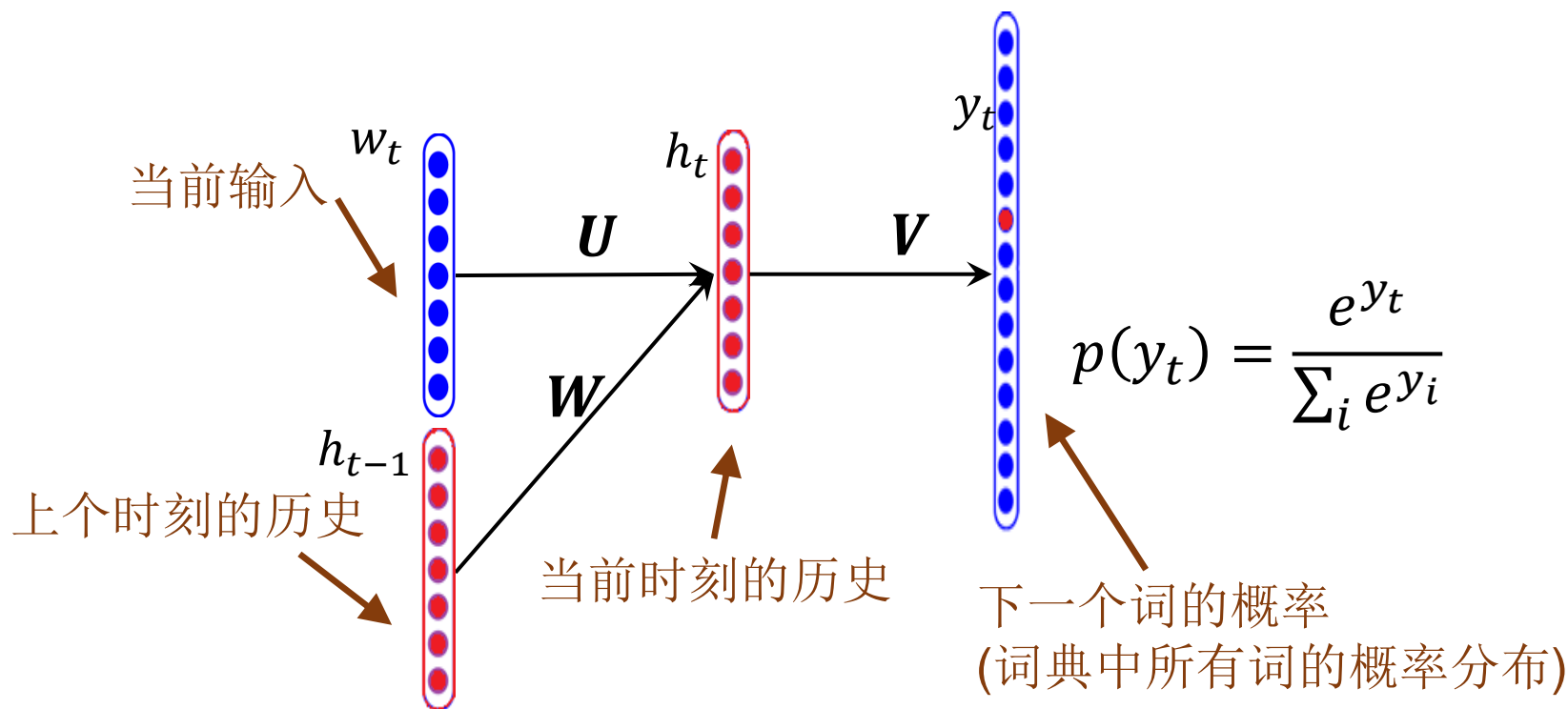
有选择地保留和遗忘  
 通过某种策略有选择地保留或者遗忘  $t$  时刻的信息

# 语言模型-长短时记忆网络(LSTM)

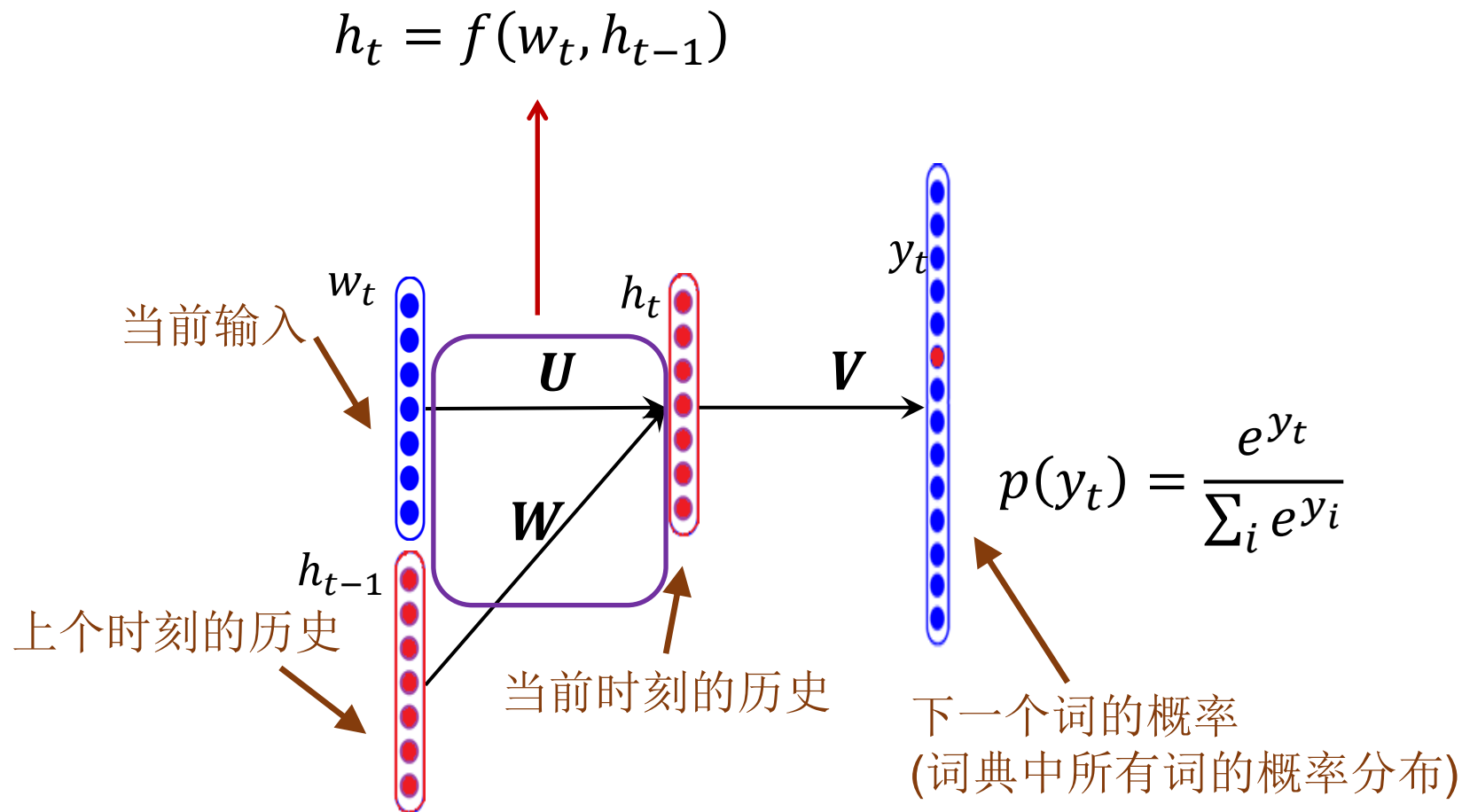


# 循环神经网络

$$h_t = \tanh(UL(w_t) + Wh_{t-1})$$



# LSTM

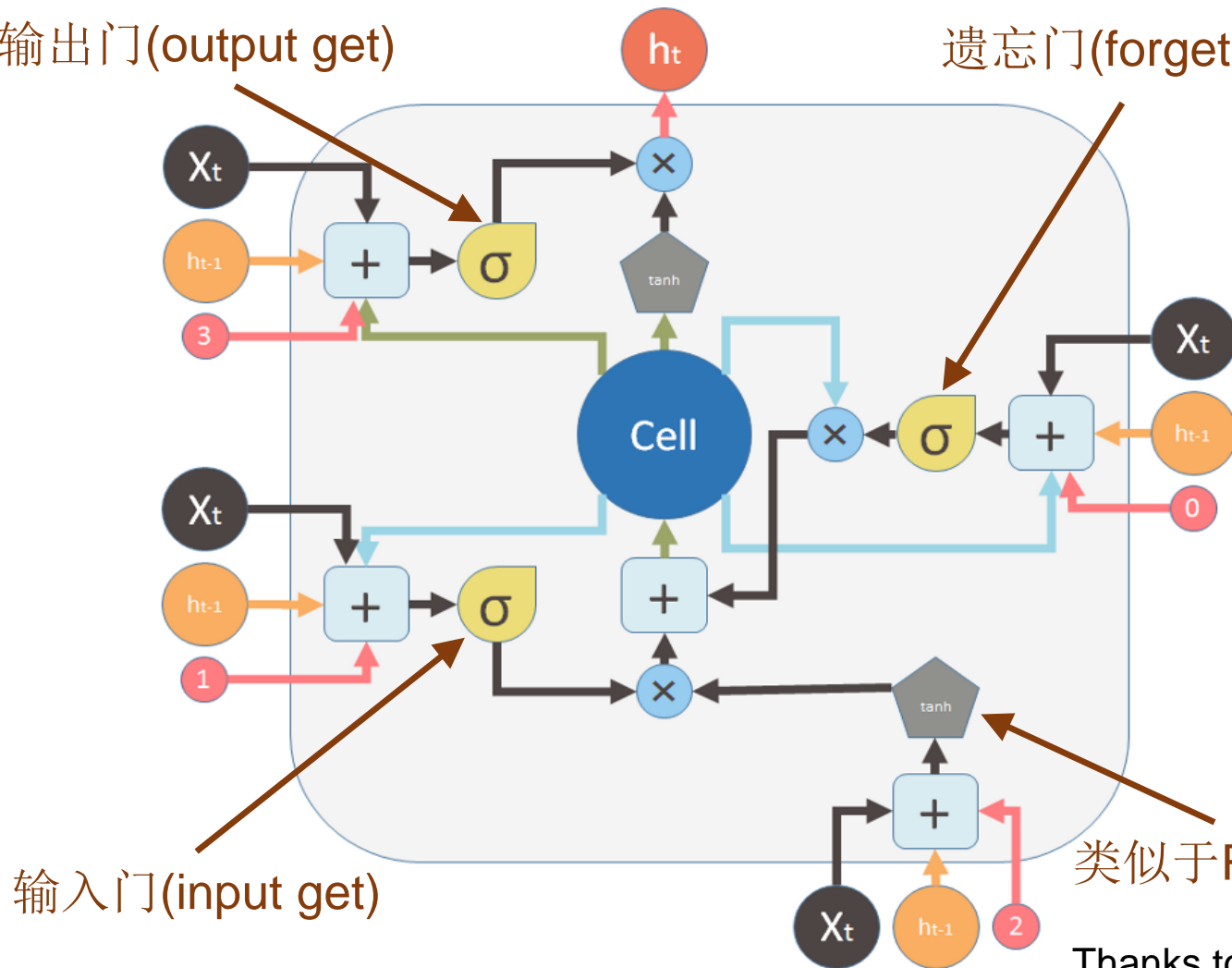


# LSTM

$$h_t = f(w_t, h_{t-1})$$

输出门(output get)

遗忘门(forget get)

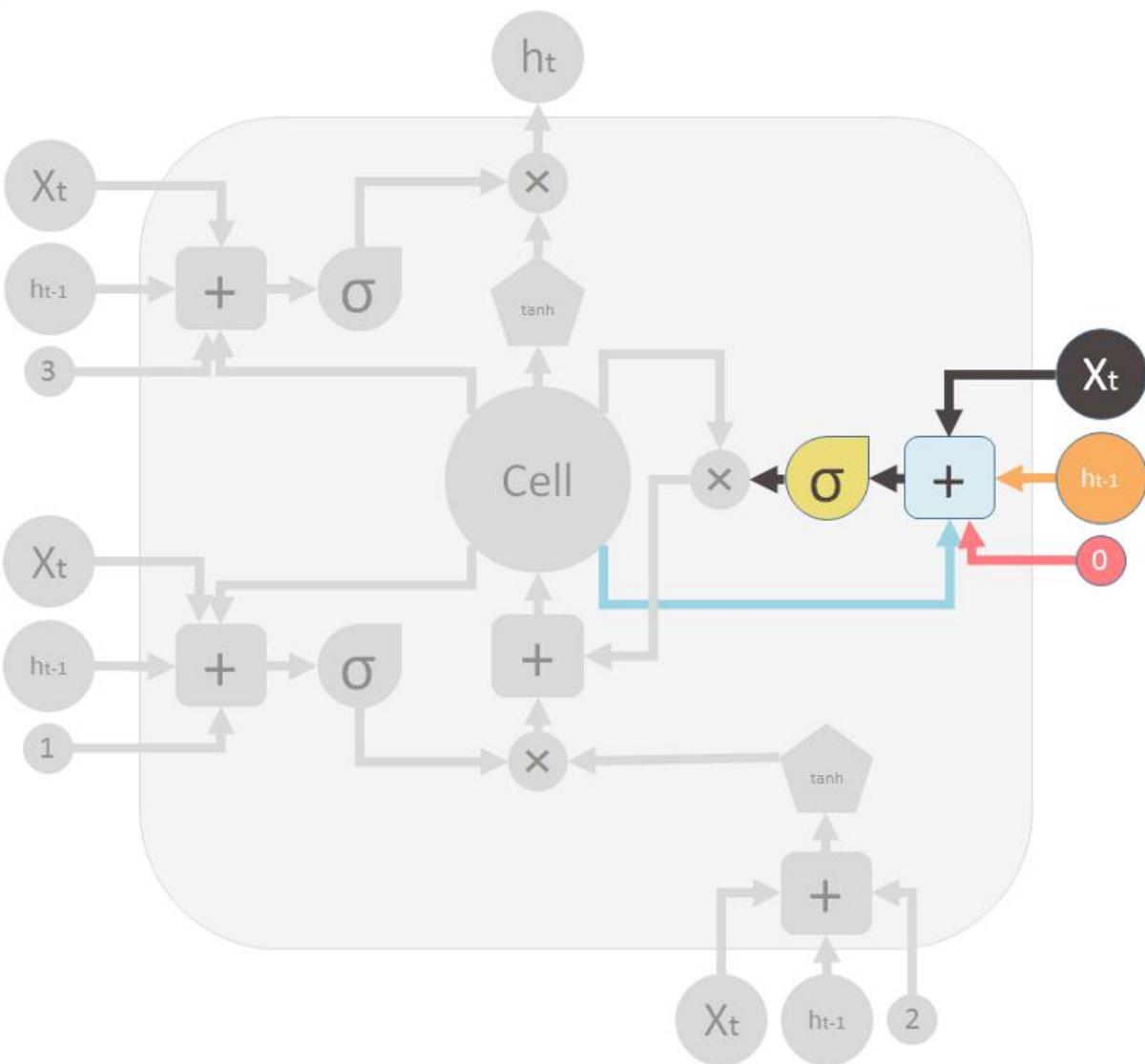


输入门(input get)

类似于RNN中 $h_t$ 的计算

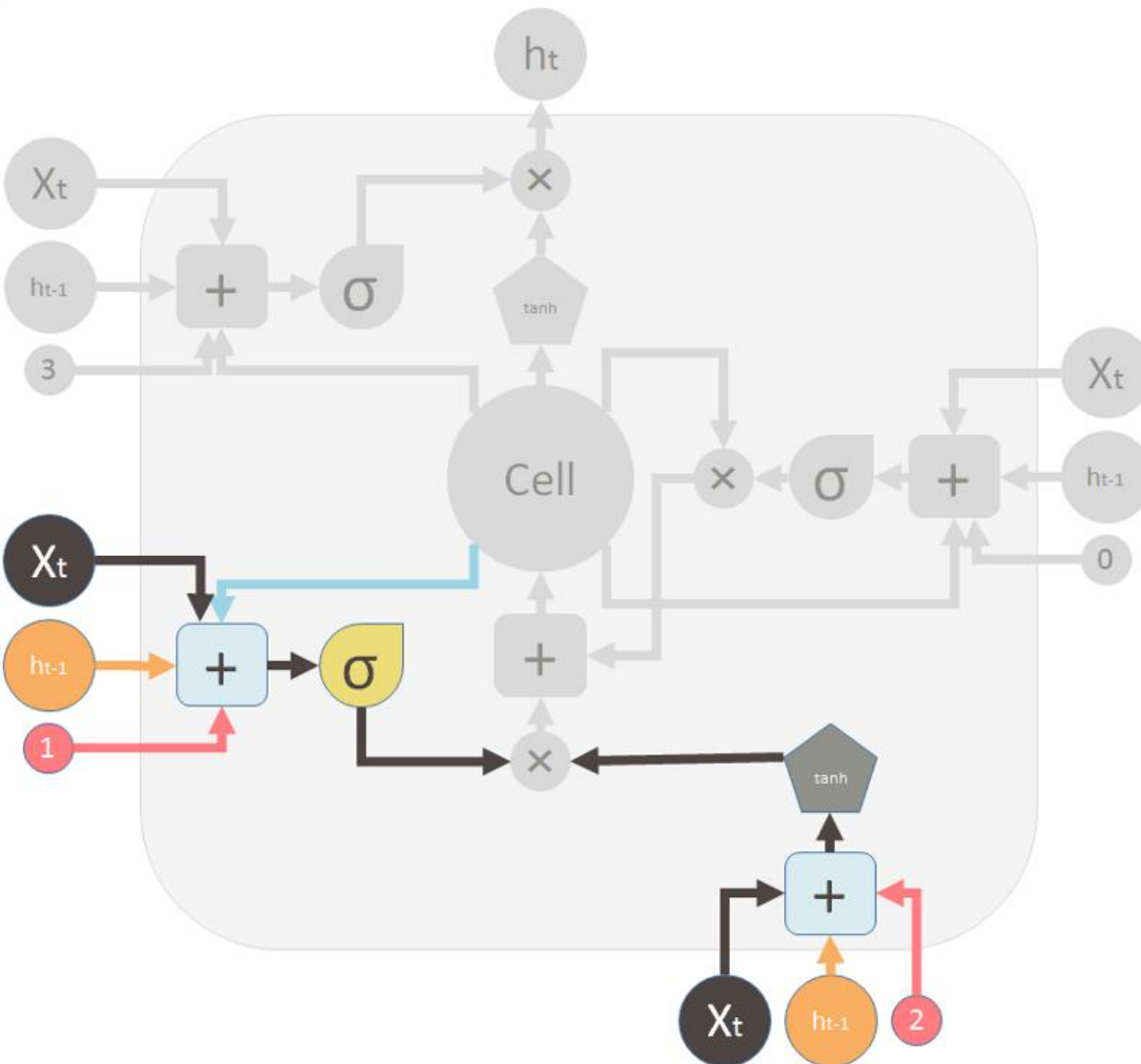
Thanks to Shi Yan!

# LSTM



$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1})$$

# LSTM

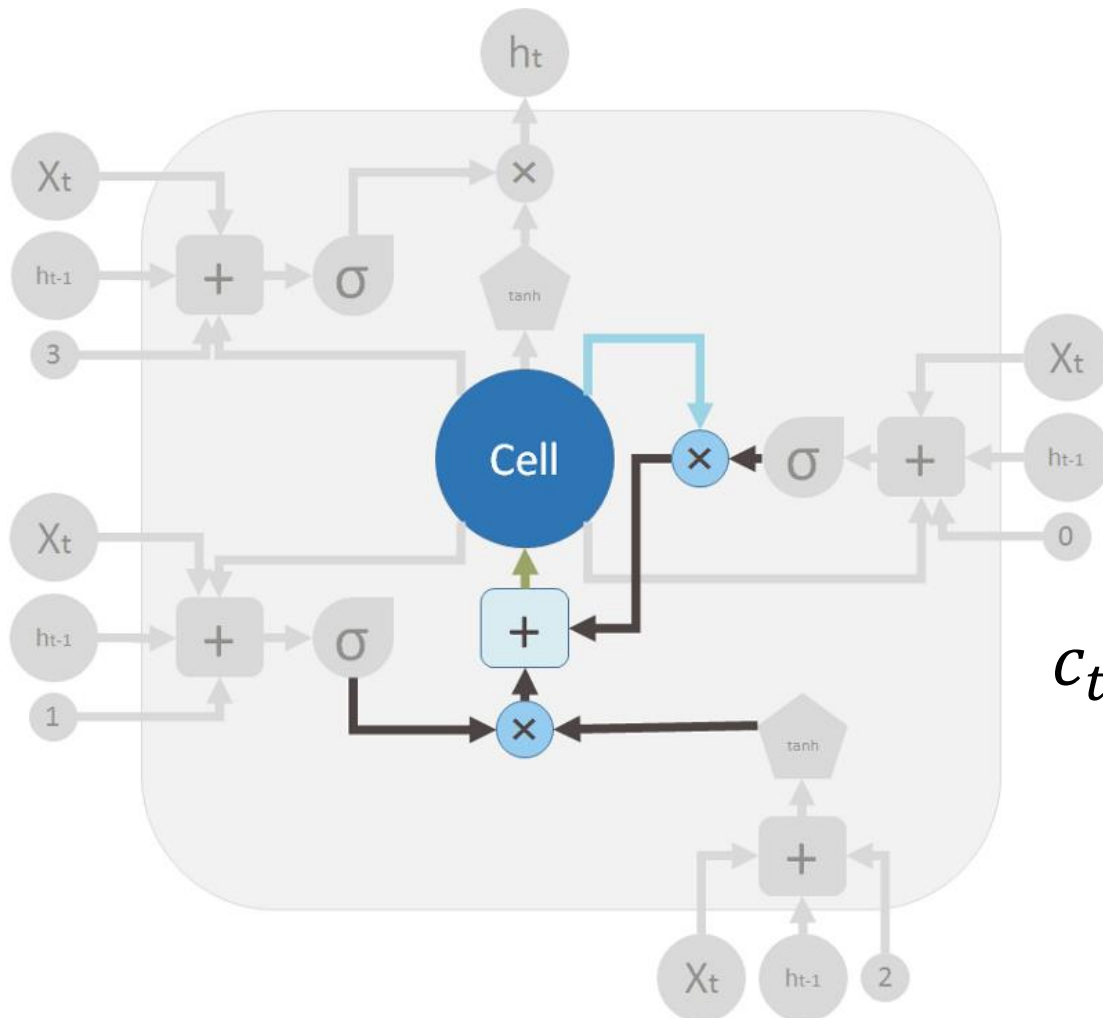


$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1})$$

$$\tilde{h}_t = \tanh(W_x x_t + W_h h_{t-1})$$

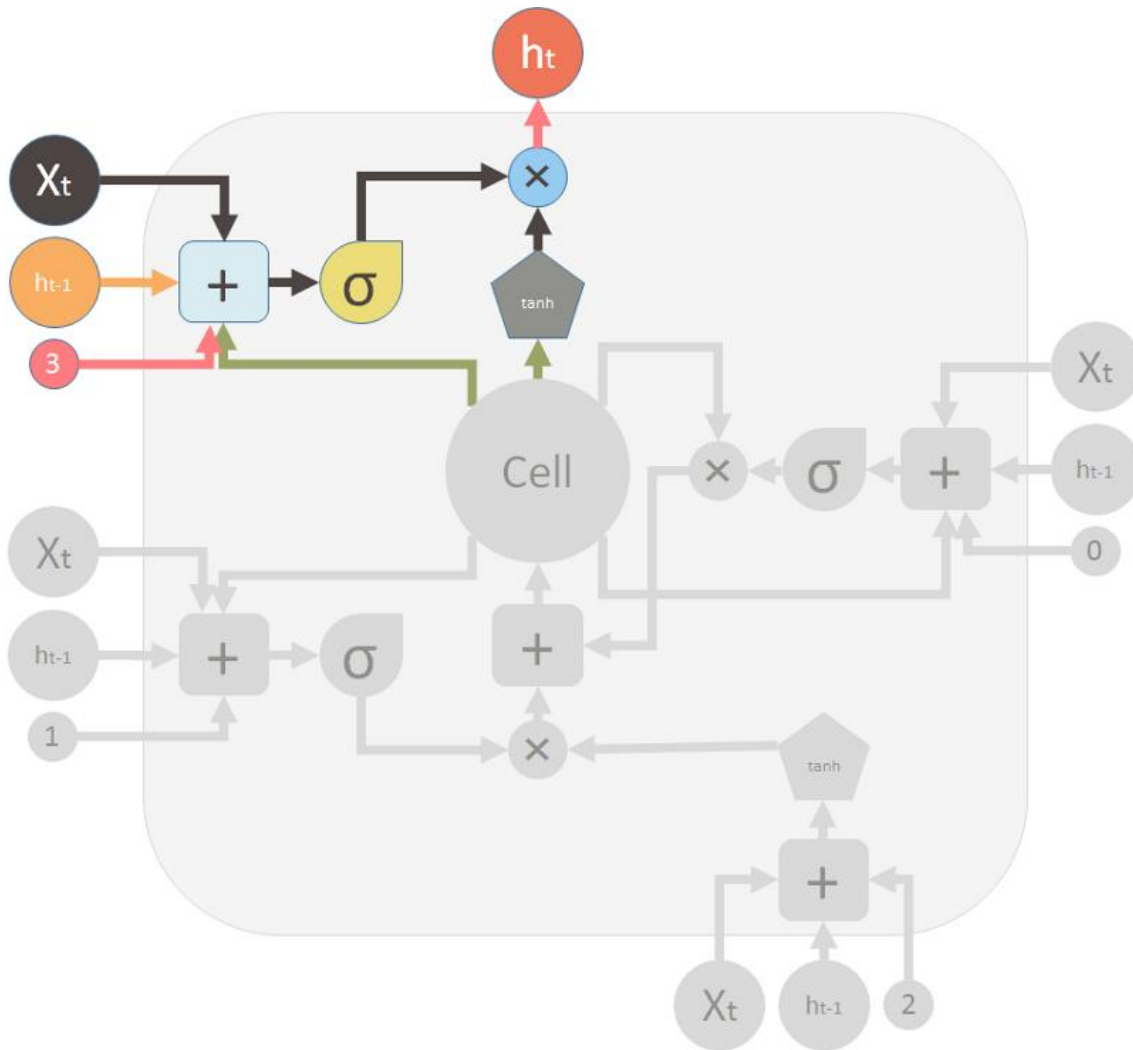


# LSTM



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

# LSTM



$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1})$$

$$h_t = o_t \tanh(c_t)$$

# 词向量规模V

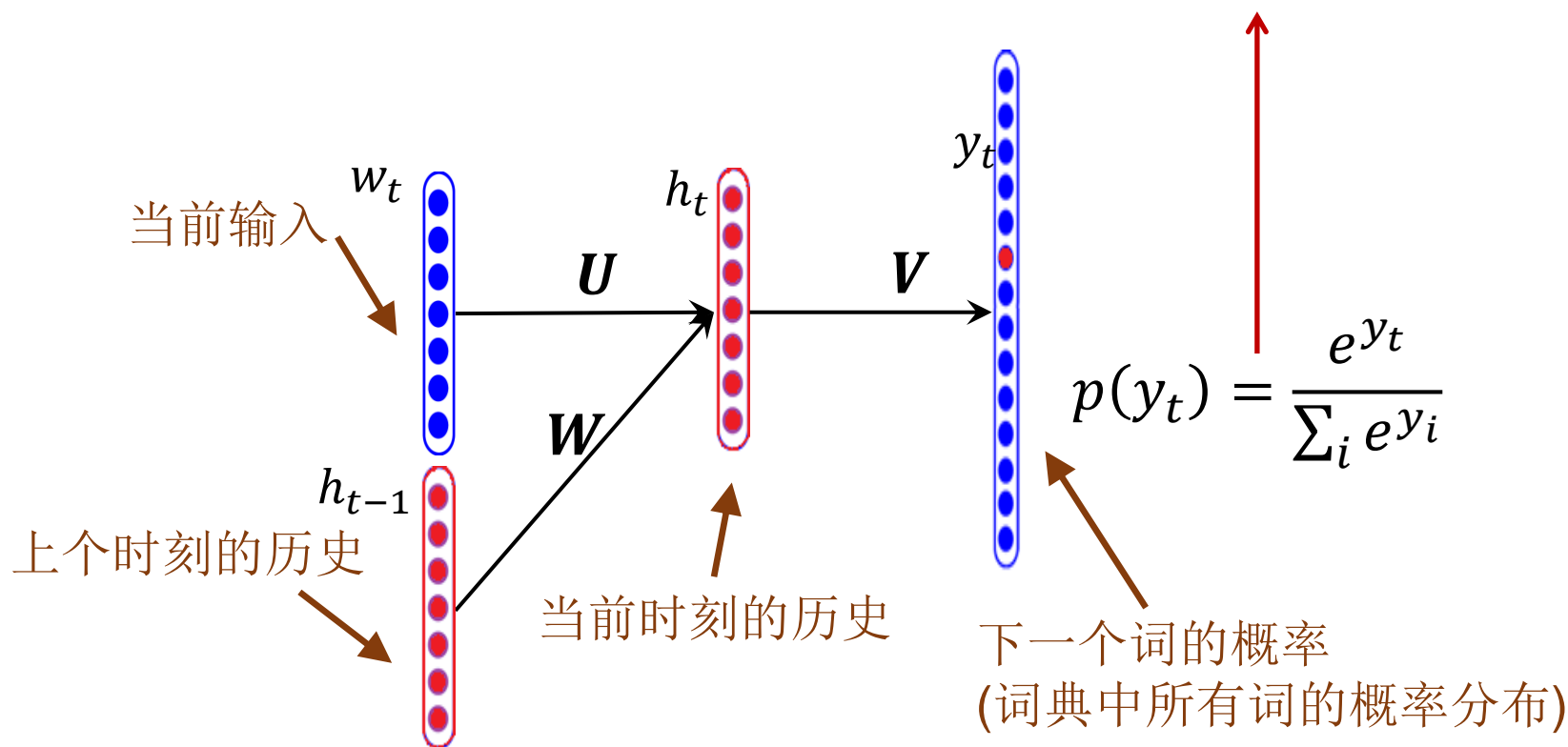
$$L = \begin{bmatrix} \text{枯燥} & \dots & \text{单调} & \text{无聊} \end{bmatrix} \quad L \in R^{D \times V}$$

The diagram illustrates a word embedding matrix  $L$  of size  $D \times V$ . The matrix is represented as a grid of red dots. The first column is labeled '枯燥' (boring), the second column is labeled '单调' (monotonous), and the third column is labeled '无聊' (boredom). The matrix is enclosed in large blue brackets, with the dimension  $D$  indicated on the right. The dimension  $V$  is indicated above the matrix. The matrix is labeled  $L \in R^{D \times V}$  on the right.

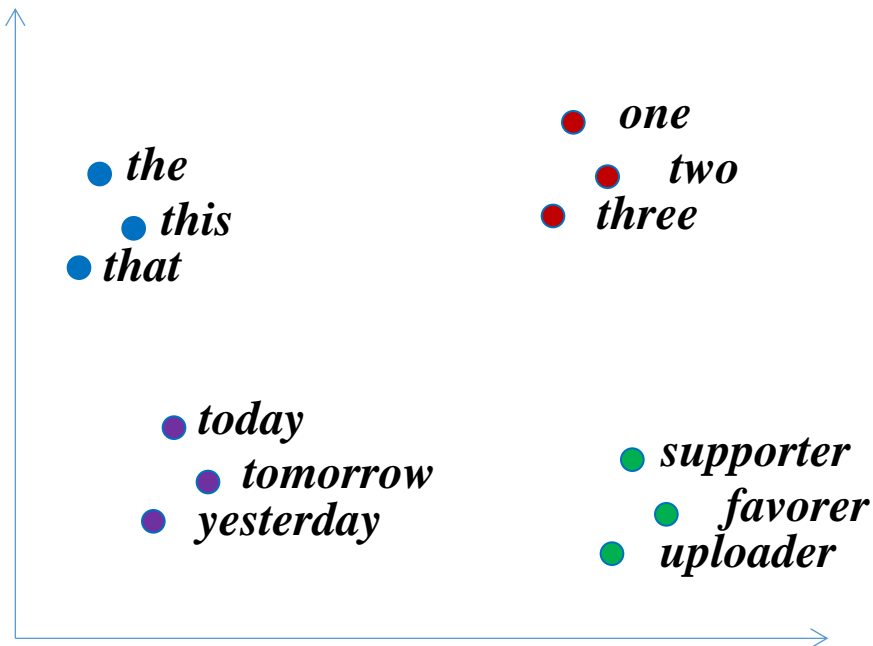
- 词表规模V的确定
  - V的确定：1, 训练数据中所有词；2, 频率高于某个阈值的所有词；**3, 前V个频率最高的词**
  - **e.g.  $V=50000$ ,  $V=80000$**

# 词向量规模V

决定了模型的复杂度  $\leftarrow \sum_i e^{y_i}$  遍历词表中的所有词



# 词向量分布



低维、稠密的实数向量空间

在低维、稠密的实数向量空间中，相似的词聚集在一起，在相同的历史上下文中具有相似的概率分布！

# 开源工具

1, NNlm, 前馈神经网络语言模型 (feed-forward n-gram neural language model) ,  
<http://nlg.isi.edu/software/nplm/>

2, RNNlm, 循环神经网络语言模型 (recurrent neural language model) ,  
<http://rnnlm.org/>;

2, LSTMlm, LSTM语言模型 (recurrent neural language model with LSTM unit) ,  
<https://www-i6.informatik.rwth-aachen.de/web/Software/rwthlm.php>

3, LSTM反向传播算法, <http://arunmallya.github.io/writeups/nn/lstm/index.html#/>

4, Google Word2Vec, <http://code.google.com/p/word2vec/>

... ..

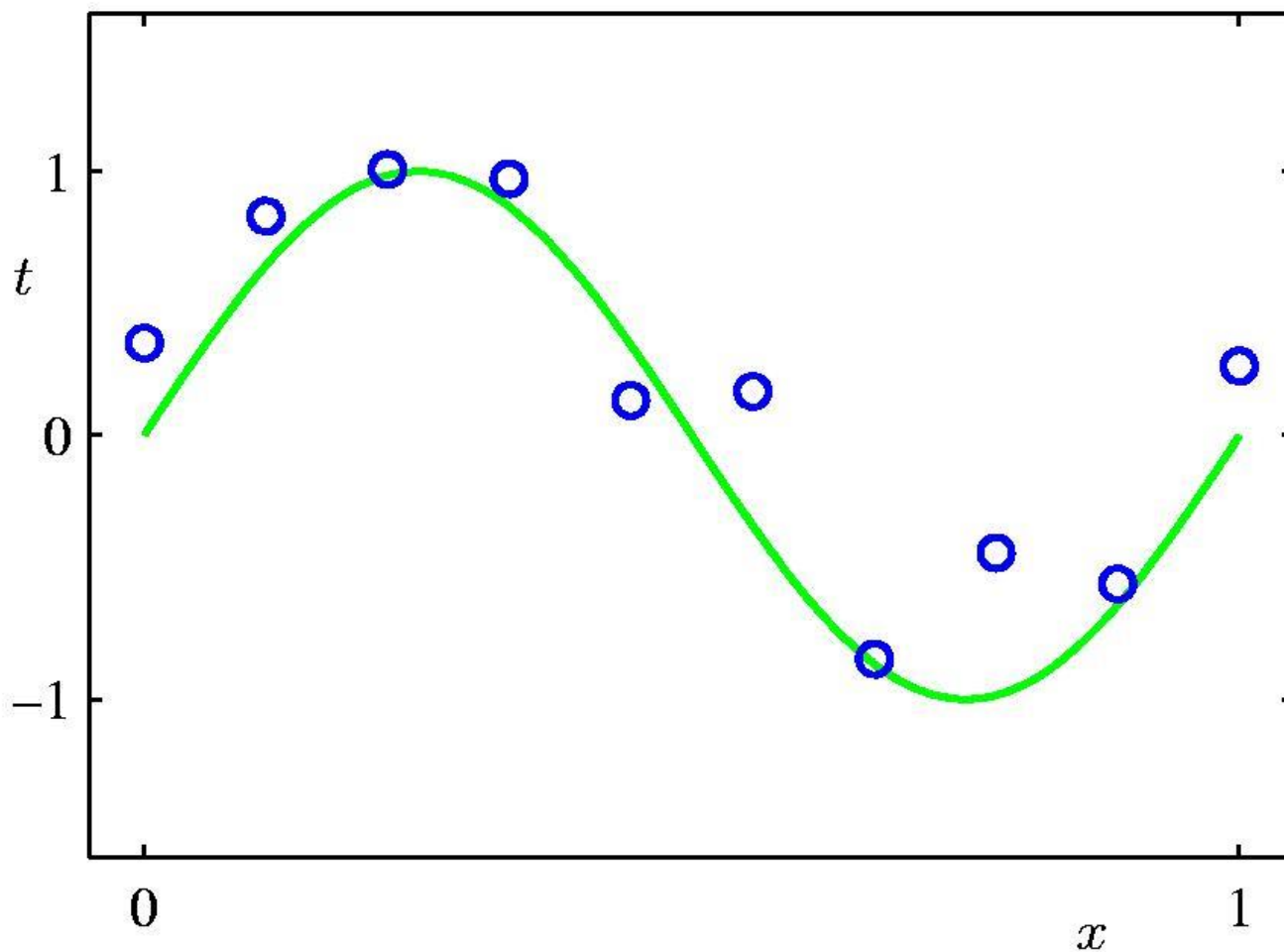
N L P R



谢谢!  
Q&A

# 参数学习：反向传播算法

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$





# 参数学习：反向传播算法

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$h_{W,b}(x) = f(W^T x + b)$$



如何学习每一层的参数  $W, b$ ?



定义损失函数

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

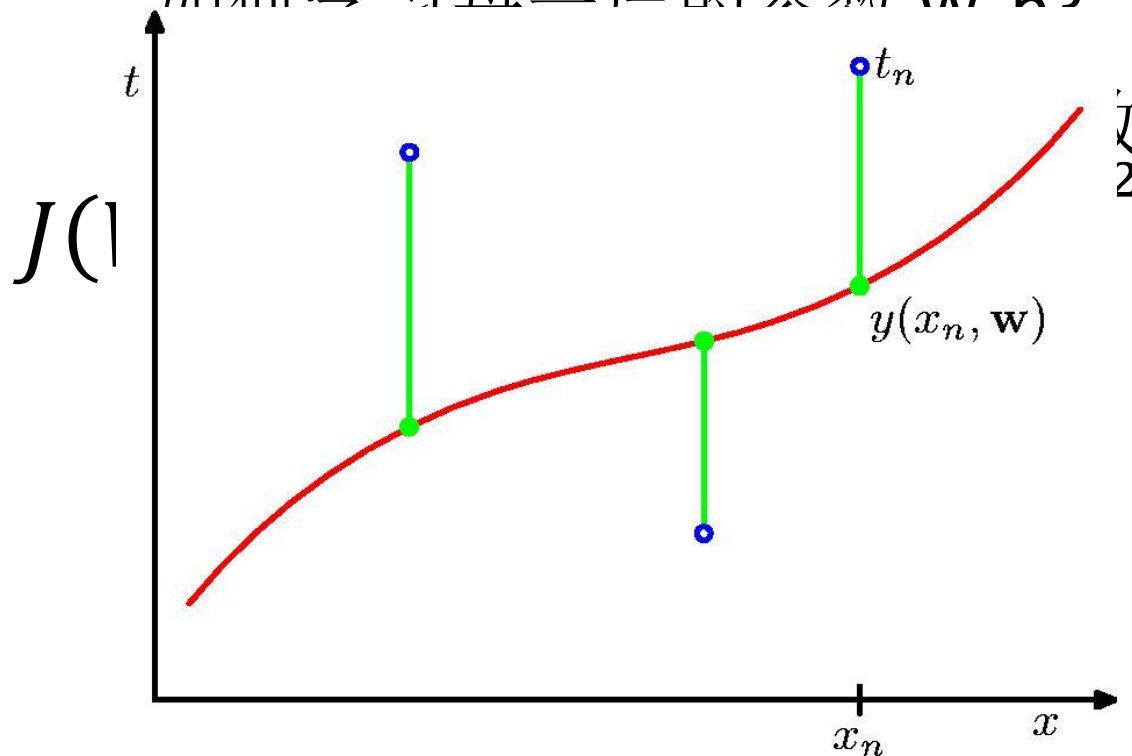
# 参数学习：反向传播算法

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$h_{W,b}(x) = f(W^T x + b)$$



如何学习每一层的参数  $w, b$



# 参数学习：反向传播算法

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$$h_{W,b}(x) = f(W^T x + b)$$



如何学习每一层的参数  $W, b$ ?



定义损失函数

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

$$J(W, b) = \left[ \frac{1}{m} \sum_{k=1}^m J(W, b; x^{(k)}, y^{(k)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{j=1}^{s_l} \sum_{i=1}^{s_{l+1}} \left( W_{ij}^{(l)} \right)^2$$

# 参数学习：反向传播算法

$$W, b = \operatorname{argmin} J(W, b)$$



Stochastic Gradient Descent Algorithm (随机梯度下降)

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

# 参数学习：反向传播算法

$$J(W, b) = \left[ \frac{1}{m} \sum_{k=1}^m J(W, b; x^{(k)}, y^{(k)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{j=1}^{s_l} \sum_{i=1}^{s_{l+1}} (W_{ij}^{(l)})^2$$



$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) = \frac{1}{m} \sum_{k=1}^m \left[ \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(k)}, y^{(k)}) \right] + \lambda W_{ij}^{(l)}$$

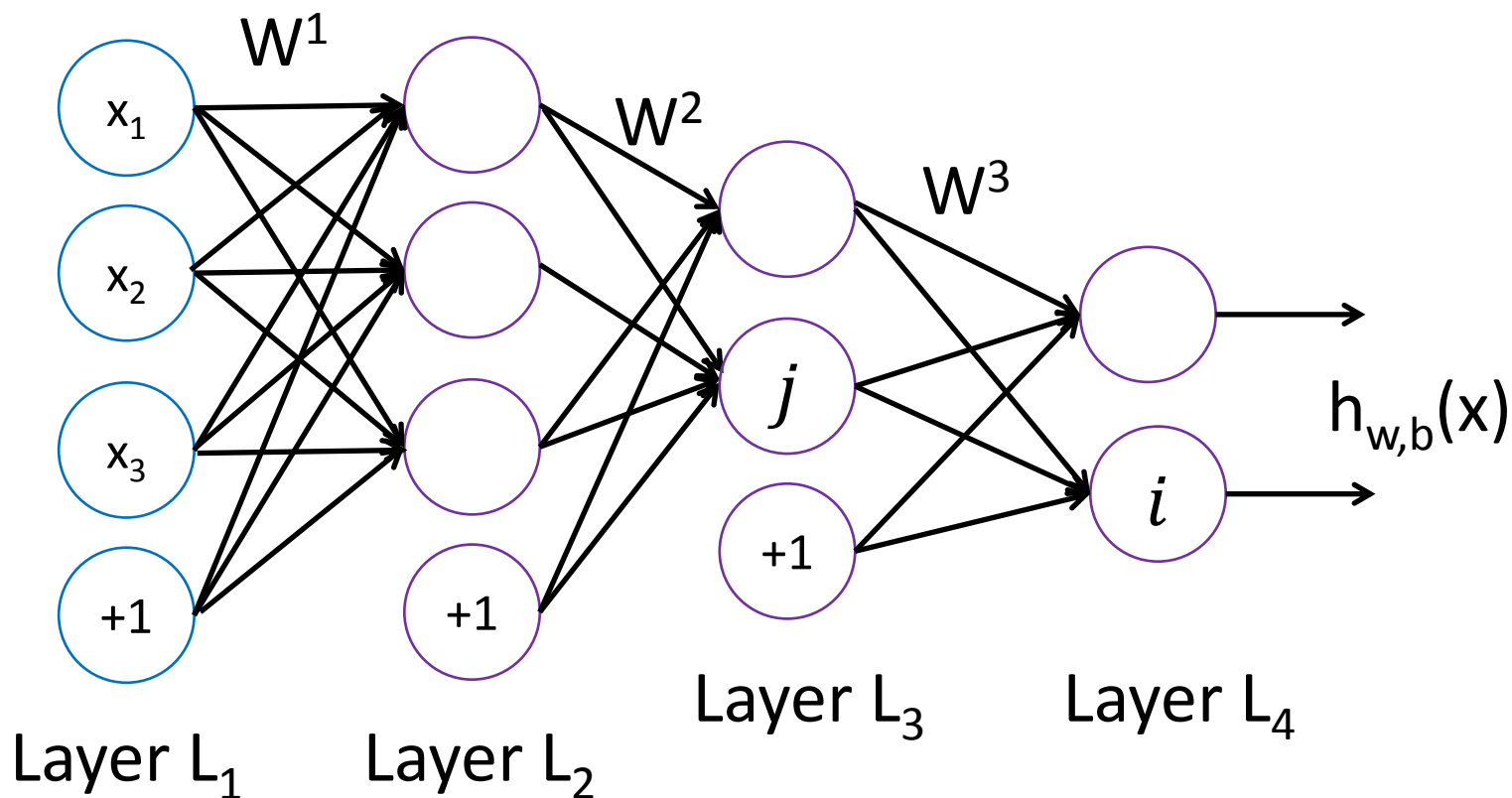
$$\frac{\partial}{\partial b_i^{(l)}} J(W, b) = \frac{1}{m} \sum_{k=1}^m \left[ \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(k)}, y^{(k)}) \right]$$

# 参数学习：反向传播算法

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(k)}, y^{(k)})$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(k)}, y^{(k)})$$

# 参数学习：反向传播算法



$$\frac{\partial}{\partial W_{ij}^{(3)}} J(W, b; x, y) \quad \longrightarrow \quad \frac{\partial}{\partial W_{ij}^{(2)}} J(W, b; x, y) \quad \longrightarrow$$

# 参数学习：反向传播算法

$$\begin{aligned}
 \frac{\partial}{\partial W_{ij}^{(3)}} J(W, b; x, y) &= \frac{\partial}{\partial W_{ij}^{(3)}} \frac{1}{2} \|h_{W,b}(x) - y\|^2 \\
 &= \frac{\partial}{\partial W_{ij}^{(3)}} \frac{1}{2} \|f(z^{(4)}) - y\|^2 = -(y_i - f(z_i^{(4)})) \frac{\partial}{\partial W_{ij}^{(3)}} f(z_i^{(4)}) \\
 &= -\left(y_i - f(z_i^{(4)})\right) f'(z_i^{(4)}) f(z_j^{(3)})
 \end{aligned}$$

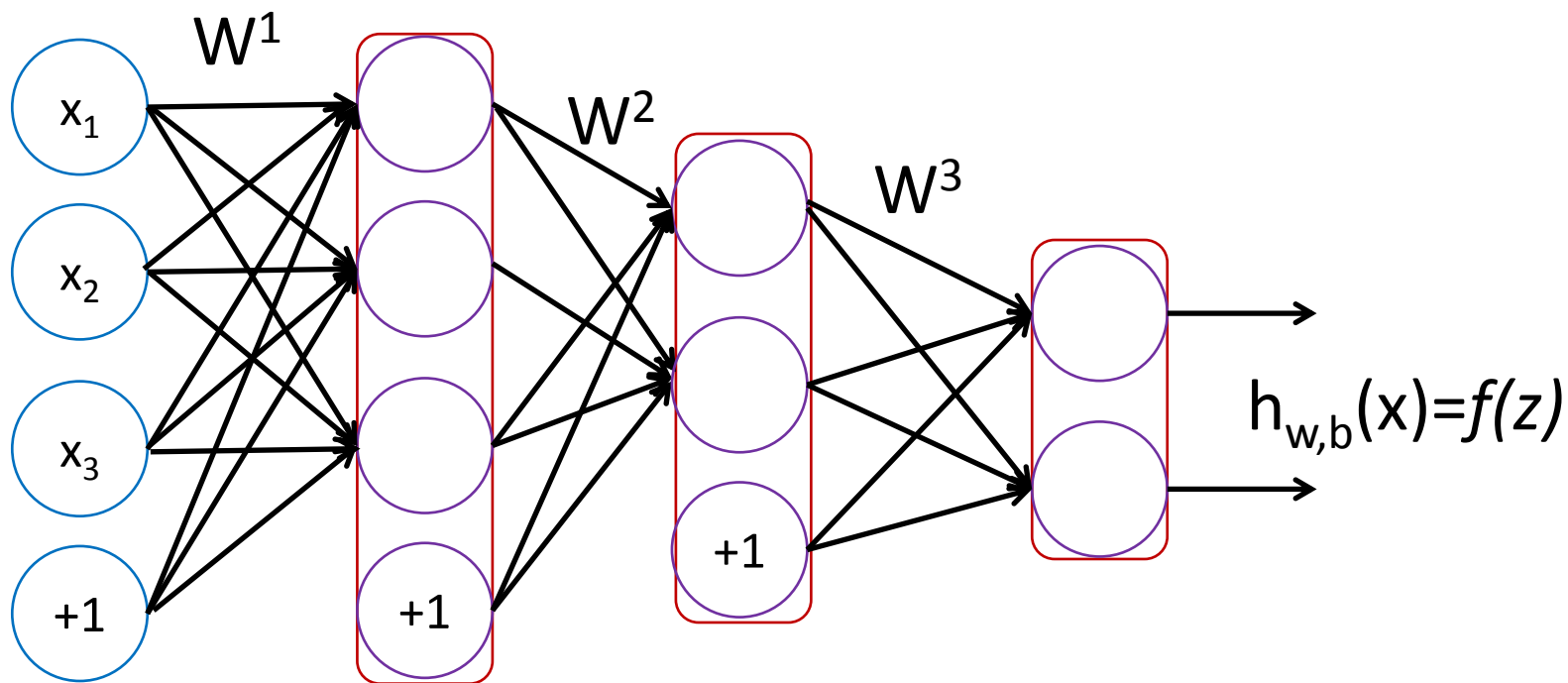
$$z_i^{(4)} = W_{i\cdot}^{(3)} f(z^{(3)}) + b^{(3)} = \sum_k W_{ik}^{(3)} f(z_k^{(3)}) + b^{(3)}$$



# 参数学习：反向传播算法

$$\begin{aligned}
 \frac{\partial}{\partial W_{ij}^{(2)}} J(W, b; x, y) &= \frac{\partial}{\partial W_{ij}^{(2)}} \frac{1}{2} \|h_{W,b}(x) - y\|^2 & z_k^{(3)} &= W_{k\cdot}^{(2)} f(z^{(2)}) + b^{(2)} \\
 &= \frac{\partial}{\partial W_{ij}^{(2)}} \frac{1}{2} \|f(z^{(4)}) - y\|^2 = -(y - f(z^{(4)})) \frac{\partial}{\partial W_{ij}^{(2)}} f(z^{(4)}) \\
 &= -\left(y - f(z^{(4)})\right) f'(z^{(4)}) \sum_k \frac{\partial f(z^{(4)})}{\partial f(z_k^{(3)})} \frac{\partial f(z_k^{(3)})}{\partial W_{ij}^{(2)}} \\
 &= \left(-\left(y - f(z^{(4)})\right) f'(z^{(4)})\right)^T W_{\cdot i}^{(3)} f'(z_i^{(3)}) f(z_j^{(2)}) \\
 &= \left(\sum_k \left[-\left(y_k - f(z_k^{(4)})\right) f'(z_k^{(4)})\right] W_{ki}^{(3)}\right) f'(z_i^{(3)}) f(z_j^{(2)})
 \end{aligned}$$

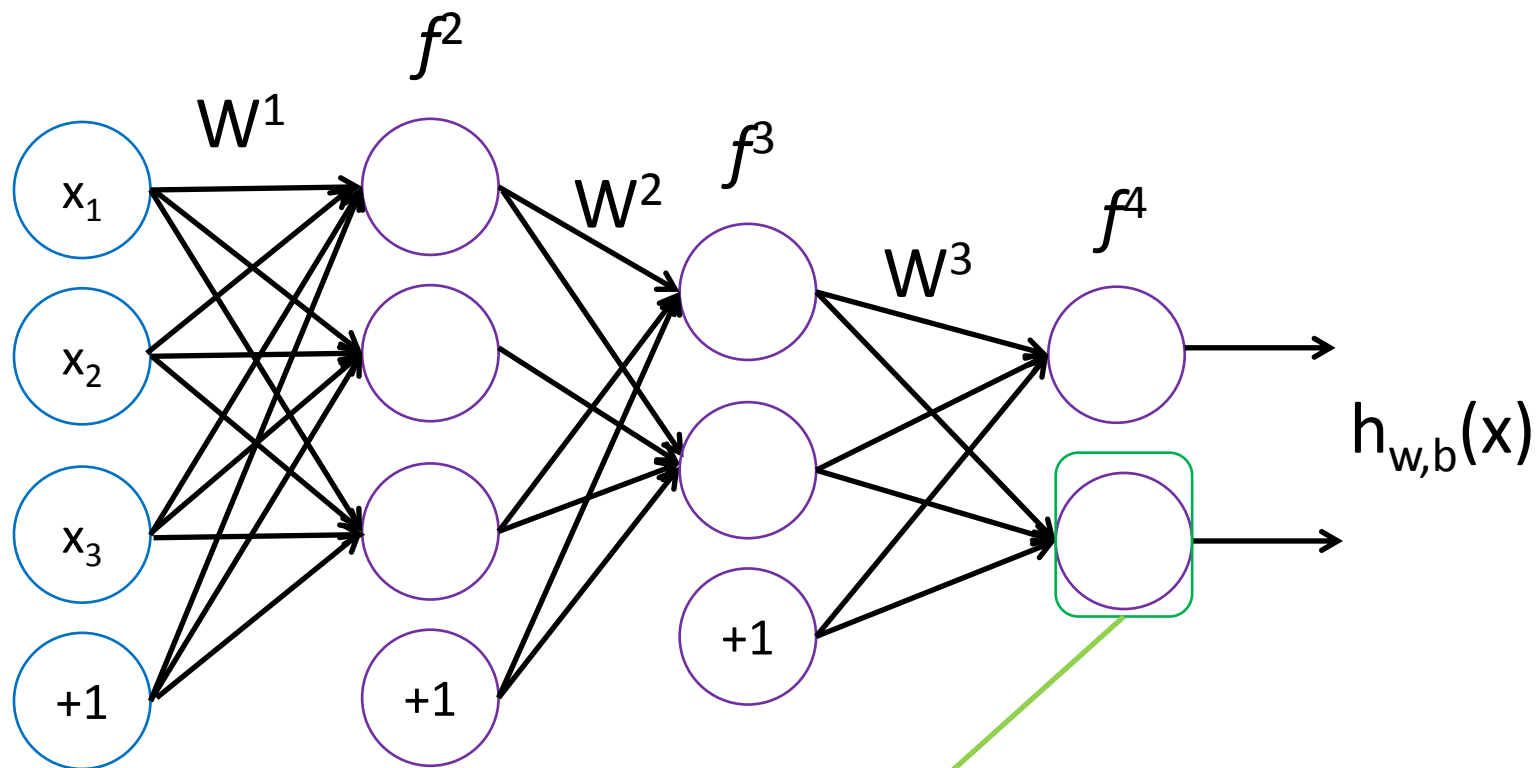
# 参数学习：反向传播算法



1, Forward

计算每个神经元的输出

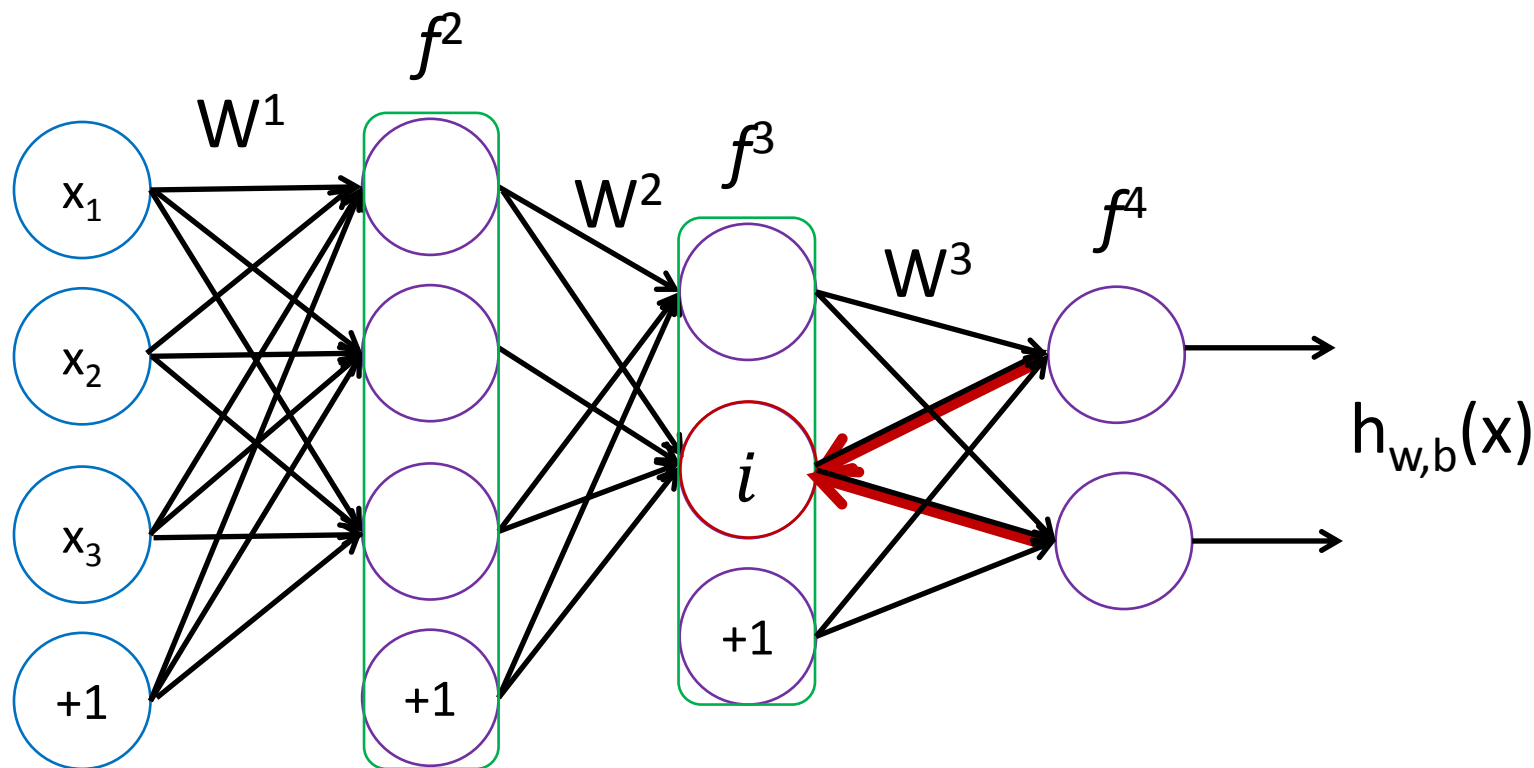
# 参数学习：反向传播算法



2, 计算输出层每个神经元的梯度

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|h_{W,b}(x) - y\|^2 = -\left(y_i - f\left(z_i^{(n_l)}\right)\right) \cdot f'\left(z_i^{(n_l)}\right)$$

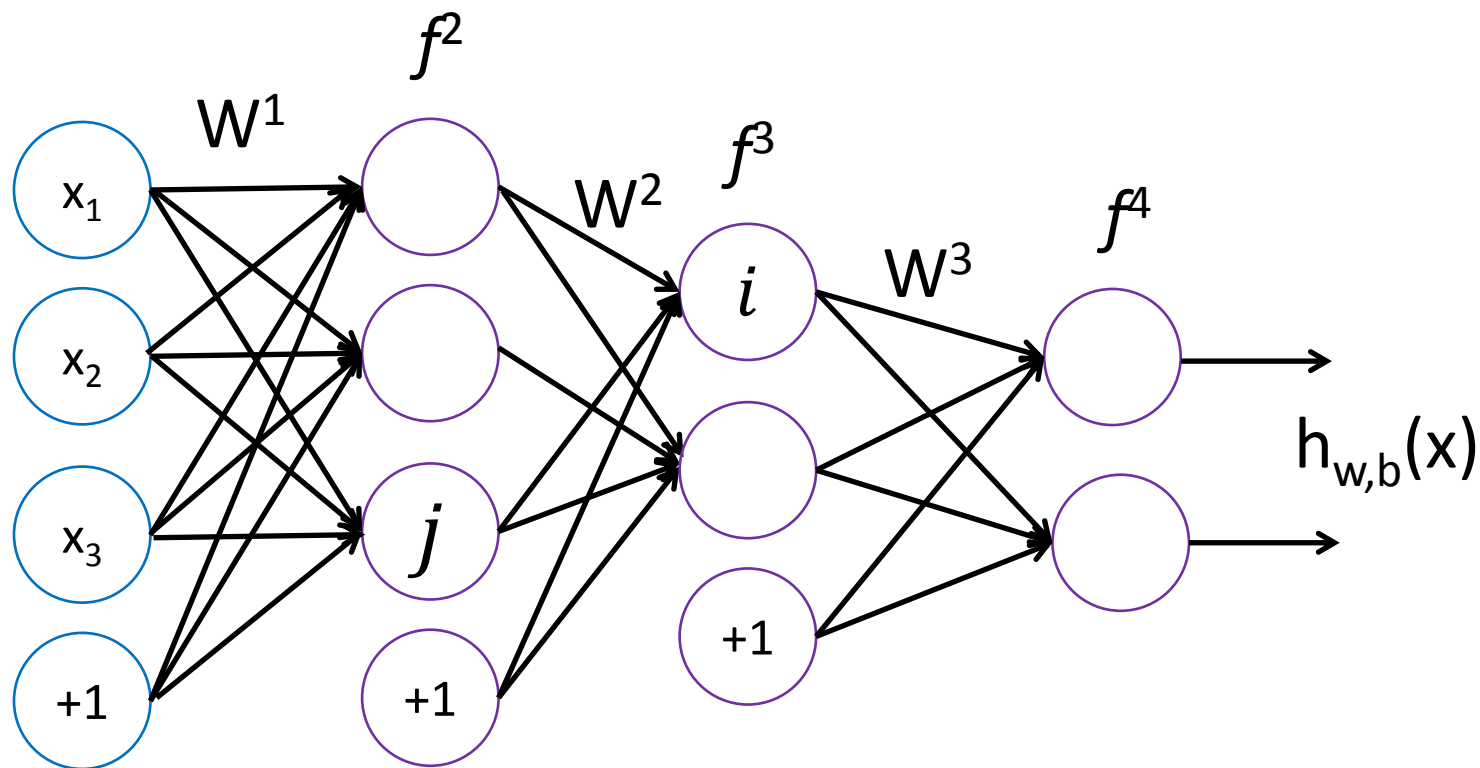
# 参数学习：反向传播算法



3, 反向计算每个神经元的梯度  $\delta_i^{(l)}$

$$\delta_i^{(l)} = \left( \sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) \cdot f' \left( z_i^{(l)} \right)$$

# 参数学习：反向传播算法



4, 计算连接权重和偏差的梯度

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = f(z_j^{(l)}) \cdot \delta_i^{(l+1)} \quad \frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$