

## 第五章 神经网络与深度学习

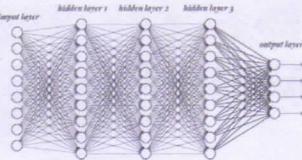
### 人工智能、机器学习与深度学习



### 什么是深度学习

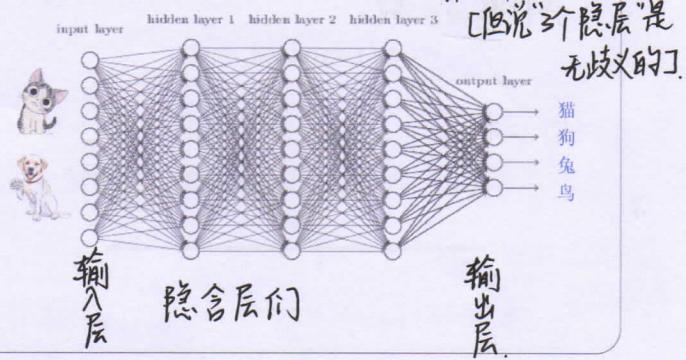
- ◆ 深度学习就是多层(人工)神经(元)网络
- ◆ 神经网络起源于40年代 → XOR问题求解不了。
- ◆ 80年代曾经掀起过高潮 (异或) (当时的神经网络不能解决)。

1984年提出了BP算法,可以解决神经网络的学习算法的缺陷。



### 神经网络示意图

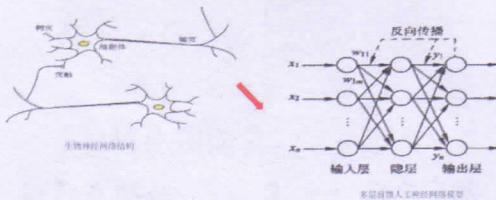
这个图,4层或5层都有人叫...  
(说清算不算输入层)。



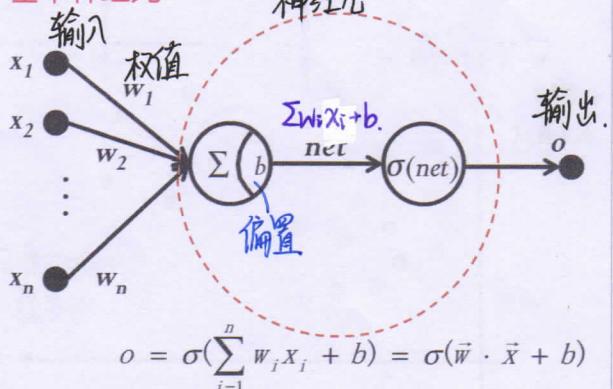
### 人工神经网络

- ◆ 人工神经网络(Artificial Neural Networks, ANN)

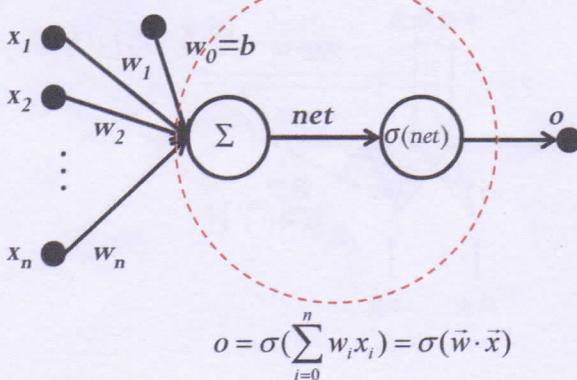
#### 生物神经网络 vs. 人工神经网络



### 基本神经元



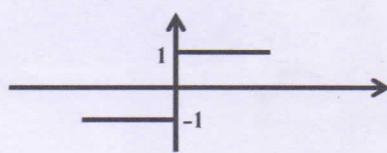
可以统一形式,不用单独写偏置。



### 常用激活函数

- ◆ 符号函数

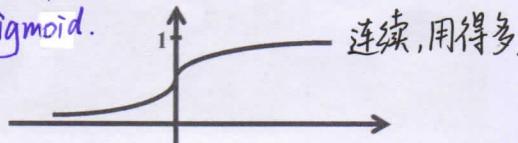
$\sigma(\text{net}) = \text{sgn}(\text{net})$  早期使用。



### ◆sigmoid函数 (0,1)

$$\sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}}$$

一般论文中的G  
专指sigmoid.

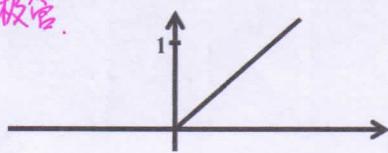


### ◆ReLU函数

线性整流函数.

$$\sigma(\text{net}) = \max(0, \text{net})$$

类似二极管.



### ◆tanh函数 (-1,1). 可经过简单变换

变为sigmoid.

$$\sigma(\text{net}) = \frac{e^{\text{net}} - e^{-\text{net}}}{e^{\text{net}} + e^{-\text{net}}}$$

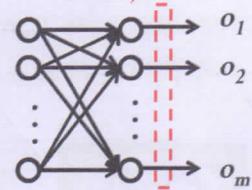


### ◆Softmax激活函数

作用在整个一层.(在输出时用)

$$o_j = \sigma(\text{net}_j) = \frac{e^{\text{net}_j}}{\sum_i e^{\text{net}_i}}$$

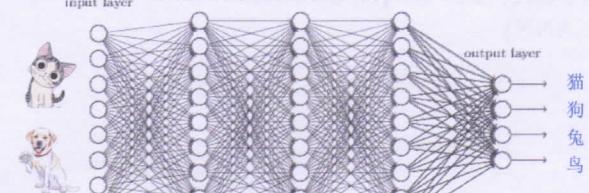
输出的和为1.  
可以看作概率.



Softmax

### 全连接网络

input layer    hidden layer 1    hidden layer 2    hidden layer 3

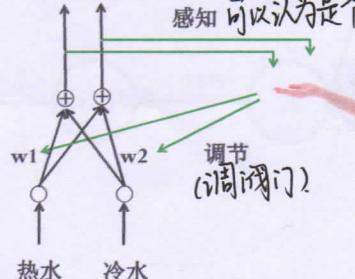


每一个神经元与上一层的所有神经元  
都有连接,与下一层……也都有连接.

### 如何训练? 对权重进行调整.



感知可以认为是个ReLU输出.



## 神经网络的学习 (训练)

◆ 对于给定的一组训练集  $D$ , 寻找到一组合适的权重  $\vec{w}$ , 使得神经网络的输出, 与对应输入希望的输出尽可能的一致。

◆ 输入、输出均以向量形式表示

## 损失函数

希望的输出

◆ 定义误差函数

↑  
实际的输出

d: 某一个样本.

最小化

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

◆ 其中:

□  $D$ : 训练样例的集合

□  $outputs$ : 输出单元的集合

□  $t_{kd}$ : 样例  $d$  在第  $k$  个输出单元的标注输出值

□  $o_{kd}$ : 样例  $d$  在第  $k$  个输出单元的实际输出值

## 反向传播算法 (BP)

◆ 梯度下降方法 求导数 (往其反方向走直到导数为0).

所有样本  
算一次才改

一批  
加一个

收敛缓慢

容易陷入局部极值点

◆ 随机梯度下降

每次处理一个样本 就修改一次.

虽然并不能保证收敛到全局最优, 但一般能得到一个不错的结果

◆ 小批量梯度下降 最常用.

每次处理少量样本 再修改一次.

## 反向传播算法 (BP)

◆ 随机梯度下降法

□ 定义训练样本  $d$  的误差  $E_d$ :

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

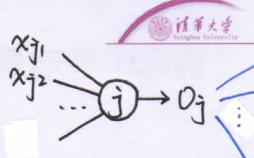
□ 对每一个样本, 更新一次权重, 反复迭代

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}, \text{ 其中 } 0 < \eta < 1 \text{ 为学习速率}$$

最小化: 往梯度的反方向走.  $w_{ji} = w_{ji} - \eta \Delta w_{ji}$  太快可能引起振荡

## 反向传播算法 (BP)

◆ 符号:  $x_{ji}$  = 单元  $j$  的第  $i$  个输入



$w_{ji}$  = 与单元  $j$  的第  $i$  个输入相关联的权值

$$net_j = \sum_i w_{ji} x_{ji} \quad (\text{单元 } j \text{ 的输入的加权和})$$

$o_j$  = 单元  $j$  的实际输出

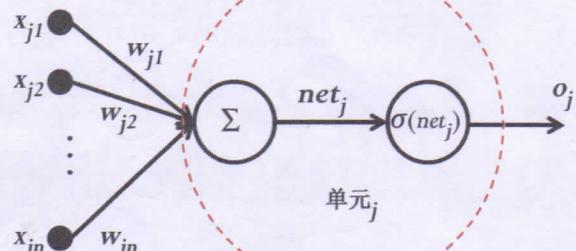
$t_j$  = 单元  $j$  的目标输出

$\sigma$  = sigmoid 函数

$outputs$  = 输出层单元的集合

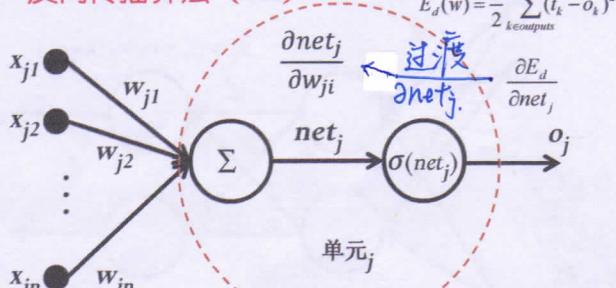
$Downstream(j) =$  以单元  $j$  作为输入的节点的集合  
下游.

Downstream( $j$ ).



$$o_j = \sigma \left( \sum_{i=0}^n w_{ji} x_{ji} \right) = \sigma(\vec{w}_j \cdot \vec{x}_j)$$

## 反向传播算法 (BP)



## 反向传播算法 (BP)

链式法则.

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\because net_j = \sum_l w_{jl} x_{jl}$$

◆ 分输出层和隐含层两种情况计算  $\frac{\partial E_d}{\partial net_j}$

### 情况1：单元j是输出层

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

#### ◆ 第一项：

$$\begin{aligned}\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j} \left( \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \right) \\ &= -(t_j - o_j)\end{aligned}$$

$$o_j = \sigma(net_j)$$

### 情况1：单元j是输出层

#### ◆ 综合上述两项：

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)o_j(1 - o_j)$$

$$\text{令 } \delta_j = \frac{\partial E_d}{\partial net_j} = (t_j - o_j)o_j(1 - o_j)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji} = \eta \delta_j x_{ji}$$

其中  $0 < \eta < 1$  为学习速率

### 情况1：单元j是输出层

#### ◆ 第二项：

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1 - o_j) \quad (\text{sigmoid函数的导数})$$

$$o_j = \sigma(net_j)$$

### 情况2：单元j是隐含层

k是输出单元

$$\begin{aligned}\frac{\partial E_d}{\partial net_j} &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \quad \text{chain rule} \\ &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \quad \text{再过渡} \\ &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} w_{kj} \frac{\partial o_j}{\partial net_j} \quad \text{前面已经求出} - \delta_k \\ &= \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} w_{kj} o_j(1 - o_j) \quad \text{sigmoid梯度} \\ &= o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \frac{\partial E_d}{\partial net_k} w_{kj}\end{aligned}$$

### 情况2：单元j是隐含层

对隐含层的也整这个  
同样用  $\delta_j$  表示  $-\frac{\partial E_d}{\partial net_j}$ , 则: 表示.

$$\delta_j = o_j(1 - o_j) \sum_{k \in \text{Downstream}(j)} \delta_k w_{kj}$$

输出层 → 最后一层隐含层 → 倒数第二层 → ...

### 情况2：单元j是隐含层

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji} = \eta \delta_j x_{ji}$$

其中  $0 < \eta < 1$  为学习速率

◆ 注意：上式与单元j是输出层的形式是一样的，只是  $\delta_j$  计算不同

反向递推  
除了输出层外  
公式都一样.

$$\begin{aligned}\delta_h &= o_h(1 - o_h) \sum_{k \in \text{Downstream}(h)} \delta_k w_{kh} \\ \delta_k &= (t_k - o_k)o_k(1 - o_k) \\ w_{ji} &= w_{ji} + \Delta w_{ji} \quad \Delta w_{ji} = \eta \delta_j x_{ji}\end{aligned}$$

## BP算法（随机梯度下降版）

初始化所有权值为小的随机值(如[-0.05,0.05])

在满足结束条件前：

对于每个训练样例

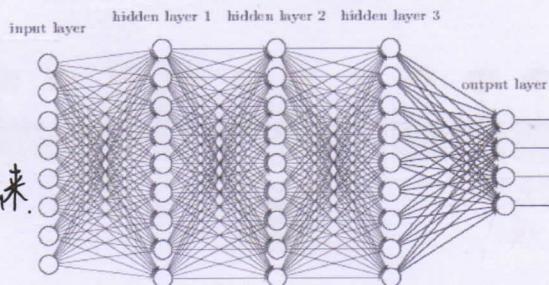
把样例输入网络，计算每个单元 $u$ 的输出 $o_u$

- 对于输出层单元 $k$ ，计算误差项： $\delta_k = (t_k - o_k)o_k(1 - o_k)$

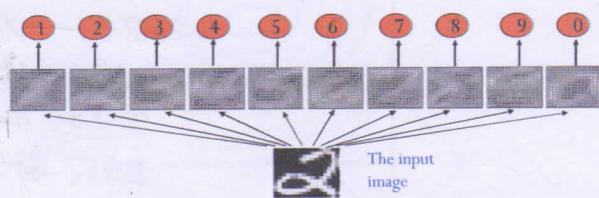
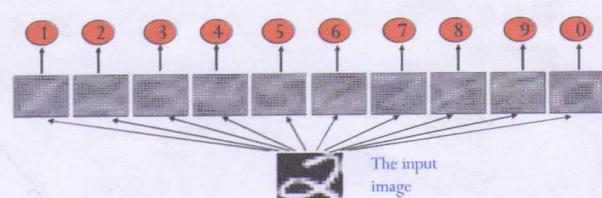
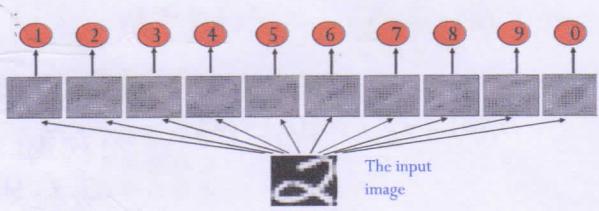
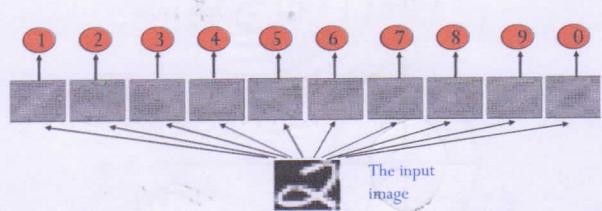
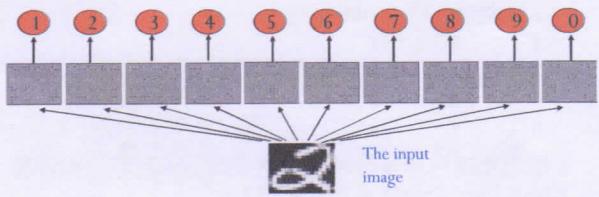
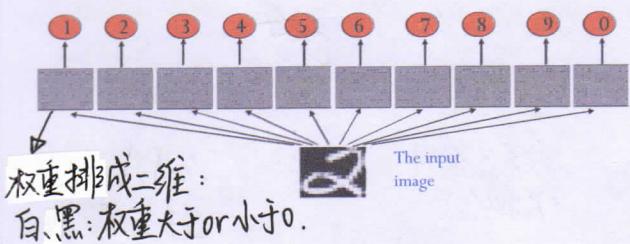
- 对于隐含层单元 $h$ ，计算误差项： $\delta_h = o_h(1 - o_h) \sum_{k \in \text{Downstream}(h)} \delta_k w_{kh}$

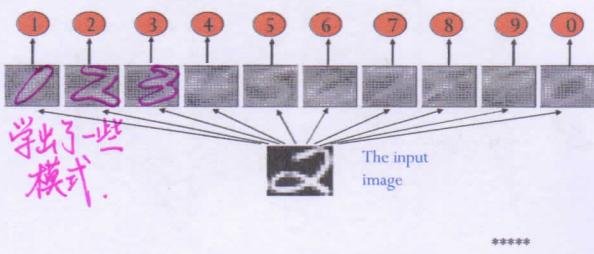
- 更新每个权值： $w_{ji} = w_{ji} + \Delta w_{ji}$
  - 其中： $\Delta w_{ji} = \eta \delta_j x_{ji}$
- 多个隐层也能  
↑由此推出来  
记住中间以哪个为过渡  
(Chain Rule)

## 学到了什么？



应用：手写数字识别。





## 练习题

- ◆ 异或 (XOR) 问题如下:

$$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$$

a	b	$a \oplus b$
1	0	1
1	1	0
0	0	0
0	1	1

- ◆ 编程实现用全连接神经网络求解XOR问题（用BP算法训练一个全连接神经网络）。

## 交叉熵损失函数

- ◆ 交叉熵损失函数

$$H_i(o) = -\sum_t t_i \log(o_i)$$

( $t_i$  和  $o_i$  分别是第  $i$  个训练样本的标注值和预测值)

- ◆ 度量的是两个概率分布的距离

- ◆ 一般在输出层配合 softmax 一起使用

- ◆ 常用于分类问题中

## 练习题

- ◆ 异或 (XOR) 问题如下:

$$a \oplus b = (\neg a \wedge b) \vee (a \wedge \neg b)$$

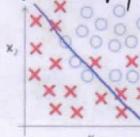
a	b	$a \oplus b$
1	0	1
1	1	0
0	0	0
0	1	1

- ◆ 编程实现用全连接神经网络求解XOR问题（用BP算法训练一个全连接神经网络）。

## 过拟合问题

误差不是越小越好.

underfit



效果不好  
(欠拟合).

可能为噪声  
/样本标注错误  
overfit



分界线  
过于复杂  
(为了解决那两个).  
(为了解决那两个).

## 奥

### 减少过拟合的解决办法

- ① 在损失函数中增加正则项 类似 penalize.

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 + \|\vec{w}\| \text{ 范数.}$$

$$H_i(o) = -\sum_i t_i \log(o_i) + \|\vec{w}\|$$

权值范数较小时,  
在 sigmoid 的 x 轴上.  
基本处于线性的  
部分 (or 非线性程度  
没那么高).

不会太复杂

也不会因某个  $x$   
的微小变化  
大震荡.

### L1 正则项

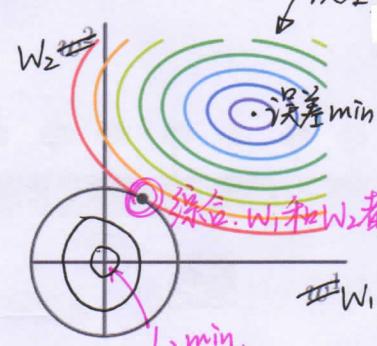
大概率落在某个轴上.

(不是每个  $w$  都很小,  
一般有的  $w$  为 0,  
有的不是 — 相当于  
对各个连接边  
做选择, 删去了  
其中的一部分).



每个  $w$  的各分量的平方和 ( $w^2$ ),

### L2 正则项



### 减少过拟合的解决办法

- ② 舍弃 (Dropout) 机制

- ◆ 在每次训练时,  
随机的舍弃一些  
隐层单元)

50% 左右.

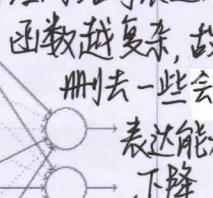
假没  
它们不存在.

(图中虚线部分).

每一轮训练舍弃不同的单元.

隐层节点越多,  
神经网络可表达的

函数越复杂, 故  
删去一些会使  
表达能力下降



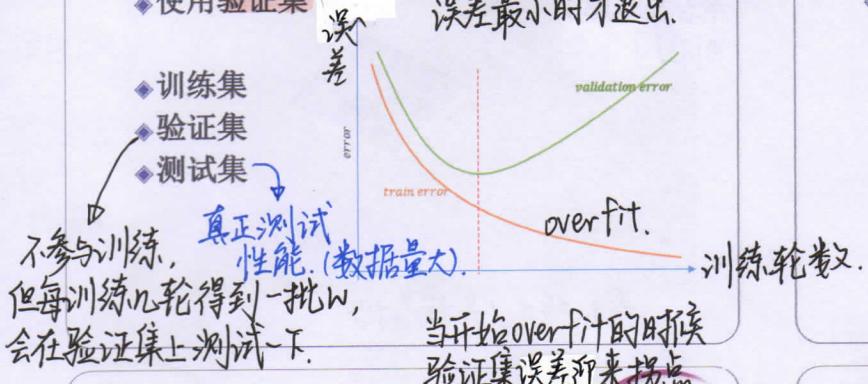
函数  
不会特别复杂.

① 数据不够多时可只有验证集没有测试集，但得到的正确率会偏高。

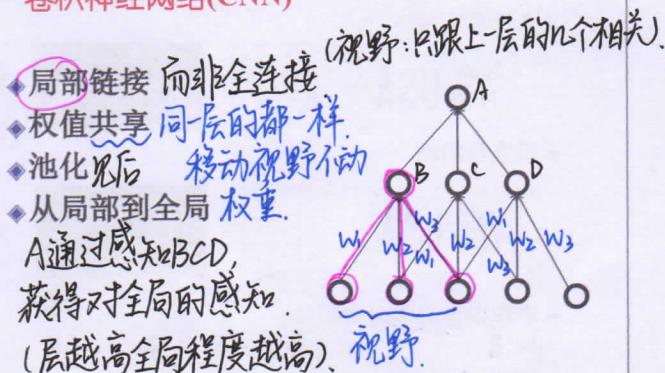
② 验证集也可以用来确定超参数（如将下图横轴换为L2的参数c ( $\text{loss} + c\|\mathbf{w}\|_2$ )）

### 减少过拟合的解决办法

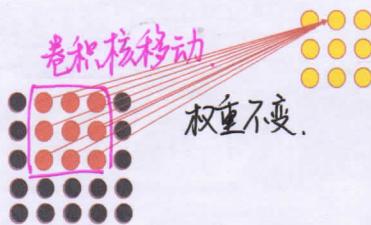
#### ◆ 使用验证集



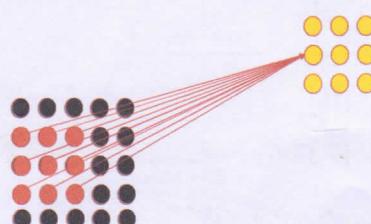
### 卷积神经网络(CNN)



### 局部连接与权值共享



### 局部连接与权值共享



### 全连接神经网络存在的问题

#### ◆ 参数过多

数字识别为例

(假设图象 $28 \times 28$ , 隐层500个节点)

$28 \times 28 \times 500 + 500 = 392,500$ 个参数

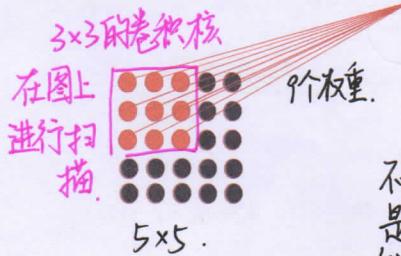
计算速度慢

容易引起过拟合

### 最早用于处理图像

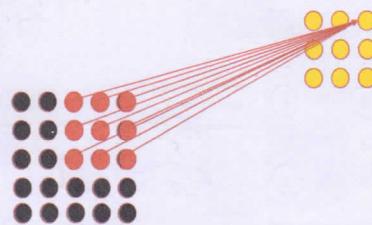
#### 局部连接与权值共享

对应第二层的这个值。



不是矩阵相乘，  
是对应位置相乘  
然后9项求和。  
(如果有偏置b, 那么求和后  
再加)。

### 局部连接与权值共享

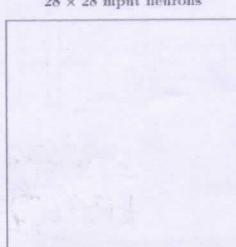


权值不同(核间无共享)。

#### 多卷积核

◆ 3个 $5 \times 5$ 的卷积核

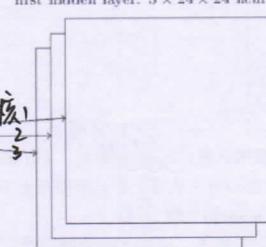
28 × 28 input neurons



-一般会有多个，每个核扫描后得到一个

隐层

first hidden layer:  $3 \times 24 \times 24$  neurons



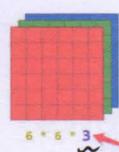
灰度图。

但一般我们还是叫它 $3 \times 3$ 的核，默认

(多出一维通道数) 通道数不说。

## 多通道卷积

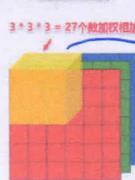
彩色图：  
RGB三原色  
(3通道)



卷积核是3维的

得到一个结果。

$3 \times 3 \times 3$   
通道数必须相同



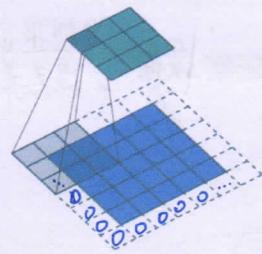
$3 \times 3 \times 3 = 27$  个数相加相加

乘积相加。

再加b!!!

但实际是27 填充与步长  
个参数。

在周围填。



◆ 原始输入： $5 \times 5$ ，填充：1，实际输入： $7 \times 7$

◆ 步长：2

卷积核未必是挨着挪

Input Volume (input 1) ( $7 \times 7 \times 3$ )	Filter W0 ( $3 \times 3 \times 3$ )	Filter W1 ( $3 \times 3 \times 3$ )	Output Volume ( $3 \times 3 \times 1$ )
0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0	-1 0 0 0 0 0 0 0 0	1 0 -9
0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0	-1 1 0 0 0 0 0 0 0	-6 1 1
0 0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0	-1 0 1 0 0 0 0 0 0	4 -9 3
0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0	-1 0 0 1 0 0 0 0 0	1 1 -4 -4
0 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0	-1 0 0 0 1 0 0 0 0	-2 -2 -4
0 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0 0	-1 0 0 0 0 1 0 0 0	-3 -3 -3
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0	-1 0 0 0 0 0 1 0 0	0 -1 0
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0	-1 0 0 0 0 0 0 1 0	1 0 1
0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	-1 0 0 0 0 0 0 0 1	0 -1 0

记得+b!!!

◆ 原始输入： $5 \times 5 \times 3$ ，填充：1，实际输入： $7 \times 7 \times 3$

◆ 卷积核：2个 $3 \times 3 \times 3$ ，步长：2，输出： $3 \times 3 \times 2$

清华大学  
Tsinghua University

## 卷积核的作用

◆ 水平卷积核



横

(特征提取)

◆ 垂直卷积核



竖

◆ 卷积核自动学习得到

$W$ 值。

同样使用BP

类似滤波器。

(但这个的BP推导起来比较麻烦)

清华大学  
Tsinghua University

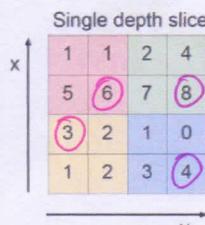
## 池化

还有min和avg(用的最多的是max).

### MAX POOLING

作用：信息压缩。

② 主要信息保留



max pool with  $2 \times 2$  filters and stride 2

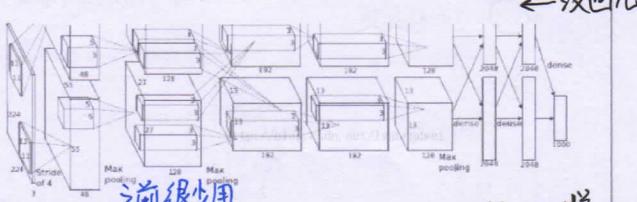
步长为2.

只留下最大值。

一般(若干个)卷积后会跟一个池化。

2012年参加某比赛，成绩超好。  
证明了深度学习good.

### 例——AlexNet

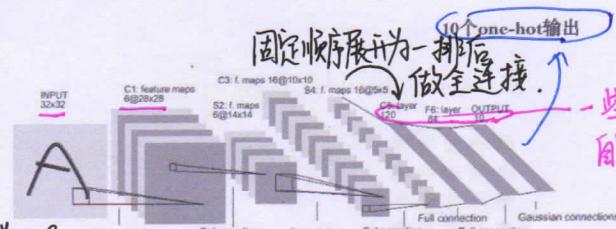


上下两部分一样，  
是两个GPU，最后合在一起。

一个原始图像随机截取多次，可得多个。  
训练集变大了！】

### 例——LeNet-5

一般做0~9的数字输出。



应为0~9。  
不知为何  
是个A...

6个 $5 \times 5$ 核

2 $\times$ 2池化

16个 $5 \times 5$ 核

2 $\times$ 2池化

3个全连接层

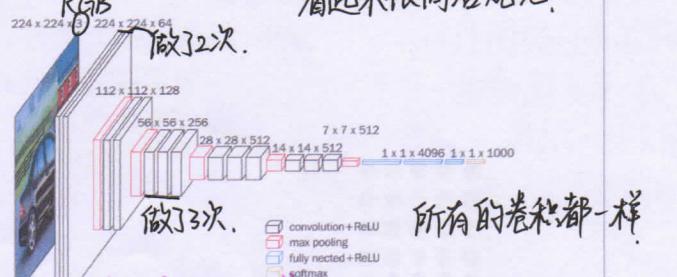
(子采样)(其实是 $5 \times 5 \times 6$ )

(步长为2). 略去一维通道数。

2014年同一个比赛的亚军。

### VGGNet——VGG-16

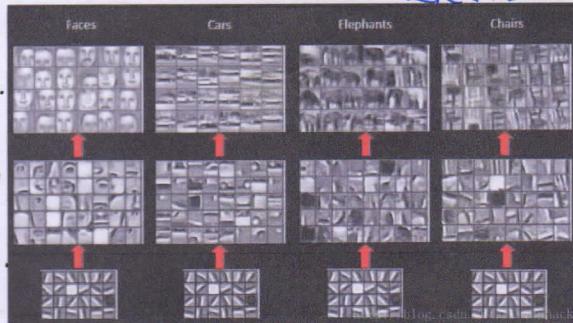
看起来很简洁规范。



卷积核： $3 \times 3$ ；步长：1；填充：1；最大池化：2 $\times$ 2；步长：2。

思想：多个小卷积代替一个大卷积，参数少，效果更好。

## 卷积核具有抽取特征的能力



## 卷积神经网络的特点

- ◆ 参数少，只与卷积核的大小和数量有关
- ◆ 具有特征抽取能力
- ◆ 特征的平移不变性

(移动位置仍能捕捉)  
(当然一般只能get到微动).

(如果担心特征旋转了(如□),可以把图像转一下再识别一次).

## 梯度消失问题 层数越多,靠近输入处梯度越小.

$$\delta_h = o_h(1 - o_h) \sum_{k \in \text{Downstream}(h)} \delta_k w_{kh} \quad (\text{乘着乘着就小了}).$$

$$o_h = \sigma(\text{net}_h) = \frac{1}{1 + e^{-\text{net}_h}}$$

$\sigma'_h = o_h(1 - o_h)$ , 最大值为  $1/4$

一般,  $|w_{kh}| < 1$

如果  $|w_{kh}|$  比较大, 会导致  $\sigma'_h \approx 0$ ,

如果  $|w_{kh}|$  过大, 也可能导致梯度爆炸

① 使用 ReLU 激活函数 ② 在网络结构上进行改善  
但 ReLU 也有别的问题

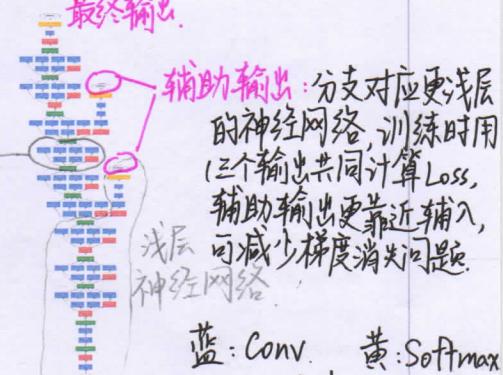
## GoogLeNet

GoogLeNet

前身是 Le-Net

inception: ↗  
先后

最终输出.



蓝: Conv. 黄: Softmax  
红: MaxPool

## Inception 结构

### GoogLeNet (出自深度空间)

尺寸不同加 padding,  
最终按通道叠在一起即可

卷积核尺寸  
不同

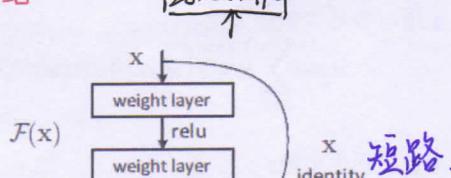
(每个尺寸都有多个)

例: GoogLeNet

第一层由 Inception:

$$① 256 = 64 + 128 + 32 + 32$$

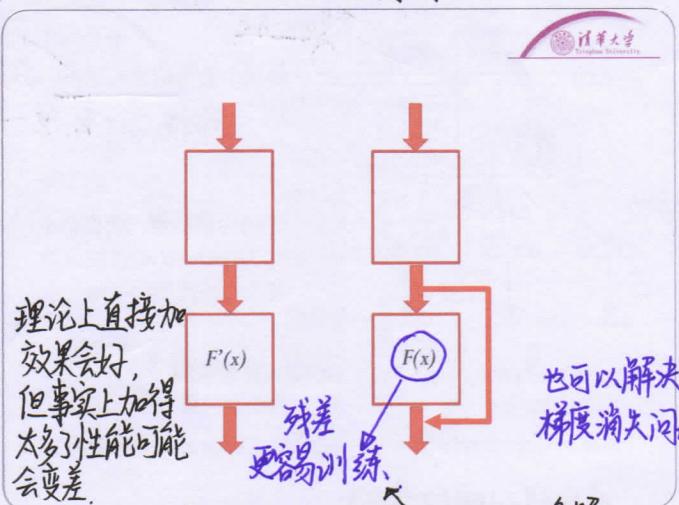
残差网络



$$\begin{aligned} F(x) &\rightarrow \text{weight layer} \rightarrow \text{relu} \\ &\rightarrow \text{weight layer} \rightarrow H(x) \\ F(x) + x &\rightarrow \text{identity} \\ H(x) &= F(x) + x \\ F(x) &= H(x) - x \end{aligned}$$

## 残差网络

应该有六七个  
残差块



找一个180cm的人 参照, 找一个187cm的人

也可以解决  
梯度消失问题.

# 如何用神经网络处理文本



## 词向量

更广义 使用嵌入方法

- ◆ 词向量又称为词嵌入 (Word Embedding)，是一种将单词表示为向量的方法。

嵌入：满足一定性质的一种变换。

## 词向量

- ◆ One Hot 表示 简单粗暴，数据且分辨不出语义是否相近

一个与词表长度等长的向量，对于具体的这个词，向量的相应位置置为1，其余为0。

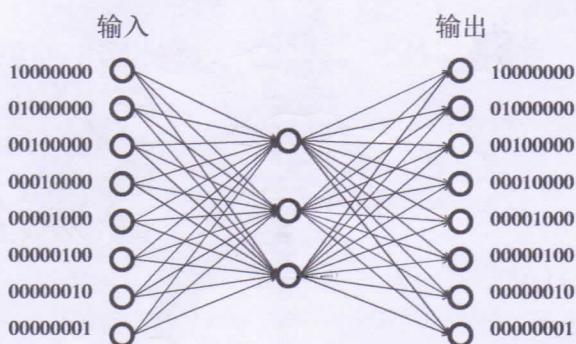
## 分布式表示

- 一种压缩表示方法，将词映射到一个较短的向量，用向量的所有位联合表示一个词。
- 可根据需要指定向量的大小。

就像用RGB表示颜色，  
则只需三维。

一般语义相近的在空间中  
也相近。

## 压缩表示举例



## 四进制变为二进制

输入值	隐层值	输出值
10000000	.89 .04 .08	1 0 0 000000
01000000	.15 .99 .99	0 1 1 1000000
00100000	.01 .97 .27	0 1 0 1100000
00010000	.99 .97 .71	1 1 1 1010000
00001000	.03 .05 .02	0 0 0 1001000
00000100	.01 .11 .88	0 0 1 1000100
00000010	.80 .01 .98	1 0 1 0000100
00000001	.60 .94 .01	1 1 0 1000001

变成3维了！  
有一定差别。

## 神经网络语言模型 (NNLM) 得到词向量

条件概率

上文 n 语法就是前 n-1 个词

- 利用神经网络计算  $p(w_t | \text{context}(w_t))$  的一种方法
  - $\text{context}(w_t) = w_{t-n+1}^{t-1} = w_{t-n+1}, w_{t-n+2}, \dots, w_{t-1}$
- 在 NNLM 中用关于  $\theta$  的函数  $F(w, \text{context}(w), \theta)$  表示概率  $p(w | \text{context}(w))$ 
  - 其中  $\theta$  为待估参数，通过语料库  $C$  求得最优的参数  $\theta^*$  得到该模型。

NLP 中大部分参数都是 MLE。  
(就不用标注了)。

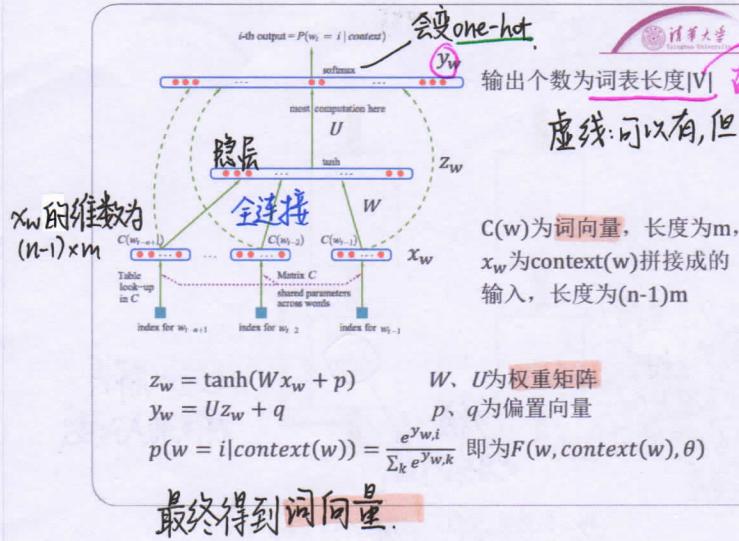
采用最大似然方法估计：

$$\max_{\theta} \left( \prod_{w \in C} F(w, \text{context}(w), \theta) \right)$$

一般用对数最大似然方法：

$$\max_{\theta} \left( \sum_{w \in C} \log(F(w, \text{context}(w), \theta)) \right)$$

积变和。



用对数最大似然方法：

$$\max_{\theta} \left( \sum_{w \in C} \log(F(w, \text{context}(w), \theta)) \right)$$

定义损失函数：

$$L = - \sum_{w \in C} \log(F(w, \text{context}(w), \theta))$$

$$\theta^* = \arg \min_{\theta} L$$

用BP算法求解

(可以当作输入为1, 词向量中的每一位为权值)。

◆ 问题：

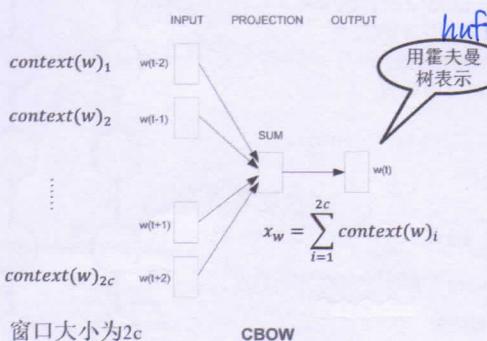
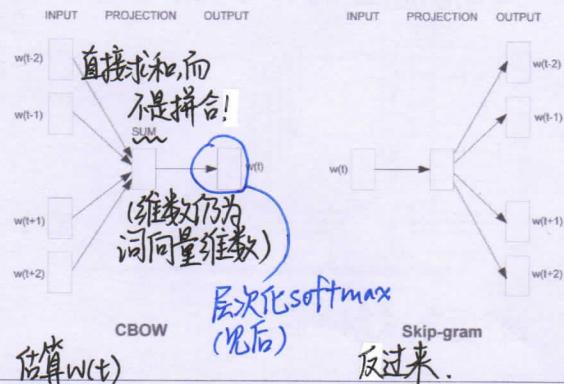
- Softmax计算太慢
- $x_w$ 太大

◆ 为此提出简化方法：word2vec

- CBOW模型（Continuous Bag-of-Words Model）**主要介绍**
- Skip gram模型

前文  
长度均为c  
后文  
(2c的窗口)

## word2vec：CBOW和Skip-gram



## 一般性描述

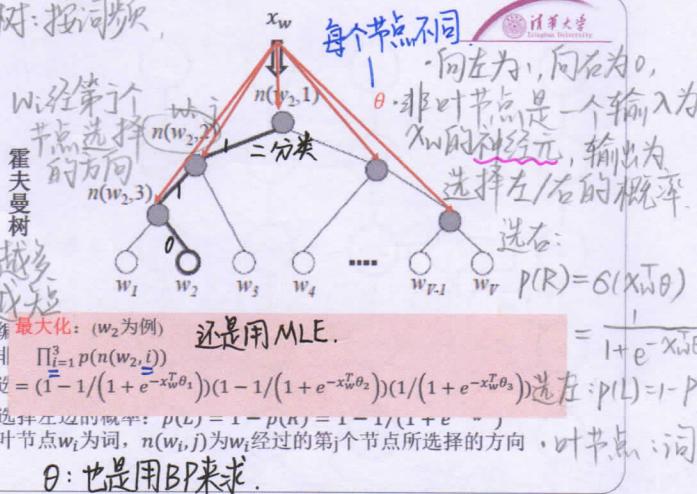
- $l_w$ : 从根节点到 $w$ 所在的叶节点包含的节点数
- $p_i^w$ : 经过的第*i*个节点，对应的霍夫曼编码为 $d_i^w \in \{0, 1\}$ 
  - $i = 2, 3, \dots, l_w$ , 根节点不对应编码
- $\theta_i^w$ :  $p_i^w$ 对应的模型参数 ( $i = 1, 2, \dots, l_w - 1$ )

- $w$ 经过霍夫曼树每个节点时的概率为：

$$p(d_i^w | x_w, \theta_{i-1}^w) = \begin{cases} \sigma(x_w^T \theta_{i-1}^w) & d_i^w = 0 \\ 1 - \sigma(x_w^T \theta_{i-1}^w) & d_i^w = 1 \end{cases}$$

故 $d_i^w$ 与 $1-d_i^w$ 一个为1一个为0。

## 建树：按词频



## 词w的最大似然函数：

$$\prod_{i=2}^{l_w} p(d_i^w | x_w, \theta_{i-1}^w) = \prod_{i=2}^{l_w} [\sigma(x_w^T \theta_{i-1}^w)]^{1-d_i^w} [1 - \sigma(x_w^T \theta_{i-1}^w)]^{d_i^w}$$

把每条路经都乘起来。向左 向右。

## 定义损失函数：

$$L = -\log \prod_{i=2}^{l_w} p(d_i^w | x_w, \theta_{i-1}^w)$$

$$= -\sum_{i=2}^{l_w} \{(1 - d_i^w) \log [\sigma(x_w^T \theta_{i-1}^w)] + d_i^w \log [1 - \sigma(x_w^T \theta_{i-1}^w)]\}$$

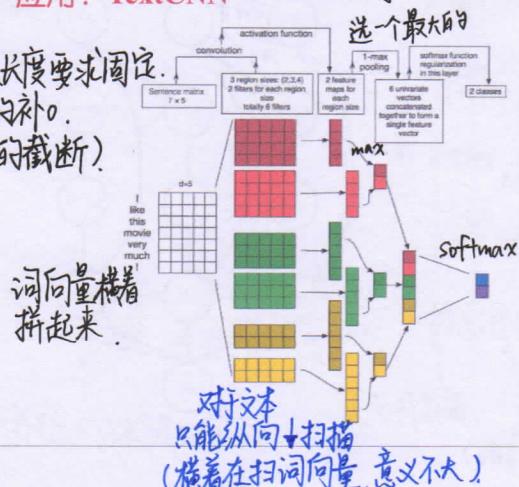
## 词向量（词嵌入）的性质

- 词义相近的词向量在空间距离比较近
- 词向量可以运算



## 应用：TextCNN

处理短句，输出类别。



## 应用: TextCNN

一句话两通道(两种词向量).

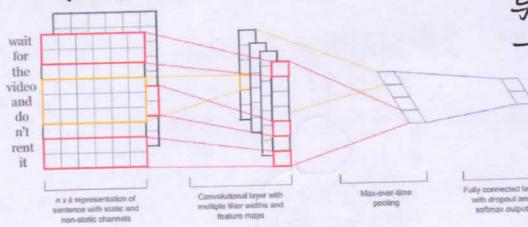


Figure 1: Model architecture with two channels for an example sentence.

这个图有一丢丢问题.

训练时:一个词向量保持不变,另一个词向量与CNN的权重一起训练.

针对性  
(适应这个领域).

通用  
循环神经网络

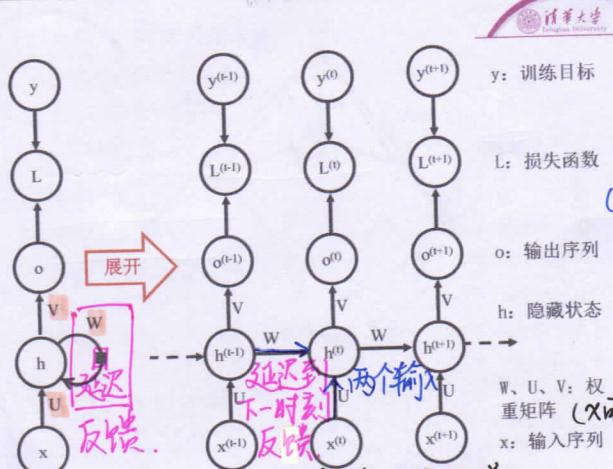
递归神经网络的缩写也叫RNN,

◆ 循环神经网络RNN(recurrent neural network)要区分、

◆ 一种处理序列数据的神经网络

时间序列、语音、语言.....

可以一直输入和获得输出  
(长度不定)



$x$ 可能为向量、一组词等  
多个输入,与 $h$ 作全连接. 清华大学计算机系

输出  
(概率)

前一个 输入 偏置

$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b$

$h^{(t)} = \tanh(a^{(t)})$  双曲正切(作用到

$o^{(t)} = Vh^{(t)} + c$  偏置

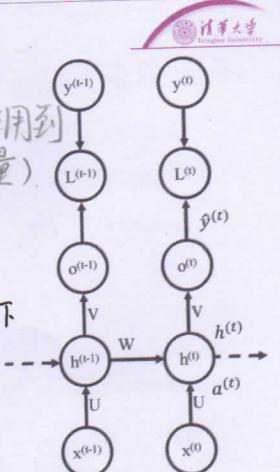
$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$  每个分量)

其中  $b, c$  为偏置

Loss:  $L(\{x^{(1)} \dots x^{(T)}\}, \{y^{(1)} \dots y^{(T)}\})$

$= -\sum_t \log p_{\text{model}}(y^{(t)} | \{x^{(1)} \dots x^{(t)}\})$

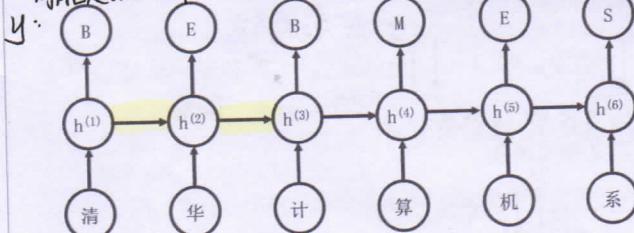
其中  $p_{\text{model}}$  表示  $\hat{y}^{(t)}$  中与  $y^{(t)}$  对应的项  
的概率



RNN举例: 汉语分词(简略图)  
可能是one-hot.

输入输出等长.

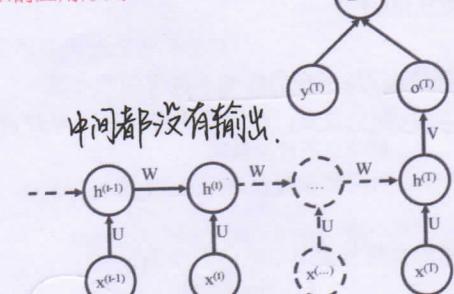
用BEMPS可以让每个字都有输出!



输入: 字向量

标记: B、E、M、S分别表示词的开始字、结束字、中间字和单字词

RNN的应用形式(一)



输入为序列, 输出为一个向量, 可以用于分类, 或者得到一个输入序列的向量表示, 作为后续的输入。

RNN的应用形式(二)

输入为一个向量,  
输出为一个序列。

训练时,  $y^{(0)}$  作为一个输入  
使用时, 用  $\hat{y}^{(t-1)}$  代替  $y^{(t-1)}$  作为输入

例:看图说话

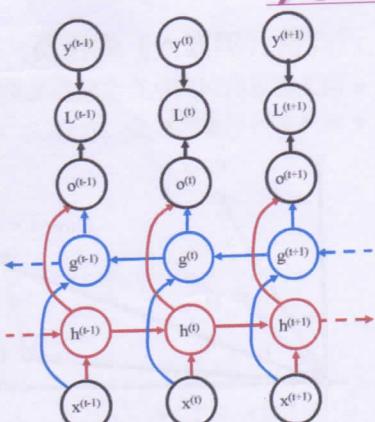
输入 脱出序列  
(可能是经过  
CNN处理后的)

( $y^{(t)}$  是  $\hat{y}^{(t)}$  的输入, 故它们用来  
计算 Loss不太对)

输出是一个序列 双向RNN

两个独立的RNN(前  
向和后向), 每个  
输出单元  $o^{(t)}$  受益于  
来自过去的相关信  
息  $h^{(t-1)}$  和未来的相关  
信息  $g^{(t)}$

→: 对于后部分  
较为关注.  
←: 对于前半部分  
较为关注



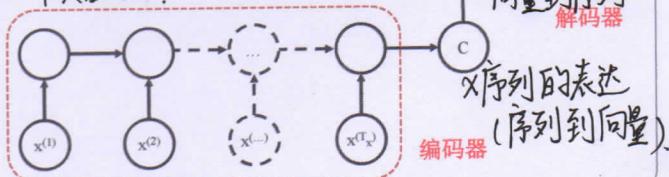
## Sequence-to-Sequence

序列到序列架构

在给定输入序列的情况下，学习生成输出序列。

两个序列不等长。

如：中英翻译



$y^{(t)}$  可能包含对后面的预测

这是干什么的？

与RNN  
应用形式(二)  
相似

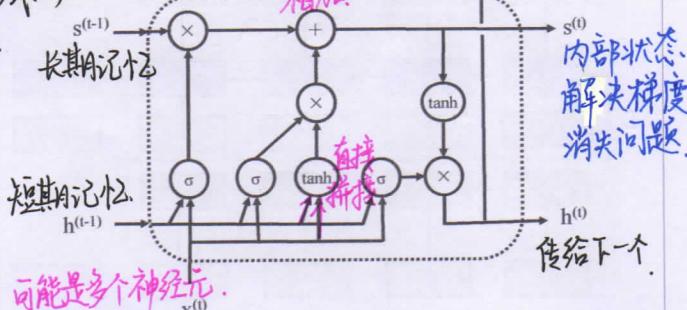
长短时记忆网络LSTM

特殊的RNN



$h^{(t)}$  (用于输出)

内部状态，  
解决梯度  
消失问题。



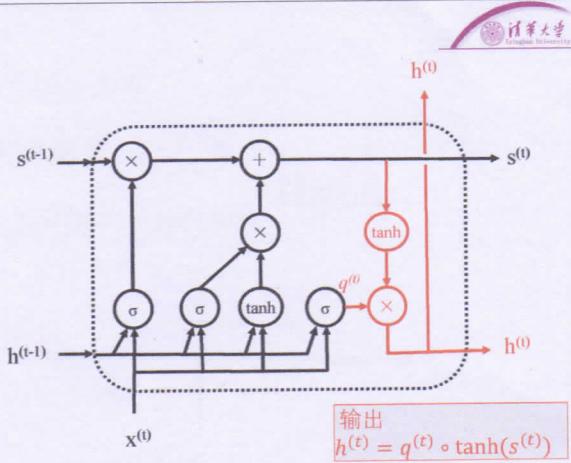
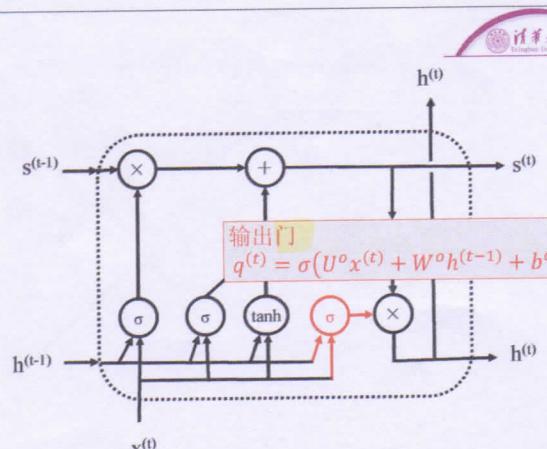
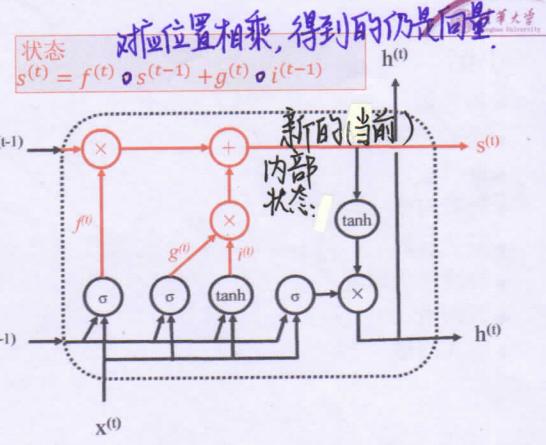
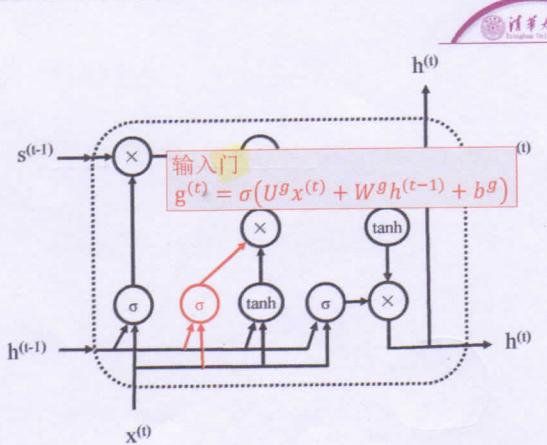
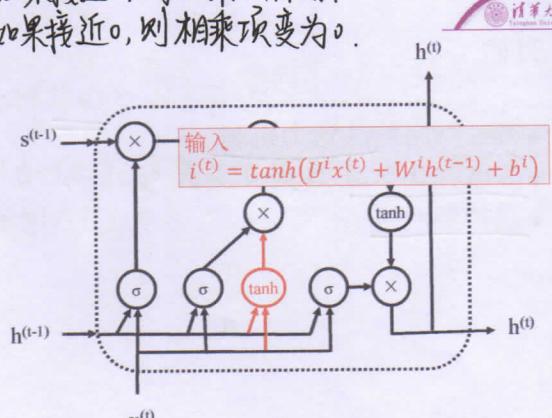
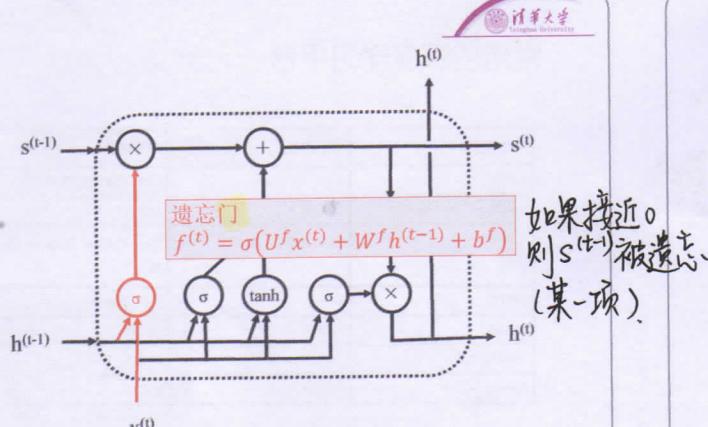
6. 门 用来和别人相乘(选择)

如果接近1，则相乘项保留。

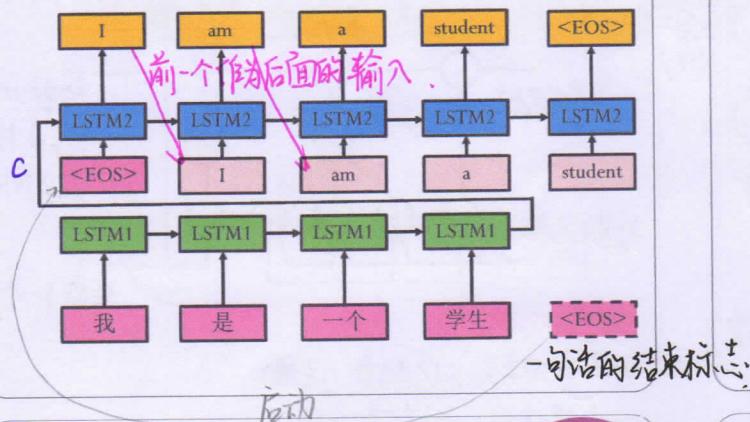
如果接近0，则相乘项变为0。



$h^{(t)}$



## 举例：机器翻译



## 其他

- ◆ 神经网络的注意力机制
- ◆ 深度强化学习 根据胜负调整之前的一系列策略
- ◆ 图神经网络 GNN (决定如何分配)

## 练习题

- ◆ 采用循环神经网络实现拼音到汉字的转换。

## 常用的深度学习平台

平台名称	底层语言	操作语言
Caffe	C++	C++, Python, Matlab
Torch	Pytorch	Lua, C, C++ Python
MXNet		C++, Python, Julia, Scala 等
CNTK	C++	C++, Python
Theano	Python, C	Python
TensorFlow	C++, Python	Python, C++
Keras	Python,	Python

## 小结

- ◆ 神经元
- ◆ 全连接网络
- ◆ BP算法
- ◆ 卷积网络
- ◆ 泛化问题
- ◆ 梯度消失问题
- ◆ 词向量
- ◆ 循环网络