第4章

人工神经网络

Artificial Neural Network

向 世 明

smxiang@nlpr.ia.ac.cn

http://www.escience.cn/people/smxiang/index.html

时空数据分析与学习课题组(STDAL)

中科院自动化研究所 模式识别国家重点实验室

助教： 方深 (shen.fang@nlpr.ia.ac.cn)

内容提要

- 前向神经网络
- 卷积神经网络
- 全卷积神经网络
- 反卷积神经网络
- GoogleNet
- ResNet
- Recurrent NN （不讲）
- LTSM （不讲）

4.1 前向神经网络

4.1.1 人工神经网络

大脑里约有800亿神经细胞。每个神经细胞由两个部分组成：细胞体和突起

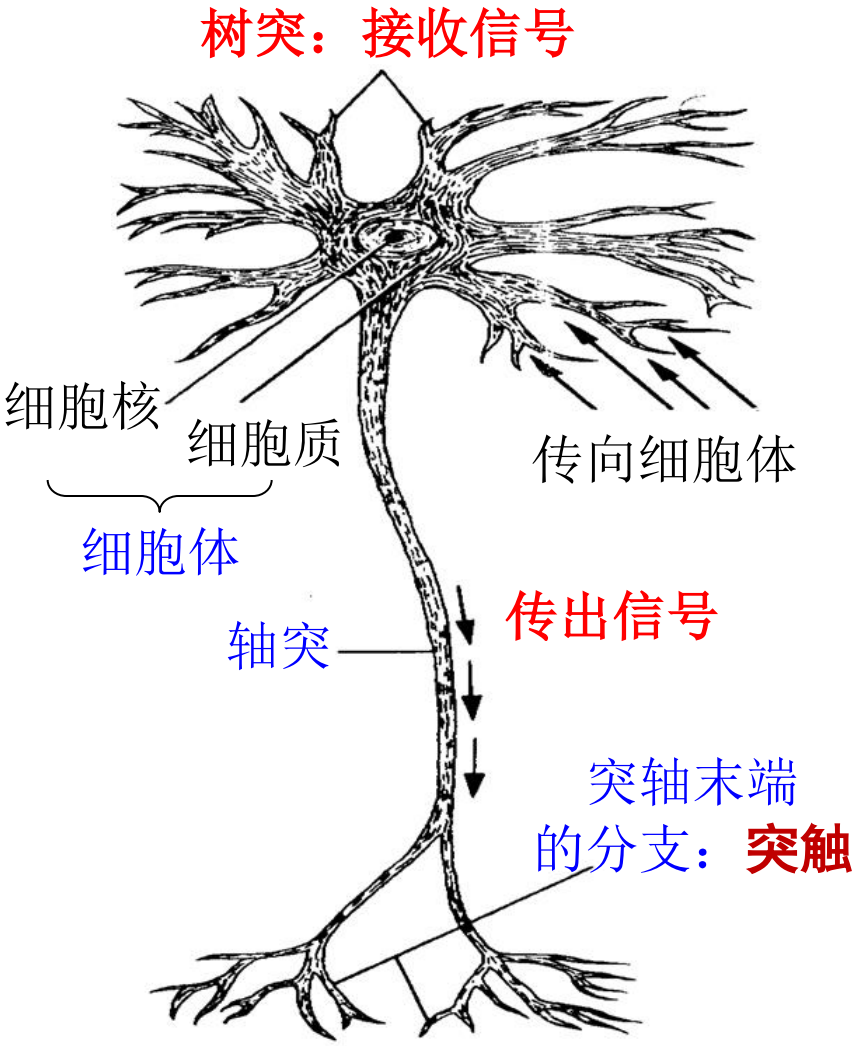
- **Biological Inspiration**

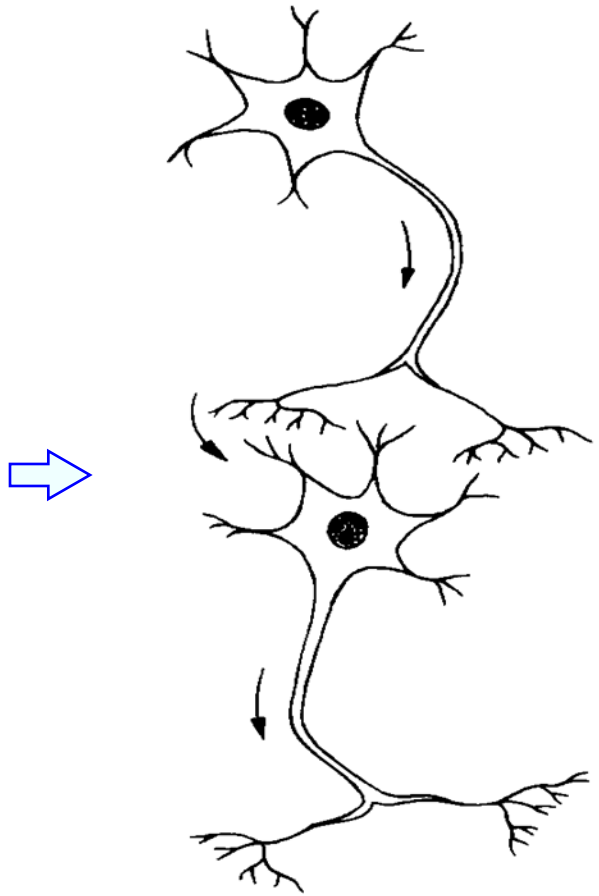
- 神经元之间通过突触两两相连。
- 轴突记录了神经元间联系的强弱。
- 只有达到一定的兴奋程度，神经元才向外界传输信息。
- 每个神经元可抽象成一个激励函数（非线性处理）。

让计算机更加鲁棒、更加智能、且具有学习功能!

神经元结构

- ✓ 神经元之间通过突触两两相连。
- ✓ 轴突记录了神经元间联系的强弱。
- ✓ 只有达到一定的兴奋程度，神经元才向外界传输信息。
- ✓ 每个神经元可抽象成一个激励函数（非线性处理）。

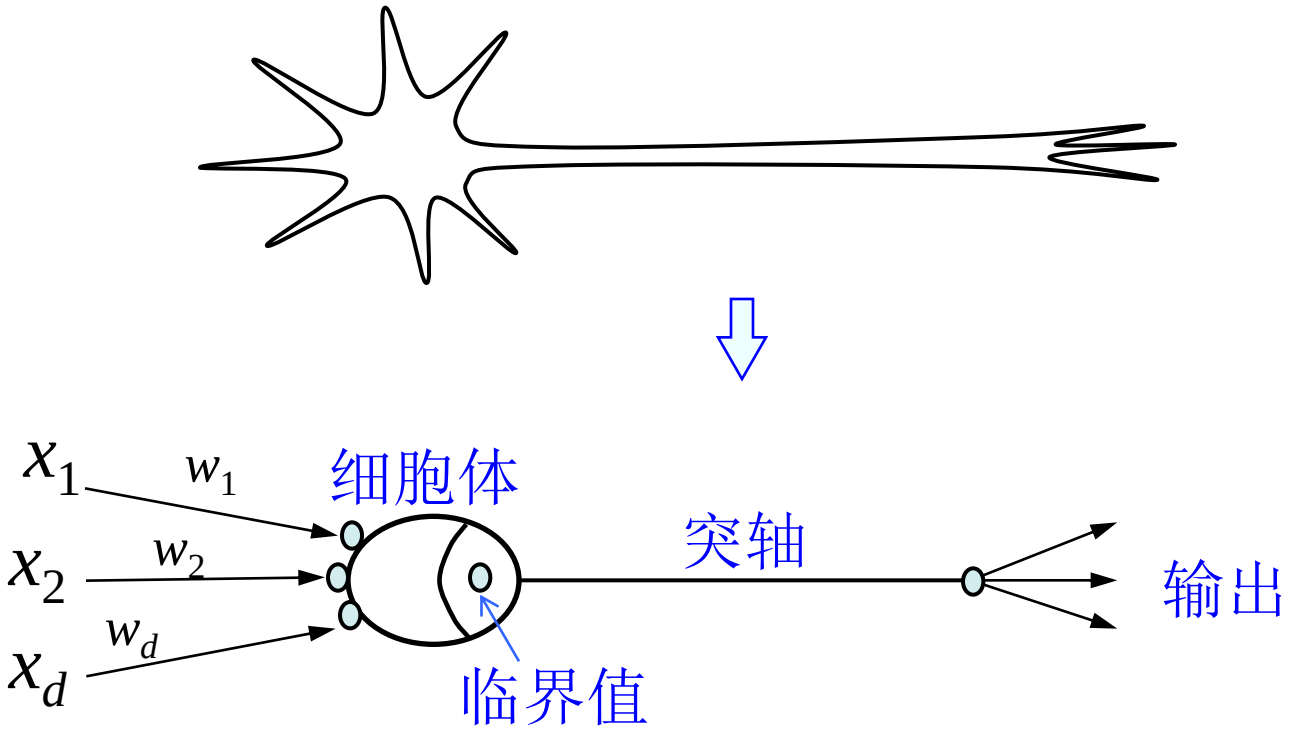




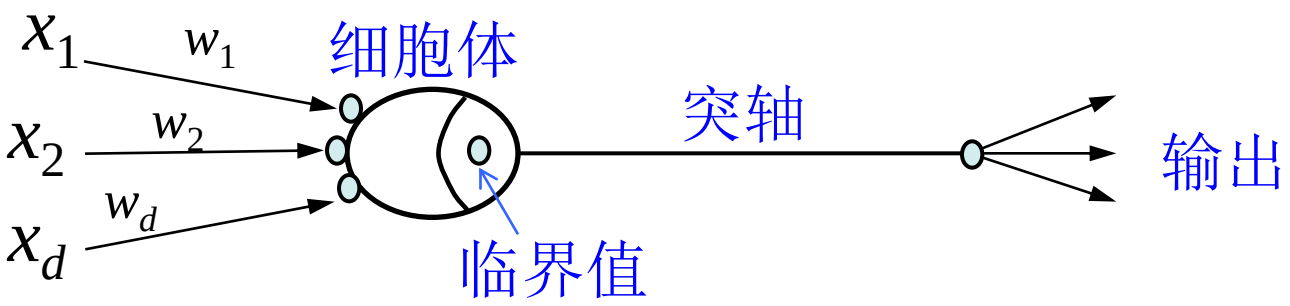
神经元间的信号**通过突触传递。**

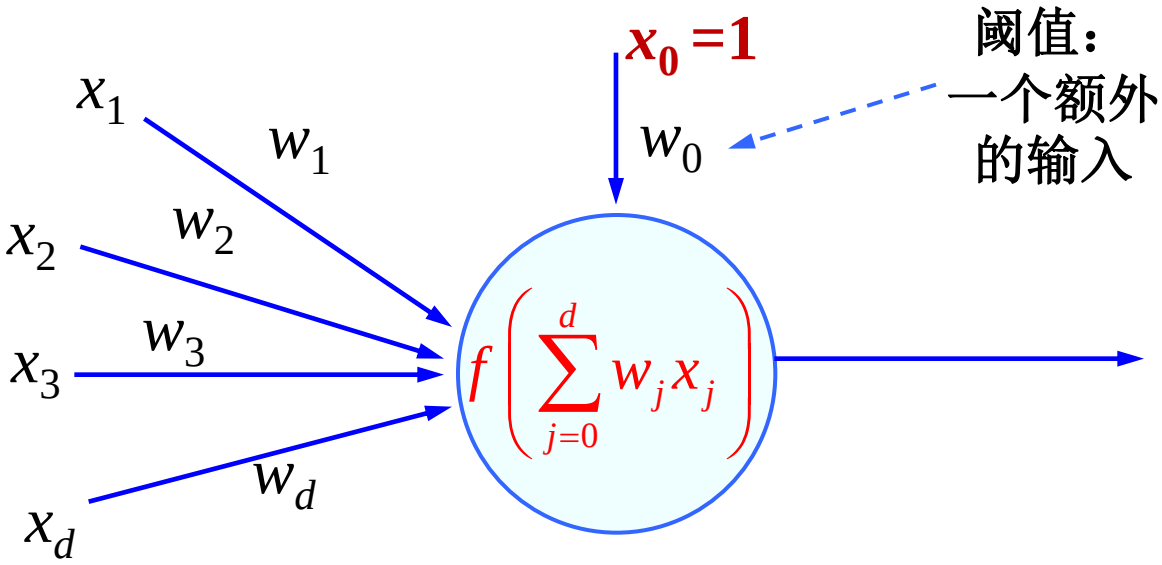
4.1.1 人工神经网络

- 神经元：人工神经网络处理单元



- 处理单元的基本结构和功能



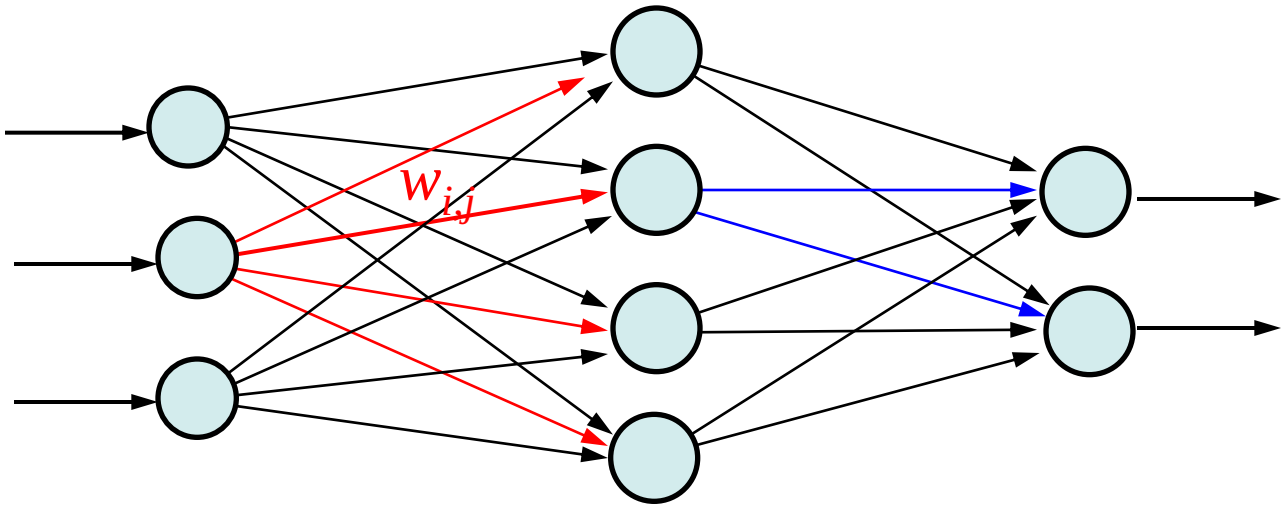


三功能
加权
求和
激励

x: 收到信号　**w**: 信号的权重　$f(.)$: 激励函数

4.1.1 人工神经网络

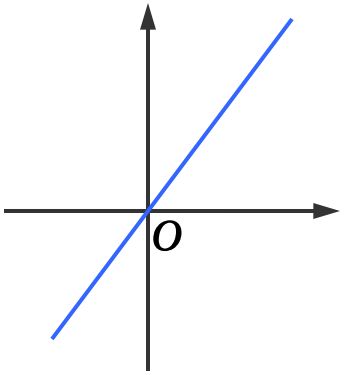
- 多个神经元级联网络
  - 虽然目前有很多网络模型，但它们的结点基本上都是按层排列的。这一点模仿了大脑皮层中的网络模块。
  - **多层网络是由单层网络进行级联构成的**，即上一层的输出作为下一层的输入。



4.1.2 激励函数

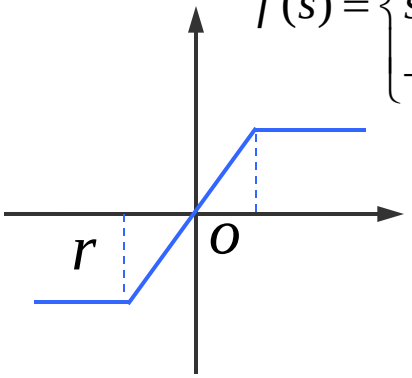
- 激励函数（转移函数、激活函数）
  - 转移函数f (•)，也称激励函数、传输函数或限幅函数，其作用是将可能的无限域变换到指定的有限范围内进行输出（类似于生物神经元的非线性转移特性）。
  - **常用的转移函数：**
    - 线性函数
    - 斜坡函数
    - 阶跃函数
    - 符号函数
    - Sigmoid函数
    - 双曲正切函数

- 转移函数（激活函数）

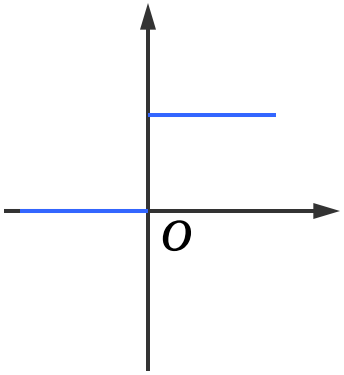


线性函数

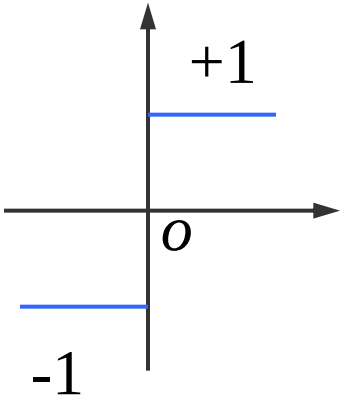
$$f(s)=\begin{cases} r, & s>r \\ s, & |s|\le r \\ -r, & s<-r \end{cases}$$



斜坡函数

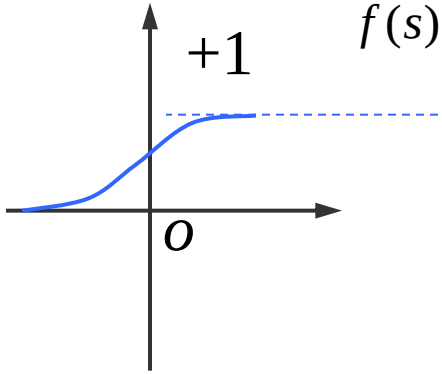


阶跃函数



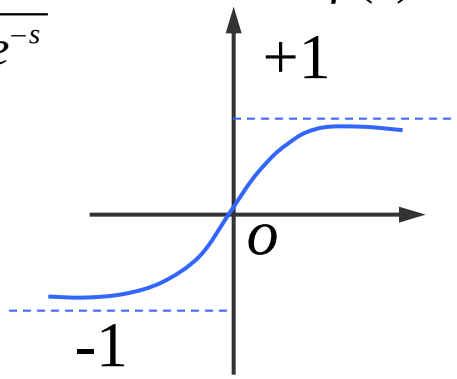
符号函数

$$f(s)=\frac{1}{1+e^{-s}}$$



Sigmoid函数

$$f(s)=\frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$$



双曲正切函数(tanh)

4.1.2 激励函数

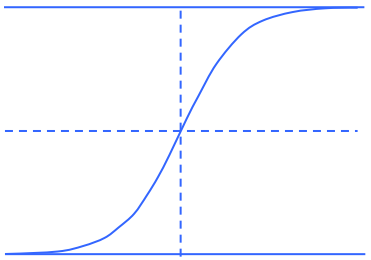
- 目前，深度学习在计算机视觉领域取得了引人注目的成果。其中，激活函数的发展也起到了重要的作用。
- 新型激活函数ReLU在一定程度上克服了梯度消失，使得深度网络的直接监督式训练成为可能。
- Bengio 等在 ICML2016 的文章“Noisy Activation Functions”中有如下定义：
  - 激活函数是映射 $h:R\rightarrow R$，且几乎处处可导。

- 软饱和函数性质：$\lim_{s\rightarrow\infty} f'(s)=0$

4.1.2 激励函数

- 软饱和函数性质：$\lim_{s\to\infty} f'(s)=0$
  - **左饱和：** $\lim_{s\to-\infty} f'(s)=0$
  - **右饱和：** $\lim_{s\to+\infty} f'(s)=0$

$$f(s)=\frac{1}{1+e^{-s}}$$



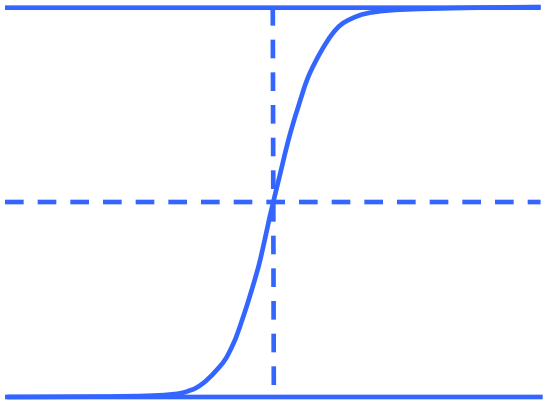
- Sigmoid 函数的软饱和性，使得深度神经网络在二三十年里一直难以有效的训练，是阻碍神经网络发展的重要原因。

- 硬饱和函数：与软饱和相对的是硬饱和激活函数，即：$f'(x)=0$，当 $|x|>c$，其中 c 为常数。同理，硬饱和也分为左饱和和右饱和。

4.1.2 激励函数

- 双曲正切函数
  - tanh(x)=2sigmoid(2x)-1
  - **软饱和函数**
  - tanh 网络的收敛速度要比sigmoid快

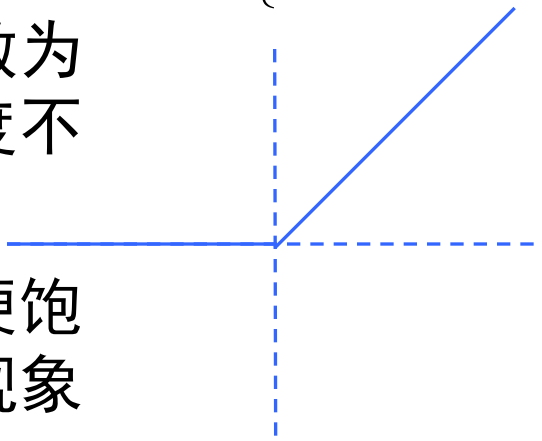
$$f(s)=\frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$$



4.1.2 激励函数

- **ReLU**

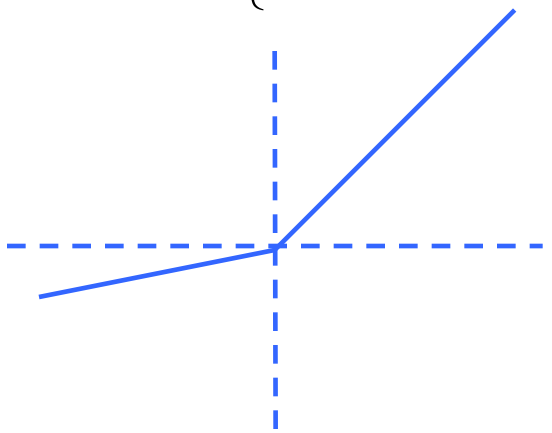
$$f(s)=\begin{cases} s & s \geq 0 \\ 0 & s < 0 \end{cases}$$



  - ReLU 在$x<0$ 时硬饱和。由于 $x>0$时导数为1，所以，ReLU 能够在$x>0$时保持梯度不衰减，从而缓解梯度消失问题。
  - 但随着训练的推进，部分输入会落入硬饱和区，导致对应权重无法更新。这种现象被称为“**神经元死亡**”。
  - ReLU还经常被“诟病”的一个问题是输出具有偏移现象，即输出均值恒大于零。
  - 偏移现象和 神经元死亡会共同影响网络的收敛性。

4.1.2 激励函数

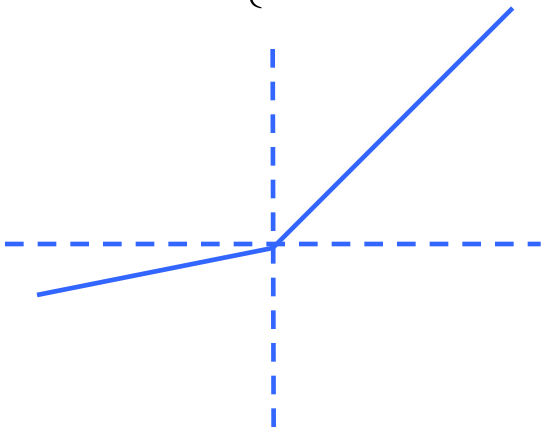
$$f(s)=\begin{cases} s & s\geq 0 \\ as & s<0 \end{cases}$$



- PReLU
  - 与LReLU相比，PReLU中的负半轴斜率a可学习而非固定。
  - 虽然PReLU 引入了额外的参数，但基本不需要担心过拟合。
  - **但也有可能导致性能有一定的下降。**
- RReLU
  - 数学形式与PReLU类似，但RReLU是一种非确定性激活函数，**其参数是随机的**。这种随机性类似于一种噪声，能够在一定程度上起到正则效果。

4.1.2 激励函数

$$f(s)=\begin{cases} s & s\geq 0 \\ as & s<0 \end{cases}$$



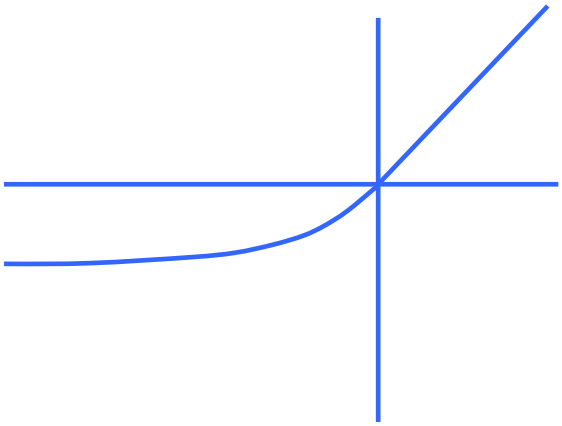
- **MAXOut**
  - MAXOout是 ReLU 的推广，其发生饱和是一个零测集事件（measure zero event），即最大化如下运算：

$$\max\left(\mathbf{w}_1^T\mathbf{x}+b_1,\mathbf{w}_2^T\mathbf{x}+b_2,\cdots,\mathbf{w}_n^T\mathbf{x}+b_n\right)$$

4.1.2 激励函数

$$f(s)=\begin{cases} s & s \geq 0 \\ a(exp(s)-1), & s<0 \end{cases}$$

- **ELU(指数线性单元)**
  - ELU融合了sigmoid和ReLU，具有左侧软饱性。
  - 右侧线性部分使得ELU能够缓解梯度消失，而左侧软饱能够让ELU对输入变化或噪声更鲁棒。

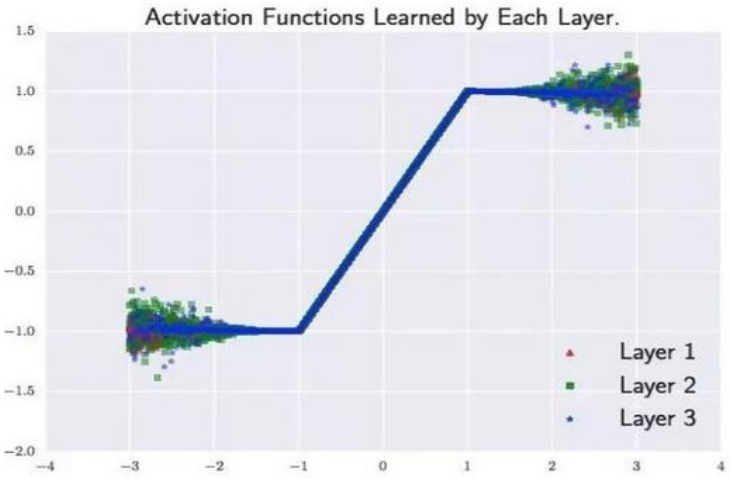


4.1.2 激励函数

- **Noisy Activation Functions**
  - Bengio教授在ICML 2016 提出了一种激活策略，可用于多种软饱和激活函数，例如 sigmoid和 tanh

Algorithm 1 Noisy Activations with Half-Normal Noise for Hard-Saturating Functions

$\Delta \leftarrow h(x) - u(x)$
$\mathrm{d}(x) \leftarrow -\operatorname{sgn}(x)\operatorname{sgn}(1-\alpha)$
$\sigma(x) \leftarrow c\,(g(p\Delta) - 0.5)^2$
$\xi \sim \mathcal{N}(0,\ 1)$
$\phi(x, \xi) \leftarrow \alpha \mathrm{h}(x) + (1-\alpha)\mathrm{u}(x) + (\mathrm{d}(x)\sigma(x)|\xi|)$



4.1.2 激励函数

- **CReLU**
  - CReLU是Wenling Shang 发表在 ICML 2016的工作：

$$\forall s \in R, \quad \rho_c(x) = \left([s]_+, [-s]_+\right)$$

其中，[]表示ReLU函数或其它函数

4.1.2 激励函数

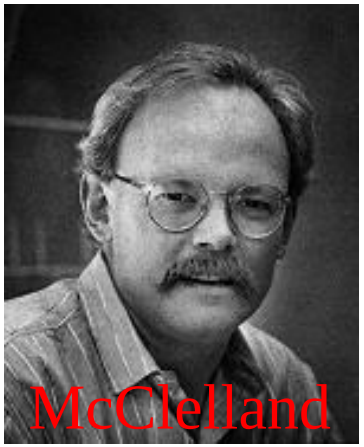
- **MPELU:**将分段线性与ELU统一到一种形式：

$$f(s)=\begin{cases} s, & s>0 \\ \alpha_c\left(e^{\beta_c s}-1\right), & s\le 0 \end{cases}$$

  - α 和 β可以使用正则。α, β 固定为1时，MPELU 退化为 ELU； β 固定为很小的值时，MPELU 近似为 PReLU；当α=0，MPELU 等价于 ReLU。
  - MPELU 的优势在于具备 ReLU、PReLU和 ELU的优点。
    - MPELU具备ELU的收敛性质，能够在无 Batch Normalization 的情况下让几十层网络收敛。
    - 作为一般化形式， MPELU较三者的推广能力更强，即MPELU = max(ReLU, PReLU, ELU)。

4.1.3 误差反向传播算法

- D. Rumelhart，J. McClelland于1985年提出了误差反向传播(Back Propagation, BP)学习算法



- 基本原理
  - 利用输出后的误差来估计**输出层的前一层**的误差，再用这个误差估计更前一层的误差，如此一层一层地反传下去，从而获得所有其它各层的误差估计

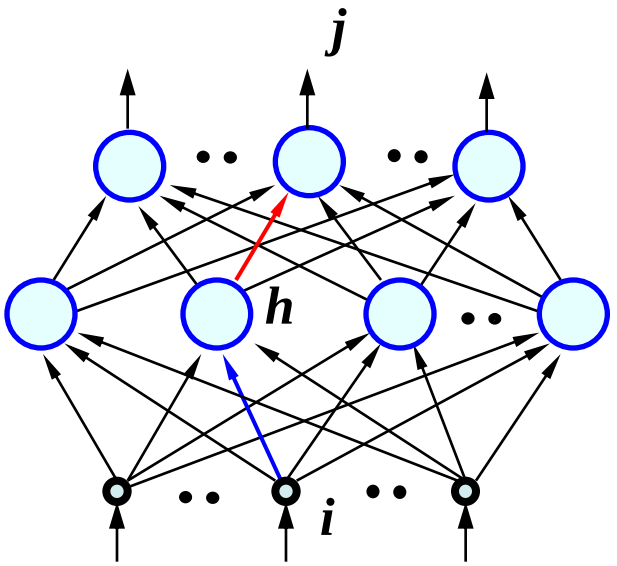
4.1.3 误差反向传播算法

- 误差反向传播训练算法
  - 属于监督学习算法，**通过调节各层的权重，使网络学会**由“输入-输出对”组成的训练组。
  - BP算法核心是梯度下降法。
  - 权重先从输出层开始修正，再依次修正各层权重
    - 首先修正：**“输出层至最后一个隐含层”**的连接权重
    - 再修正：**“最后一个隐含层至倒数第二个隐含层”**的连接权重，….
    - 最后修正：**“第一隐含层至输入层”**的连接权重。

学习的本质：对网络各连接权重作动态调整！

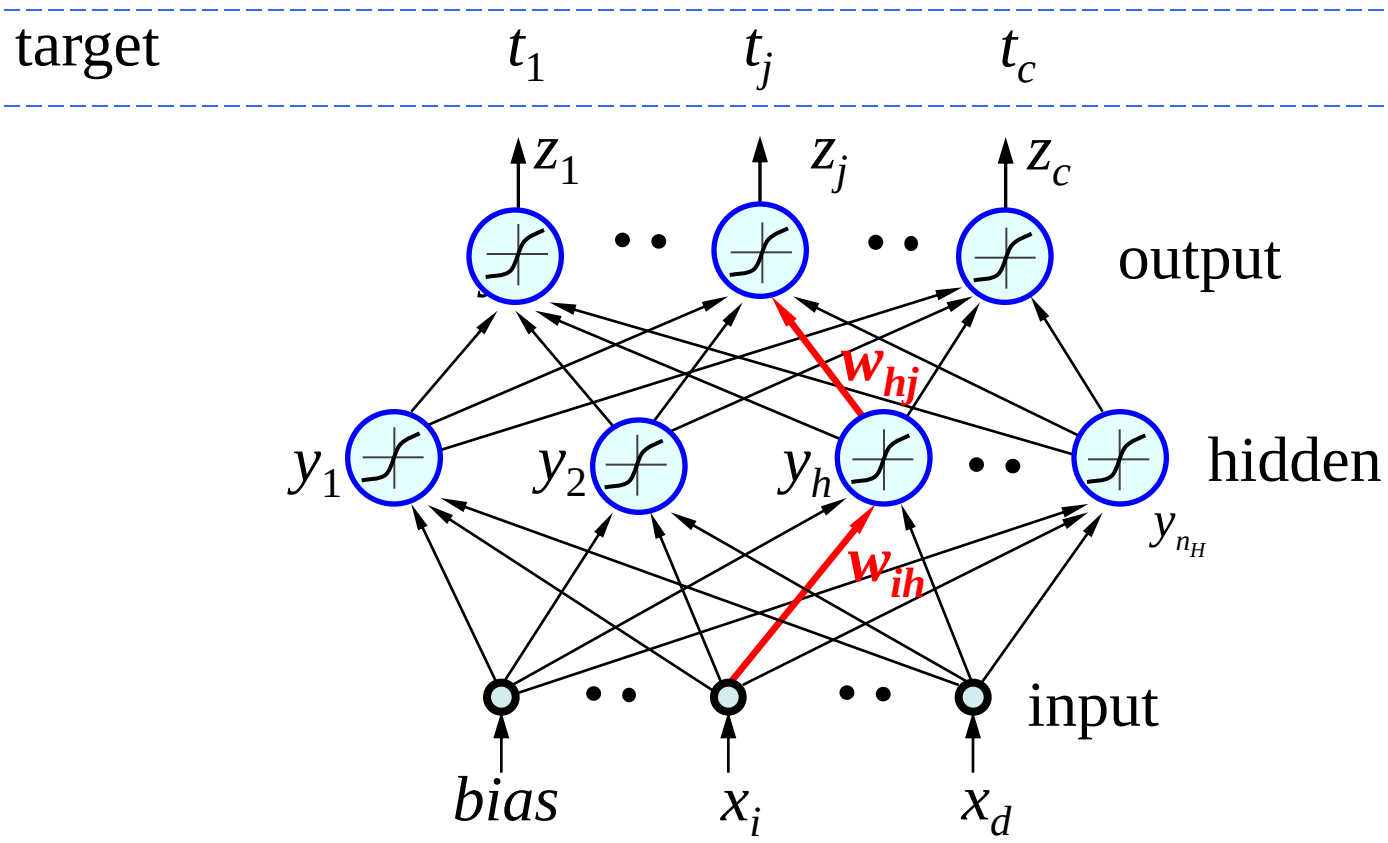
4.1.3 误差反向传播算法

- 三层网络的描述
  - 训练数据输入输出对：$\{x_i^k, t_j^k\}$
  - 输出层结点的输出：z_j^k
  - 隐含层结点的输出：y_h^k
  - 输入信号：x_i^k
  - 输入端点数目：$d+1$
  - 输入层结点 i 至隐含层结点 h 的权重：w_{ih}
  - 隐含层结点 h 至输出层结点 j 的加权表示：w_{hj}
  - 上标 k 表示训练对的序号，$k=1, 2, \ldots, n$



4.1.3 误差反向传播算法

- **Hope:** $z_1 \approx t_1, \cdots, z_c \approx t_c$, **for all samples:** $J(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{c}(t_j - z_j)^2 \approx 0$



- 网络描述-每个样本所经历的计算

上标 k 联系
第 k 个样本

对第 k 个样本，隐含层 h 结点的输入加权和为：

$$net_h^k = \sum_i w_{ih} x_i^k$$

经过激励，隐含层 h 结点的输出：

$$y_h^k = f(net_h^k) = f\left(\sum_i w_{ih} x_i^k\right)$$

输出层 j 结点的输入加权和为：

$$net_j^k = \sum_h w_{hj} y_h^k = \sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)$$

经过激励，输出层 j 结点的输出：

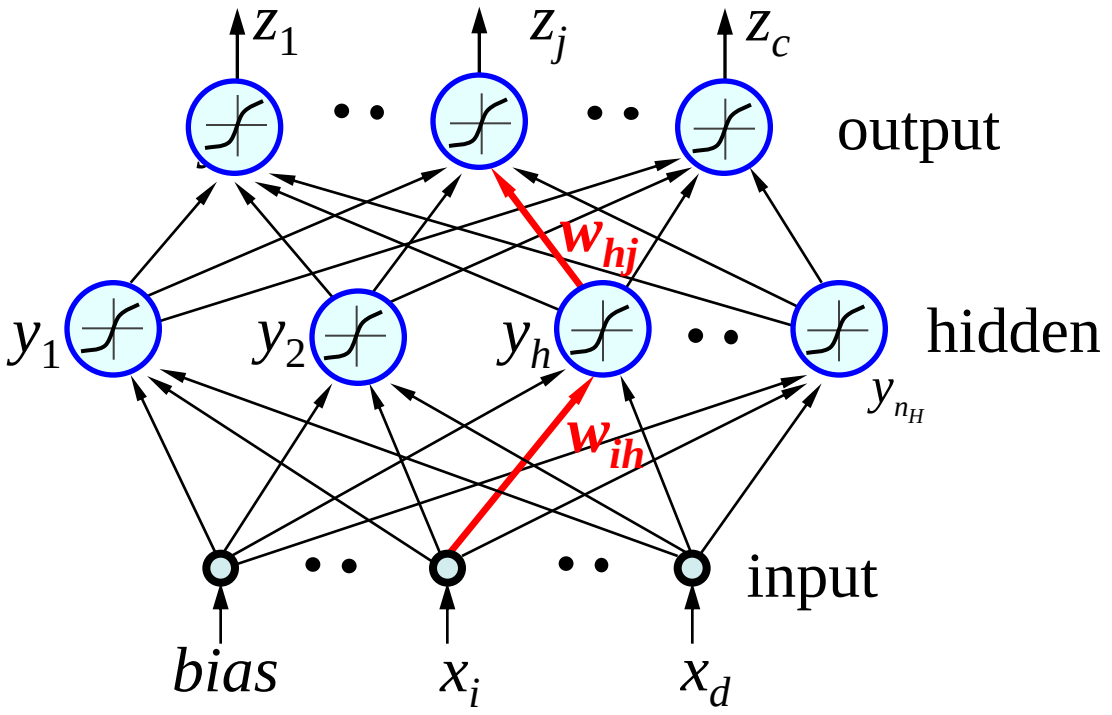
$$z_j^k = f(net_j^k) = f\left(\sum_h w_{hj} y_h^k\right) = f\left(\sum_h w_{hj} f(\sum_i w_{ih} x_i^k)\right)$$

- 网络描述–每个样本所经历的计算

上标k：第 k 个样本

$$z_j^k = f\left(\sum_h w_{hj} y_h^k\right)$$

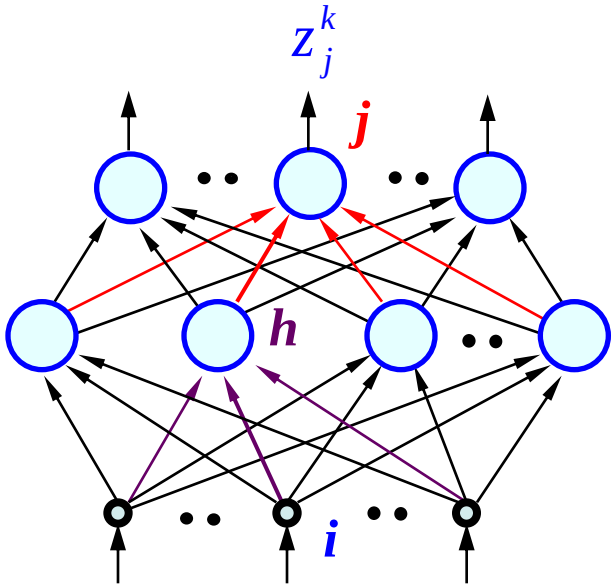
$$y_h^k = f\left(\sum_i w_{ih} x_i^k\right)$$



$$z_j^k = f(net_j^k) = f\left(\sum_h w_{hj} y_h^k\right) = f\left(\sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)\right)$$

- 误差函数—单个样本

上标 k 联系
第 k 个样本



$$z_j^k = f\left(net_j^k\right) = f\left(\sum_h w_{hj} y_h^k\right)$$

$$y_h^k = f\left(net_h^k\right) = f\left(\sum_i w_{ih} x_i^k\right)$$

$$E(\mathbf{w})^k = J(\mathbf{w})^k = \frac{1}{2}\sum_j (t_j^k - z_j^k)^2$$

$$= \frac{1}{2}\sum_j \left(t_j^k - f(net_j^k)\right)^2$$

$$= \frac{1}{2}\sum_j \left\{t_j^k - f\left(\sum_h w_{hj} y_h^k\right)\right\}^2$$

$$= \frac{1}{2}\sum_j \left\{t_j^k - f\left(\sum_h w_{hj} f\left(net_h^k\right)\right)\right\}^2$$

$$= \frac{1}{2}\sum_j \left\{t_j^k - f\left(\sum_h w_{hj} f\left(\sum_i w_{ih} x_i^k\right)\right)\right\}^2$$

- 网络训练：隐含层－输出层

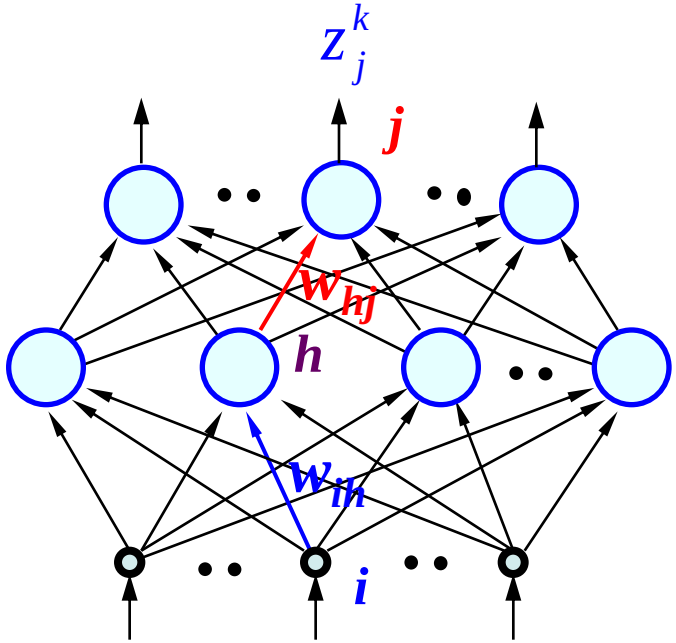
隐含层到输出层的连接权重调节量：

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \sum_k \frac{\partial E}{\partial net_j^k} \frac{\partial net_j^k}{\partial w_{hj}}$$

$$= \eta \sum_k (t_j^k - z_j^k) f'(net_j^k) y_h^k$$

$$= \eta \sum_k \delta_j^k y_h^k$$

(δ规则)

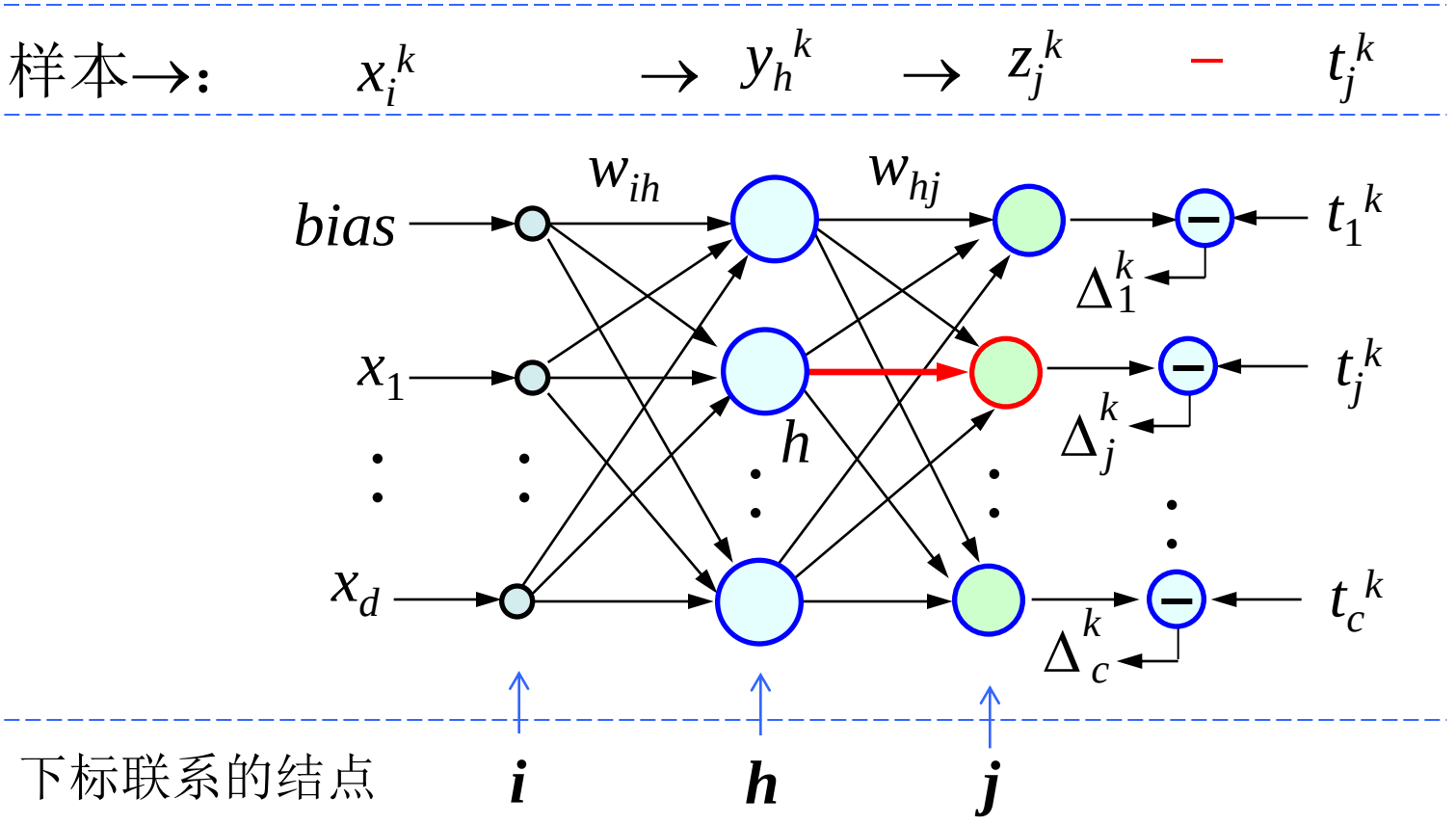


$$\delta_j^k = \frac{-\partial E}{\partial net_j^k} = f'(net_j^k)(t_j^k - z_j^k) = f'(net_j^k)\Delta_j^k,$$

$$\Delta_j^k = t_j^k - z_j^k$$

边的指向结点的误差信号（局部梯度）

- 隐含层—输出层，**准备好输出层的误差：** $\Delta_j^k = t_j^k - z_j^k$



注：上标 k 联系第 k 个样本

- 隐含层—输出层：第 k 个训练样本对权重 w_{hj} 的贡献

$h \to j$, for sample k:

δ规则： $$\Delta w_{hj}\big|_{\text{sample } k} = \eta \delta_j^k y_h^k$$

权重所联边的起始结点（隐含结点 h）的输出

权重所联边的指向结点（输出结点 j）收集到的误差信号

$$\delta_j^k = f'(net_j^k)\Delta_j^k, \quad \Delta_j^k = t_j^k - z_j^k$$

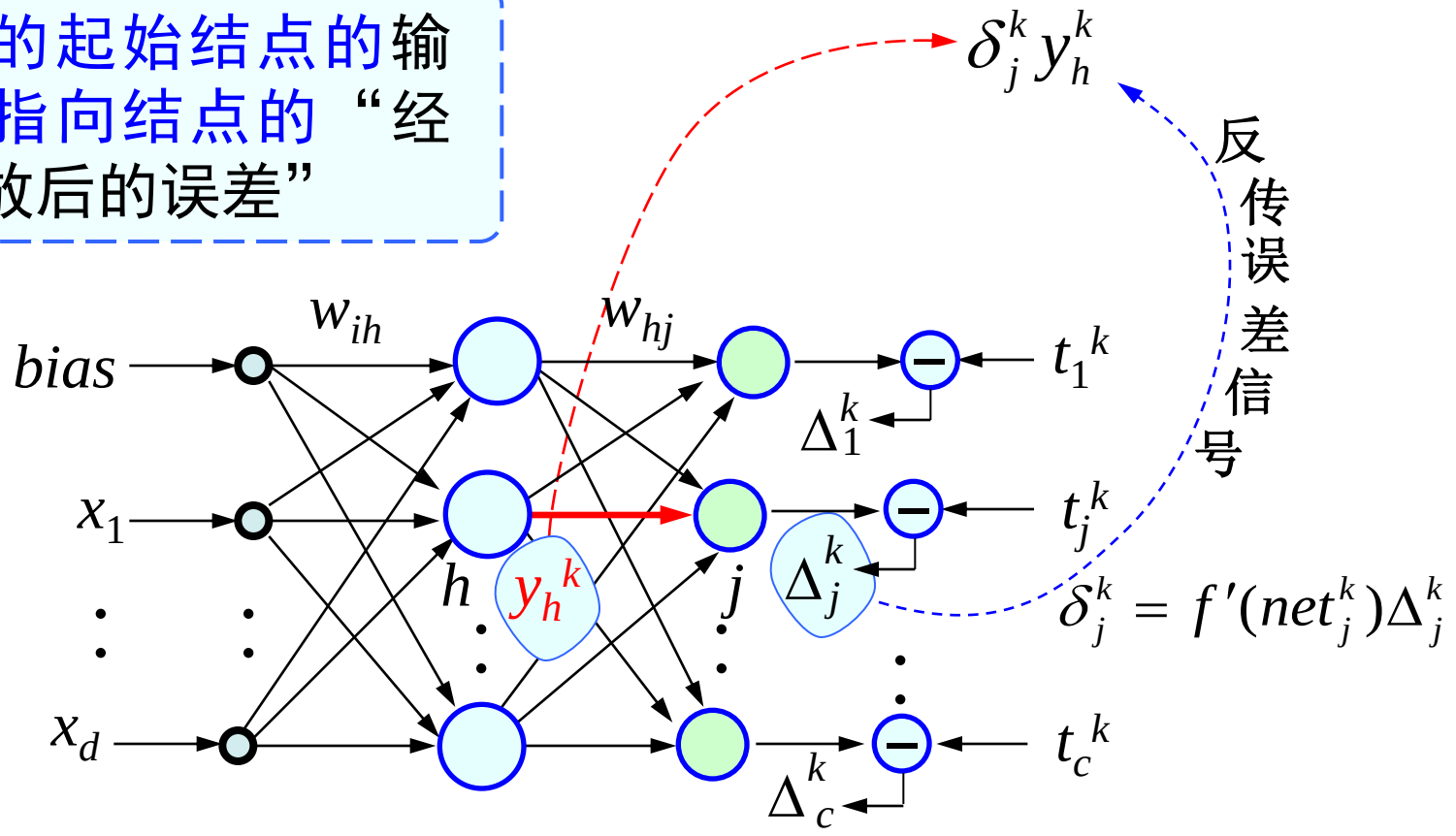
误差**在权重所联边的指向结点处**计算。

误差大小等于：该结点收集到的误差乘以激励函数对“该结点加权和”的导数。

误差反传与权重更新：

所联边的起始结点的输出乘以指向结点的“经导数缩放后的误差”

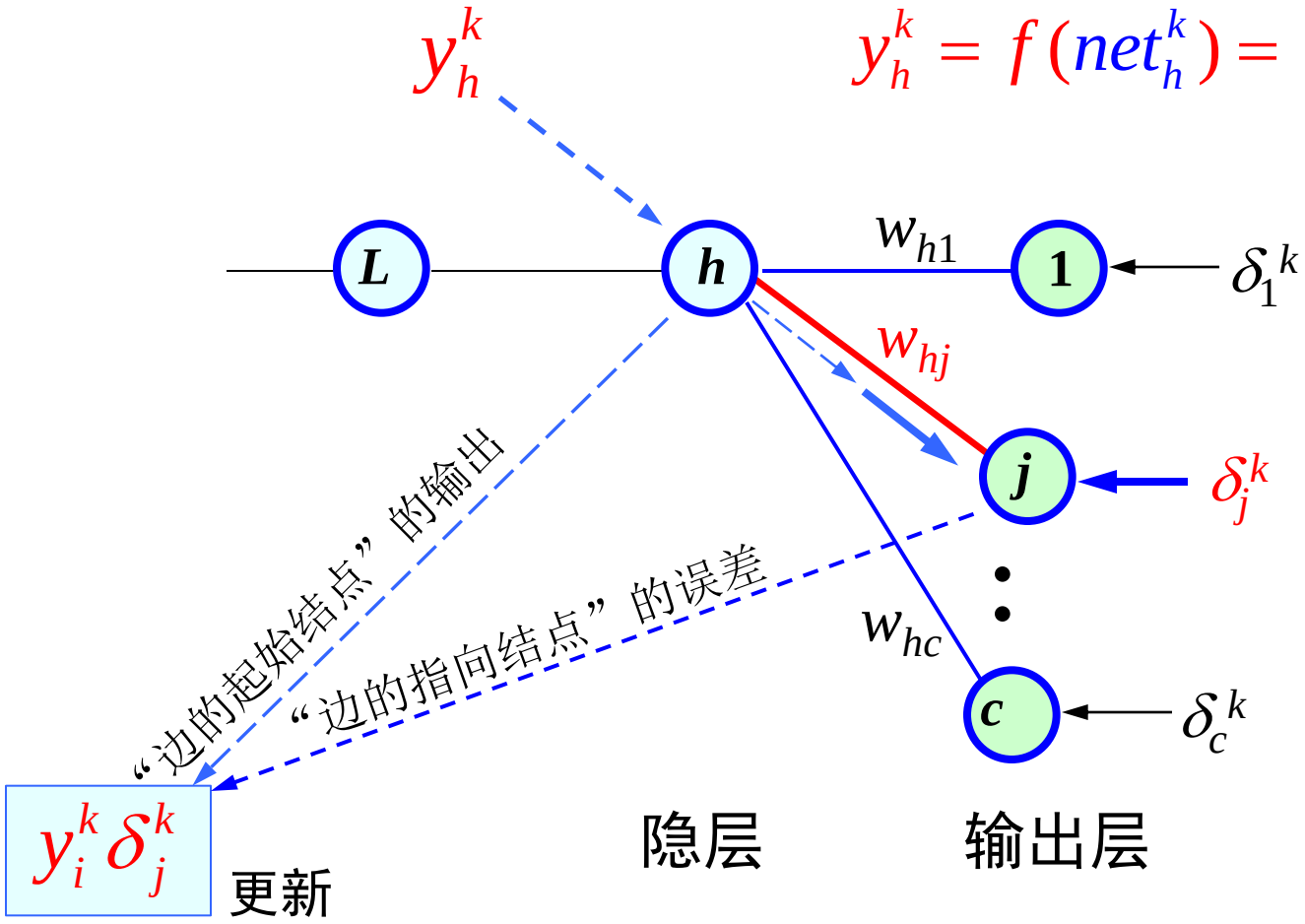
第 k 个样本对权重更新的贡献



隐层-输出权重更新示意：

w_{hj}所连接的边的起始结点向外传递的信号值

$$y_h^k = f(net_h^k) = f\left(\sum_i w_{ih} x_i^k\right)$$

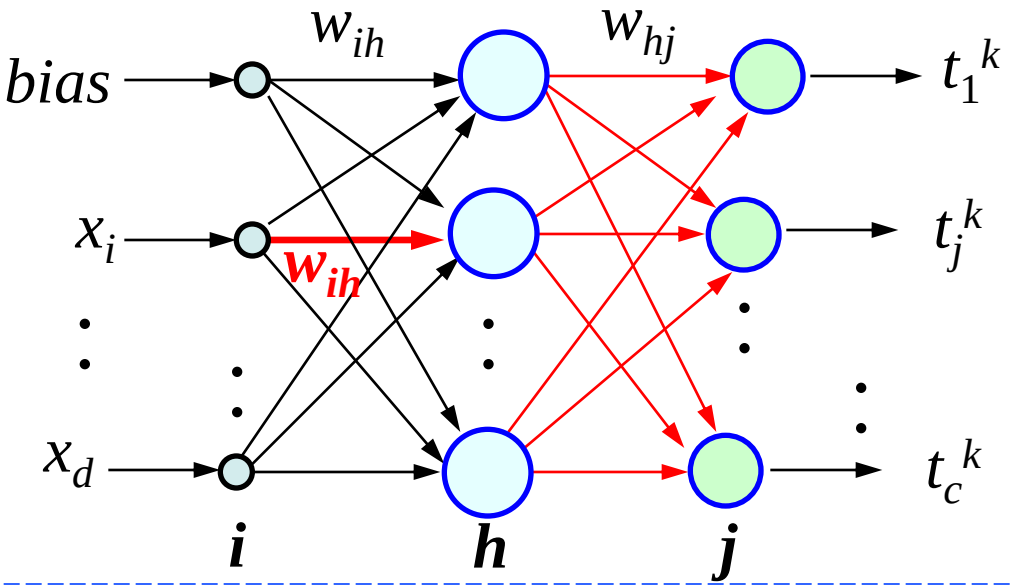


w_{hj}所连接的边的指向结点的误差

对于**输入层到隐含层**结点连接的边的权重修正量 Δw_{ih}，必须考虑将 $E(\mathbf{w})$ 对 w_{ih} 求导，需利用分层链路法。

输入层至隐含层权重更新：

$$E(\mathbf{w})=\sum_k J(\mathbf{w})=\frac{1}{2}\sum_{k,j}(t_j^k-z_j^k)^2=\frac{1}{2}\sum_{k,j}\left\{t_j^k-f\left(\sum_h w_{hj}\,f\left(\sum_i w_{ih}x_i^k\right)\right)\right\}^2$$



样本→：　$x_i^k \rightarrow net_h^k \rightarrow y_h^k \rightarrow net_h^k \rightarrow z_j^k$　　t_j^k

输入层至隐含层权重更新：

用别名$\tilde{h}$代替h

$$E(\mathbf{w})=\frac{1}{2}\sum_{k,j}(t_j^k-z_j^k)^2=\frac{1}{2}\sum_{k,j}\left\{t_j^k-f\left(\sum_{\tilde{h}}w_{\tilde{h}j}f\left(\sum_i w_{i\tilde{h}}x_i^k\right)\right)\right\}^2$$

$$net_{\tilde{h}}^k=\sum_i w_{i\tilde{h}}x_i^k,\quad y_{\tilde{h}}^k=f(net_{\tilde{h}}^k)$$

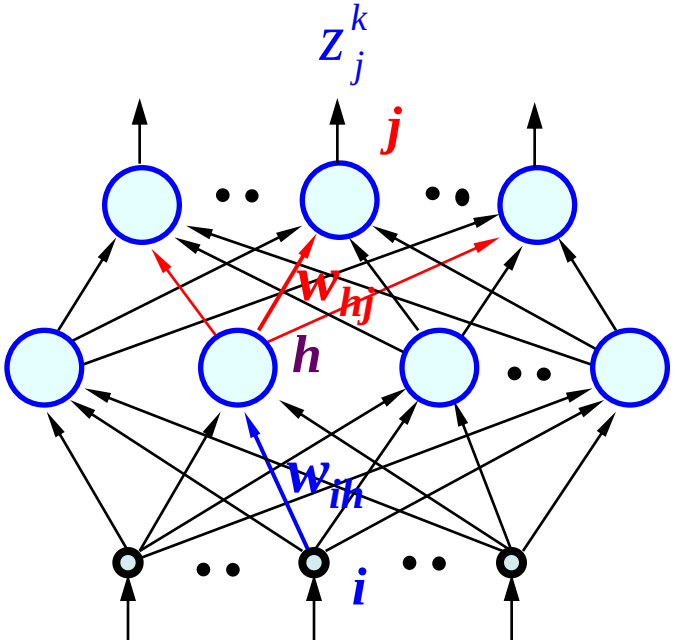
$$net_j^k=\sum_{\tilde{h}} w_{\tilde{h}j}y_{\tilde{h}}^k,\quad z_j^k=f(net_j^k)$$

z_j^k　net_j^k　$y_{\tilde{h}}^k$　$net_{\tilde{h}}^k$

包含权重w_{ih}

待更新权重w_{ih}的增量：

$$\Delta w_{ih} = -\eta \frac{\partial E}{\partial w_{ih}} = -\eta \sum_{k,j} \frac{\partial E}{\partial z_j^k} \frac{\partial z_j^k}{\partial w_{ih}}$$



$$= -\eta \sum_{k,j} \frac{\partial E}{\partial z_j^k} \frac{\partial z_j^k}{\partial net_j^k} \frac{\partial net_j^k}{\partial w_{ih}}$$

$$= -\eta \sum_{k,j} \frac{\partial E}{\partial z_j^k} \frac{\partial z_j^k}{\partial net_j^k} \frac{\partial net_j^k}{\partial y_h^k} \frac{\partial y_h^k}{\partial w_{ih}}$$

$$= -\eta \sum_{k,j} \frac{\partial E}{\partial z_j^k} \frac{\partial z_j^k}{\partial net_j^k} \frac{\partial net_j^k}{\partial y_h^k} \frac{\partial y_h^k}{\partial net_h^k} \frac{\partial net_h^k}{\partial w_{ih}}$$

(链式法则)

$$= -\eta \sum_{k,j} \frac{\partial E}{\partial net_h^k} \frac{\partial net_h^k}{\partial w_{ih}}$$

待更新权重的增量（具体化）

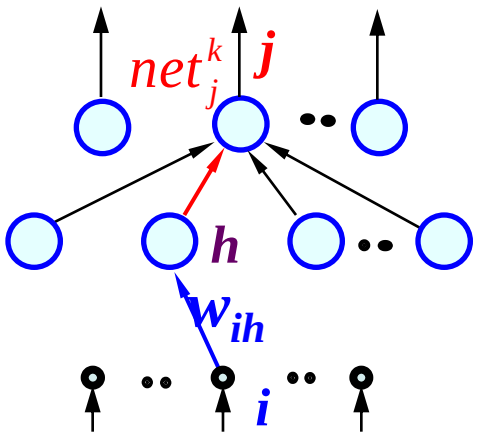
$$\Delta w_{ih} = -\eta \frac{\partial E}{\partial w_{ih}} = -\eta \sum_{k,j} \frac{\partial E}{\partial z_j^k} \cdot \frac{\partial z_j^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k \right) \frac{\partial z_j^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k \right) \frac{\partial z_j^k}{\partial net_j^k} \frac{\partial net_j^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k \right) f'\left(net_j^k \right) \frac{\partial net_j^k}{\partial y_h^k} \frac{\partial y_h^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k \right) f'\left(net_j^k \right) w_{hj} \frac{\partial y_h^k}{\partial w_{ih}}$$



$$net_j^k = \sum_{\tilde{h}} w_{\tilde{h}j} y_{\tilde{h}}^k$$

（只有当$\tilde{h} = h$时y_h^k才包含w_{ih}）

（接前一页）

$$\Delta w_{ih} = \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} \frac{\partial y_h^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} \frac{\partial y_h^k}{\partial net_h^k} \frac{\partial net_h^k}{\partial w_{ih}}$$

$$= \eta \sum_{k,j} \left(t_j^k - z_j^k\right) f'\left(net_j^k\right) w_{hj} f'\left(net_h^k\right) x_i^k$$

$$= \eta \sum_{k,j} \delta_j^k w_{hj} f'(net_h^k) x_i^k$$

$$\delta_j^k = f'\left(net_j^k\right)\left(t_j^k - z_j^k\right)$$

$$= \eta \sum_{k} \left(f'(net_h^k) \sum_j \delta_j^k w_{hj} \right) x_i^k$$

$$= \eta \sum_{k} \delta_h^k x_i^k$$

$$\delta_h^k = \frac{-\partial E}{\partial net_h^k} = f'(net_h^k) \sum_j w_{hj} \delta_j^k = f'(net_h^k) \Delta_h^k, \quad \Delta_h^k = \sum_j w_{hj} \delta_j^k$$

输入-隐层：第 k 个训练样本对权重 w_{ih} 的贡献

$i \to h$, for sample k:

δ规则：

$$\Delta w_{ih}\,|_{sample\ k} = \eta \delta_h^k x_i^k$$

w_{ih}**所连接的边的起始结点**（输入层结点 i）的输出(此时即为样本第 i 个分量)

w_{ih}**所连接的边的指向结点**（隐含结点 h）**收集到的误差信号**

输入-隐层：第 k 个训练样本对权重 w_{ih} 的贡献

$$\delta_h^k = \frac{-\partial E}{\partial net_h^k} = f'(net_h^k) \sum_j w_{hj}\delta_j^k$$

从前一层收集误差：加权和

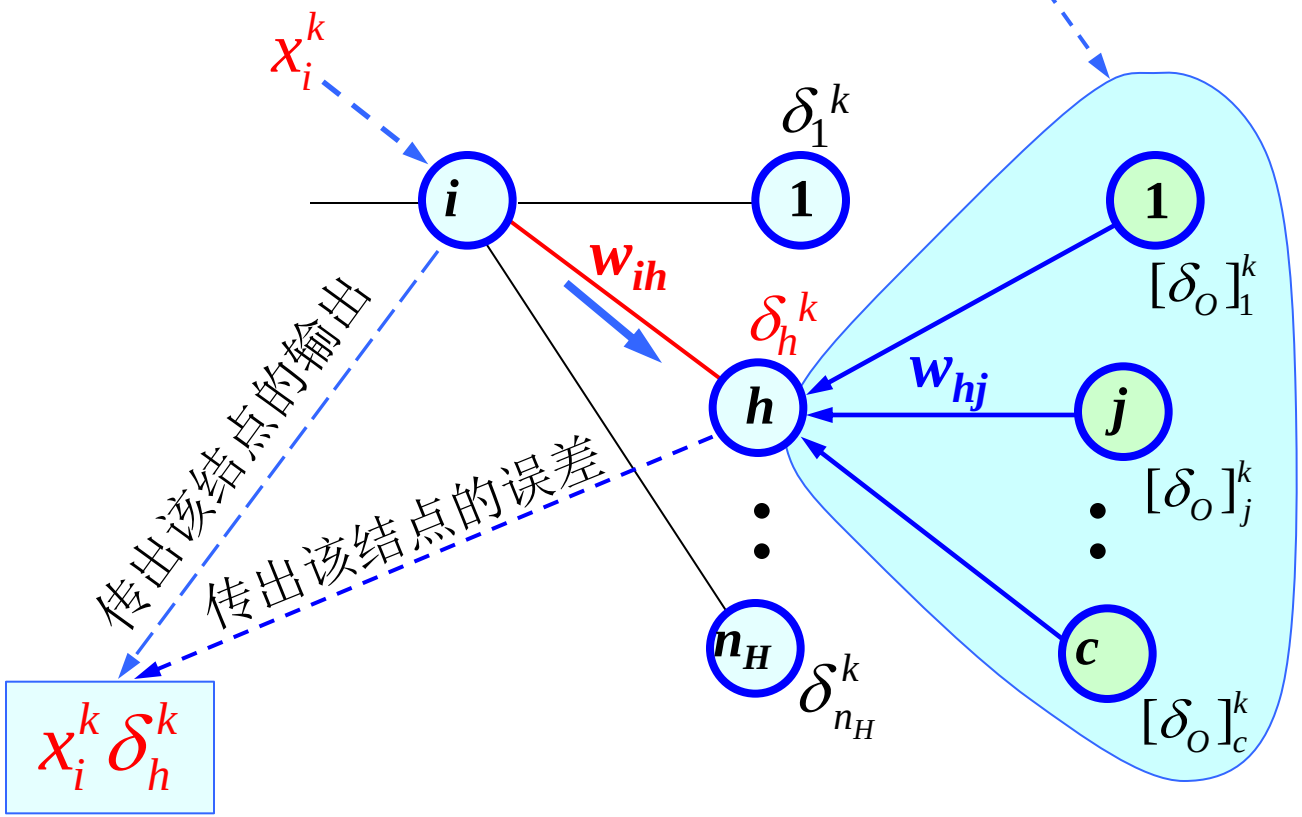
误差**在权重所联边的指向结点处**计算。

误差大小等于：该结点收集到的误差乘以激励函数对“该结点加权和”的导数。

输入—隐层权重更新示意：

误差收集: $\delta_h^k = f'(net_h^k)\sum_j w_{hj}[\delta o]_j^k$

注：已将上页中的 δ 更为 δ_O (对输出层)



x_i^k ： 边*i-h* 起点的输出，即向外传递的信号值

4.1.3 误差反向传播算法

- 网络训练
  - BP算法对任意层的加权修正量的一般形式：

$$\Delta w_{in\to o} = \eta \sum_{all\ samples} \delta_o y_{in}$$

从后一层各结点h收集误差

  - 单个训练样本的贡献：

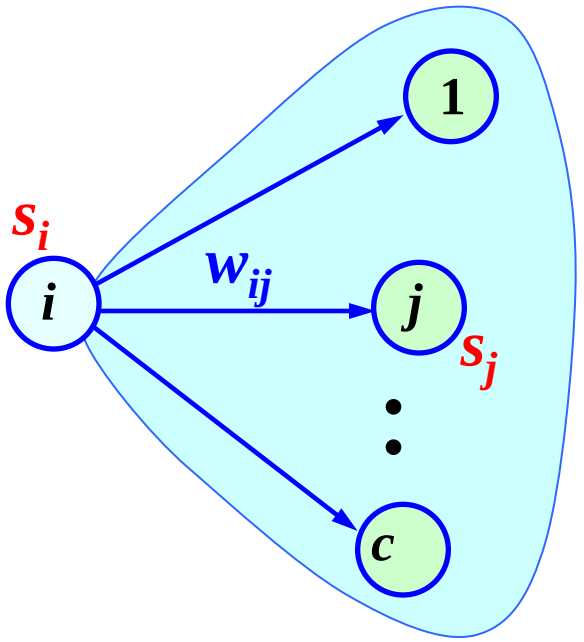
$$\Delta w_{in\to o} = \eta \cdot \delta_o \cdot y_{in} = \eta \cdot \left(\sum_h w_{o\to h} [\delta_o]_h \right) \cdot y_{in}$$

下标 in 和 o 分别指**“待更新权重”**所连边的起始结点和指向结点，y_{in}代表起始结点的实际输出，δ_o表示指向结点的误差 (由后一层收集得到)。

- 网络训练
  - 从更一般的角度来认识网络
    - 目标函数不是误差平方损失，比如交叉熵、softmax、hinge loss等，对于权重更新是否有上述同样的文字表述？
  - 目标函数对某一层结点 i 的输出 s_i 的梯度（见右上）。
  - 目标函数对权重的梯度(导数)：

$$\nabla_{w_{ij}} E = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial s_i} = \sum_{j \in N(i)} \frac{\partial E}{\partial s_j} \frac{\partial s_j}{s_i}$$



$j \in$ {结点 i 指向的所有结点}

4.2 卷积神经网络

4.2 卷积神经网络

- 对于语音等信号，采用时间延迟卷积神经网络可以很好地对信号进行描述学习。
- 对于自然图像，希望直接从图像底层开始进行学习（非结构化特征学习）。即，对于图像识别任务，不必事先提取出人为设计的特征，比如Gabor纹理特征、多尺度小波特征、SIFT特征、HOG特征，等等。
  - 人为设计的特征之缺点：这些特征具有一些参数，如尺度、梯度方向、频域划分等，其泛化能力不强
- 但是，如果以图像直接作为输入，并将每个像素看成一个结点，对于200×200大小的图像，则仅输入层就有4万个结点；如果第一隐含层仅仅只包含1000个结点，则权重数量将达到4千万。这显然是一个巨大的计算负担。

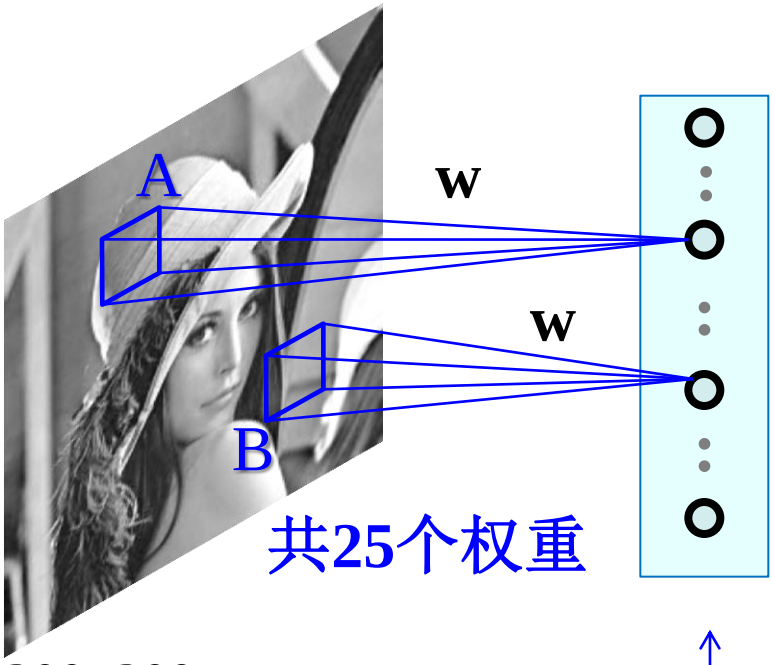
4.2 卷积神经网络

- 降低网络权重数量—局部连接（方法一）
  - **视觉系统局部感受野**
    - 视觉生理学相关研究普遍认为：人对外界的认知是从局部到全局的。视觉皮层的神经元就是局部接受信息的（即只响应某些特定区域的刺激）
  - **图像空间相关性：** 对图像而言，局部邻域内的像素联系较紧密，距离较远的像素相关性则较弱。
- 神经网络由全连接变为部分连接
  - 每个神经元其实没有必要对全局图像进行感知，只需要对局部进行感知。
  - 在更高层将局部的信息综合起来获得全局信息。

4.2 卷积神经网络

- 降低网络权重数量—权值共享（方法二）
  - 局部连接可降低网络的权重数量，但仍不足够。
  - **引入权值共享机制**。这一机制是：**“从图像任何一个局部区域内连接到同一类型的隐含结点，其权重保持不变”**。
  - 采用滤波器（卷积核）的术语，则更能够清晰地描述这一机制。**滤波器是由一组待学习的权重所组成**，采用该滤波器对图像进行卷积滤波（局部区域内线性加权求和），并将结果输出至隐含层对应的结点。
  - **每个滤波器可以看成是学习一种特征**。采用多个滤波器，则可以理解为学习多个特征。

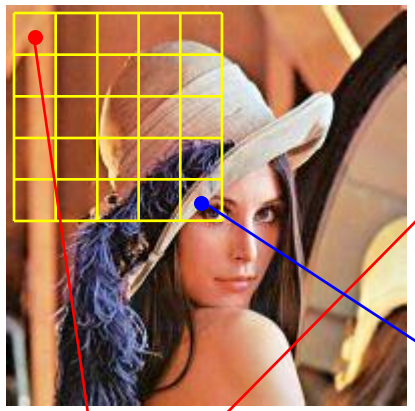
- 局部连接
  - 考虑5×5大小的窗口



结点可排列
成图像（卷积）

回忆图像滤波操作：

Image * filter

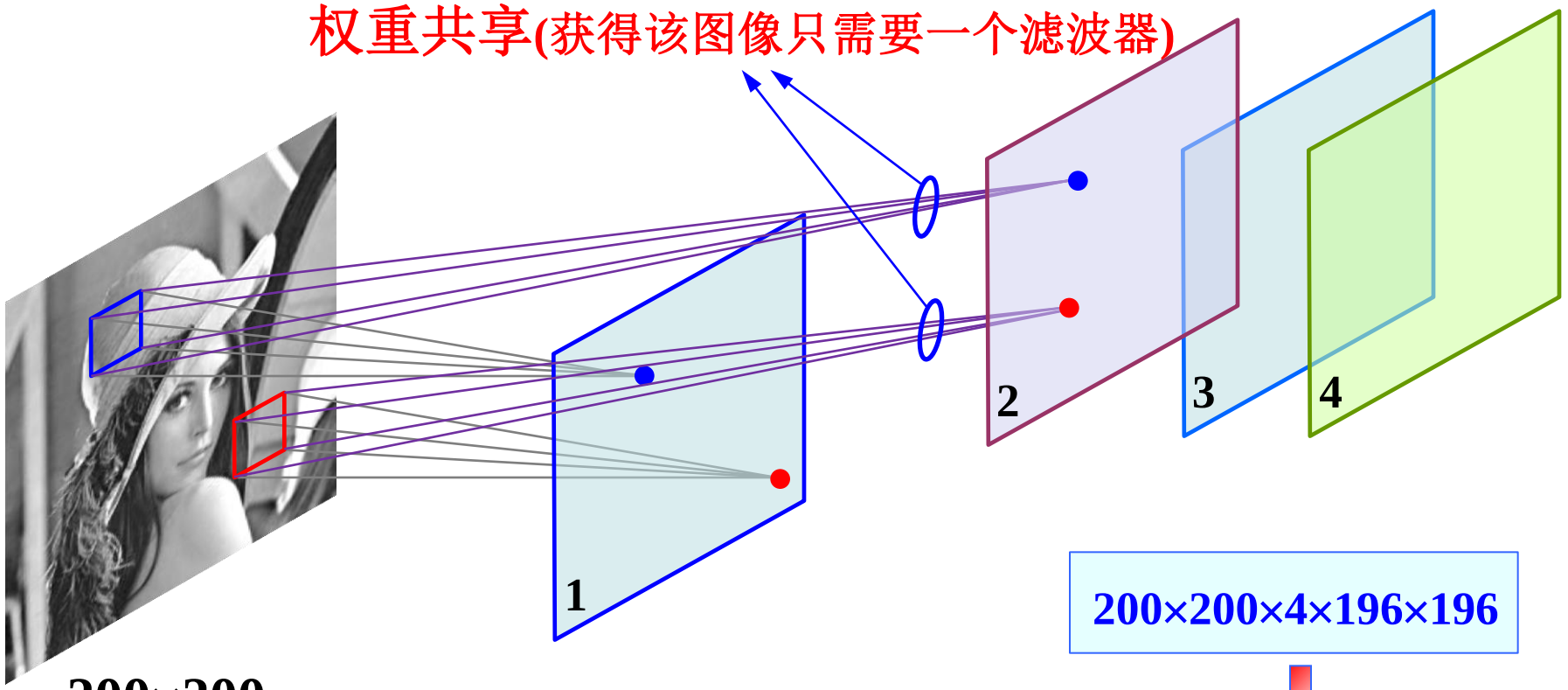


$$y = p_1 w_1 + p_2 w_2 + \cdots + p_{25} w_{25}$$

（Just 加权求和!）

4.2 卷积神经网络

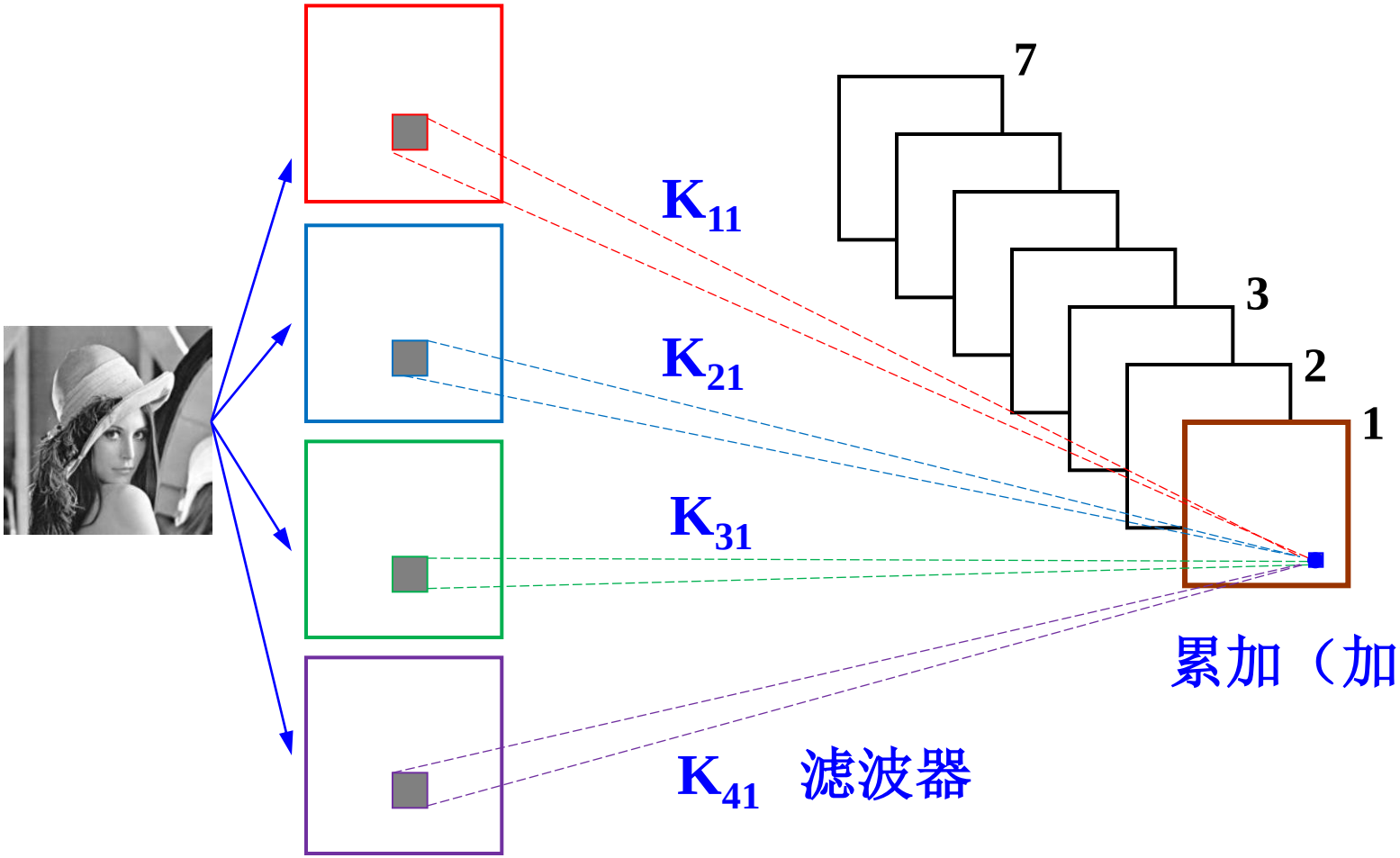
- 可以有多个滤波器：比如，4个



4.2 卷积神经网络

- 第一个隐含层
  - **考虑学习4个滤波器的情形**
    - 4个滤波器通过卷积滤波（就是局部加权求和）获得的结果，每个结果图像的大小为196×196
    - 每个滤波器对应一种特征提取。
    - 如果采用这些特征设计分类器，原始数据空间的维数将达到4× 196×196。实际应用中滤波器更多，可能导致一个高维问题。采用高维数据设计分类器，容易产生过拟合（过学习）情形。
    - 一个自然的想法就是对不同位置的特征进行聚合统计，比如，计算图像一个区域上的某个特定特征的平均值(或最大值)

- 第二个隐含层的设计：假定第二个输出7个图像，采用3×3大小的滤波器。

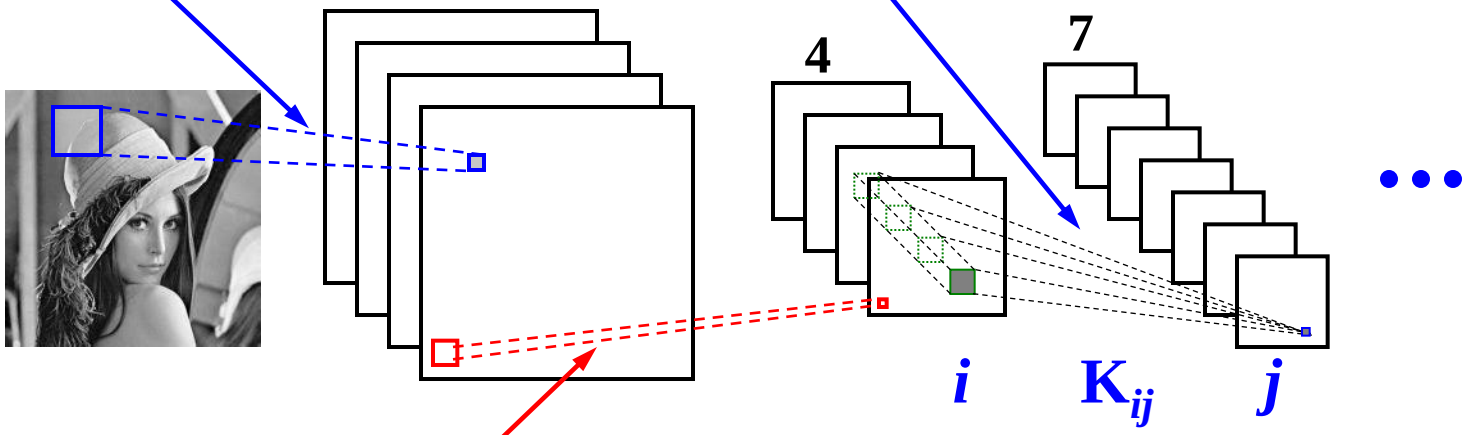


注：仍然只考虑局部连接

- 第二个隐含层的设计：假定第二个输出7个图像，采用3×3大小的滤波器

4个滤波器，均作用于输入图像

28个滤波器：K_{ij} 作用于上一层第 *i* 个图像，结果累加至下一层第 *j* 个图像

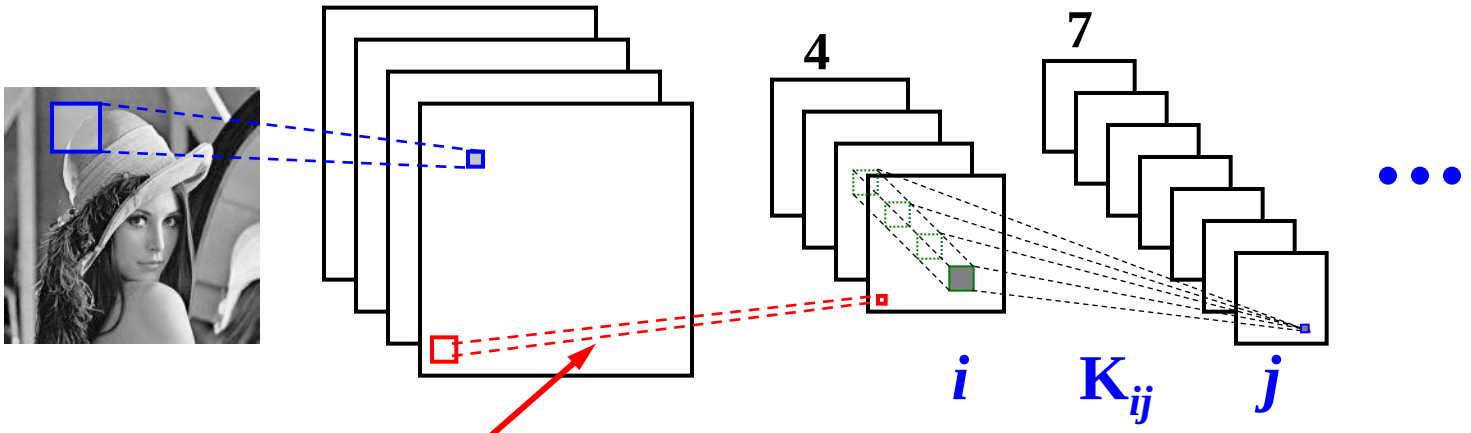


Max pooling: 用于同一卷积结果的(2x2)局部

$$y_j^L = f\left(\sum_i \left(y_i^{L-1} * K_{ij}\right) + w_j\right)$$

激励！　加权和　阈值

- 第一隐层至第二隐层：由于第一隐层包含4个图像，第二隐层包含7个图像，因此需要28个滤波器。如果第一隐层只包含1个图像，则只需要7个滤波器。

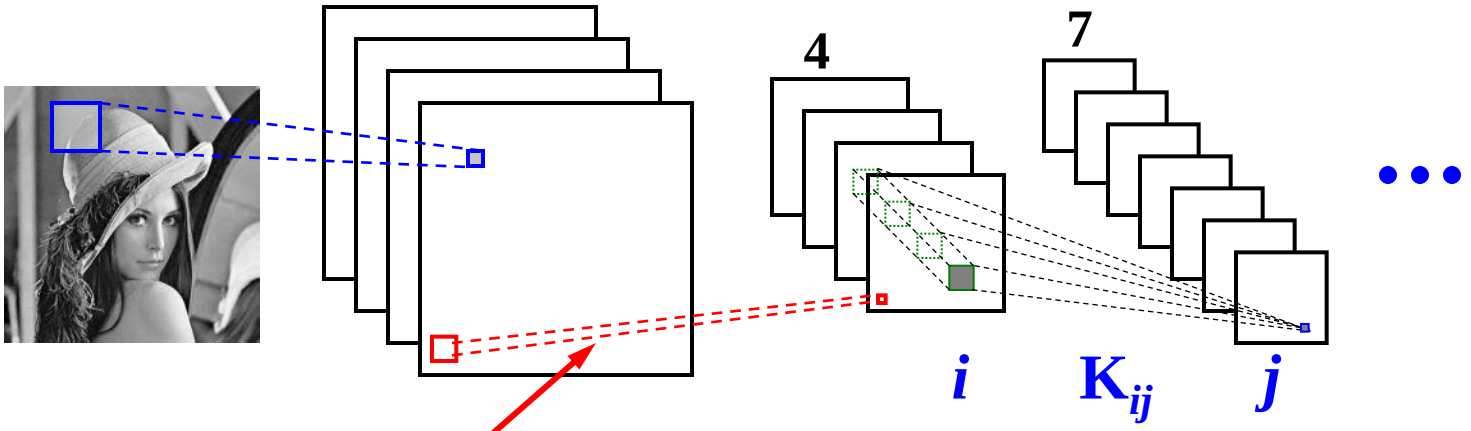


Max pooling: 用于同一卷积结果的(2x2)局部

$$y_j^L = f\left(\sum_i \left(y_i^{L-1} * K_{ij}\right) + w_j\right)$$

激励！　加权和　阈值

- 反过来考虑输入层至第一隐层：由于只有一个图像，则不需要累加。但第一隐含层包含4个图像（每个图像可以视为一个大的结点集组），因此需要4个滤波器。

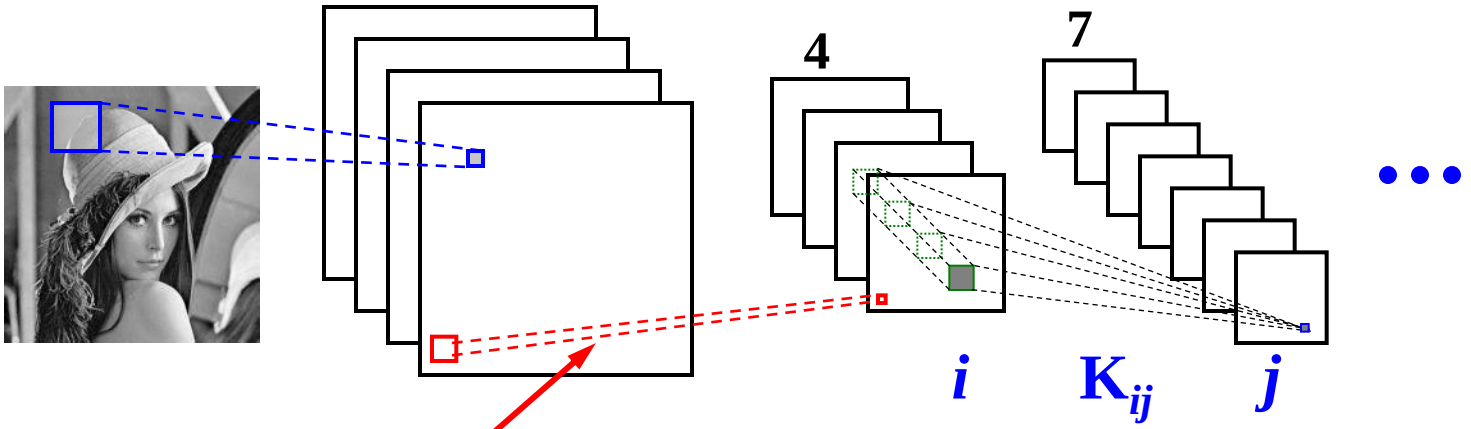


Max pooling: 用于同一卷积结果的(2x2)局部

$$y_j^L = f\left(\sum_i \left(y_i^{L-1} * K_{ij}\right) + w_j\right)$$

激励！　加权和　阈值

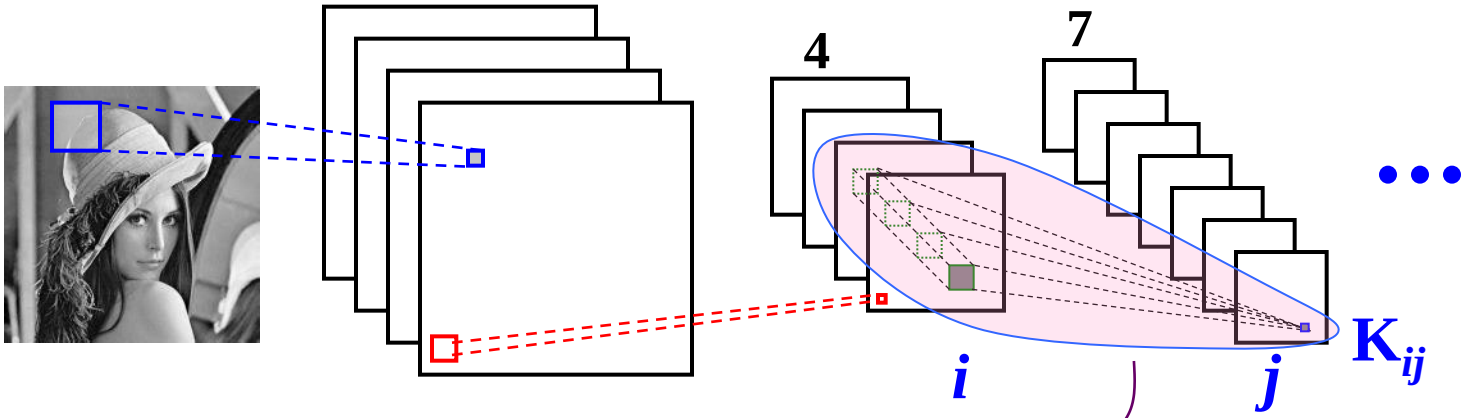
- 第一隐层至第二隐层：由于第一隐层包含4个图像，即4个大的结点集群（图像），需要将这些大的结点集群的输出值进行加权(即卷积)累加，累加到第二隐层的同一个集群结点（图像）。这正是前向神经网络的普遍操作特点。



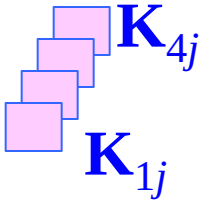
$$y_j^L = f\left(\sum_i \left(y_i^{L-1} * K_{ij}\right) + w_j\right)$$

激励！　加权和　阈值

- 第一隐层至第二隐层：如果将4个卷积结果排列成一个立体，这种累加操作也可以直接在立体数据上进行。



$$y_j^L = f\left(\sum_i \left(y_i^{L-1} * K_{ij}\right) + w_j\right)$$

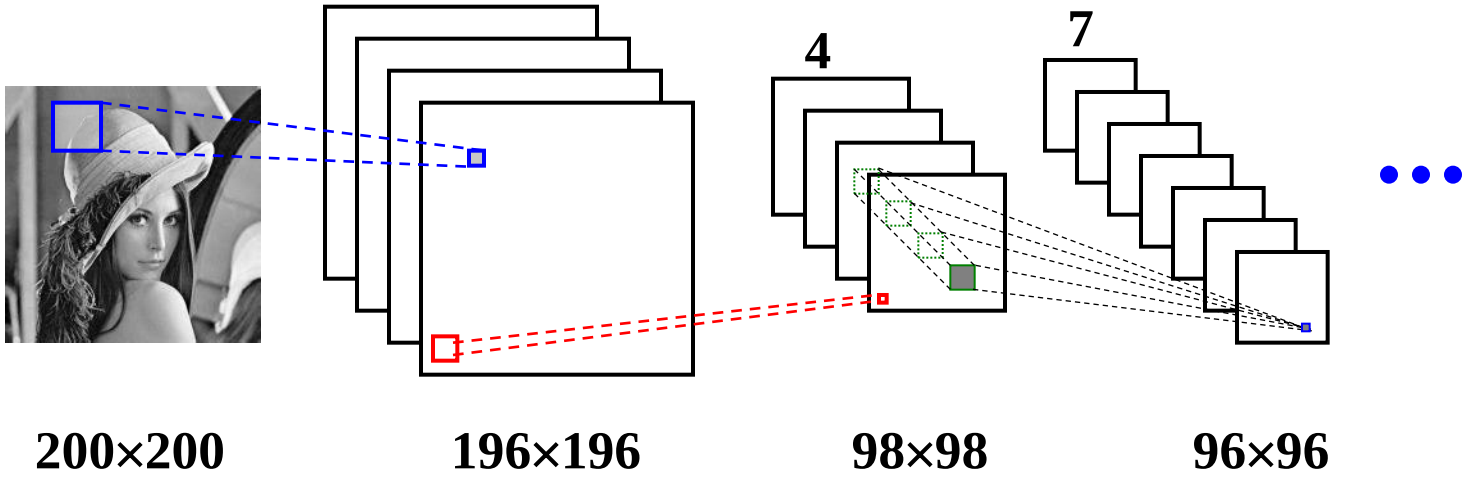


也可以将4个图像看成一个立方体，称为volume，待学习的滤波器为一个三维滤波器。三维滤波器对立体数据进行卷积操作。此时只需要学习7个三维滤波器。

- 网络结构描述（举例）
  - 图像大小：200×200
  - 第一隐含层滤波器（卷积核）大小：5×5
  - 第一隐含层结点集群（图像）个数：4
  - 第一隐含层图像大小：196 ×196
  - 第一隐含层pooling窗口大小：2×2
  - 第一隐含层pooling之后图像大小：98×98
  - 第二层滤波器（卷积核）大小：3×3
  - 第二隐含层结点集群(图像）个数：7 ← (也可以理解为7个3×3 ×4的三维滤波器)
  - 第二隐含层滤波器总数：28
  - 第二隐含层图像大小：96 ×96
  - 第二隐含层pooling窗口大小：2×2
  - 第二隐含层pooling之后图像大小：48×48

4.2 卷积神经网络

- 第一隐含层至第二隐含层权重数：
  - 若采用全连接： 输入×输出 ＝ (4×98×98) × (4×96×96)
  - 若采用局部连接+权值共享（3×3滤波器）：$9 \times 4 \times 7$

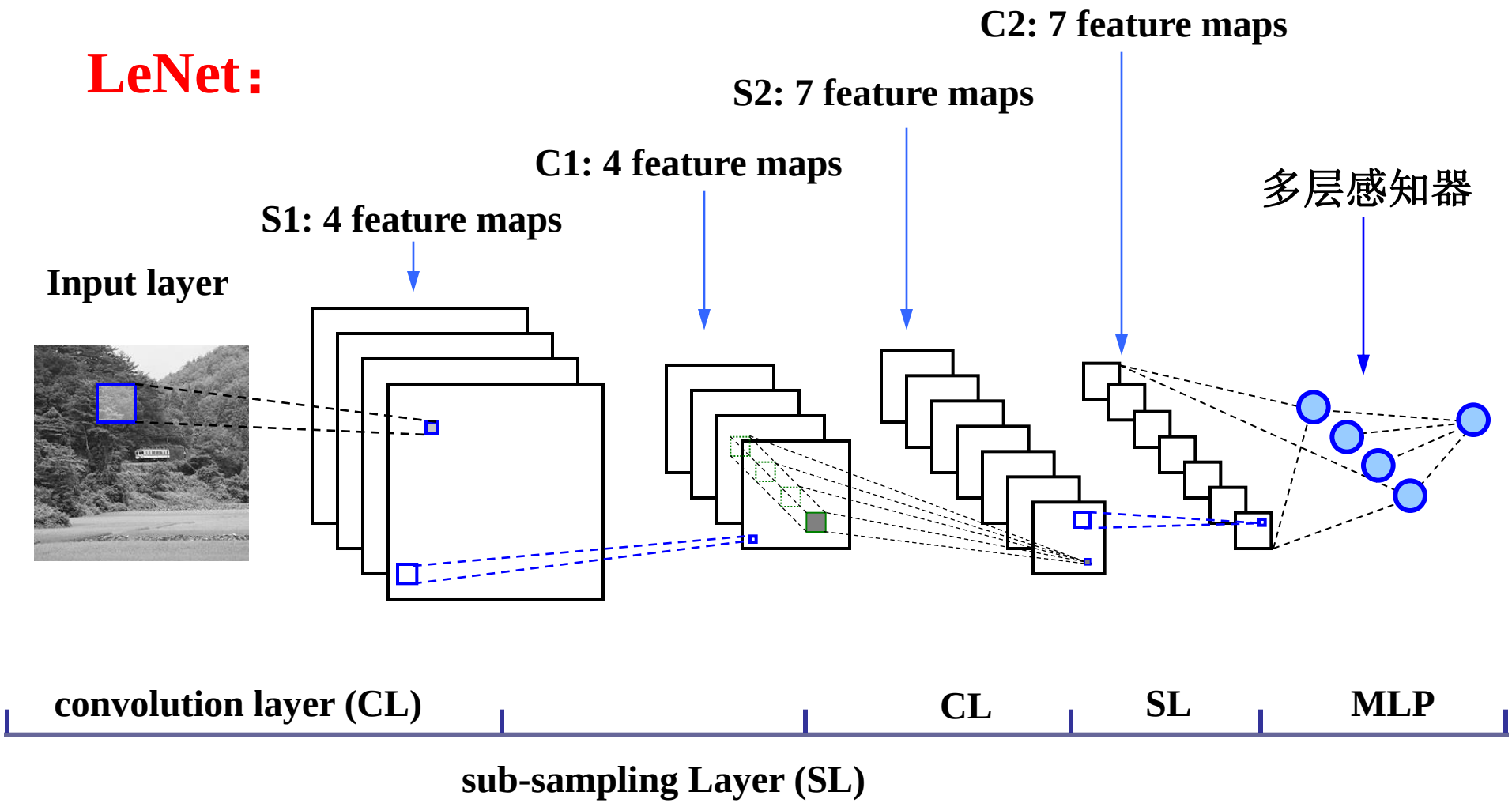


4.2 卷积神经网络

- 层与层之间的基本操作
  - 卷积 (Convolution)
    - 将卷积之和加到下一层
    - 对卷积之和进行激励 **（特别指出：也可以在pooling之后求激励）**
  - 聚合 (Pooling)
    - 下采样，两种基本的运算：
      - 2×2窗口取平均
      - 2×2窗口取最大值

4.2 卷积神经网络

LeNet：



4.2 卷积神经网络

- 网络训练
  - **采用反向传播算法！**
  - 选择一个样本 **x**，信息从输入层经过逐级的变换，传送到输出层；
  - 计算该样本的实际输出 **o** 与相应的理想输出 **t** 的差；
  - 按极小化误差反向传播方法调整权矩阵；
  - 可以采取小的样本组，逐步进行训练。
- 思考题：
  - 卷积神经网络中有一个pooling 操作，因为这一点需要对现有BP算法做一些修改，请问如何改？

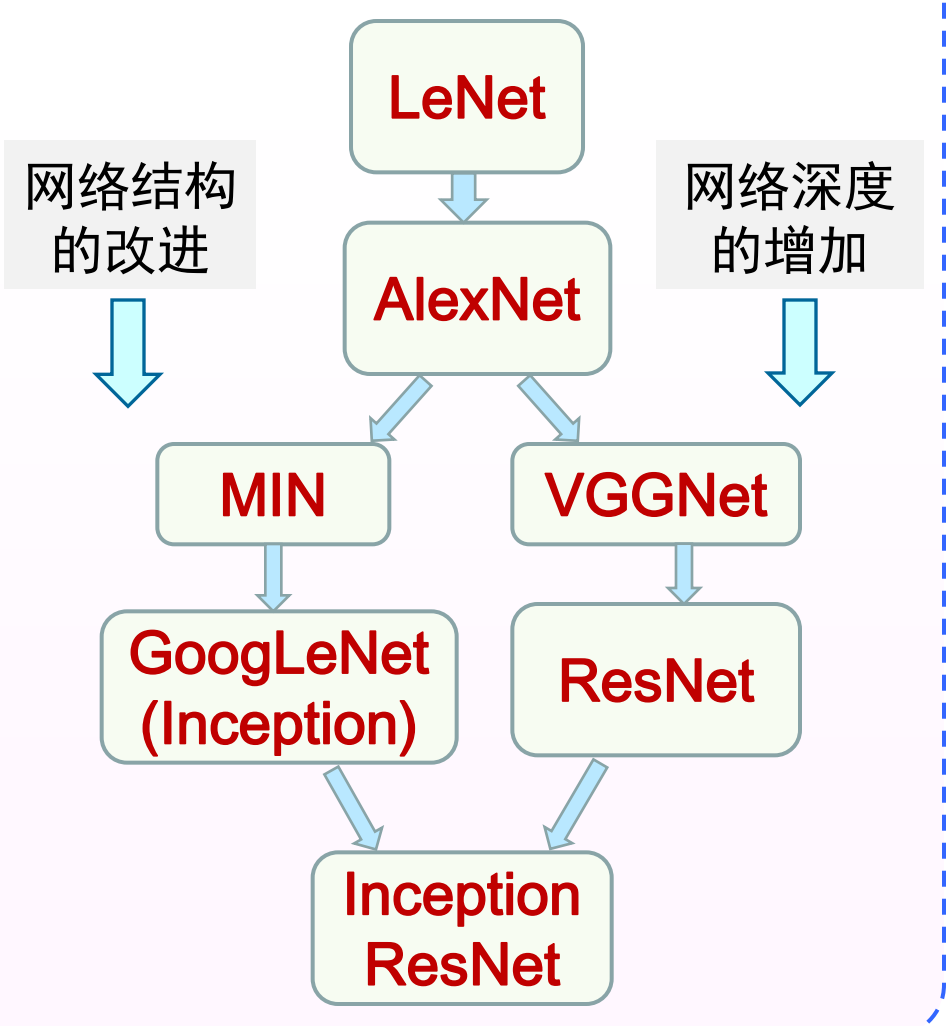
4.2 卷积神经网络

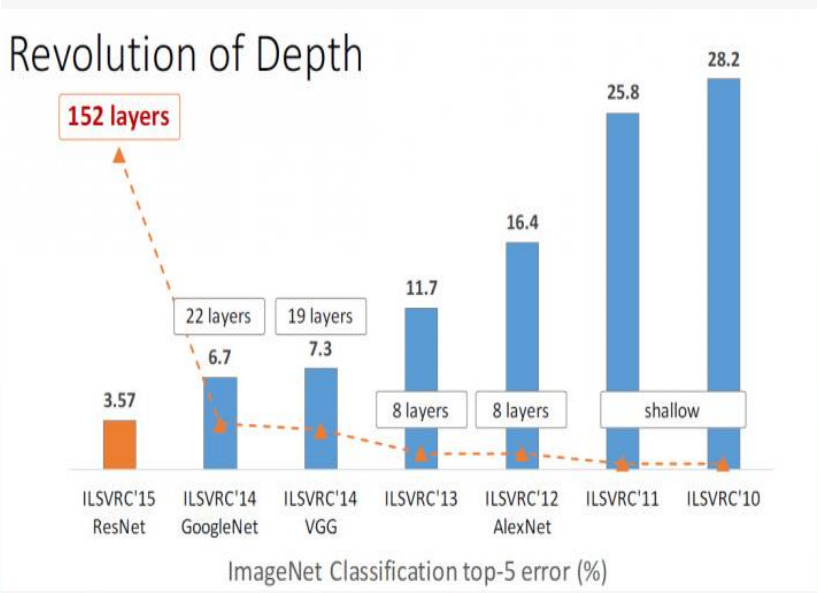
- 卷积
  - 局部特征提取
  - 训练中进行参数学习
  - 每个卷集核提取特定模式的特征
- 池化 (聚合Pooling)
  - 降低数据维度，避免过拟合
  - 增强局部感受野，因而更容易获取到抽象的特征
  - 提高平移不变性
- 全连接
  - 特征提取到分类的桥梁

4.2 卷积神经网络

- 核心思想（总结）
  - 我们之所以决定使用卷积后的特征是因为图像具有一种“静态性(stationarity)”的属性，这也就意味着在一个图像区域有用的特征极有可能在另一个区域同样适用。
  - 局部感受野、权值共享（或者权值复制）以及时间或空间下采样这三种结构化思想结合起来获得了某种程度的位移、尺度、形变不变性。
  - CNN的强大之处在于多层结构能自动学习特征，且可学习到多个层次的特征：较浅的卷积层感知域较小，学习到一些局部区域的特征；较深的卷积层具有较大的感知域，能够学习到更加抽象一些的特征。这些抽象特征对物体的大小、位置和方向等敏感性更低，从而有助于识别性能的提高。

- 深度卷积神经网络发展图





ILSVRC图像分类竞赛结果

4.3 全卷积神经网络

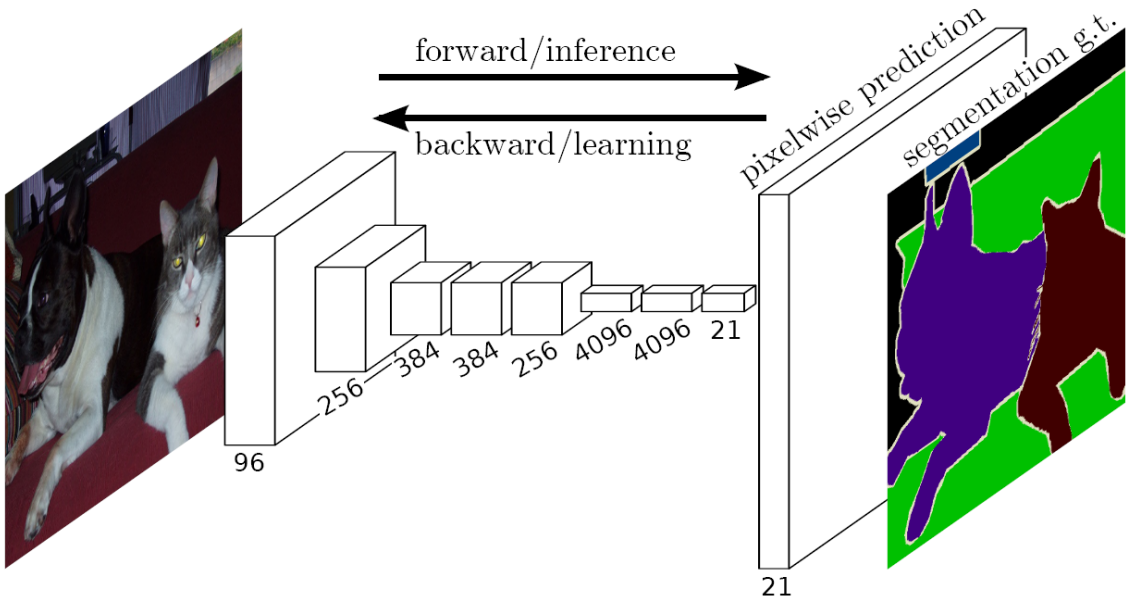
4.3 全卷积神经网络

- Image segmentation is very challenging task in computer vision.

Segment image into different regions such that each region has the same semantic meaning.

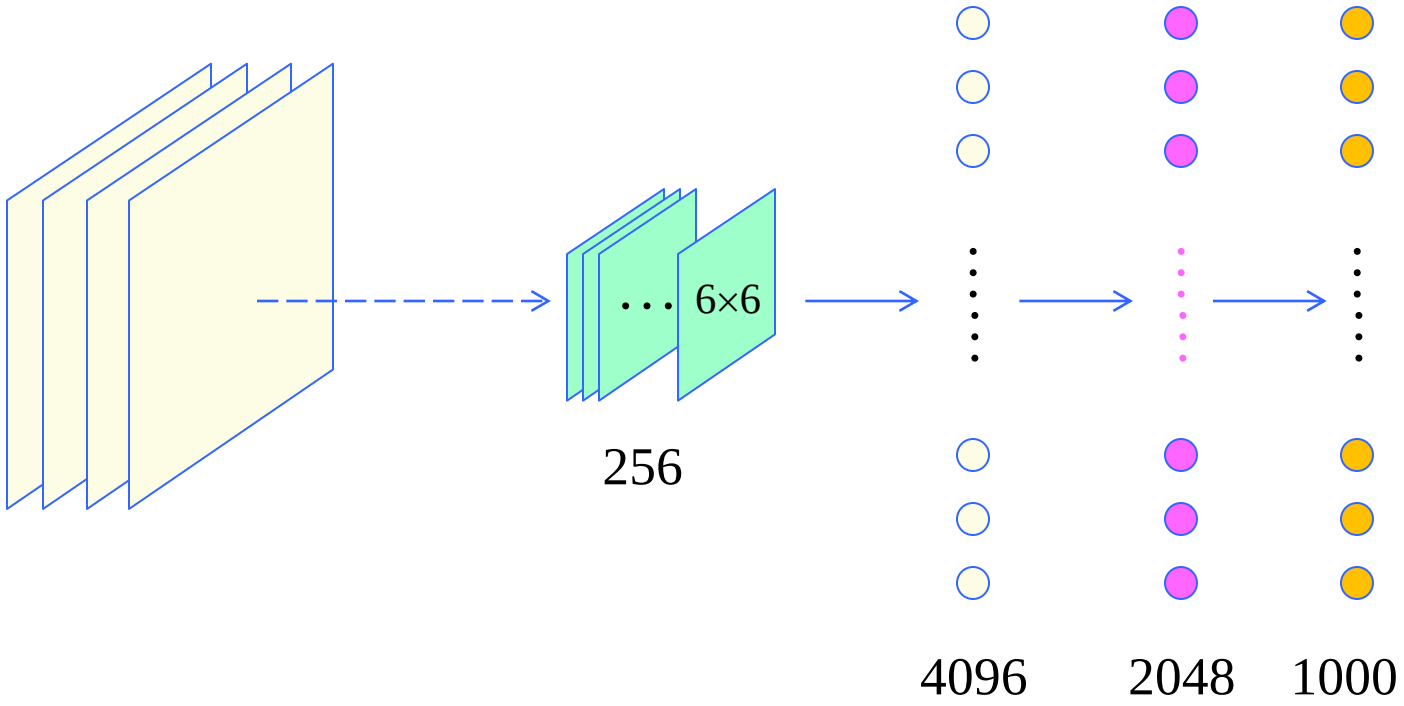
4.3 全卷积神经网络

- 如何采用CNN做图像分割
  - J. Long, E. Shelhamer and T. Darrell, Fully Convolutional networks for Semantic Segmentation, CVPR, 2015



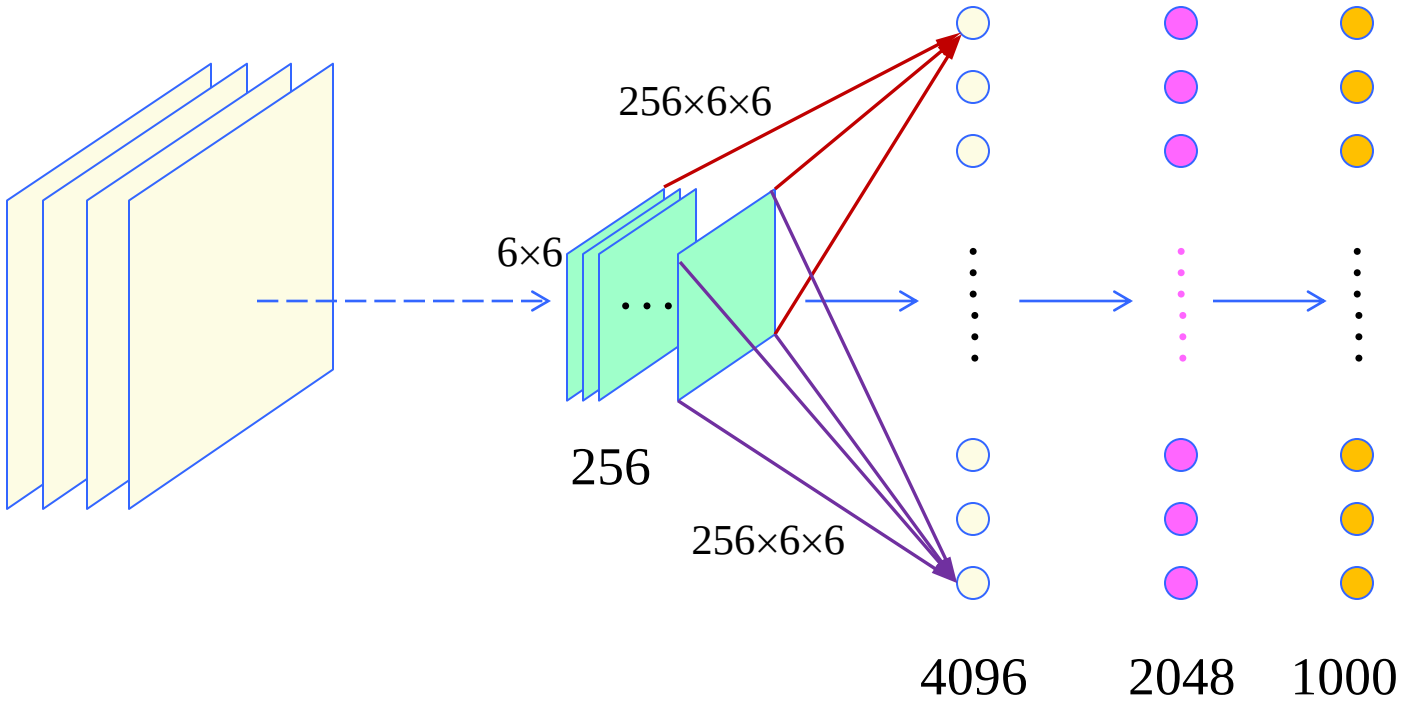
4.3 全卷积神经网络

- 如何从卷积操作的角度理解全连接层



4.3 全卷积神经网络

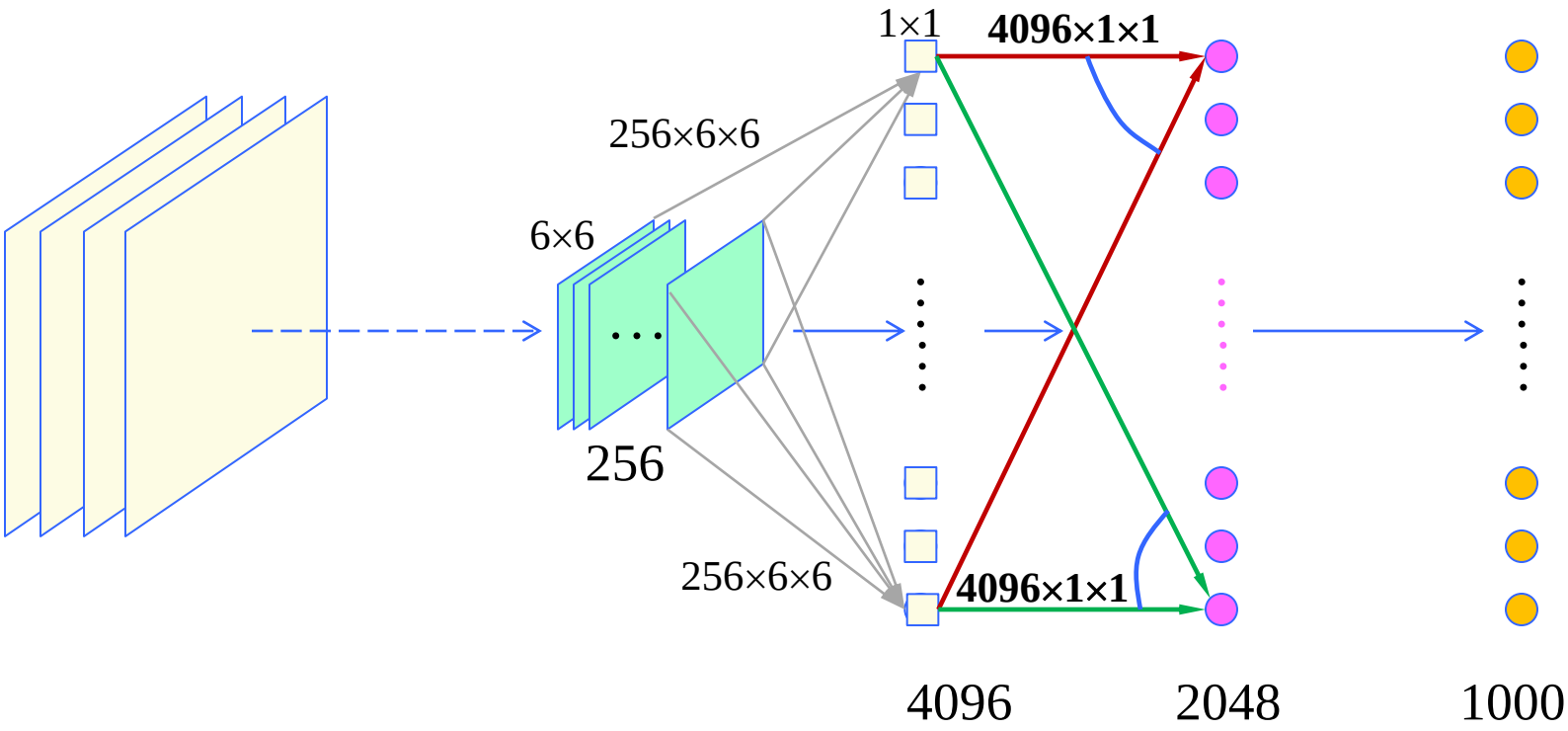
- 如何从卷积操作的角度理解全连接层



4096个滤波器，每一个滤波器的大小为：256×6×6

4.3 全卷积神经网络

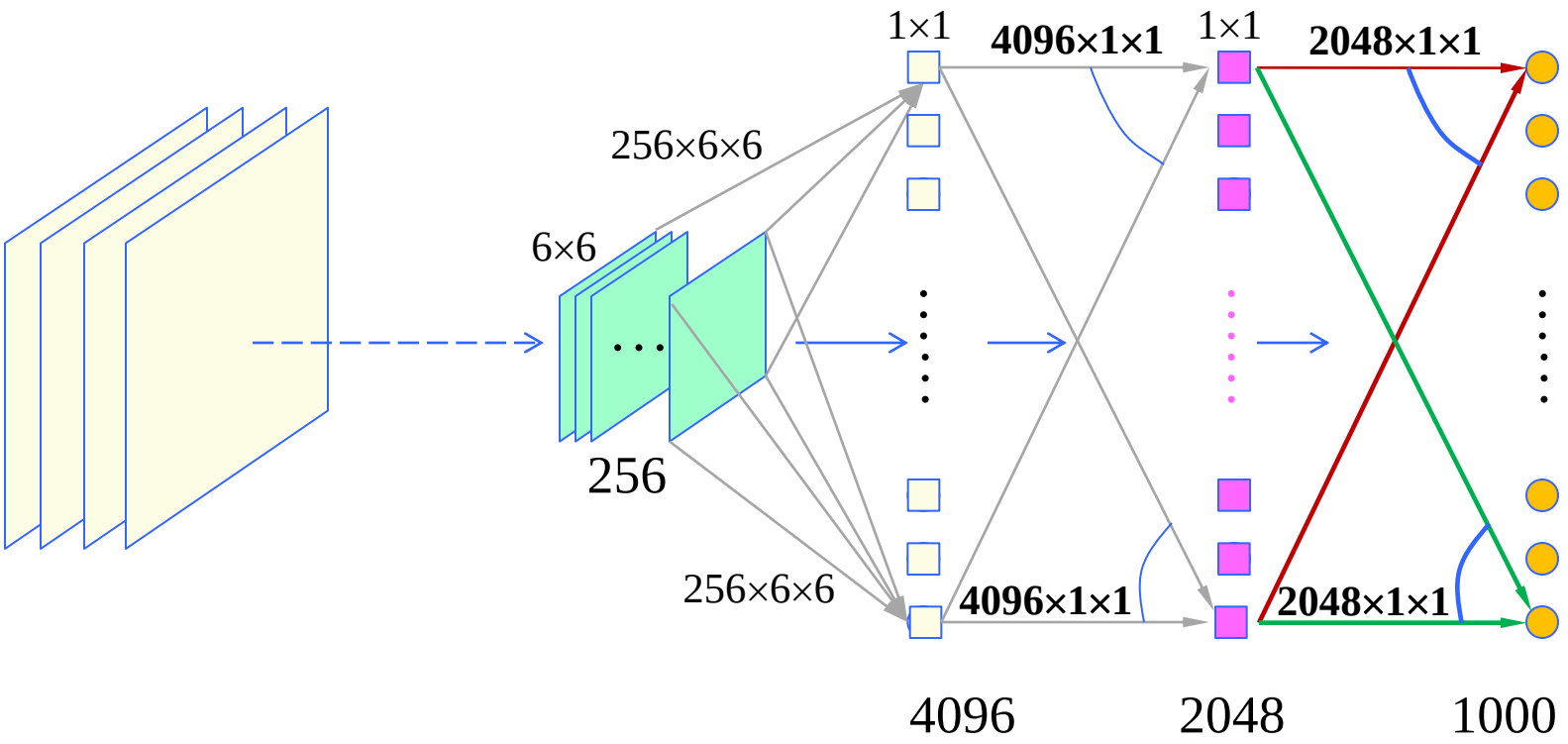
- 如何从卷积操作的角度理解全连接层



2048个滤波器，每一个滤波器的大小为：4096×1×1

4.3 全卷积神经网络

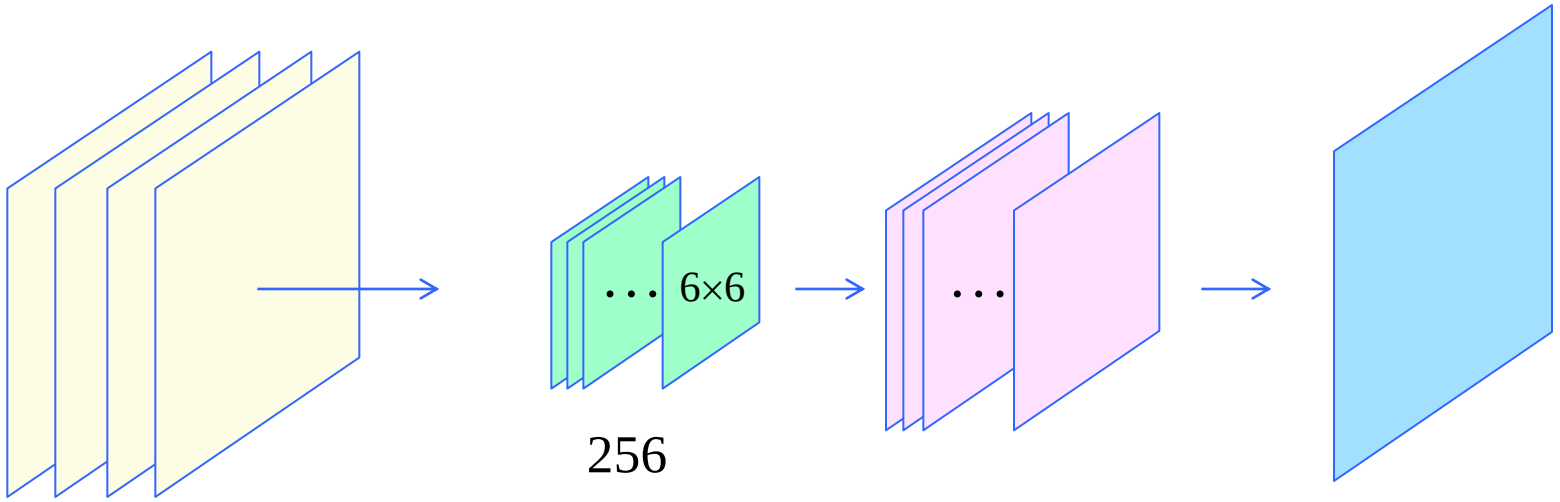
- 如何从卷积操作的角度理解全连接层



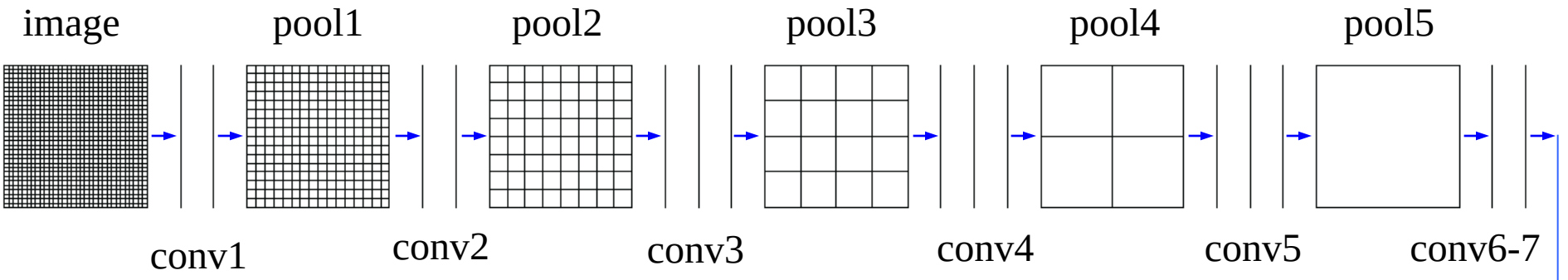
1000个滤波器，每一个滤波器的大小为：2048×1×1

4.3 全卷积神经网络

- 从图像分割的角度：输出是一幅图像

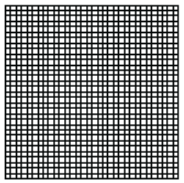


Fully Convolutional networks for Semantic Segmentation：
CVPR, 2015, Long et al..



pool1：相对于原图，2 倍下采样
pool2：相对于原图，4 倍下采样
pool3：相对于原图，8 倍下采样
pool4：相对于原图，16倍下采样
pool5：相对于原图，32倍下采样

32x upsampled

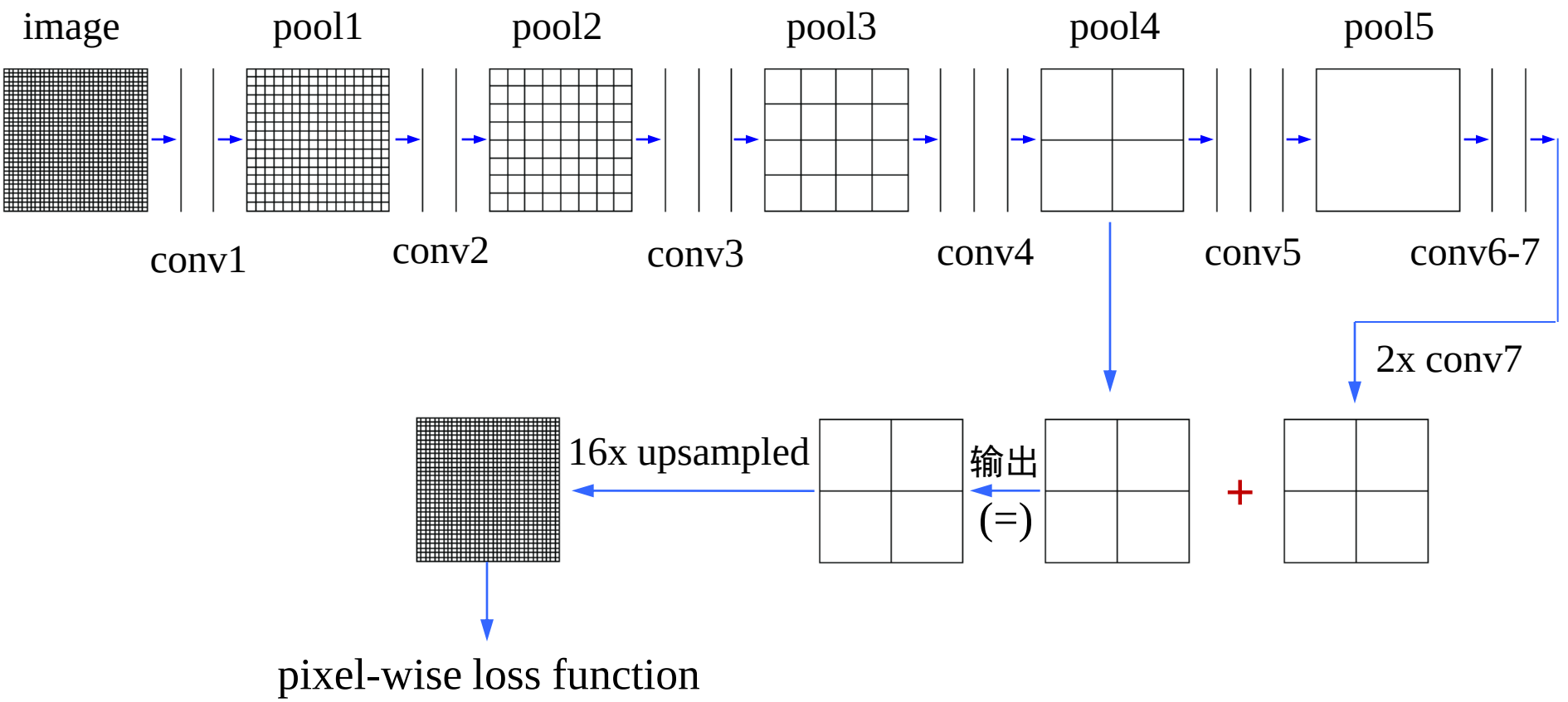


pixel-wise loss fuction

最后，将下采样32倍的结果，通过双线性插值上采样32倍，获得与原图等同大小的图像，并构造基于单位像素损失的目标函数。

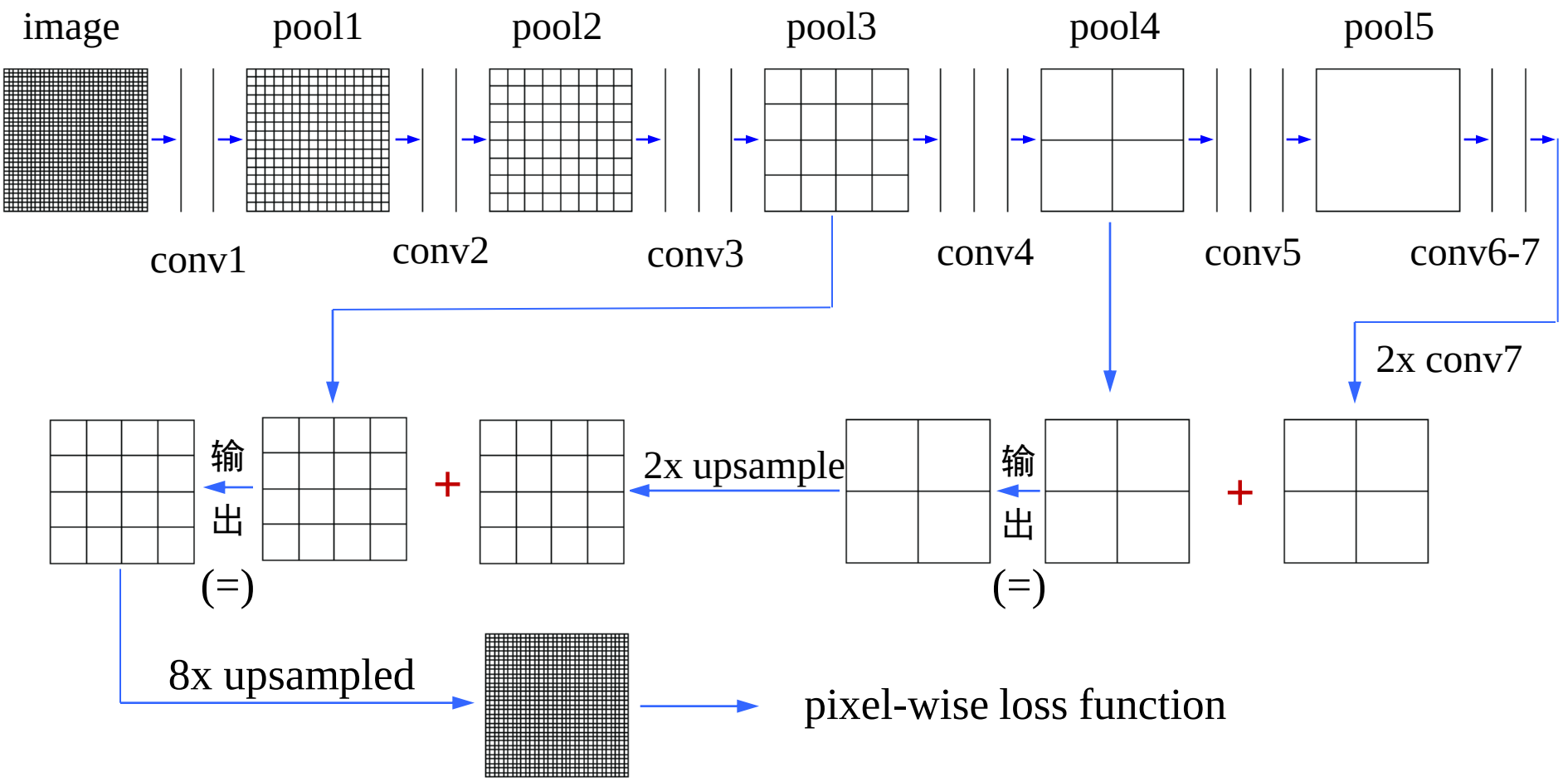
此网络称为FCN-32s

Fully Convolutional networks for Semantic Segmentation：CVPR, 2015, Long et al..



此网络称为FCN-16s

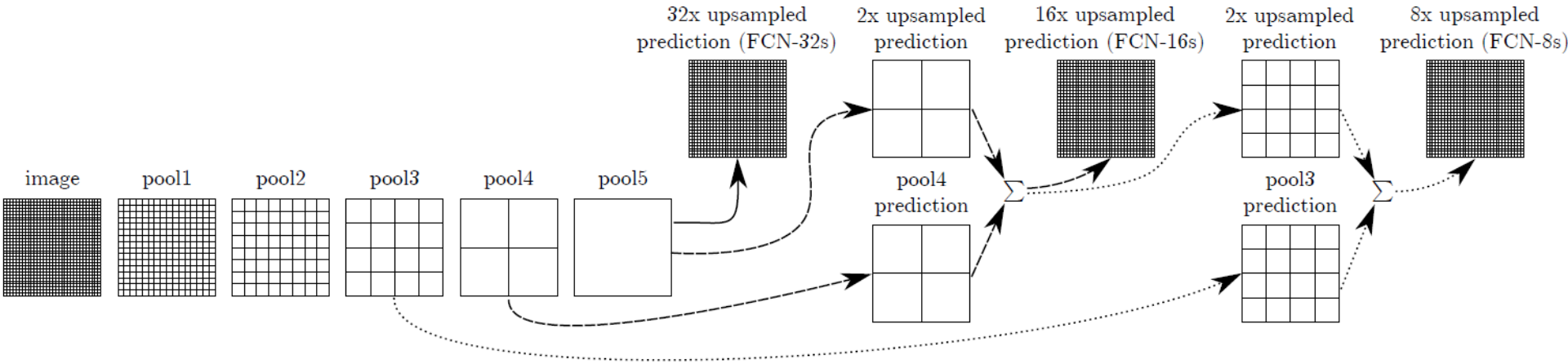
Fully Convolutional networks for Semantic Segmentation：CVPR, 2015, Long et al..



此网络称为FCN-8s

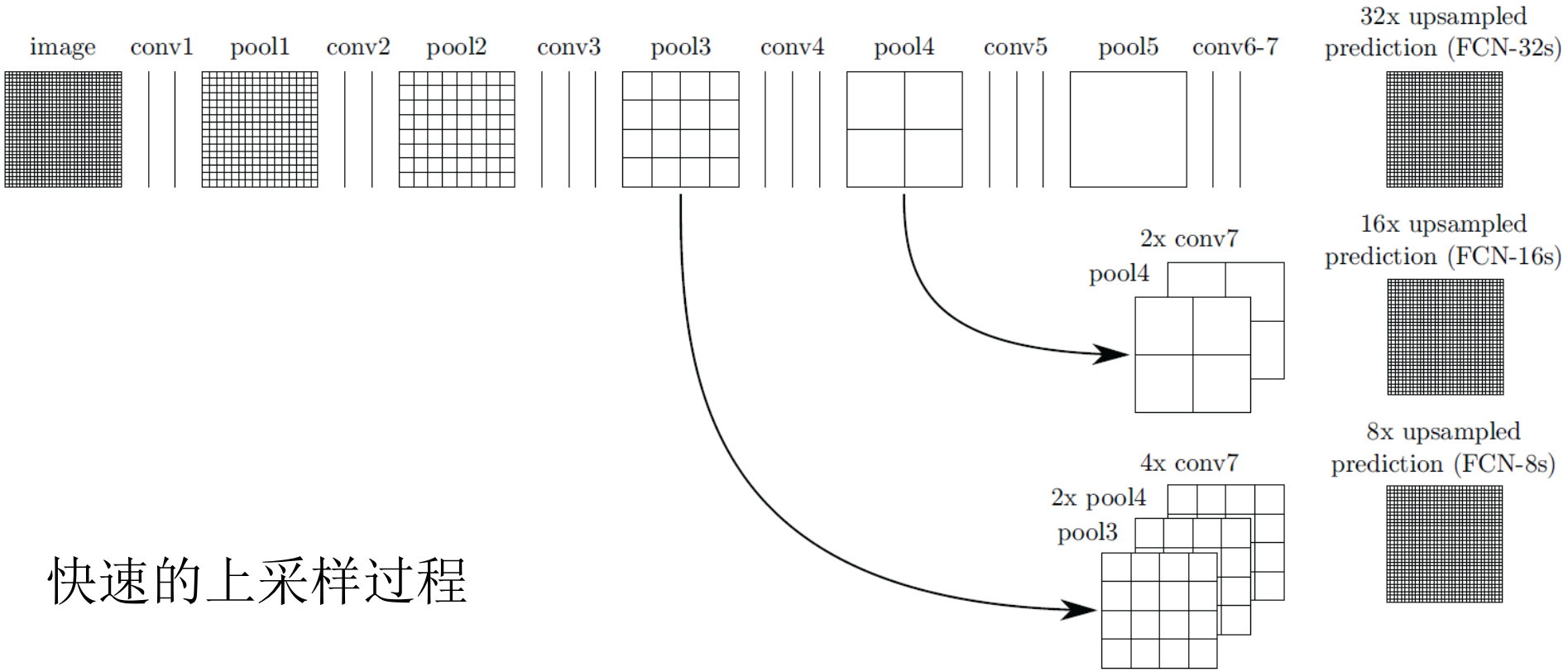
Fully Convolutional networks for Semantic Segmentation：
CVPR, 2015, Long et al..

综合起来看：



注意: 这里得到了三个不同的网络：FCN-32s, FCN-16s, FCn-8s

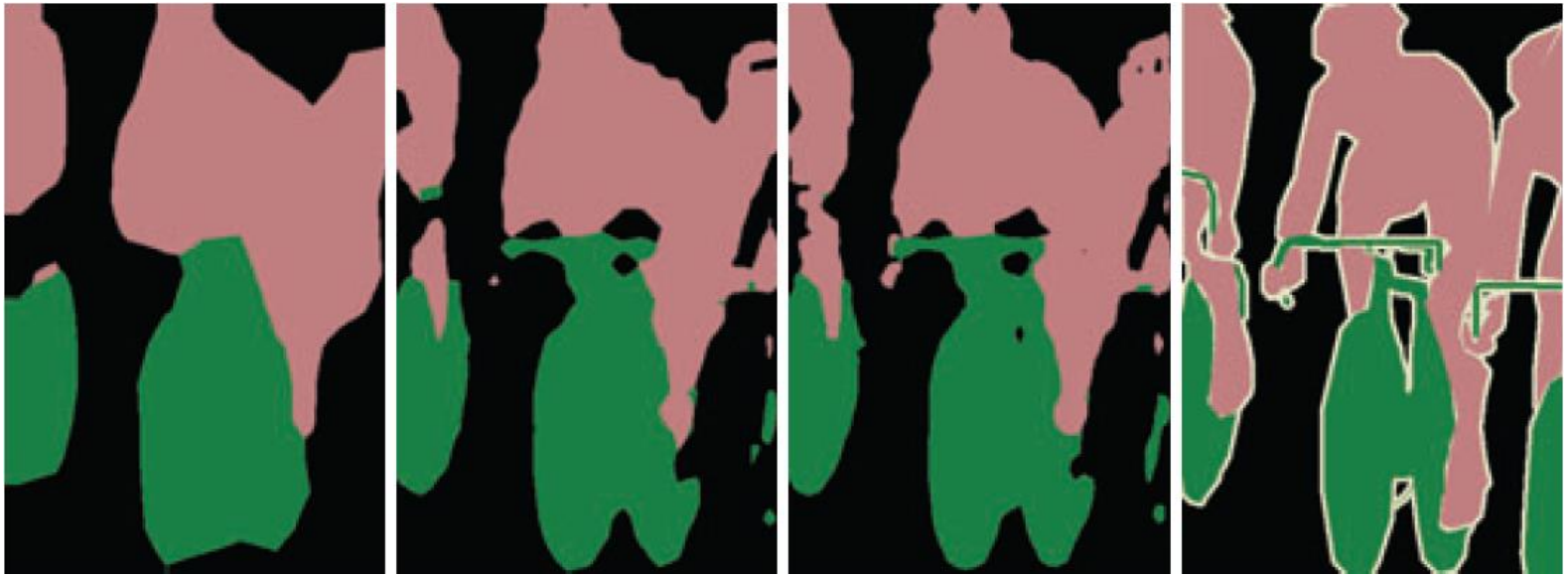
Fully Convolutional networks for Semantic Segmentation：CVPR, 2015, Long et al..



快速的上采样过程

略微有些不同（每个网络一次到位，**不是逐步上采样2倍**）

Fully Convolutional networks for Semantic Segmentation：
CVPR, 2015, Long et al..

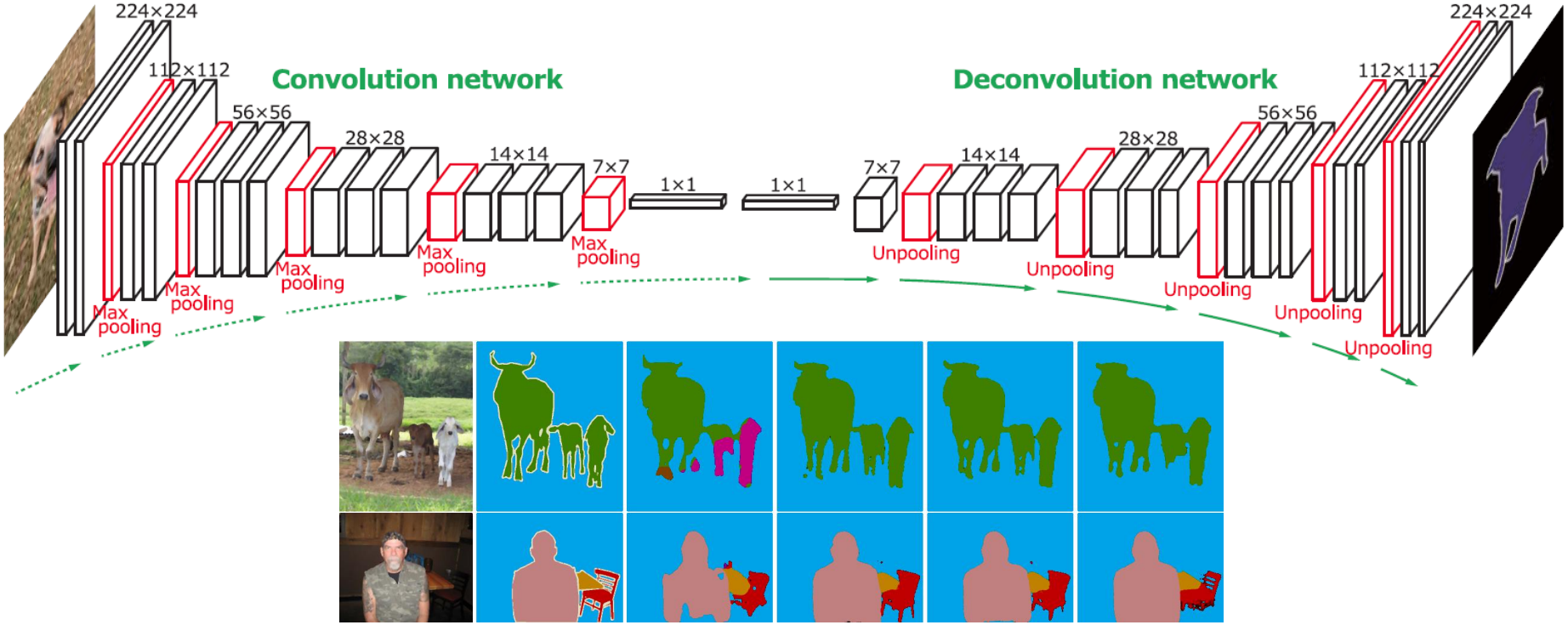


FCN-32s **FCN-16s** **FCN-8s** **Ground Truth**

4.4 反卷积神经网络

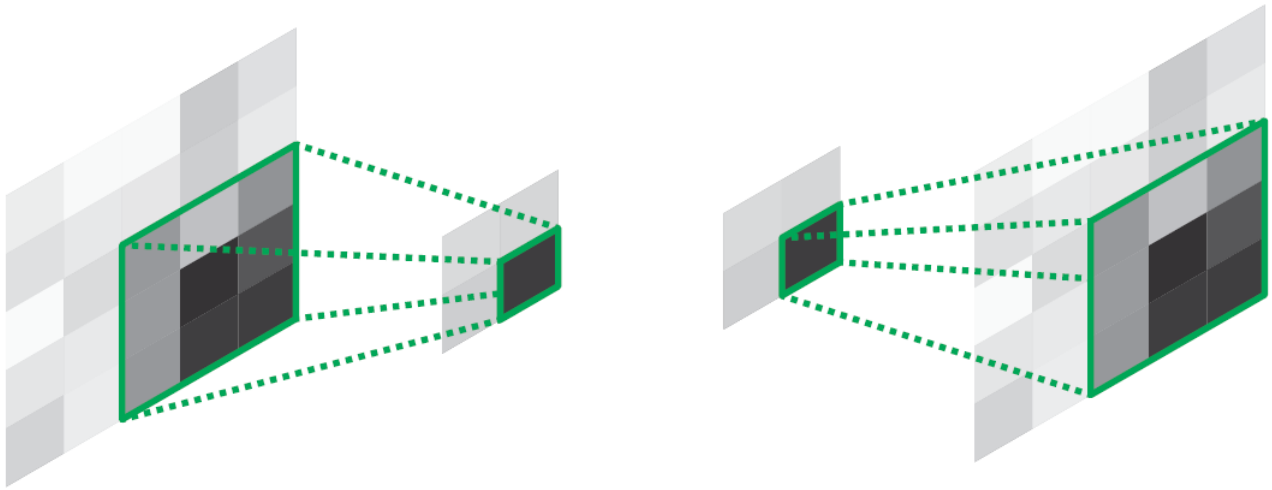
4.4 反卷积神经网络

- 从图像分割的角度：输出是一幅图像
  - **Deconvolutional NN：** Hyeonwoo Noh, Seunghoon Hong, Bohyung Han. Learning Deconvolution Network for Semantic Segmentation, ICCV, 2015.



4.4 反卷积神经网络

- 上采样方式之一：Deconvolution

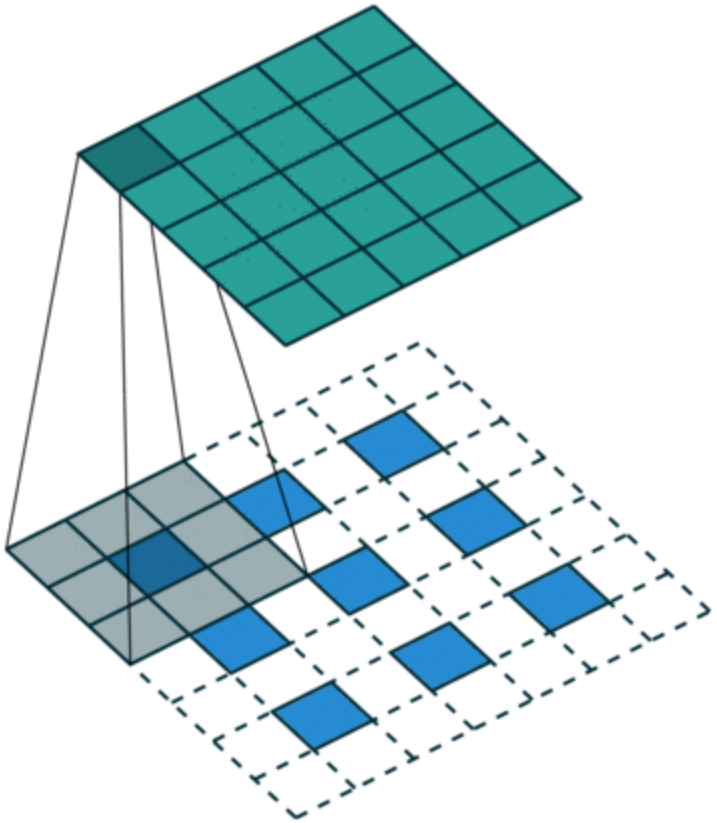


Convolution　　Deconvolution

扩大补零，然后再按传统的卷积操作方式生成上一层

4.4 反卷积神经网络

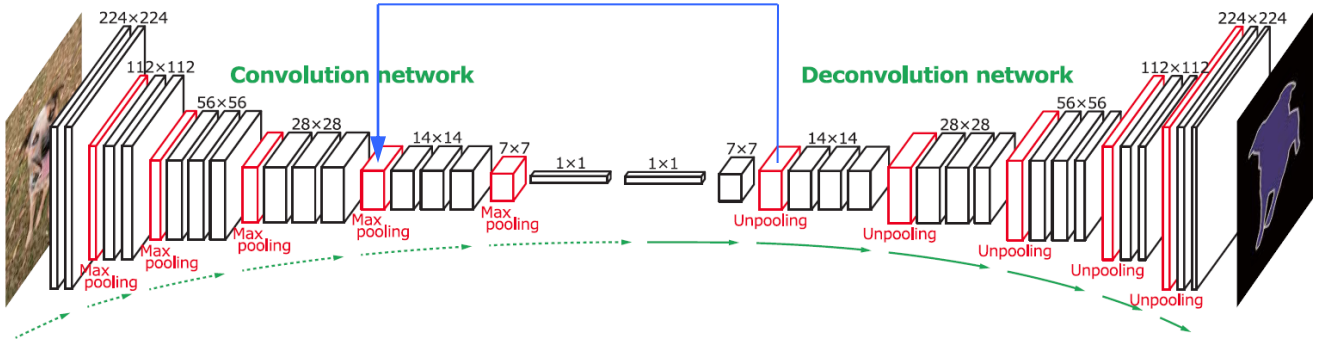
- 上采样方式之一：Deconvolution



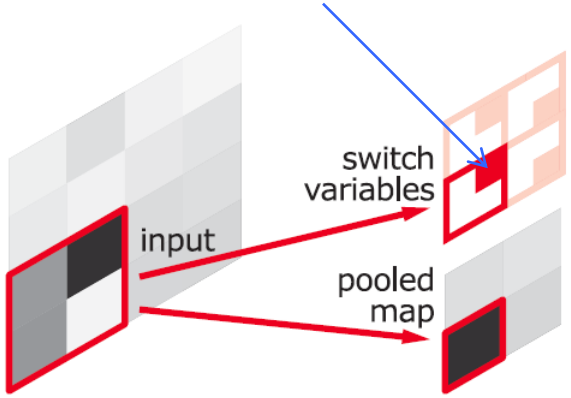
扩大补零，然后再按传统的卷积操作方式生成上一层

上采样方式之二：Unpooling

在这里去找最大值的位置

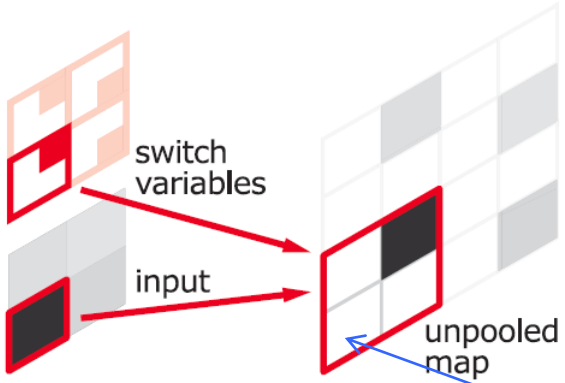


记录最大值的位置



Pooling

黑色代表最大值



Unpooling

补零

该操作相当于最近邻插值，但保持对应的空间位置。

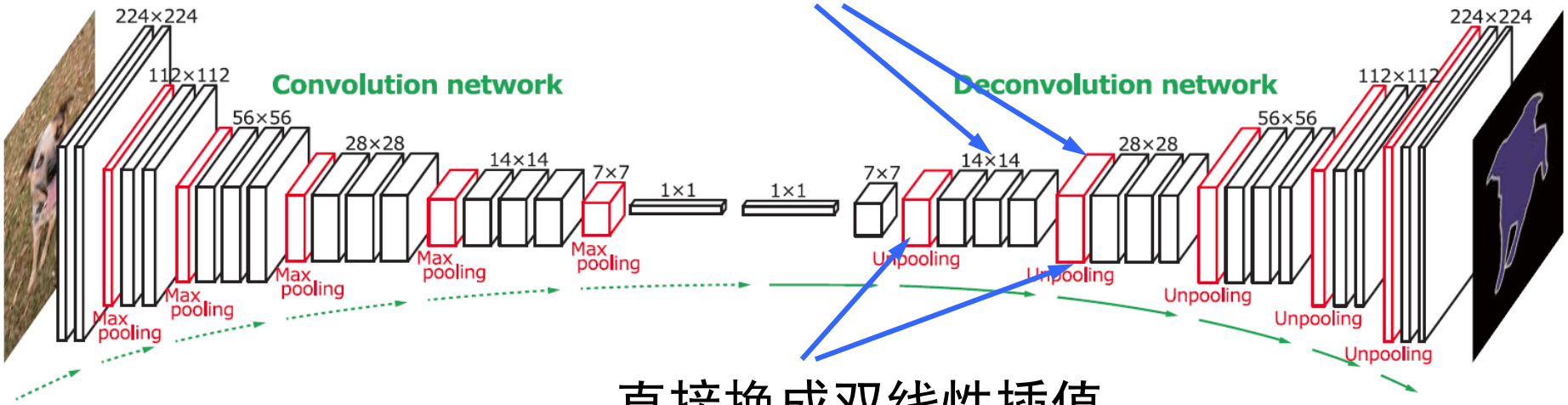
4.4 反卷积神经网络

- **Training**
  - 加入Batch Normalization：standard Gaussian distribution:
    - 作用于每层卷积的输出：对所有样本的feature mapping作零均值和一方差输出变换，很管用。
      - 在技术发展中，其所起作用点相当于“将普通的sigmoid激活函数变为ReLU激活函数”
  - 最早的反卷积网络在最高层加入：CRF
    - CRF有很多参数需要手工设置；
    - 现在：不用了，太慢了；
    - 现在：提特征的网络更强大；
    - 现在：训练数据更多了。

4.4 反卷积神经网络

- 现在还在做传统的反反卷积吗？
  - Unpooling操作不要了！反卷积也不要了
    - 也就是说，右边红色的上采样操作既不采用unpooling技术，也不采用deconvolution技术
  - 上采样部分：只用双线性插值，并且内部没有学习的参数

真正的卷积（不是在作反卷积）



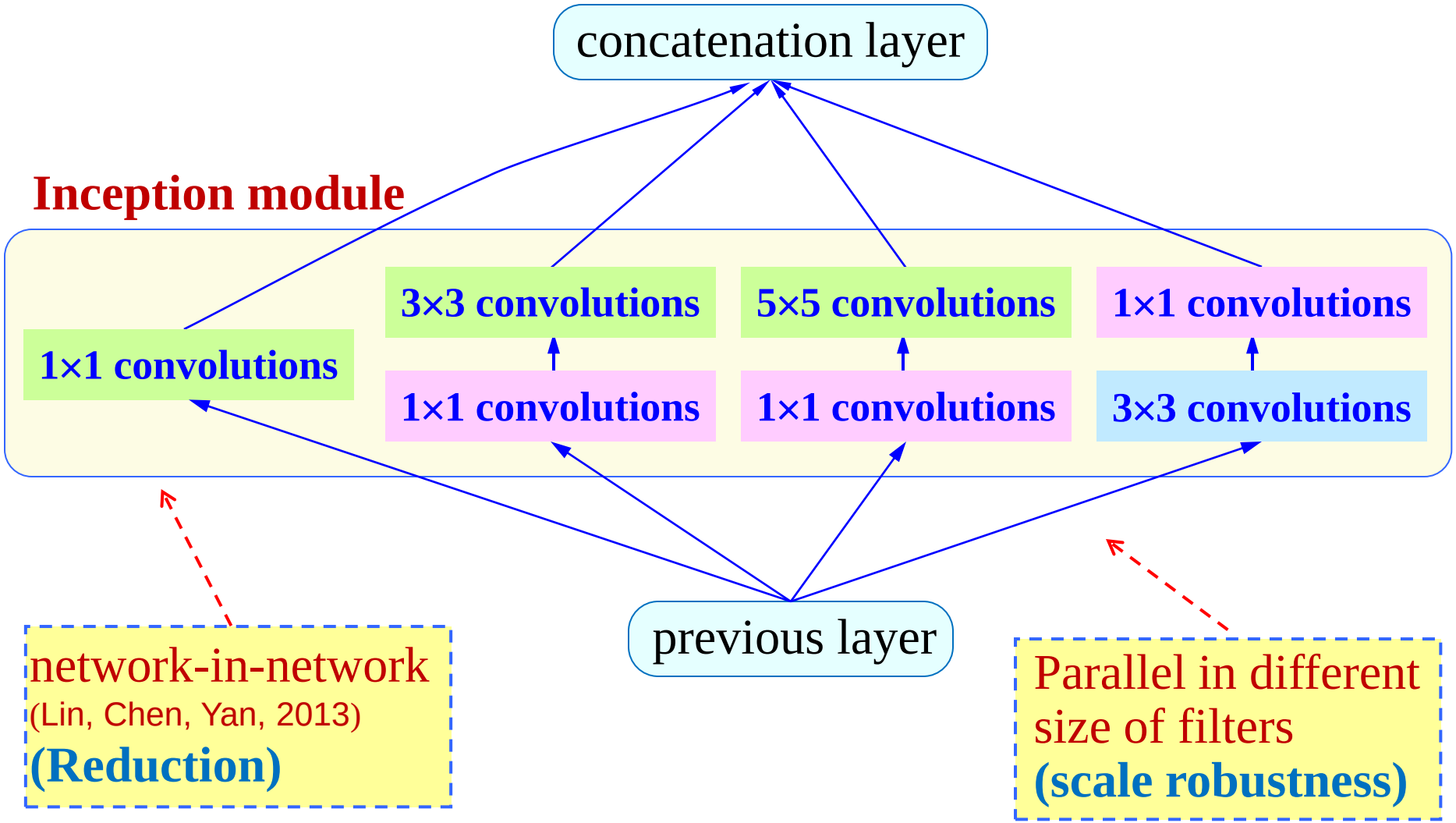
直接换成双线性插值

4.5 GoogleNet

4.5 GoogLeNet

- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, et al.. Going Deeper with Convolutions, CVPR 2015
  - 与Network in Netwok 类似，GoogLeNet采用子网络堆叠的方式搭建，每个子网络为一个Inception 模块
  - Inception module 包含四个分支：
    - Shortcut连接：将前一层输入通过1×1卷积
    - 多尺度滤波：输入通过1×1卷积之后分别连接卷集核大小为3和5的卷积
    - 池化分支：相继连接3×3 pooling和1×1卷积

- **GoogLeNet**


concatenation layer
Inception module
1×1 convolutions
3×3 convolutions
5×5 convolutions
1×1 convolutions
1×1 convolutions
1×1 convolutions
3×3 convolutions
previous layer
network-in-network
(Lin, Chen, Yan, 2013)
(Reduction)
Parallel in different size of filters
(scale robustness)

4.5 GoogLeNet

- **Inception module 优点之一**
  - **减少网络参数，降低运算量**
    - 对输入和输出通道为256的3×3卷积层，需要的操作次数为：

$$256\times256\times3\times3=589824$$

    - 假设内部第一层卷积输出为32mappings，Inception模块的操作次数为：

$$256\times1\times1\times256 + 256\times1\times1\times32 + 256\times1\times1\times32 + 256\times3\times3\times32 + 32\times3\times3\times256 + 32\times5\times5\times256 + 32\times1\times1\times256 = 434176$$

因此，1×1卷积的作用之一是通过降维减少网络开销

4.5 GoogLeNet

- **Inception module 优点之二**
  - **多尺度多层次滤波**
    - 多尺度：对输入特征图像分别在3×3和5×5的卷集核上进行滤波，提高了所学特征的多样性，增强了网络对不同尺度的鲁棒性。
    - 多层次：符合Hebbian原理，即通过1×1卷积把具有高度相关性的不同通道的滤波结果进行组合，构建出合理的稀疏结构。

GoogLeNet的网络参数为AlexNet的1/12，ILSVRC 2014 top-5错误率降至6.67%。

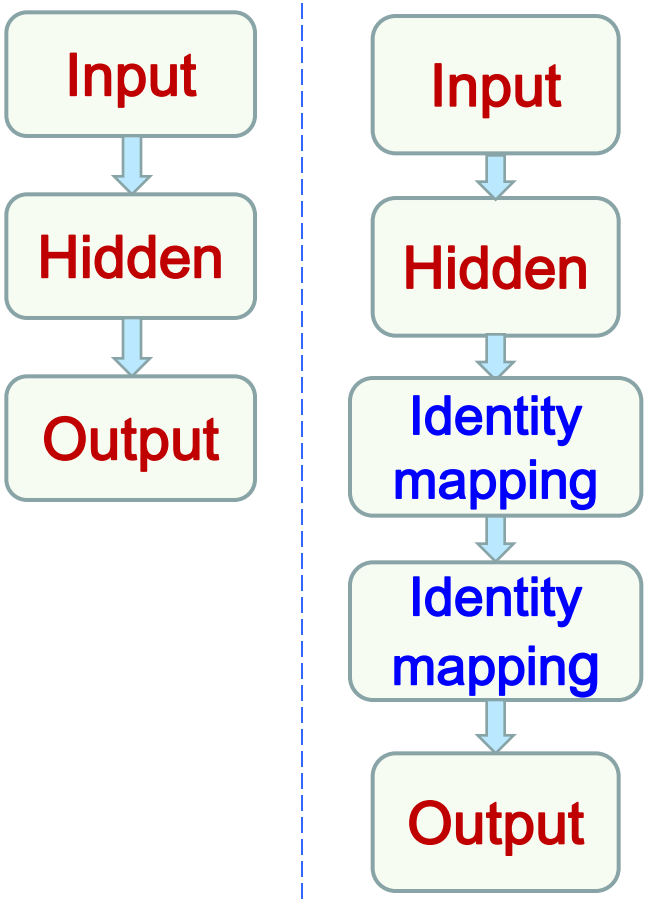
4.6 ResNet

4.6 ResNet

- Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. Deep Residual Learning for Image Recognition. CVPR, 2016.
- 动机：
  - 是否可以通过简单的层数堆叠学习更好的网络？
    - 梯度消失和爆炸?
      - 通过ReLU可以一定程度上缓解。
    - 网络退化（degradation）：随着网络的加深，准确率首先达到饱和，然后 快速退化。
      - 在训练集上的错误率同样增加，因此并非受过拟合的影响。
  - 简单的层数堆叠不能提升网络性能，如何利用网络加深带来的优势？

4.6 ResNet

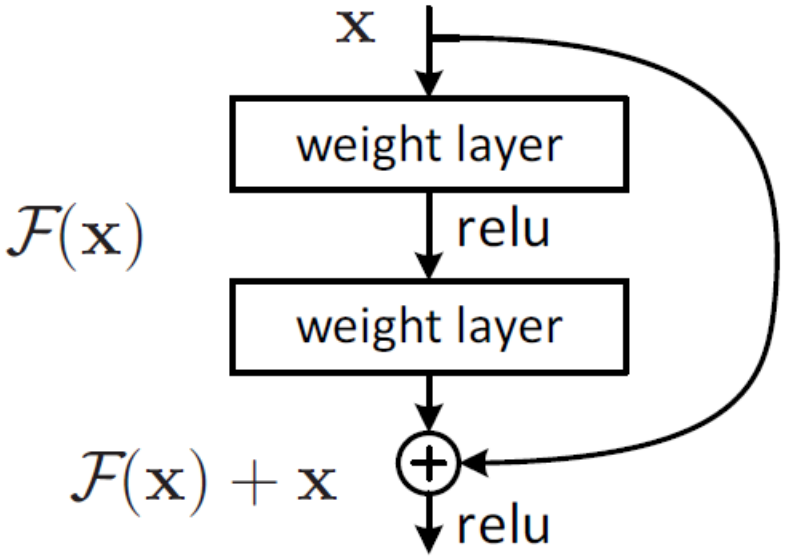
- 动机：
  - 实验表明，通过添加单位映射不能提高网络准确率，因此，网络对单位映射的逼近存在困难。
  - 相比于单位映射，网络对单位映射附近扰动的学习更加简单。



当增加单位映射将网络变得更深的时间，右边网络的性能反而更差（但是，因为是单位映射，按道理它们应该是一样或者不会更差）。这说明网络并没有很好地学习出真正的单位映射。

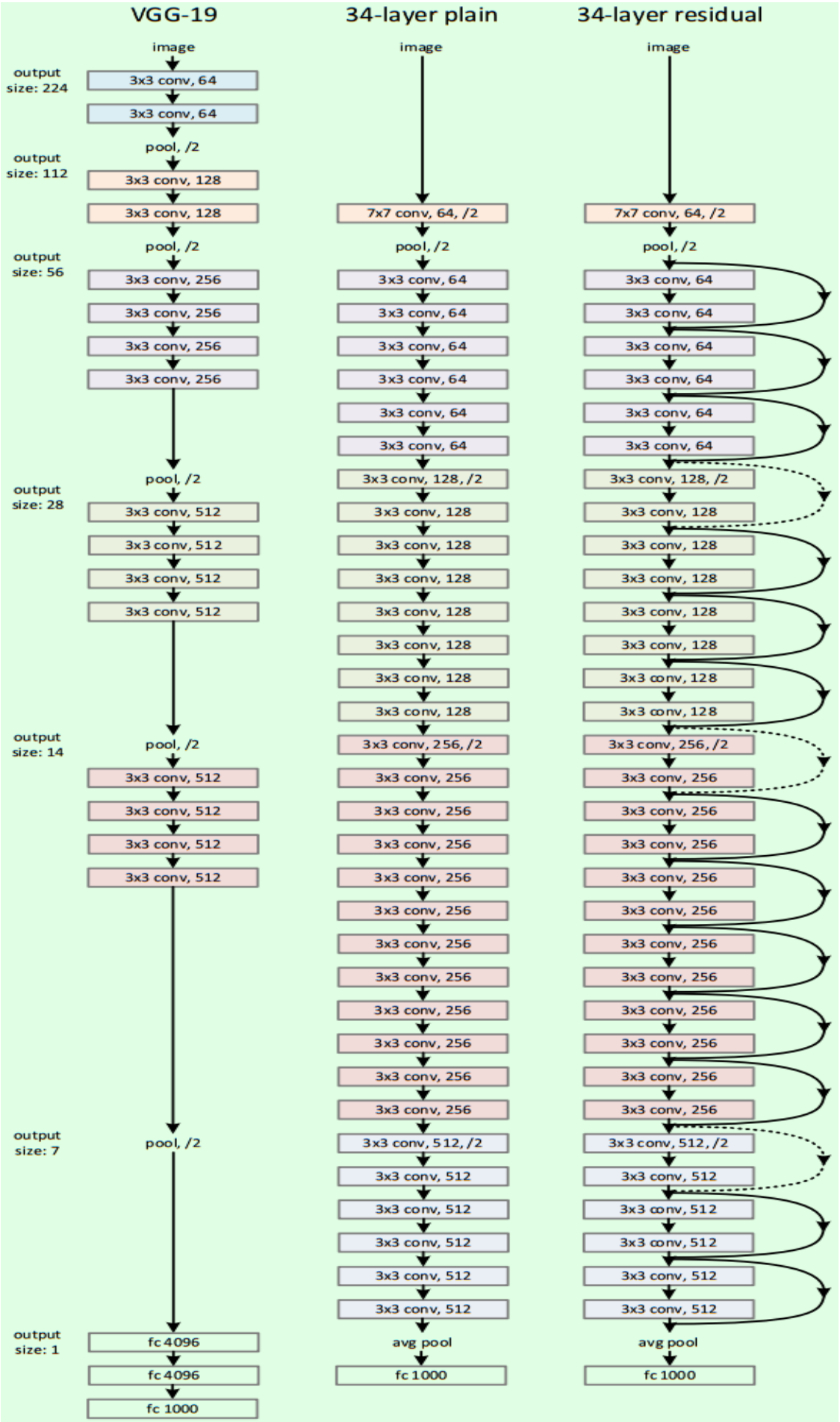
4.6 ResNet

- 网络设计：
- 输入通过shortcut与输出相加。其中，F(x) 为残差映射，线性映射则用于维度匹配。



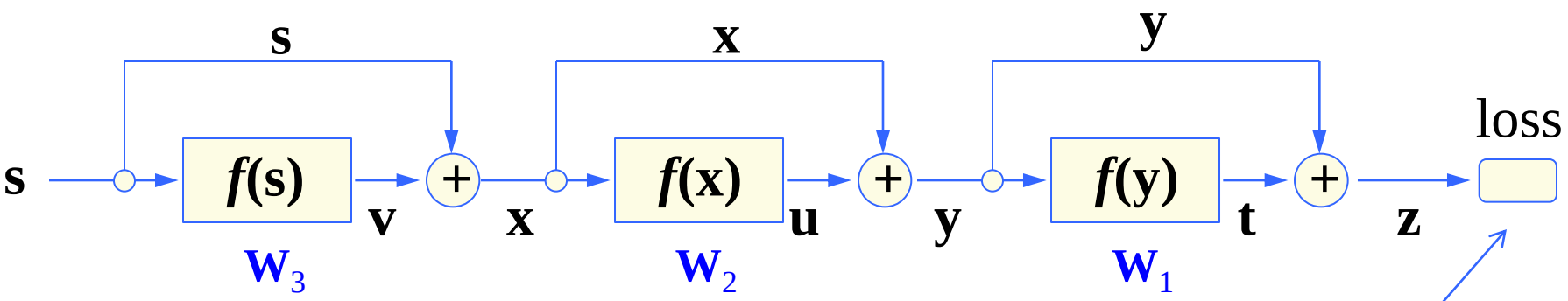
$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

残差模块



4.6 ResNet

- 为何Resnet可以很深



$$\mathbf{z}=\mathbf{y}+\mathbf{t}, \qquad \mathbf{t}=f(\mathbf{y})$$

$$\mathbf{y}=\mathbf{x}+\mathbf{u}, \qquad \mathbf{u}=f(\mathbf{x})$$

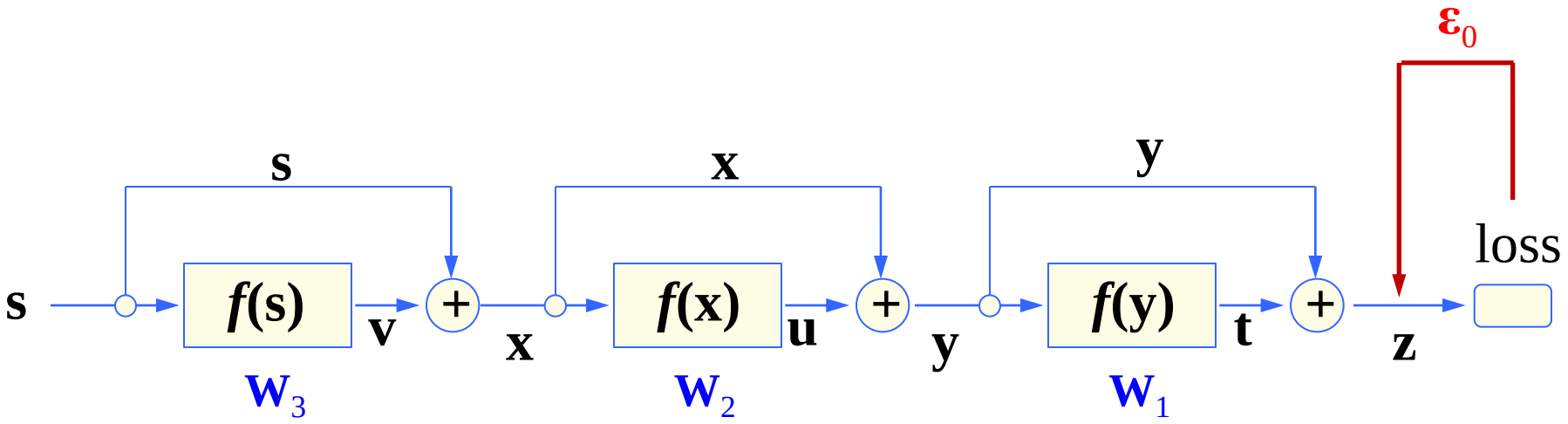
$$\mathbf{x}=\mathbf{s}+\mathbf{v}, \qquad \mathbf{v}=f(\mathbf{s})$$

$$e.g. \quad E=\left\|\mathbf{z}-\tilde{\mathbf{z}}\right\|_2^2$$

$f()$表示两层神经网络，其中$\mathbf{W}$为对应的网络参数。

4.6 ResNet

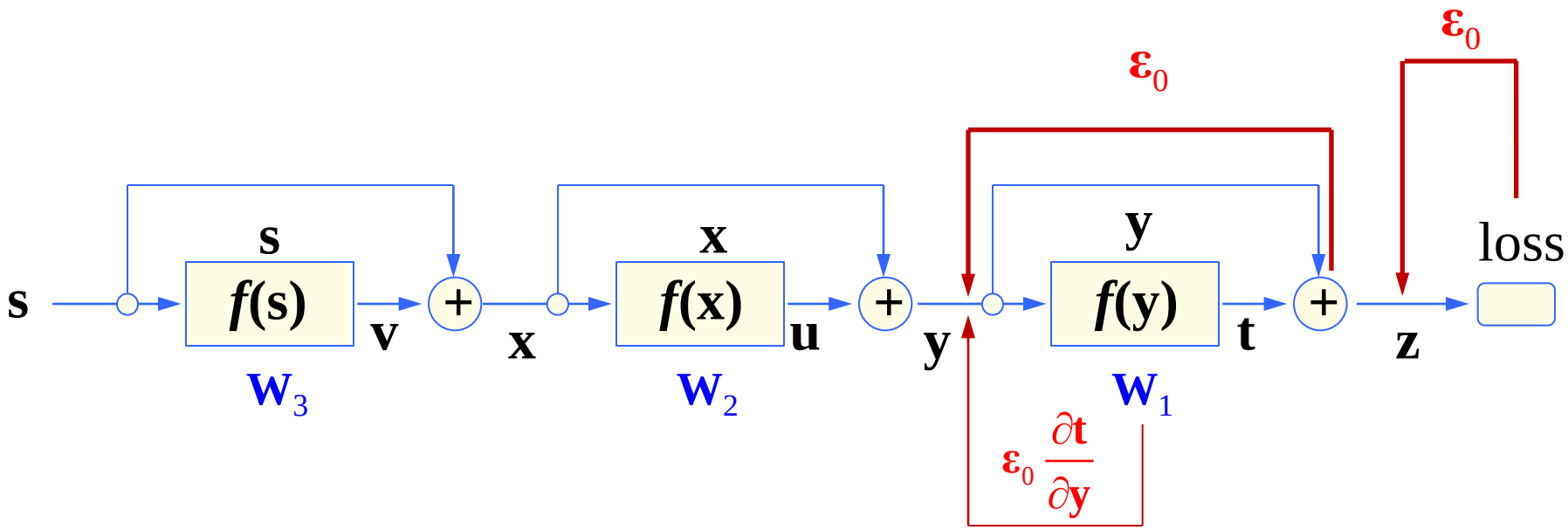
- 为何Resnet可以很深



$$\frac{\partial E}{\partial \mathbf{W}_1} = \frac{\partial E}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{W}_1} = 2(\mathbf{z}-\tilde{\mathbf{z}}) \frac{\partial \mathbf{z}}{\partial \mathbf{W}_1} = \boldsymbol{\varepsilon}_0 \frac{\partial \mathbf{z}}{\partial \mathbf{W}_1} = \boldsymbol{\varepsilon}_0 \left(\frac{\partial \mathbf{y}}{\partial \mathbf{W}_1} + \frac{\partial \mathbf{t}}{\partial \mathbf{W}_1} \right) = \boldsymbol{\varepsilon}_0 \frac{\partial \mathbf{t}}{\partial \mathbf{W}_1}$$

$$\left(\because \quad \mathbf{z} = \mathbf{y} + \mathbf{t} \right)$$

其中，$\frac{\partial E}{\partial \mathbf{z}} := \boldsymbol{\varepsilon}_0$

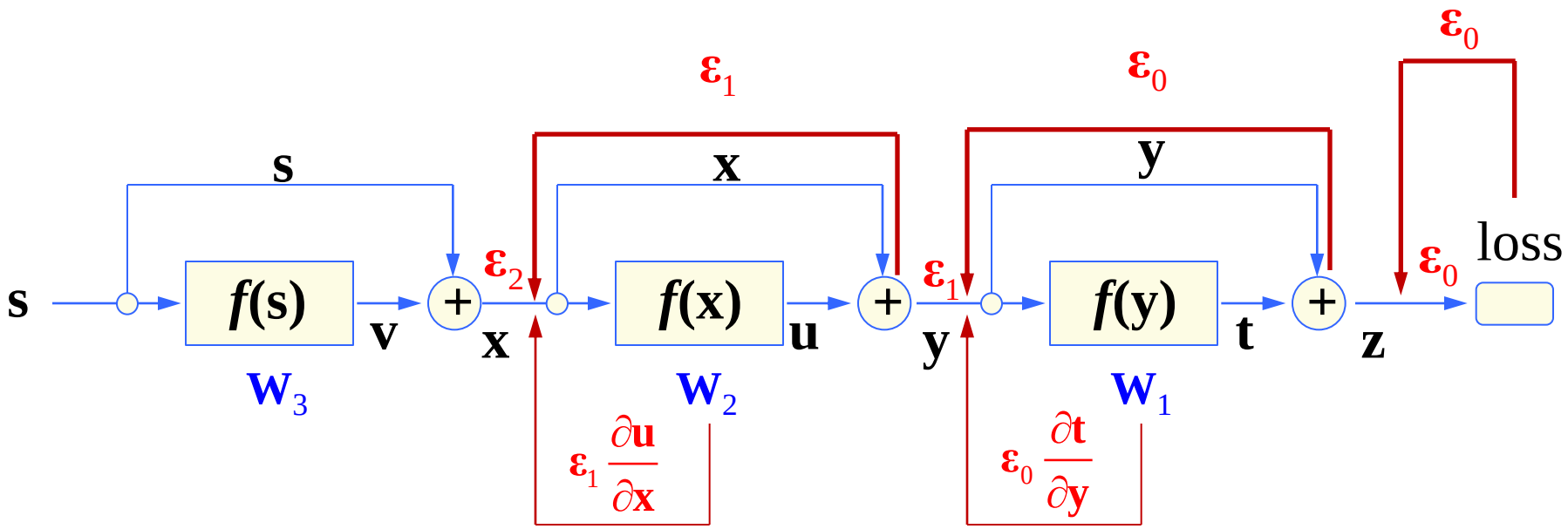


$$\frac{\partial E}{\partial \mathbf{W}_2}=\frac{\partial E}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{W}_2}=\frac{\partial E}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{W}_2}$$

$$=\frac{\partial E}{\partial \mathbf{z}}\left(\frac{\partial(\mathbf{y}+\mathbf{t})}{\partial \mathbf{y}}\right)\frac{\partial \mathbf{z}}{\partial \mathbf{W}_2}=\frac{\partial E}{\partial \mathbf{z}}\left(1+\frac{\partial \mathbf{t}}{\partial \mathbf{y}}\right)\frac{\partial \mathbf{y}}{\partial \mathbf{W}_2}=\boldsymbol{\varepsilon}_0\left(1+\frac{\partial \mathbf{t}}{\partial \mathbf{y}}\right)\frac{\partial \mathbf{y}}{\partial \mathbf{W}_2}$$

$$=\boldsymbol{\varepsilon}_1\frac{\partial \mathbf{y}}{\partial \mathbf{W}_2}$$

其中，$\dfrac{\partial E}{\partial \mathbf{y}}:=\boldsymbol{\varepsilon}_1$



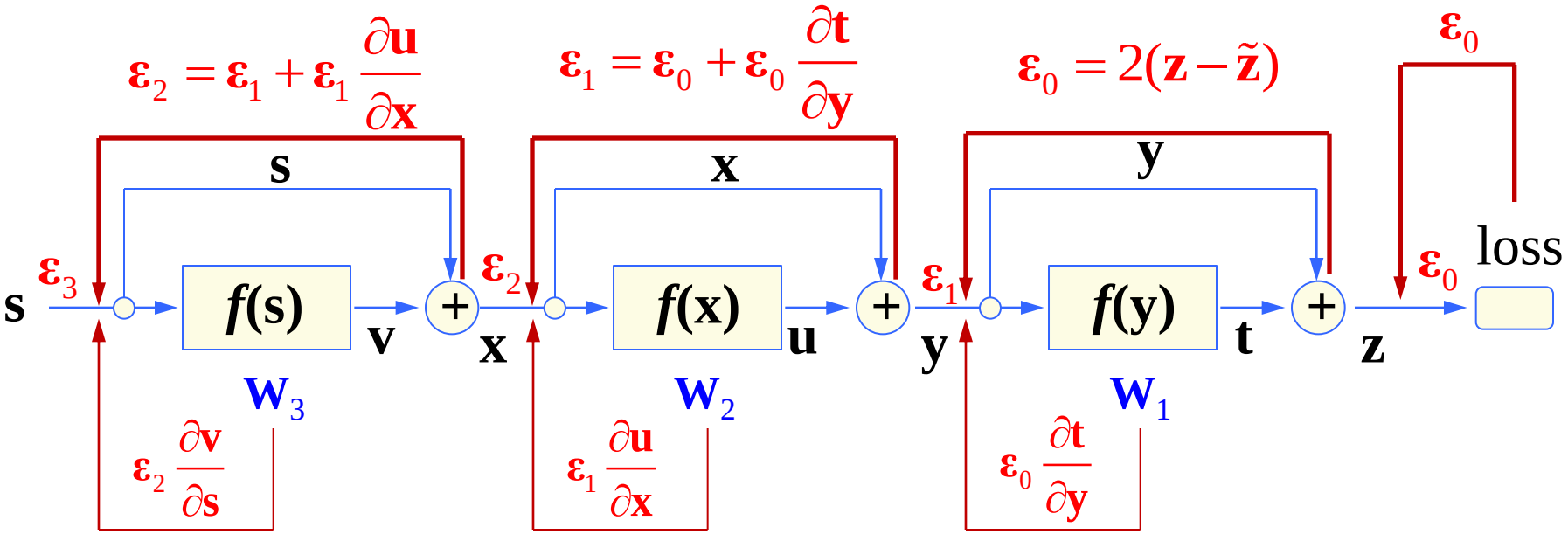
$$\frac{\partial E}{\partial \mathbf{W}_3}=\frac{\partial E}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{W}_3}=\frac{\partial E}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}$$

$$=\frac{\partial E}{\partial \mathbf{y}}\left(\frac{\partial(\mathbf{x}+\mathbf{u})}{\partial \mathbf{x}}\right)\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}=\frac{\partial E}{\partial \mathbf{y}}\left(\frac{\partial(\mathbf{x}+\mathbf{u})}{\partial \mathbf{x}}\right)\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}=\frac{\partial E}{\partial \mathbf{y}}\left(1+\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}$$

$$=\boldsymbol{\varepsilon}_1\left(1+\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}=\boldsymbol{\varepsilon}_2\frac{\partial \mathbf{x}}{\partial \mathbf{W}_2}$$

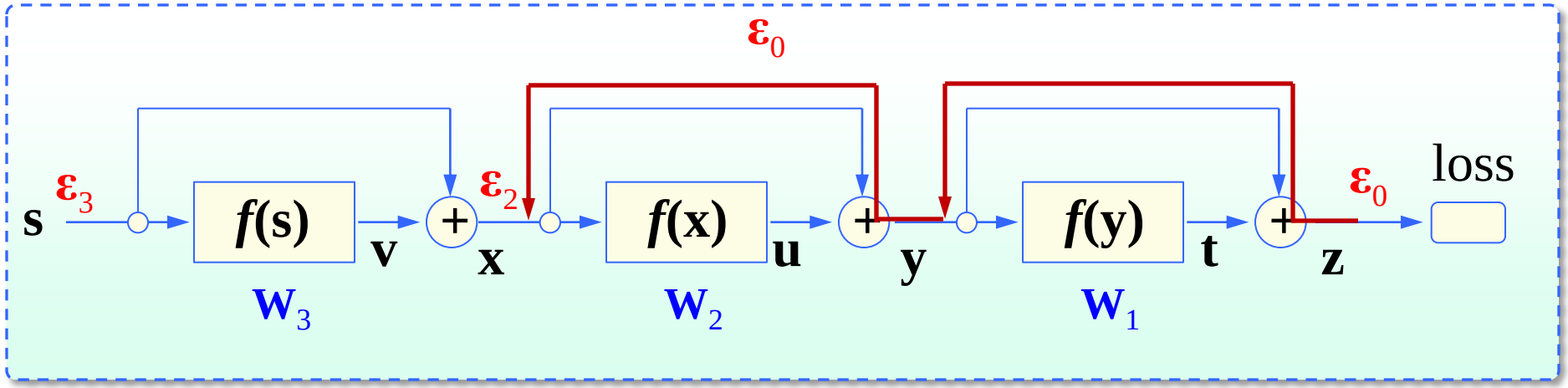
其中，$\frac{\partial E}{\partial \mathbf{x}} := \boldsymbol{\varepsilon}_2$

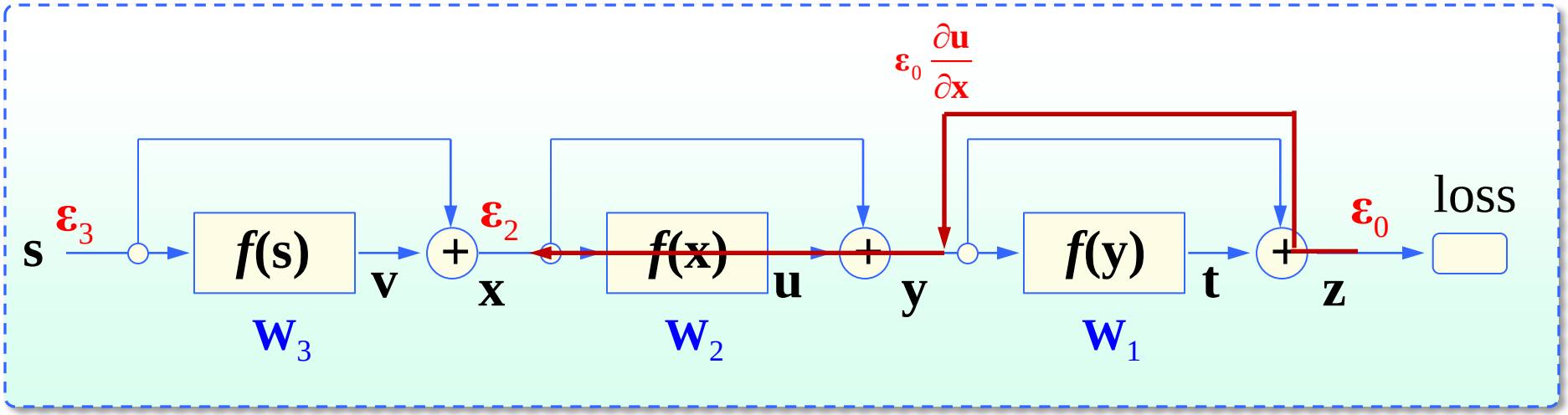
为什么Resnet可以建得很深：

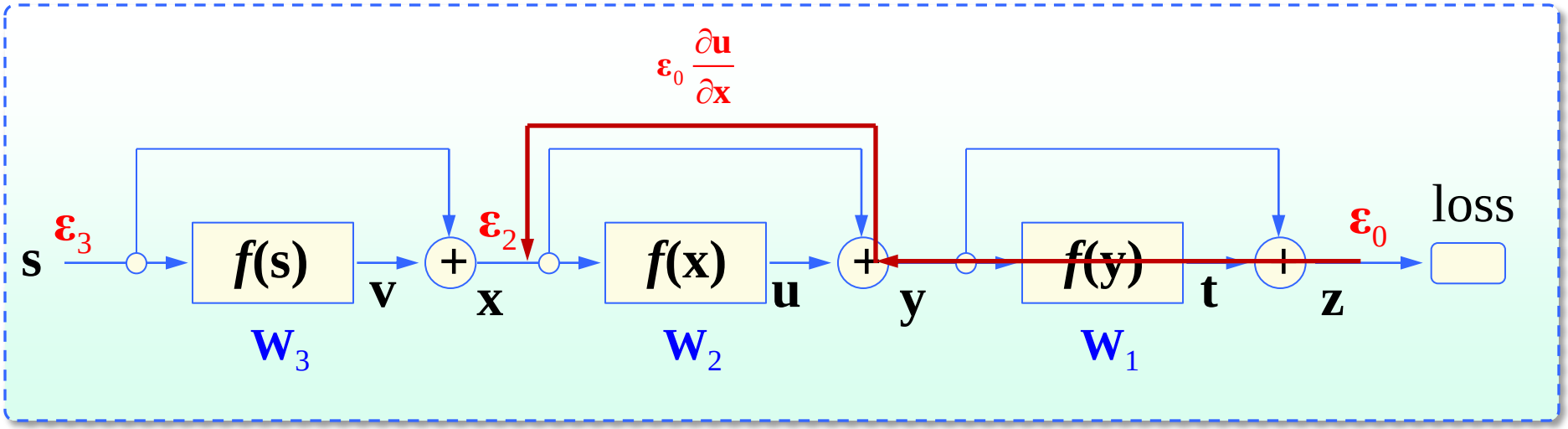


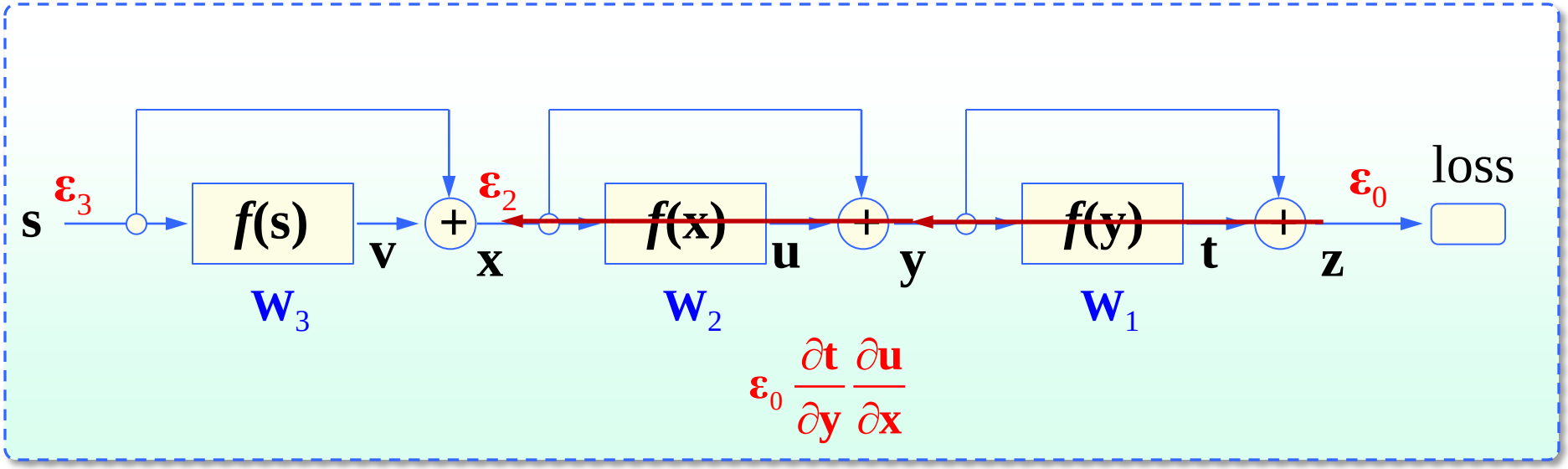
$$\varepsilon_2 = \varepsilon_1 + \varepsilon_1 \frac{\partial u}{\partial x} = \left(\varepsilon_0 + \varepsilon_0 \frac{\partial t}{\partial y}\right) + \left(\varepsilon_0 + \varepsilon_0 \frac{\partial t}{\partial y}\right)\frac{\partial u}{\partial x}$$

$$= \varepsilon_0 + \varepsilon_0 \frac{\partial t}{\partial y} + \varepsilon_0 \frac{\partial u}{\partial x} + \varepsilon_0 \frac{\partial t}{\partial y}\frac{\partial u}{\partial x}$$


ε0
ε3
ε2
ε0
loss
s
f(s)
v
+
x
f(x)
u
+
y
f(y)
t
+
z
W3
W2
W1


ε0 ∂u/∂x
ε3
ε2
ε0
loss
s
f(s)
v
+
x
f(x)
u
+
y
f(y)
t
+
z
W3
W2
W1


$\varepsilon_0 \frac{\partial u}{\partial x}$
s
ε_3
f(s)
v
+
x
ε_2
f(x)
u
+
y
f(y)
t
+
z
ε_0
loss
W_3
W_2
W_1


s
ε_3
f(s)
v
+
x
ε_2
f(x)
u
+
y
f(y)
t
+
z
ε_0
loss
W_3
W_2
W_1
$\varepsilon_0 \frac{\partial t}{\partial y} \frac{\partial u}{\partial x}$

4.7 RNN

后面的内容请自学！

4.7 Recurrent NN

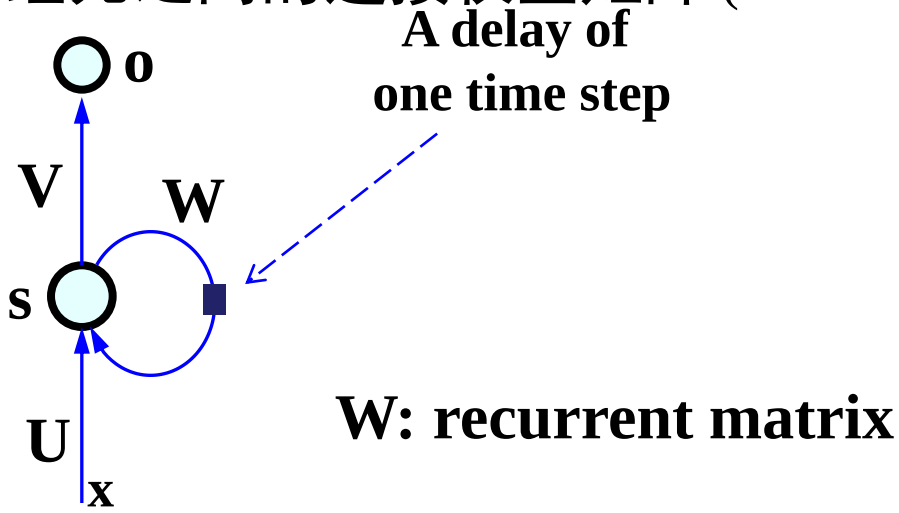
- 前馈神经网络
  - 信息向前传递，层内结点之间并不连接，适合于处理静态数据分析，如回归、分类等任务。
- 反馈神经网络
  - 全部或者部分神经元可以接受来自其它神经元或自身的神经网络结构，其拓扑结构可以是网状的，也可以是具有一定层级的。
  - 人们通常将反馈神经网络视为一个动态系统，主要关心其随时间变化的动态过程。
    - Hopfield 网络

4.7 Recurrent NN

- Recurrent NN（RNN）
  - 至少包含一个反馈连接的神经网络结构。因此，**网络的激励可以沿着一个loop 进行流动**。这种网络结构特别适合于处理时序数据。
- 几种经典的RNN：
  - Hopfield 网络 – 全连接
  - Elman 网络：包含输入层、隐含层和输出层，但隐含层和输入层之间存在一个反馈连接，这种连接通常称为recurrent 连接，即回归连接。
    - 这种回归连接使得Elman 网络具有检测和产生时变模式的能力
  - 对角自反馈神经网络：隐含层的神经元具有反馈连接，但隐含层神经元之间并不连接

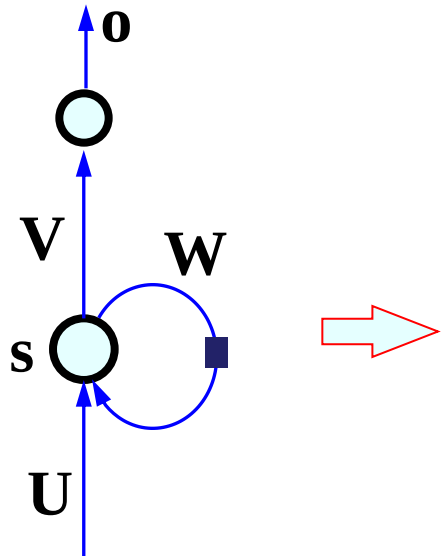
4.7 Recurrent NN

- 层次型Recurrent NN的基本结构(普通RNN)
  - **x**： 输入信号；**o**：输出信号
  - **s**： 隐含层结点的输出
  - **U**：输入至隐含层的连接权重矩阵 (input to hidden)
  - **V**：隐含层至输出层的连接权重矩阵 (hidden to output)
  - **W**：隐含层神经元之间的连接权重矩阵 (hidden to hidden)



4.7 Recurrent NN

- 网络结构



$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{U}\mathbf{x}_t$$
$$\mathbf{s}_t = \tanh(\mathbf{a}_t)$$
$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{s}_t$$

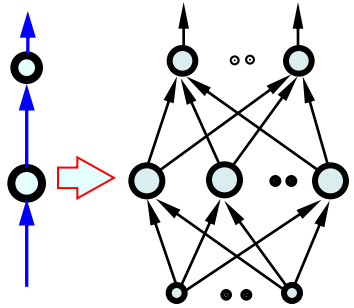
$$\mathbf{p}_t = f(\mathbf{o}_t)$$

比如，对分类问题，激励函数 ***f*(.)** 可取为**softmax**，见后面的**PPT**

b, c: 偏移向量
U, V, W: 权重矩阵
$\mathbf{p}_t$: 属于*c*类的概率向量

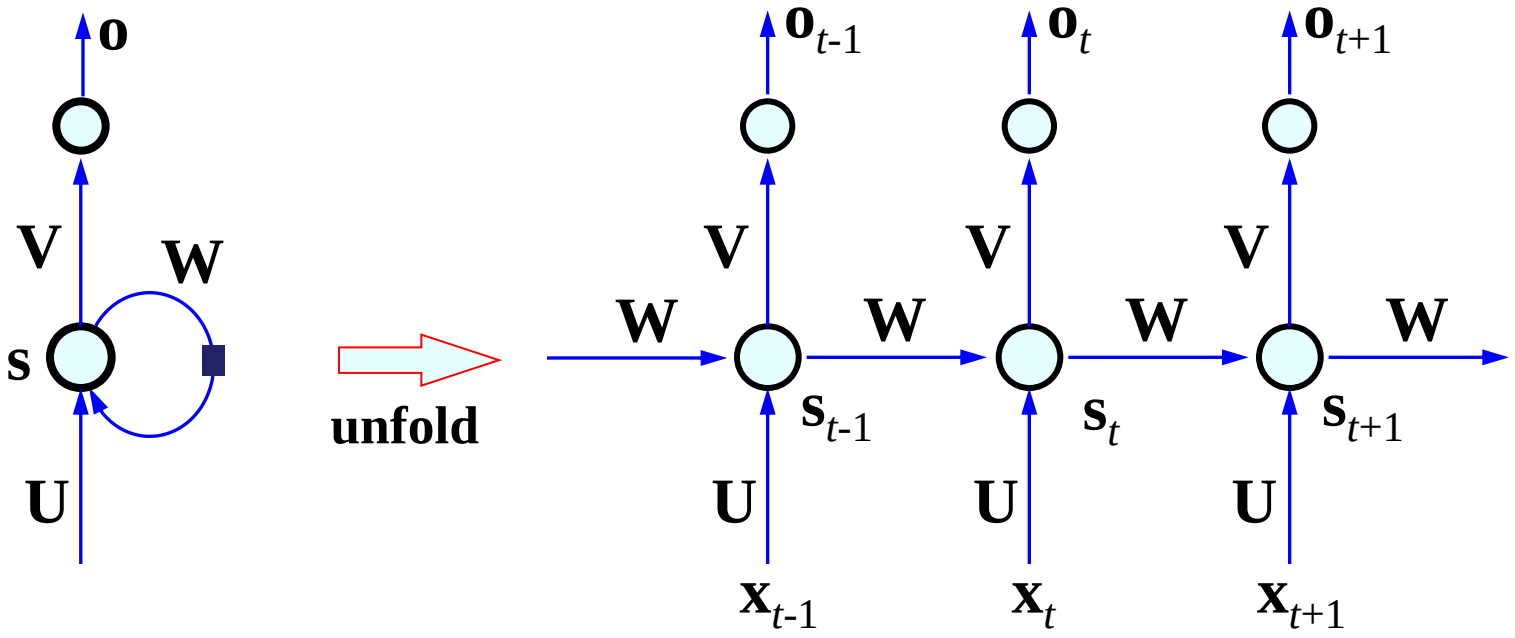
网络的功能可以解释为： How the state s_t at time *t* captures and summarizes the information from the previous inputs $x_1, x_2, \ldots, x_t$.

4.7 Recurrent NN



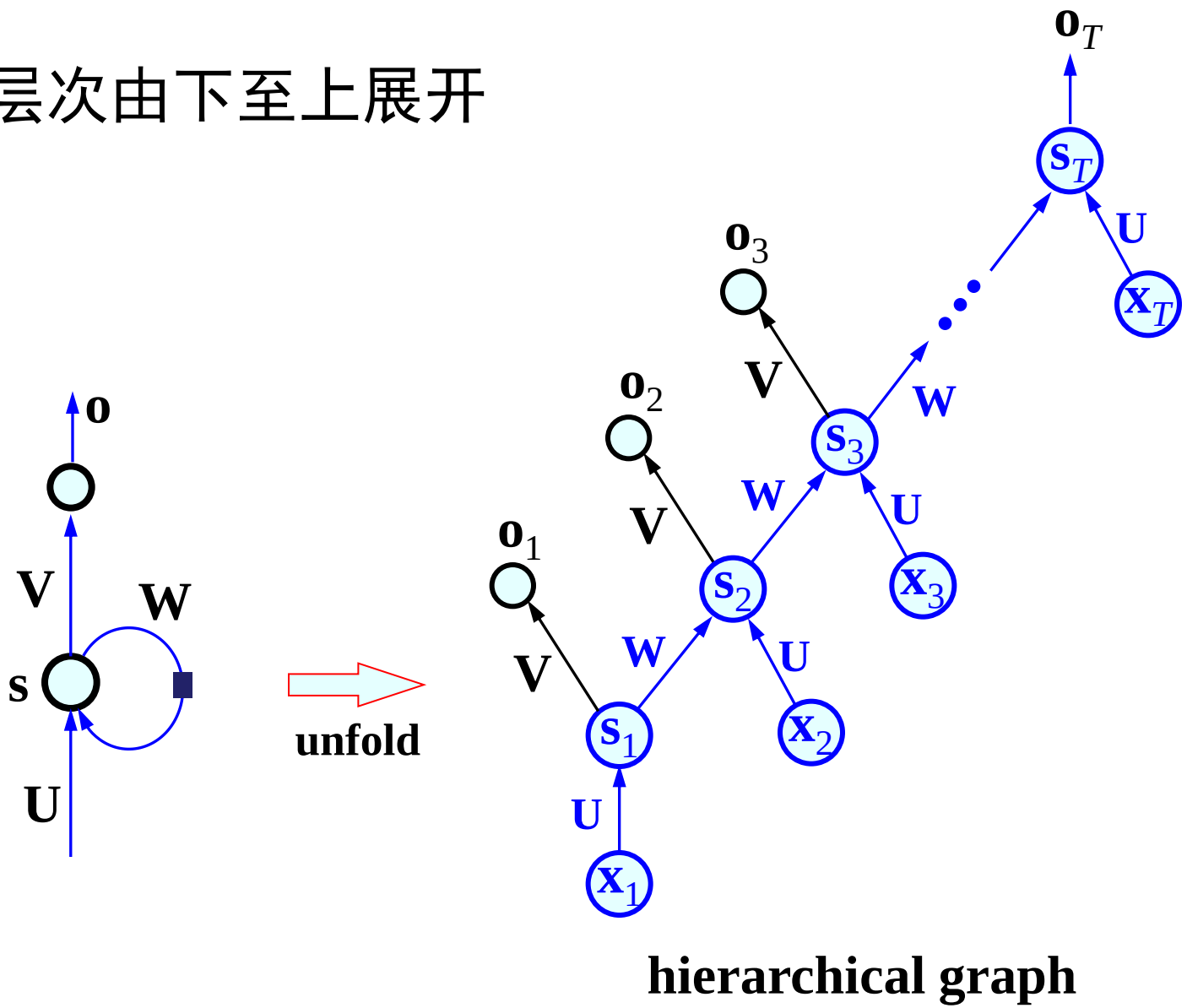
- 按时间顺序展开

权值共享: U, V, W



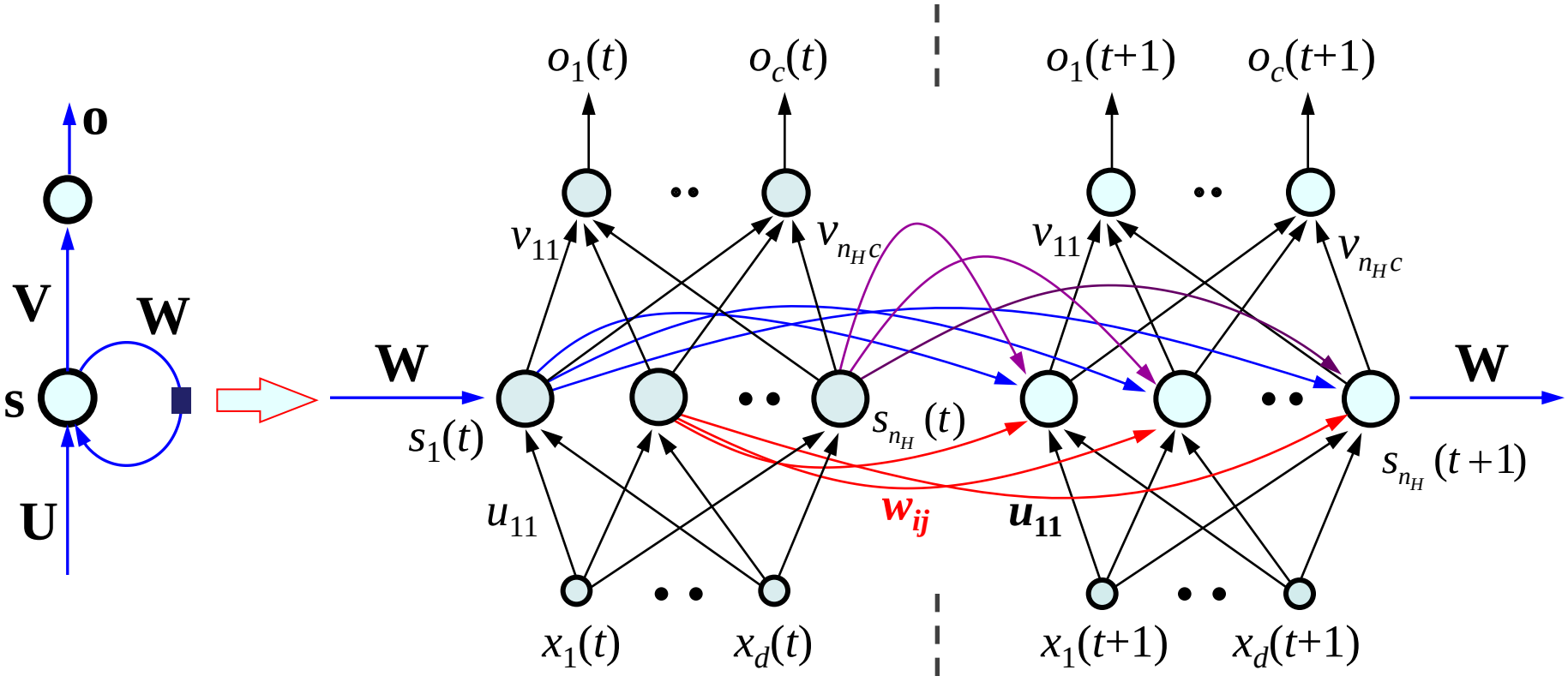
flow graph

- 按层次由下至上展开


o
V
W
s
U
unfold
o_T
s_T
U
x_T
o_3
V
W
o_2
s_3
V
W
U
o_1
s_2
x_3
V
W
U
s_1
x_2
U
x_1
hierarchical graph

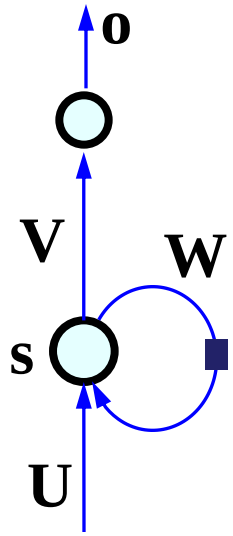
4.7 Recurrent NN

- 按时间顺序全结点展开



4.7 Recurrent NN

- 网络训练



$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{s}_{t-1} + \mathbf{U}\mathbf{x}_t$$

$$\mathbf{s}_t = \tanh(\mathbf{a}_t) \quad \text{(hyperbolic tangent func.)}$$

$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{s}_t$$

$$\mathbf{p}_t = \text{soft max}(\mathbf{o}_t) \quad \text{(例如)}$$

The total loss for a given input/target sequence pair (**x**,y) would then be just the sum of the losses over all the time steps:

$$L(\mathbf{x}, \mathbf{y}) = \sum_t L(\mathbf{x}, y_t),$$

对分类：$L(\mathbf{x}, \mathbf{y}) := -\sum_t \log p_{y_t}$

4.7 Recurrent NN

O_i即为第i个结点收集到的加权和

- softmax vs softmax loss

$$\text{soft}\max(\mathbf{o}) = [\sigma_1(\mathbf{o}), \sigma_2(\mathbf{o}), \ldots, \sigma_c(\mathbf{o})]^T$$

样本x属于c类的概率

$$\sigma_i(\mathbf{o}) = \frac{\exp(o_i)}{\sum_{j=1}^{c} \exp(o_j)}, \quad i = 1, 2, \ldots, c,$$

比如，多类线性判别：

$$o = \mathbf{w}^T \mathbf{x} + w_0$$

假设数据 **x** 对应的类别为 y, **y取 1,2,…,c 中的某个整数**。计算最大似然就是要最大化 $\sigma_y(\mathbf{o})$ 的值 (即第 y 个分量）。

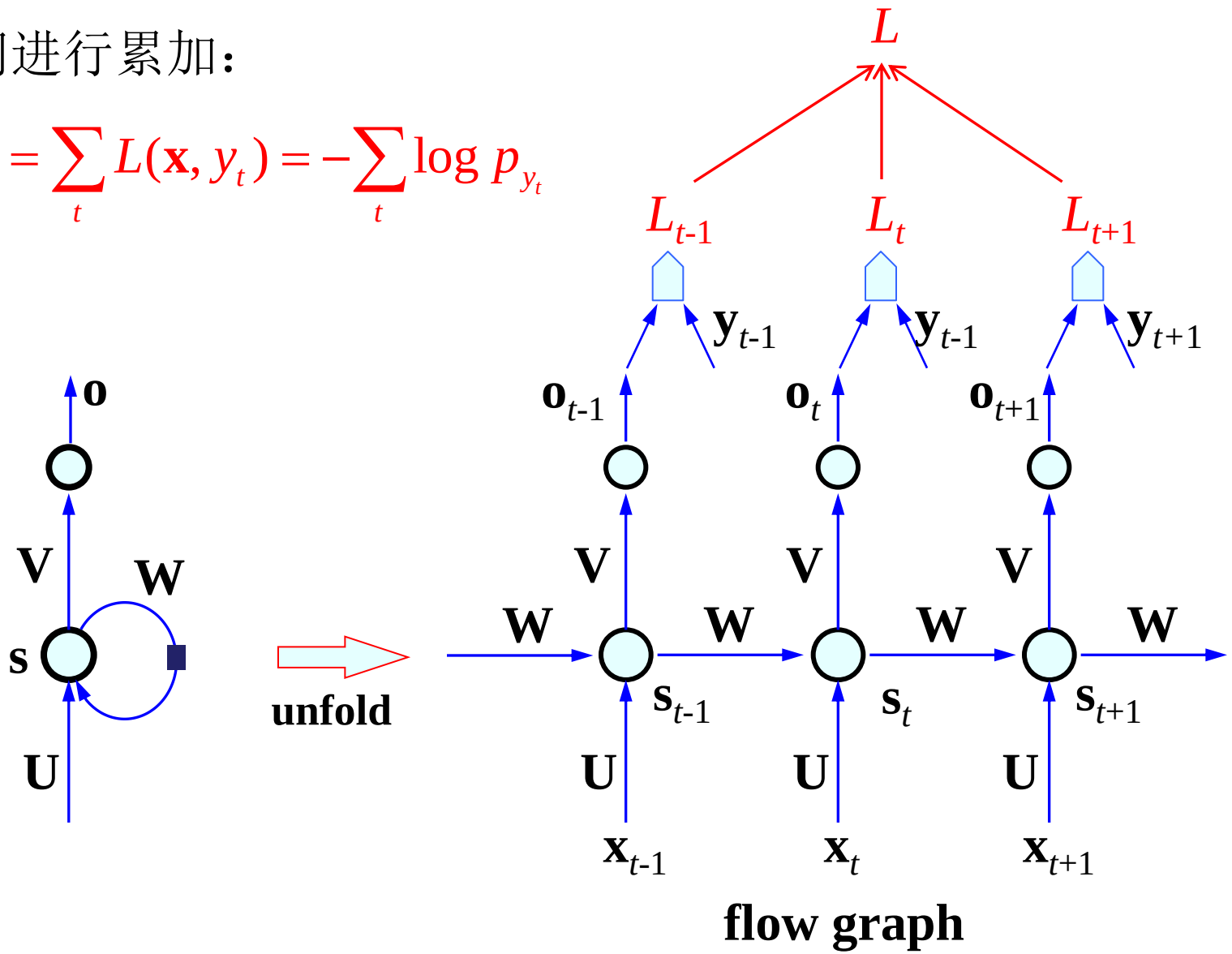
采用负对数似然，有：

$$l(\mathbf{x}, y) = -\log(\sigma_y(\mathbf{o})) = -\log\left(\frac{e^{o_y}}{\sum_{j=1}^{c} e^{o_j}}\right) = \log\left(\sum_{j=1}^{c} e^{o_j}\right) - o_y$$

y 取对应的类别整数

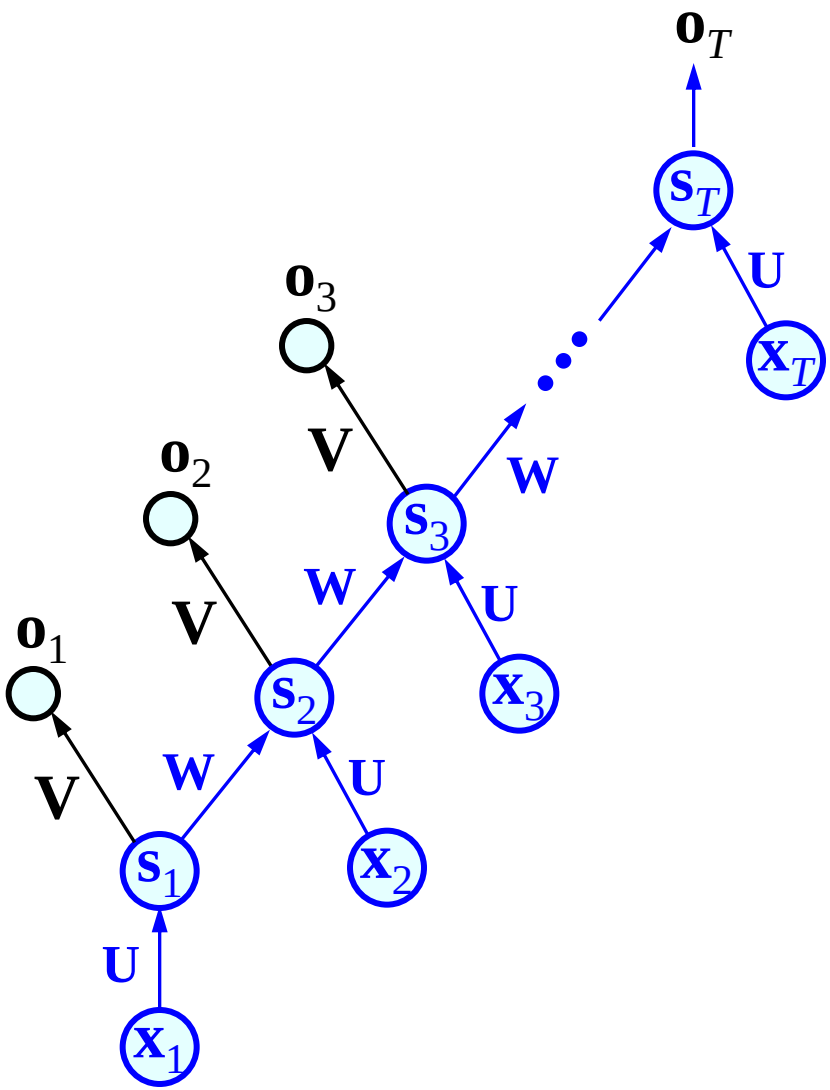
按时间进行累加：

$$L(\mathbf{x},\mathbf{y}) = \sum_t L(\mathbf{x}, y_t) = -\sum_t \log p_{y_t}$$



4.7 Recurrent NN

- 网络训练
  - 从多层前向网络的角度来看，其实上已经清楚怎样来训练这个网络，即采用类似的反向传播（BP）算法完成此事。通过准备误差来启动BP算法。
  - 唯一不同之处在于：权重是共享的，是相同的。因此简单地采用现有BP方法显得效率不高。



hierarchical graph

4.8 LSTM

4.8 Long Short Term Memory

- 采用一些控制门来减少梯度累积的长度
  - **RNN**
    - **L1 / L2 regularization on the recurrent weights**
    - **Smart initialization (no one knows..)**
    - **Self-loop, leak units with linear self-connections**
    - **Echo State Networks (RNN with a sparse-connected hidden layer)**
    - **Hessian-Free optimization**
    - **Gradient Clip**
    - **…**

Not content-dependent

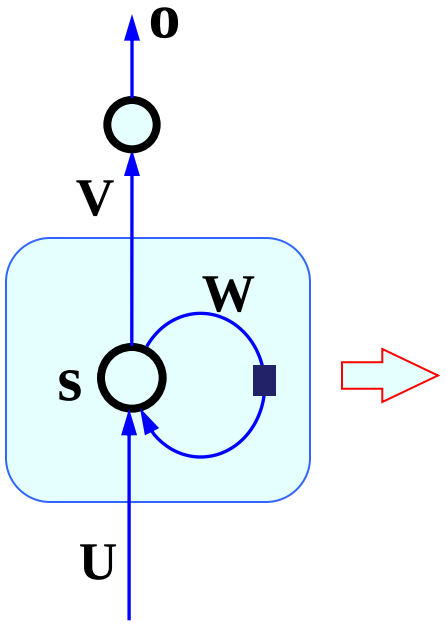
LSTM一旦信息得到利用，我们希望该结点能释放(遗忘)这种累积效应，从而使网络更具有灵活性和自主学习性(依赖于学习内容来自我决定)。**Just LSTM!**

4.8 LSTM

- 结构
  - 传统的RNN仅仅应用激励函数作用于“当前时刻输入信号＋上一时刻的隐含结点输出”的仿射变换结果，作为当前隐含结点的输出。
  - LSTM网络则将隐含层的细胞单元（隐含单元）设计为所谓的LSTM细胞单元
  - 每一个LSTM细胞含有与传统的RNN细胞相同的输入和输出，但它额外包含一个控制信息流动的“门结点系统”。
  - 门系统包含三个部分，除了对LSTM细胞的输入、输出进行加权控制之外，还对记忆（遗忘）进行加权控制

4.8 LSTM

- 结构
  - 由传统的细胞结构变为带有三个权重门的结构



由简单的输入、输出、隐含层自循环增加了三个门单元：

—遗忘门：控制对细胞内部状态的遗忘程度

—输入门：控制对细胞输入的接收程度

—输出门：控制对细胞输出的认可程度

三个门均由当前输入信号和隐含层前一时刻的输出共同决定

- 输入门“input gate”的作用是提供一个是否接收当前正常输入(包含信号和隐结点状态)信息的权重(0~1)。
  - 其值取决于当前输入信号和前一时刻隐含层的输出:

$$in_{t,i} = \text{sigmod}\left(b_{in}(i) + \sum_j U_{in}(i,j)x_{t,j} + \sum_j W_{in}(i,j)h_{t-1,j}\right)$$

time t, node i

$\mathbf{x}_t$ ： 当前输入向量；
$\mathbf{h}_{t-1}$ ： 当前隐含层向量，所有结点在**t-1**时刻的输出；
$\mathbf{b}_{in}$ ： 输入门的偏移向量；
$\mathbf{U}_{in}$ ： 输入门的输入权重矩阵；
$\mathbf{W}_{in}$ ： 输入门的回归权重矩阵；
$\mathbf{i}_t$ ： $=[in_{t,1}, in_{t,2}, in_{t,3}, \ldots.]$ (用向量记录所有权重）

- 遗忘门“forget gate”的作用是提供一个遗忘当前状态的权重(0~1)。
  - 其值取决于当前输入信号和前一时刻隐含层的输出:

$$f_{t,i} = \text{sigmod}\left(b_f(i) + \sum_j U_f(i,j)x_{t,j} + \sum_j W_f(i,j)h_{t-1,j}\right)$$

time t, cell i

$\mathbf{x}_t$ ： 当前输入向量；
$\mathbf{h}_{t-1}$ ： 当前隐含层向量，包含所有结点的t-1时刻输出；
$\mathbf{b}_f$ ： 遗忘门的偏移向量；
$\mathbf{U}_f$ ： 遗忘门的输入权重矩阵；
$\mathbf{W}_f$ ： 遗忘门的回归权重矩阵；
$\mathbf{f}_t$ ： $=[f_{t,1}, f_{t,2}, f_{t,3}, \ldots]$ (用向量记录所有权重)

- LTSM 细胞产生的新记忆，由输入与遗忘两部分组成：

产生新记忆　　读出旧记忆　　贡献部分新忘记

$$s_{t+1,i} = f_{t,i} \cdot s_{t,i} + in_{t,i} \cdot c_{t,i}$$

$$c_{t,i} = \tanh\left(b(i) + \sum_j U(i,j) x_{t,j} + \sum_j W(i,j) h_{t-1,j}\right)$$

$\mathbf{x_t}$ ： 当前输入向量；

$\mathbf{h_{t-1}}$ ： 当前隐含层向量，包含所有结点的t-1时刻的输出；

b ： 正常的输入层至隐层的偏移向量；

U ： 正常的输入层至隐层的输入权重矩阵；

W ： 正常的输入层至隐层的回归权重矩阵；

贡献部分新忘记

$\mathbf{s_{t+1}}$ ： $=[s_{t+1,1}, s_{t+1,2}, s_{t+1,3}, \ldots]$ (用向量记录所有新记忆)

$\mathbf{c_t}$ ： $=[c_{t,1}, c_{t,2}, c_{t,3}, \ldots]$ (用向量记录所有候选记忆)

- 输出门“output gate”的作用是对当前输出提供一个 0~1之间的权重（对输出的认可程度）。
  - 其值取决于当前输入信号和前一时刻隐含层的输出:

$$o_{t,i} = \text{sigmod}\left(b_o(i) + \sum_j U_o(i,j)x_{t,j} + \sum_j W_o(i,j)h_{t-1,j}\right)$$

time t, cell i

$\mathbf{x}_t$ ： 当前输入向量；

$\mathbf{h}_{t-1}$ ： 当前隐含层向量，包含所有结点的t-1时刻的输出；

$\mathbf{b}_o$ ： 输出门的偏移向量；

$\mathbf{U}_o$ ： 输出门的输入权重矩阵；

$\mathbf{W}_o$ ： 输出门的回归权重矩阵；

$\mathbf{o}_t$ ： $=[o_{t,1}, o_{t,2}, o_{t,3}, \ldots.]$ (用向量记录所有权重)

4.8 LSTM

- 输出
  - 隐含层结点的输出由激励后的内部状态（此时称为记忆）与输出门权重共同决定：

$$h_{t,i} = o_{t,i} \tanh\left(s_{t,i}\right)$$

输出门提供的权重

4.8 LSTM

- 结构描述——采用矩阵形式

遗忘门、输入门、输出门：

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

$$\mathbf{i}_t = \text{sigmod}\left(\mathbf{b}_{in} + \mathbf{U}_{in} \mathbf{x}_t + \mathbf{W}_{in} \mathbf{h}_{t-1}\right)$$

$$\mathbf{o}_t = \text{sigmod}\left(\mathbf{b}_o + \mathbf{U}_o \mathbf{x}_t + \mathbf{W}_o \mathbf{h}_{t-1}\right)$$

Cell产生的新记忆：

$$\mathbf{s}_t = \mathbf{f}_t \otimes \mathbf{s}_{t-1} + \mathbf{i}_t \otimes \mathbf{c}_t$$

Cell的输出：

$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh\left(\mathbf{s}_t\right)$$

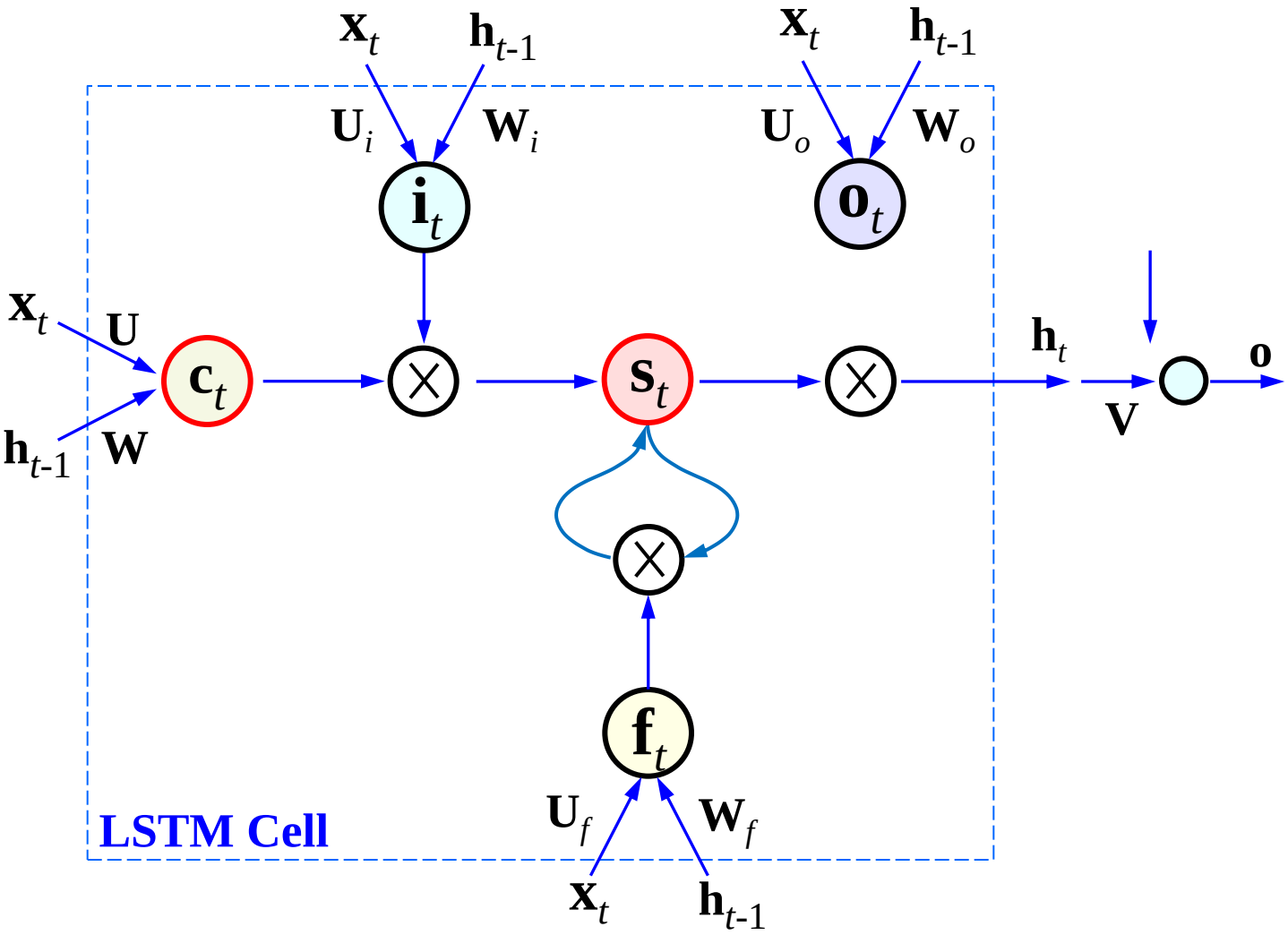
候选记忆（新贡献部分）：

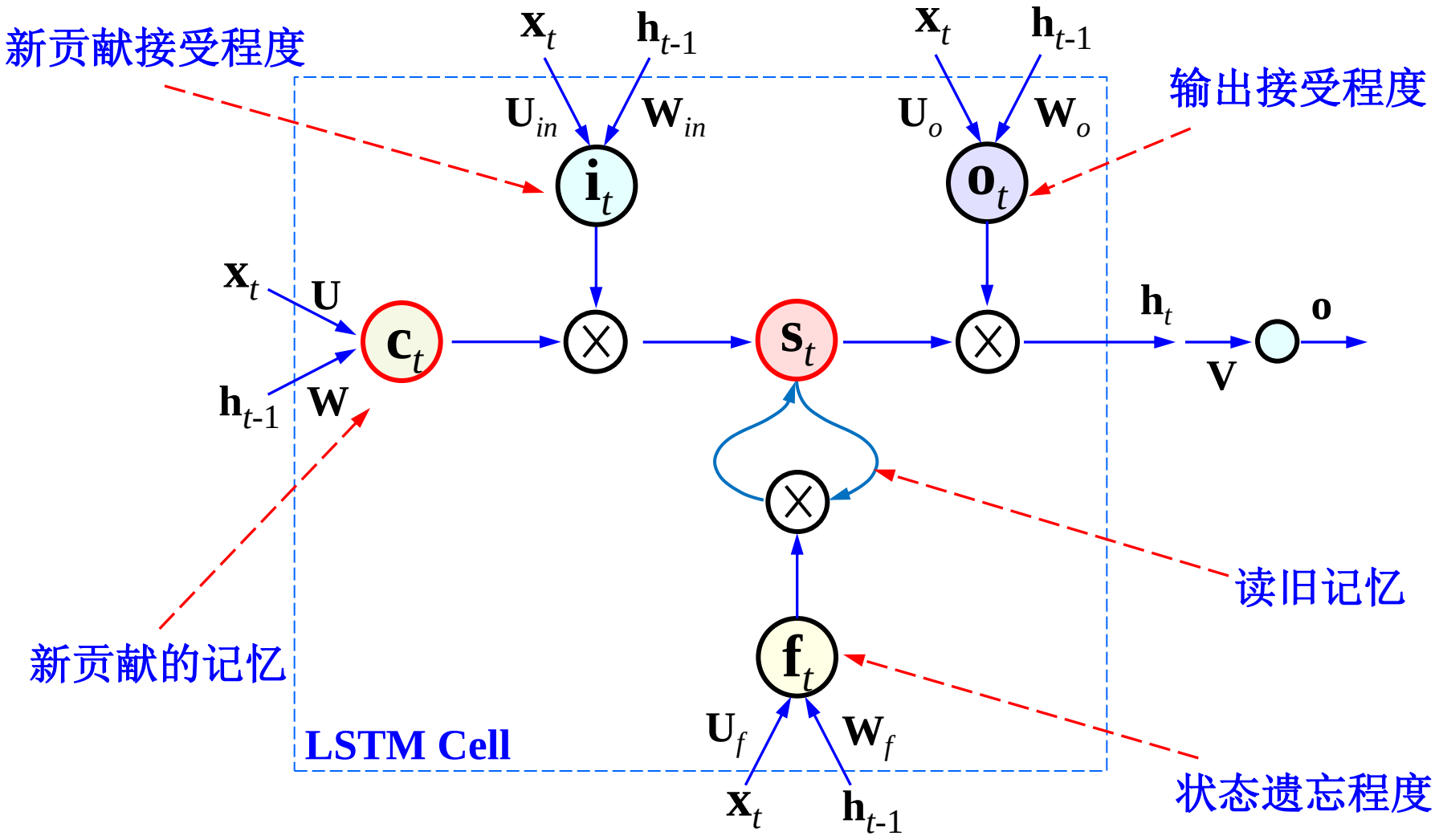
$$\mathbf{c}_t = \tanh\left(\mathbf{b} + \mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}\right)$$

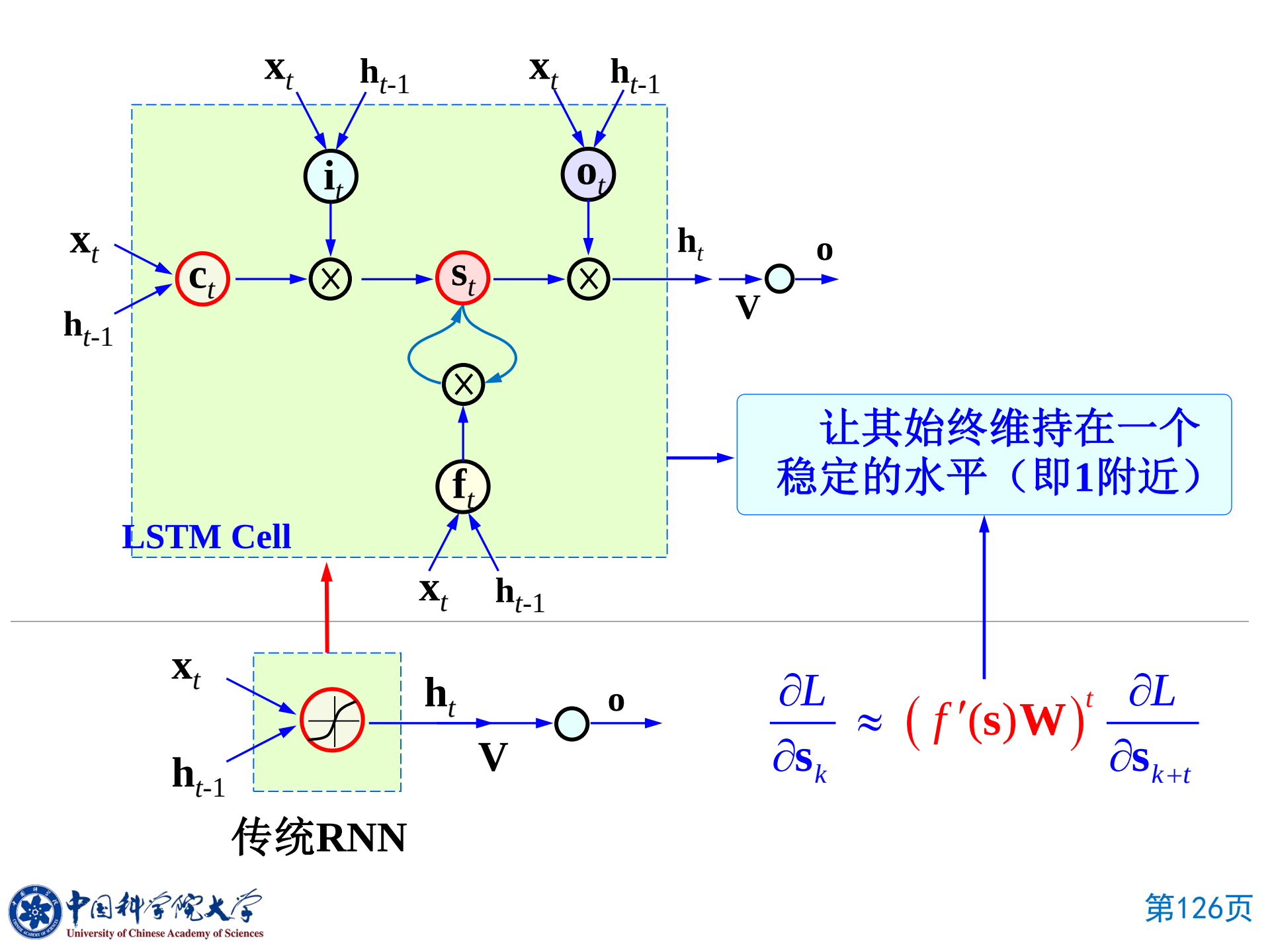
网络的输出：

$$\mathbf{z}_t = \text{softmax}\left(\mathbf{V}\mathbf{h}_t + \mathbf{c}\right)$$

LTSM网络结构（另一种展示）




新贡献接受程度
x_t
h_{t-1}
U_{in}
W_{in}
i_t
x_t
h_{t-1}
U_o
W_o
o_t
输出接受程度
x_t
U
c_t
h_{t-1}
W
s_t
h_t
V
o
新贡献的记忆
读旧记忆
f_t
U_f
W_f
x_t
h_{t-1}
状态遗忘程度
LSTM Cell


x_t h_{t-1} x_t h_{t-1}
i_t o_t
x_t c_t s_t h_t o
h_{t-1} V
f_t
LSTM Cell
x_t h_{t-1}
让其始终维持在一个稳定的水平（即1附近）
x_t h_t o
h_{t-1} V
传统RNN
$\frac{\partial L}{\partial s_k} \approx \left(f'(s)W \right)^t \frac{\partial L}{\partial s_{k+t}}$

4.8 LSTM

- 结构描述——采用矩阵形式

遗忘门、输入门、输出门：

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

$$\mathbf{i}_t = \text{sigmod}\left(\mathbf{b}_{in} + \mathbf{U}_{in} \mathbf{x}_t + \mathbf{W}_{in} \mathbf{h}_{t-1}\right)$$

$$\mathbf{o}_t = \text{sigmod}\left(\mathbf{b}_o + \mathbf{U}_o \mathbf{x}_t + \mathbf{W}_o \mathbf{h}_{t-1}\right)$$

候选记忆（新贡献部分）：

$$\mathbf{c}_t = \tanh\left(\mathbf{b} + \mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}\right)$$

Cell产生的新记忆：

$$\mathbf{s}_t = \mathbf{f}_t \otimes \mathbf{s}_{t-1} + \mathbf{i}_t \otimes \mathbf{c}_t$$

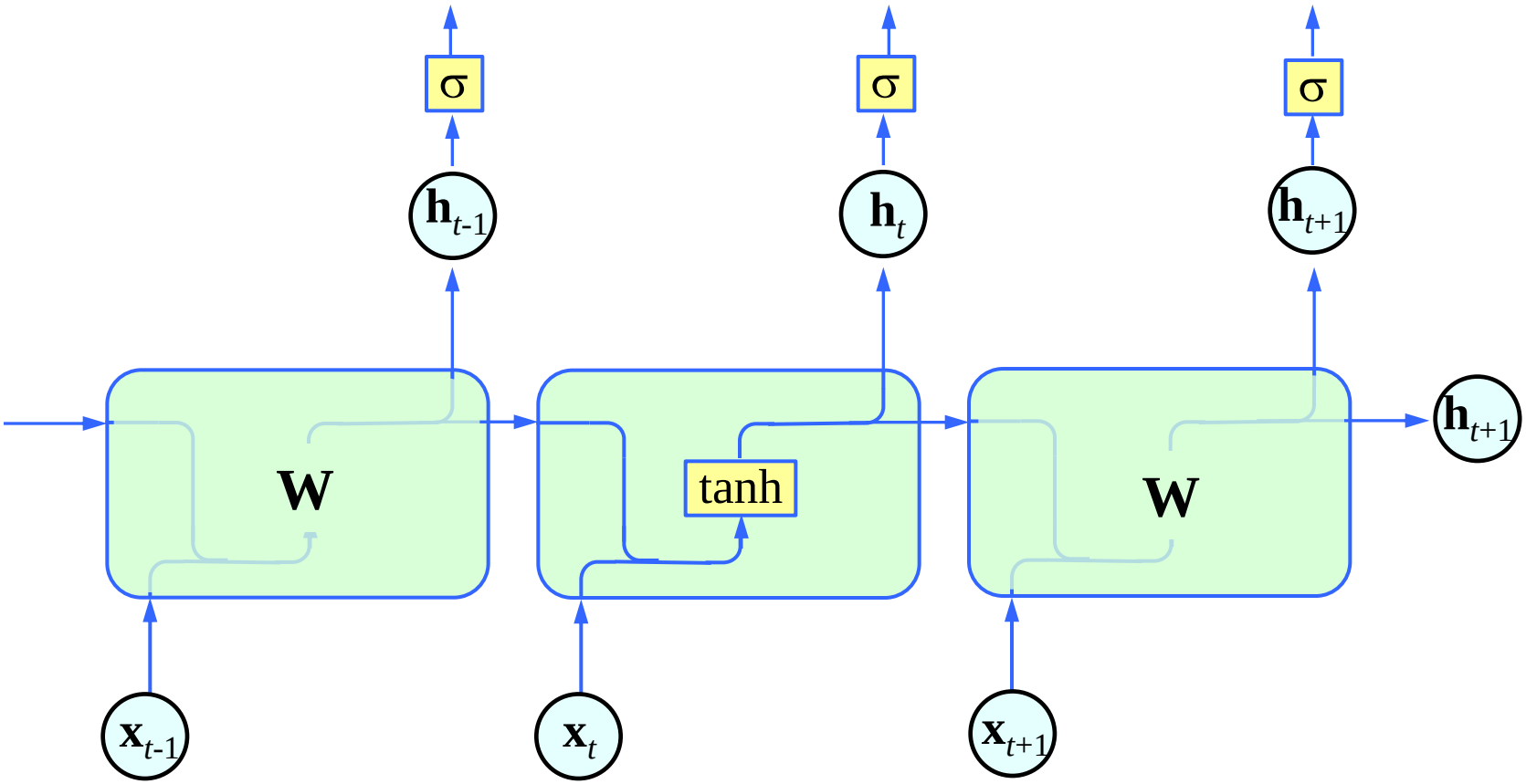
Cell的输出：

$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh\left(\mathbf{s}_t\right)$$

网络的输出：

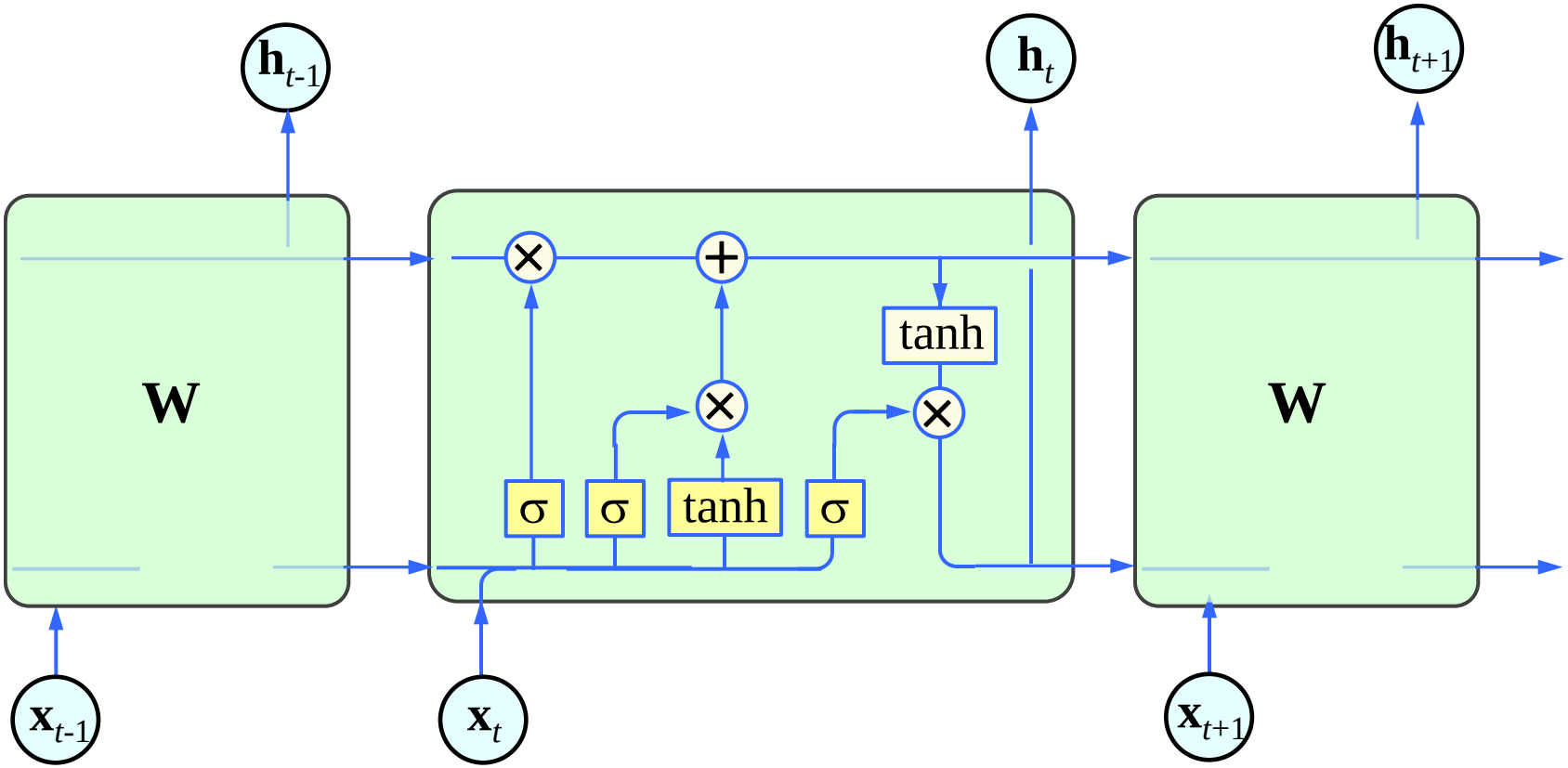
$$\mathbf{z}_t = \text{softmax}\left(\mathbf{V}\mathbf{h}_t + \mathbf{c}\right)$$

From RNN to LSTM:



标准 RNN 中的重复模块包含单一的层

From RNN to LSTM:

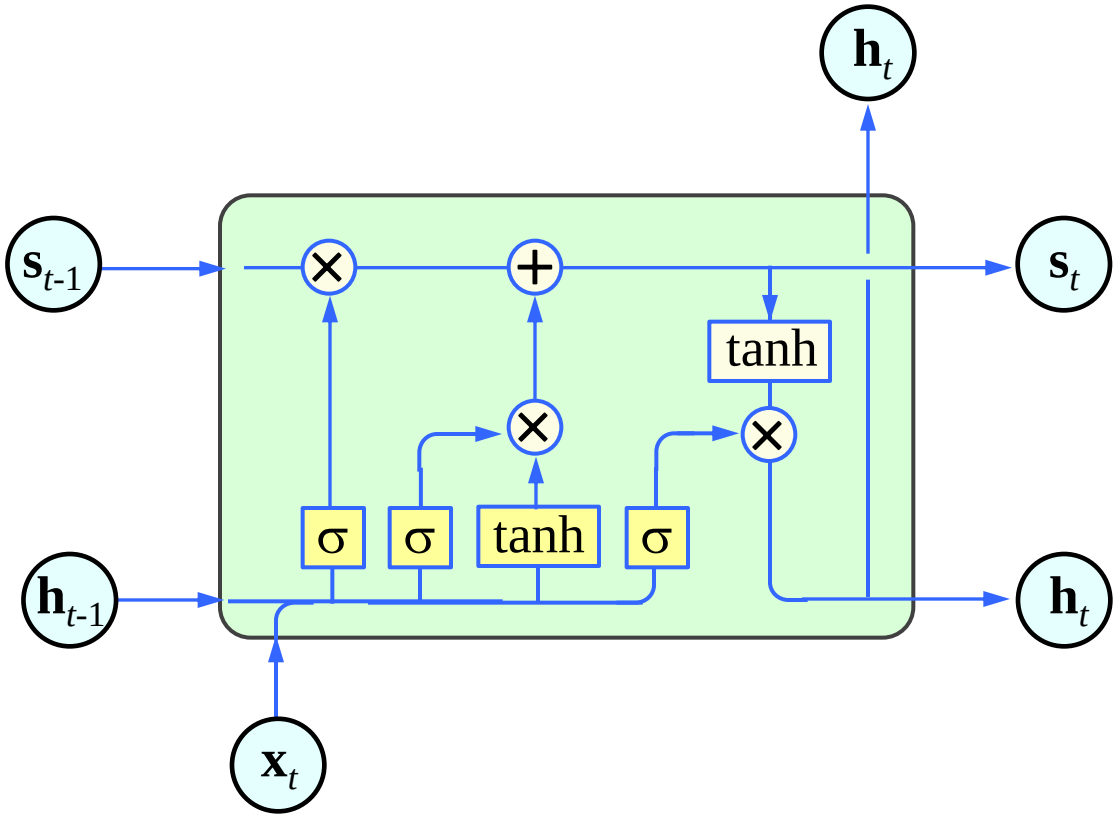


LSTM 中的重复模块包含四个交互的层

：神经网络层　：pointwise operation　：concatenate

：vector transfer　：copy

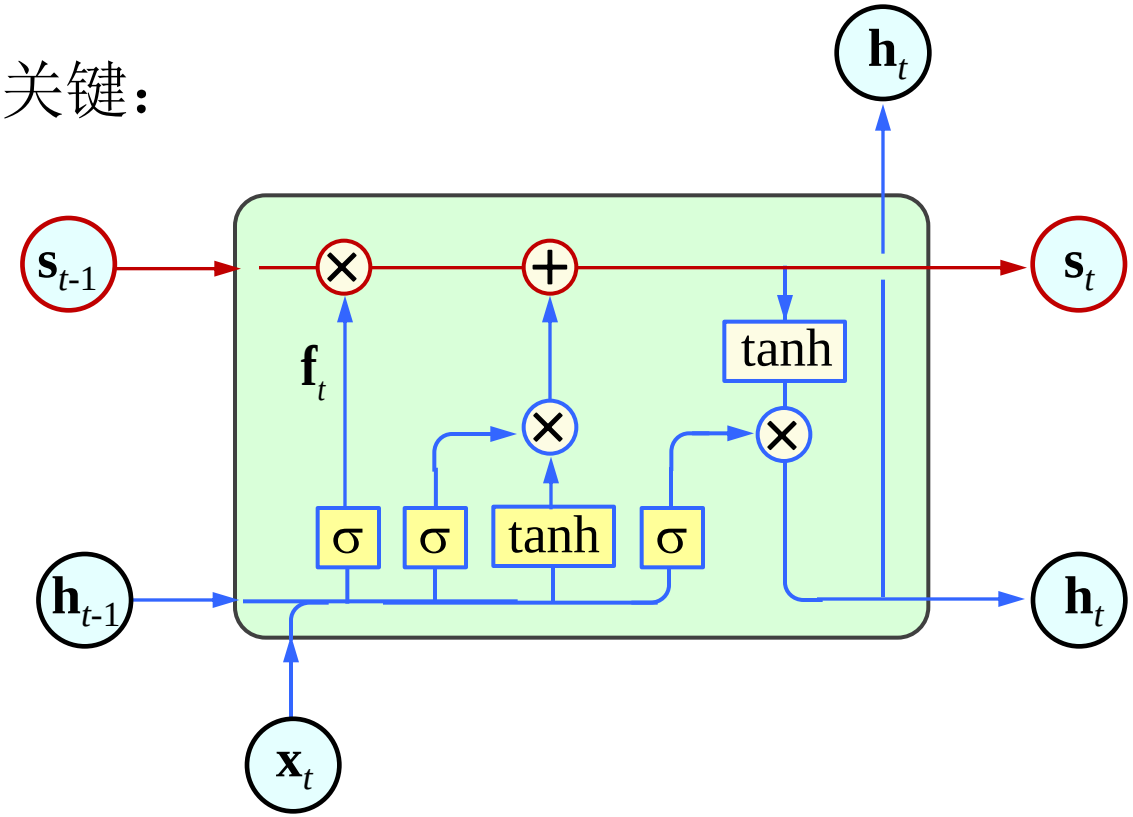
From RNN to LSTM:



LSTM 中的一个时序单元的输入输出

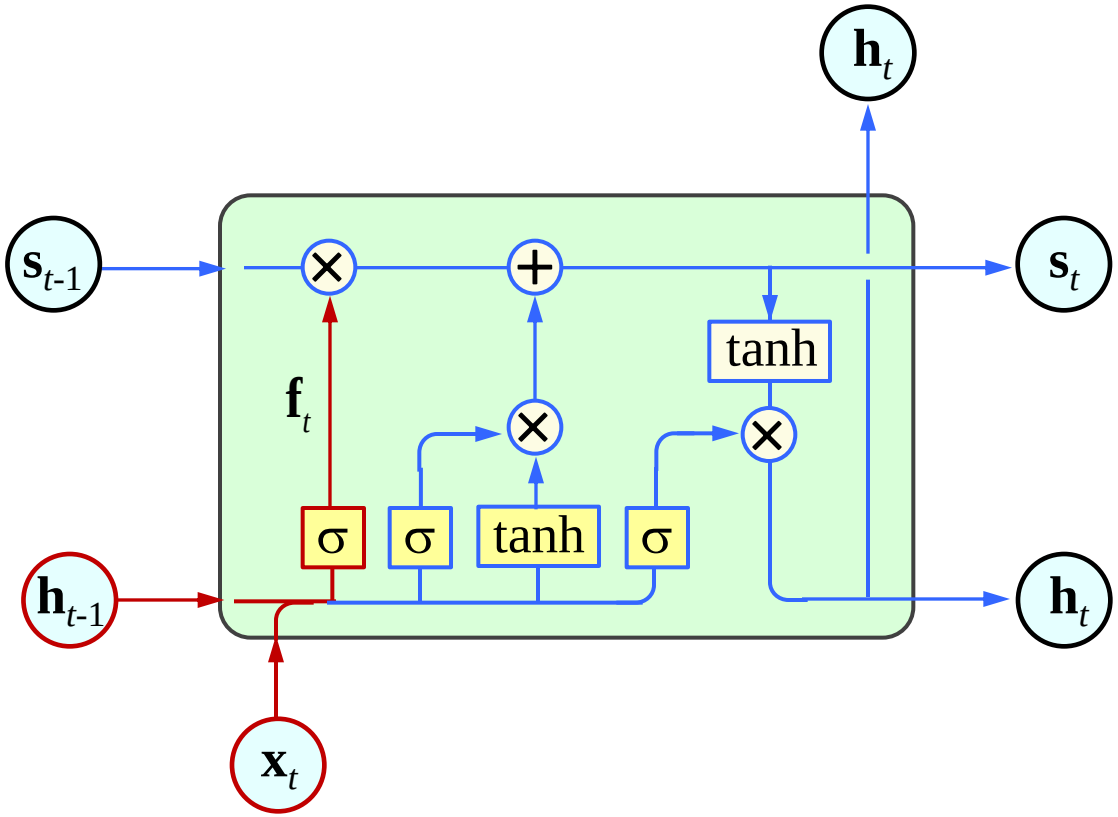
3.3.8 From RNN to LSTM

LSTM的关键：



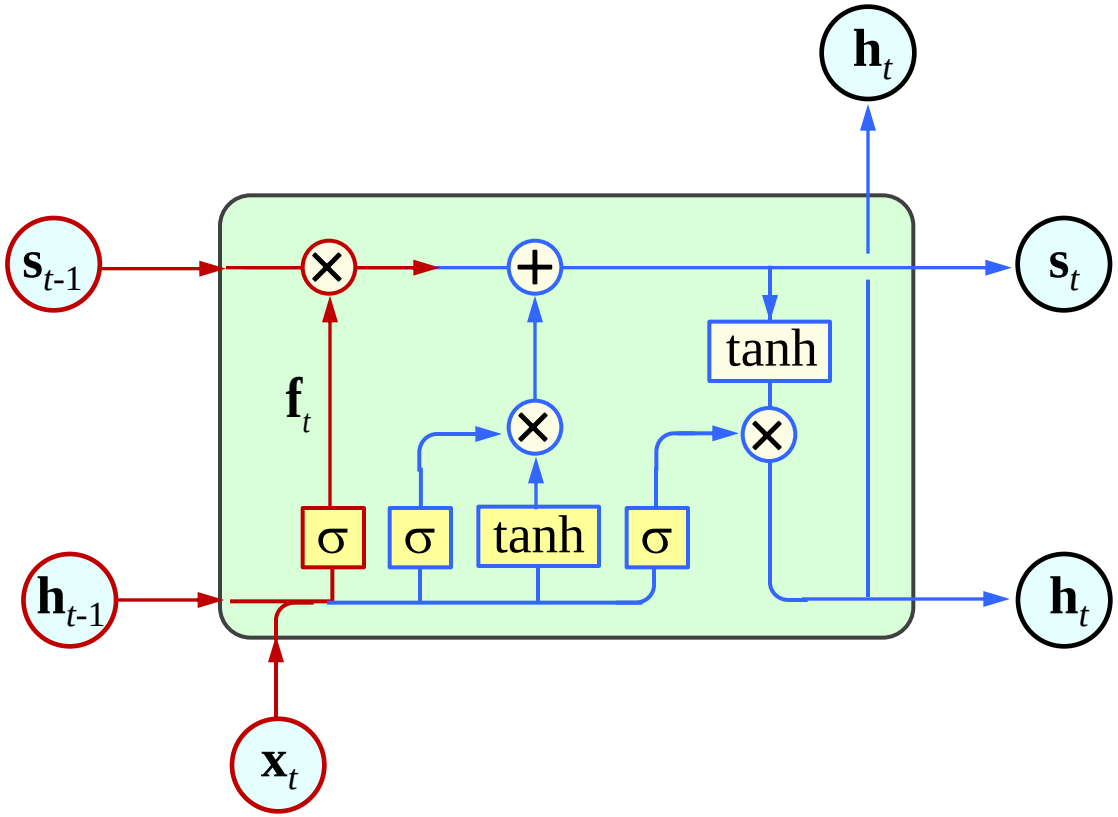
LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易。

3.3.8 From RNN to LSTM



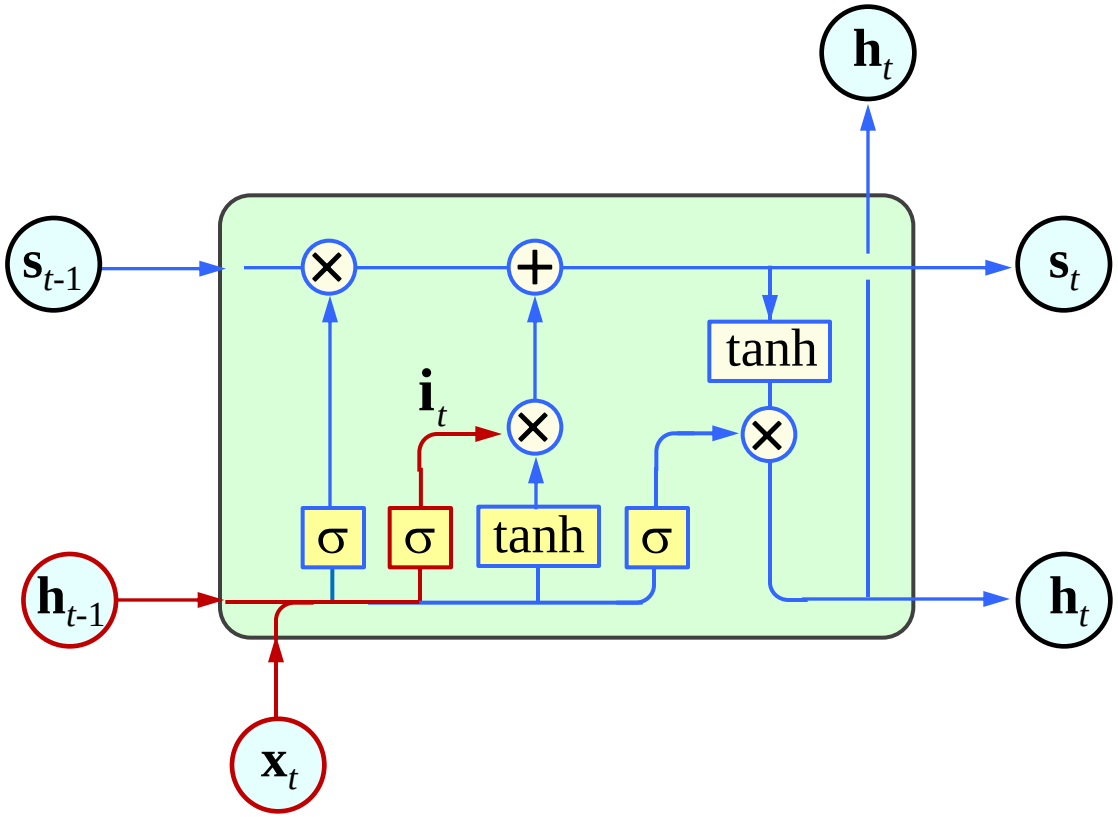
$$\mathbf{f}_t = \text{sigmod}\left(\mathbf{b}_f + \mathbf{U}_f \mathbf{x}_t + \mathbf{W}_f \mathbf{h}_{t-1}\right)$$

From RNN to LSTM:



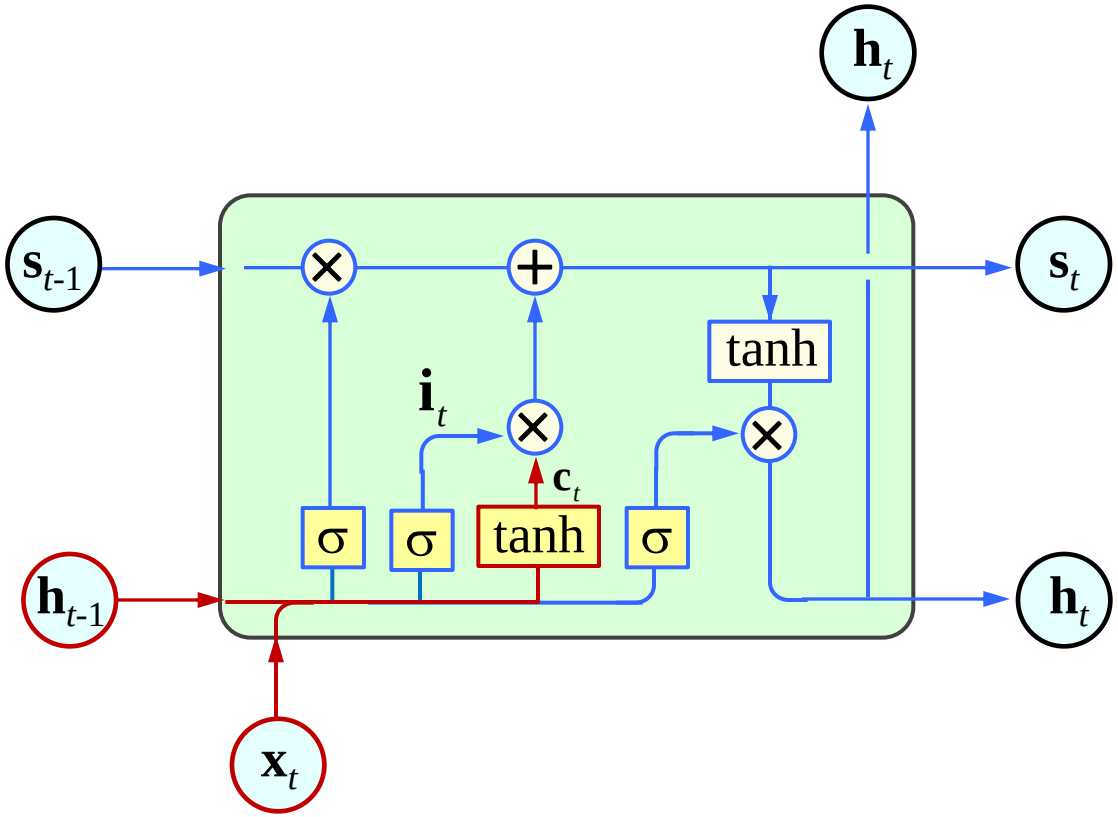
$$\mathbf{f}_t = \text{sigmod}\left(\mathbf{b}_f + \mathbf{U}_f \mathbf{x}_t + \mathbf{W}_f \mathbf{h}_{t-1}\right) \longrightarrow \mathbf{f}_t \otimes \mathbf{s}_{t-1}$$

From RNN to LSTM:



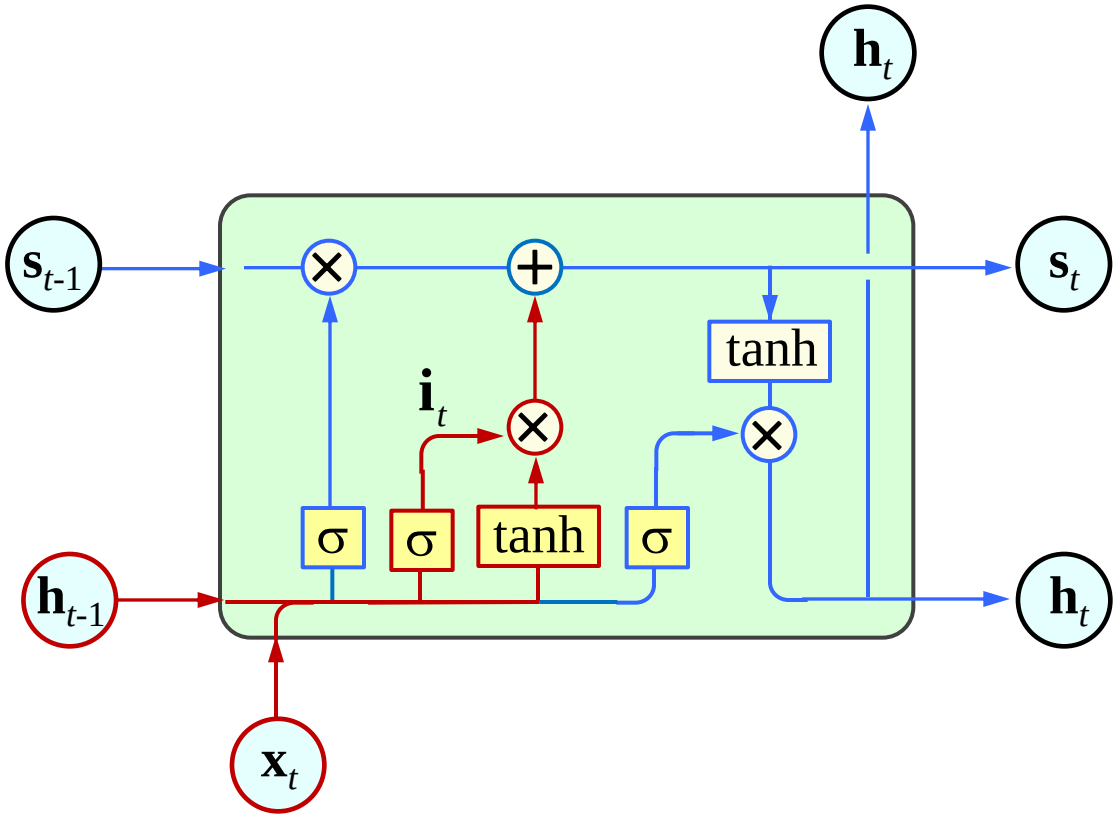
$$\mathbf{i}_t = \text{sigmod}\left(\mathbf{b}_{in} + \mathbf{U}_{in}\mathbf{x}_t + \mathbf{W}_{in}\mathbf{h}_{t-1}\right)$$

From RNN to LSTM:



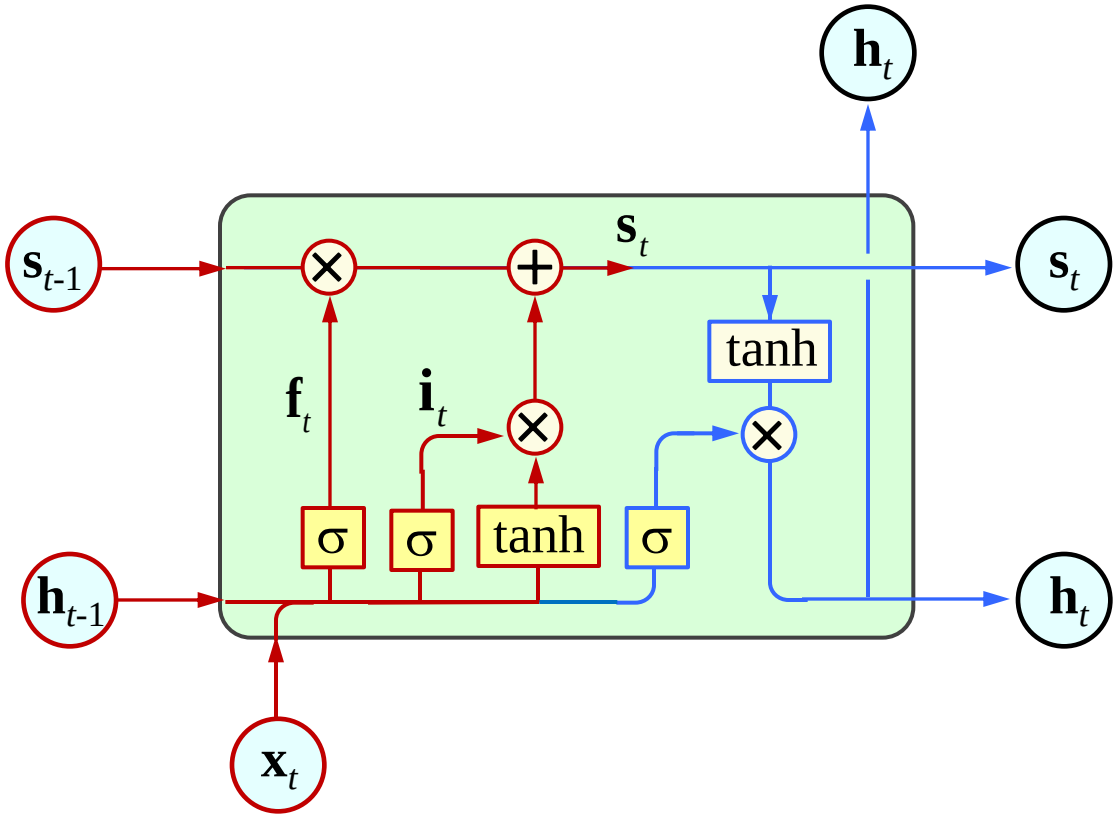
$$\mathbf{c}_t = \tanh\left(\mathbf{b} + \mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}\right)$$

From RNN to LSTM:



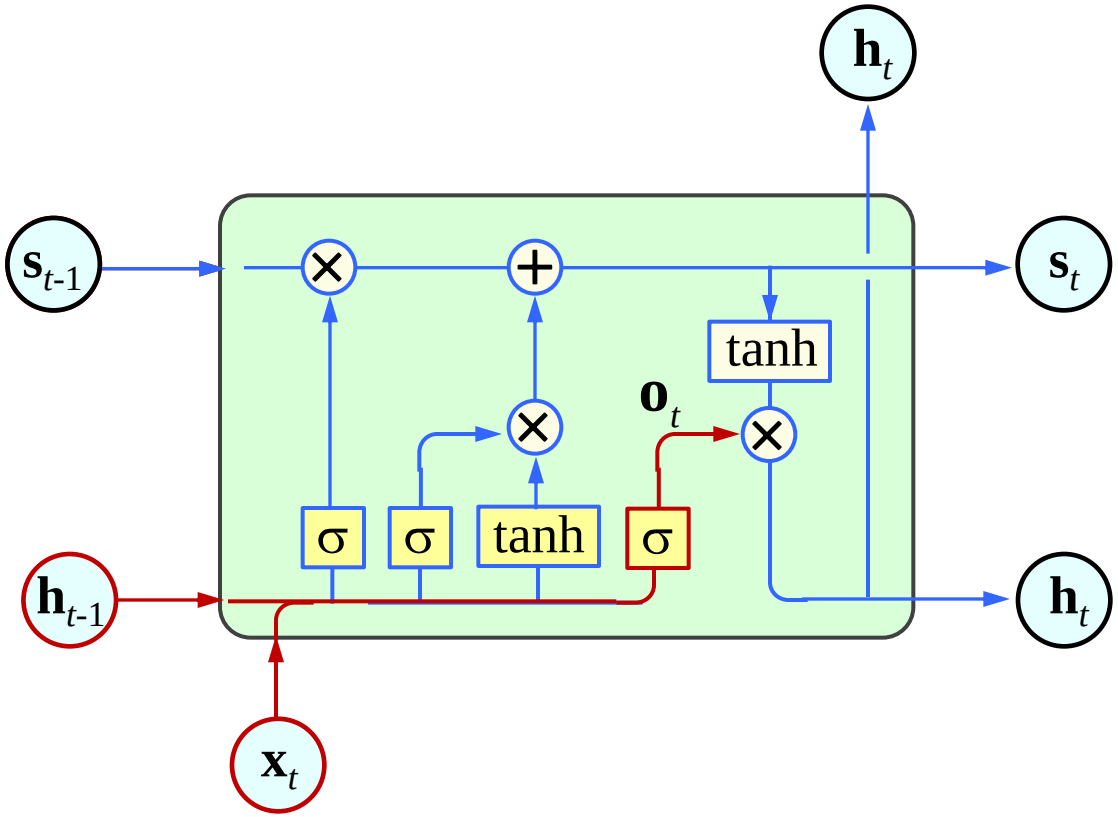
$$\mathbf{i}_t \otimes \mathbf{c}_t$$

From RNN to LSTM:



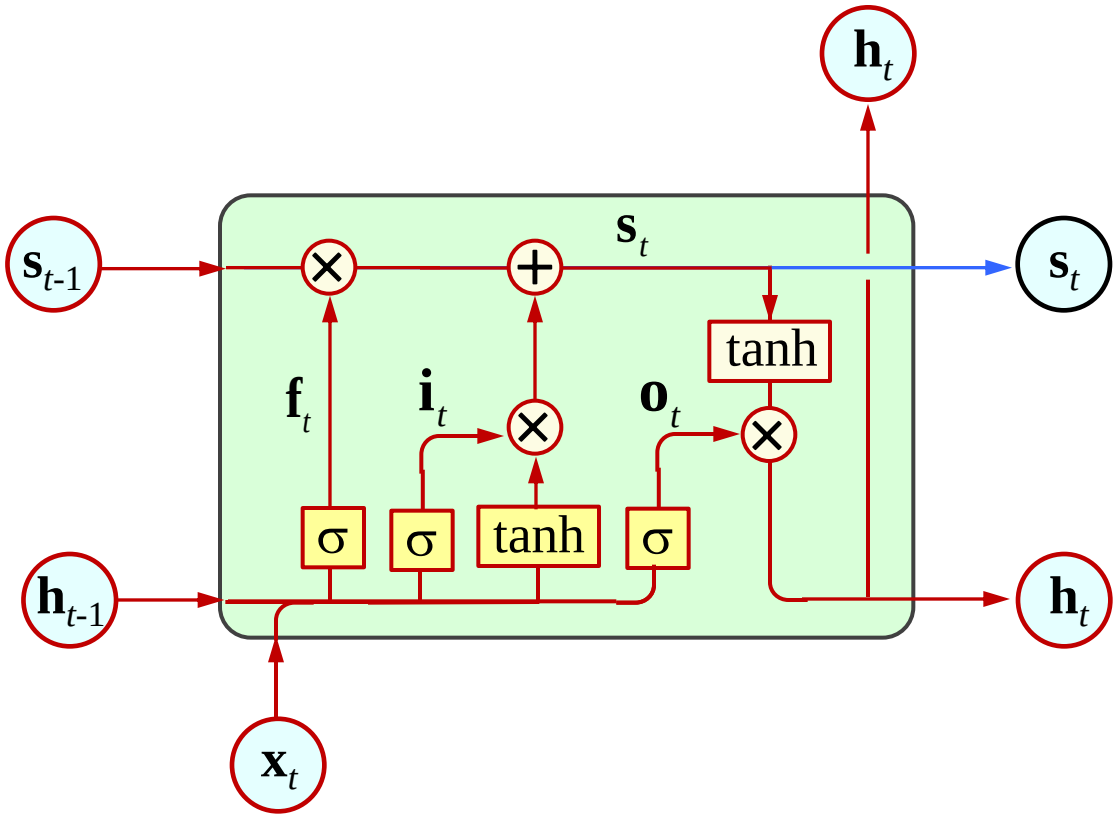
$$\mathbf{s}_t = \mathbf{f}_t \otimes \mathbf{s}_{t-1} + \mathbf{i}_t \otimes \mathbf{c}_t$$

From RNN to LSTM:



$$\mathbf{o}_t = \text{sigmod}\left(\mathbf{b}_o + \mathbf{U}_o\mathbf{x}_t + \mathbf{W}_o\mathbf{h}_{t-1}\right)$$

From RNN to LSTM:



$$\mathbf{o}_t = \text{sigmod}\left(\mathbf{b}_o + \mathbf{U}_o\mathbf{x}_t + \mathbf{W}_o\mathbf{h}_{t-1}\right)$$

- 网络训练
  - 估计如下矩阵或向量

    $\mathbf{U}_{in}$，$\mathbf{W}_{in}$，$\mathbf{b}_{in}$，$\mathbf{U}_{o}$，$\mathbf{W}_{o}$，$\mathbf{b}_{o}$，$\mathbf{U}_{f}$，$\mathbf{W}_{f}$，$\mathbf{b}_{f}$，$\mathbf{U}$，$\mathbf{W}$，$\mathbf{V}$，$\mathbf{c}$

  - 基于两点：
    - 基于信息流动的方式，采用反向传播算法
    - 计算目标函数、隐状态(t)对各变量的偏导数
  - 并不是说LSTM完全解决了梯度消失或扩散的问题，只是有所缓和。仍可考虑如下技术：
    - 充分利用二阶梯度的信息（但通常并不鼓励）
    - 更好的初始化
    - 动量机制
    - 对梯度的模或者元素作一些裁剪
    - 正则化—鼓励信息流动

Thank All of You!

(Questions?)

向世明

smxiang@nlpr.ia.ac.cn

http://www.escience.cn/people/smxiang

时空数据分析与学习课题组(STDAL)

中科院自动化研究所· 模式识别国家重点实验室