第9章 神经网络

Neural Networks

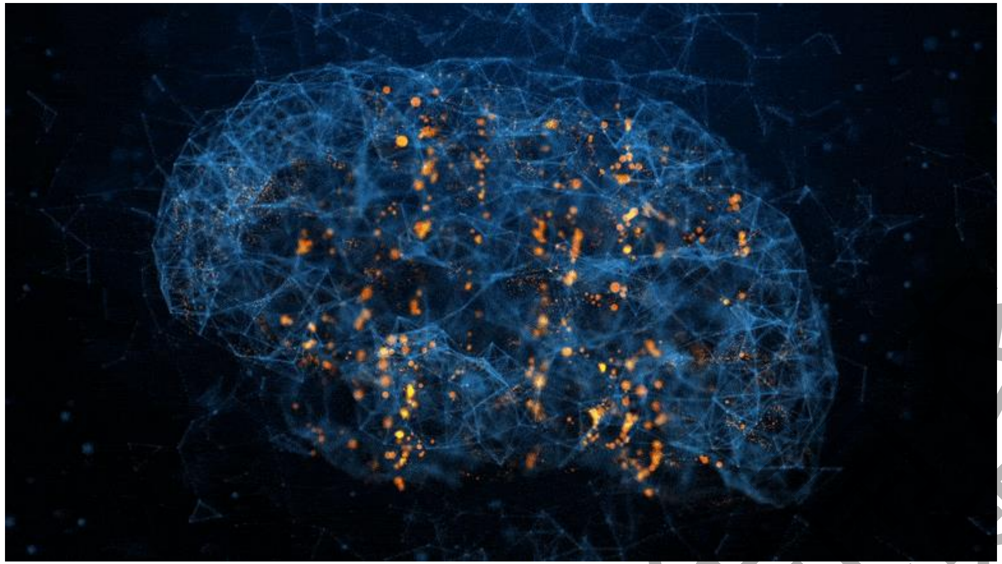
电子工程学院、人工智能学院
college of Electronic Engineering , college of Artificial Intelligence

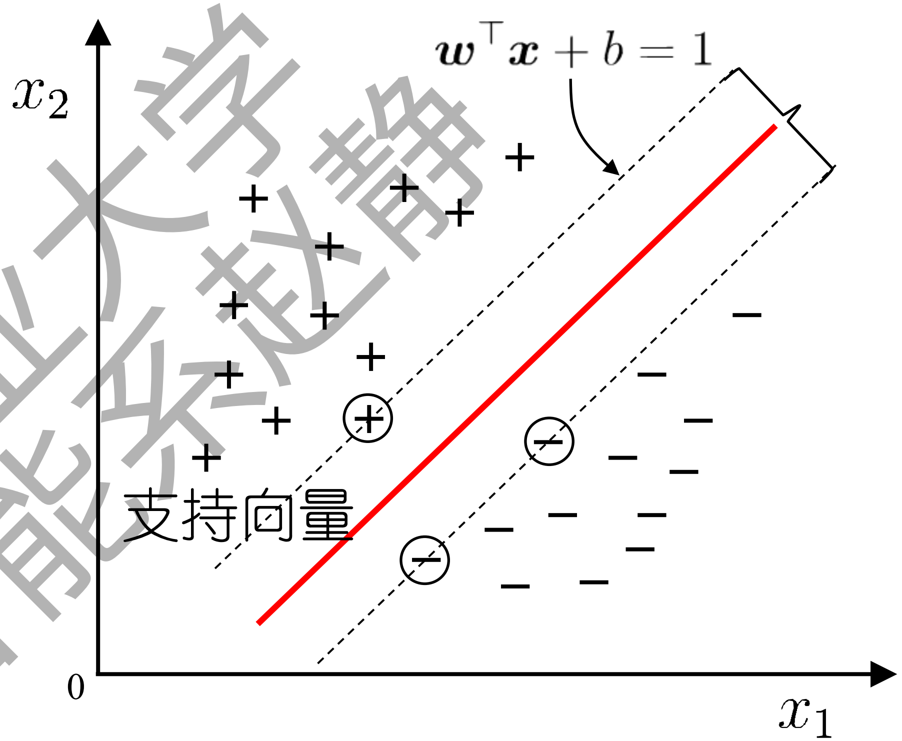
深度学习发展史

- 1957：感知机(Perceptron，线性模型)
- 1980s：多层感知器 (Multi-Layer Perceptron, MLP)

  与现在的DNN没有显著差异
- 1986: 反向传播（Backpropagation）
- 1994：LeNet5
- 2006：深度置信网络(Deep Belief Nets)
- 2010：使用GPU加速端到端BP神经网络
- 2011：在语音识别领域开始流行
- 2012：AlexNet 在ImageNet图像分类大赛上完胜

- 神经元结构
- 前馈全连接神经网络DNN
- 卷积神经网络CNN



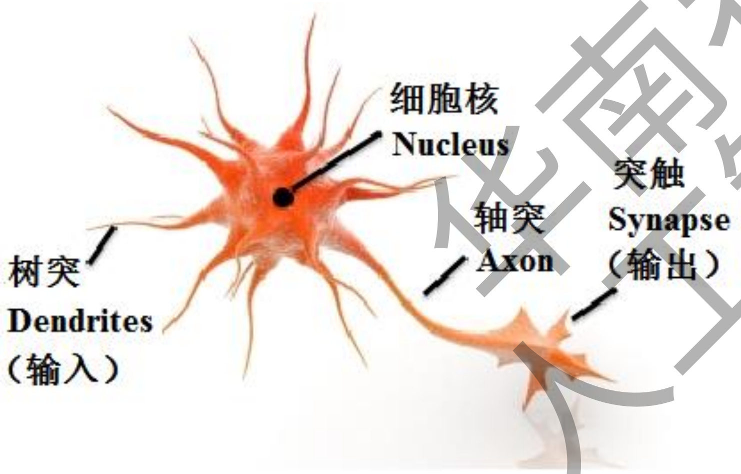


仿生学派与数理学派

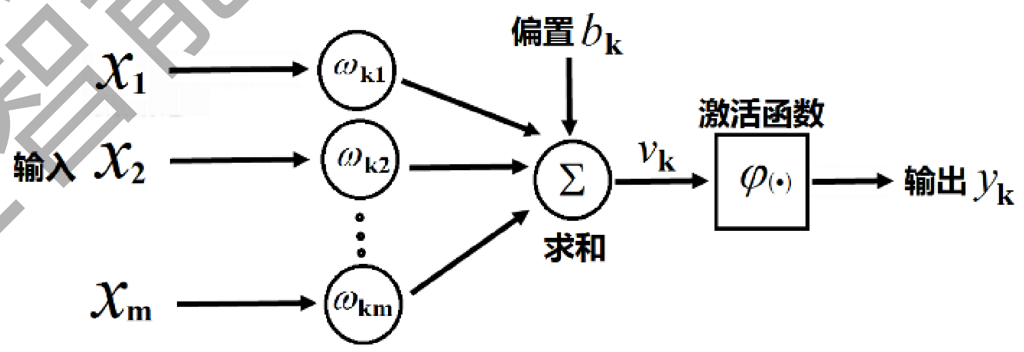
1. 神经元结构

➢ MP模型

1943年，心理学家W. S. McCulloch和数理逻辑学家W. Pitts基于神经元的生理特征，建立了单个神经元的数学模型（MP模型）。

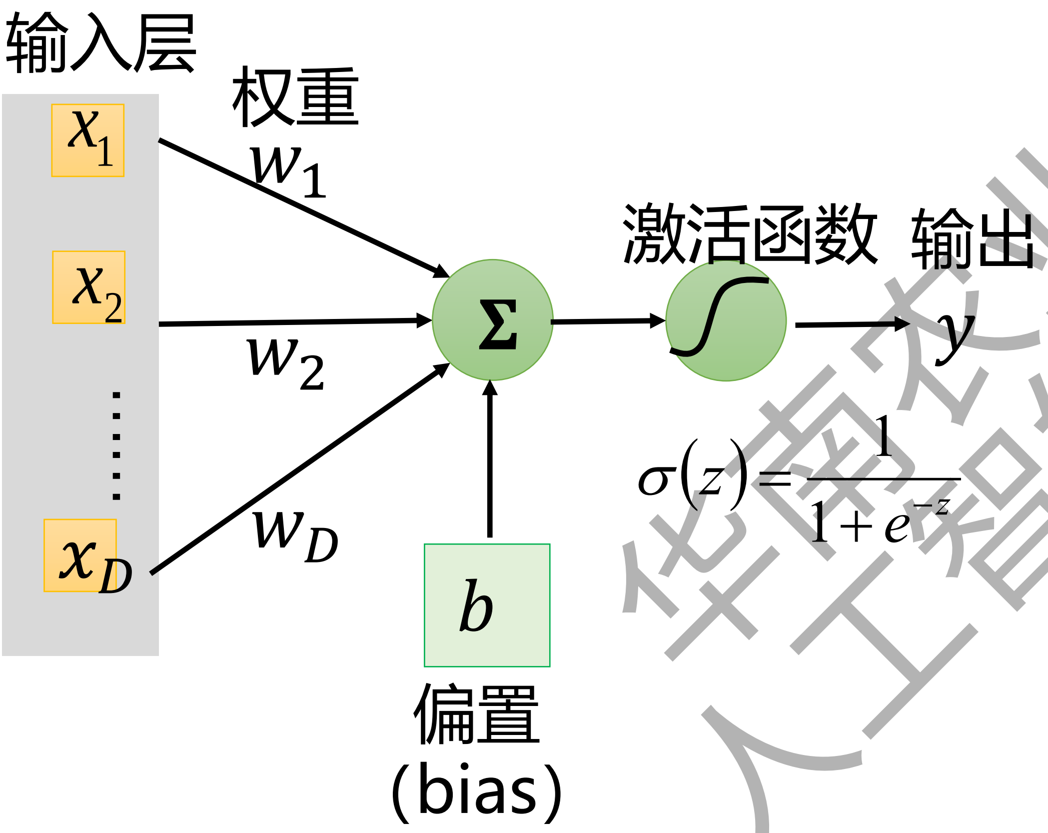


神经元生理结构示意图



神经元的数学模型示意图

$$y_k = \varphi\left(\sum_{\mathrm{j}} w_j x_j + b\right) = \varphi\left(W^T X + b\right)$$



神经元的数学模型示意图

$$y = f(x_1, x_2, \cdots x_m)$$

$$\approx f(0\ ,\ 0, \cdots 0) + \sum_{i=1}^{m}\left[\frac{\partial f}{\partial x_i}\Big|(0\ ,\ 0, \cdots 0)\right] x_i + \ldots$$

$$= \sum_{i=1}^{m} \omega_i x_i + b$$

复杂函数的一阶泰勒近似

➢ 感知机(perceptron)

1957年，Frank Rosenblatt从纯数学的度重新考察MP模型，指出能够从一些输入输出对(X,y)中通过学习算法自动的获得权重 W 和b 。

Frank Rosenblatt
1928-1971

问题：给定一些输入输出对 (X,y)，其中 $y=\pm1$，求一个函数，使：$f(X)=y$

感知机算法：设定 $f(X)=sign(W^TX+b)$，从一堆输入输出中自动学习，获得W和 b 。

感知机算法（Perceptron Algorithm）：

（1）随机选择W和 b 。

（2）取一个训练样本 (X, y)

（i）若 $W^T X + b > 0$且 $y = -1$,则：

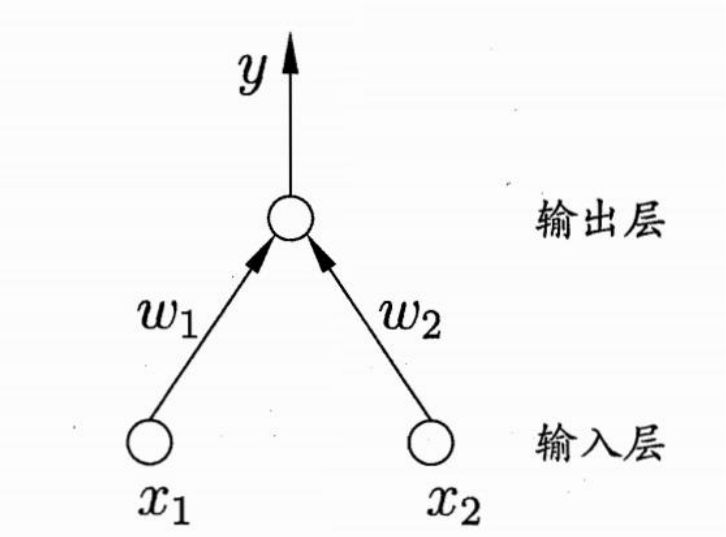
$$W = W - X \quad b = b - 1$$

（ii）若 $W^T X + b < 0$且 $y = +1$,则：

$$W = W + X \quad b = b + 1$$

（3）再取另一个(X, y)，回到（2）

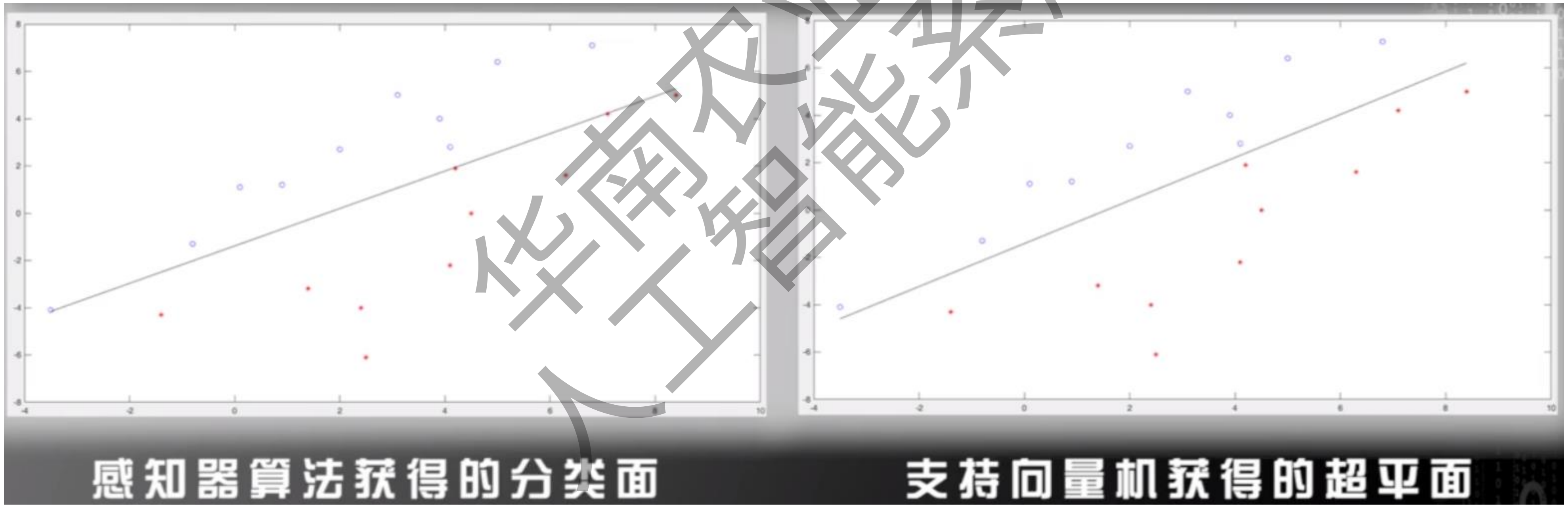
（4）终止条件：直到所有输入输出对 都不满足（2）中(i)和(ii)之一，退出循环。



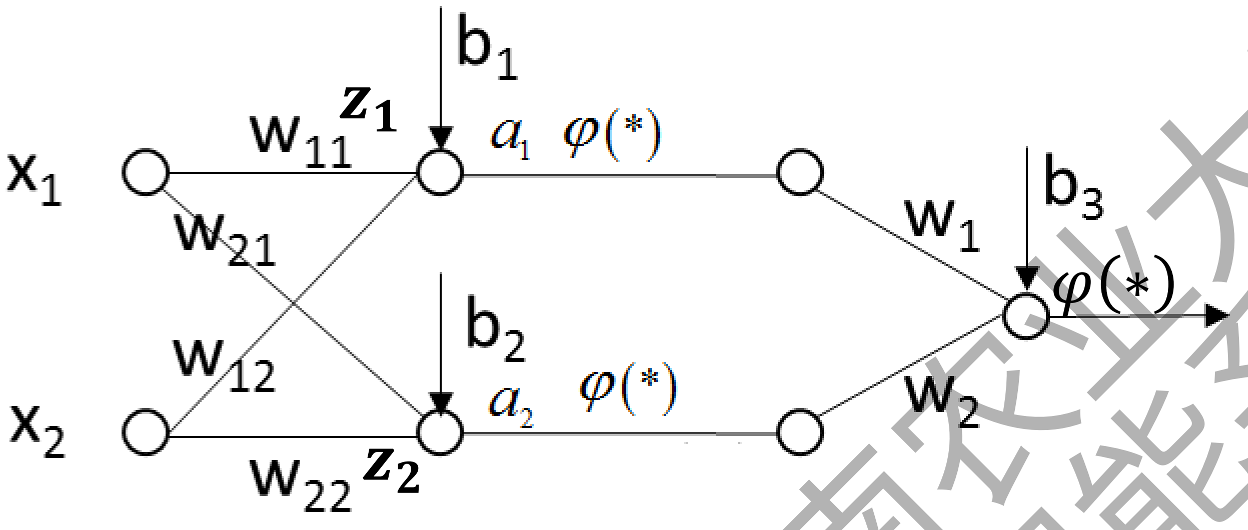
感知机结构

感知机算法的意义和局限

- 单层感知机只能处理线性问题，无法处理非线性问题
- 单层感知机处理线性问题，一般情况没有SVM效果好



➢ 两层神经网络



$$z_1 = w_{11}x_1 + w_{12}x_2 + b_1$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + b_2$$

$$a_1 = \varphi(z_1)$$

$$a_2 = \varphi(z_2)$$

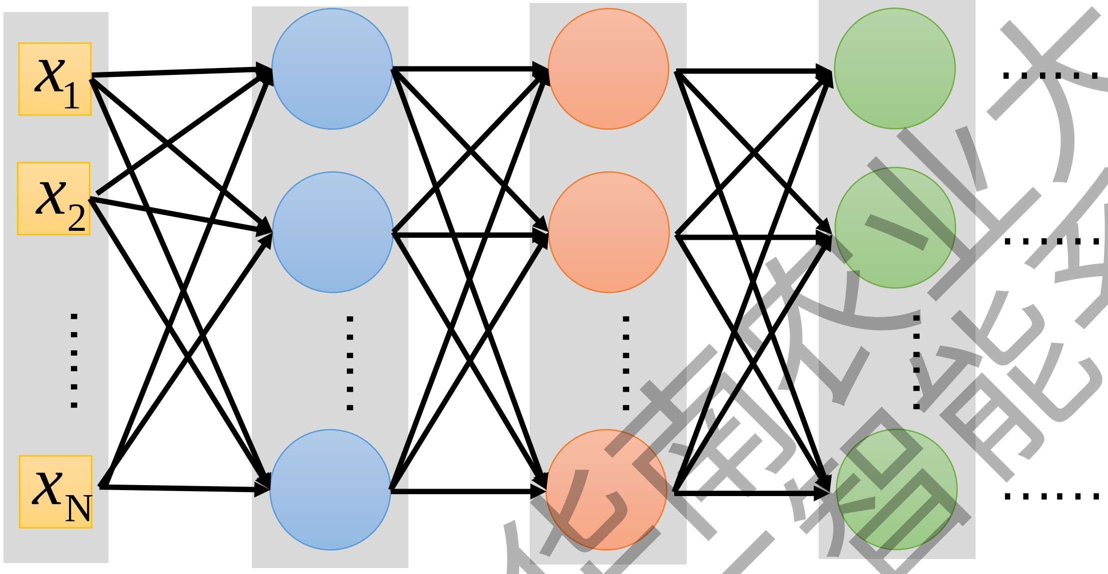
$$\hat{y} = (w_1a_1 + w_2a_2 + b_3)$$

（注意：其中$\varphi(*)$为非线性函数）

输入 (X,Y)，其中 $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$，$Y$ 是标签值（label），即我们希望改变 w 和 b，使得标签值 Y 与实际的网络输出值 $\hat{y}$ 尽量接近。

定义目标函数： $Minimize: L(\omega, b) = \frac{1}{2}(Y - \hat{y})^2$

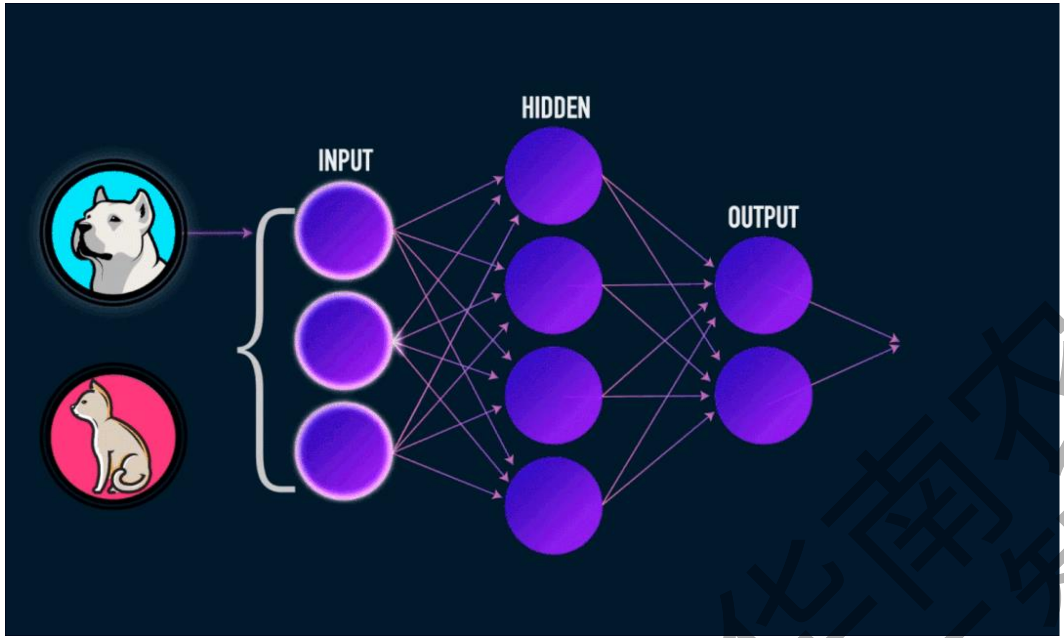
➢ 多层神经网络

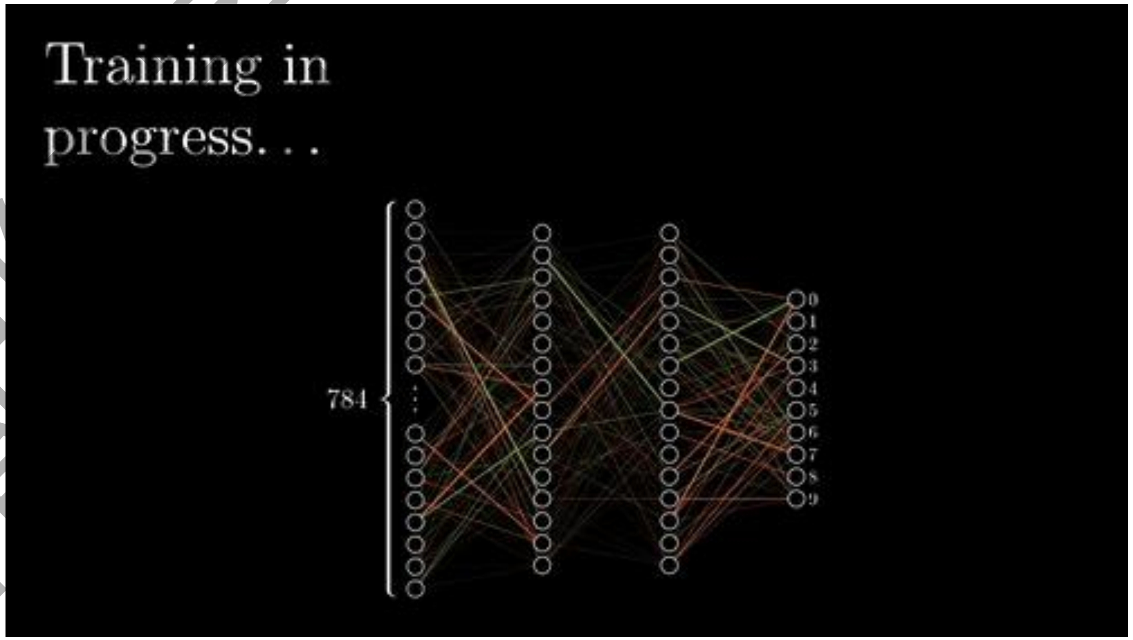


设计神经网络的结构

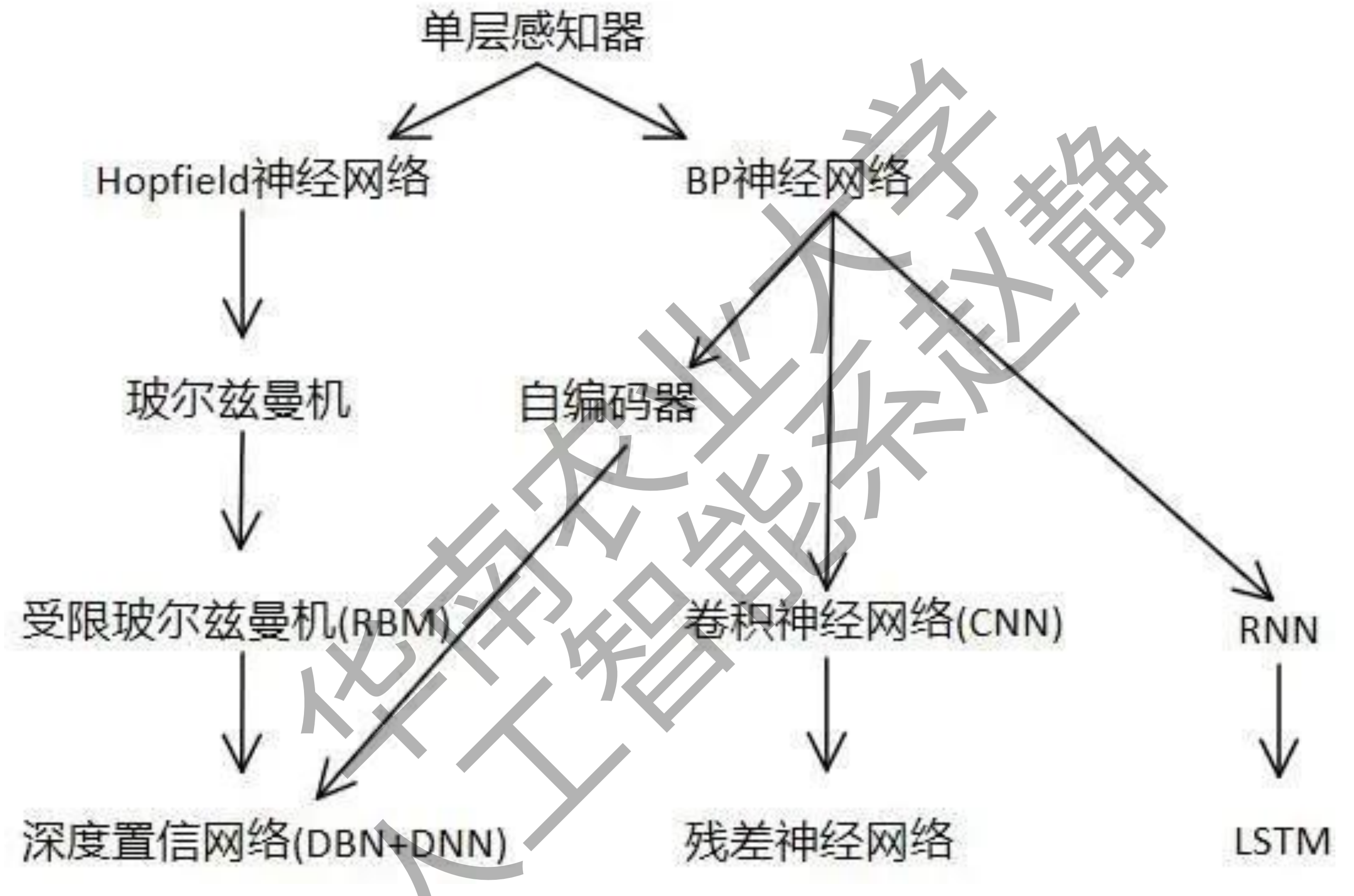
将训练数据输入网络中

估计网络代求参数





以神经网络为基础的深度学习网络



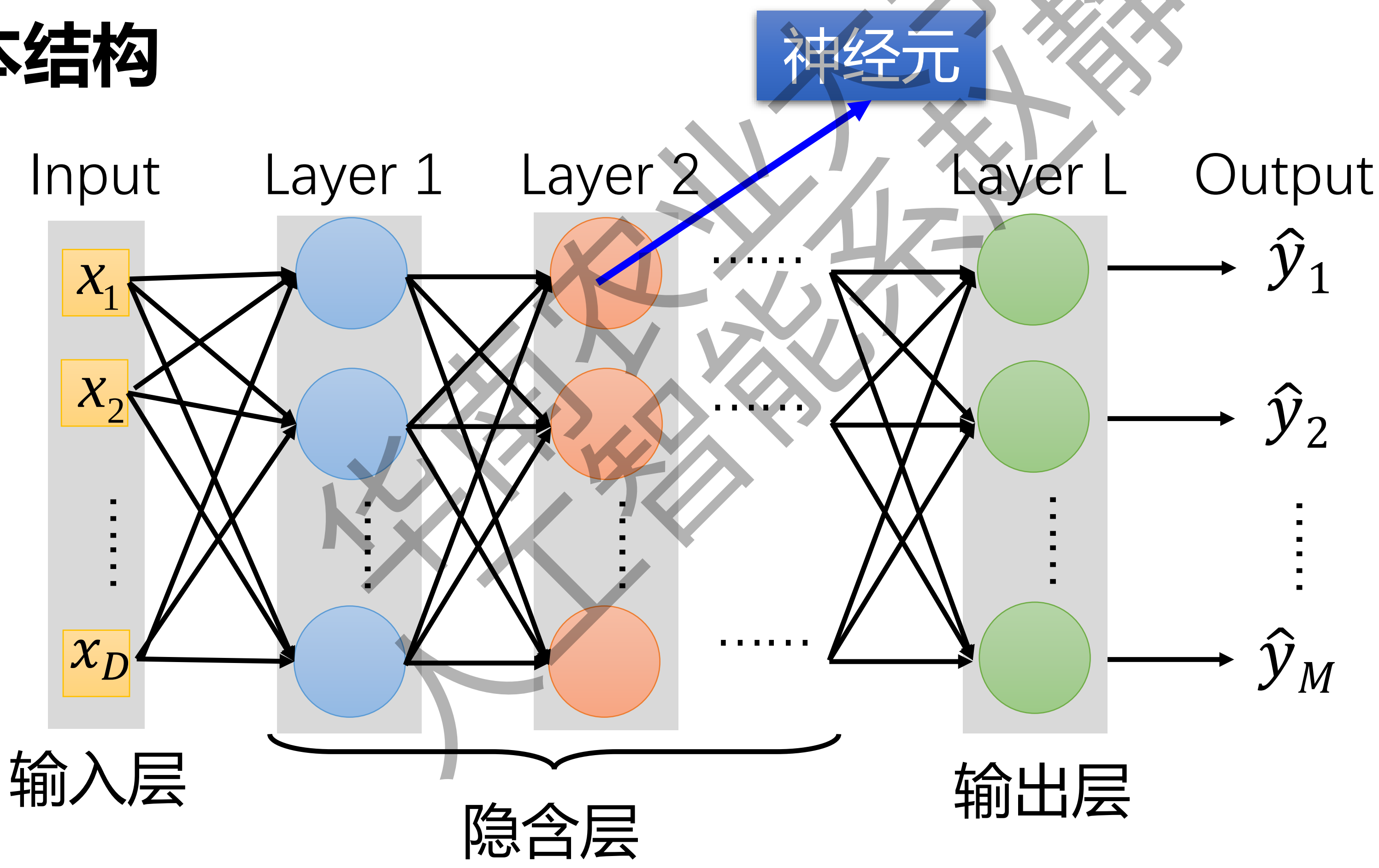
单层感知器
Hopfield神经网络
BP神经网络
玻尔兹曼机
自编码器
受限玻尔兹曼机(RBM)
卷积神经网络(CNN)
RNN
深度置信网络(DBN+DNN)
残差神经网络
LSTM

- 神经元结构
- 前馈全连接神经网络DNN
  - 网络结构
  - 模型训练：反向传播
- 卷积神经网络

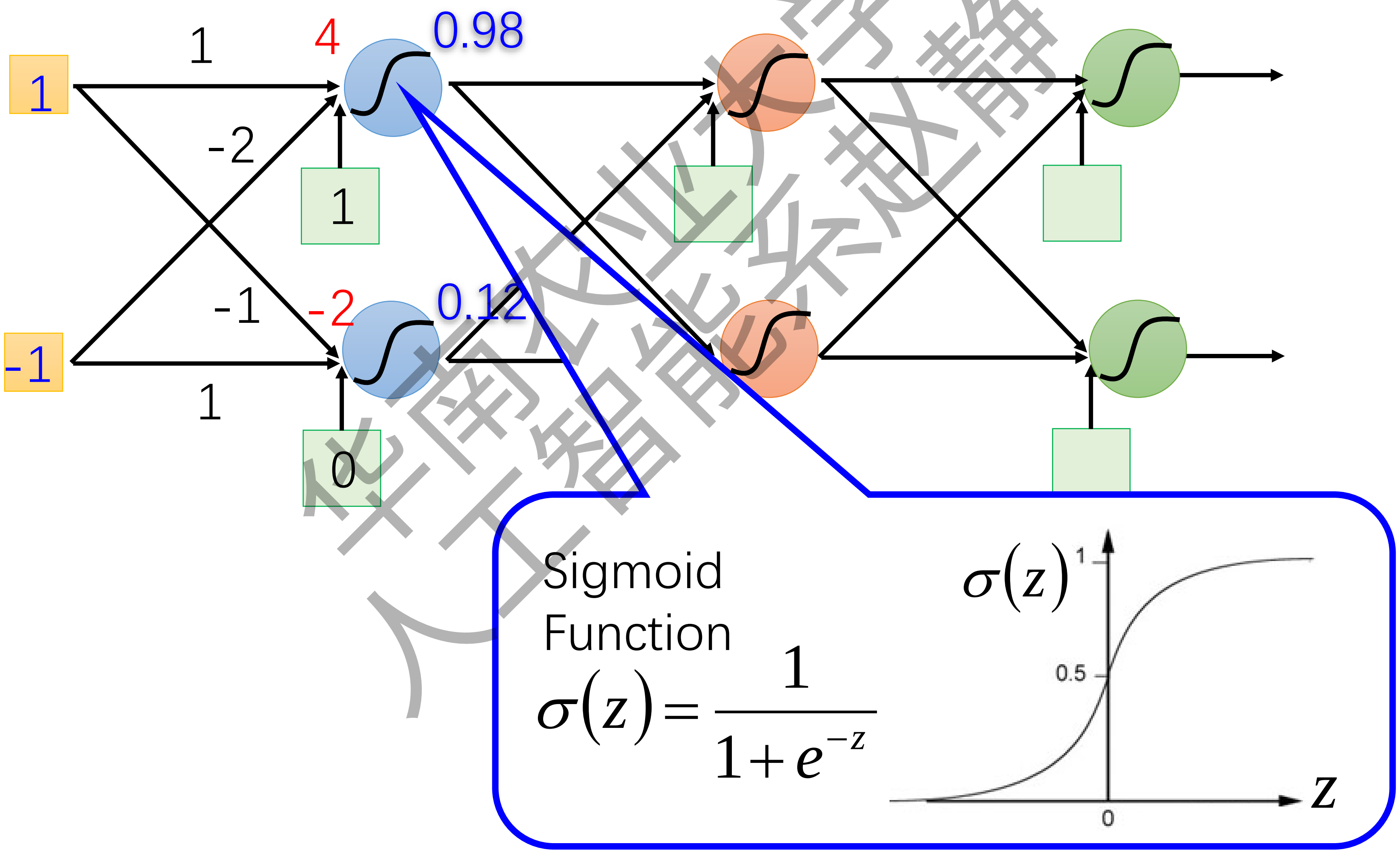
2. 前馈全连接神经网络

➢ 网络结构

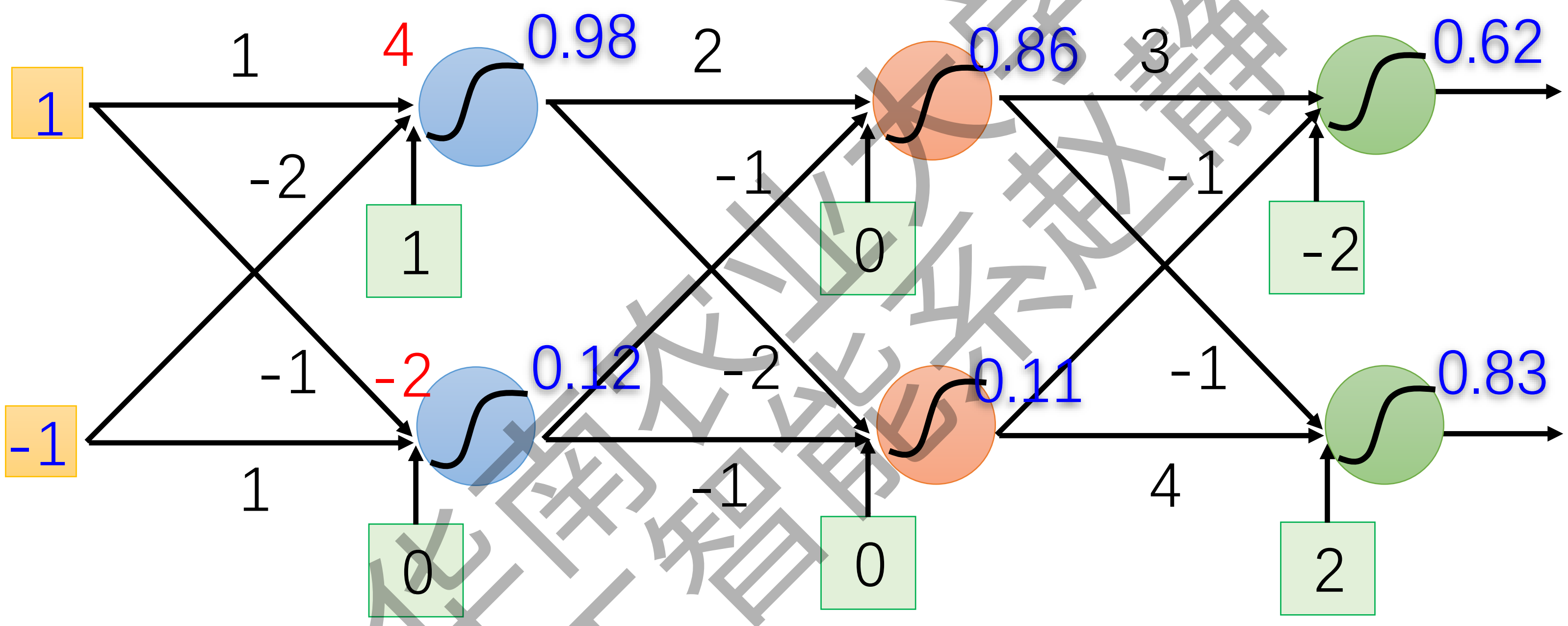
✓ 基本结构

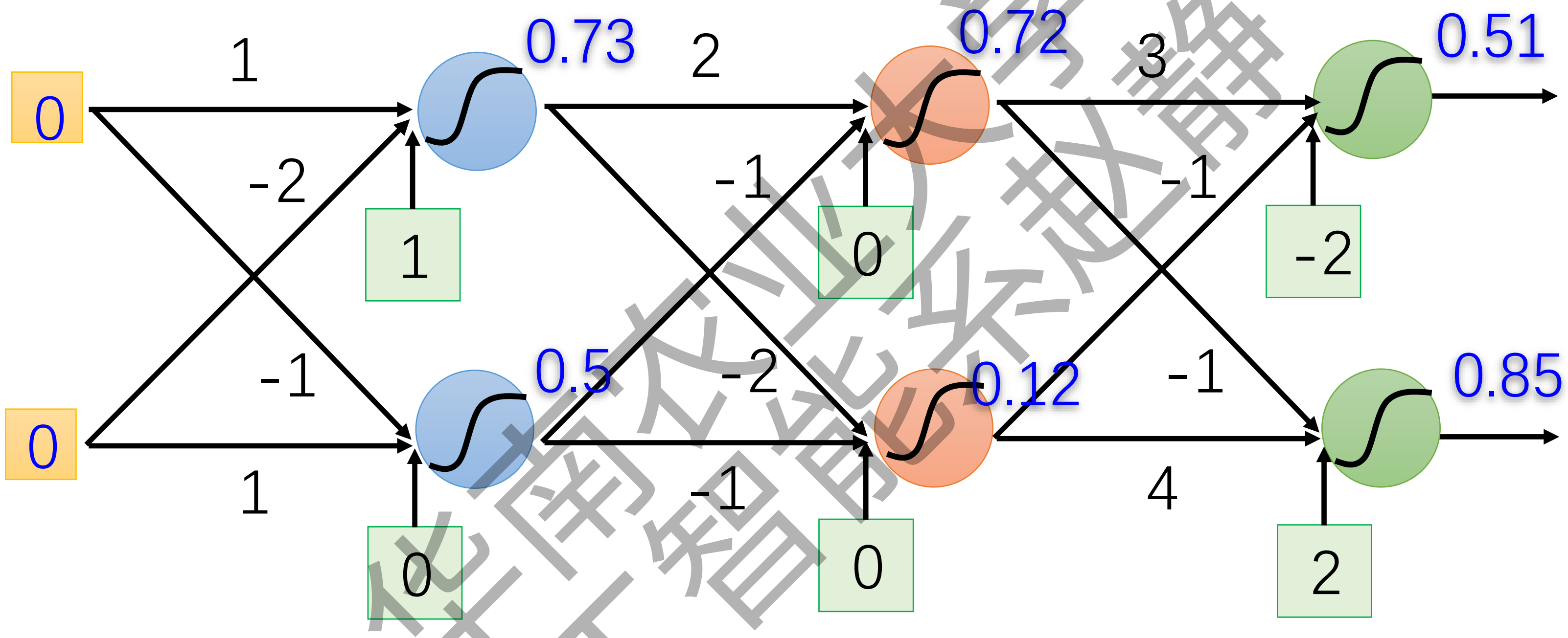


- **前馈机制**



1
-1
1
-2
-1
1
4
-2
1
0
0.98
0.12
Sigmoid
Function
$\sigma(z)=\frac{1}{1+e^{-z}}$
$\sigma(z)$
1
0.5
0
z

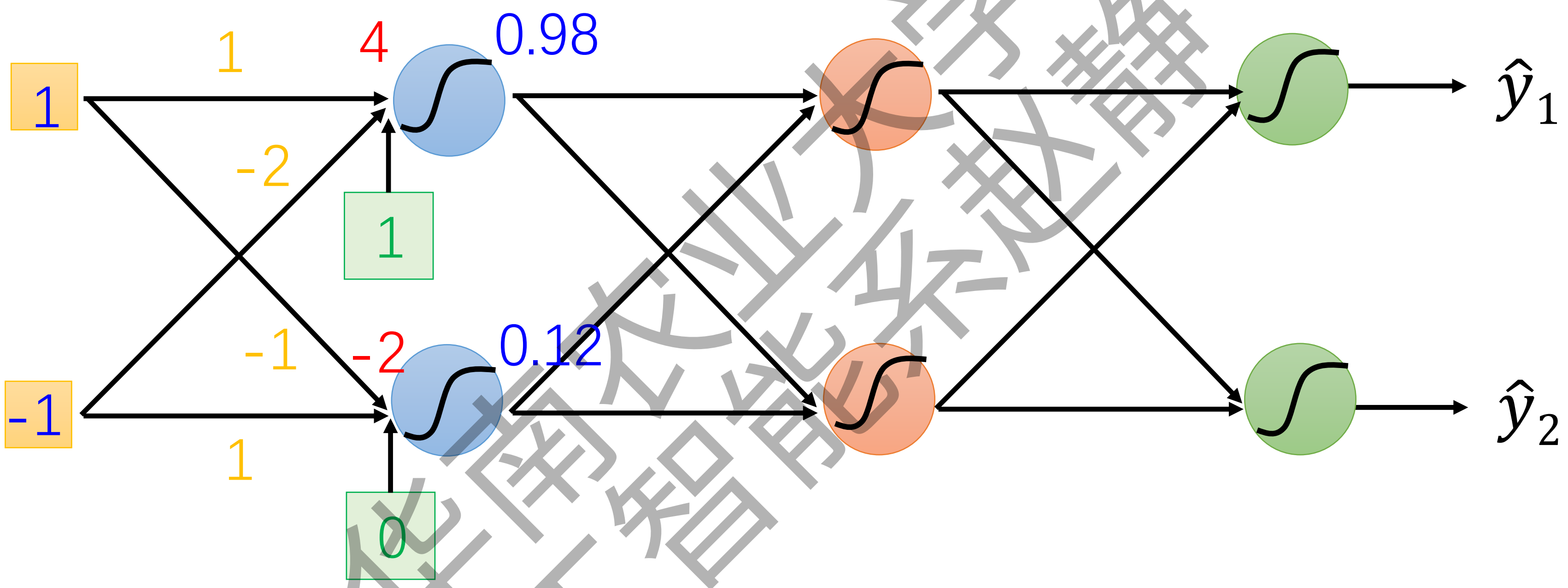

1
-1
1
-2
-1
1
4
-2
1
0
0.98
0.12
2
-1
-2
-1
0
0
0.86
0.11
3
-1
-1
4
-2
2
0.62
0.83



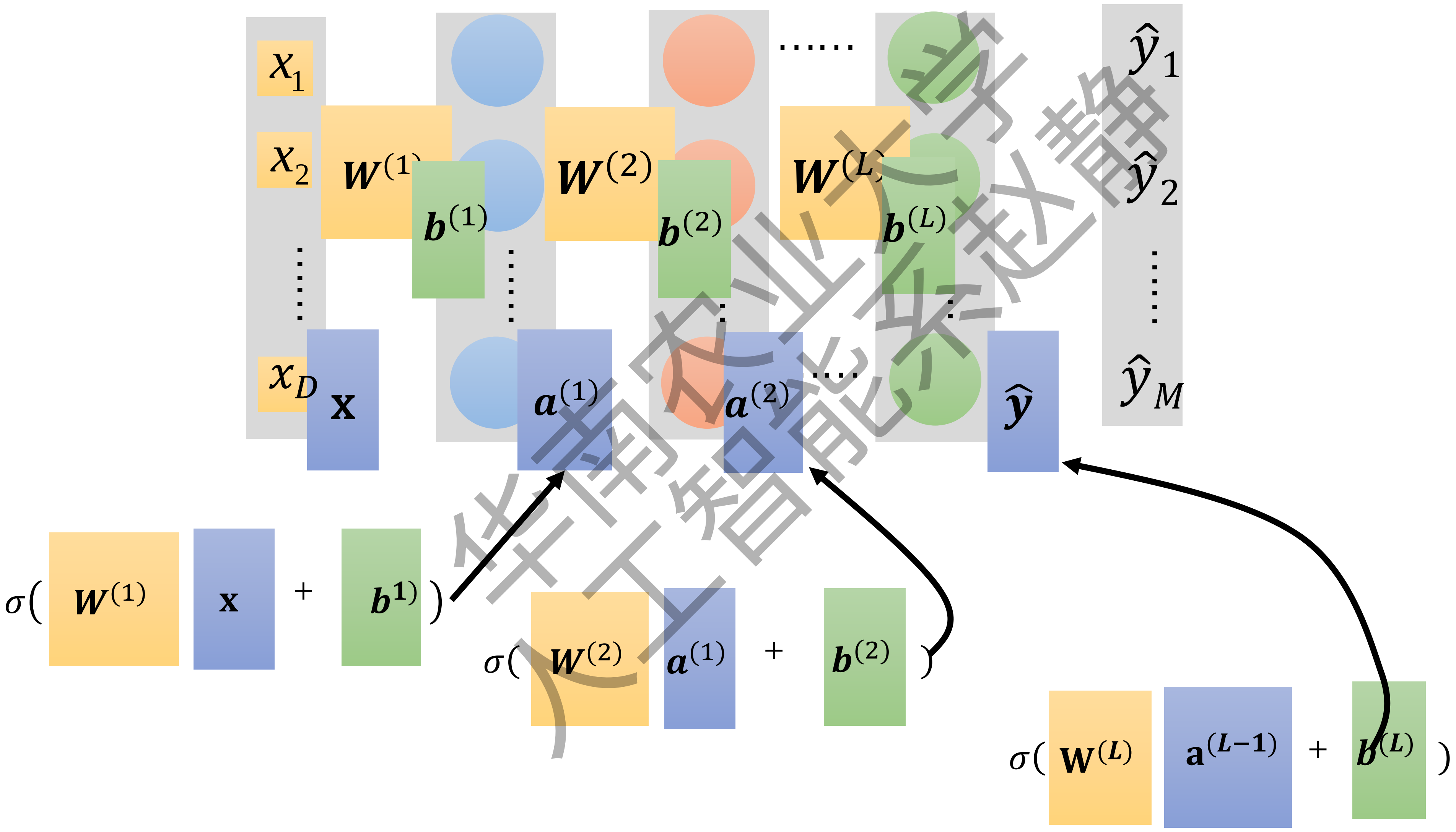
$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

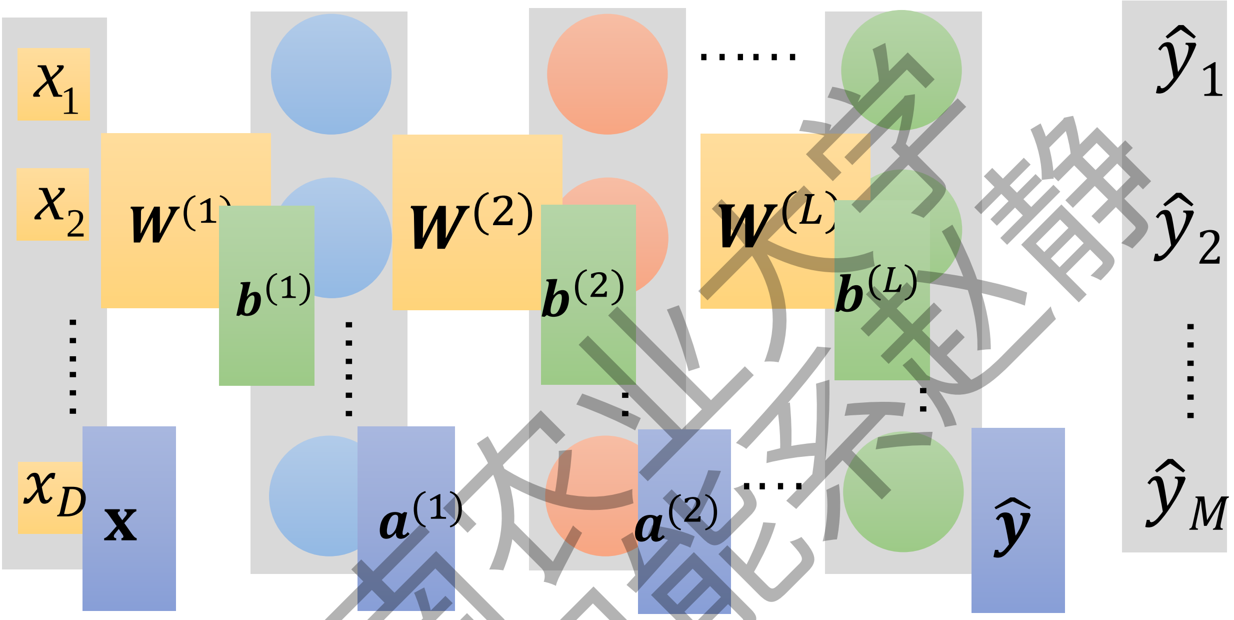
$$f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

- **矩阵操作**



$$\sigma\Big(\underbrace{\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}}_{\begin{bmatrix}4\\ -2\end{bmatrix}}\Big)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$


x_1
x_2
x_D
$\mathbf{x}$
$W^{(1)}$
$b^{(1)}$
$W^{(2)}$
$b^{(2)}$
$W^{(L)}$
$b^{(L)}$
$a^{(1)}$
$a^{(2)}$
$\hat{\mathbf{y}}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_M$
$\sigma(\ W^{(1)}\ \mathbf{x}\ +\ b^{1)}\ )$
$\sigma(\ W^{(2)}\ a^{(1)}\ +\ b^{(2)}\ )$
$\sigma(\ \mathbf{W}^{(L)}\ \mathbf{a}^{(L-1)}\ +\ b^{(L)}\ )$

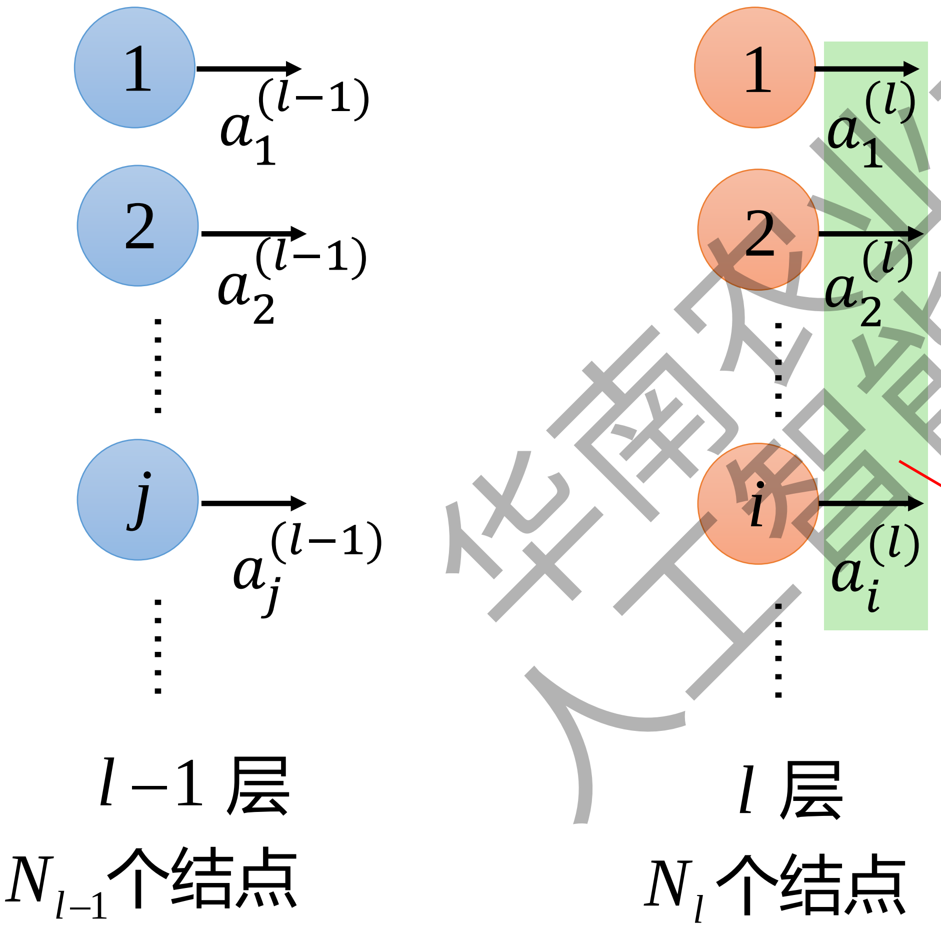


$$\mathbf{y} = f(\mathbf{x})$$

可采用并行计算加快矩阵操作

$$= \sigma(\boldsymbol{W}^{(L)} \cdots \sigma(\boldsymbol{W}^{(2)}\, \sigma(\boldsymbol{W}^{(1)}\, \mathrm{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)}) \cdots + \boldsymbol{b}^{(L)})$$

- 符号表示



神经元的输出:

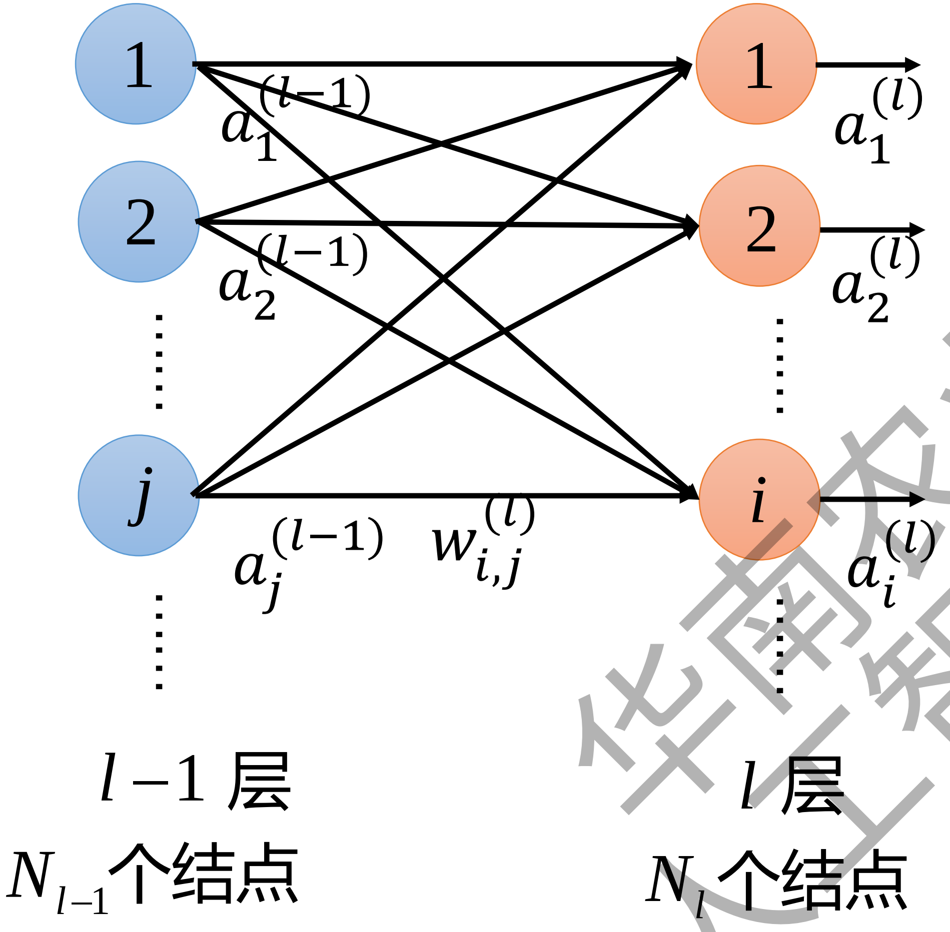
$$a_i^{(l)}$$

层 l

神经元i

一层的输出:

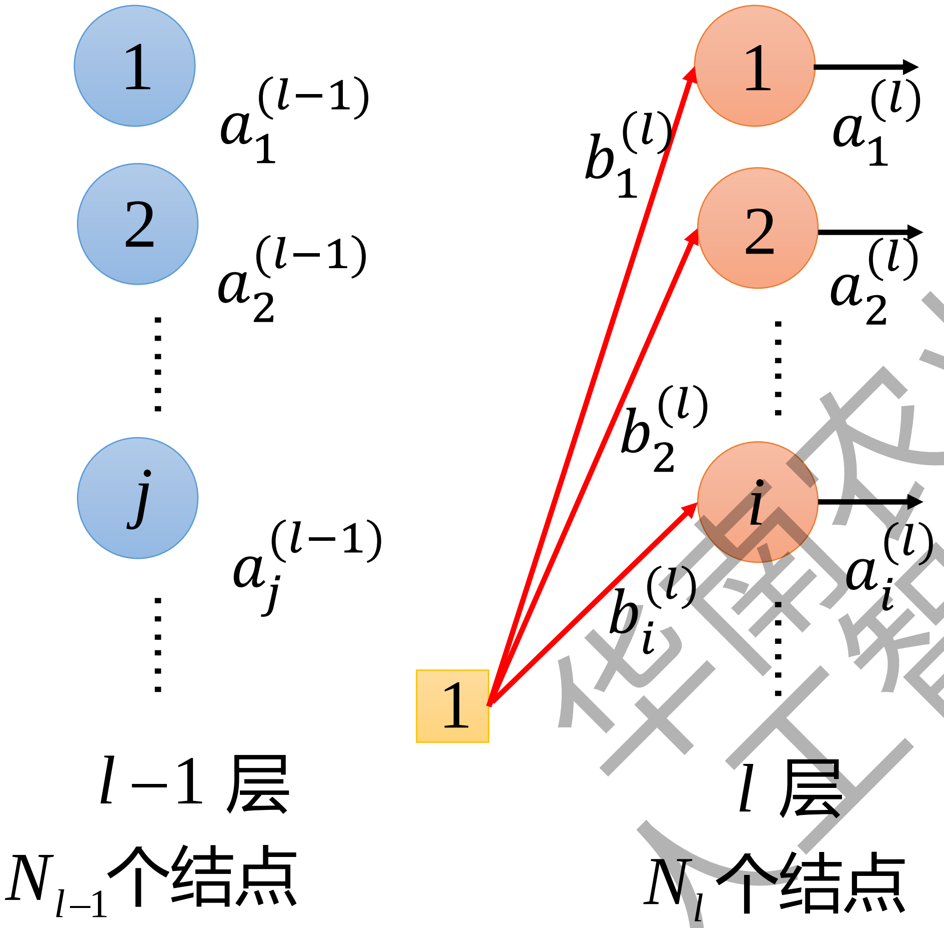
$\mathbf{a}^{(l)}$: 向量



$w_{i,j}^{(l)} \rightarrow$ 层 $l-1$ 到层 l

从神经元j (Layer $l-1$)
到神经元i (Layer l)

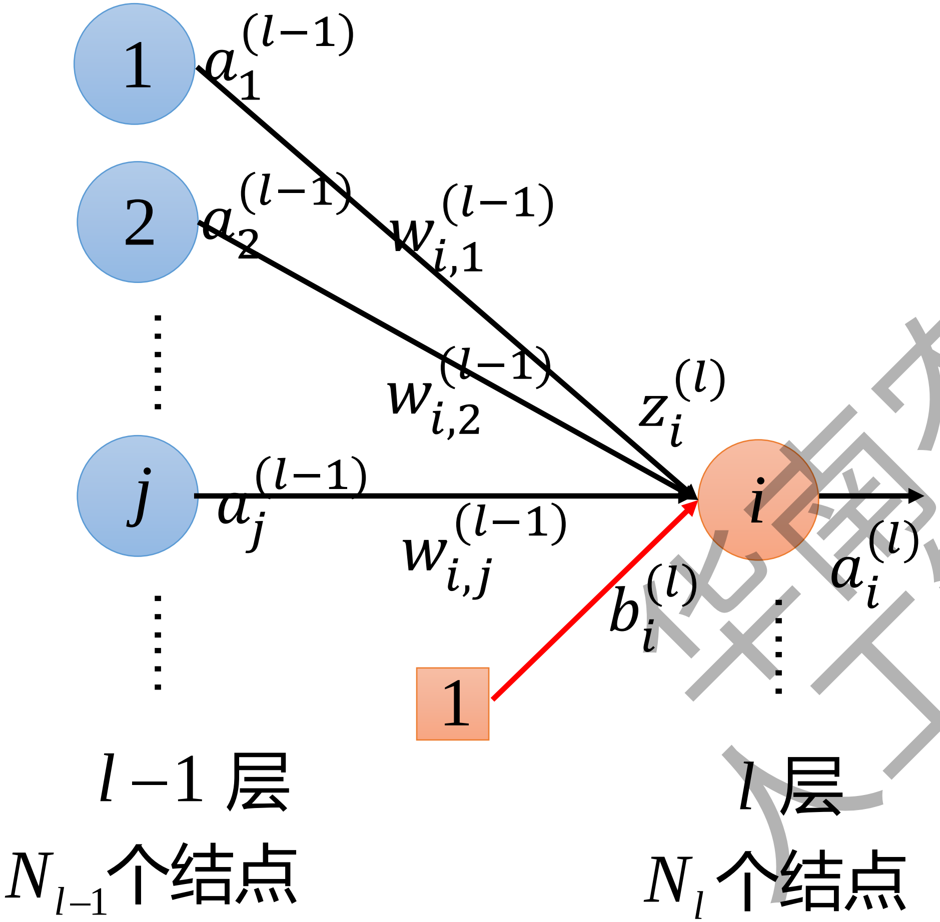
$$\mathbf{W}^{(l)} = \begin{array}{c} \overbrace{\hphantom{w_{1,1}^{(l)} \quad w_{1,2}^{(l)} \quad \cdots \quad}}^{N_{l-1}} \\ \left.\begin{bmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \\ \vdots & & \ddots \end{bmatrix}\right\} N_l \end{array}$$



$b_i^{(l)}$: bias for neuron i at layer l

$$\mathbf{b}^{(l)} = \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_i^{(l)} \\ \vdots \end{bmatrix}$$

bias for all neurons in layer l



$z_i^{(l)}$: input of the activation function for neuron i at layer l

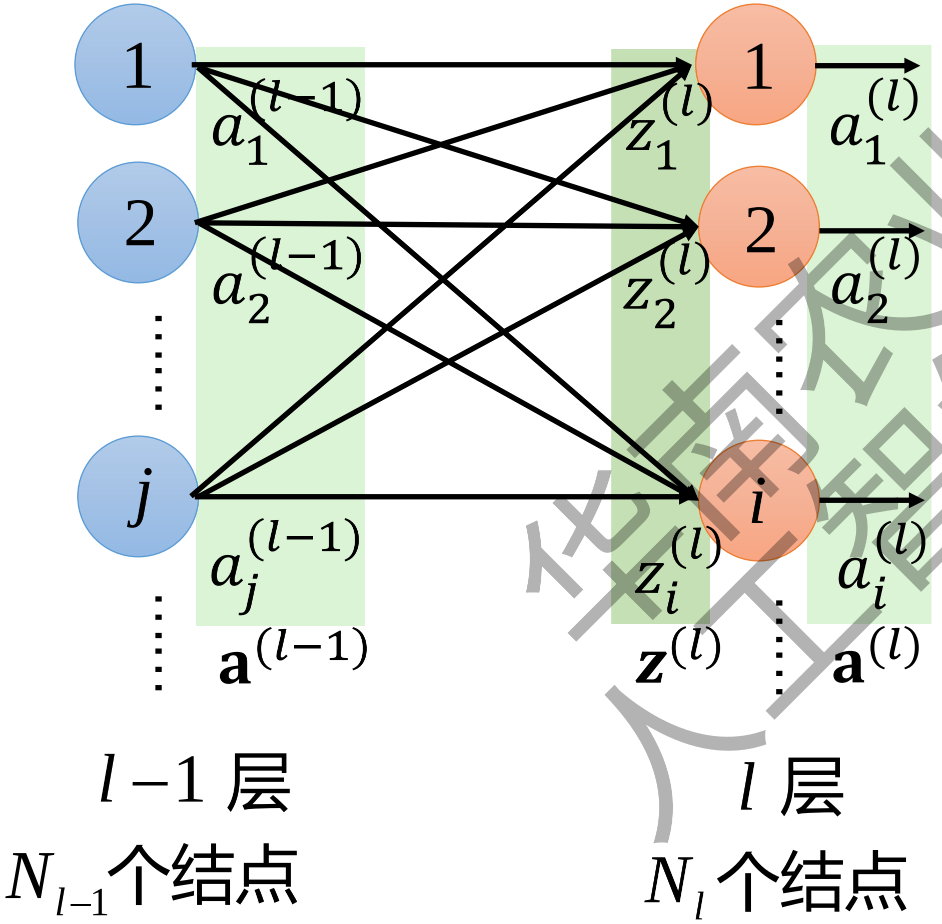
$\mathbf{z}^{(l)}$: input of the activation function all the neurons in layer l

$$z_i^{(l)} = w_{i,1}^{(l)} a_1^{(l-1)} + w_{i,2}^{(l)} a_2^{(l-1)} \dots + w_{i,j}^{(l)} a_j^{(l-1)} \dots + b_i^{(l)}$$

$$= \sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)}$$

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

- **相邻层输出之间的关系**

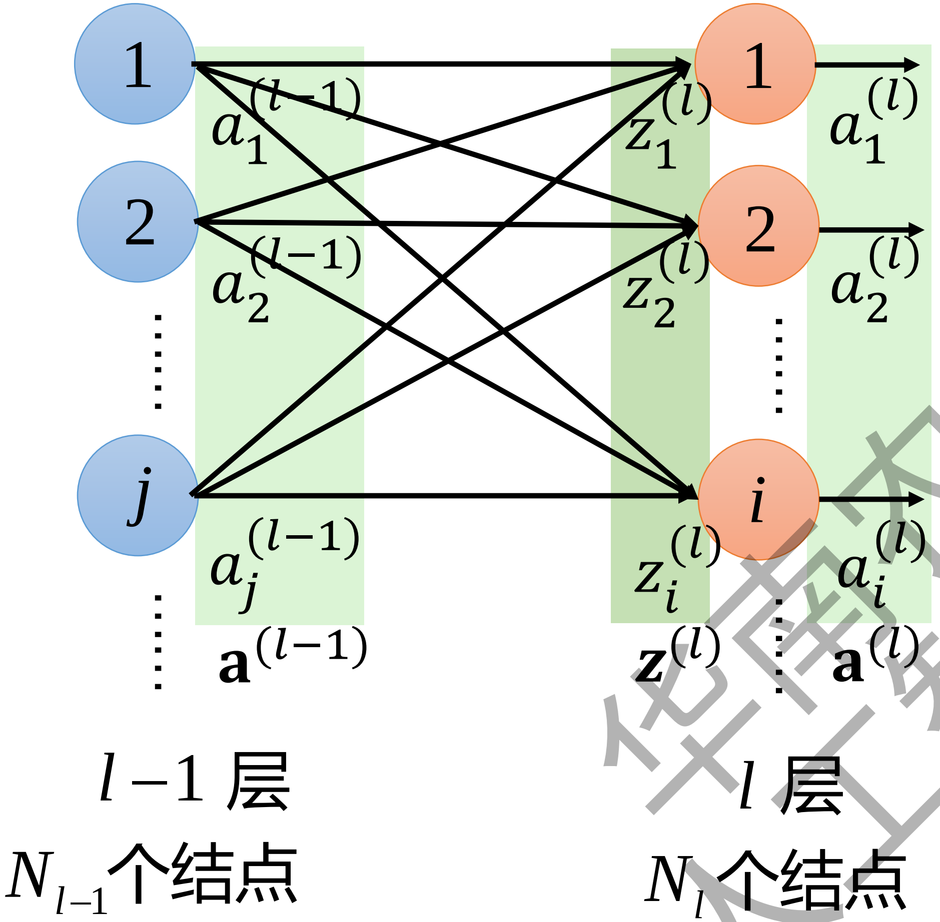


$$z_i^{(l)} = w_{i,1}^{(l)} a_1^{(l-1)} + w_{i,2}^{(l)} a_2^{(l-1)} \dots + w_{i,j}^{(l)} a_j^{(l-1)} \dots + b_i^{(l)}$$

$$= \sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)}$$

$$\begin{bmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_i^{(l)} \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_i^{(l)} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_i^{(l)} \\ \vdots \end{bmatrix}$$

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

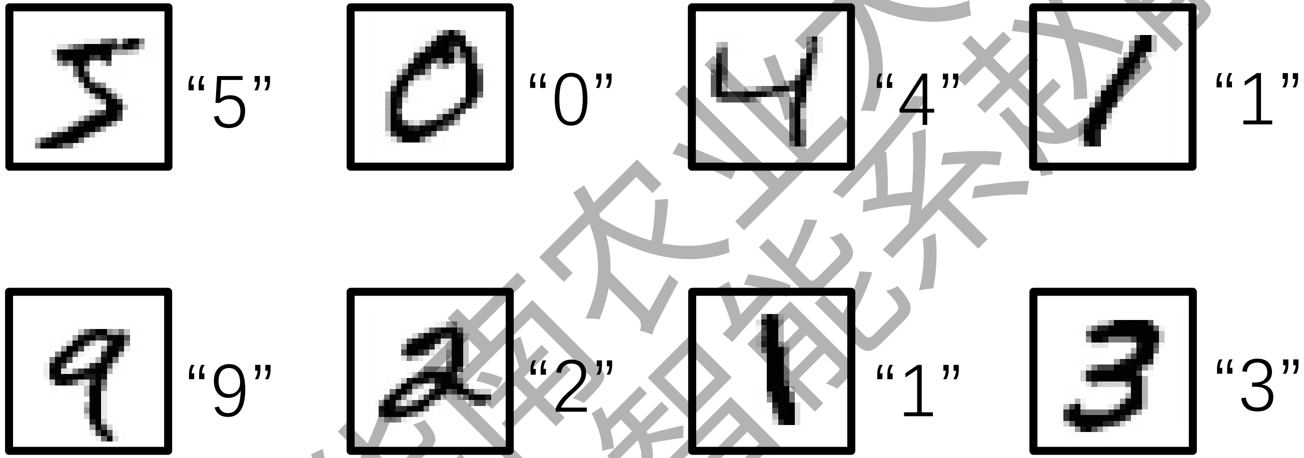


$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

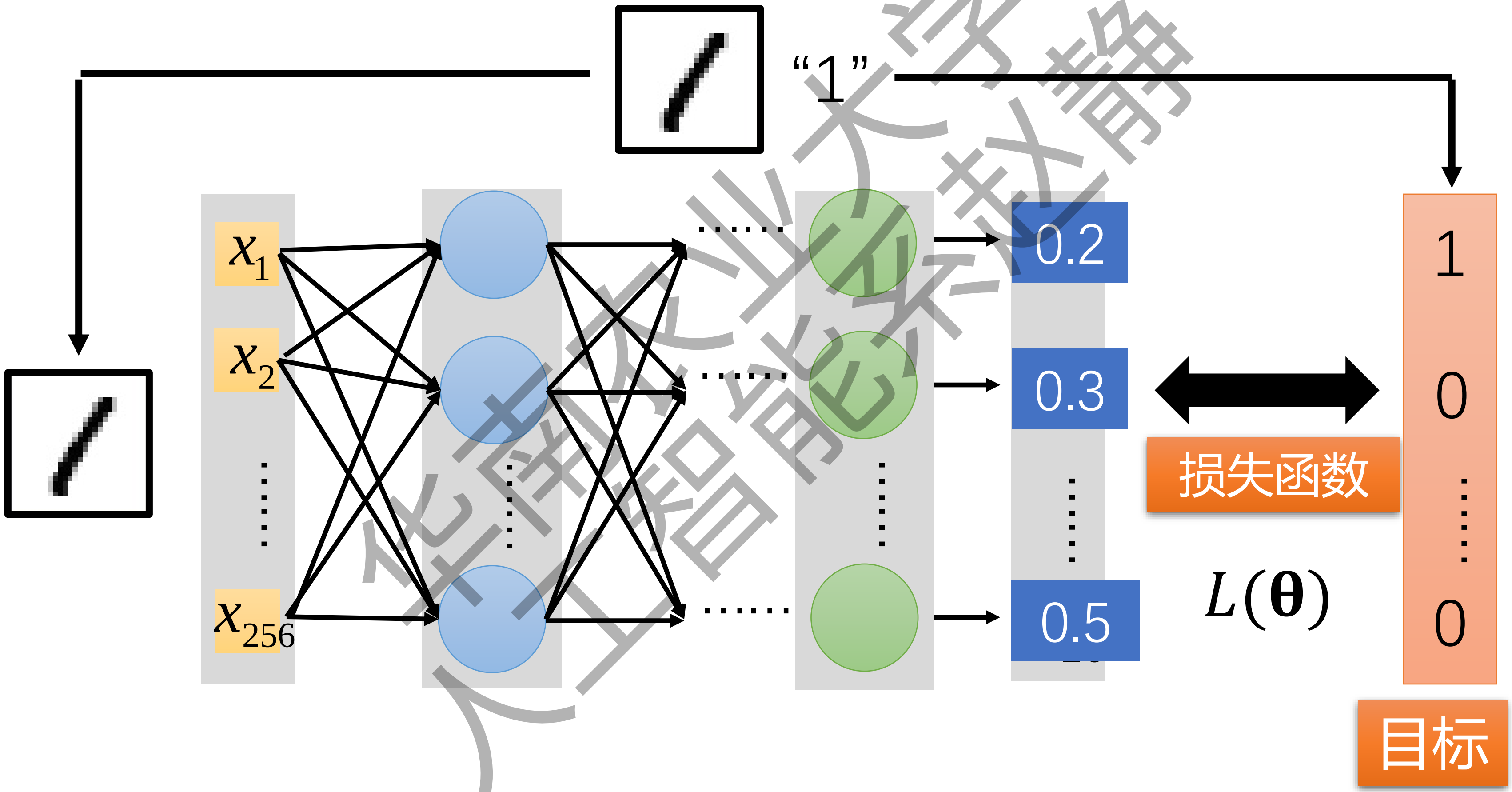
$$a_i^{(l)} = \sigma\left(z_i^{(l)}\right)$$

$$\begin{aligned}\mathbf{a}^{(l)} &= \sigma\left(\mathbf{z}^{(l)}\right) \\ &= \sigma\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)\end{aligned}$$

✓ 训练数据



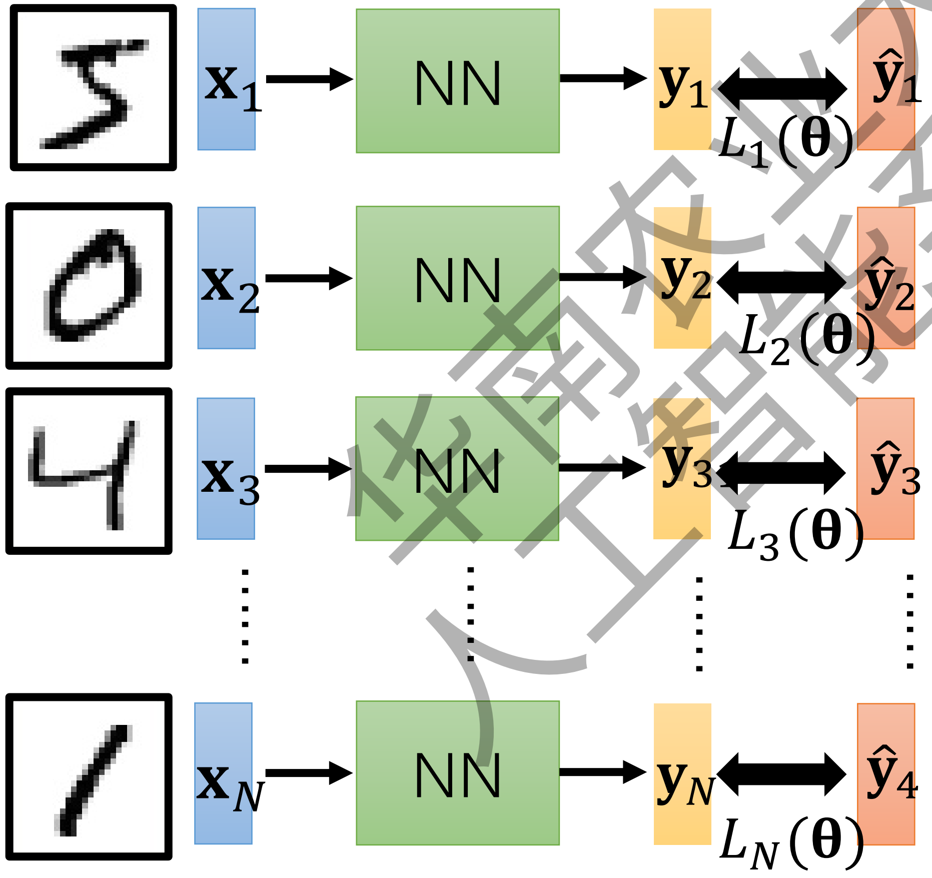
✓ 损失函数



损失函数根据任务要求定义：如交叉熵损失

✓ 目标函数

对所有样本 ...

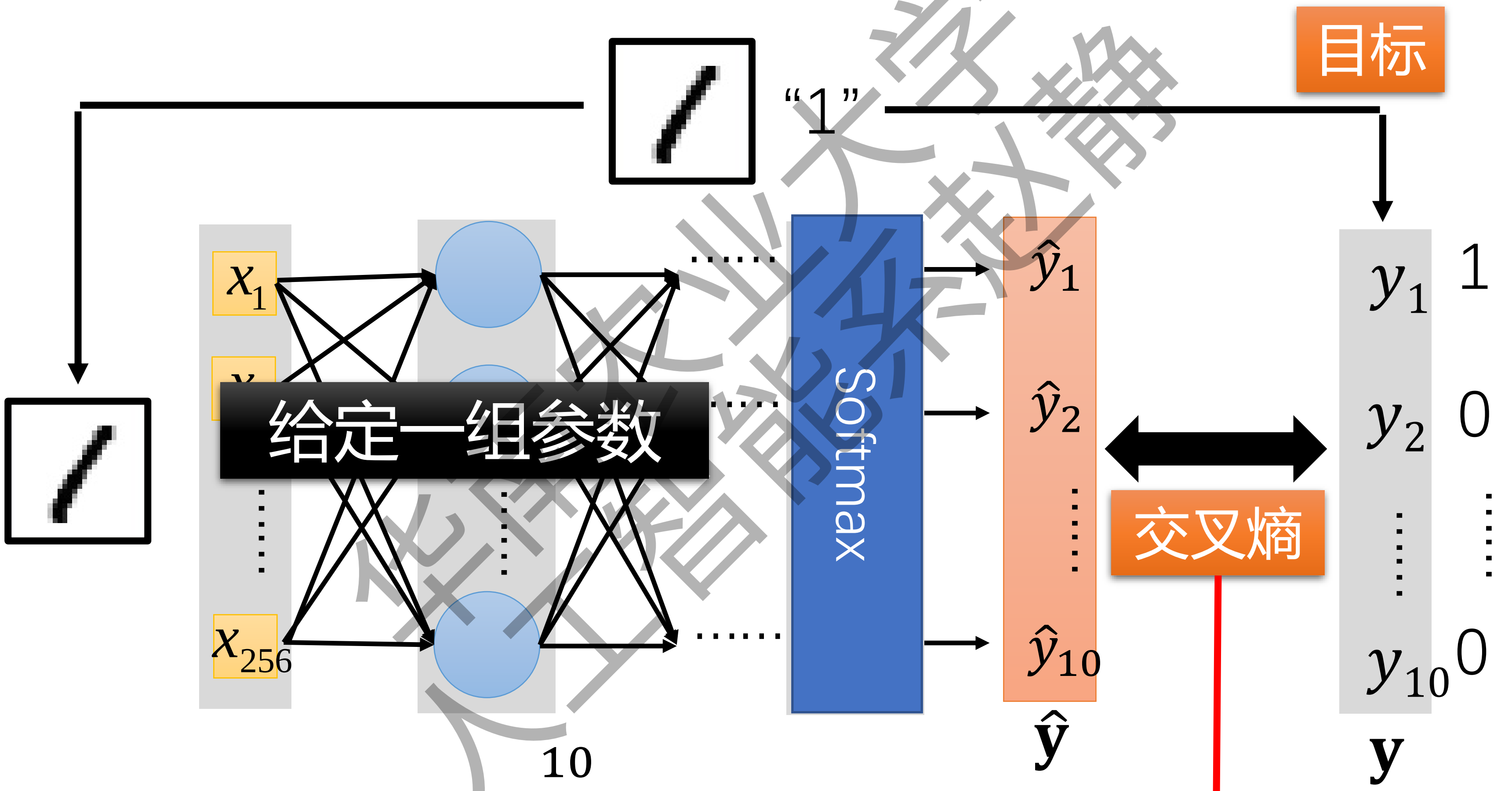


目标函数:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} L_i(\boldsymbol{\theta})$$

找到使得目标函数最小的网络参数 $\boldsymbol{\theta}^*$

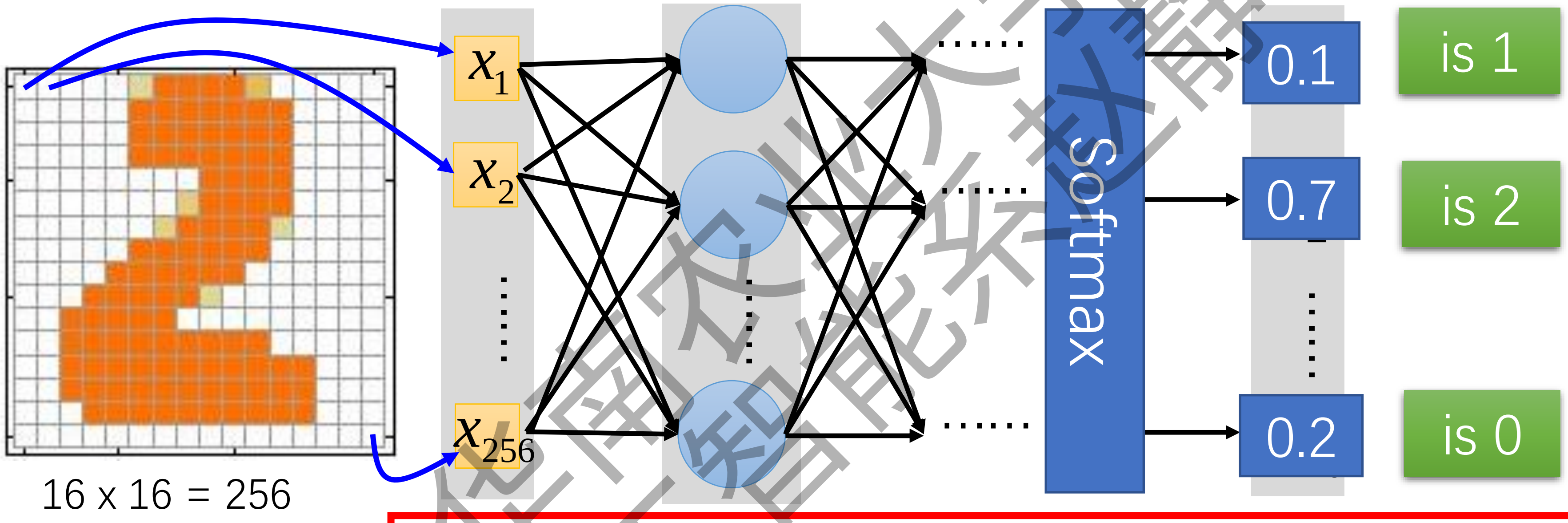
例



$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{m=1}^{10} \hat{y}_m \log(y_m)$$

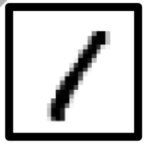
➢ 模型训练：反向传播 (Back Propogation, BP)算法

$$\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \cdots \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\}$$



Ink → 1
No ink → 0

设置网络参数$\boldsymbol{\theta}$，使得：

输入：  ➡ y_1 是最大值

输入：  ➡ y_2 是最大值

✓ 常规优化算法：梯度下降

网络参数：$\boldsymbol{\theta} = \left\{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \cdots \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\right\}$

初始参数：$\boldsymbol{\theta}^{(0)} \longrightarrow \boldsymbol{\theta}^{(1)} \longrightarrow \boldsymbol{\theta}^{(2)} \longrightarrow$ ......

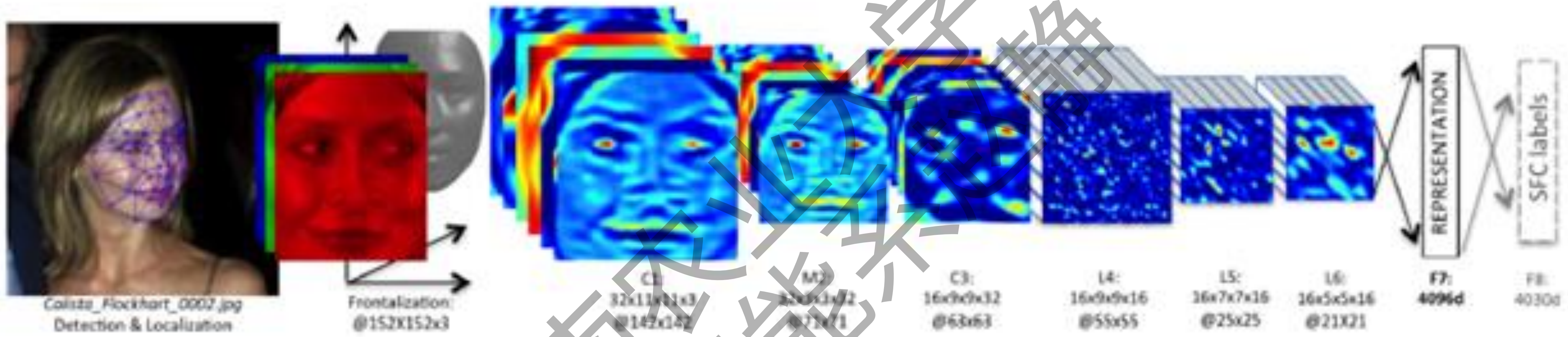
$\nabla J(\boldsymbol{\theta})$

$$\begin{bmatrix} \partial J(\boldsymbol{\theta})/\partial \mathbf{W}(1) \\ \partial J(\boldsymbol{\theta})/\partial b(1) \\ \vdots \\ \partial J(\boldsymbol{\theta})/\partial \mathbf{W}(2) \\ \partial J(\boldsymbol{\theta})/\partial b(2) \\ \vdots \end{bmatrix}$$

计算$\nabla J\left(\boldsymbol{\theta}^{(0)}\right)$　　$\boldsymbol{\theta}^{(1)} = \boldsymbol{\theta}^{(0)} - \eta \nabla L\left(\boldsymbol{\theta}^{(0)}\right)$

计算$\nabla J\left(\boldsymbol{\theta}^{(1)}\right)$　　$\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} - \eta \nabla L\left(\boldsymbol{\theta}^{(1)}\right)$

百万数量级的参数 ......



2014年 Facebook提出的Deepface人脸识别算法，
通过400多万张人脸图片，求出1800多万个参数分量

反向传播：更有效地计算梯度

神经网络符号表示

- $\mathbf{x}$：输入向量
- $w_{i,j}^{(l)}$：连接第$(l-1)$层的第j个神经元到第l层的第i个神经元的权重
- $b_i^{(l)}$：第l层的第i个神经元的权重
- $z_i^{(l)}$：第l层的第i个神经元的激活函数的输入 $z_i^{(l)} = \sum_{j=1}^{N_{l-1}} w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)}$
- $\sigma(.)$：激活函数
- $a_i^{(l)}$：第l层的第i个神经元的激活函数的输出 $a_i^{(l)} = \sigma\left(z_i^{(l)}\right)$
- $\hat{\mathbf{y}}$：输出向量 $\hat{y}_i = a_i^{(L)}$
- $\mathbf{y}$：真实标签
- J：目标函数 $J = \sum_{n=1}^{N} L_n(\hat{\mathbf{y}}, \mathbf{y})$
- $\delta_i^{(l)}$：第l层的第i个神经元的误差函数 $\delta_i^{(l)} = \frac{\partial J}{z_i^{(l)}}$

Review: 链式法则（Chain Rule)

Case 1

$$z = f(x) \quad \Rightarrow \quad y = g(x) \quad z = h(y)$$

$$x \xrightarrow{g} y \xrightarrow{h} z$$

$$\Delta x \to \Delta y \to \Delta z$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

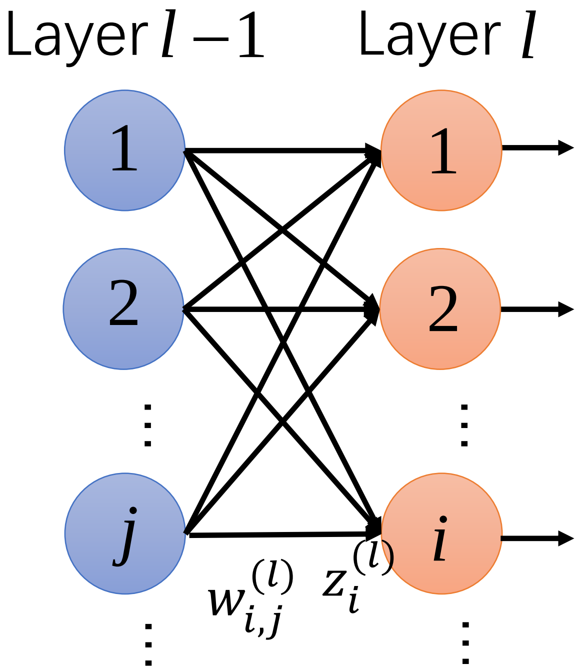
$$z = f(s) \quad \Rightarrow \quad x = g(s) \quad y = h(s) \quad z = k(x, y)$$

$$\begin{tikzcd} & x \arrow[dr] & \\ s \arrow[ur] \arrow[dr] & & z \\ & y \arrow[ur] & \end{tikzcd}
\qquad
\begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd}$$

$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

✓ BP算法

- 链式法则



$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \frac{\partial J}{\partial z_i^{(l)}}$$

$$\frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} = \begin{cases} a_j^{(l-1)} & l > 1 \\ x_j & l = 1 \end{cases}$$

Error signal

$$\frac{\partial J}{\partial z_i^{(l)}} = \delta_i^{(l)}$$

Forward Pass

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{z}^{(1)}\right)$$

……

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

$$\mathbf{a}^{(l-1)} = \sigma\left(\mathbf{z}^{(l-1)}\right)$$

Backward Pass

$$\boldsymbol{\delta}^{(L)} = \sigma'(\mathbf{z}^{(L)}) \odot \nabla_{\hat{\mathbf{y}}} J$$

$$\boldsymbol{\delta}^{(L-1)} = \sigma'(\mathbf{z}^{(L-1)}) \odot (\mathbf{W}^{(L)})^T \boldsymbol{\delta}^{(L)}$$

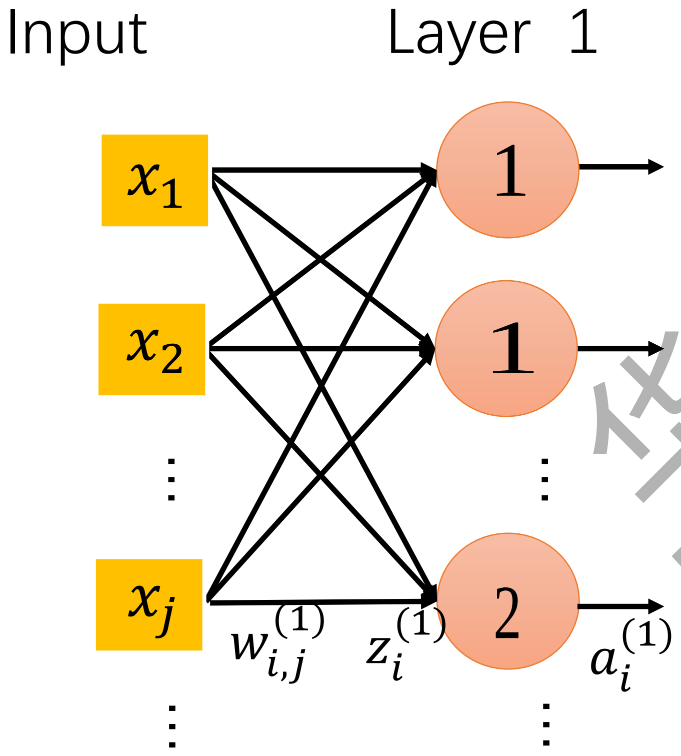
……

$$\boldsymbol{\delta}^{(l)} = \sigma'(\mathbf{z}^{(l)}) \odot (\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}$$

……

- $\partial J/\partial w_{ij}^{(l)}$——第1项

$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \boxed{\frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}}} \frac{\partial J}{\partial z_i^{(l)}}$$



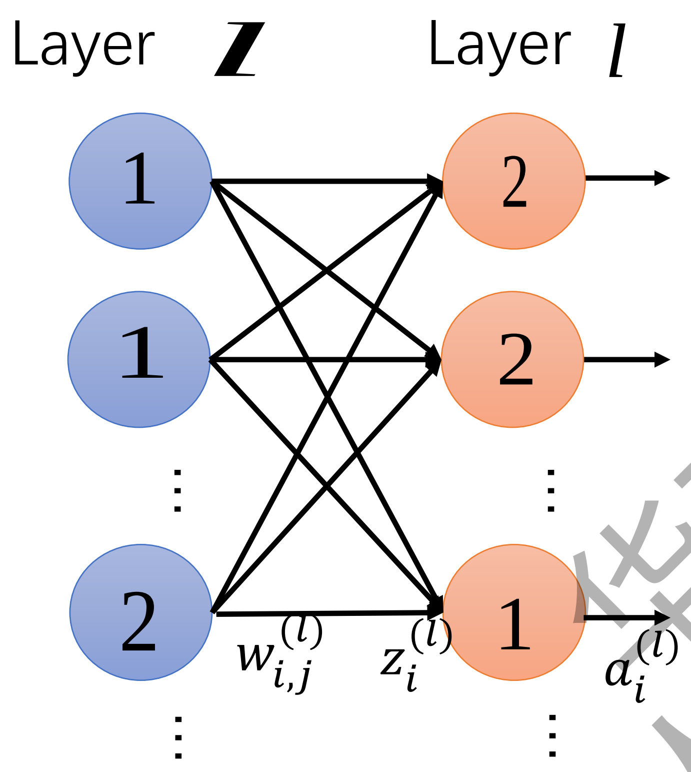
$If\ l > 1$

$$z_i^{(l)} = \sum_j w_{i,j}^{(l)} a_j^{(l-1)} + b_i^{(l)} \qquad \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} = a_j^{(l-1)}$$

$If\ l = 1$

$$z_i^{(1)} = \sum_j w_{i,j}^{(1)} x_j + b_i^{(1)} \qquad \frac{\partial z_i^{(1)}}{\partial w_{i,j}^{(1)}} = x_j$$

- $\partial J/\partial w_{ij}^{(l)}$ —— 第2项

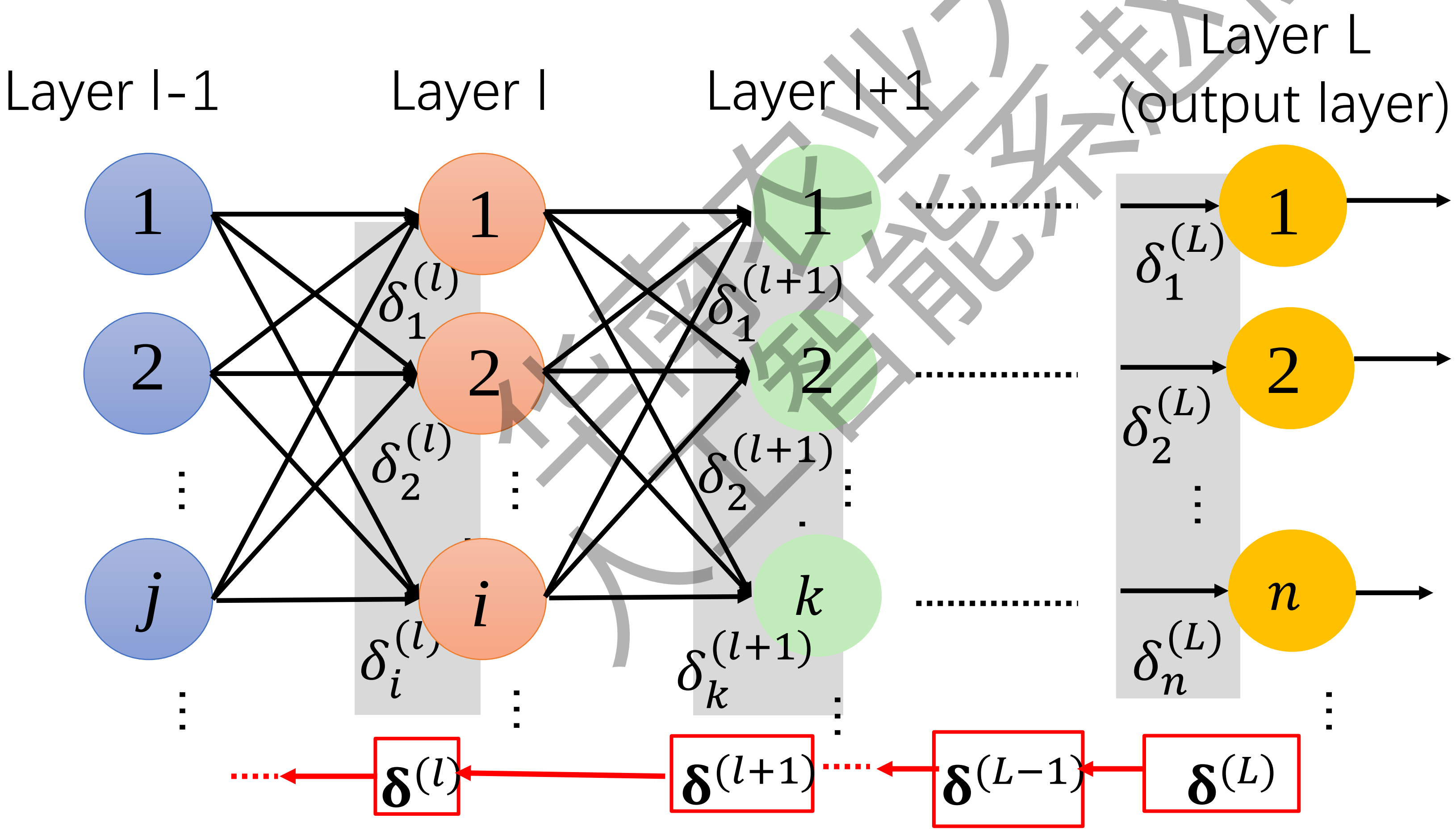


$$\Delta w_{i,j}^{(l)} \rightarrow \Delta z_i^{(l)} \cdots\cdots \rightarrow \Delta J$$

$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

1. 怎样计算 $\boldsymbol{\delta}^{(L)}$
2. $\boldsymbol{\delta}^{(l)}$ 和 $\boldsymbol{\delta}^{(l+1)}$之间的关系



$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

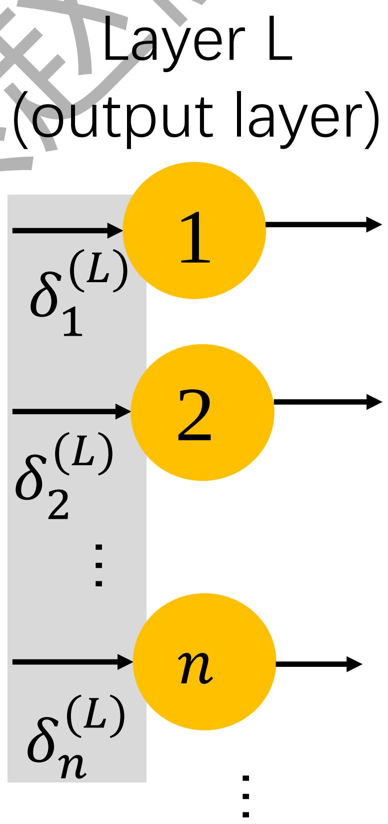
1. 怎样计算 $\boldsymbol{\delta}^{(L)}$

2. $\boldsymbol{\delta}^{(\mathrm{l})}$ 和 $\boldsymbol{\delta}^{(\mathrm{l+1})}$ 之间的关系

$$\delta_n^{(L)} = \frac{\partial J}{\partial z_n^{(L)}}$$

$$\Delta z_n^{(L)} = \Delta a_n^{(L)} = \Delta \hat{y}_n \rightarrow \Delta J$$

$$= \underbrace{\frac{\partial \hat{y}_n}{\partial z_n^{(L)}}}_{\sigma'(z_n^{(L)})} \frac{\partial J}{\partial \hat{y}_n} \rightarrow \text{与目标函数有关}$$



$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

1. 怎样计算 $\boldsymbol{\delta}^{(L)}$

2. $\boldsymbol{\delta}^{(l)}$ 和 $\boldsymbol{\delta}^{(l+1)}$ 之间的关系

$$\delta_n^{(L)} = \frac{\partial J}{\partial z_n^{(L)}}$$

$$= \frac{\partial \hat{y}_n}{\partial z_n^{(L)}} \frac{\partial J}{\partial \hat{y}_n}$$

$$= \sigma' \left(z_n^{(L)}\right) \frac{\partial J}{\partial \hat{y}_n}$$

$$\sigma'\left(z^{(L)}\right) = \begin{bmatrix} \sigma'\left(z_1^{(L)}\right) \\ \sigma'\left(z_2^{(L)}\right) \\ \vdots \\ \sigma'\left(z_n^{(L)}\right) \\ \vdots \end{bmatrix} \qquad \nabla J(\hat{\mathbf{y}}) = \begin{bmatrix} \partial J / \partial \hat{y}_1 \\ \partial J / \partial \hat{y}_2 \\ \vdots \\ \partial J / \partial \hat{y}_n \\ \vdots \end{bmatrix}$$

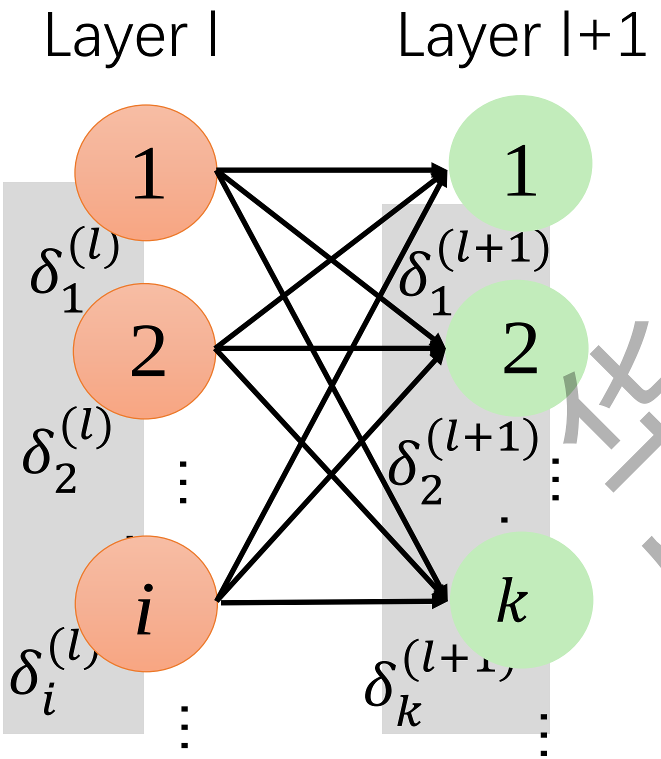
$$\boldsymbol{\delta}^{(L)} = \sigma'\left(\mathbf{z}^{(L)}\right) \odot \nabla J(\hat{\mathbf{y}})$$

按元素乘

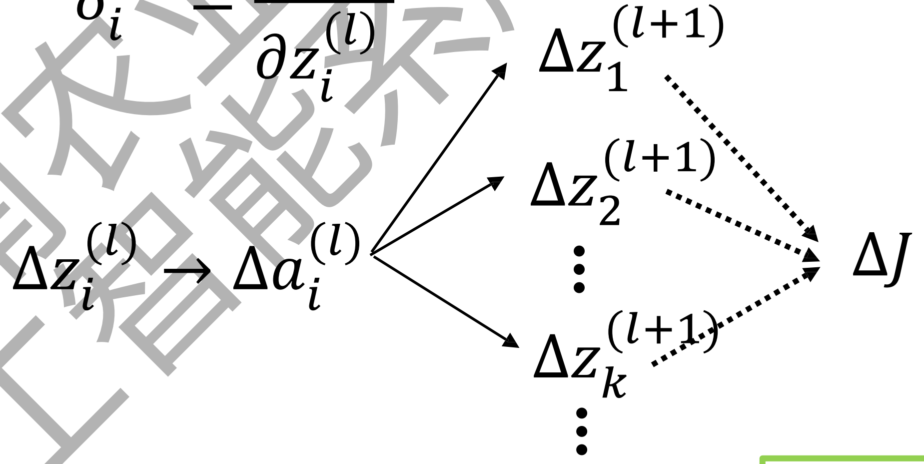
$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

1. 怎样计算 $\boldsymbol{\delta}^{(L)}$

2. $\boldsymbol{\delta}^{(l)}$ 和 $\boldsymbol{\delta}^{(l+1)}$之间的关系

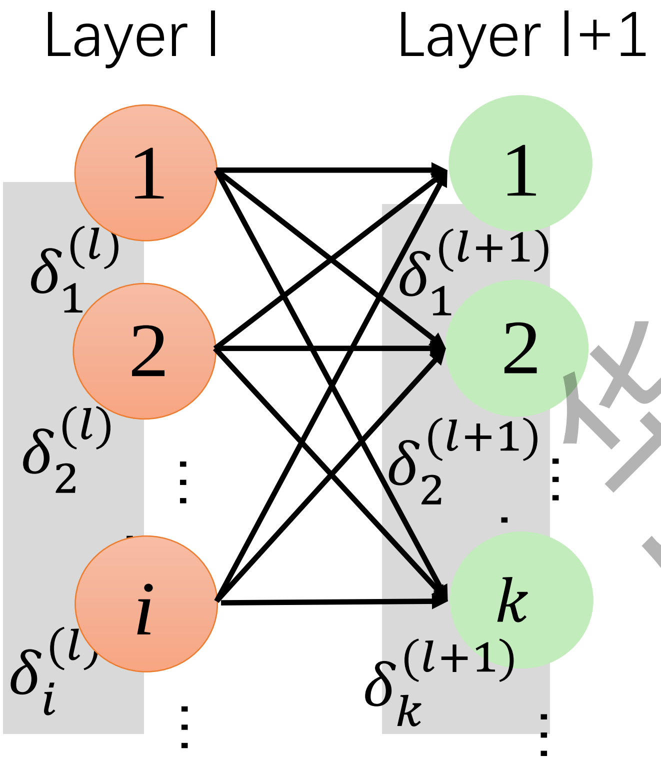


$$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}}$$



$$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} = \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \sum_k \frac{\partial z_k^{(l+1)}}{\partial a_i^{(l)}} \boxed{\frac{\partial J}{\partial z_k^{(l+1)}}} \rightarrow \delta_k^{(l+1)}$$

$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$



$$\Delta z_i^{(l)} \rightarrow \Delta a_i^{(l)} \rightarrow \Delta z_1^{(l+1)}, \Delta z_2^{(l+1)}, \ldots, \Delta z_k^{(l+1)}, \ldots \rightarrow \Delta J$$

$$\delta_i^{(l)} = \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \sum_k \frac{\partial z_k^{(l+1)}}{\partial a_i^{(l)}} \boxed{\frac{\partial J}{\partial z_k^{(l+1)}}} \rightarrow \delta_k^{(l+1)}$$

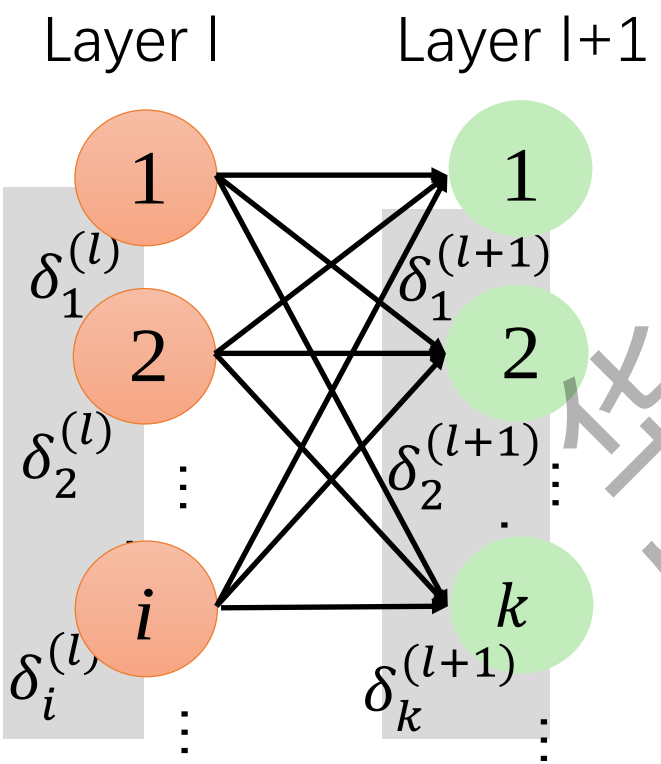
$$\sigma'(z_i^{(l)})$$

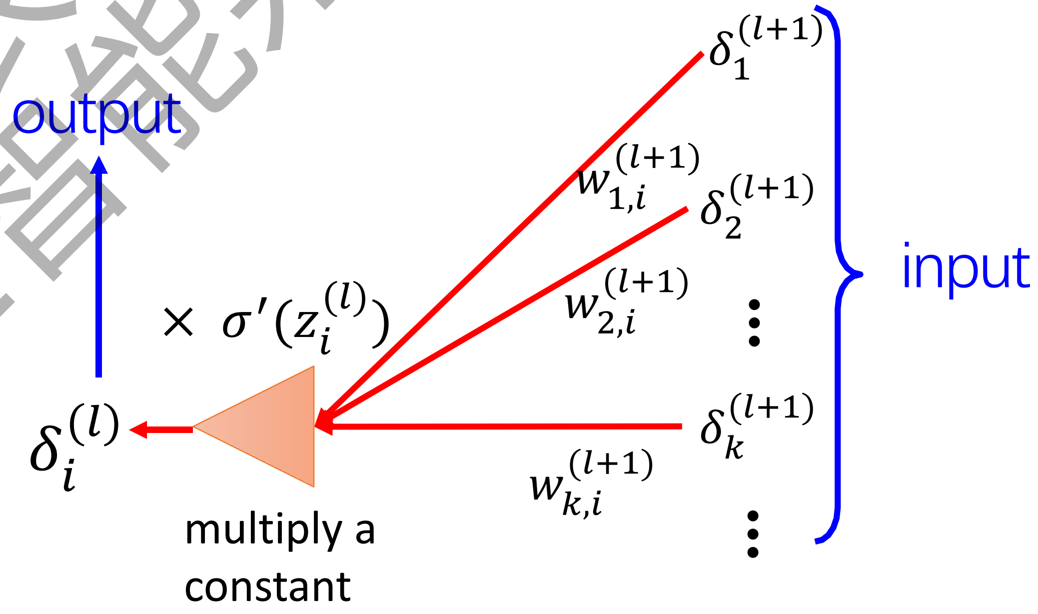
$$z_k^{(l+1)} = \sum_i w_{k,i}^{(l+1)} a_i^{(l)} + b_k^{(l+1)}$$

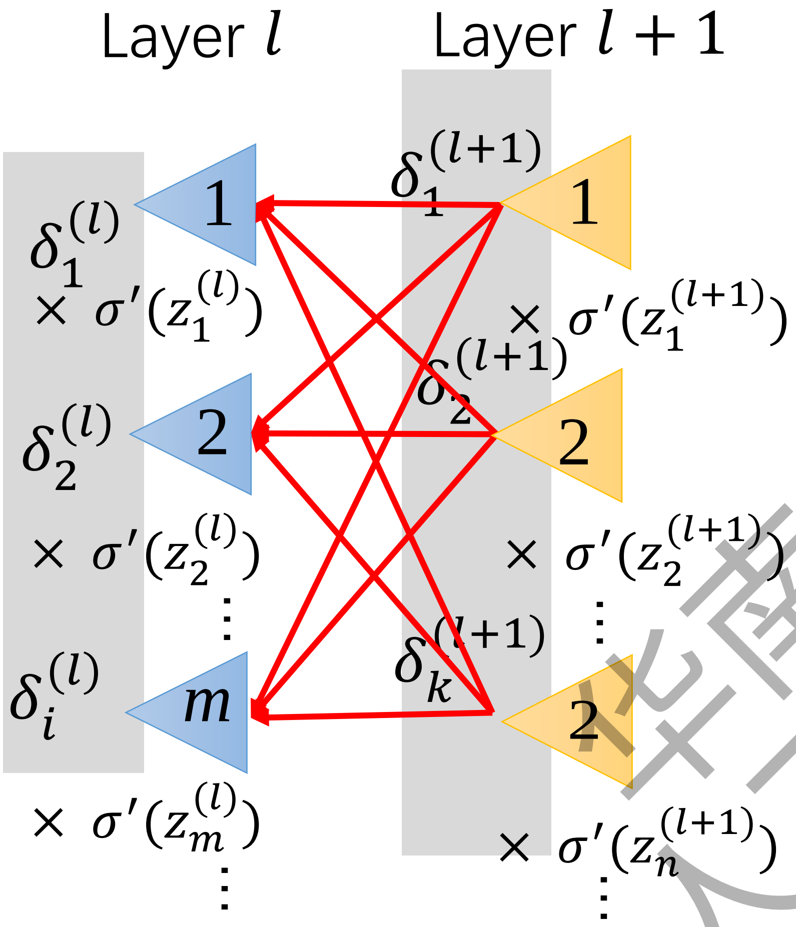
$$\delta_i^{(l)} = \sigma'(z_i^{(l)}) \sum_k w_{ki}^{(l+1)} \delta_k^{(l+1)}$$

$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \delta_i^{(l)}$$

$$\delta_i^{(l)} = \sigma'(z_i^{(l)}) \sum_k w_{k,i}^{(l+1)} \delta_k^{(l+1)}$$





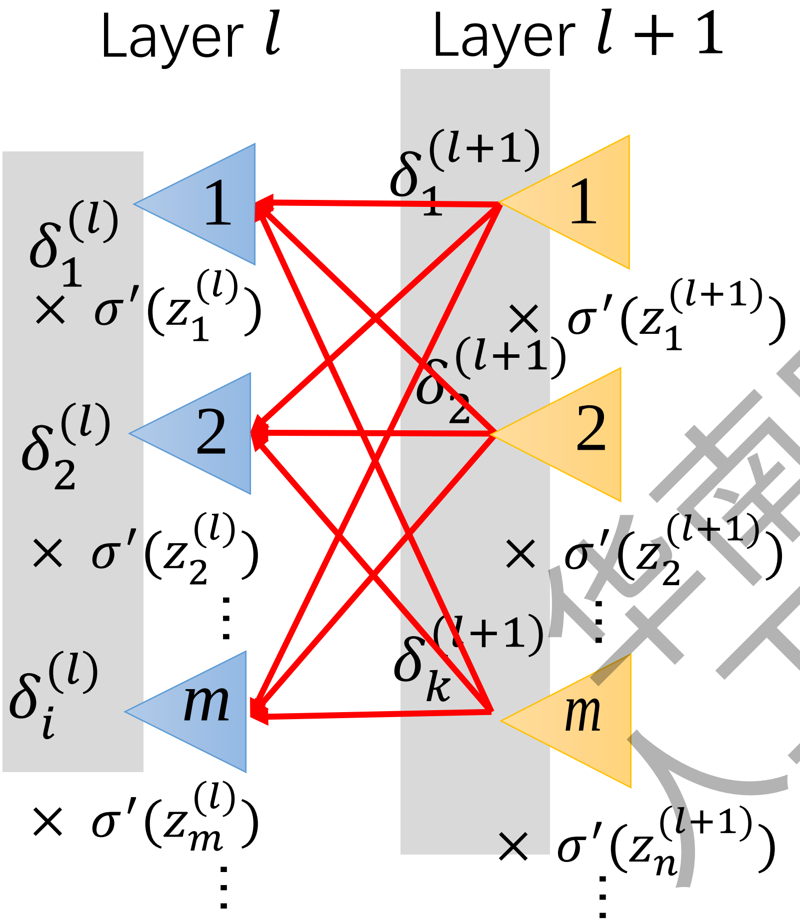


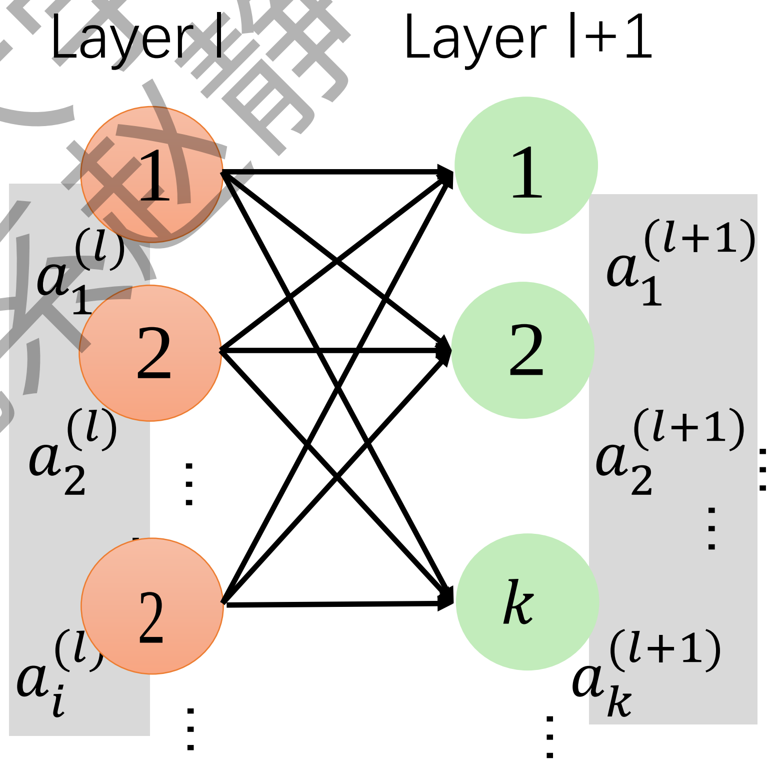
$$\delta_i^{(l)} = \sigma'(z_i^{(l)}) \sum_k w_{k,i}^{(l+1)} \delta_k^{(l+1)}$$

$$\sigma'(\mathbf{z}^{(l)}) = \begin{bmatrix} \sigma'\left(z_1^{(l)}\right) \\ \sigma'\left(z_2^{(l)}\right) \\ \vdots \\ \sigma'\left(z_n^{(l)}\right) \\ \vdots \end{bmatrix}$$

$$\boldsymbol{\delta}^l = \sigma'(\mathbf{z}^l) \cdot \mathbf{W}^{(l+1)^T} \boldsymbol{\delta}^{(l+1)}$$

反向 VS 前向





$$\boldsymbol{\delta}^l = \sigma'(\boldsymbol{z}^l)\ \cdot\ \boldsymbol{W}^{(l+1)^T}\boldsymbol{\delta}^{(l+1)}$$

$$\boldsymbol{a}^{(l+1)} = \sigma\left(\boldsymbol{W}^{(l+1}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l+1)}\right)$$

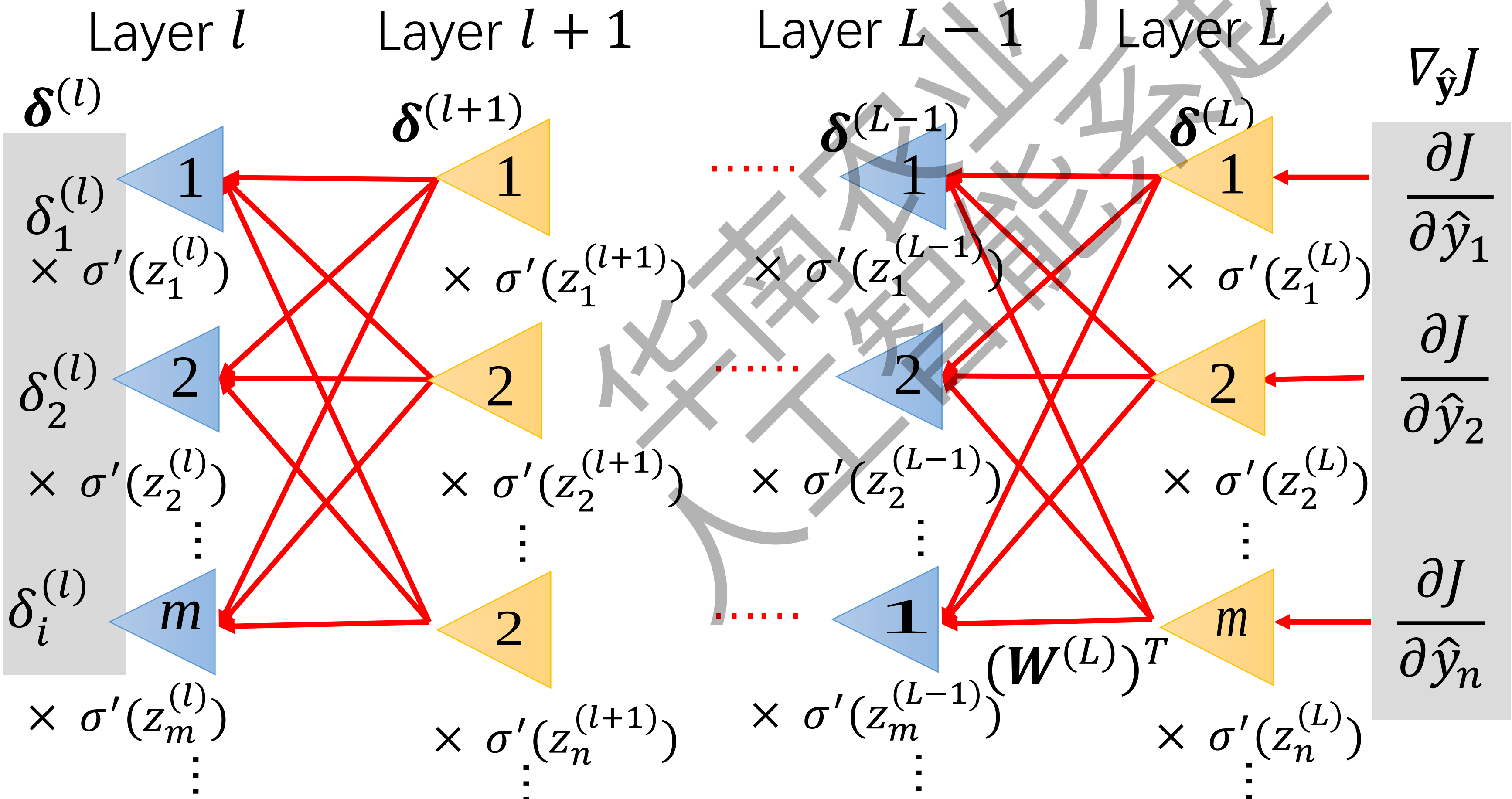
$$\frac{\partial J}{\partial w_{i,j}^{(l)}} = \boxed{\frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}}}\,\boxed{\frac{\partial J}{\partial z_i^{(l)}}} \rightarrow \boxed{\delta_i^l}$$

1. 怎样计算 $\delta^{(L)}$

$$\boldsymbol{\delta}^{(L)} = \sigma'\left(\mathbf{z}^{(L)}\right) \cdot \nabla J(\hat{\mathbf{y}})$$

2. $\delta^{(l)}$和 $\delta^{(l+1)}$之间的关系

$$\boldsymbol{\delta}^{l} = \sigma'(\mathbf{z}^{l}) \cdot \boldsymbol{W}^{(l+1)^T}\boldsymbol{\delta}^{(l+1)}$$

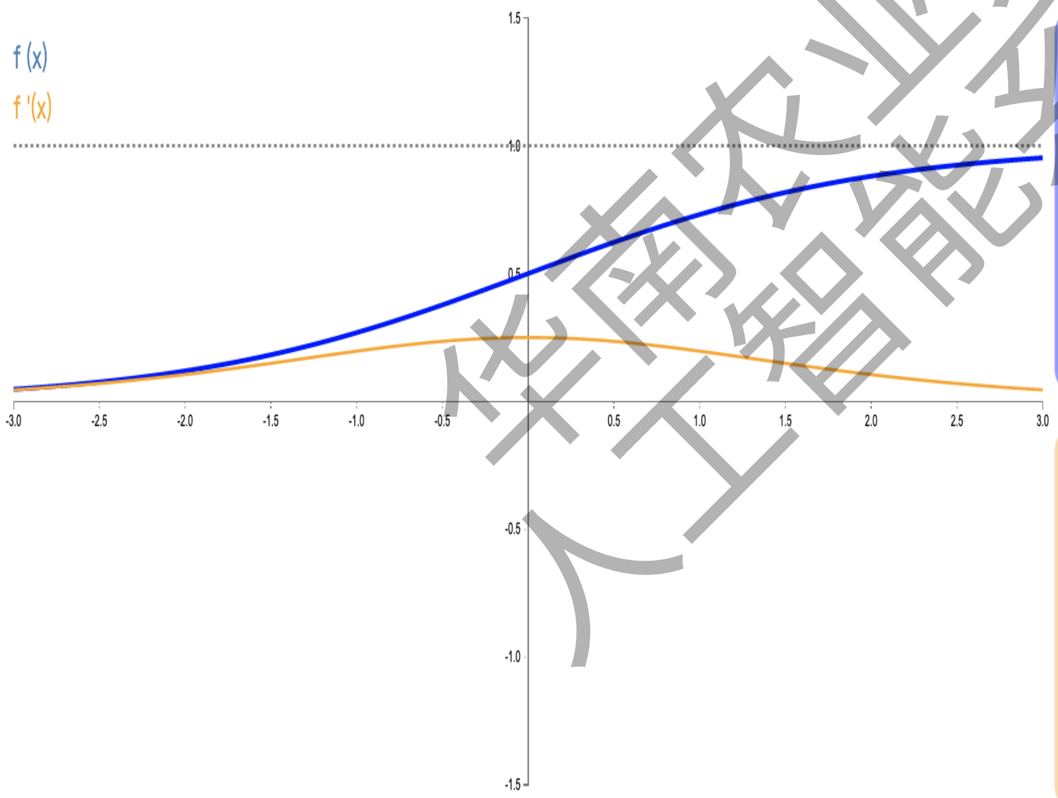


✓ 激活函数

Sigmoid函数：$sigmoid(x) = \frac{1}{(1+e^{-x})}$

梯度消失问题

Sigmoid



值域: (0, 1)

单调性: ✅

连续性: C∞

Identity at Origin: ❌

对称性: 不对称

值域: (0,0.25)

单调性: ❌

连续性: ✅

梯度消失: Yes

梯度爆炸: No

神经元饱和: Yes

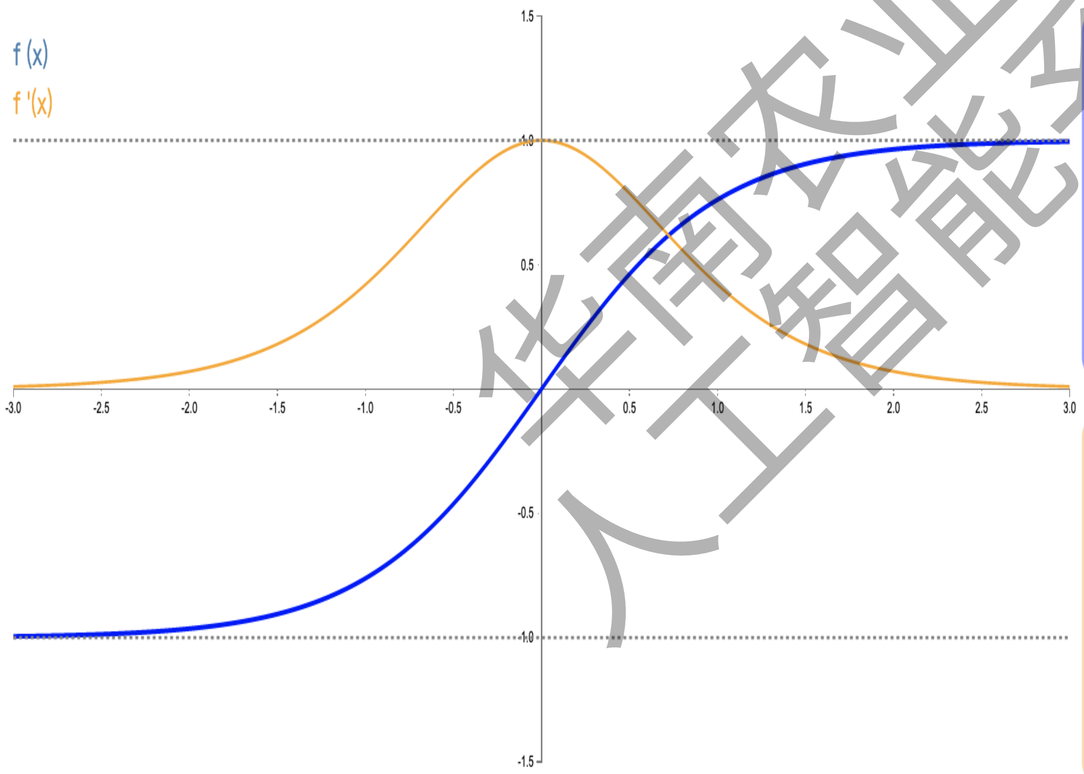
神经元死亡: No

Tanh函数：$\tanh(x)=\frac{1-\exp(-2x)}{1+\exp(-2x)}=2sigmoid(x)-1$

$$\varphi'(x)=1-[\varphi(x)]^2$$

收敛更快，减轻梯度消失现象

Tanh



值域: (-1, -1)

单调性: ✅

连续性: C∞

Identity at Origin: ✅

对称性: Anti-Symmetrical

值域: (0, 1]

单调性: ❌

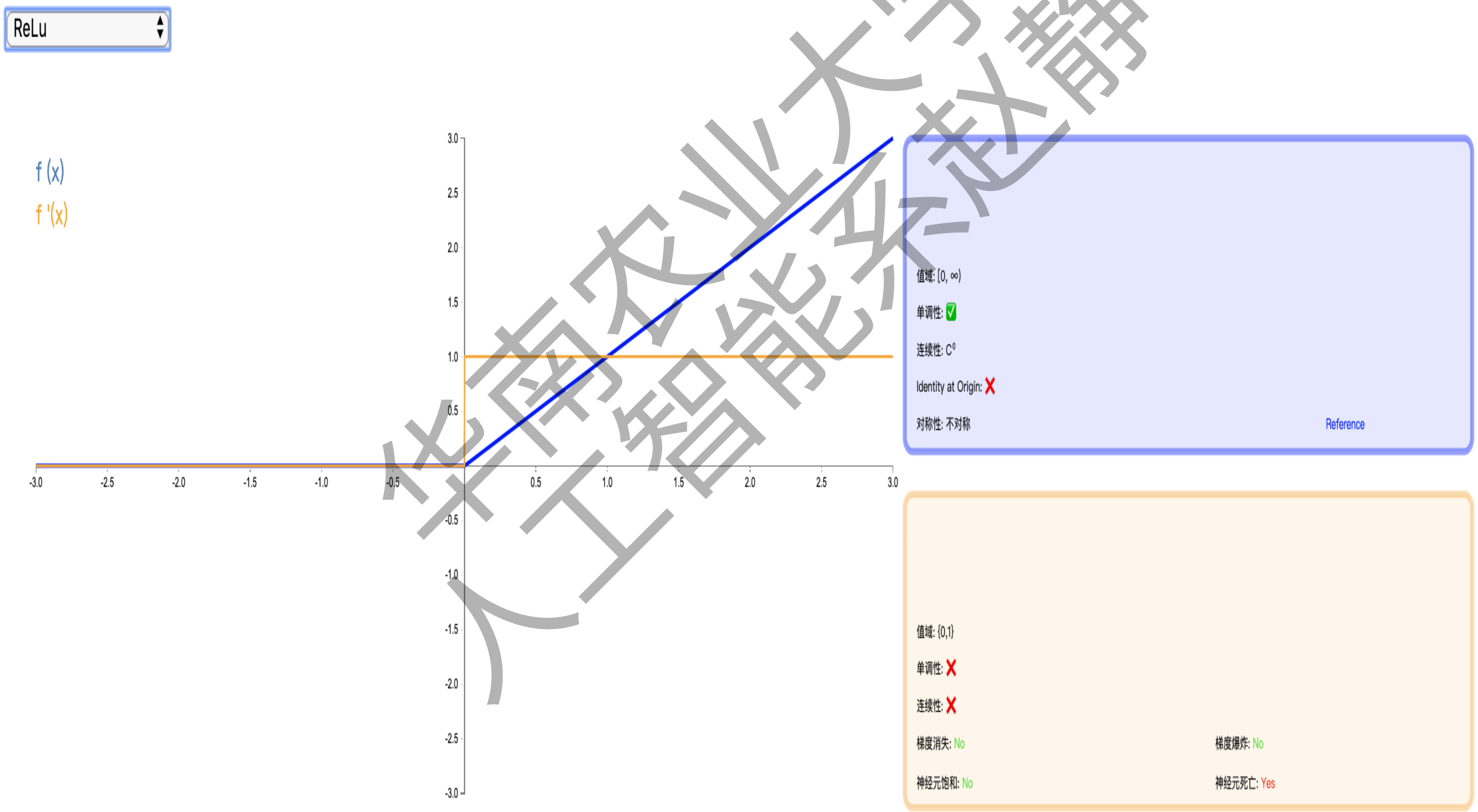
连续性: ✅

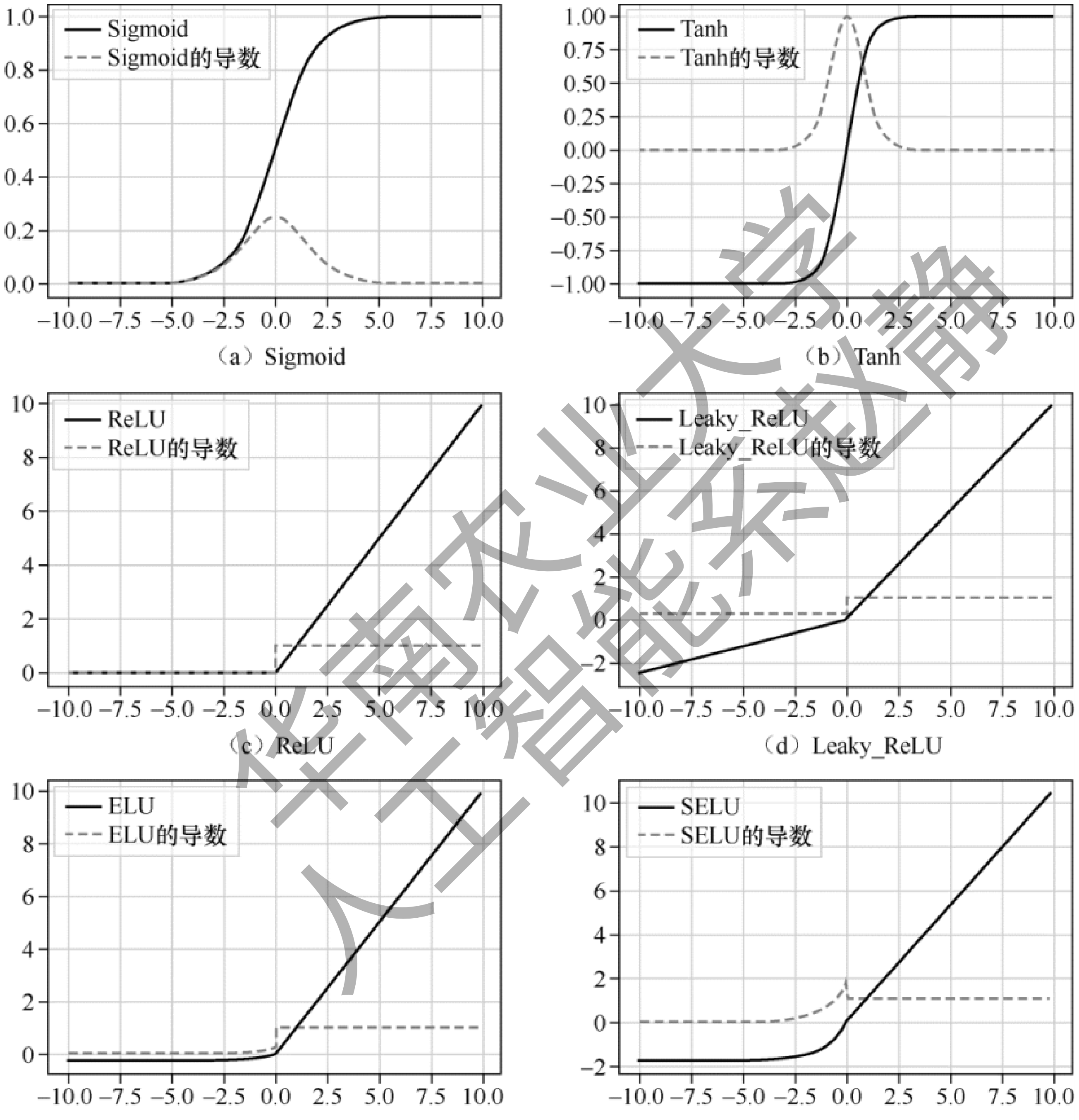
梯度消失: Yes　　梯度爆炸: No

神经元饱和: Yes　　神经元死亡: No

ReLU (**Rectified Linear Unit**)函数：$ReLU(x) = max(x, 0)$

计算量小，缓解过拟合，解决了梯度消失问题，收敛速度快





（a）Sigmoid （b）Tanh

（c）ReLU （d）Leaky_ReLU

（e）ELU （f）SELU

激活函数的选择

- 首选ReLU，速度快，但是要注意学习速率
- 如果 ReLU 效果欠佳，尝试使用ReLU的改进版本：Leaky ReLU、ELU 或 MaxOut 等
- 可以尝试使用 tanh
- Sigmoid 和 tanh 在 RNN（LSTM、注意力机制等）结构中作为门控或者概率值。其它情况下，减少 sigmoid 的使用

- ◆ 神经元结构
- ◆ 前馈全连接神经网络DNN
- ◆ 卷积神经网络CNN
  - 卷积层（Convolutional Layer)
  - 池化层（Pooing Layer）

3. 卷积神经网络（Convolutional Neural Network，CNN）

2012年，Hinton 组参加 ImageNet 竞赛，使用 CNN 模型以超过第二名10个百分点的成绩夺得当年竞赛的冠军

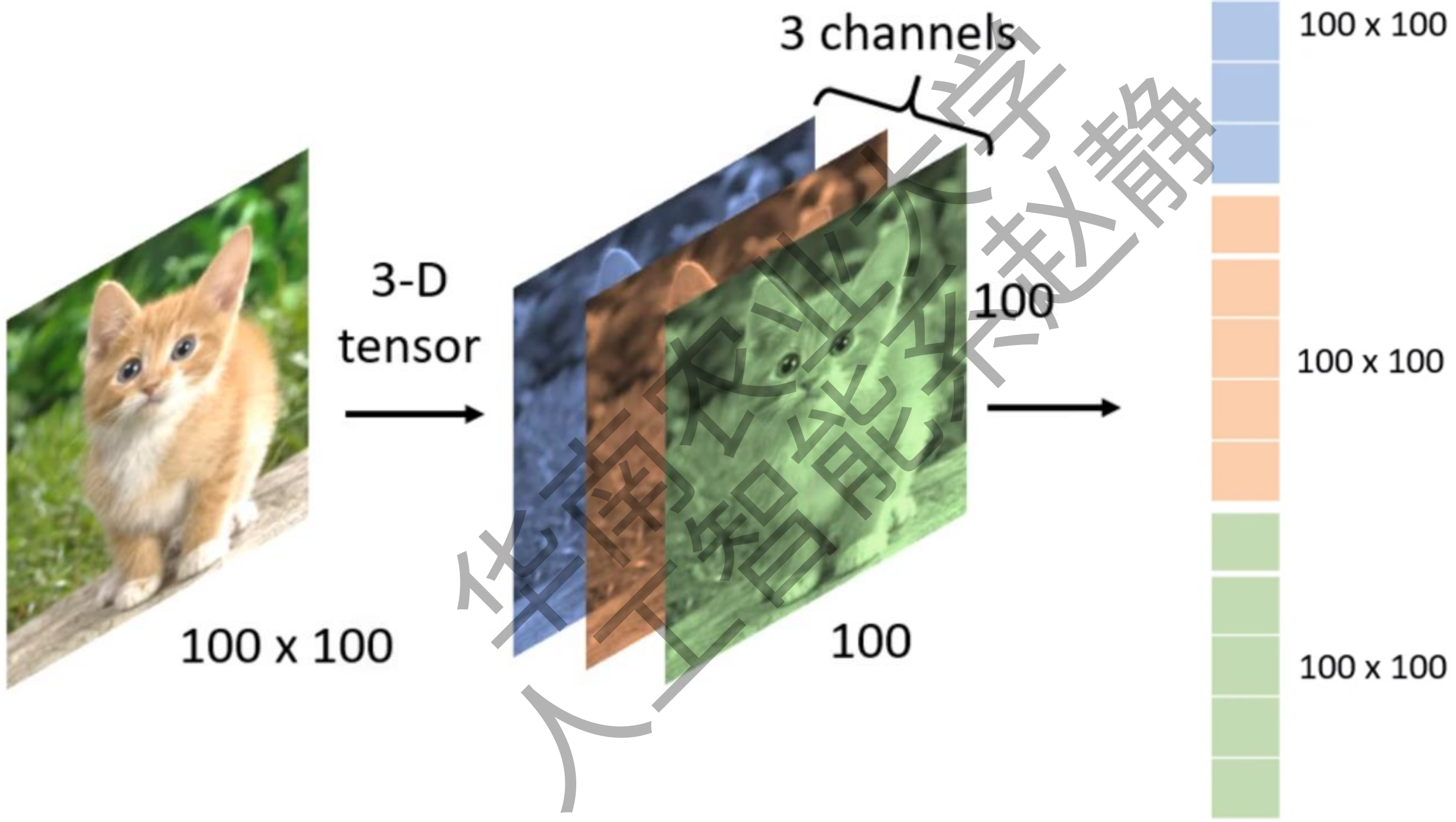
- 全连接神经网络FCN

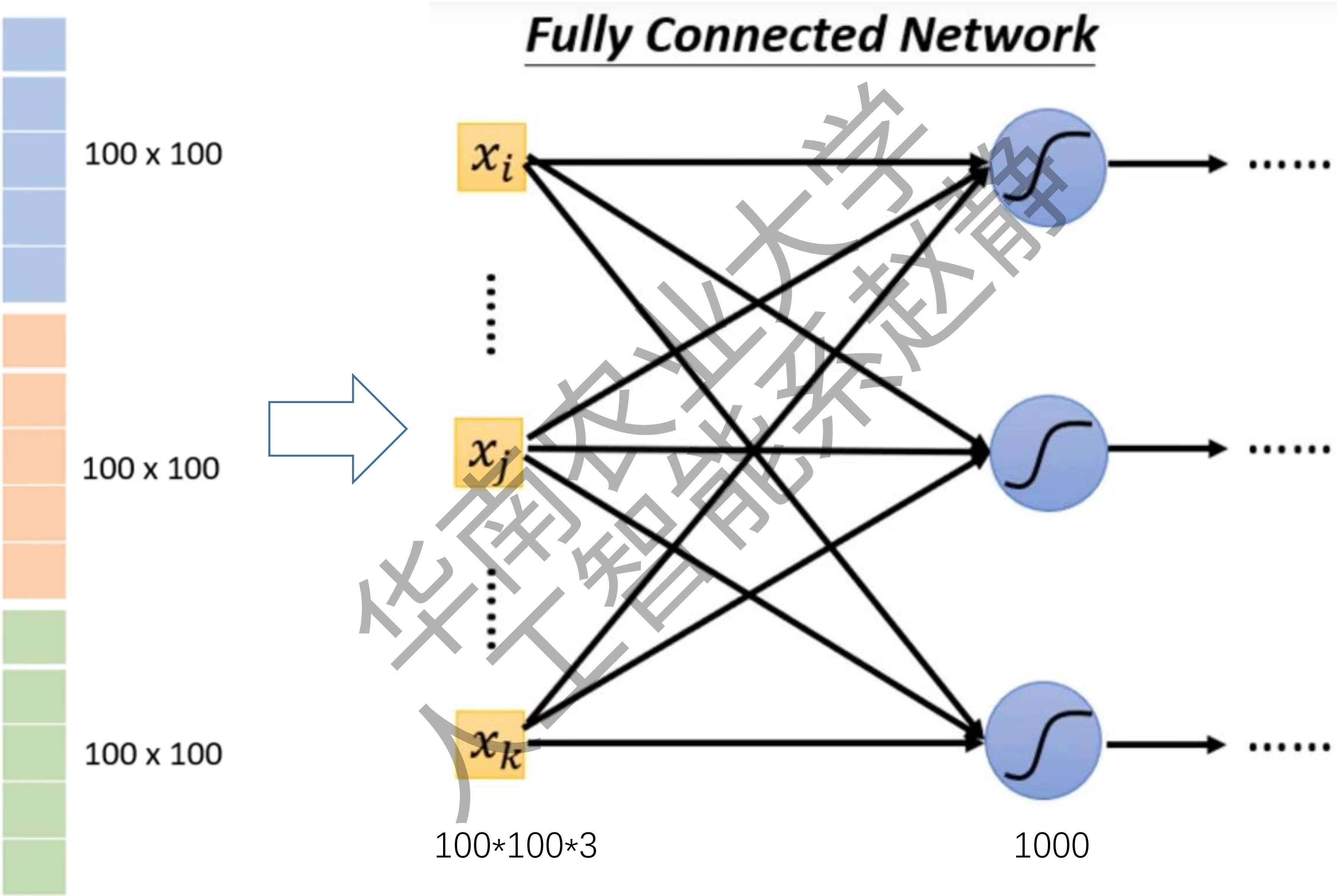
例： 图像

Binary

Gray Scale

Color

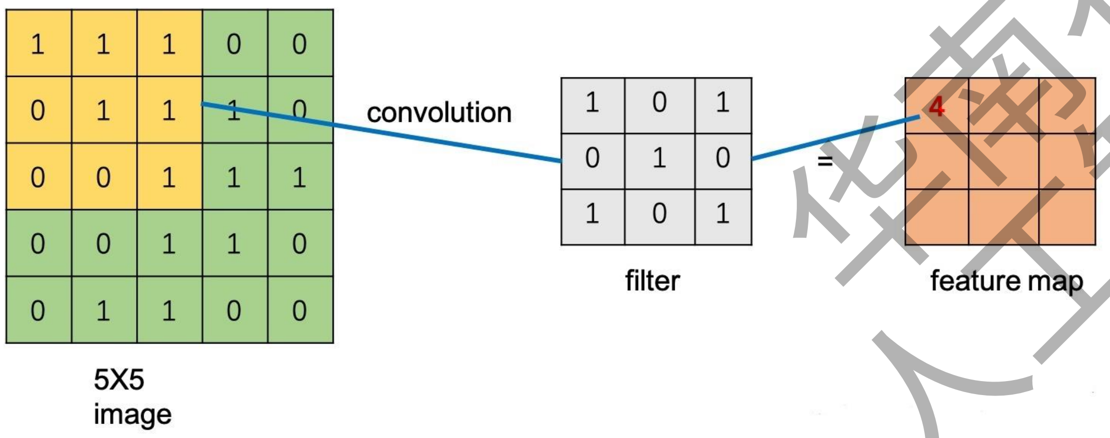

3-D
tensor
100 x 100
3 channels
100
100
100 x 100
100 x 100
100 x 100


Fully Connected Network
100 x 100
100 x 100
100 x 100
x_i
x_j
x_k
100*100*3
1000

➢ 卷积层

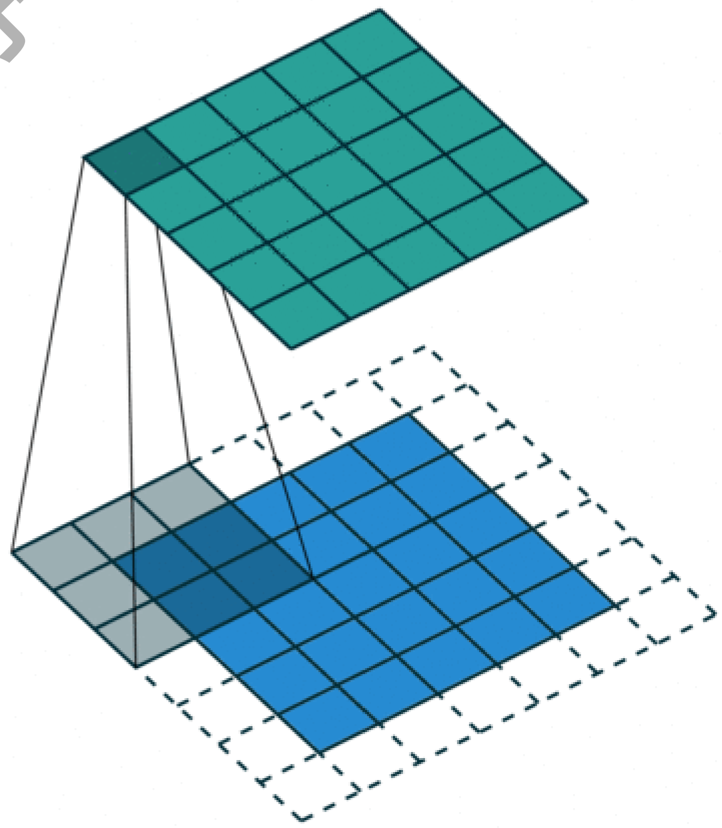
- **卷积**convolution

通过两个函数f和g生成第三个函数的一种数学算子

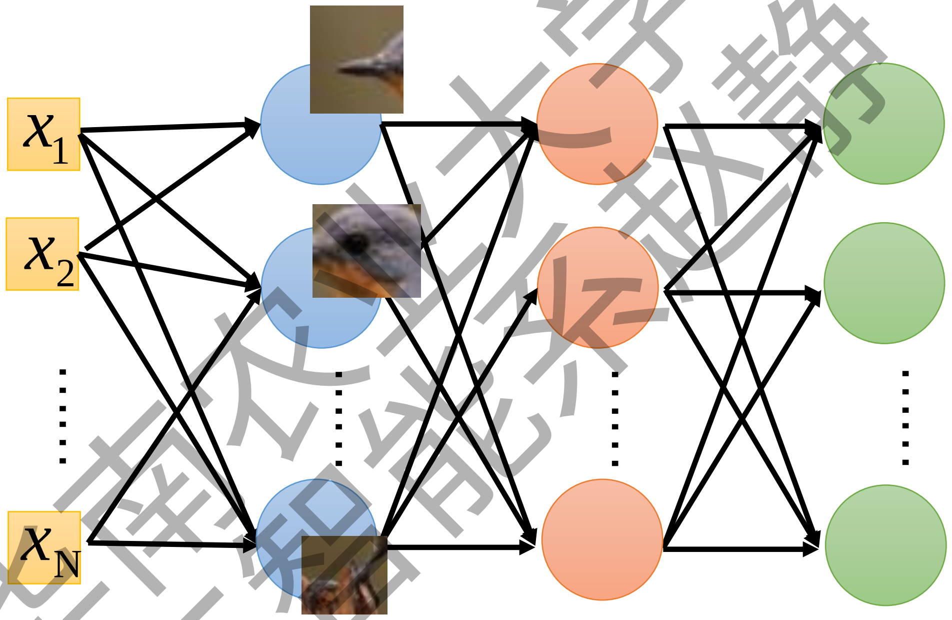


1	1	1
1	1	1
1	1	1

filter kernel



- **CNN**



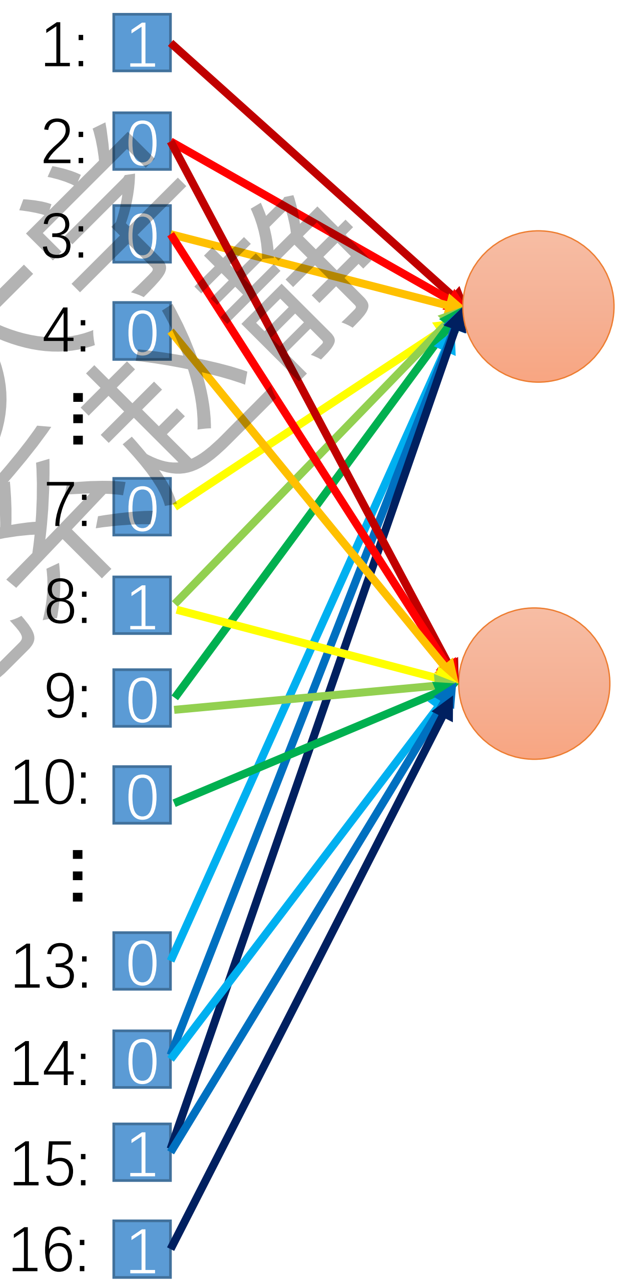
- 使用感受野，而非整幅图；
- 稀疏连接，而非全连接；
- 参数共享

- **感受野**

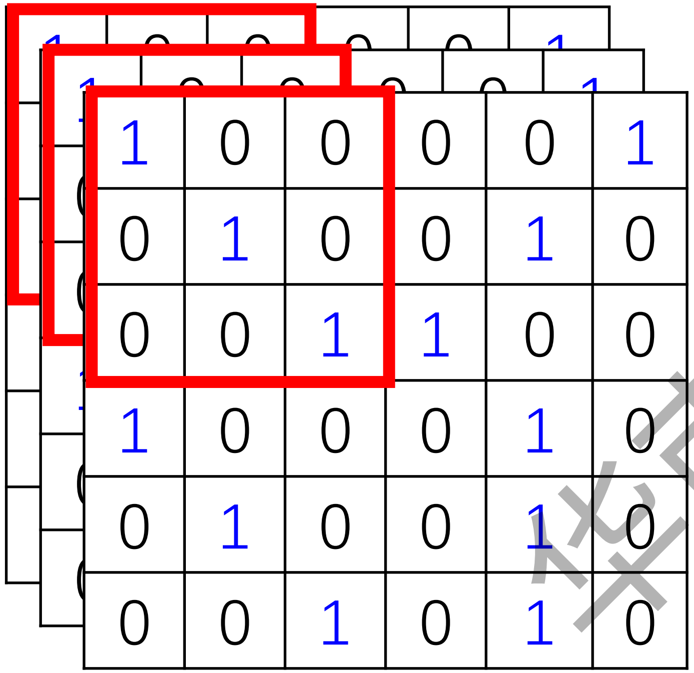
感受野大小： 3x3
Stride： 1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

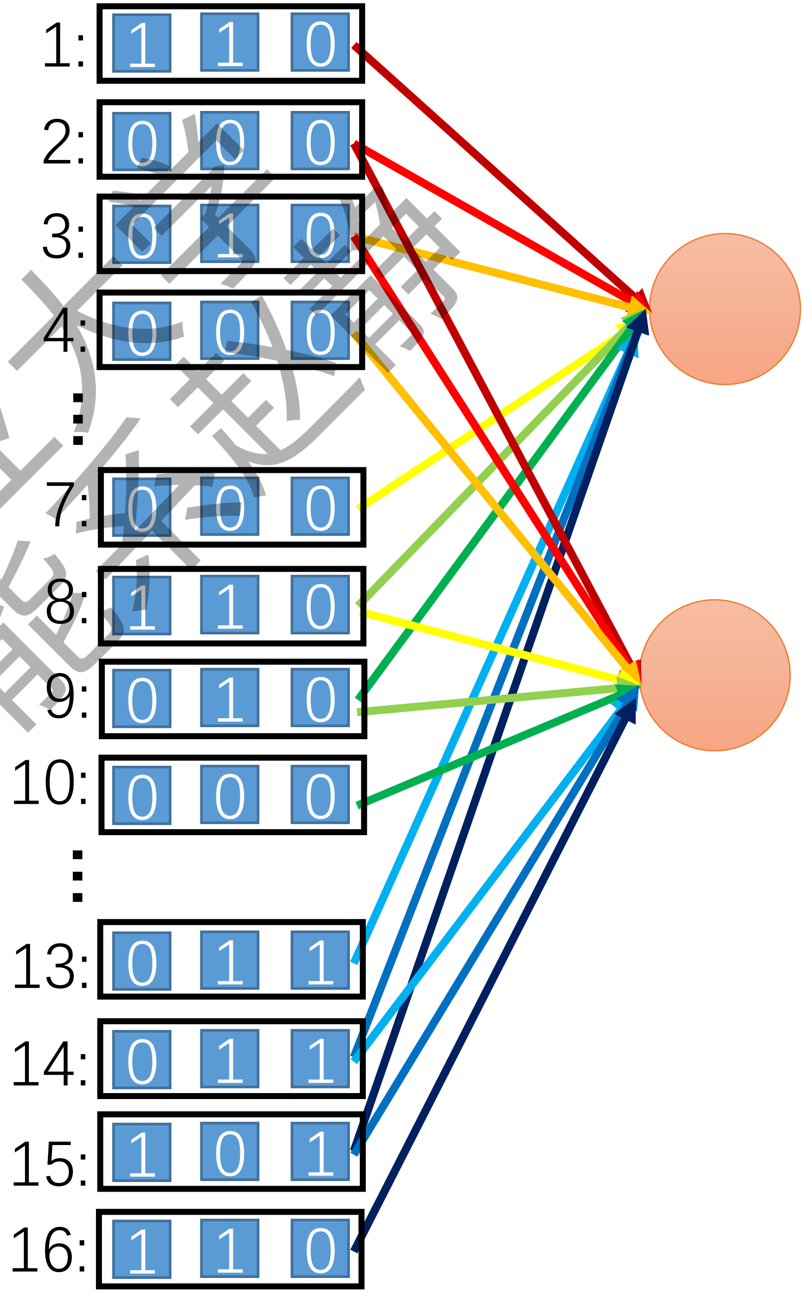
6 x 6 black & white
picture image



Size of Receptive field is 3x3x3, Stride is 1



6 x 6 RGB三个颜色通道



- **特征提取**

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

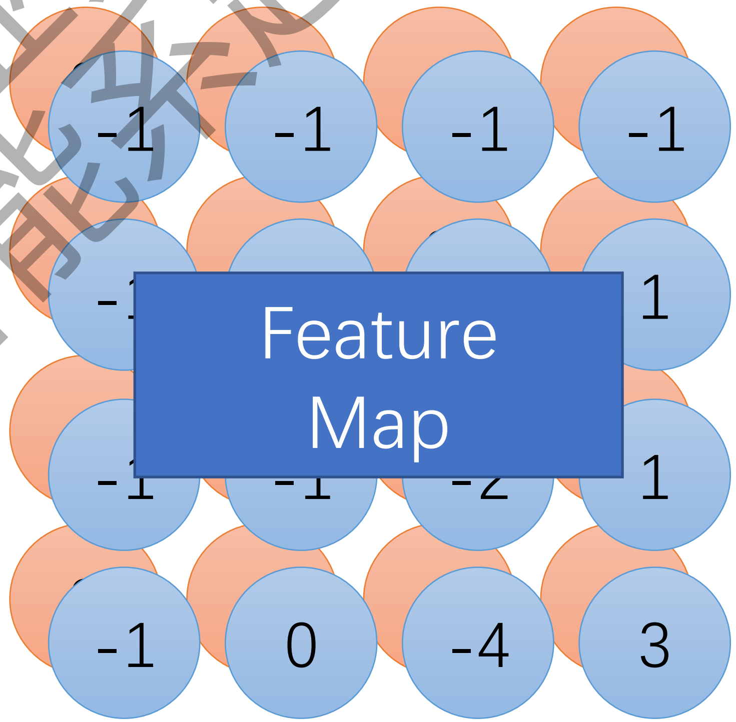
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

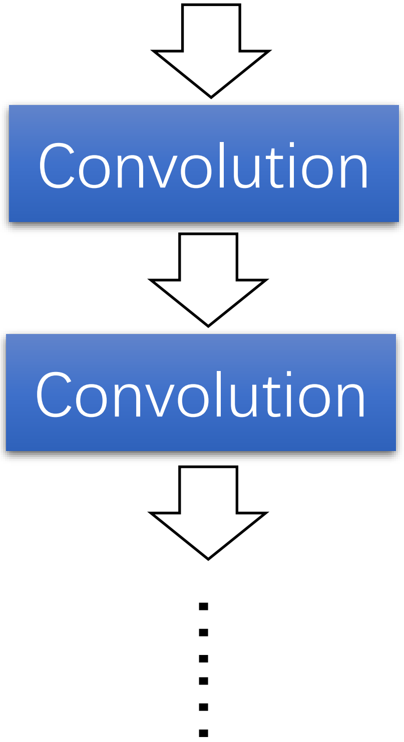
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

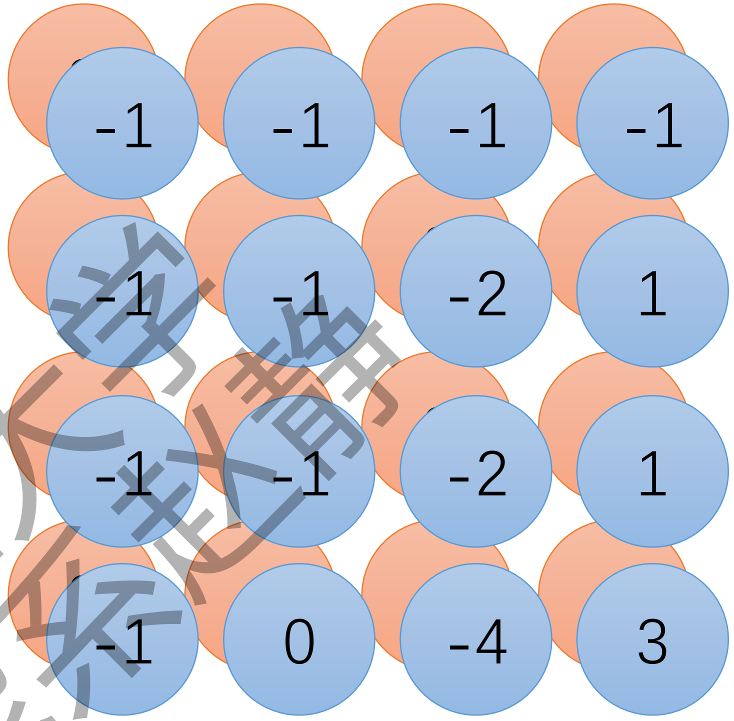
6 x 6 image



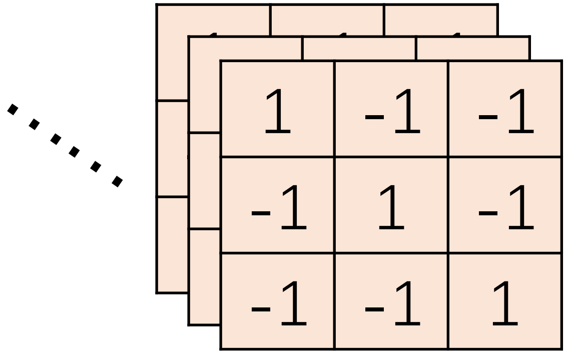
4 x 4 image



3*3*3 filter
64 filters



4*4，64 Channels



3*3*64 filter

卷积后Feature Map的宽度：$W_2 = \left\lfloor \frac{W_1 - F + 2P}{S} \right\rfloor + 1$

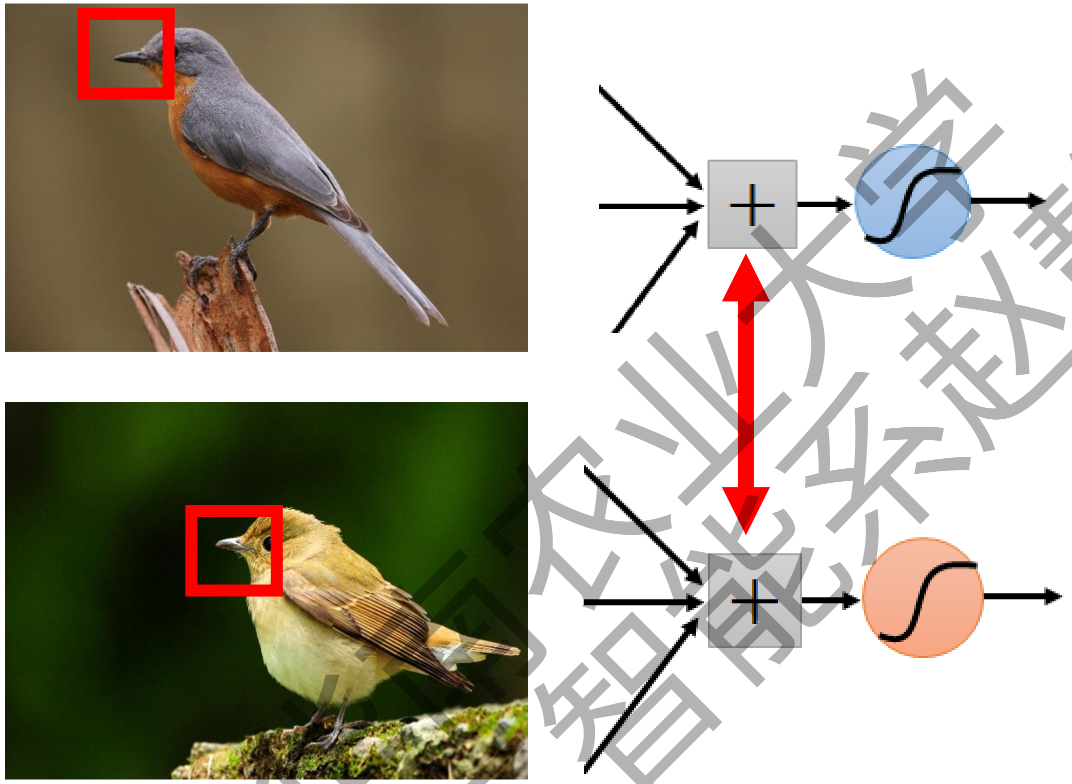
W_2： 卷积后Feature Map的宽度

W_1： 卷积前的输入图像的宽度

F： 卷积核宽度

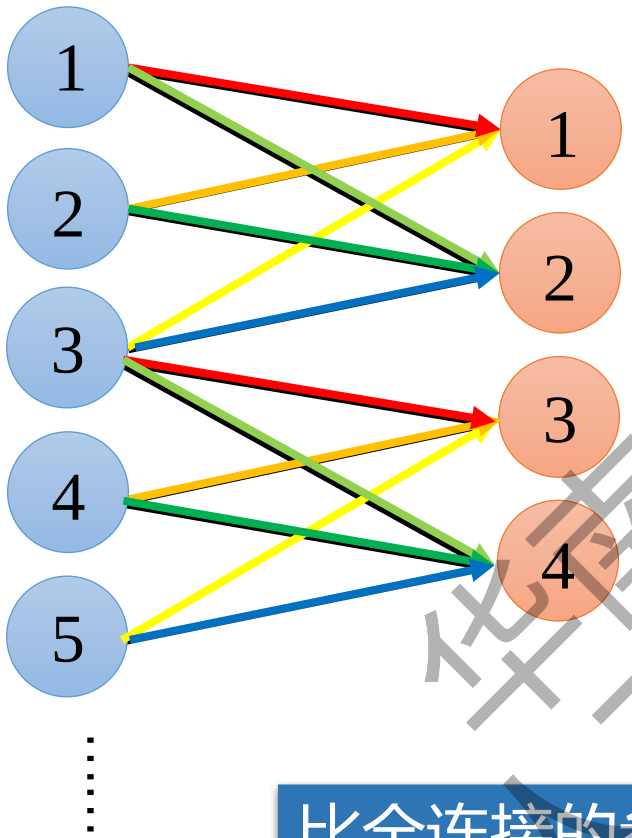
P： 单边填充区域列数

S： 步幅



空间位置改变

- 参数共享



稀疏连接

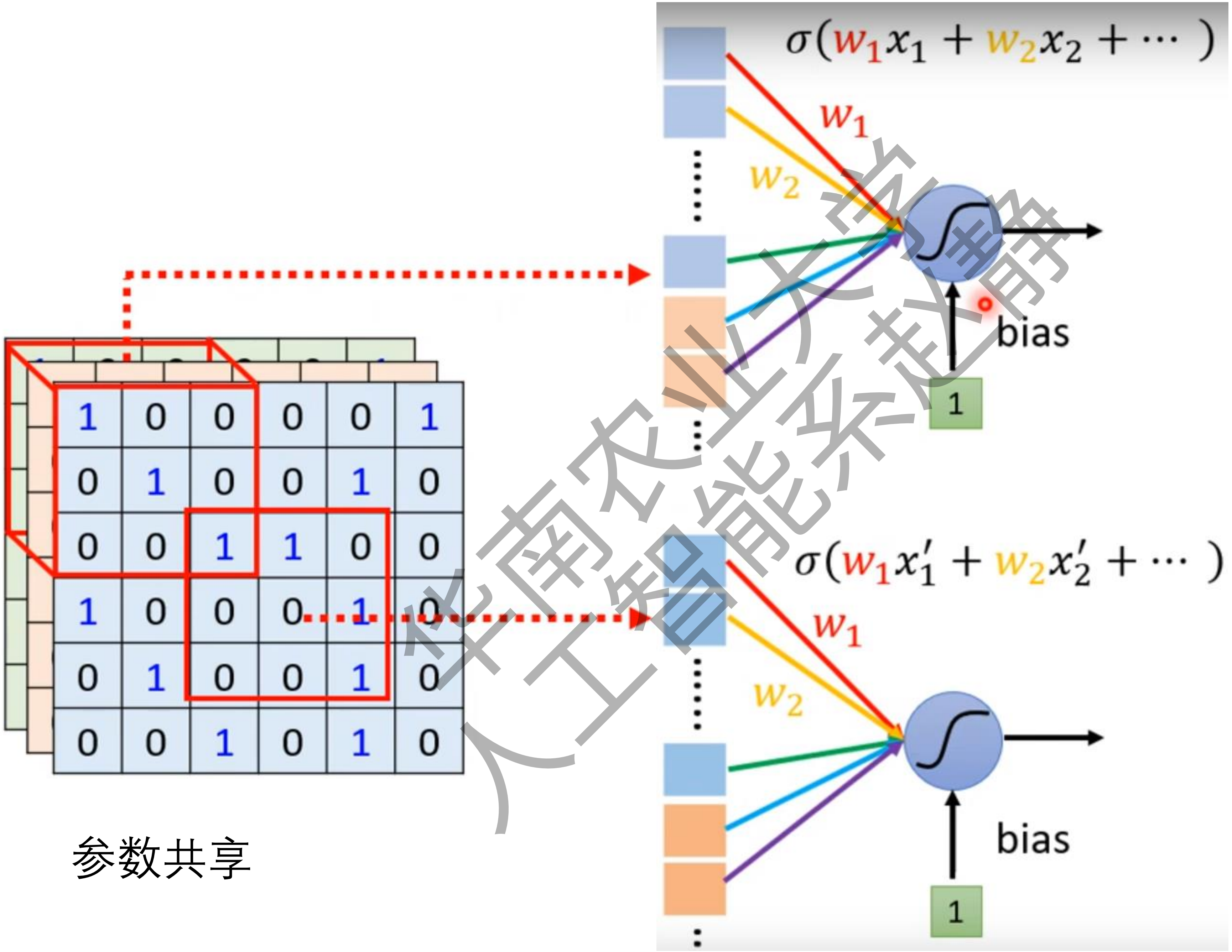
每个神经元只连接前一层部分神经元

参数共享

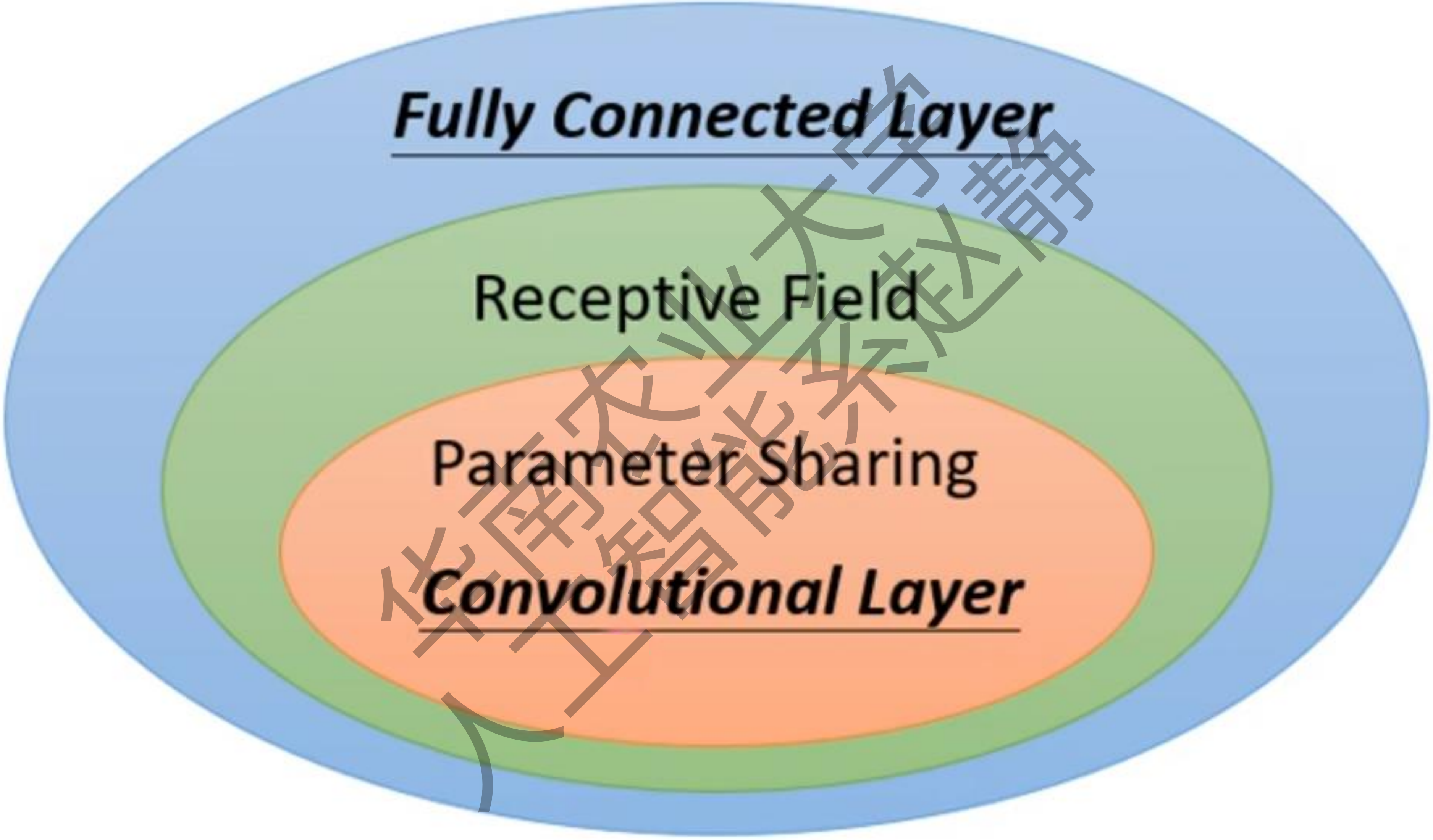
不同感受野的神经元使用相同的参数。

比全连接的参数少很多

核的大小、滤波器的数目、步幅都是开发者需要确定的参数。


$\sigma(w_1x_1 + w_2x_2 + \cdots)$
w_1
w_2
bias
1
$\sigma(w_1x_1' + w_2x_2' + \cdots)$
w_1
w_2
bias
1

参数共享


Fully Connected Layer
Receptive Field
Parameter Sharing
Convolutional Layer

➢ 池化层Pooling

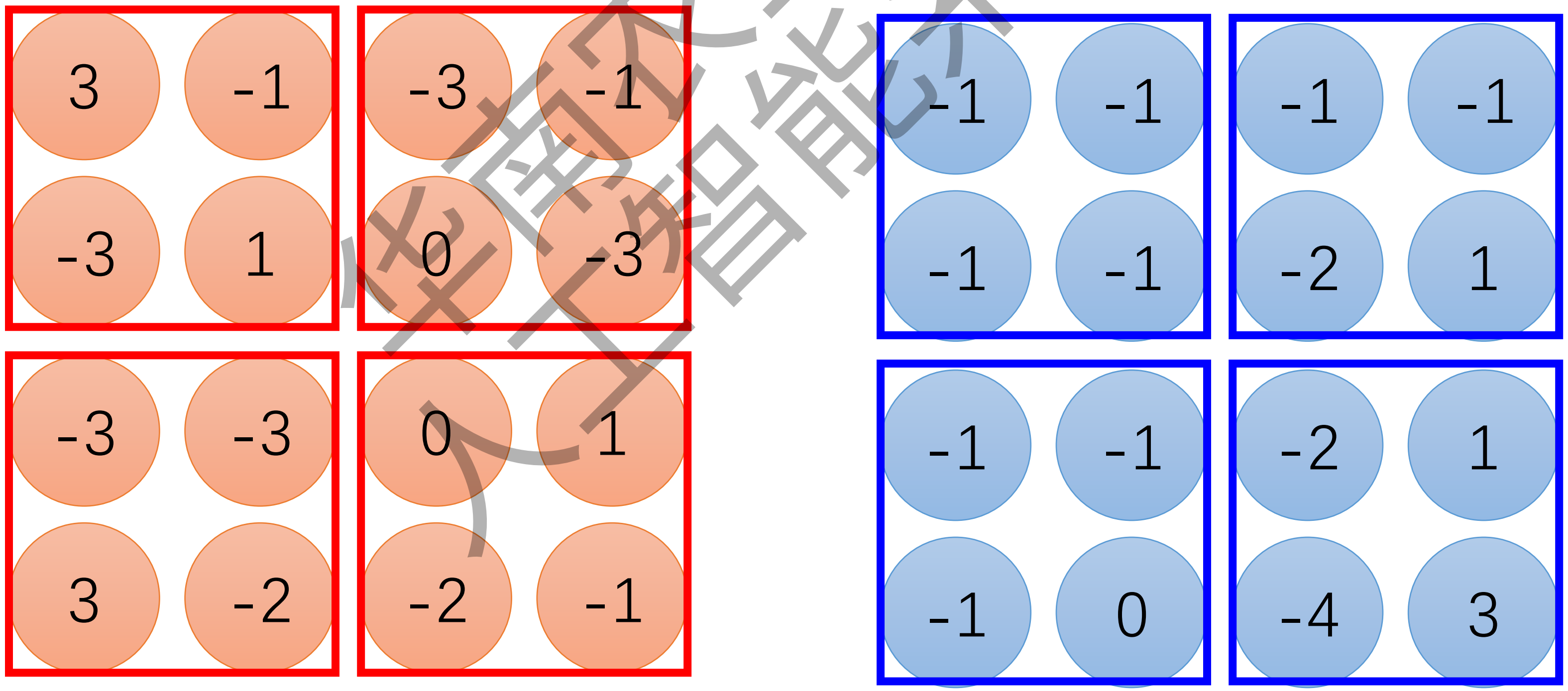
- **Max Pooling**

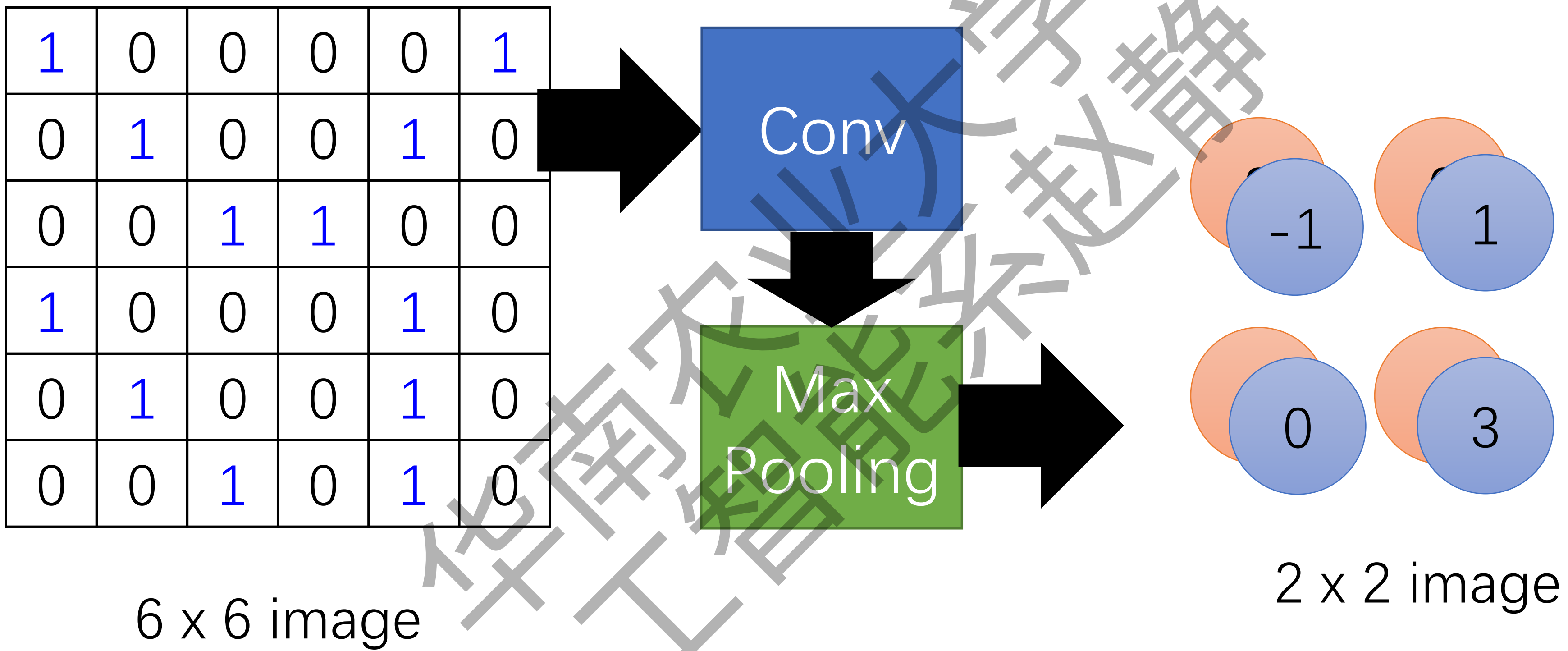
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2




1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Conv
Max Pooling
-1
1
0
3
2 x 2 image

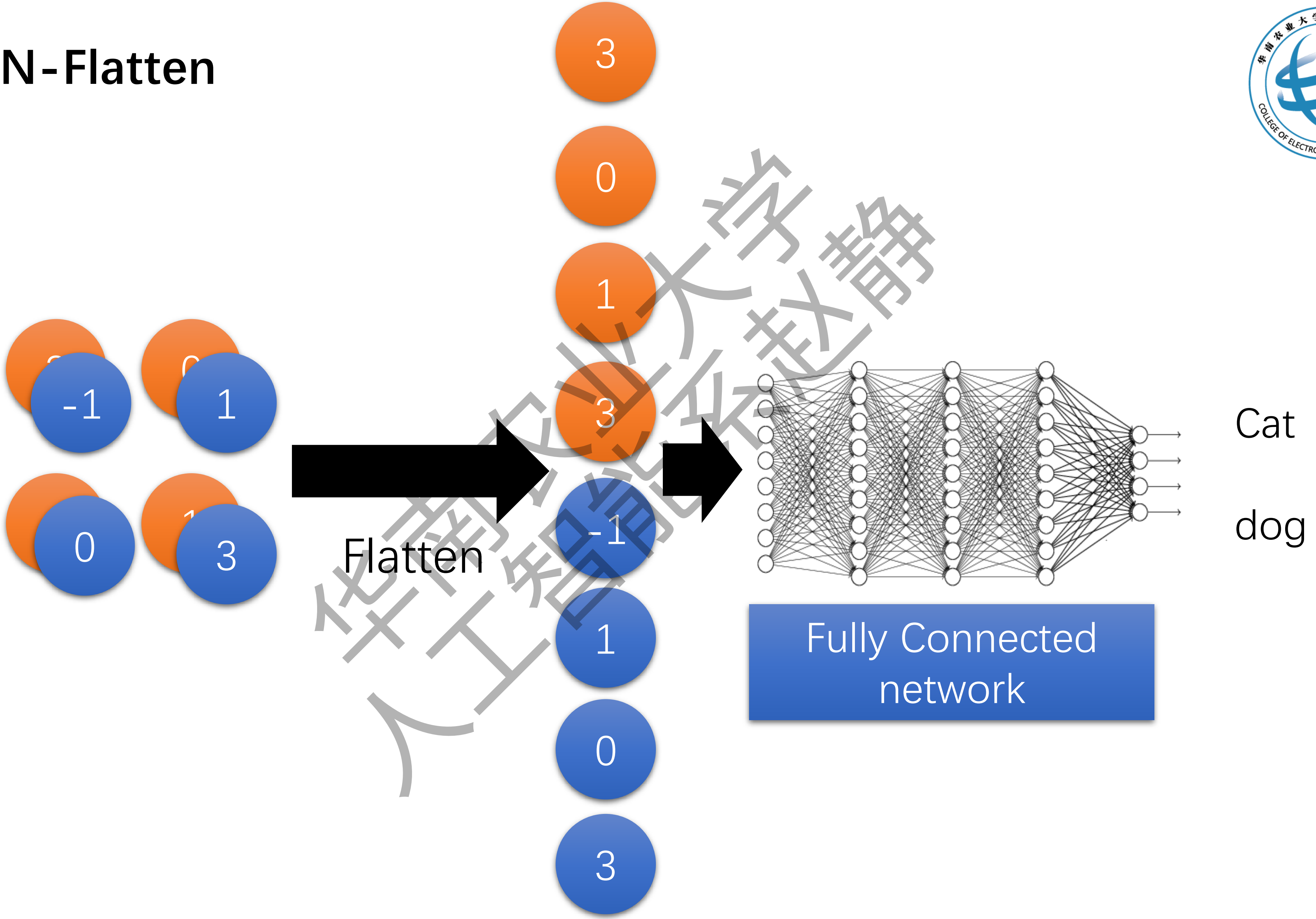
bird

bird

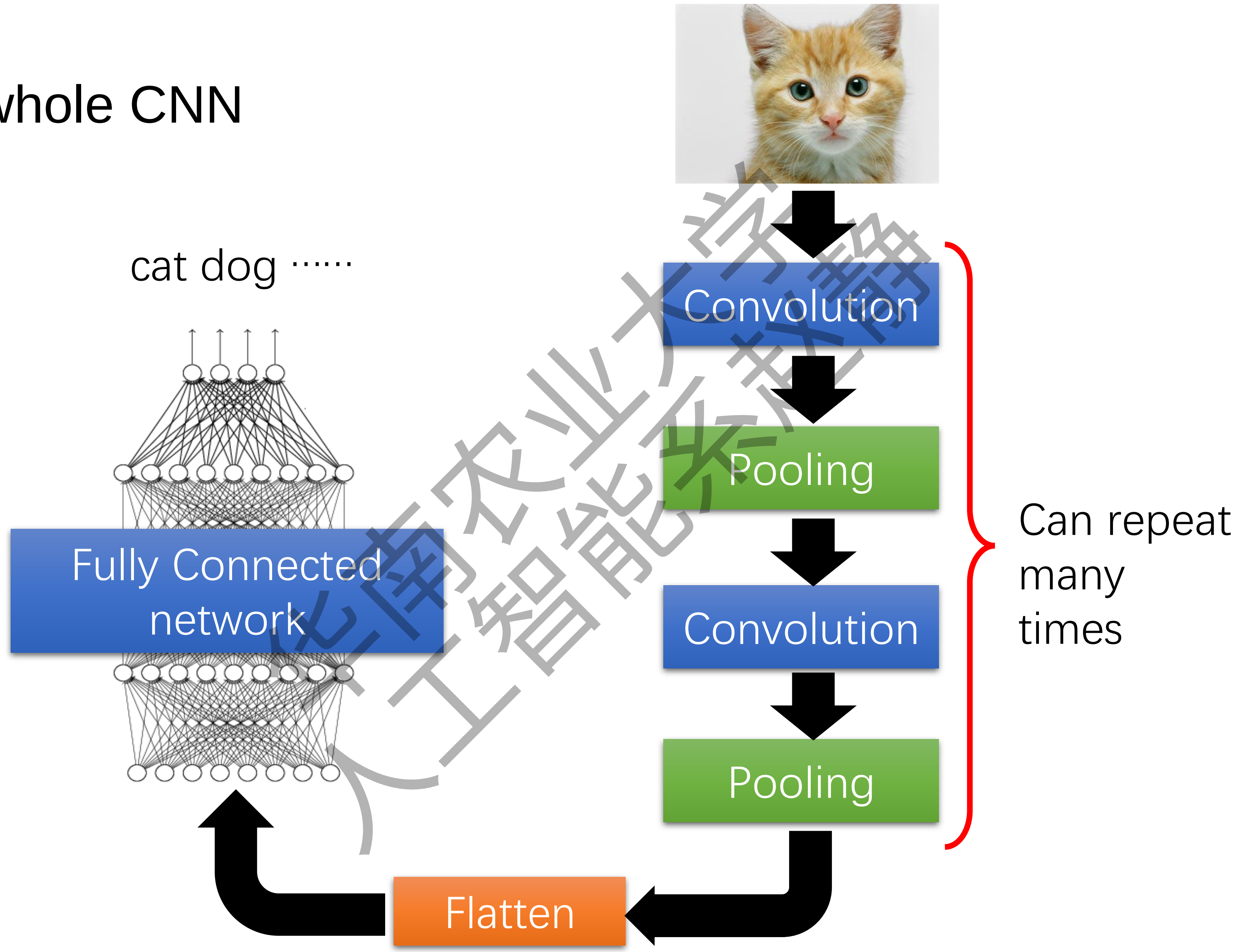
subsampling

尺度改变

CNN-Flatten

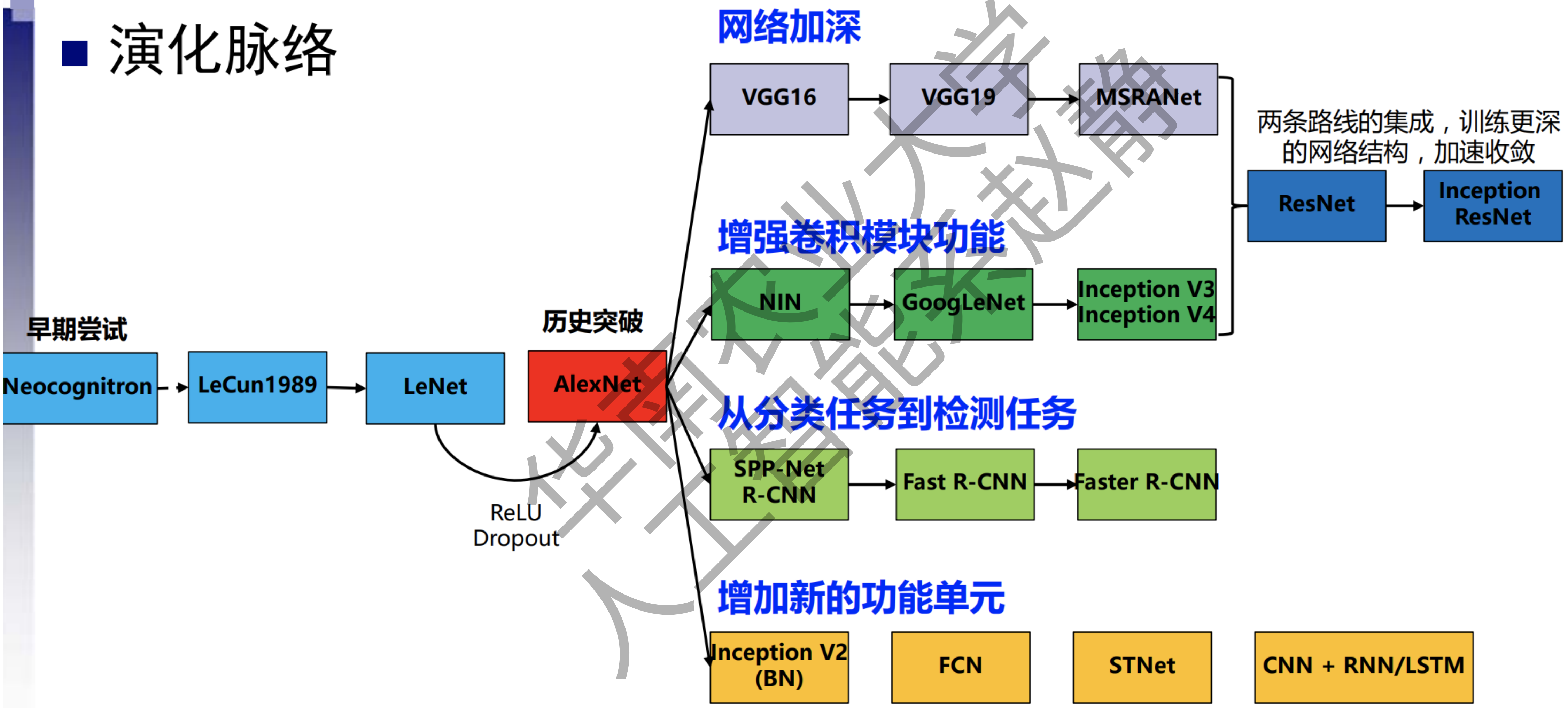


➢ The whole CNN

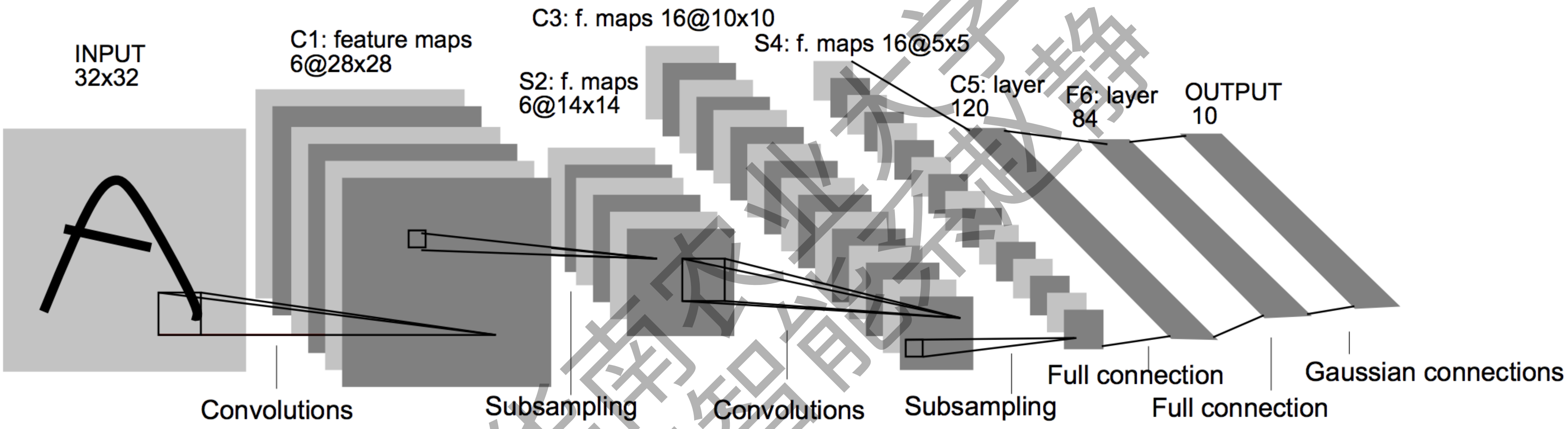


➢ 卷积神经网络CNN演化

■ 演化脉络



✓ LeNet5



C1、C3、C5为卷积层，S2、S4为池化层/下采样层，F6为全连接层，一个输出分类层Output

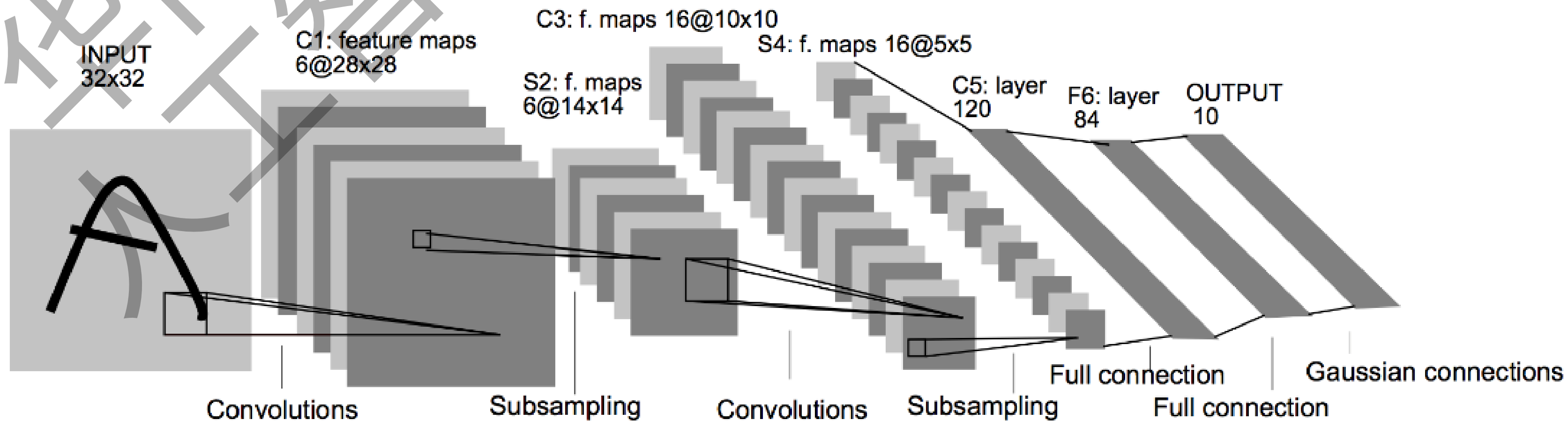
C1: 卷积层，共6个特征图（对应6个卷积核）

- 输入$\mathbf{x}$： 32×32
- 核$\mathbf{W}$大小： 5×5
- 步幅（Stride）：1×1
- 无填充
- 输出大小： 28×28×6

$$\mathbf{a}_f^{(1)} = \sigma\left(\mathbf{W}_f^{(1)} * \mathbf{x} + \mathbf{b}_f^{(1)}\right), \qquad f = 1,2,\dots,6$$

$$\left\lfloor \frac{32-5+0}{1} \right\rfloor + 1 = 28$$

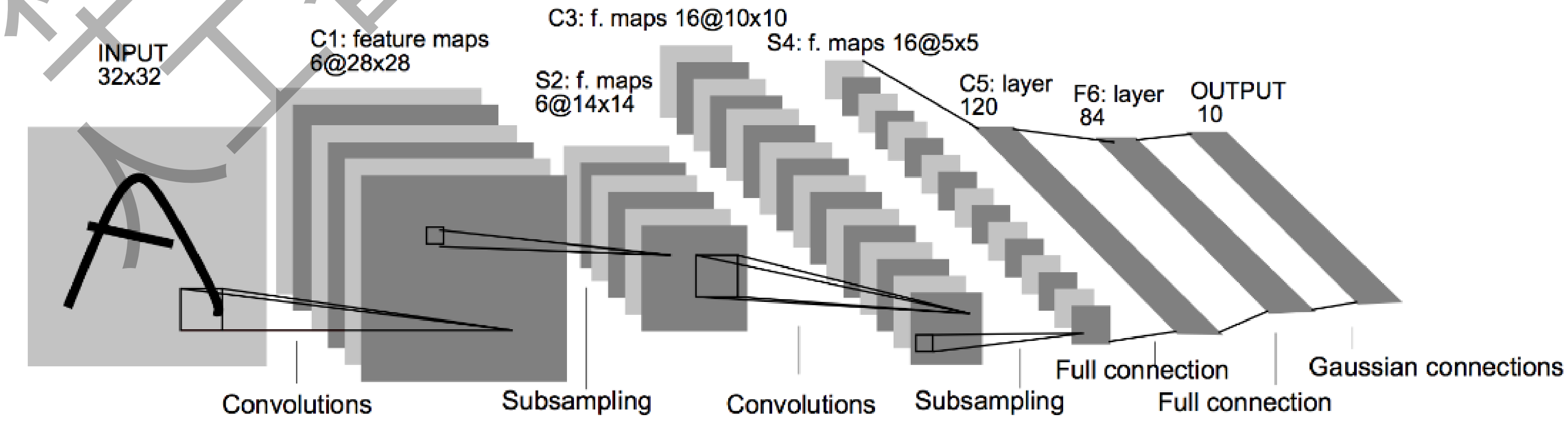
参数数目：(5x5+1)x6=156（其中5x5对应kernel size，1为bias，6为feature map 数目）



S2: 下采样层

- Kernel size: 2×2
- Stride: 2×2
- 把2×2的一个unit的所有数值相加，然后乘以系数，加上偏置bias，得出的结果再送入一个sigmoid函数作为最终这一层的输出。这里的系数和偏置都是可以训练的

参数数目：(1+1)x6=12个



C3: 卷积层

- Input size: 14×14×6
- Output channel: 16
- Kernel size: 5×5
- Stride: 1×1
- Output size: 10×10×16
- C3与S2并不是全连接而是部分连接，通过这种方式提取更多特征

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

3 4 6

C3层参数数目：

(5×5×3+1) ×6 + (5×5×4+1) ×9 +(5×5×6+1) = 1516

S4: 下采样层

C5: 卷积层（120个卷积核）

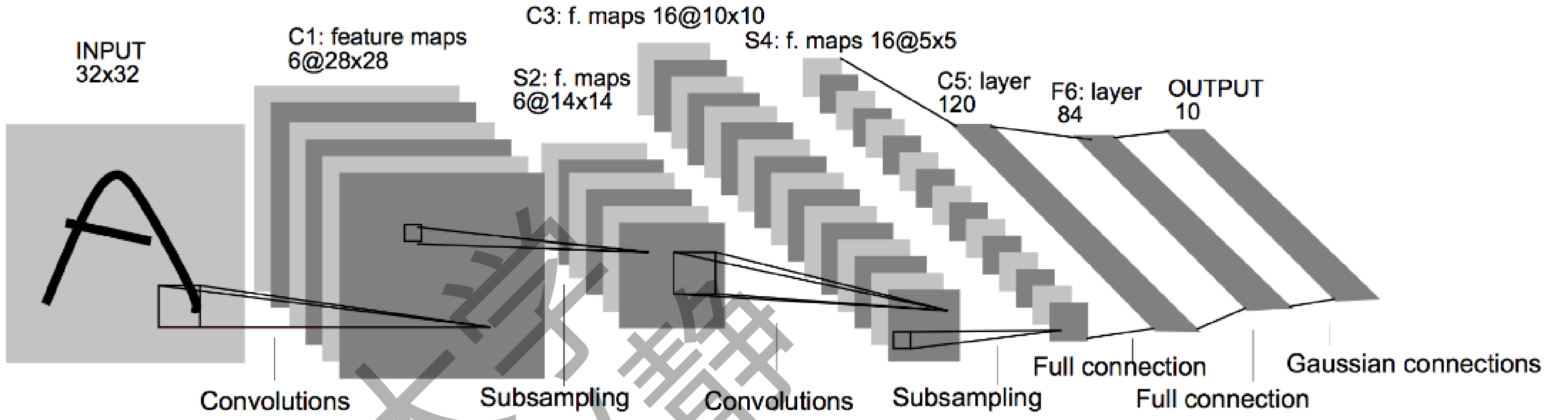
- Input size: 5×5×16
- Output channel: 120
- Kernel size: 5×5
- Stride: 1×1
- Output size: 1×1×120

参数数目：120×（5×5×16+1）= 48120

F6: 全连接层

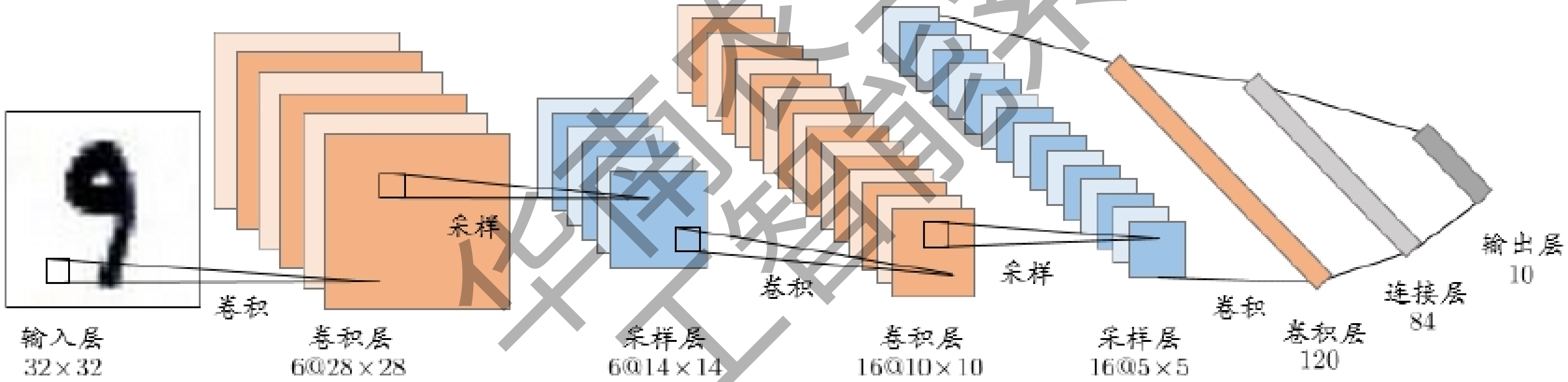
- 输入：120
- 输出：84

参数数目：（120+1）×84=10164



输出层：全连接

- 输入：84
- 输出：10
- 该层采用径向基函数（RBF）的网络连接方式，



卷积神经网络用于手写数字识别 [LeCun et al., 1998]