

# 第5章 神经网络与神经语言模型

宗成庆

中国科学院自动化研究所

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



# 本章内容



1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

# 1. 问题提出

## ◆ 回顾 $n$ -gram 模型

句子  $s$  的概率:  $p(s) = \prod_{t=1}^m p(w_t | w_1 \dots w_{t-1})$

$$= \prod_{t=1}^m p(w_t | w_{t-n+1}^{t-1})$$

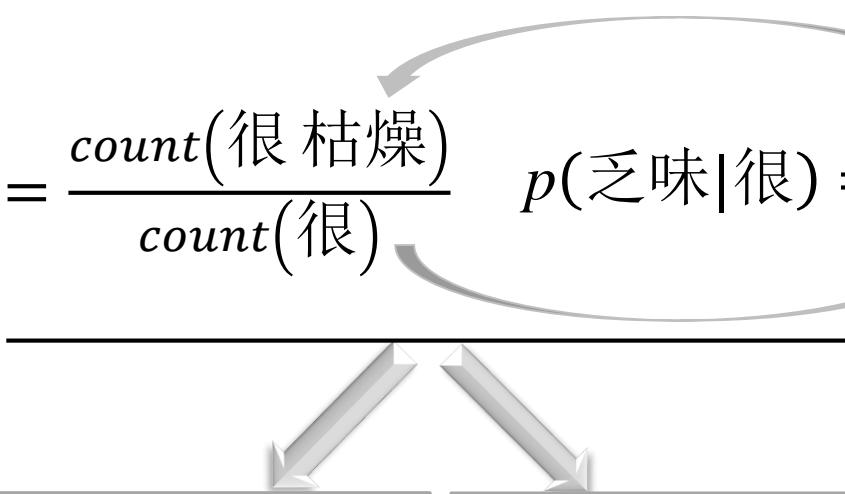


$$p(w_t | w_{t-n+1}^{t-1}) = f(w_t | w_{t-n+1}^{t-1}) = \frac{c(w_{t-n+1}^t)}{\sum_{w_t} c(w_{t-n+1}^t)}$$

# 1. 问题提出

## ● 问题

这本小说很枯燥，读起来很乏味。

$$p(\text{枯燥}|\text{很}) = \frac{\text{count}(\text{很枯燥})}{\text{count}(\text{很})} \quad p(\text{乏味}|\text{很}) = \frac{\text{count}(\text{很乏味})}{\text{count}(\text{很})}$$


①容易产生数据稀疏问题

$n$ -gram “很乏味”有可能未出现在训练样本中。

②忽略了语义相似性

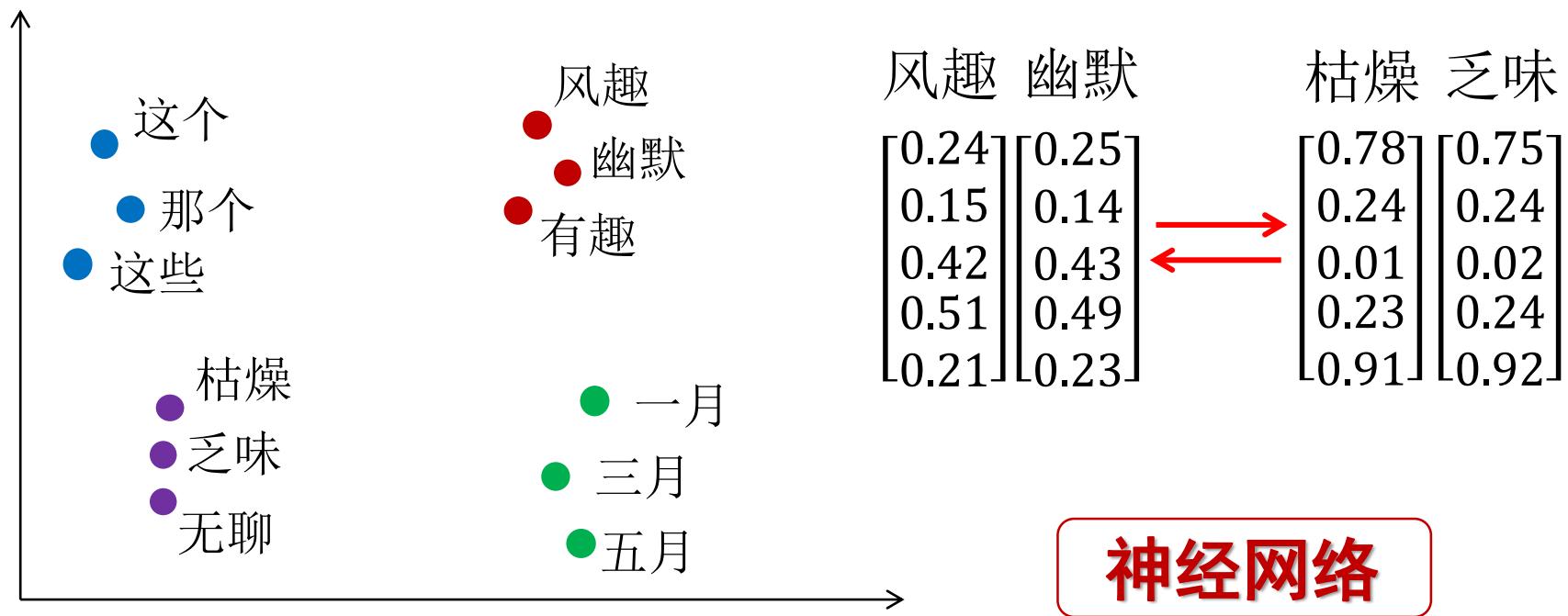
“枯燥”与“乏味”虽然语义相似，但无法共享信息。

# 1. 问题提出

## ● 分析原因

“词”以词形本身表示，是离散的符号。

是否可以用连续空间的分布式表示为每个词赋予一个向量呢？



低维、稠密的连续实数空间



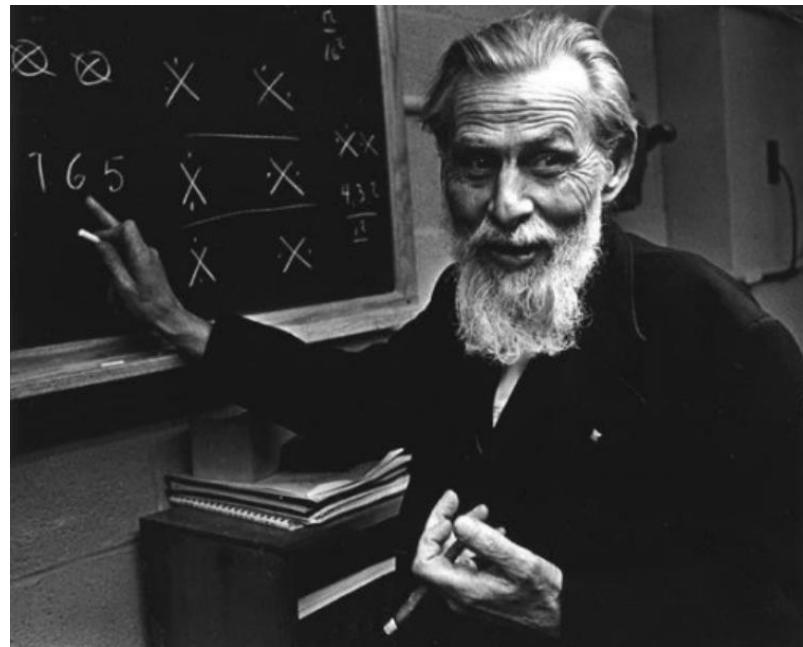
# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

## 2. 神经网络概述

### ◆ 人工神经网络 (artificial neural networks, ANN)

1943年心理学家 沃伦•麦卡洛克(Warren S. McCulloch) 和数理逻辑学家沃尔特·皮茨(Walter H. Pitts)建立了神经网络的数学模型，提出了神经元的形式化数学描述和网络结构方法。



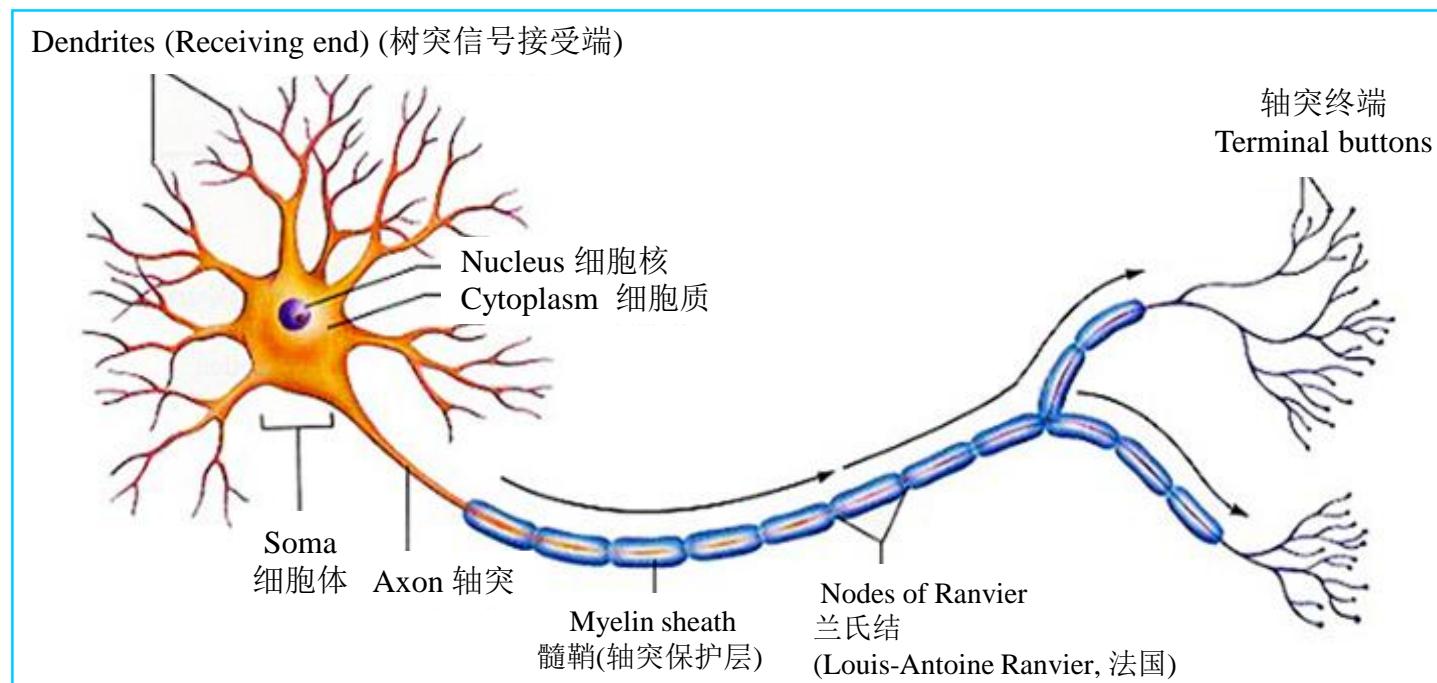
McCulloch (Nov. 16, 1898 – Sept. 24, 1969)



Pitts  
(1923-1969)

## 2. 神经网络概述

### ◆ 神经元(neuron)结构



- 神经元的结构：胞体+树突+轴突
- 胞体和树突接受和整合信息
- 轴突始段产生动作电位，轴突传递信息，轴突末梢将信息传给下一个神经元

## 2. 神经网络概述

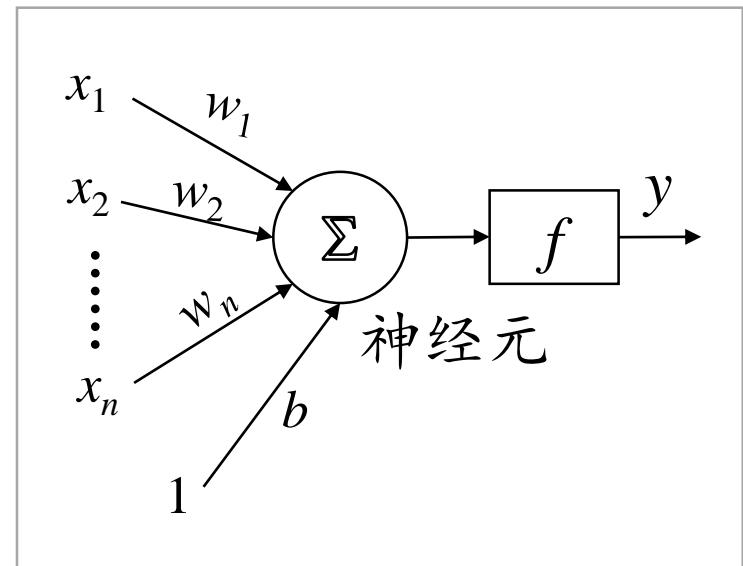
### ◆ 数学描述

#### ● 符号表示

- $x_1 \sim x_n$  为输入向量的各分量；
- $w_1 \sim w_n$  为权值系数(传递效率)；
- $b$  为偏置；
- $f$  为传递函数(激发/激活函数)，

通常是非线性的；

- $\Sigma$  是神经元的阈值；  $y = f(\vec{W} \cdot \vec{X}^T + b)$
  - $y$  为神经元的输出：
- $\vec{W}$  为权值向量；  $\vec{X}$  为输入向量， $\vec{X}^T$  为  $\vec{X}$  的转置。



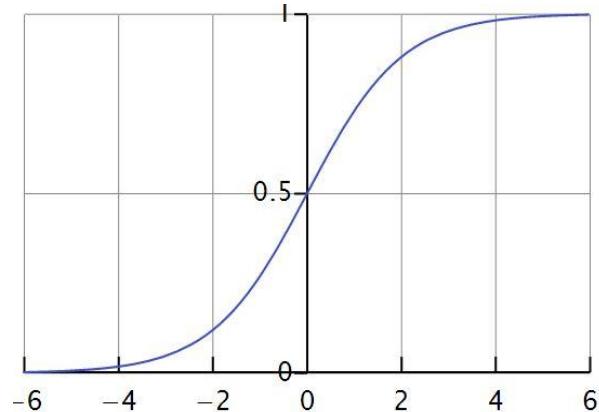
## 2. 神经网络概述

### ● 常用的激活函数

#### ➤ Sigmoid 函数

##### ① Logistic 函数：

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



Logistic 函数可以看成是一个“挤压”函数，把一个实数域的输入“挤压”到(0, 1)。当输入值在0附近时，似为线性函数；当输入值靠近两端时，对输入进行抑制。输入越小，越接近于0；输入越大，越接近于1。这种特点与生物神经元类似，对某些输入会产生兴奋（输出为1），对另一些输入产生抑制（输出为0）。

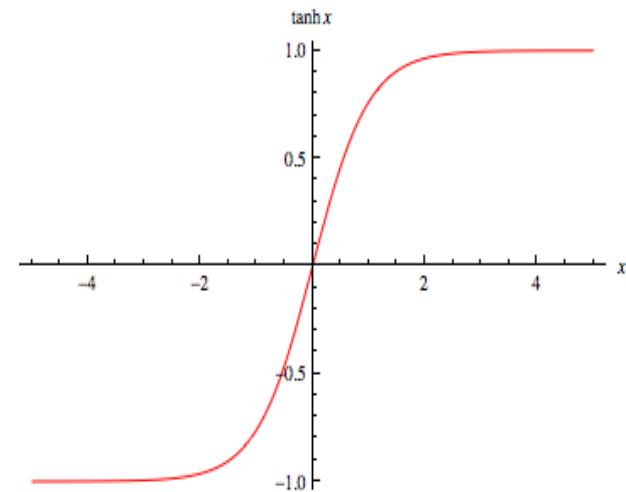
## 2. 神经网络概述

### ②Tanh 函数

$$\tanh = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

或者：

$$\tanh(x) = 2\sigma(2x) - 1$$

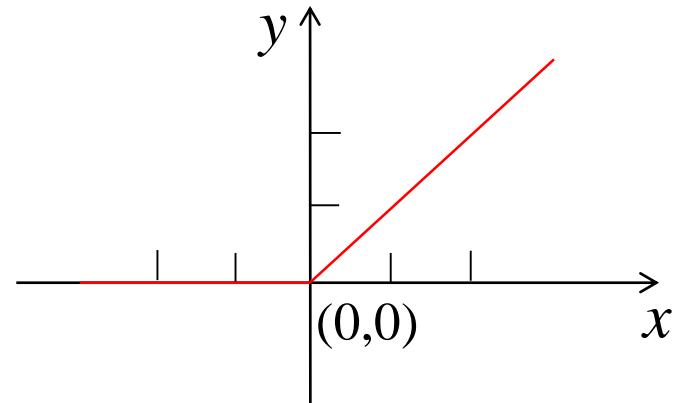


Tanh 函数可以看作是放大并平移的 Logistic 函数，其值域是  $(-1, 1)$ 。

## 2. 神经网络概述

➤ ReLU 函数 (Rectified Linear Unit, 修正线性单元)[Nair et al., 2010]

$$\begin{aligned} \text{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x) \end{aligned}$$



**优点:** 操作简单、高效。在优化方面，相对于Sigmoid 函数的两端饱和，ReLU为左饱和函数，且在 $x>0$ 时导数为1，在一定程度上缓解了神经网络的梯度消失问题，加速梯度下降的收敛速度。

**弱点:** 输出是非0中心化的，给后一层的神经网络引入偏置偏移，影响梯度下降的效率。训练时容易导致神经元“死亡”。



## 2. 神经网络概述

### ◆ 常用的几种神经网络：

- 前馈神经网络 (Feed-forward Neural Network, FNN)
- 卷积神经网络 (Convolutional Neural Network, CNN)
- 循环神经网络 (Recurrent Neural Network, RNN)
- 长短时记忆网络 (Long-Short Term Memory, LSTM)



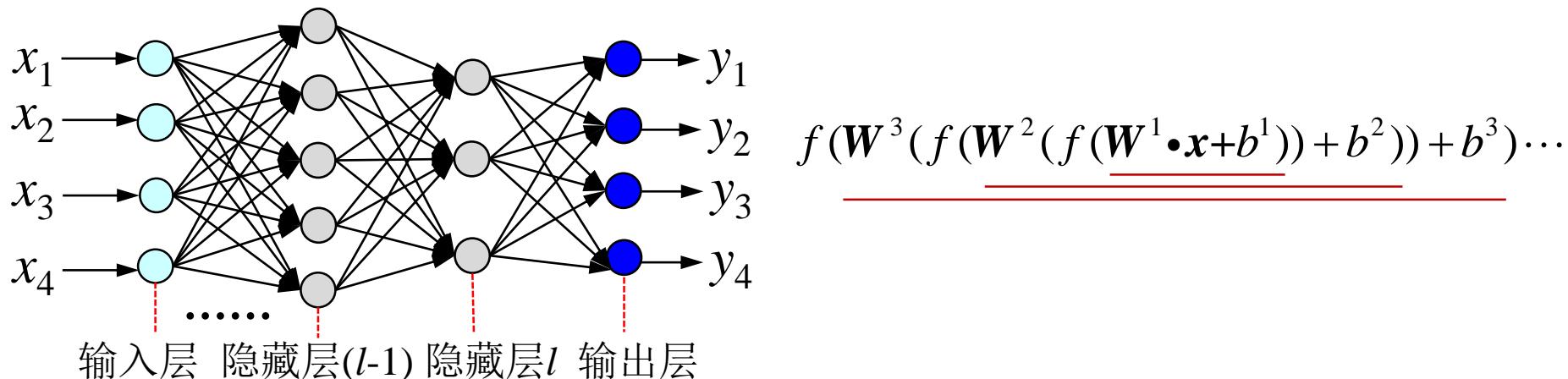
# 本章内容

1. 问题提出
2. 神经网络概述
-  3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

# 3. 前馈神经网络及语言模型

## ◆ FNN描述

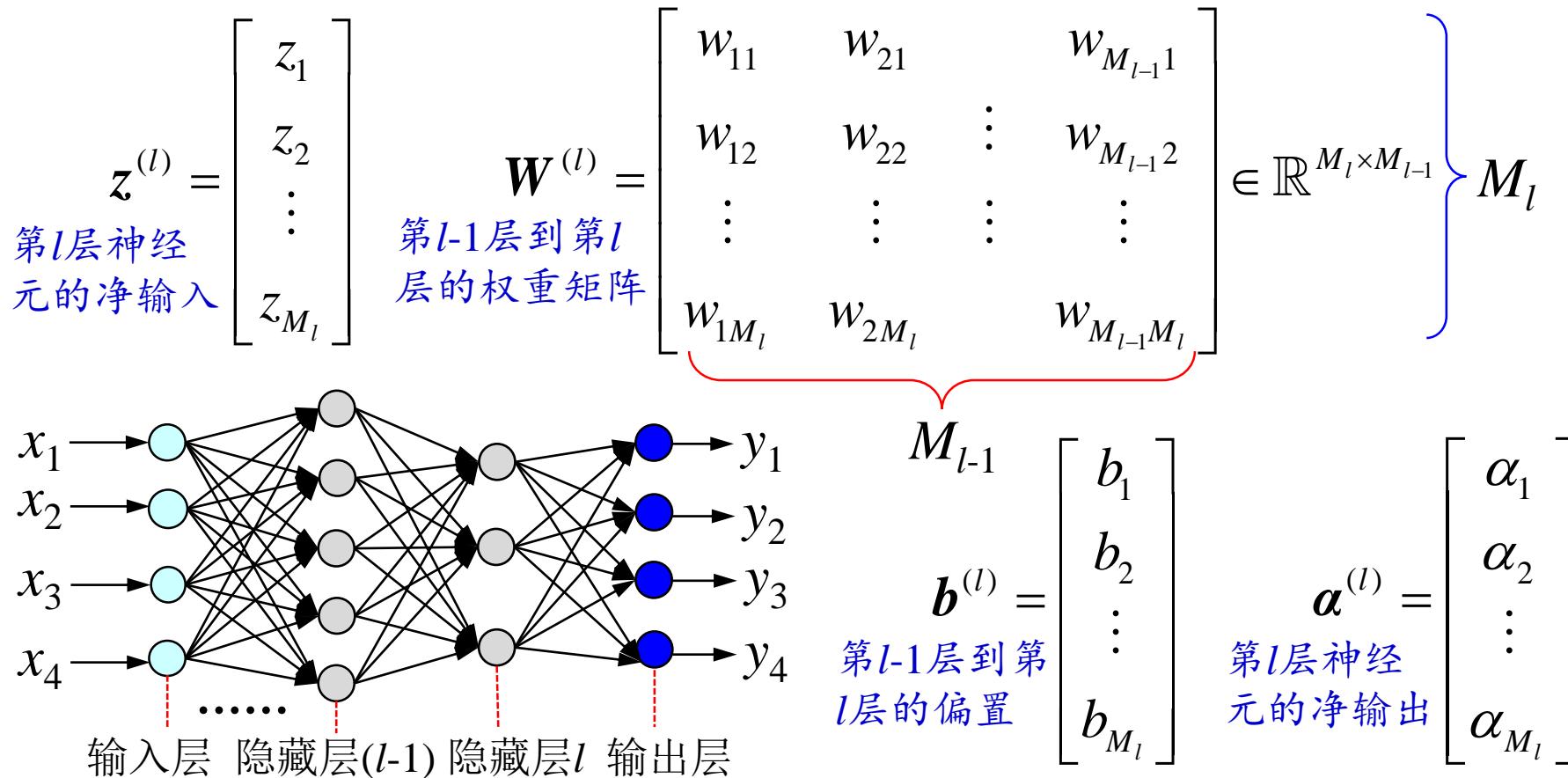
FNN是最早发明的简单人工神经网络，也经常被称为**多层感知器**(Multi-Layer Perceptron, MLP)。前馈网络中各个神经元按接收信息的先后分为不同的组，每一组可以看作一个神经层，每一层中的神经元接收前一层神经元的输出，并输出到下一层神经元，整个网络中的信息是朝一个方向传播，没有反向的信息传播，可以看作是一个有向的无环图，节点全连接。



# 3. 前馈神经网络及语言模型

## ● 网络表示

$L$ 为神经网络的层数；第 $l$ 层有 $M_l$ 个神经元； $f_l(z^{(l)})$ 为第 $l$ 层神经元的激活函数； $\mathbf{b}^{(l)} \in \mathbb{R}^{M_l}$ ,  $\alpha^{(l)} \in \mathbb{R}^{M_l}$ 。





### 3. 前馈神经网络及语言模型

令  $\alpha^{(0)} = \mathbf{x}$ , 前馈神经网络通过迭代进行信息传播:

$$z^{(l)} = \mathbf{W}^{(l)} \cdot \alpha^{(l-1)} + \mathbf{b}^{(l)}$$

$$\alpha^{(l)} = f_l(z^{(l)})$$

根据第  $l-1$  层神经元的**活性值**(Activation)  $\alpha^{(l-1)}$  计算出第  $l$  层神经元的**净活性值**(Net Activation)  $z^{(l)}$ , 然后经过一个激活函数得到第  $l$  层神经元的活性值。上述两个式子可以合并为:

$$\alpha^{(l)} = f_l(\mathbf{W}^{(l)} \cdot \alpha^{(l-1)} + \mathbf{b}^{(l)}) \quad (6-1)$$

整个网络可以看作一个复合函数  $\phi(\mathbf{x}; \mathbf{W}, \mathbf{b})$ , 将向量  $\mathbf{x}$  作为第 1 层的输入  $\alpha^{(0)}$ , 将第  $L$  层的输出  $\alpha^{(L)}$  作为整个函数的输出:

$$\mathbf{x} = \alpha^{(0)} \rightarrow z^{(1)} \rightarrow \alpha^{(1)} \rightarrow z^{(2)} \rightarrow \cdots \rightarrow \alpha^{(L-1)} \rightarrow z^{(L)} \rightarrow \alpha^{(L)} = \phi(\mathbf{x}; \mathbf{W}, \mathbf{b})$$



### 3. 前馈神经网络及语言模型

- 参数学习：确定网络中所有的  $\mathbf{W}$  和  $\mathbf{b}$

如果采用交叉熵作为损失函数，那么对于样本  $(\mathbf{x}, \mathbf{y})$ ，其损失函数为：

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^T \log \hat{\mathbf{y}}$$

第2章课件： $H(X, q) = -\sum_{x \in X} p(x) \log q(x)$

给定训练集  $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$ ，对于每个输入样本  $\mathbf{x}^{(n)}$  得到的网络输出为： $\hat{\mathbf{y}}^{(n)}$ ，那么在数据集  $\mathcal{D}$  上的结构化风险函数为：

$$\mathcal{R}(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{\varepsilon}{2} \|\mathbf{W}\|_F^2 \quad (6-2)$$

其中  $\mathbf{W}$  和  $\mathbf{b}$  分别表示网络中所有的权重矩阵和偏置向量。 $\|\mathbf{W}\|_F^2$  是正则化项（只包括权重  $\mathbf{W}$ ，不包含偏置  $\mathbf{b}$ ），用以防止过拟合。 $0 < \varepsilon < 1$  为超参数， $\varepsilon$  越大， $\mathbf{W}$  越接近于 0。



### 3. 前馈神经网络及语言模型

$\|W\|_F^2$  一般采用 Frobenius 范数(弗罗宾尼斯范数, F-范数):

$$\|W\|_F^2 = \sum_{l=1}^L \sum_{i=1}^{M_l} \sum_{j=1}^{M_{l-1}} (w_{ij}^{(l)})^2 \quad (6-3)$$

网络参数可以通过随机梯度下降法(stochastic gradient descent algorithm)进行学习。梯度表示某一函数在该点处的**方向导数**沿着该方向取得最大值，即函数在该点处沿着该梯度的方向变化最快，变化率最大。对于单变量的实值函数，梯度只是导数，或者说，对于一个线性函数，也就是线的斜率。对于曲面而言，梯度是曲面沿着给定方向的倾斜程度。



### 3. 前馈神经网络及语言模型

在梯度下降方法的每次迭代中，第  $l$  层的参数  $\mathbf{W}^{(l)}$  和  $\mathbf{b}^{(l)}$  参数更新方式为：

$$\begin{aligned}\mathbf{W}^{(l)} &\leftarrow \mathbf{W}^{(l)} - \ell \frac{\partial \mathcal{R}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{W}^{(l)}} \\ &= \mathbf{W}^{(l)} - \ell \left( \frac{1}{N} \sum_{n=1}^N \left( \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{W}^{(l)}} \right) + \varepsilon \mathbf{W}^{(l)} \right)\end{aligned}\quad (6-4)$$

其中， $\ell$  为学习率， $0 < \ell < 1$ 。

$$\begin{aligned}\mathbf{b}^{(l)} &\leftarrow \mathbf{b}^{(l)} - \ell \frac{\partial \mathcal{R}(\mathbf{W}, \mathbf{b})}{\partial \mathbf{b}^{(l)}} \\ &= \mathbf{b}^{(l)} - \ell \left( \frac{1}{N} \sum_{n=1}^N \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{b}^{(l)}} \right)\end{aligned}\quad (6-5)$$

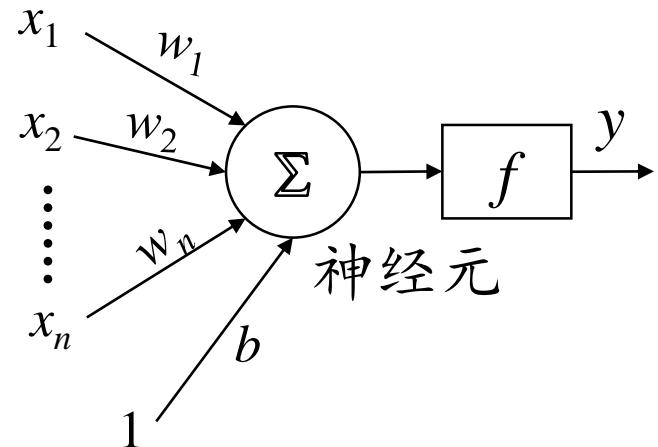
在神经网络的训练中通常使用反向传播(Back Propagation, BP)算法高效地计算梯度。详细推导过程请见本章附录。

# 3. 前馈神经网络及语言模型

## ◆ 基于FNN的语言模型

$$p(\text{风趣|很}) = f \begin{pmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.24 \\ 0.15 \\ 0.42 \\ 0.51 \\ 0.21 \end{pmatrix} \xrightarrow{?} (0, 1)$$

$$p(\text{幽默|很}) = f \begin{pmatrix} 0.01 \\ 0.59 \\ 0.18 \\ 0.05 \\ 0.47 \\ 0.25 \\ 0.14 \\ 0.43 \\ 0.49 \\ 0.23 \end{pmatrix} \xrightarrow{?} (0, 1)$$

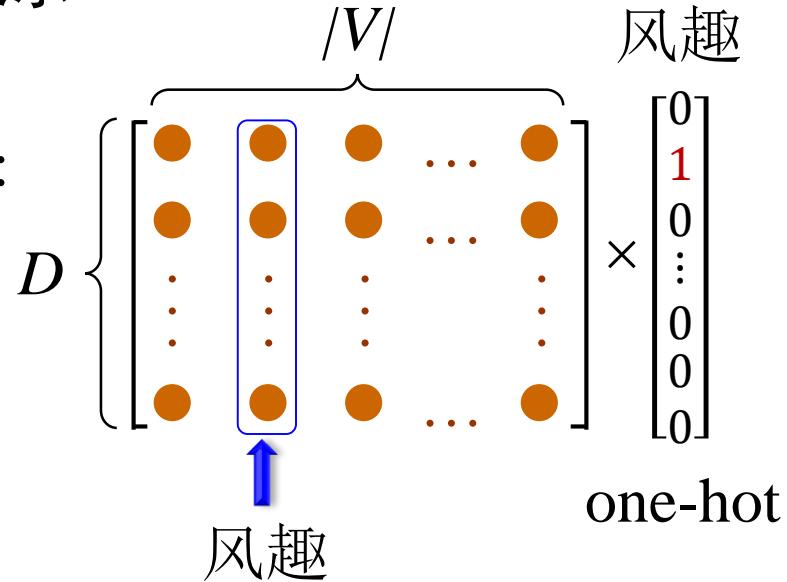


词向量 ?  
 $W, b = ?$

# 3. 前馈神经网络及语言模型

## ● 词向量表示

Look-up  
table( $L_T$ ):



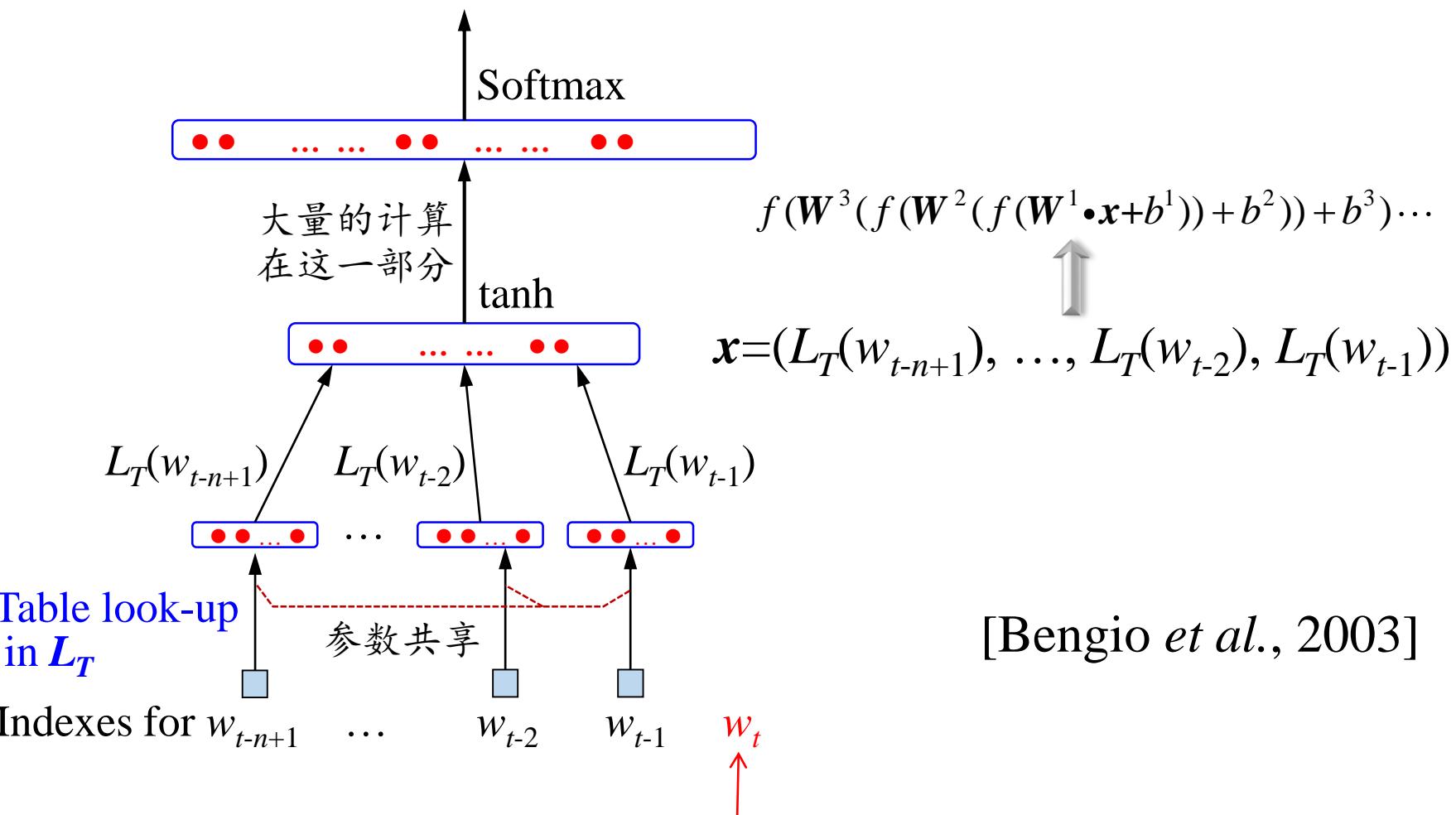
如何学习 $L_T$ ?

通常先随机初始化，  
然后通过目标函数  
优化词的向量表示  
(如最大化语言模  
型似然度)。

- 词汇表 $V$ 的三种确定方法: (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。
- 维度 $D$ 的确定: 超参数, 人工设定, 一般从几十到几百。

# 3. 前馈神经网络及语言模型

输出:  $w_t = p(w_t | \text{context})$





# 3. 前馈神经网络及语言模型

## ● Softmax 回归(regression)

Softmax 回归也称为多项(Multinomial)或多类(Multi-Class)的 Logistic 回归, 是Logistic 回归在多分类问题上的推广, 它也可以看作是一种条件最大熵模型。

对于多类问题, 类别标签  $y \in \{1, 2, \dots, C\}$  可以有  $C$  个取值, 给定一个样本  $\mathbf{x}$ , Softmax 回归预测的属于类别  $c$  的条件概率为:

$$p(y = c | \mathbf{x}) = \text{Softmax}(\mathbf{w}_c^T \mathbf{x}) = \frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{c'=1}^C \exp(\mathbf{w}_{c'}^T \mathbf{x})}$$
$$\mathbf{w}_c = \begin{bmatrix} w_{c1} \\ w_{c2} \\ \vdots \\ w_{cn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

其中,  $\mathbf{w}_c$  是第  $c$  类的权重向量。

$$\text{决策函数: } \hat{y} = \arg \max_{c=1}^C p(y = c | \mathbf{x}) = \arg \max_{c=1}^C \mathbf{w}_c^T \mathbf{x}$$



### 3. 前馈神经网络及语言模型

向量表示：用  $\hat{y} \in \mathbb{R}^C$  表示所有类别预测的条件概率组成的向量：

$$\hat{y} = \text{Softmax}(W^T x) = \frac{\exp(W^T x)}{\mathbf{1}_C^T \exp(W^T x)}$$

其中， $W = [w_1, \dots, w_C]$  是由  $C$  个类的权重向量组成的矩阵，即：

$$W = \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{C1} \\ w_{12} & w_{22} & \cdots & w_{C2} \\ \vdots & \vdots & & \vdots \\ w_{1n} & w_{2n} & \cdots & w_{Cn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{1}_C = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \quad C$$

$$W^T x = \begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix} \quad \exp(W^T x) = \begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix} \quad \mathbf{1}_C^T \exp(W^T x) = \Delta \quad \hat{y} = \frac{\exp(W^T x)}{\mathbf{1}_C^T \exp(W^T x)} = \frac{1}{\Delta} \begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix}$$

$\sum_{c=1}^C \exp(w_c^T x)$

### 3. 前馈神经网络及语言模型

例如：

$$\hat{y} = \begin{bmatrix} 0.03 \\ 0.16 \\ \vdots \\ 0.08 \end{bmatrix}$$

类别1对应的概率  
类别2对应的概率  
类别C对应的概率

表示C个类别预测的概率。第c维的值样本 $x$ 被预测为c类的概率。

# 3. 前馈神经网络及语言模型

## ◆ 举例说明

1. 假设已知 FNN，激活函数用 tanh 函数。

$$L_T = \begin{bmatrix} \dots x_i \dots x_j \dots x_k \dots x_{\cdot} \dots x_{\cdot} \dots \\ \dots 0.1 \dots 0.5 \dots 0.3 \dots 0.4 \dots 0.2 \dots \\ \dots 0.3 \dots 0.4 \dots 0.2 \dots 0.2 \dots 0.1 \dots \end{bmatrix} \quad 2 \times 5000$$

↑    ↑    ↑    ↑    ↑

本 ... 很...乏味...书 ... 这...

$$W = \begin{pmatrix} \dots w_i \dots w_j \dots w_k \dots w_{\cdot} \dots w_{\cdot} \dots w_{\cdot} \dots w_{\cdot} \dots \\ \dots 0.1 \dots 0 \dots 0.2 \dots 0.4 \dots 0.2 \dots 0.1 \dots 0 \dots 0.3 \dots \\ \dots 0.5 \dots 0.4 \dots 0.2 \dots 0 \dots 0.2 \dots 0.6 \dots 0 \dots 0.2 \dots \end{pmatrix}$$

暂不考虑偏置  $b$ 。

$$y = \tanh(w \cdot x + b) \qquad p(\text{乏味} | \text{这}, \text{本}, \text{书}, \text{很}) ?$$

→ Softmax(y)

# 3. 前馈神经网络及语言模型

输入窗口

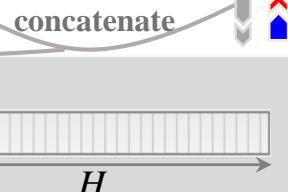
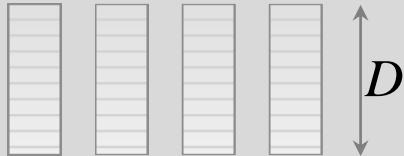
Text 这 本 书 很 ...

查表

$L_T$

Linear

$W^1 \times \odot$



$W^1$

$$\begin{pmatrix} 0.1 & 0 & 0.2 & 0.4 & 0.2 & 0.1 & 0 & 0.3 \\ 0.5 & 0.4 & 0.2 & 0 & 0.2 & 0.6 & 0 & 0.2 \end{pmatrix}$$

$$x = \begin{pmatrix} 0.2 \\ 0.1 \\ 0.1 \\ 0.3 \\ 0.4 \\ 0.2 \\ 0.2 \\ 0.5 \\ 0.4 \end{pmatrix}$$

$$W^1 \times x \rightarrow \begin{pmatrix} 0.38 \\ 0.44 \end{pmatrix}$$

暂不考虑偏置  $b$

$p(\text{乏味} | \text{这, 本, 书, 很})$

①查词表

0.2	0.1	0.4	0.5
0.1	0.3	0.2	0.4
↑	↑	↑	↑
这	本	书	很

②拼接所有的条件词的向量, 形成一个向量。

这 本 书 很

或者加和, 取平均

$$\rightarrow (0.2, 0.1, 0.1, 0.3, 0.4, 0.2, 0.5, 0.4)^T$$

③隐藏层: 线性映射+非线性变换。

# 3. 前馈神经网络及语言模型

输入窗口

Text 这 本 书 很 ...

查表

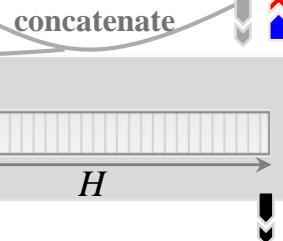
$L_T$

Linear

$W^1 \times \odot$

tanh

$$\begin{pmatrix} 0.38 \\ 0.44 \end{pmatrix}$$



$$p(\text{乏味} | \text{这, 本, 书, 很})$$

①查词表

0.2	0.1	0.4	0.5
0.1	0.3	0.2	0.4
↑	↑	↑	↑
这	本	书	很

②拼接所有的条件词的向量, 形成一个向量。

这 本 书 很

或者加和, 取平均

$$\rightarrow (0.2, 0.1, 0.1, 0.3, 0.4, 0.2, 0.5, 0.4)^T$$

③隐藏层: 线性映射 + 非线性变换。

$$W^1 \times x \rightarrow \begin{pmatrix} 0.38 \\ 0.44 \end{pmatrix} \xrightarrow{\tanh(\bullet)} \begin{pmatrix} 0.36 \\ 0.41 \end{pmatrix}$$

# 3. 前馈神经网络及语言模型

输入窗口

Text 这 本 书 很 ...

查表

$L_T$

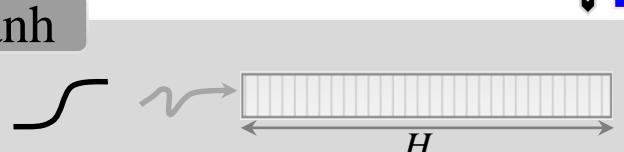
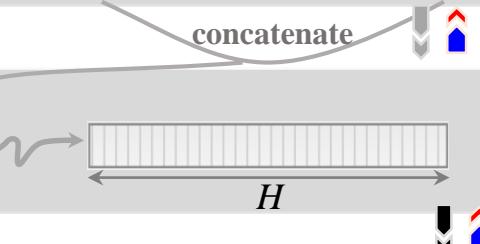
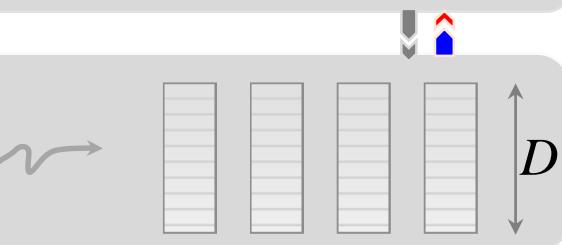
Linear

$W^1 \times \odot$

tanh

$$H' = \begin{pmatrix} 0.36 \\ 0.41 \end{pmatrix}$$

内积, 相关性



$p(\text{乏味} | \text{这, 本, 书, 很})$

①查词表

0.2	0.1	0.4	0.5
0.1	0.3	0.2	0.4
↑	↑	↑	↑
这	本	书	很

②拼接所有的条件词的向量, 形成一个向量。

这 本 书 很

或者加和, 取平均

$$\rightarrow (0.2, 0.1, 0.1, 0.3, 0.4, 0.2, 0.5, 0.4)^T$$

③隐藏层: 线性映射+非线性变换。

$$L_T^T \times H' = \begin{pmatrix} 0.2 & 0.1 \\ 0.1 & 0.3 \\ 0.4 & 0.2 \\ 0.5 & 0.4 \\ 0.3 & 0.2 \\ \dots & \dots \end{pmatrix} \times \begin{pmatrix} 0.36 \\ 0.41 \end{pmatrix}$$

相当于计算给定历史  $H'$  时, 词表中每个词出现在  $H'$  后面时的情况。

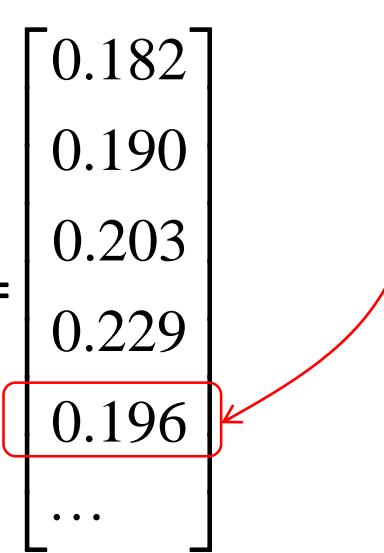
### 3. 前馈神经网络及语言模型

$$\mathbf{L}_T^T \times \mathbf{H}' = \begin{pmatrix} 0.2 & 0.1 \\ 0.1 & 0.3 \\ 0.4 & 0.2 \\ 0.5 & 0.4 \\ \boxed{0.3 & 0.2} \\ \dots \dots \end{pmatrix} \times \begin{pmatrix} 0.36 \\ 0.41 \end{pmatrix} = \begin{pmatrix} 0.113 \\ 0.159 \\ 0.226 \\ 0.344 \\ \boxed{0.190} \\ \dots \dots \end{pmatrix}$$

乏味

$p'(\text{乏味} | \text{这, 本, 书, 很})$

计算Softmax( $\bullet$ )后对应位置上的值就是概率:  $p(\text{乏味} | \text{这, 本, 书, 很})$ 。

$$\hat{\mathbf{y}} = \text{Softmax} \left( \begin{bmatrix} 0.113 \\ 0.159 \\ 0.226 \\ 0.344 \\ 0.190 \\ \dots \end{bmatrix} \right) = \frac{\exp([\bullet])}{\sum \exp(\bullet)} = \begin{bmatrix} 0.182 \\ 0.190 \\ 0.203 \\ 0.229 \\ \boxed{0.196} \\ \dots \end{bmatrix}$$




### 3. 前馈神经网络及语言模型

2. 如果 FNN 未知，需要通过训练样本集获得参数。

(1) 收集足够多的训练样本：

- ① 人间四月芳菲尽，山寺里的桃花才刚刚盛开...
- ② 人们都觉得这本书很乏味。
- ③ 近水楼台先得月，向阳的花木易获阳光和春风...
- ④ 他说这本书很有趣。
- ⑤ 这本书很真实地反映了当时的情况。
- ⑥ 他认为这本书很不适合小学生阅读。
- ⑦ 书架上面这本书很快将被卖完。

.....



### 3. 前馈神经网络及语言模型

- (2) 从训练样本中筛选词汇，确定词汇表  $L_T$ ，根据词频或硬性规定词汇个数；
- (3) 确定表示词汇的向量的维数，初始化词汇表（每一维向量的赋值在 0~1 之间，用均匀分布或高斯分布）：

$$L_T = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_i & \dots \dots \\ 0.3 & 0.5 & 0.3 & \dots & 0.3 & \dots \dots \\ 0.3 & 0.4 & 0.2 & \dots & 0.2 & \dots \dots \\ 0.2 & 0.4 & 0.2 & \dots & 0.3 & \dots \dots \end{bmatrix} \quad 3 \times 30000$$

- (4) 初始化权重  $W$  和偏置  $b$ （取值在 0~1 之间，均匀或高斯分布）；
- (5) 确定激活函数；确定  $n$ -gram 的  $n$  的取值；
- (6) 在整个训练集上尽量准确地预测每一个  $n$  元文法，通过反向传播算法反复调整权重  $W, b$  和每一个词的向量表示  $x_i$ ，直到在训练集上收敛，即风险损失最小。



### 3. 前馈神经网络及语言模型

#### ◆ 开源工具

- 基于FNN的语言模型(feed-forward n-gram neural language model)  
<http://nlg.isi.edu/software/nplm/>

#### ◆ 代表论文

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin. 2003. Neural Probabilistic Language Models[J]. Journal of Machine Learning Research 3 (2003) 1137–11556.



# 3. 前馈神经网络及语言模型

## ◆ 问题分析

- 在FNN中信息传播是单向的，网络模型的能力受到了限制；
- 在FNN中，每次输入是独立的，网络输出只依赖于当前的输入，而在很多任务中，当前时刻的输出不仅与当前的输入有关，还与之前一段时间内的输出都相关；
- FNN要求输入和输出的维数是固定的，不能任意改变，而时序数据（如语音、文本、视频等）的长度一般是不固定的，这样得到的向量维数（以句子为例）就不固定，这就为FNN带来了困难；
- 在 $n$ -gram语言模型和基于FNN的语言模型中，仅对有限范围窗口内的历史信息进行建模，仅考虑前面 $n-1$ 个词的历史信息：

$$p(s) = \prod_{t=1}^m p(w_t | w_1 \dots w_{t-1}) = \prod_{t=1}^m p(w_t | w_{t-n+1}^{t-1})$$

能否对更长的甚至所有的历史信息进行建模呢？



# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
- 4. 循环神经网络及语言模型**
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

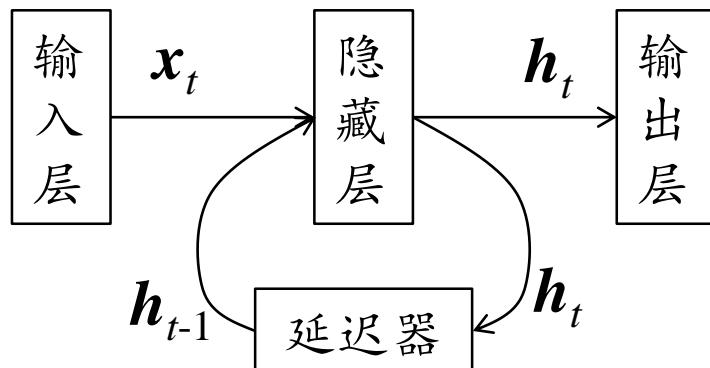
# 4. 循环神经网络及语言模型

## ◆ RNN 模型描述

RNN通过使用带自反馈的神经元，能够处理任意长度的时序数据。给定一个输入序列： $\mathbf{x}_1^T = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ ，RNN通过下面的公式更新带反馈边的隐藏层的活性值 $\mathbf{h}_t$ ：

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

其中， $\mathbf{h}_0=0$ ,  $f(\cdot)$  为一个非线性函数，可以是一个前馈网络。



“延迟器”是一个虚拟单元，记录神经元最近一次（或几次）的活性值。

$\mathbf{h}_t$  也被称作“状态(state)”或“隐状态(hidden state)”。



## 4. 循环神经网络及语言模型

假设  $\mathbf{x}_t \in \mathbb{R}^M$  是  $t$  时刻的网络输入,  $\mathbf{h}_t \in \mathbb{R}^D$  是隐藏层的状态(即隐藏层神经元的活性值), 那么,  $\mathbf{h}_t$  不仅与当前时刻的输入  $\mathbf{x}_t$  有关, 而且与上一时刻的隐藏层状态  $\mathbf{h}_{t-1}$  有关, 因此, 一个简单的RNN在  $t$  时刻的更新公式为:

$$\begin{aligned}\mathbf{z}_t &= \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b} \\ \mathbf{h}_t &= f(\mathbf{z}_t)\end{aligned}$$

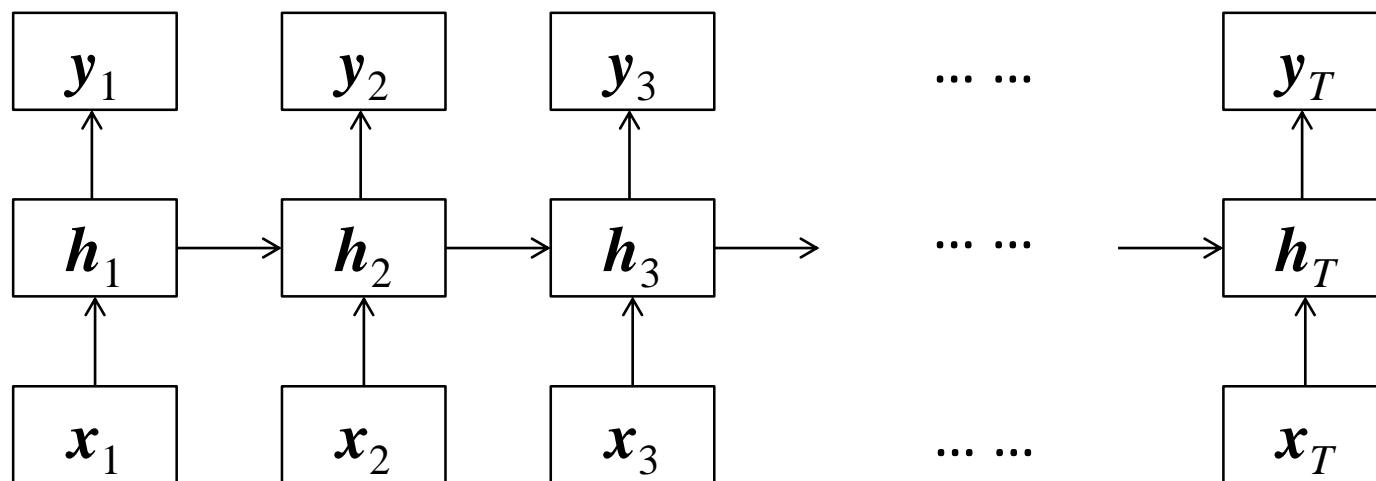
其中,  $\mathbf{z}_t$  为隐藏层的净输入,  $\mathbf{U}_t \in \mathbb{R}^{D \times D}$  为**状态-状态**的权重矩阵;  $\mathbf{W} \in \mathbb{R}^{D \times M}$  为**状态-输入**的权重矩阵;  $\mathbf{b} \in \mathbb{R}^D$  为偏置向量。 $f(\cdot)$  为非线性激活函数, 通常使用Logistic 函数或者 tanh函数。

上面的两个公式可以直接写成:

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$$

# 4. 循环神经网络及语言模型

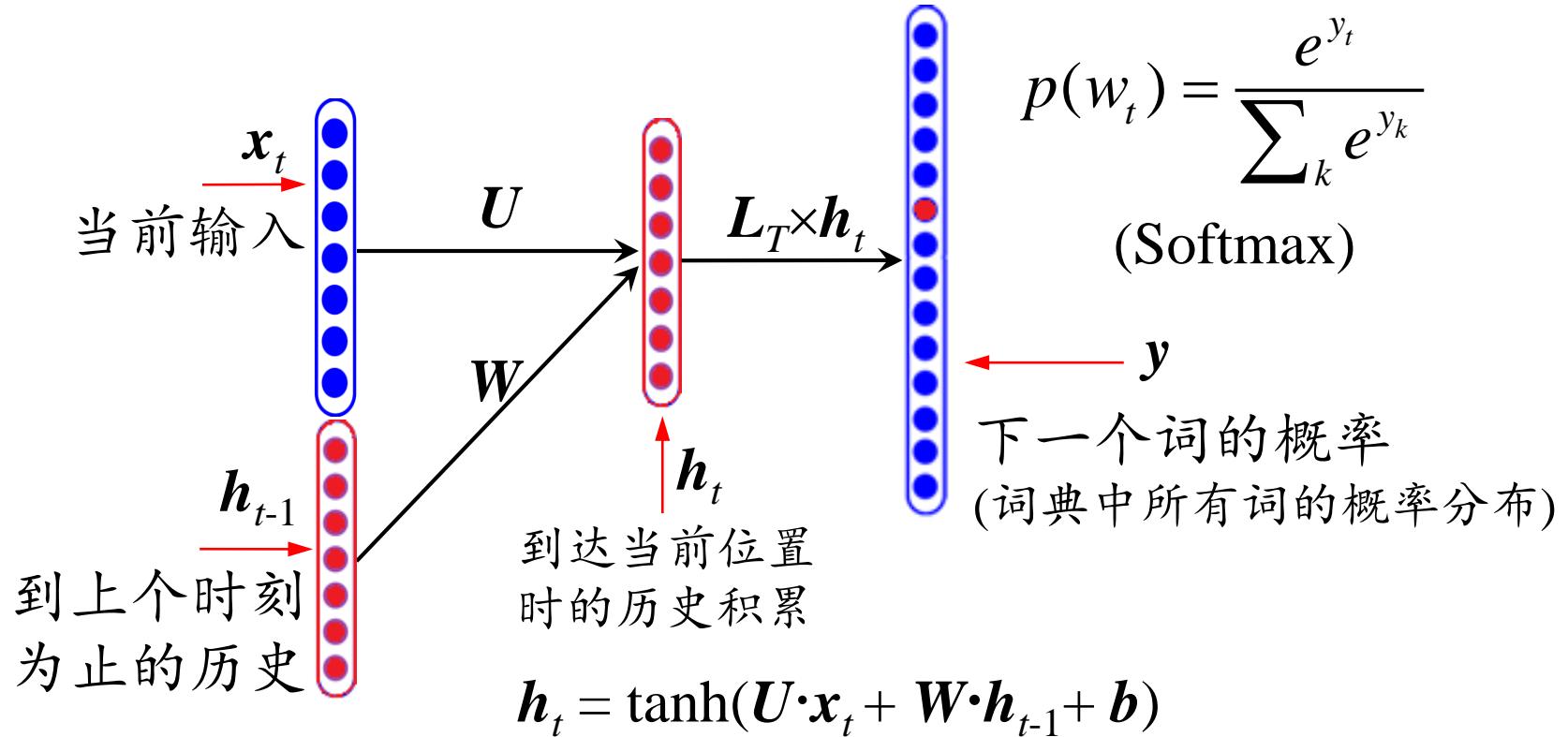
如果把每个时刻的状态都看作是FNN的一层，RNN可以看作是在时间维度上权值共享的神经网络。下面的图是按时间展开的RNN：



# 4. 循环神经网络及语言模型

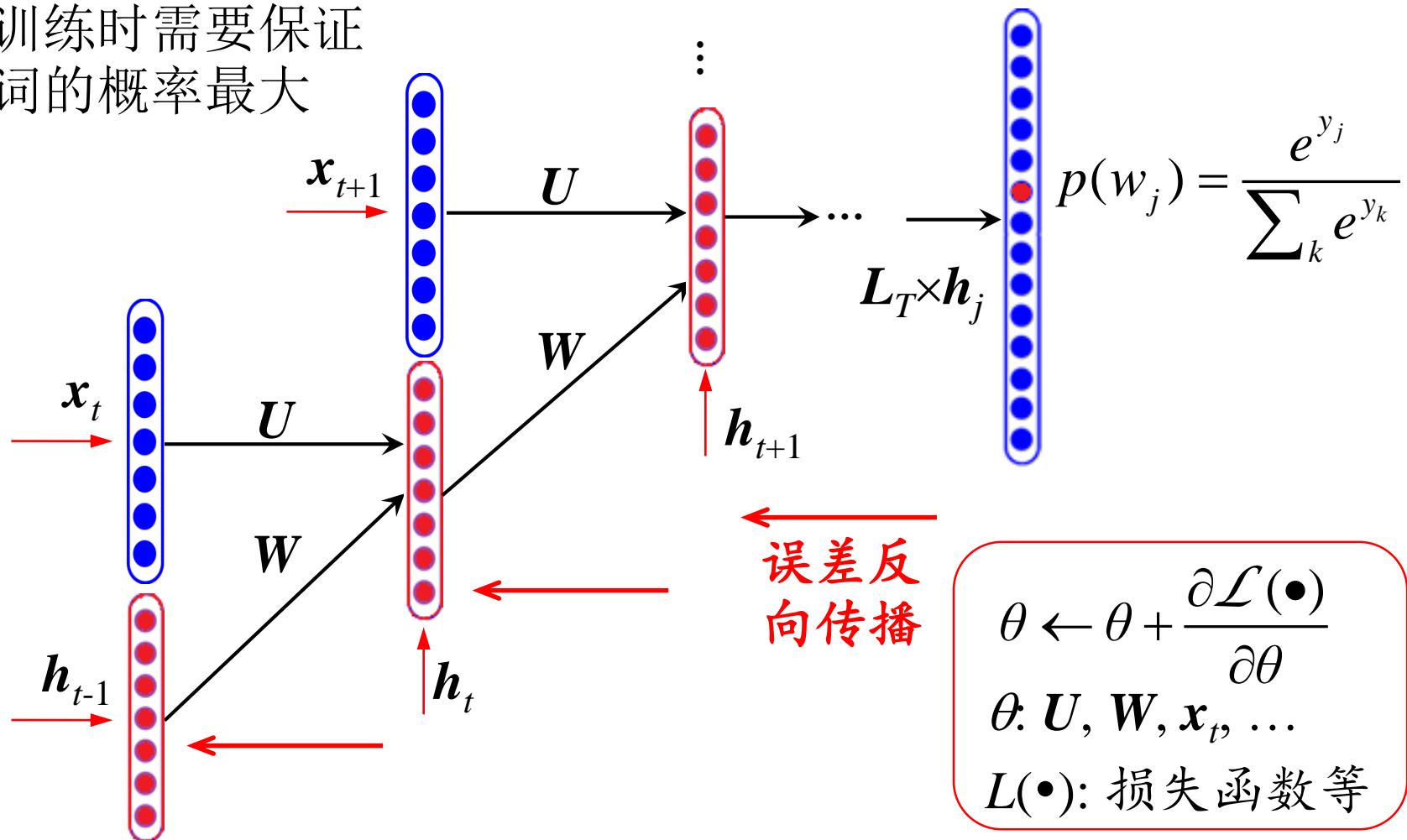
## ◆ 基于RNN的语言模型

- 输入: 从开始到  $t-1$  时刻的历史  $\mathbf{h}_{t-1}$ ; 当前位置  $t$  的词向量  $\mathbf{x}_t$ ;
- 输出: 到  $t$  位置时的历史积累  $\mathbf{h}_t$  及其该位置上词的概率。



# 4. 循环神经网络及语言模型

模型训练时需要保证  
当前词的概率最大



# 4. 循环神经网络及语言模型

$$x_6 = [0.1, 0.1]^T \text{ 看}$$

$$x_5 = [0.1, 0.2]^T \text{ 好}$$

$$x_4 = [0.1, 0.3]^T \text{ 很}$$

$$x_3 = [0.2, 0.1]^T \text{ 书}$$

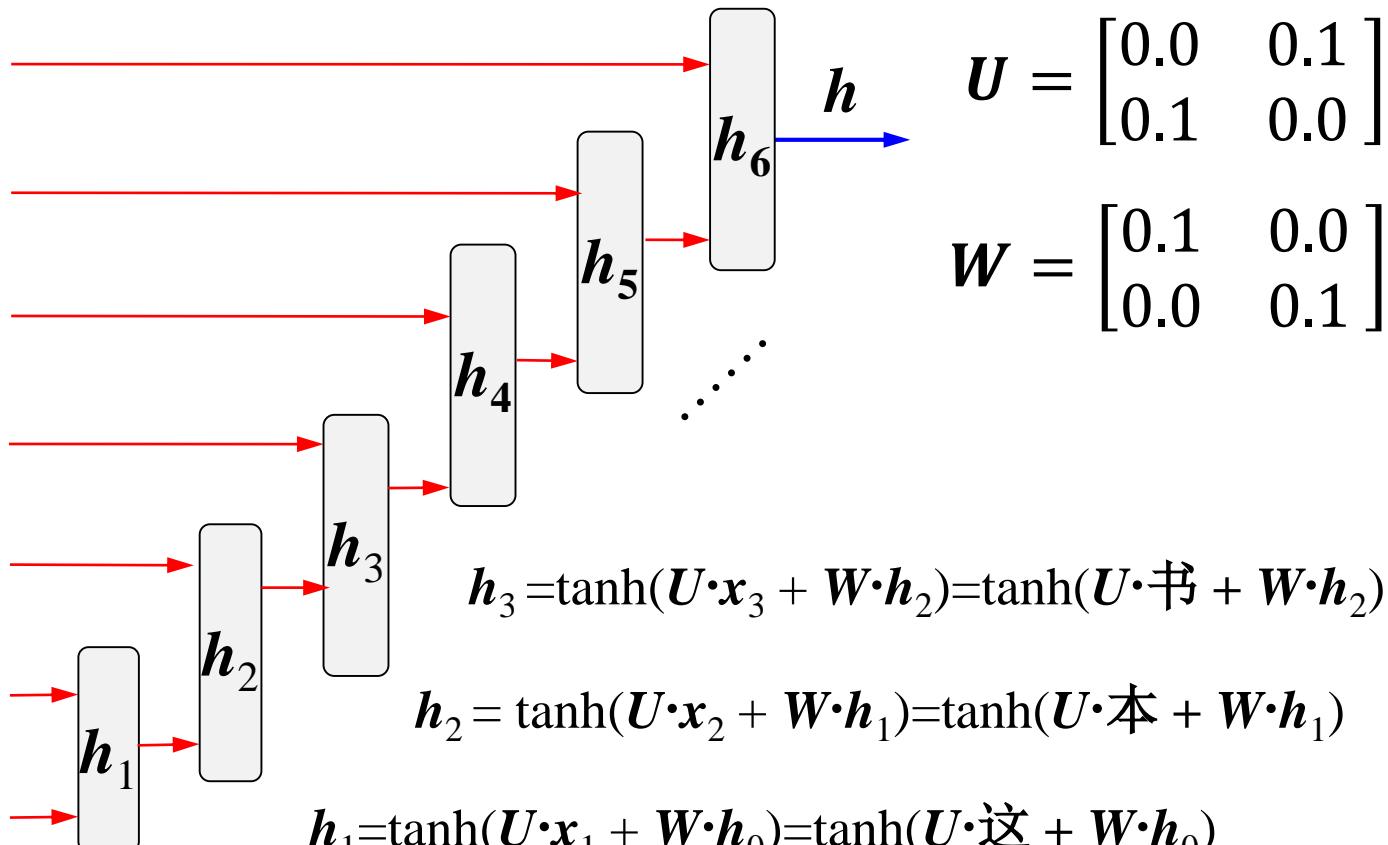
$$x_2 = [0.3, 0.1]^T \text{ 本}$$

$$x_1 = [0.2, 0.2]^T \text{ 这}$$

$$h_0 = [0.0, 0.0]^T$$

$$h_t = \tanh(U \cdot x_t + W \cdot h_{t-1})$$

不考虑偏置。



$$h_3 = \tanh(U \cdot x_3 + W \cdot h_2) = \tanh(U \cdot \text{书} + W \cdot h_2)$$

$$h_2 = \tanh(U \cdot x_2 + W \cdot h_1) = \tanh(U \cdot \text{本} + W \cdot h_1)$$

$$h_1 = \tanh(U \cdot x_1 + W \cdot h_0) = \tanh(U \cdot \text{这} + W \cdot h_0)$$

$$= \tanh\left(\begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 0.0 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

# 4. 循环神经网络及语言模型

$x_6 = [0.1, 0.1]^T$  看

$x_5 = [0.1, 0.2]^T$  好

$x_4 = [0.1, 0.3]^T$  很

$x_3 = [0.2, 0.1]^T$  书

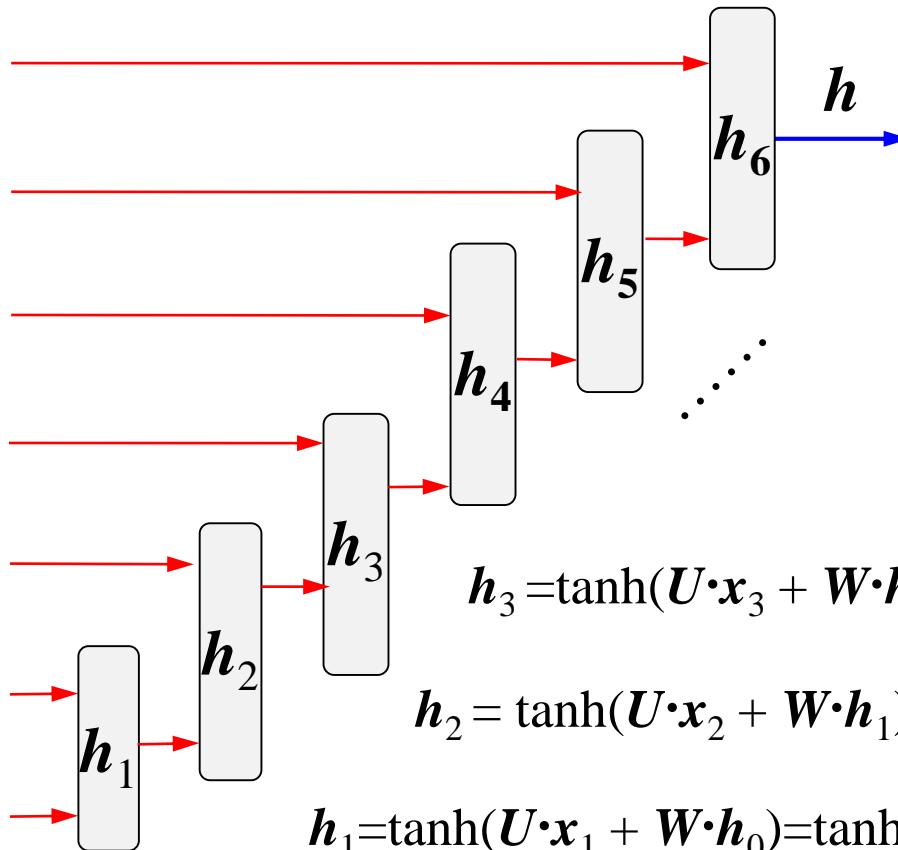
$x_2 = [0.3, 0.1]^T$  本

$x_1 = [0.2, 0.2]^T$  这

$\mathbf{h}_0 = [0.0, 0.0]^T$

$\mathbf{h}_t = \tanh(\mathbf{U} \cdot \mathbf{x}_t + \mathbf{W} \cdot \mathbf{h}_{t-1})$

不考虑偏置。



①  $\mathbf{h}$  可以看作是整个句子的向量表示；

② 如果预测下一个词：

$$\text{softmax}(\mathbf{L}_T^T \times \mathbf{h})$$

$$\mathbf{h}_3 = \tanh(\mathbf{U} \cdot \mathbf{x}_3 + \mathbf{W} \cdot \mathbf{h}_2) = \tanh(\mathbf{U} \cdot \text{书} + \mathbf{W} \cdot \mathbf{h}_2)$$

$$\mathbf{h}_2 = \tanh(\mathbf{U} \cdot \mathbf{x}_2 + \mathbf{W} \cdot \mathbf{h}_1) = \tanh(\mathbf{U} \cdot \text{本} + \mathbf{W} \cdot \mathbf{h}_1)$$

$$\mathbf{h}_1 = \tanh(\mathbf{U} \cdot \mathbf{x}_1 + \mathbf{W} \cdot \mathbf{h}_0) = \tanh(\mathbf{U} \cdot \text{这} + \mathbf{W} \cdot \mathbf{h}_0)$$

$$= \tanh\left(\begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 0.0 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$



# 4. 循环神经网络及语言模型

## ◆ 开源工具

- 基于循环神经网络的语言模型(recurrent neural language model)  
<http://rnnlm.org/>

## ◆ 代表论文

Mikolov, T. , M Karafi át, Burget, L. , Cernock, J. , and Khudanpur, S. 2010. Recurrent neural network based language model. In *Proceedings of the International Conference on Speech Communication Association* (Interspeech), Makuhari, Chiba, Japan. Pages 1045-1048.

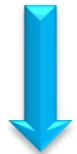


# 4. 循环神经网络及语言模型

## ◆ 问题分析

**梯度消失或爆炸：**参数W 经过多次传递后有可能导致梯度消失(小于1时)或者爆炸(大于1)。

是否能够通过某种策略选择性地保留或者遗忘某些信息？



长短时记忆网络LSTM (Long Short-Term Memory)。

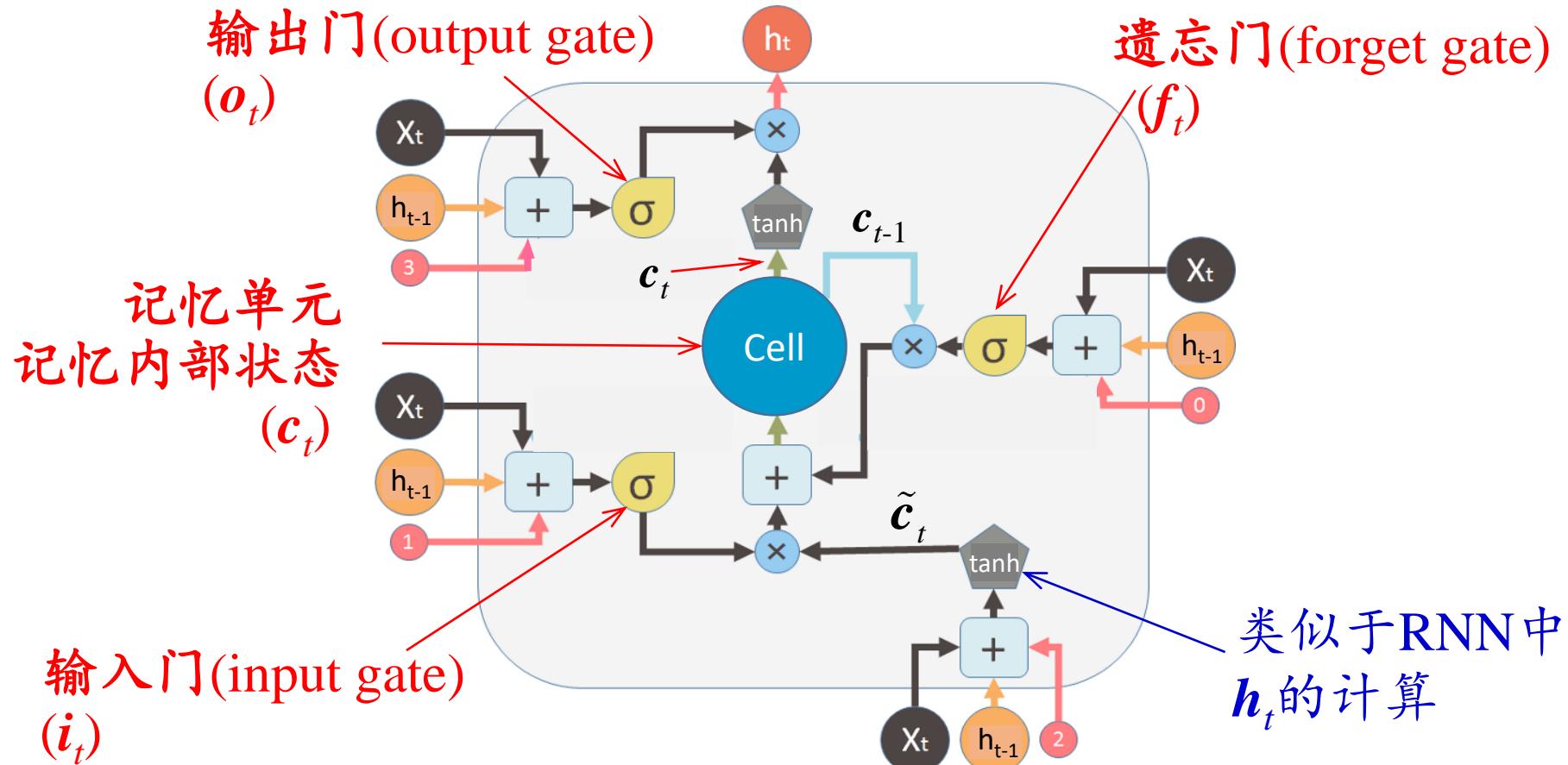


# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
-  5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

# 5. 长短时记忆网络

## ◆ LSTM描述



LSTM 又称基于门控机制的循环神经网络。



# 5. 长短时记忆网络

## ● 门控机制 (0/1开关)

(1) 遗忘门  $f_t$ : 控制上一时刻的内部状态  $c_{t-1}$  需要遗忘多少信息。

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

(2) 输入门  $i_t$ : 控制当前时刻的候选状态  $\tilde{c}_t$  有多少信息需要保存。

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

(3) 输出门  $o_t$ : 控制当前时刻的内部状态  $c_t$  有多少信息需要输出给外部状态  $h_t$ 。

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

其中,  $\sigma(\cdot)$  是 Logistic 函数, 输出区间为(0,1),  $x_t$  为当前时刻的输入,  $h_{t-1}$  为上一时刻的外部状态。



## 5. 长短时记忆网络

● 内部状态： $c_t = \mathbb{R}^D$  进行线性的循环信息传递，同时（非线性地）输出信息给隐藏层的外部状态  $h_t = \mathbb{R}^D$ 。 $c_t$  的计算公式如下：

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

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

其中， $f_t \in [0, 1]^D$ ,  $i_t \in [0, 1]^D$  和  $o_t \in [0, 1]^D$  控制传递信息的路径；  
○表示向量元素的乘积； $c_{t-1}$  为上一时刻的记忆单元； $\tilde{c}_t \in \mathbb{R}^D$  是通过非线性函数得到的候选状态：

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

在每一个时刻  $t$ , LSTM 内部网络的内部状态  $c_t$  记录了到当前时刻为止的历史信息。



## 5. 长短时记忆网络

● 内部状态： $c_t = \mathbb{R}^D$  进行线性的循环信息传递，同时（非线性地）输出信息给隐藏层的外部状态  $h_t = \mathbb{R}^D$ 。 $c_t$  的计算公式如下：

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

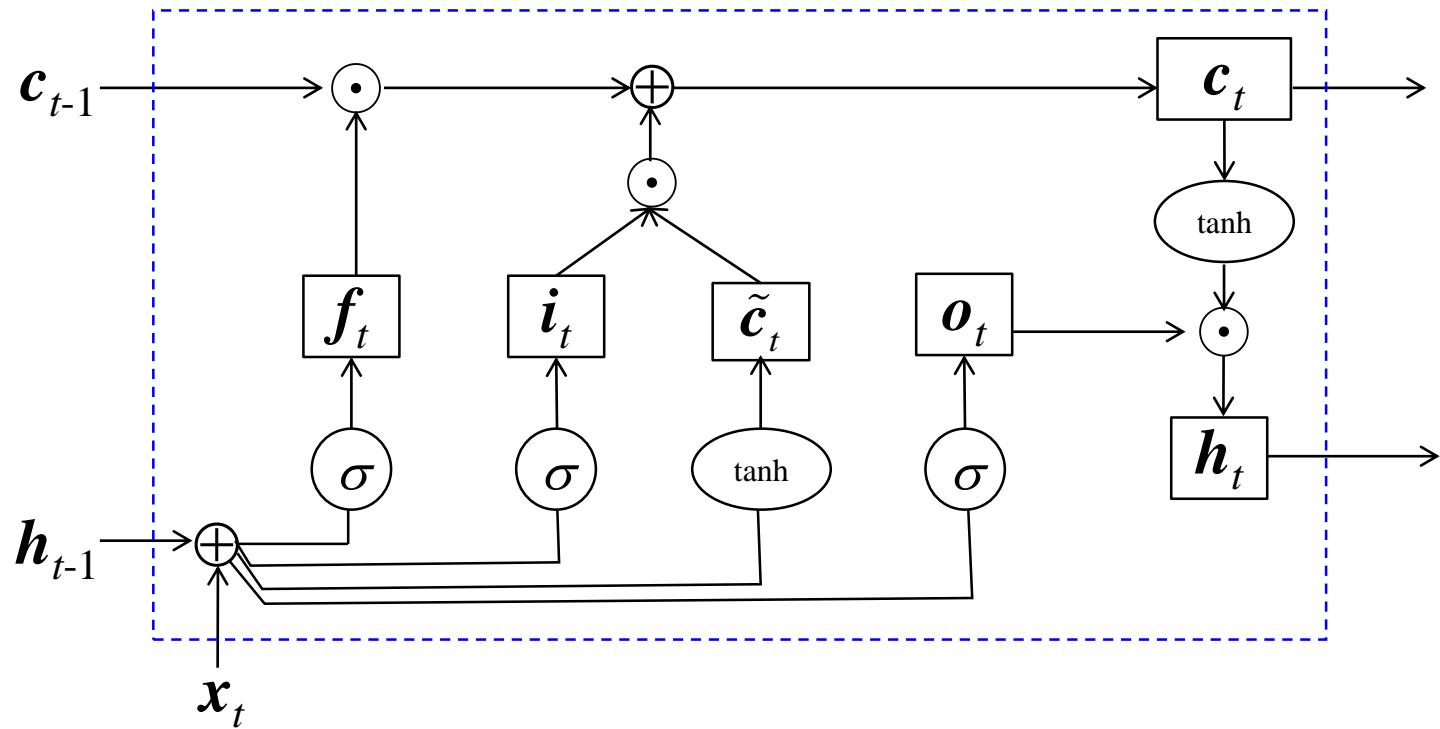
$$h_t = o_t \odot \tanh(c_t)$$

其中， $f_t \in [0, 1]^D$ ,  $i_t \in [0, 1]^D$  和  $o_t \in [0, 1]^D$  控制传递信息的路径；  
○表示向量元素的乘积； $c_{t-1}$  为上一时刻的记忆单元； $\tilde{c}_t \in \mathbb{R}^D$  是通过非线性函数得到的候选状态：

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

在每一个时刻  $t$ , LSTM 内部网络的内部状态  $c_t$  记录了到当前时刻为止的历史信息。

# 5. 长短时记忆网络

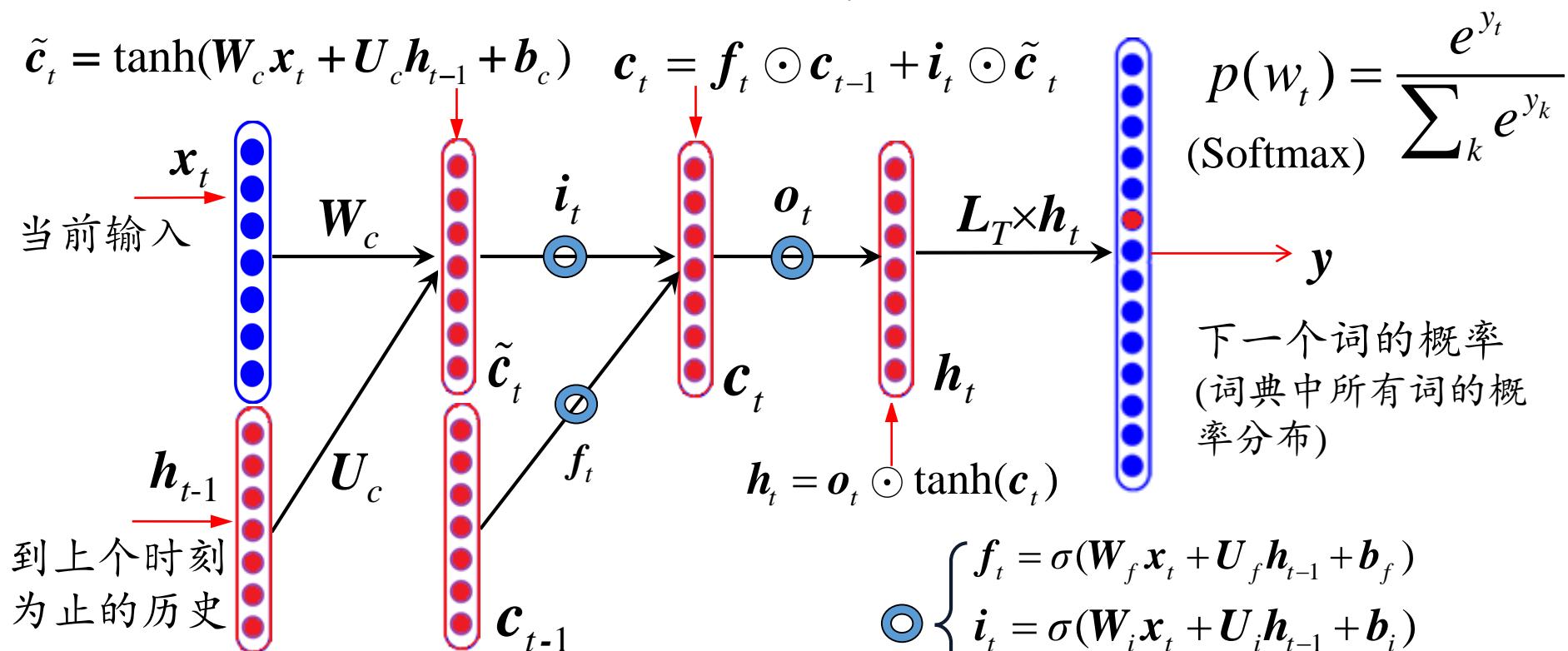


- (1) 利用上一时刻的外部状态  $h_{t-1}$  和当前时刻的输入  $x_t$ ，计算出三个门  $f_t, i_t, o_t$  和候选状态  $\tilde{c}_t$ ；
- (2) 结合遗忘门  $f_t$  和输入门  $i_t$ ，更新记忆单元  $c_t$ ；
- (3) 结合输出门  $o_t$ ，将内部状态的信息传递给外部状态  $h_t$ 。

# 5. 长短时记忆网络

## ◆ 基于LSTM的语言模型

- 输入: 从开始到  $t-1$  时刻的历史  $\mathbf{h}_{t-1}$ ; 当前位置  $t$  的词向量  $\mathbf{x}_t$ ;
- 输出: 到  $t$  位置时的历史积累  $\mathbf{h}_t$  及其该位置上词的概率。



# 5. 长短时记忆网络

(1) 计算三个门控值

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

(不考虑偏置  $b$ 。)

$$x_3 = [0.2, 0.1]^T \text{ 书}$$

$$x_2 = [0.3, 0.1]^T \text{ 本}$$

$$x_1 = [0.2, 0.2]^T \text{ 这}$$

$$h_0 = [0.0, 0.0]^T$$

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

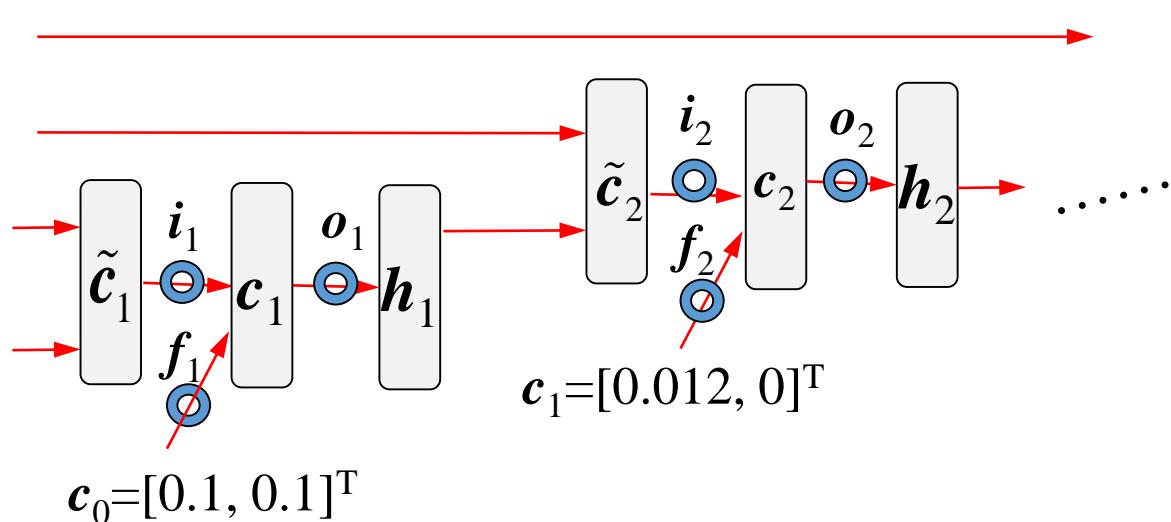
(2) 计算  $\tilde{c}_1, c_1$  和  $h_1$

$$\tilde{c}_1 = \tanh(W_c x_1 + U_c h_0)$$

$$= \tanh(\begin{bmatrix} 0.0 & 0.1 \\ 0.1 & 0.0 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix}) = \begin{bmatrix} 0.02 \\ 0.02 \end{bmatrix}$$

$$c_1 = f_1 \odot c_0 + i_1 \odot \tilde{c}_1 = \begin{bmatrix} 0.1 \\ 0.0 \end{bmatrix} \odot \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.0 \end{bmatrix} \odot \begin{bmatrix} 0.02 \\ 0.02 \end{bmatrix} = \begin{bmatrix} 0.012 \\ 0 \end{bmatrix}$$

$$h_1 = o_1 \odot \tanh(c_1) = \begin{bmatrix} 0.1 \\ 0.0 \end{bmatrix} \odot \tanh(\begin{bmatrix} 0.012 \\ 0 \end{bmatrix})$$





# 工具与文献

## ◆开源工具

- LSTM语言模型(recurrent neural language model with LSTM unit)  
<https://www-i6.informatik.rwth-aachen.de/web/Software/rwthlm.php>
- LSTM反向传播算法  
<http://arunmallya.github.io/writeups/nn/lstm/index.html#/>

## ◆代表论文

- [1] Sepp Hochreiter, Jürgen Schmidhuber, 1997. Long short-term memory.  
*Neural computation*, 1997, 9(8): 1735-1780.
- [2] Sepp Hochreiter, Jürgen Schmidhuber, 1997. LSTM can Solve Hard Long Time Lag Problems, *Advances in Neural Information Processing Systems*.
- [3] Felix A. Gers and Jürgen Schmidhuber, 2001. LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages. *IEEE Transactions on Neural Networks*. 12 (6): 1333–1340.



# 5. 长短时记忆网络

- 对门控机制的改进（0/1开关）：不但依赖于输入 $x_t$ 和上一时刻的隐状态 $h_{t-1}$ ，还依赖于上一时刻的记忆单元 $c_{t-1}$ ，即：

(1) 遗忘门  $f_t$ :  $f_t = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1} + b_f)$

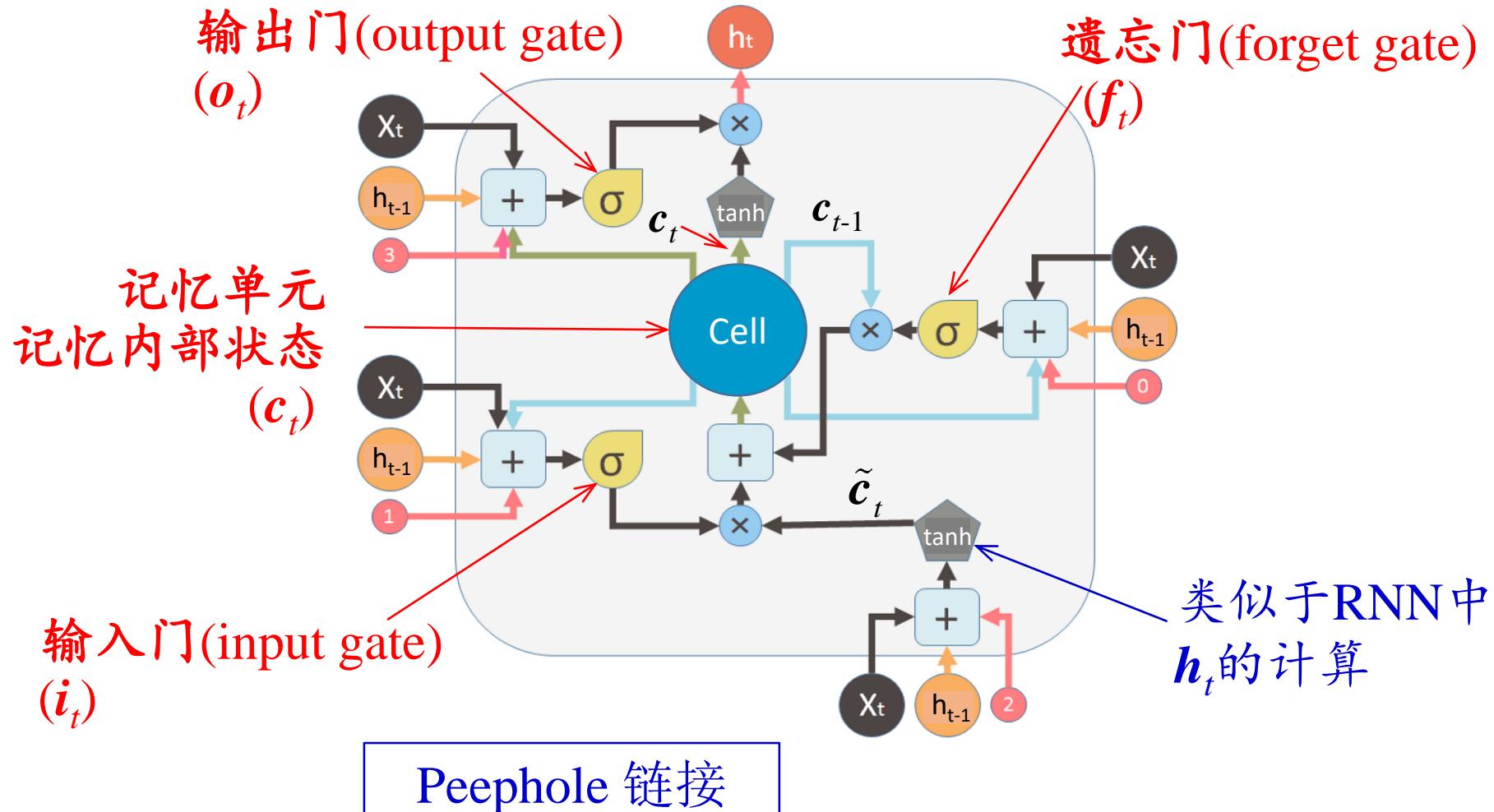
(2) 输入门  $i_t$ :  $i_t = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1} + b_i)$

(3) 输出门  $o_t$ :  $o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t + b_o)$

Gers F A, Schmidhuber J. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*, 2000, 3: 189-194

# 5. 长短时记忆网络

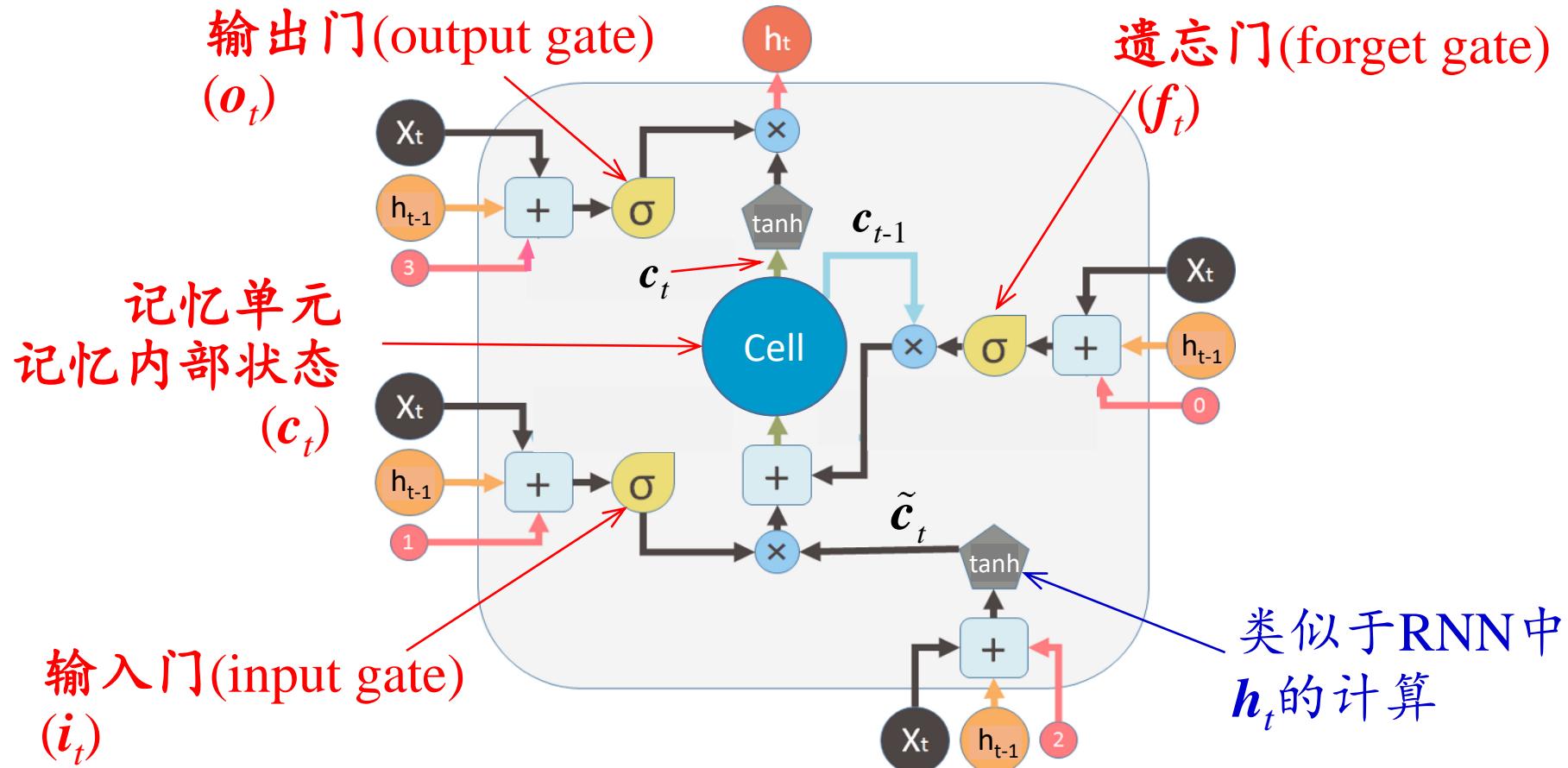
◆ 改进后的LSTM模型结构：



Peephole 链接

# 5. 长短时记忆网络

◆ 改进前的LSTM模型结构 (本章P47):

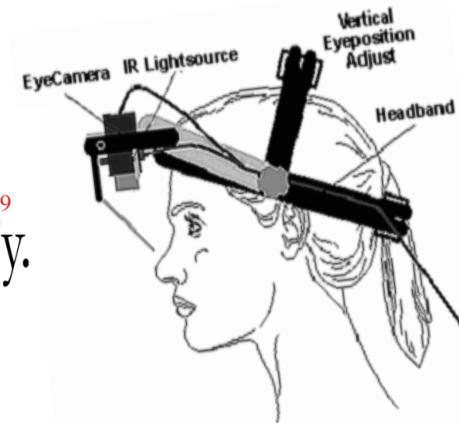
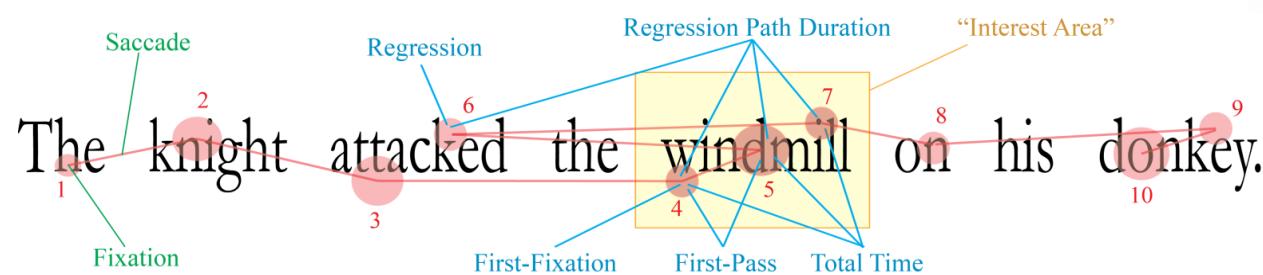


LSTM 又称基于门控机制的循环神经网络。

# 5. 长短时记忆网络

## ◆ 问题分析

给定前面 $t-1$ 个单词( $x_1, x_2, \dots, x_{t-1}$ )预测当前单词 $x_t$ 时，前面每个词的重要程度是一样的。而人的阅读语言句子时对于不同的词汇关注程度是不一样的。



例如：

Reading time/word	the	two	young	sea-lions	took
Rtfpass (第1遍时间)	27.2	138.7	155.5	<b>314.8</b>	169.3
Rtgopast (回看时间)	27.2	138.7	178.4	<b>426.1</b>	180.9
RTrb (右边界时间)	27.2	138.7	155.5	<b>339.2</b>	169.3



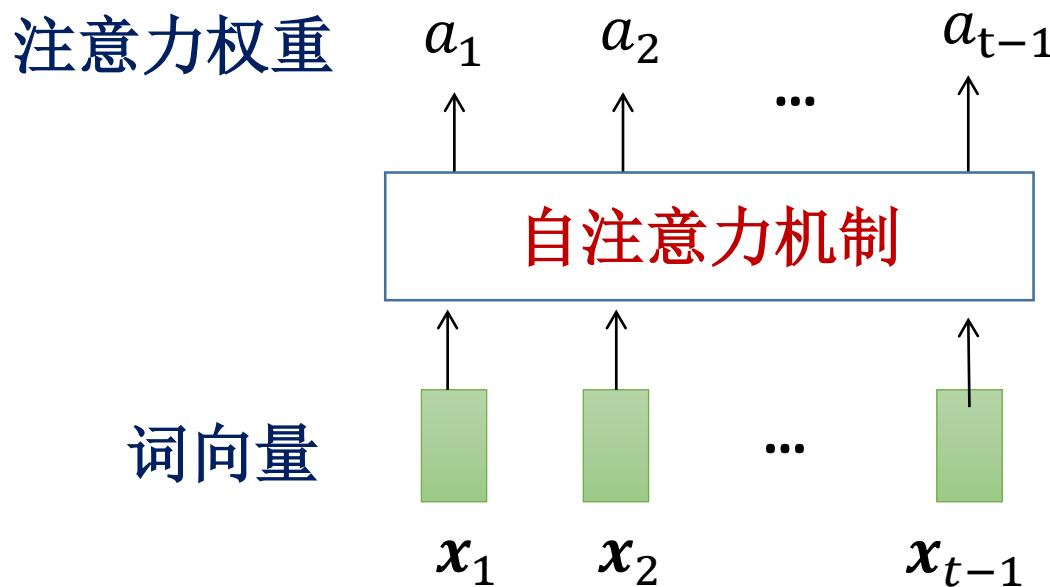
# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

# 6. 自注意力机制

## ◆ 注意力机制的动机

对前 $t-1$ 个词( $x_1, x_2, \dots, x_{t-1}$ )的重要程度设置权重:  
( $a_1, a_2, \dots, a_{t-1}$ )。



# 6. 自注意力机制

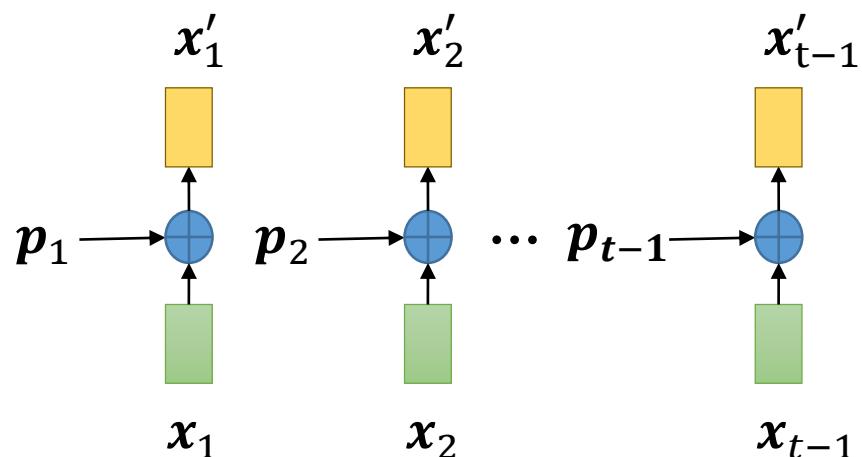
## ◆ 注意力机制的执行过程

**Step 1:** 查询 $t-1$ 个单词的词向量 $(x_1, x_2, \dots, x_{t-1})$ , 添加位置向量 $(p_1, p_2, \dots, p_{t-1})$ , 计算方式(直接相加) :

$$(x'_1, x'_2, \dots, x'_{t-1}) = (x_1, x_2, \dots, x_{t-1}) + (p_1, p_2, \dots, p_{t-1})$$

向量位置如何确定?

- (1)看作参数, 随机初始化,  
根据训练数据进行优化;
- (2)按照规则确定。





# 6. 自注意力机制

按规则确定：

$$\mathbf{p}_{pos} = \begin{bmatrix} y_0 \\ \vdots \\ y_i \\ \vdots \\ y_{d-1} \end{bmatrix} \quad d \text{ (如: } d=500\text{)}$$

桌子 上 有 一 本 书  
1 2 3 4 5 6

其中， $pos$ 是位置， $d$ 是总的维度数。 $2i$ 或 $2i+1$ 分别表示第 $2i$ 或者 $2i+1$ 维元素， $0 \leq i \leq d/2-1$ 。

$$y_{2i} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$y_{2i+1} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$pos = 1, 2, \dots, 6$$

We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset  $k$ ,  $PE_{pos+k}$  (Positional Embeddings) can be represented as a linear function of  $PE_{pos}$ . [Vaswani et al., 2017]



# 6. 自注意力机制

桌子 上 有 一 本 书  
pos: 1 2 3 4 5 6

↑                      ↑  
                 $+k$

$$y_{pos,2i} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$y_{pos,2i+1} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$\mathbf{p}_{pos+k} : \quad y_{pos+k,2i} = \sin\left(\frac{pos+k}{10000^{(2i/d)}}\right)$$

$$y_{pos+k,2i+1} = \cos\left(\frac{pos+k}{10000^{(2i/d)}}\right)$$

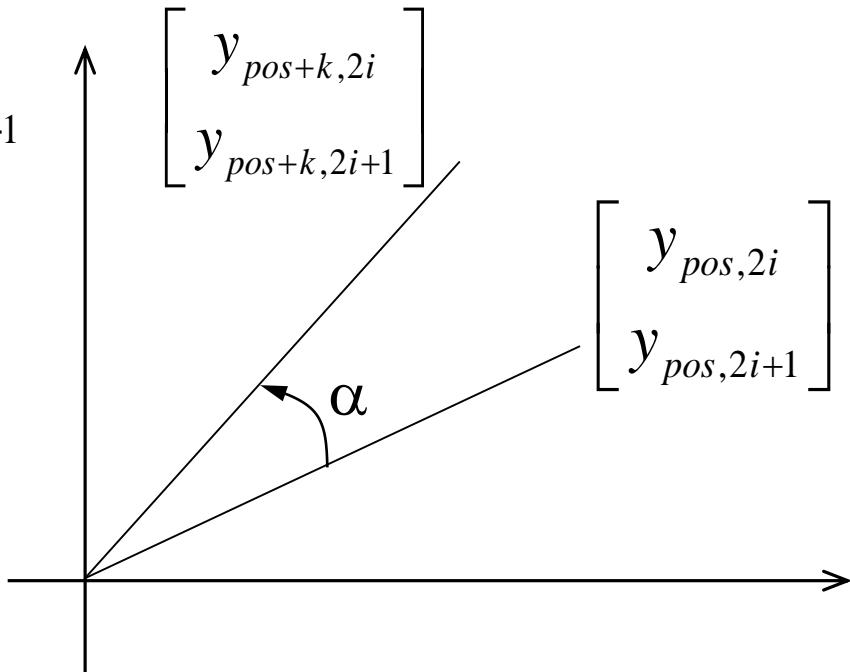
$$\mathbf{p}_{pos} = \begin{bmatrix} y_0 \\ \vdots \\ y_i \\ \vdots \\ y_{d-1} \end{bmatrix}$$

# 6. 自注意力机制

$$y_{pos+k,2i} = \cos \alpha \cdot y_{pos,2i} + \sin \alpha \cdot y_{pos,2i+1}$$

$$y_{pos+k,2i+1} = \cos \alpha \cdot y_{pos,2i+1} - \sin \alpha \cdot y_{pos,2i}$$

其中,  $\alpha = \frac{k}{10000^{2i/d}}$



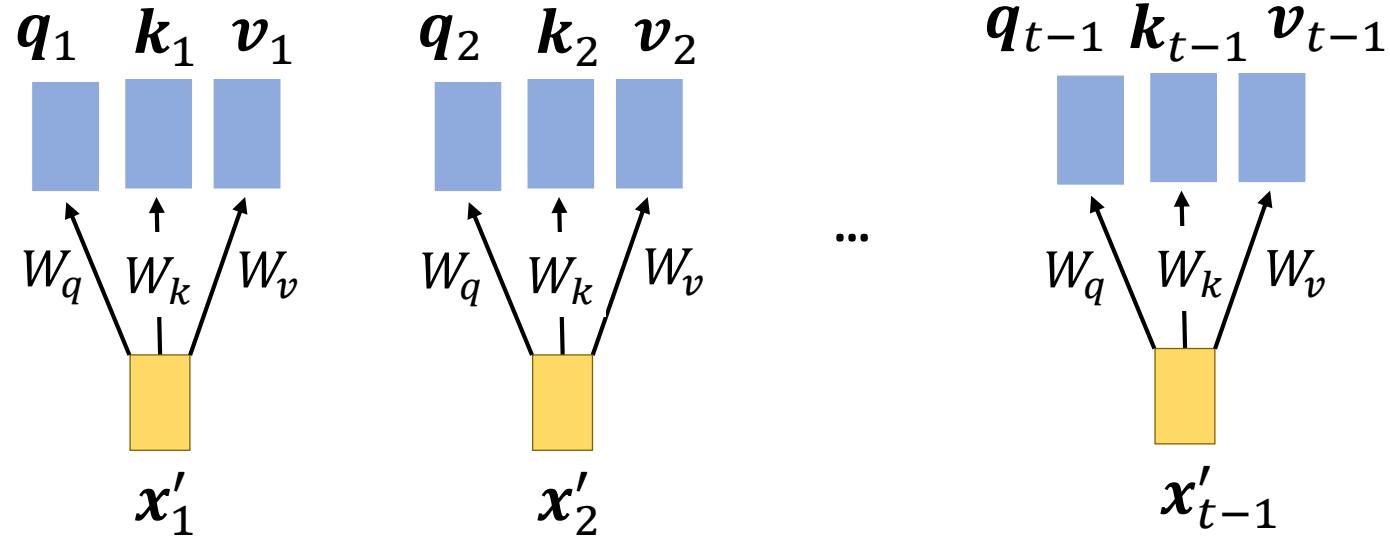
$$\begin{pmatrix} y_{pos+k,2i} \\ y_{pos+k,2i+1} \end{pmatrix} = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix}^T \begin{pmatrix} y_{pos,2i} \\ y_{pos,2i+1} \end{pmatrix} \text{ (矩阵)}$$

$$= \begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} y_{pos,2i} \\ y_{pos,2i+1} \end{pmatrix} = \begin{pmatrix} y_{pos,2i} \cdot \cos \alpha + y_{pos,2i+1} \cdot \sin \alpha \\ y_{pos,2i+1} \cdot \cos \alpha - y_{pos,2i} \cdot \sin \alpha \end{pmatrix}$$

# 6. 自注意力机制

**Step 2:** 根据每个单词的表示( $x'_1, x'_2, \dots, x'_{t-1}$ )，计算对应的查询向量 $q$ 、键向量 $k$ 和值向量 $v$ ，计算公式：

$$q_i = W_q x'_i \quad k_i = W_k x'_i \quad v_i = W_v x'_i$$



查询向量 $q$ : 用于查询其它位置（包括自己）的向量；键向量 $k$ : 被查询的值，用于与 $q$ 向量运算；值向量 $v$ : 词义本身（当然， $q$ 和 $k$ 值也表达词义）。

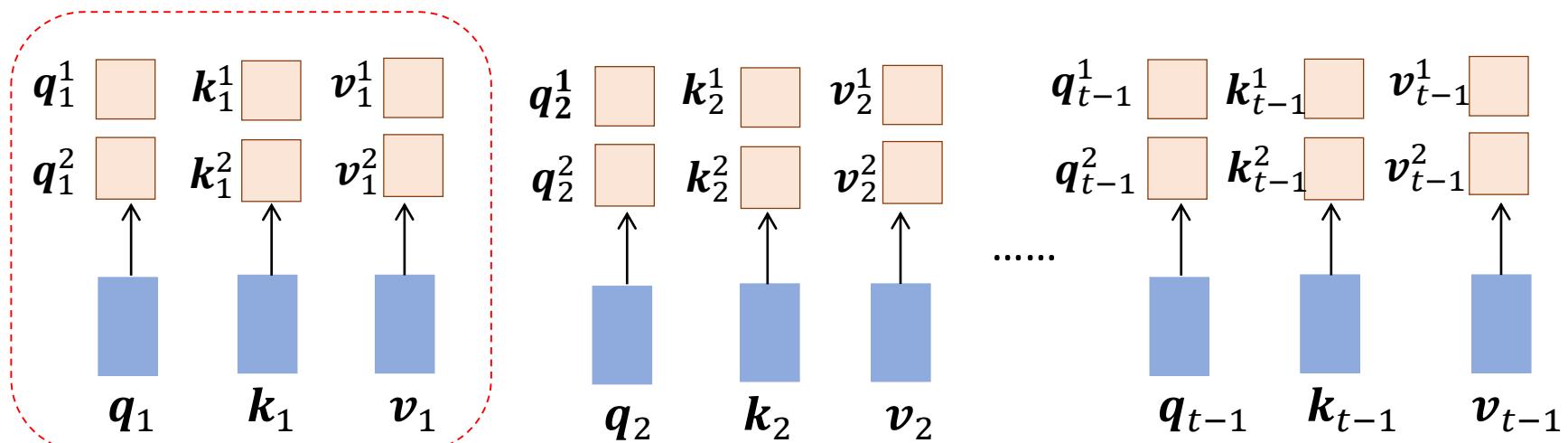
# 6. 自注意力机制

**Step 3:** 将查询向量 $q$ 、键向量 $k$ 和值向量 $v$ 分解为多个向量（称为“**多头**”(multi-head)），分解过程如下(以两个头为例)：

$$q_i = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.5 \end{bmatrix} \quad q_i^1 = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} \quad q_i^2 = \begin{bmatrix} 0.2 \\ 0.5 \end{bmatrix}$$

$$k_i \quad k_i^1 \quad k_i^2$$

$$v_i \quad v_i^1 \quad v_i^2$$

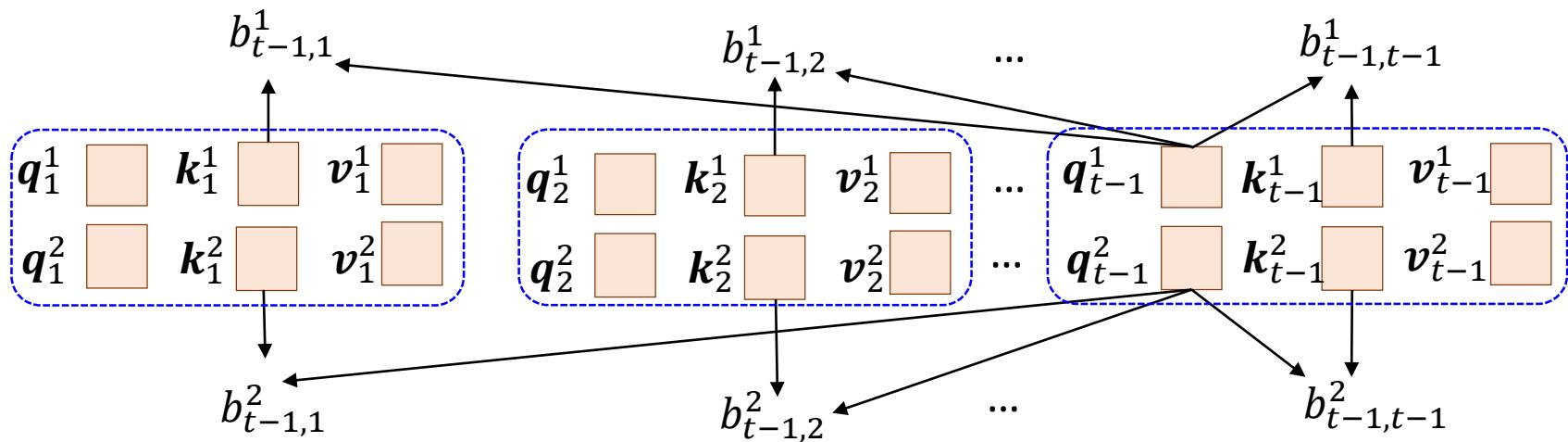


$x_1 \rightarrow x_1' \rightarrow q_1, k_1, v_1 \rightarrow$  拆分成多头

# 6. 自注意力机制

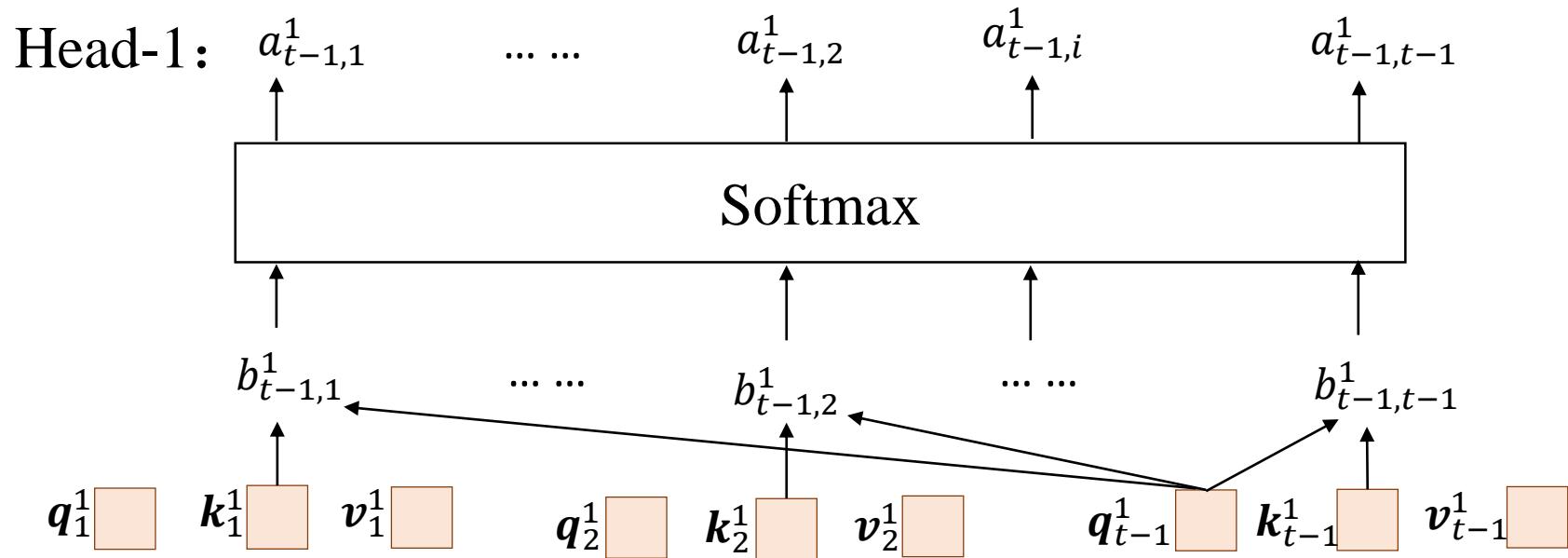
**Step 4:** 对每个头分别计算其注意力权重:  $b = \frac{q^T k}{\sqrt{d_k}}$ 。其中,  $d_k$ 为 $q$ 和 $k$ 的维度数。在上一页中, 每个头的 $q$ 和 $k$ 为2维的向量, 那么分母就是 $\sqrt{2}$ 。

在语言模型中,  $q$ 取 $t-1$ 时刻的查询向量,  $k$ 为前 $t-1$ 的所有键向量。



# 6. 自注意力机制

**Step 5:** 将所有注意力权重，归一化处理（Softmax函数）。



(图中仅画出头1的计算过程)

$$a_{t-1,i}^1 = \text{Softmax}(b_{t-1,i}^1)$$

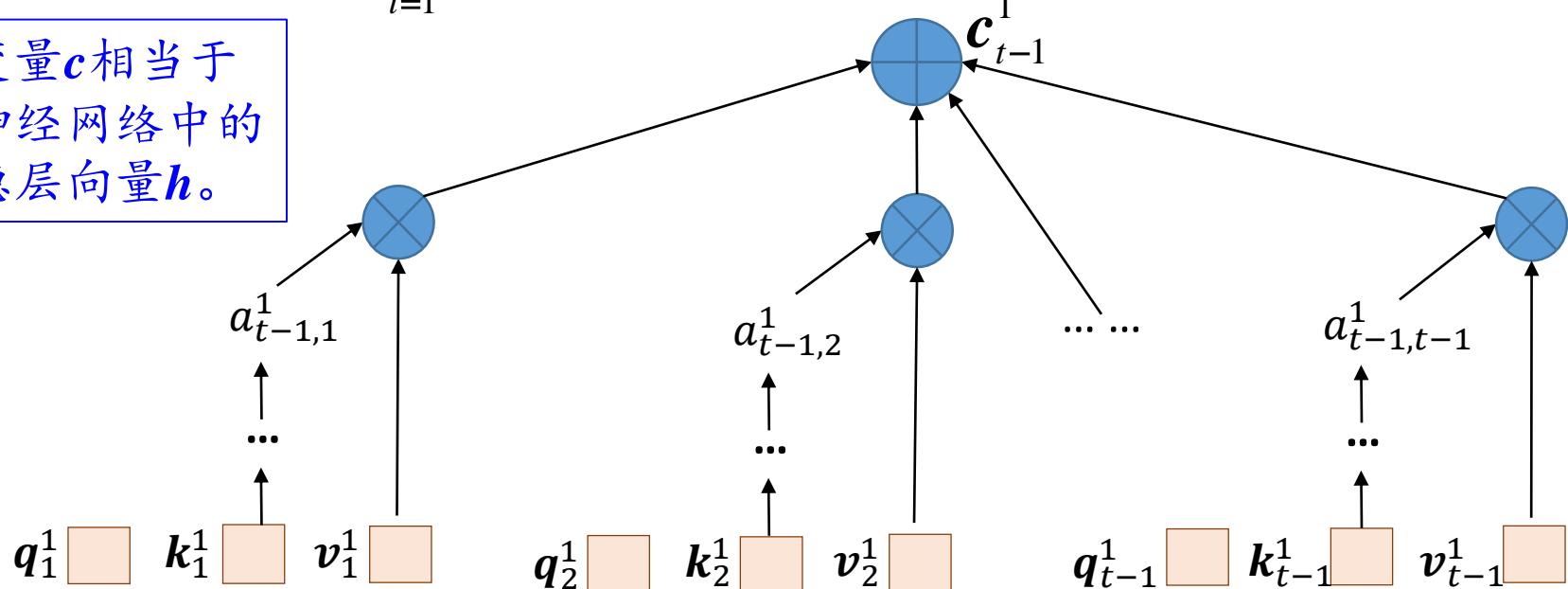
# 6. 自注意力机制

**Step 6:** 根据归一化的注意力权重 $a$ , 聚合前 $t-1$ 个词的信息:

$$\text{Head-1: } \mathbf{c}_{t-1}^1 = \sum_{i=1}^{t-1} a_{t-1,i}^1 v_i^1 = a_{t-1,1}^1 v_1^1 + a_{t-1,2}^1 v_2^1 + \cdots + a_{t-1,t-1}^1 v_{t-1}^1$$

$$\text{Head-2: } \mathbf{c}_{t-1}^2 = \sum_{i=1}^{t-1} a_{t-1,i}^2 v_i^2 = a_{t-1,1}^2 v_1^2 + a_{t-1,2}^2 v_2^2 + \cdots + a_{t-1,t-1}^2 v_{t-1}^2$$

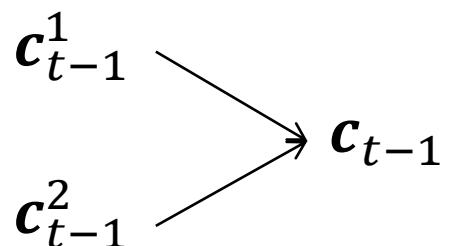
变量 $c$ 相当于  
神经网络中的  
隐层向量 $h$ 。



(图中仅画出头1的计算过程)

# 6. 自注意力机制

**Step 7:** 将不同头 (Head-1, Head-2 ...) 的语义向量进行拼接，拼接后的语义向量则包含了前 $t-1$ 个单词的信息。

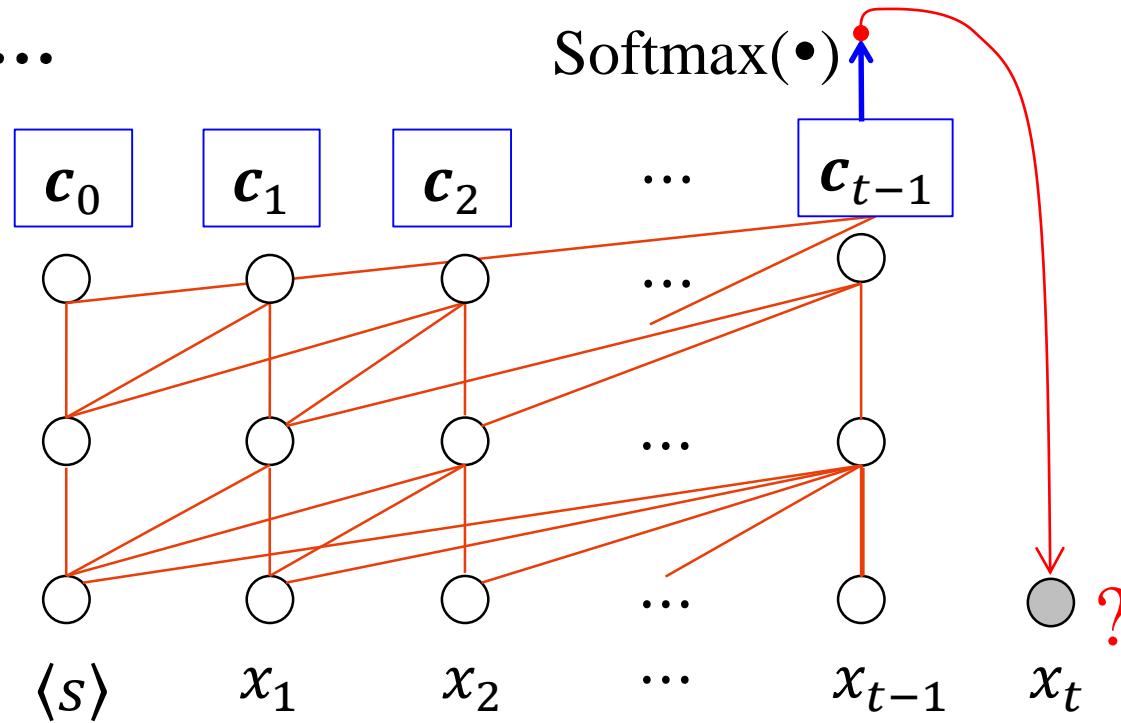


**Step 8:** 利用 $c_{t-1}$ 对所有的单词进行预测打分（内积和Softmax函数），得到最终的预测结果。

$$p(x_t | c_{t-1}) = \text{Softmax}\left(L_T^T \cdot c_{t-1}\right)$$

# 6. 自注意力机制

◆ 相当于…



◆ 代表论文：

- A. Vaswani et al. [Attention Is All You Need](#). In *Proceedings of the 31<sup>st</sup> Conference on Neural Information Processing Systems (NIPS'2017)*, December 4-7, 2017, Long Beach, CA, USA, Pages 6000–6010



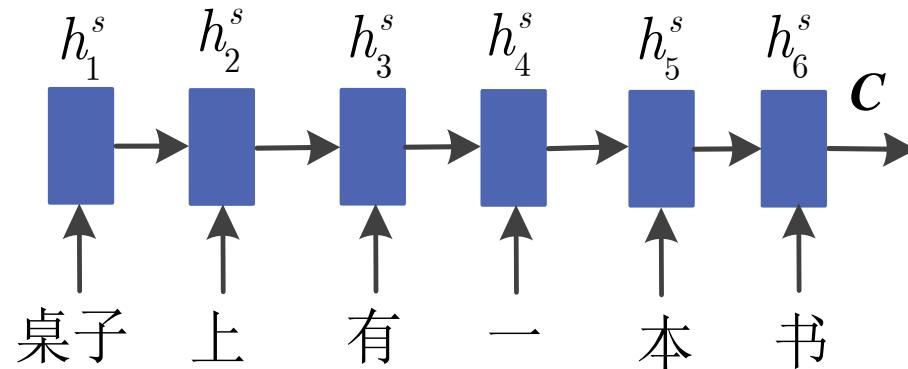
# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

# 7. Transformer 架构

## ◆ 分析 RNN / LSTM 的问题

- 在 RNN/ LSTM 中，词汇间的语义关系需要逐词传递，远程依赖关系的学习能力不足；
- 前面单词编码结束后，才能编码后续单词，无法实现并行处理。



# 7. Transformer 架构

为了缓解上述问题，谷歌公司于2017年提出了基于自注意力机制的端到端(End-to-End, E2E)的转换框架，并将其命名为Transformer。  
架构结构如右图所示。

A. Vaswani et al., [Attention Is All You Need.](#)  
*Proc. of NIPS'2017*, pp 6000–6010

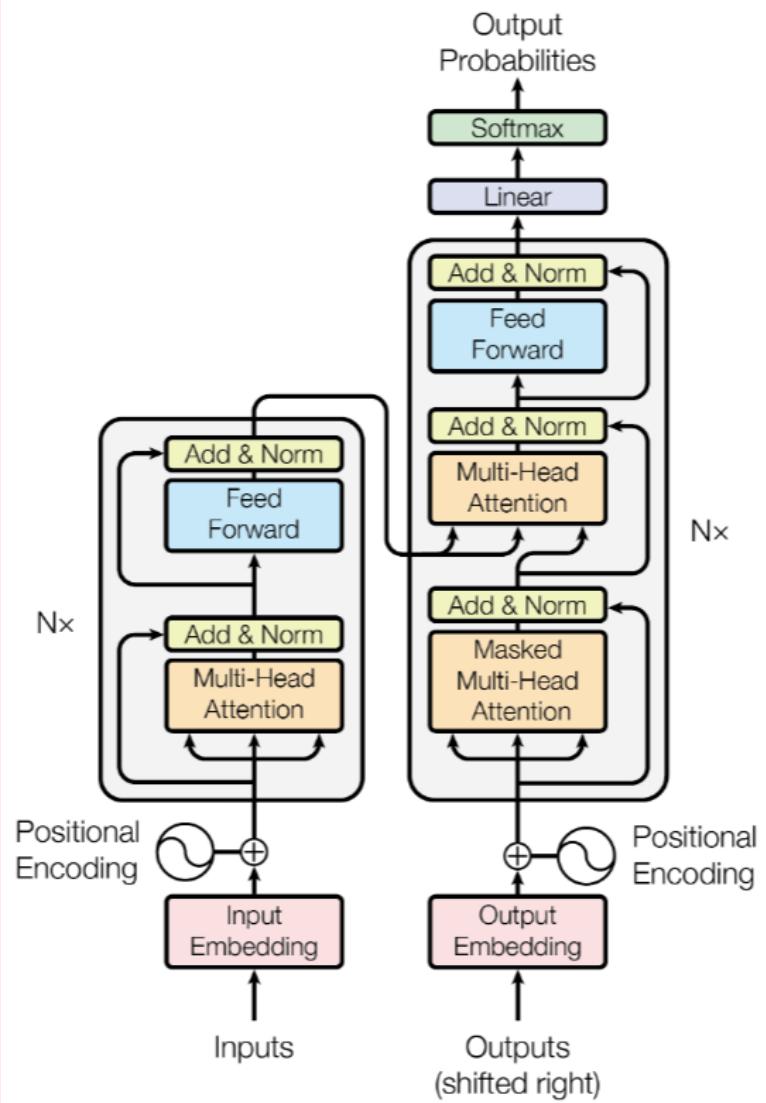


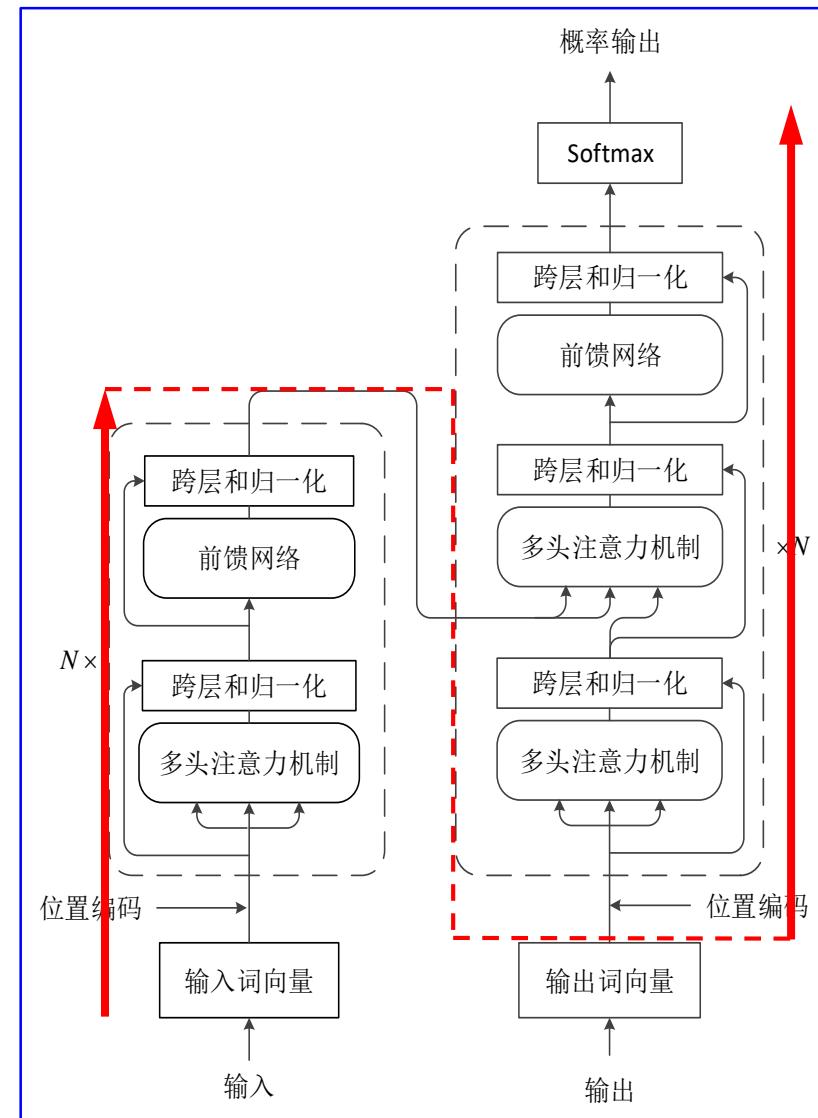
Figure 1: The Transformer - model architecture.

# 7. Transformer 架构

为了缓解上述问题，谷歌公司于2017年提出了基于自注意力机制的端到端(End-to-End, E2E)的转换框架，并将其命名为Transformer。

架构结构如右图所示。

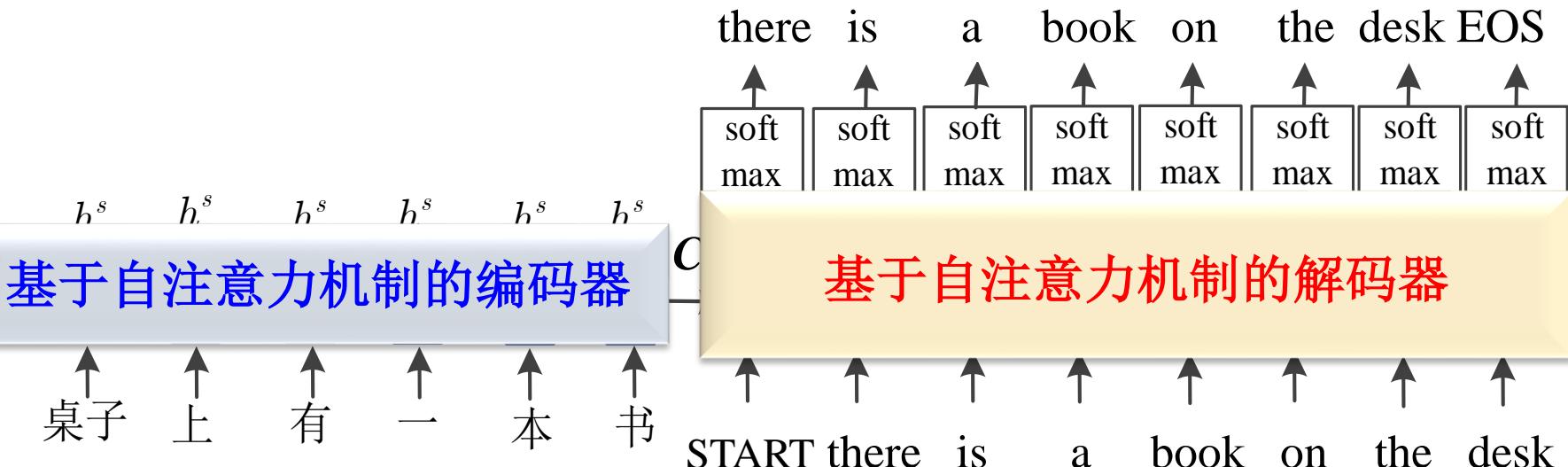
- Machine translation
- Human-dialogue system
- Q&A
- Sentiment/ emotion analysis
- Text classification
- Automatic summarization
- Automatic writing / language generation
- Speech recognition
- Image caption
- .....



# 7. Transformer 架构

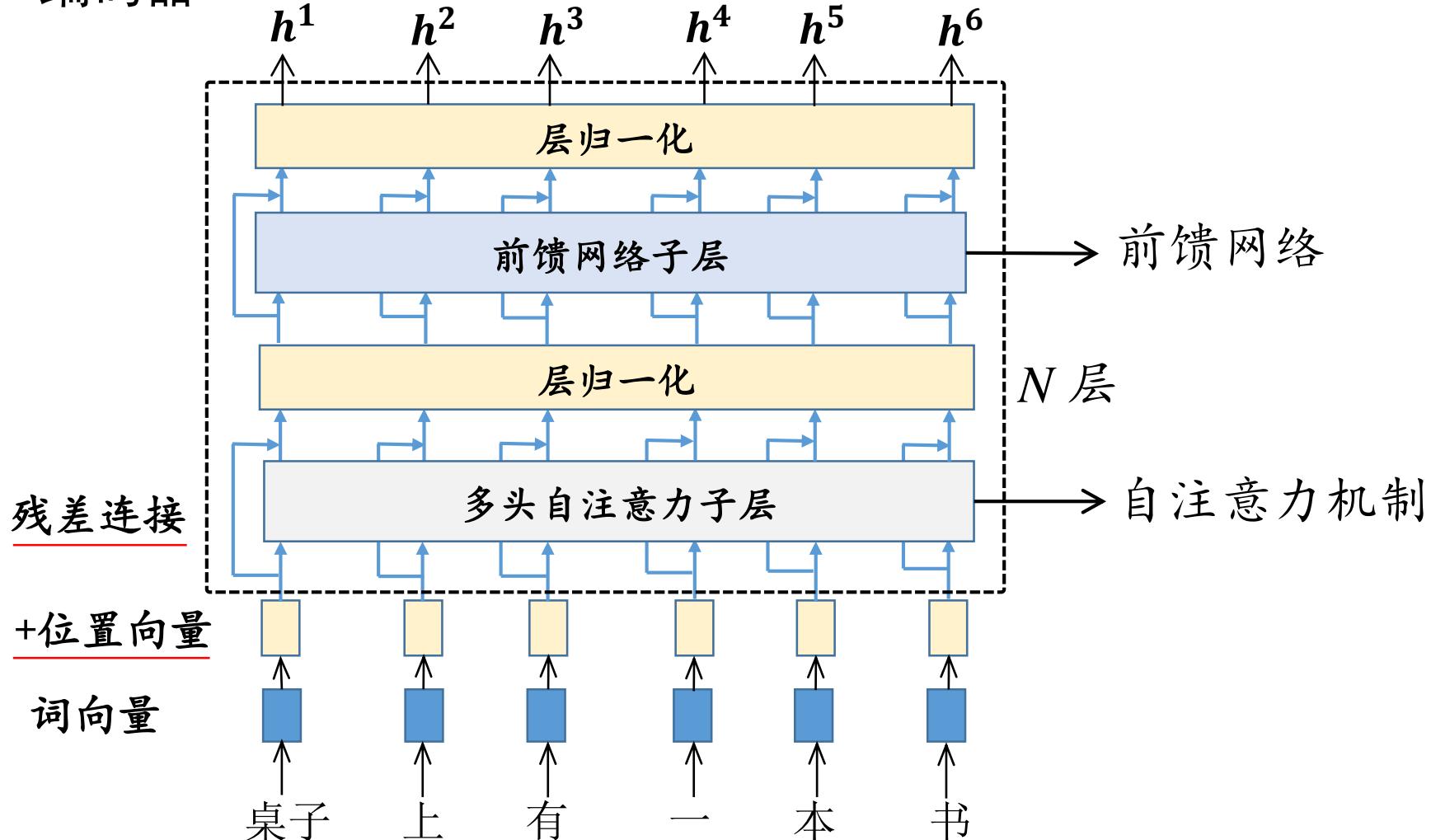
## ◆ 基本思路

(以机器翻译为例)



# 7. Transformer 架构

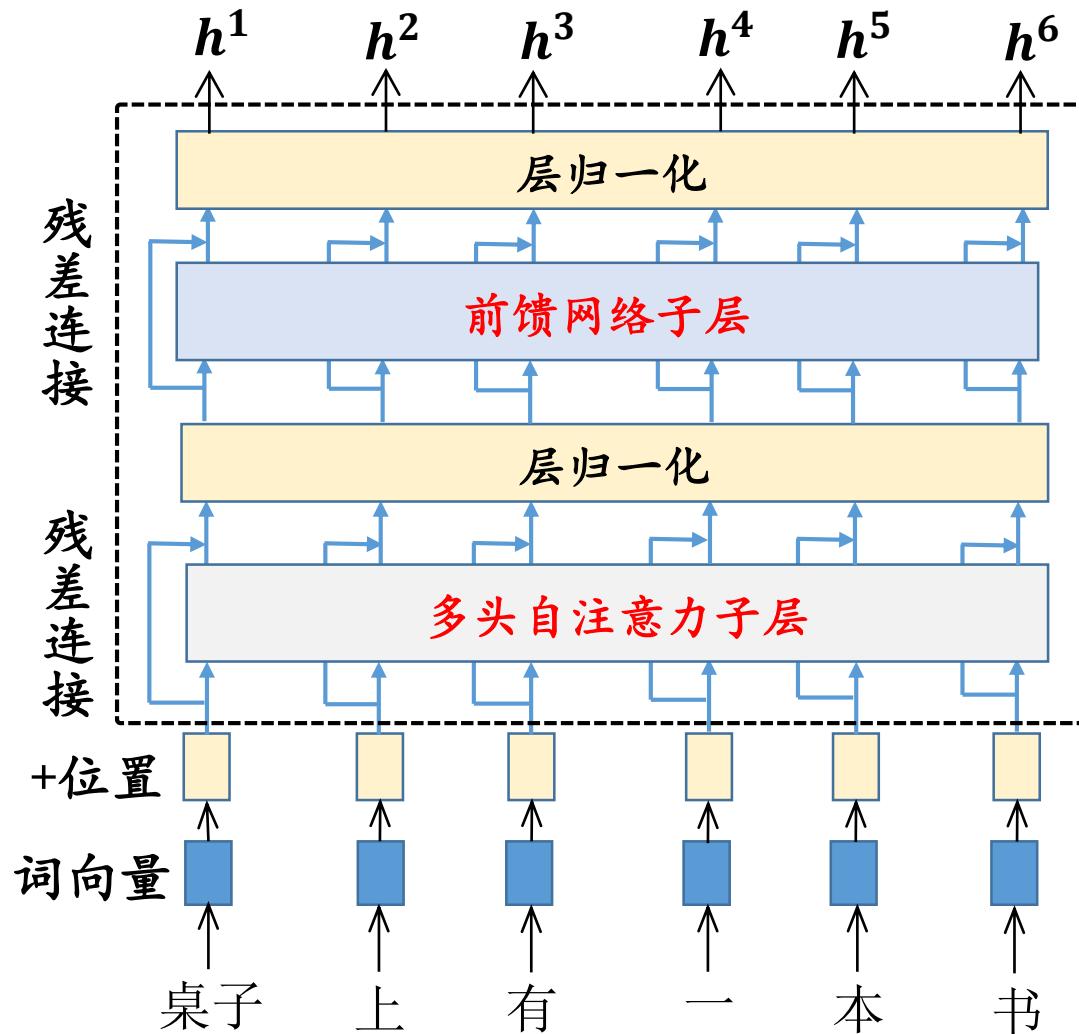
## ● 编码器



# 7. Transformer 架构

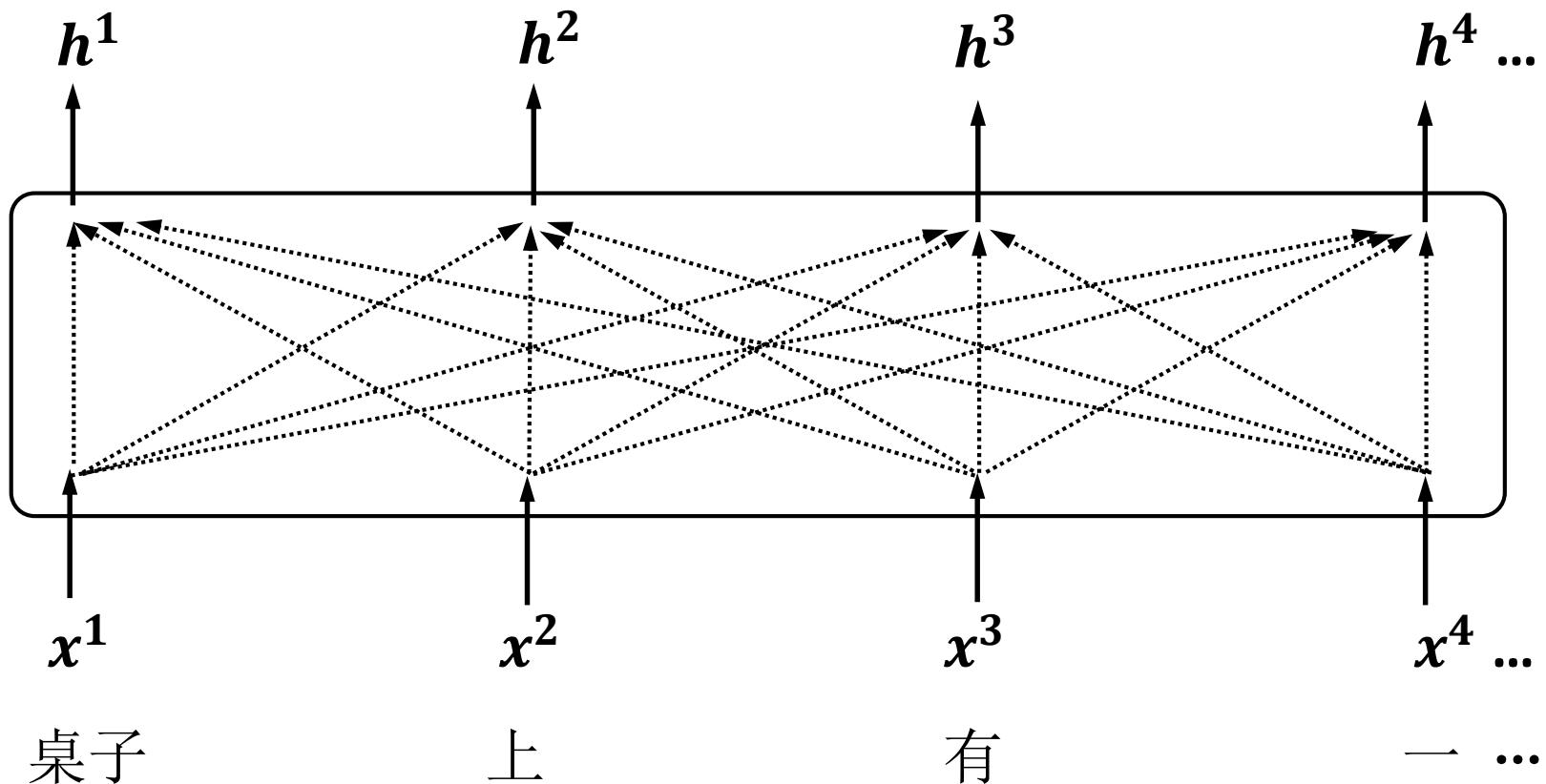
## ● 编码器

- 编码器由 $N$ 层组成，每一层包括2个子层：
  - ①多头自注意力子层；
  - ②前馈网络子层。
- 每个子层之间采用了**残差连接**(residual connection)和**层归一化**(layer normalization)方法。



# 7. Transformer 架构

➤ 编码器的结构：

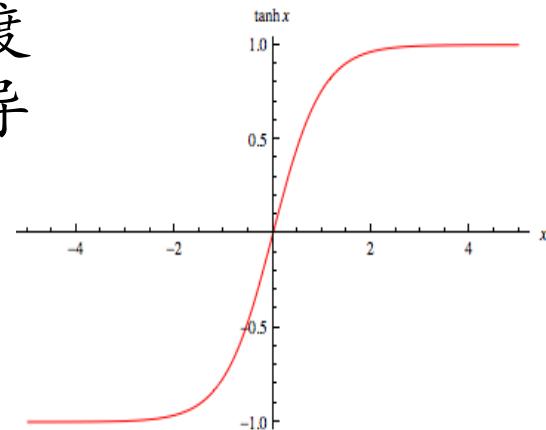


# 7. Transformer 架构

## ➤ 残差连接:

- 目的: 残差网络是深度学习中的一个重要方法, 能够将误差信号不经过中间权重矩阵变换直接传播到低层, 缓解**梯度弥散(gradient diffusion)**问题。

使用反向传播算法时, 随着传播深度的增加, 梯度的变化幅度可能会减小, 导致权重更新非常缓慢, 不能有效地学习。



$$z^{(l)} = W^{(l)} \cdot \alpha^{(l-1)} + b^{(l)}$$

$$\alpha^{(l)} = f_l(z^{(l)})$$

$$x = \alpha^{(0)} \rightarrow z^{(1)} \rightarrow \alpha^{(1)} \rightarrow z^{(2)} \rightarrow \cdots \rightarrow \alpha^{(L-1)} \rightarrow z^{(L)} \rightarrow \alpha^{(L)} = \phi(x; W, b))$$

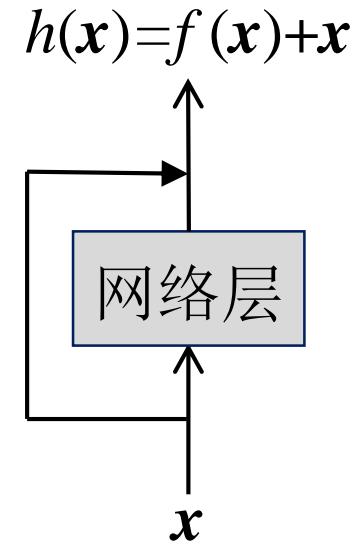


# 7. Transformer 架构

- 解决方法：

① **正向计算时**，将输入 $x$ 与网络层变换 $f(x)$ 后直接相加，作为该层的最终输出结果，如图所示：

② **反向传播时**，误差对输入 $x$ 的导数  $\frac{\partial E}{\partial x}$  将增加一个常数项1，从而将误差对输出 $h(x)$ 的导数  $\frac{\partial E}{\partial h}$  直接传递给输入 $x$ ，以缓解梯度弥散问题。



误差 $E$ 对输入 $x$ 的导数  $\rightarrow \frac{\partial E}{\partial x} = \frac{\partial E}{\partial h} \times \frac{\partial h}{\partial x}$  常数项1

$$= \frac{\partial E}{\partial h} \times \left( \frac{\partial f}{\partial x} + 1 \right) = \boxed{\frac{\partial E}{\partial h} \times \frac{\partial f}{\partial x}} + \boxed{\frac{\partial E}{\partial h}}$$

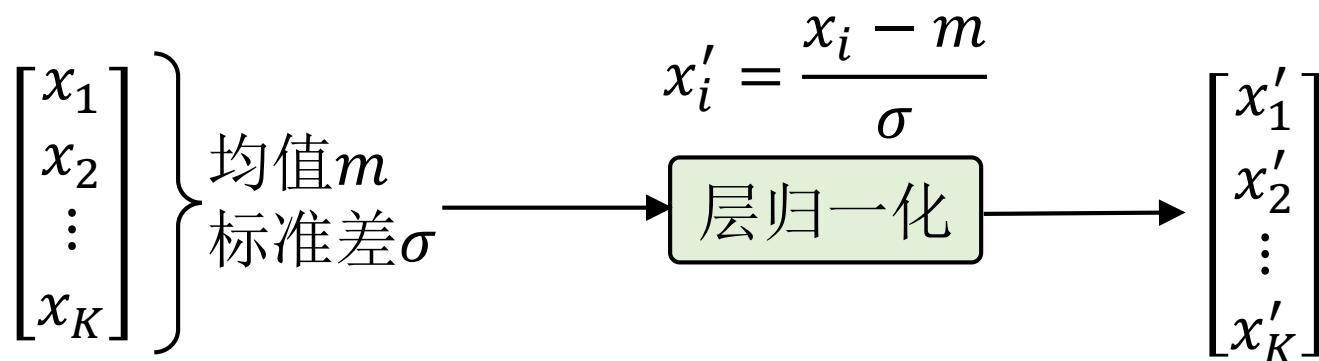
将误差对输出导数直接传递给输入 $x$ 。

经过网络层后的导数连续乘法可能会导致梯度弥散

# 7. Transformer 架构

## ➤ 层归一化

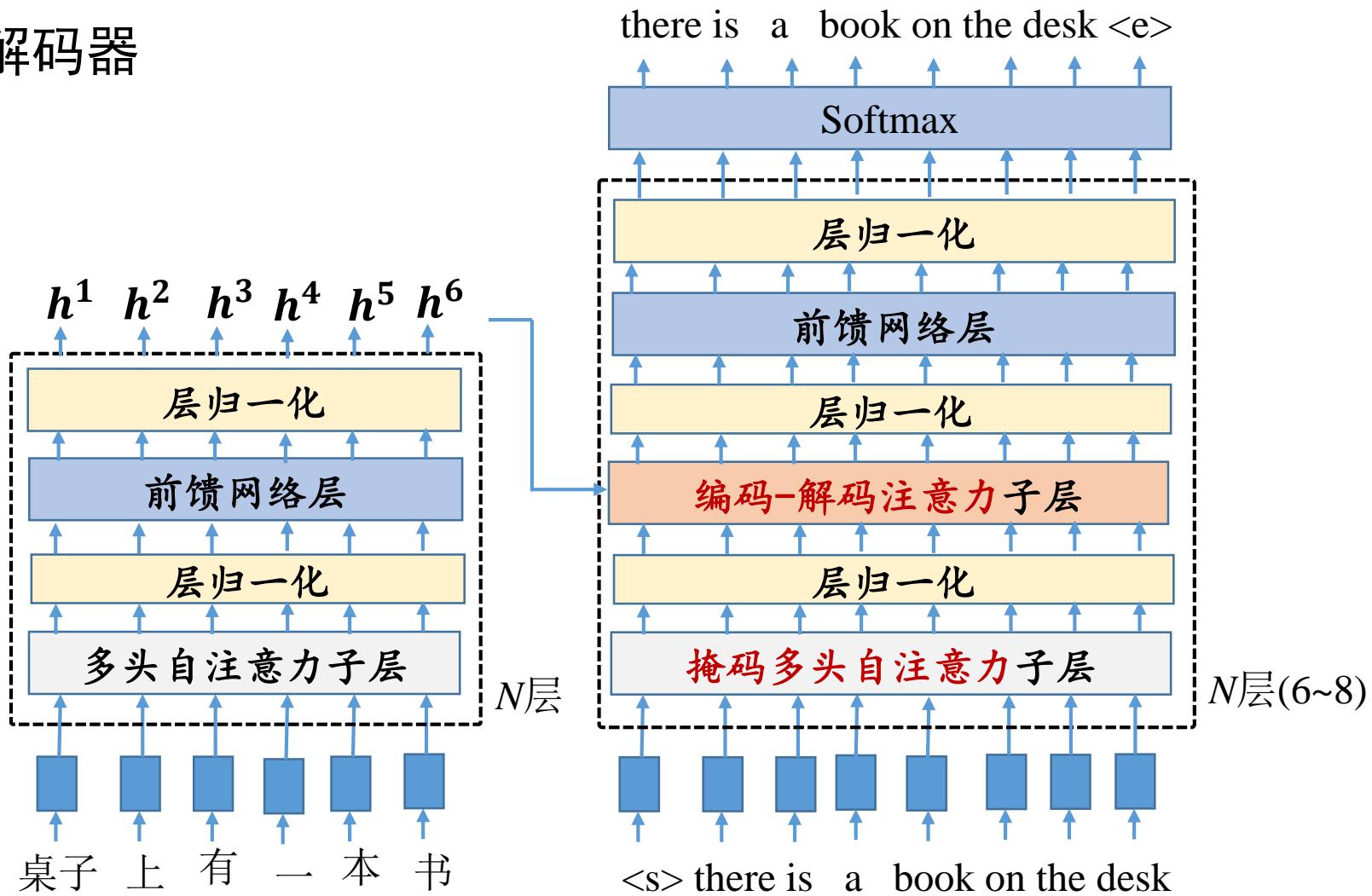
- 目的：使得网络中每层输入数据的分布相对稳定，加速模型学习速度。
- 方法：



Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton. 2016. [Layer normalization](#).  
*arXiv preprint arXiv:1607.06450*, 2016.

# 7. Transformer 架构

## ● 解码器



注：图中未标出残差连接。



# 7. Transformer 架构

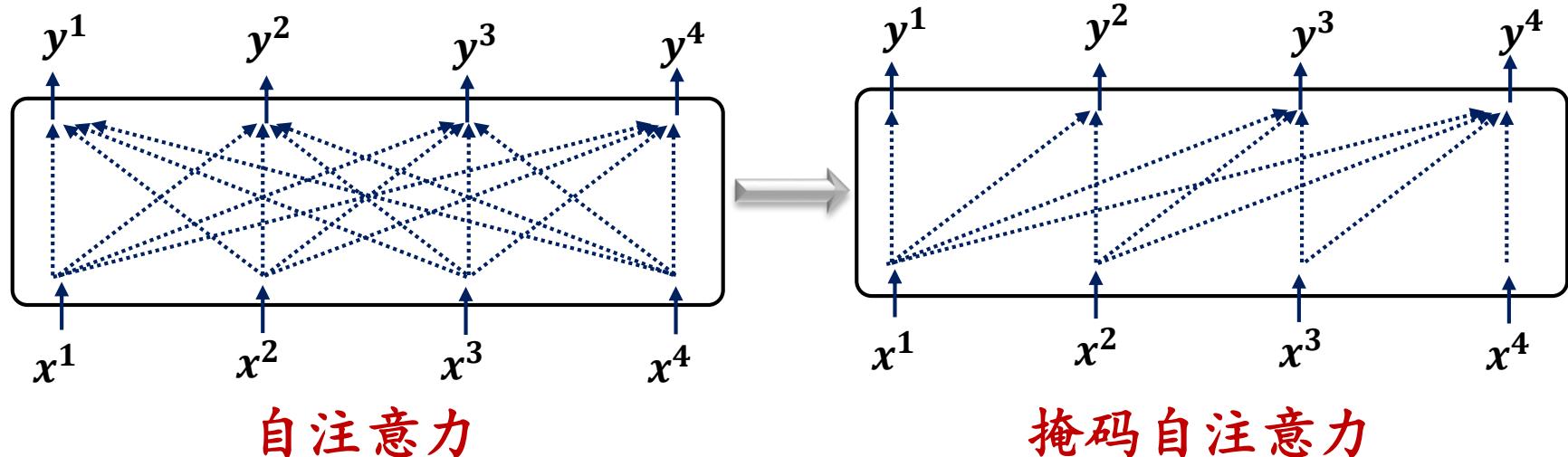
## ● 解码器

- 解码器由 $N$ 层组成，每一层包括3个子层：
  - ①掩码多头自注意力子层；
  - ②编码-解码注意力子层；
  - ③前馈网络子层。
- 每个子层之间同样采用了**残差连接**(residual connection)和**层归一化**(layer normalization)方法。

# 7. Transformer 架构

## ● 解码器

- ✧ **掩码多头自注意力子层**是指对于某一个目标端单词，只与**之前的**单词进行注意力计算，即把后面的单词掩盖。
- ✧ **测试阶段与训练阶段的区别：**在训练阶段，标准译文是已知的，而在测试阶段，模型仅已知之前的单词（模型自身预测得到）。
- ✧ 为了保证模型在训练阶段和测试阶段保持一致，因此在测试阶段，需要将后续的单词掩盖。





# 7. Transformer 架构

## ● 解码器

- ◆ 编码-解码注意力子层是指对于某一个目标端单词，只与源端单词的隐层状态进行注意力计算。
- ◆ 编码-解码注意力子层能够在解码过程中利用源语言信息生成单词，具体实现方法：  
    注意力计算中查询向量( $q$ )来自目标端单词的隐层状态，而键向量( $k$ )和值向量( $v$ )来自源端隐层状态。



# 7. Transformer 架构

## ● Transformer 框架的优势：

- (1) 自注意力机制能够直接建模任意两个词汇之间的语义关系，提升对于词汇间远程依赖关系的建模能力。
- (2) 模型在训练阶段，可以同时对所有源端单词和目标端单词进行编码和解码，提升模型并行计算能力。

Transformer不仅用于机器翻译任务，而且在自然语言处理的其它任务、计算机视觉和图像处理、语音识别等众多任务中均得到了广泛应用。



# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录

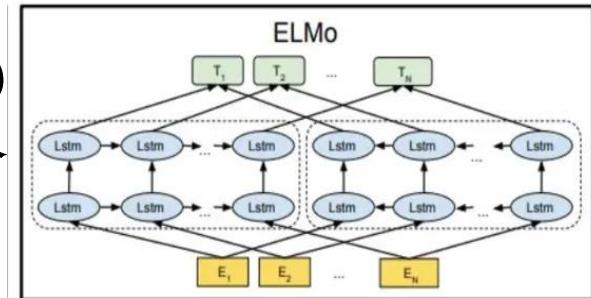


# 8. 预训练语言模型

## ◆ 三个代表性预训练语言模型(Pre-training LM, PLM)

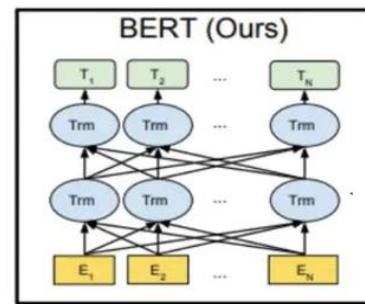
### ➤ ELMo [Peters et al., 2018](AllenAI&WU)

- 基于双向LSTM，实现了一词多义区分
- 不足：特征抽取器；双向拼接融合



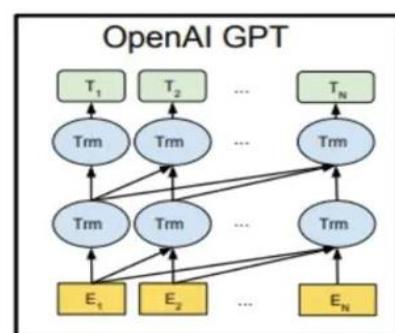
### ➤ BERT [Devlin et al., 2019](Google)

- 采用Transformer；一体化融合特征
- 上下文全向预测，深层特征融合



### ➤ GPT [Radford et al., 2018](OpenAI)

- Generative Pre-Trained Transformer
- 采用Transformer；单向上下文特征





# 8. 预训练语言模型

## ➤ 代表性论文

[1] **ELMo** — Embeddings from Language Models:

M. Peters et al. Deep contextualized word representations. *Proc. NAACL-HLT 2018* (<https://allenai.org/allennlp/software/elmo>)

[2] **BERT** — Bidirectional Encoder Representations from Transformers:

J. Devlin et al. BERT: pre-training of deep bidirectional transformers for language understanding. *Proc. NAACL-HLT 2019* (<https://github.com/google-research/bert>)

[3] **GPT** — Generative Pre-trained Transformer:

A. Radford et al. Improving language understanding by generative pre-training. *Technical report, OpenAI 2018, GPT 1.0*

A. Radford et al. Language models are unsupervised multitask learners. *OpenAI Blog, 2019, GPT 2.0*

T. Brown et al. Language models are fewshot learners. *Proc. NeurIPS 2020, 33, 1877-1901, GPT 3.0* (<https://github.com/openai/gpt-3>)



# 8. 预训练语言模型

## ◆ GPT 模型

### ● 模型训练

在整个训练集上（含 $T$ 个句子）使所有句子的条件最大似然概率最大化：

$$L_1 = \sum_{t=1}^T \sum_{j=1}^{n+1} \log \left( x_j^{(t)} \mid \langle s \rangle, x_1^{(t)}, x_2^{(t)}, \dots, x_{j-1}^{(t)}; \theta \right)$$

$n$ 是句子的长度， $\langle s \rangle$ 是句子的开始标记（即 $x_0$ 位置）。

### ● 模型推断

基于条件概率  $p(x_{j+1} \mid \langle s \rangle, x_1, \dots, x_j)$  预测下一时刻的“单词” $x_{j+1}$ 。



# 8. 预训练语言模型

## ● GPT 微调

在完成具体的NLP任务时，预训练的GPT作为起点模型，针对具体的任务进行适应性参数微调。

以有监督的文本分类任务为例：有若干训练样本  $(x, y)$ ,  $x$  为给定的文本， $x = (\langle s \rangle x_1 \dots x_j \dots x_n \langle e \rangle)$ ,  $y$  为标签。预训练好的 GPT 将  $x$  作为输入，经过 L 层堆叠的掩码自注意力层的计算，产生最上层的表示： $(\mathbf{h}_{\langle s \rangle}, \mathbf{h}_1, \dots, \mathbf{h}_j, \dots, \mathbf{h}_n, \mathbf{h}_{\langle e \rangle})$ 。最后，模型使用新的线性输出层和 softmax 回归函数对  $\mathbf{h}_{\langle e \rangle}$  进行分类：

$$p(y|x) = p(y|\langle s \rangle, x_1, \dots, x_j, \dots, x_n, \langle e \rangle) = \text{softmax}(\mathbf{h}_{\langle e \rangle} \cdot \mathbf{W}_y)$$



## 8. 预训练语言模型

预训练好的GPT模型参数和线性映射层的参数矩阵 $\mathbf{W}_y$ 将按照最大化似然概率的目标进行微调：

$$L_2 = \sum_{(x,y)} \log p(y | x)$$

为了提升泛化能力，同时加速收敛，GPT 在微调过程中将两个优化目标 $L_1$ 和 $L_2$ 进行组合：

$$L = L_2 + \lambda \times L_1$$

对于输入中包含多个序列的下游任务，GPT 采用简单的序列拼接方式，即将所有的序列拼接在一起，中间用分隔符做记号，得到长序列后与GPT进行匹配。



# 8. 预训练语言模型

## ◆ BERT 模型

- 与 GPT 相比：

- ① 使用双向上下文编码：

$$x = (\langle s \rangle x_1 \dots x_{j-1} x_j x_{j+1} \dots x_n \langle e \rangle)$$

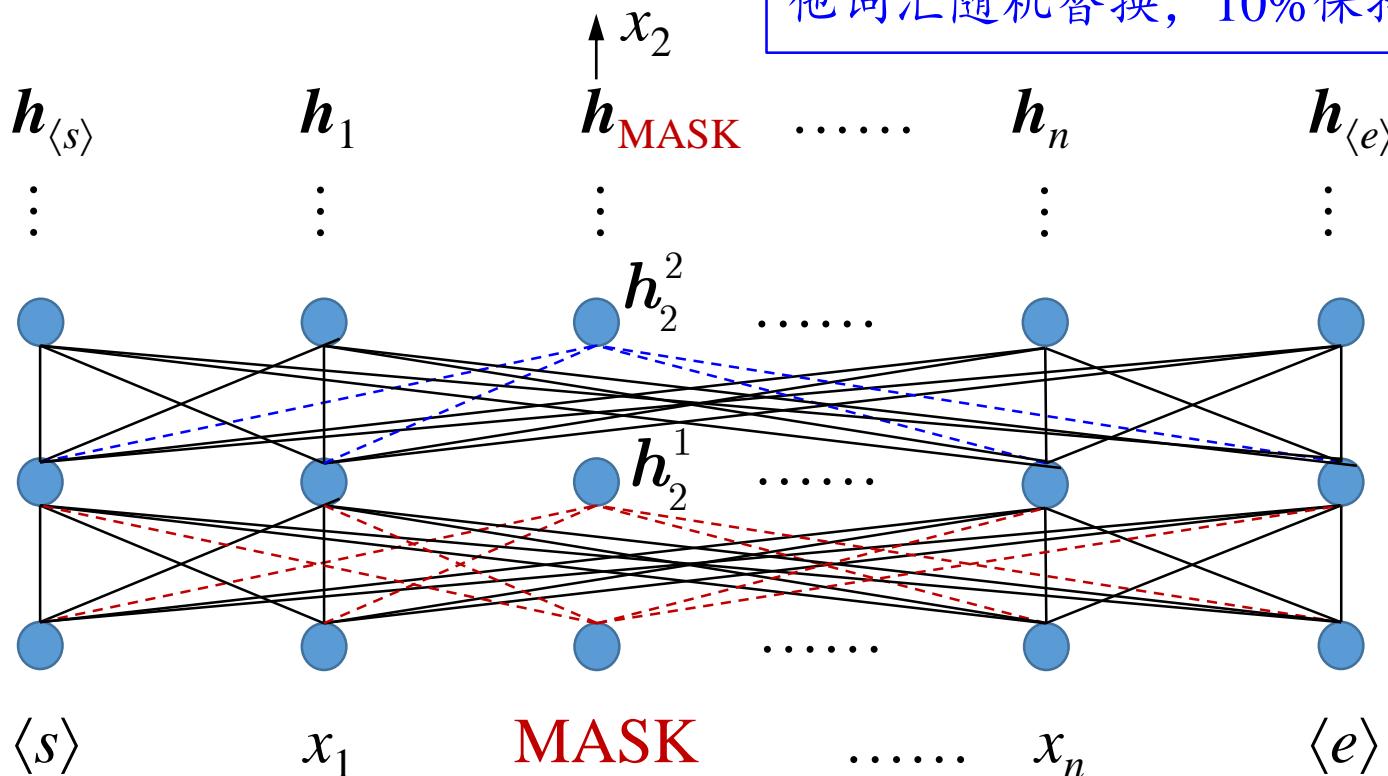

- ② 采用两个无监督的目标函数：掩码语言模型；下一句预测。
  - ③ 使用比GPT使用的更大规模的数据集进行训练：8亿词汇的 BookCorpus 和25亿词汇的英文维基百科。

# 8. 预训练语言模型

## ● 模型训练

### ➤ 掩码语言模型

BERT 随机地掩盖了每个序列中全部词汇的15%，对于15%被掩盖词汇中的80%使用MASK替换，10%使用其他词汇随机替换，10%保持不变。





# 8. 预训练语言模型

## ➤ 两个邻近的句子拼接

- 目的: 解决两个输入序列问题, 例如:

A: 如果明天下雨。B: 运动会延期一周进行。

句子A蕴含句子B吗?

- 方法: 对于每个预训练样例(A B), 其中, A和B属于同一批语料中两个邻近的句子, B是A后面的句子, 50%这样的句对作为正例。对于另外50%的样例, 让A不变, 而B却是随机从语料中选取的句子, 这样的样例为负样本。

训练时将A和B拼接成一个序列: (A⟨seg⟩B), 其中, ⟨seg⟩是分隔符。BERT学习该序列 $L$ 的向量表示, 最终将第一个词⟨ $s$ ⟩的隐层表示  $h_{\langle s \rangle}^L$  输入到线性映射层, 并使用 Softmax 预测句子B是否在A的后面。

# 8. 预训练语言模型

## ● 模型微调

- 分类任务
- 序列标注



将给定的序列进入到BERT模型中，第一个词对应的最后一层的隐层表示  $h_{(s)}^L$  被参数矩阵  $W_o$  映射后进入到 Softmax 函数中，预测输入序列的类别标签。网络参数和映射矩阵  $W_o$  在分类任务的数据集上微调，以最大化概率  $p(y|x)$ 。其中， $y$  为类别标签。

将给定的序列进入到BERT模型中，每个词汇  $x_j$  都可以得到最后一层的隐层表示  $h_j^L$ ，该表示被参数矩阵  $W_o$  映射后进入到 Softmax 函数中，预测  $x_j$  对应的标签  $y_j$ 。网络参数和映射矩阵  $W_o$  在序列标注任务的数据集上微调，以最大化概率  $p(y|x)$ 。其中， $y$  为标签序列。



# TensorFlow 和 PyTorch 框架

- TensorFlow: <https://www.tensorflow.org/>

研发者: 谷歌人工智能团队谷歌大脑(Google Brain) 团队

- Pytorch: <https://pytorch.org/>

研发者: Facebook人工智能研究院(FAIR)

- 说明:

TensorFlow 和 Pytorch 等深度学习框架能够提供两方面功能:

- ① 提供GPU加速;
- ② 能够对神经网络的参数进行自动求导和优化。



# 本章小结

## ◆人工神经网络简介

基本概念；网络描述；激活函数；Softmax回归

## ◆FNN 及语言模型

FNN描述；参数训练(反向传播算法)；基于FNN的语言模型

## ◆RNN 及语言模型： RNN描述； 基于RNN的语言模型

## ◆LSTM 及语言模型： LSTM 描述； 基于LSTM的语言模型

## ◆自注意力机制： 提出注意力机制的动机； 执行过程

## ◆Transformer 架构： 模型结构； 残差链接，位置向量

## ◆预训练语言模型： GPT， BERT



# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录





## 8. 习题

请下载调试FNN、RNN和LSTM模型的开源工具。利用北京大学标注的《人民日报》1998年1月份的分词语料，或者利用网络爬虫自己从互联网上收集足够多的英文文本语料，借助FNN或者RNN/ LSTM开源工具，完成如下任务：

- (1)获得汉语或英语词语的词向量。
- (2)在任务(1)的基础上，随机选择20个单词，计算与其词向量最相似的前10个单词。
- (3)对于同一批词汇，对比分别用FNN, RNN 或 LSTM获得的词向量的差异。



# 8. 习题

## ● 作业要求：

完成一份实验报告（3周时间，具体时间由助教确定）。

## ● 说明

- 如果计算资源的限制，神经网络参数不必选择过大，例如：词表选择1000个左右单词即可，其余单词用<UNK>代替；词向量的维度可设为10左右；神经网络的层数设置为1到2层；
- 可以使用某一种开放的深度学习框架，如 TensorFlow 或者 PyTorch。
- 如果不借助开源工具和开放的深度学习框架，题目中的任务(3)可以不做。



# 本章内容

1. 问题提出
2. 神经网络概述
3. 前馈神经网络及语言模型
4. 循环神经网络及语言模型
5. 长短时记忆网络
6. 自注意力机制
7. Transformer 架构
8. 预训练语言模型
9. 习题
10. 附录



## 8. 附录

### ◆关于沃伦·麦卡洛克和沃尔特·皮茨

沃伦·麦卡洛克(Warren S. McCulloch)是一位美国神经生理学家和控制论专家，以其对某些大脑理论基础的研究和对控制论运动的贡献而闻名。沃伦·麦卡洛克于1898年出生于新泽西州的奥兰治。他就读于哈弗福德学院，并在耶鲁大学学习哲学和心理学，1921年获得学位，之后他继续在哥伦比亚大学学习心理学，并获得了文学硕士学位。1927年，他从纽约的哥伦比亚大学医学院获得了医学博士学位，并在纽约的贝尔维尤医院实习。然后他在 Rockland State Hospital 的精神病院为工作。1934年，他回到学术界。1934年至1941年，在耶鲁大学神经生理学实验室工作。1941年，他搬到了芝加哥，加入了精神科医院，在伊利诺大学芝加哥分校任精神病学教授，同时担任伊利诺伊州神经精神病学研究所主任，直到1951年。从1952年起，他与诺伯特·维纳一起在马萨诸塞州剑桥的麻省理工学院工作。

沃尔特·皮茨(Walter H. Pitts)出生在底特律的穷人区，家里都是文盲，从小体弱又内向的沃尔特只有一个避难所，那就是一家公立图书馆。他常常在安静的图书馆里，躲避着这个世界无尽的恶意：殴打，暴力，侮辱，还有贫穷带来的无力的辍学。在这里，他不仅找到了宁静，还自学了希腊语，拉丁语，哲学，和数学。



## 8. 附录

有一天，他在书架上发现了闪闪发光的Principia Mathematica（拉丁语：数学原理）。Bertrand Russell 和 Alfred Whitehead共同撰写的将数学和逻辑学结合的三卷大部头。他认为应该告知作者Russell教授，于是就给他写信。Russell教授看到一个年仅12岁的少年给他写信，十分惊讶。然而他被深深触动了，不仅回了信，还邀请Pitts到剑桥来当他的研究生。

然而，沃尔特的父母不懂什么叫剑桥，什么叫研究生，反正就是不许他去，一通打击和暴力之后，此事不了了之。三年之后，当沃尔特听说Russell教授要去芝加哥大学讲学的时候，就毅然离家出走了。从此，他这一生再没回家。由于沃尔特高中没有毕业，芝加哥大学不能让他正式上课，于是只能边打工边去到处蹭讲座听。沃尔特后来通过朋友介绍认识了McCulloch教授。两人虽然表面上风马牛不相及，一个42岁的富贵高壮教授，一个18岁的赤贫羞涩少年，却同时热烈相信着大脑是二进制计算系统，一拍即合，成了灵魂伴侣。McCulloch 教授把沃尔特带回了他在芝加哥郊区的家住下。每当夜深人静，McCulloch 教授的太太Rook带着三个孩子睡觉时，他就和沃尔特彻夜不眠地讨论计算神经问题。他们坚信，神经的运行，也是逻辑系统，充满了‘与’‘或’‘非’的系统。



## 8. 附录

1943年，沃尔特去了波士顿，在MIT结识了控制论之父维纳(Norbert Wiener)教授。维纳第一次见到沃尔特的那一天，也不做自我介绍，就把他领到了一块黑板前开始进行数学推导。在他推导过程中，沃尔特不断给维纳提问题和建议。写满两块黑板之后，维纳已经决定让沃尔特成为MIT的博士生。维纳指导沃尔特将神经网络计算更加复杂化、真实化，并且引入更多的统计学原理。奠定了后来的统计神经网络基础。

那年冬天，维纳还带着沃尔特去了普林斯顿，结识了天才冯·纽曼(John von Neumann)。至此，控制论核心天才组合形成了。麦卡洛克也卖掉了芝加哥的大房子，放弃正教授的职位，追随他们来到了MIT，此时控制论学派达到了顶峰。

上述关于沃尔特的故事来自如下网站：[https://www.sohu.com/a/488634442\\_701814](https://www.sohu.com/a/488634442_701814)



## 8. 附录

### ◆ 反向传播算法的推导

不失一般性，对第 $l$ 层中的参数  $\mathbf{W}^{(l)}$  和  $\mathbf{b}^{(l)}$  求偏导。由于  $\frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{W}^{(l)}}$  的计算涉及向量对矩阵的微分，十分繁琐，因此我们先计算  $\mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})$  关于参数矩阵中每个元素的偏导： $\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}}$ 。

$$\text{根据链式法则: } \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \boxed{\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}}} \boxed{\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}} \quad (6-6)$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \boxed{\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad (6-7)$$

由于上面两个式子中第二项都是目标函数关于第 $l$ 层神经元  $\mathbf{z}^{(l)}$  的偏导数，称为误差项，可以一次计算得到，因此，只需要计算三个偏导数（红框所示）。

# 8. 附录

(1) 计算  $\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}}$

回顾:  $\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \cdot \boldsymbol{\alpha}^{(l-1)} + \mathbf{b}^{(l)}$ , 那么,

$$\begin{aligned}
 \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[ \frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}} \right] \\
 &= \left[ 0, \dots, \frac{\partial (\mathbf{w}_{i:}^{(l)} \cdot \boldsymbol{\alpha}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \dots, 0 \right] \\
 &= \left[ 0, \dots, \alpha_j^{(l-1)}, \dots, 0 \right] \\
 &\triangleq \Delta_i(\boldsymbol{\alpha}_j^{(l-1)}) \tag{6-8}
 \end{aligned}$$

其中,  $\mathbf{w}_{i:}^{(l)}$  为权重矩阵  $\mathbf{W}^{(l)}$  的第  $i$  行,  $\Delta_i(\boldsymbol{\alpha}_j^{(l-1)})$  表示第  $i$  个元素为  $\alpha_j^{(l-1)}$ , 其余为 0 的行向量。



## 8. 附录

(2) 计算  $\frac{\partial z^{(l)}}{\partial b^{(l)}}$

由于:  $z^{(l)} = W^{(l)} \cdot \alpha^{(l-1)} + b^{(l)}$ ,

所以  $\frac{\partial z^{(l)}}{\partial b^{(l)}} = I_{M_l} \in \mathbb{R}^{M_l \times M_l}$  ( $M_l \times M_l$  的单位矩阵) (6-9)



## 8. 附录

(3) 计算  $\frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l)}}$

该偏导数表示第 $l$  层神经元对最终损失的影响，也反映了最终损失对第 $l$  层神经元的敏感程度，因此一般将其称为第 $l$  层神经元的**误差项**，用  $\delta^{(l)}$  表示。

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l)}} \in \mathbb{R}^{M_l} \quad (6-10)$$

误差项也间接地反映了不同神经元对网络能力的贡献程度，从而较好地解决了**贡献度的分配问题**(credit assignment problem, CAP)。



## 8. 附录

根据  $z^{(l)} = W^{(l)} \cdot \alpha^{(l-1)} + b^{(l)}$ , 有

$$\frac{\partial z^{(l)}}{\partial \alpha^{(l-1)}} = (W^{(l)})^T \in \mathbb{R}^{M_{l-1} \times M_l} \quad (6-11)$$

根据  $\alpha^{(l)} = f_l(z^{(l)})$ , 其中  $f_l(\bullet)$  为按位计算的函数, 因此有

$$\begin{aligned} \frac{\partial \alpha^{(l)}}{\partial z^{(l)}} &= \frac{\partial f_l(z^{(l)})}{\partial z^{(l)}} \\ &= \text{diag}(f'_l(z^{(l)})) \in \mathbb{R}^{M_l \times M_l} \end{aligned} \quad (6-12)$$



## 8. 附录

因此，根据链式法则，第 $l$ 层神经元的误差项为：

$$\begin{aligned}\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \times \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \times \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\ &= \text{diag}(f_l'(\mathbf{z}^{(l)})) \times (\mathbf{W}^{(l+1)}) \times \delta^{(l+1)} \\ &= f_l'(\mathbf{z}^{(l)}) \odot ((\mathbf{W}^{(l+1)})^T \cdot \delta^{(l+1)}) \quad \in \mathbb{R}^{M_l}\end{aligned}\tag{6-13}$$

其中， $\odot$ 是向量的点积运算符，表示每个元素相乘。

通过上式可以看出，第 $l$ 层的误差项可以通过第 $l+1$ 层的误差项计算得到，即误差反向传播，含义是：第 $l$ 层的一个神经元的误差项(或敏感性)是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和，然后再乘上该神经元激活函数的梯度。



## 8. 附录

得到上面的三个偏导数之后，前面的(6-6)式可以做如下改写：

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} &= \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\&= \Delta_i(\alpha_j^{(l-1)}) \cdot \delta^{(l)} \\&= [0, \dots, \alpha_j^{(l-1)}, \dots, 0] \cdot [\delta_1^{(l)}, \dots, \delta_i^{(l)}, \dots, \delta_{M_l}^{(l)}] \\&= \delta_i^{(l)} \cdot \alpha_j^{(l-1)}\end{aligned}\tag{6-14}$$

其中， $\delta_i^{(l)} \cdot \alpha_j^{(l-1)}$ 相当于向量 $\delta_i^{(l)}$ 和向量 $\alpha^{(l-1)}$ 的外积的第*i,j*个元素。



## 8. 附录

上面的(6-14)式可以进一步改写为：

$$\left[ \frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathbf{W}^{(l)}} \right]_{ij} = [\delta^{(l)}(\boldsymbol{\alpha}^{(l-1)})^T]_{ij} \quad (6-15)$$

因此,  $\mathcal{L}(y, \hat{y})$  关于第  $l$  层权重  $\mathbf{W}^{(l)}$  的梯度为：

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}(\boldsymbol{\alpha}^{(l-1)})^T \in \mathbb{R}^{M_l \times M_{l-1}} \quad (6-16)$$

同理,  $\mathcal{L}(y, \hat{y})$  关于第  $l$  层偏置  $\mathbf{b}^{(l)}$  的梯度为：

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)} \in \mathbb{R}^{M_l} \quad (6-17)$$



## 8. 附录

计算出每一层的误差项之后，就可以得到每一层参数的梯度，因此，使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步：

- (a) 前馈计算每一层的净输入  $z^{(l)}$  和激活值  $\alpha^{(l)}$ ，直到最后一层；
- (b) 反向传播计算每一层的误差项  $\delta^{(l)}$ ；
- (c) 计算每一层参数的偏导数，更新参数。



# 8. 附录

## ● 基于反向传播算法的随机梯度下降参数训练过程

**输入:** 训练集  $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$ , 验证集  $V$ , 学习率  $\ell$ , 正则化系数  $\varepsilon$ , 网络层数  $L$ , 神经元数量  $M_l$ ,  $1 \leq l \leq L$ 。

**输出:**  $\mathbf{W}, \mathbf{b}$

重复执行  
该过程:

(1) 对训练集  $D$  中的样本随机重排序;  
(2) for  $n=1 .. N$  do  
    从训练集  $D$  中选取样本  $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ ;  
    前馈计算每一层的净输入  $\mathbf{z}^{(l)}$  和激活值  $\mathbf{\alpha}^{(l)}$ , 直到最后一层;  
    反向传播计算每一层的误差  $\delta^{(l)}$ ; 公式(6-13)

// 计算每一层参数的导数:  $\left\{ \begin{array}{l} \forall l, \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} (\mathbf{\alpha}^{(l-1)})^T \\ \forall l, \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)} \end{array} \right.$  公式(6-16)

$\forall l, \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$  公式(6-17)

// 更新参数:  $\left\{ \begin{array}{l} \mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \ell (\delta^{(l)} \cdot (\mathbf{\alpha}^{(l-1)})^T + \varepsilon \mathbf{W}^{(l)}) \\ \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \delta^{(l)}; \end{array} \right.$

end

直到模型在验证集  $V$  上的损失函数值收敛, 结束过程, 输出  $\mathbf{W}, \mathbf{b}$ 。

谢谢！

*Thanks!*

