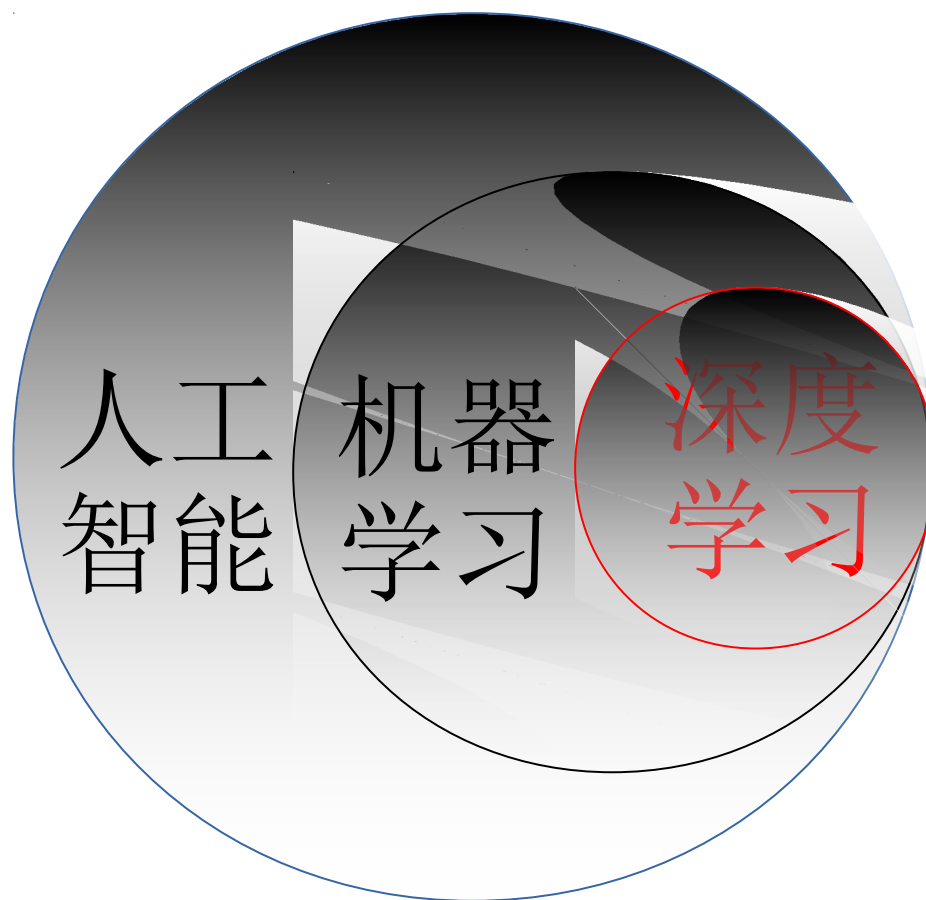


# 前馈神经网络

赵永红

四川师范大学物理系



机器如何学习？ → 人类如何学习？

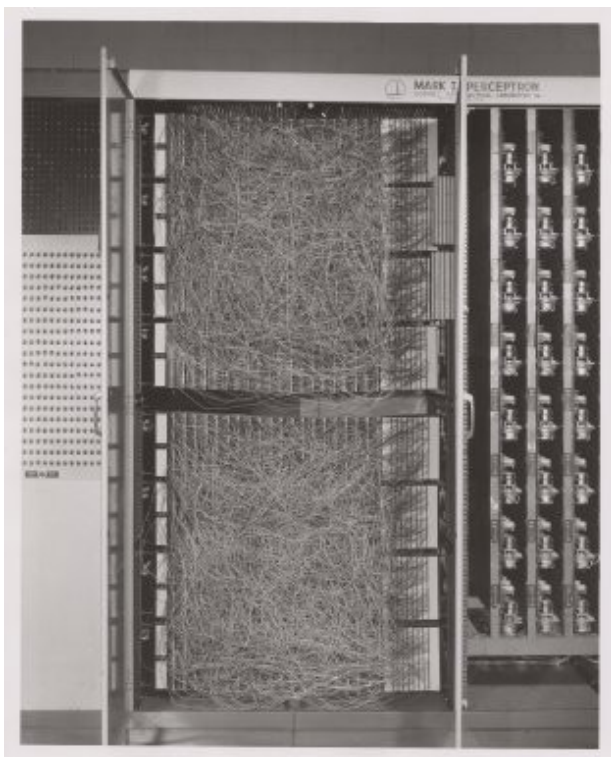
# 如何描述复杂函数？

	经典 机器学习	深度学习
学习复杂函数	X	网络层数 神经元个数
复杂化学环境	TB	Plane wave

# 前馈网络：基石

应用	图像识别	自然语言处理
神经网络	卷积网络	循环网络

# 感知机： Perceptron



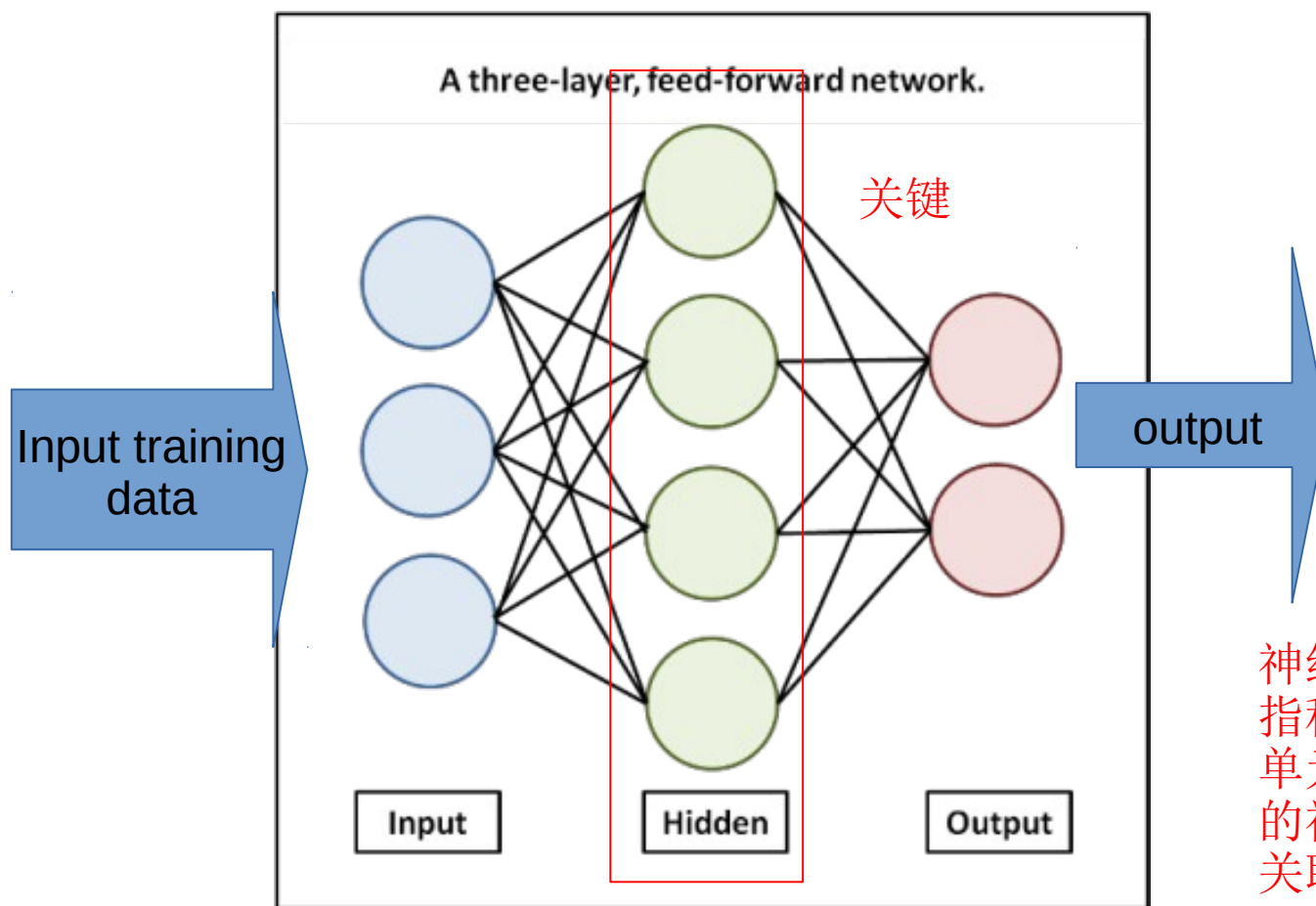
硬件实现：  
美国海军， Frank Rosenblatt



软件实现：  
IBM 704

# 基本概念：神经网络

$$y = f(x; \theta)$$

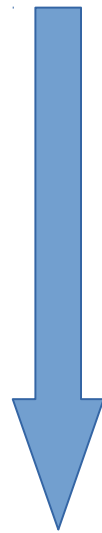


神经网络：形象指称互联的计算单元，与生物学的神经网络并无关联。

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

# 非线性：深度学习之前

- 采用“核变换”的思想， $x \rightarrow \phi(x)$   $z = \mathbf{w}^T \phi(\mathbf{x}) + b$
- 如果是普适性的  $\phi(x)$ ，针对特定问题效果差；
- 定制化的  $\phi(x)$ ，设计困难；



$\phi_1(\mathbf{x}) = x_1$	$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$	$\phi_3(\mathbf{x})$	$t$
$\phi_2(\mathbf{x}) = x_2$	0	0	0	0
$\phi_3(\mathbf{x}) = x_1 x_2$	0	1	0	1
	1	0	0	1
	1	1	1	0

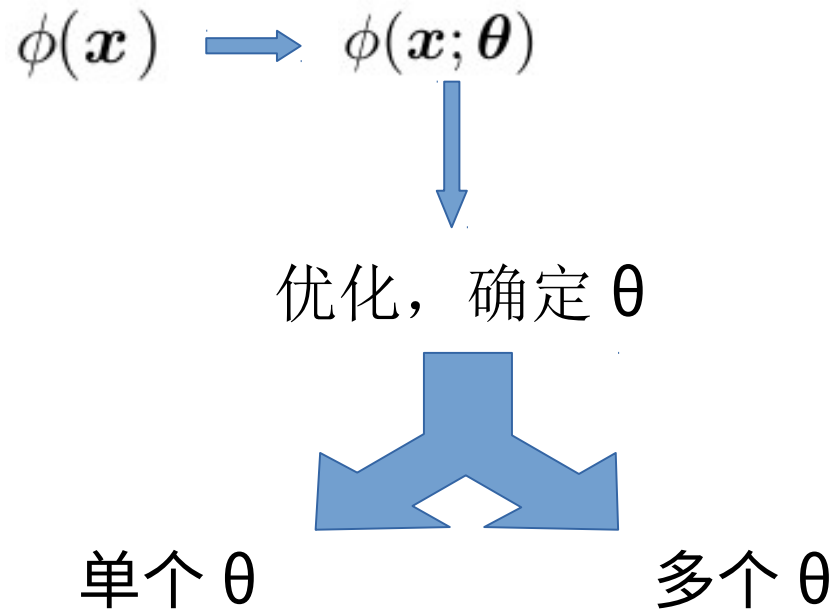
$$z = \mathbf{w}^T \phi(\mathbf{x}) + b$$

$$b = -0.5 \quad w_1 = 1 \quad w_2 = 1 \quad w_3 = -2$$

- 通过“学习”得到有针对性的解决方案；

$$y = f(x; \theta, w) = \phi(x; \theta)^T w$$

# 如何学习？



通过多个  $\theta$ ，提取多层次信息，更符合人的决策习惯。



# 再论：异或

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \xrightarrow[\begin{matrix} y = f^*(\mathbf{x}) \end{matrix}]{\begin{matrix} y = f(\mathbf{x}; \boldsymbol{\theta}) \end{matrix}} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

损失函数

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbb{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \boldsymbol{\theta}))^2$$

# 线性模型

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{x}^\top \mathbf{w} + b \quad \text{普适的线性函数!}$$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}, x_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, x_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, x_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, x_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{aligned} J(; w_1, w_2, b) = & b^2 \\ & + [1 - 2(w_2 + b) + (w_2 + b)^2] \\ & + [1 - 2(w_1 + b) + (w_1 + b)^2] \\ & + (w_1 + w_2 + b)^2 \end{aligned}$$

$$\frac{\partial J}{\partial w_1} = 0$$

$$\frac{\partial J}{\partial w_2} = 0$$

$$\frac{\partial J}{\partial b} = 0$$

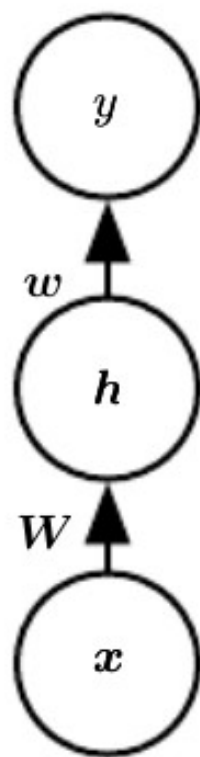
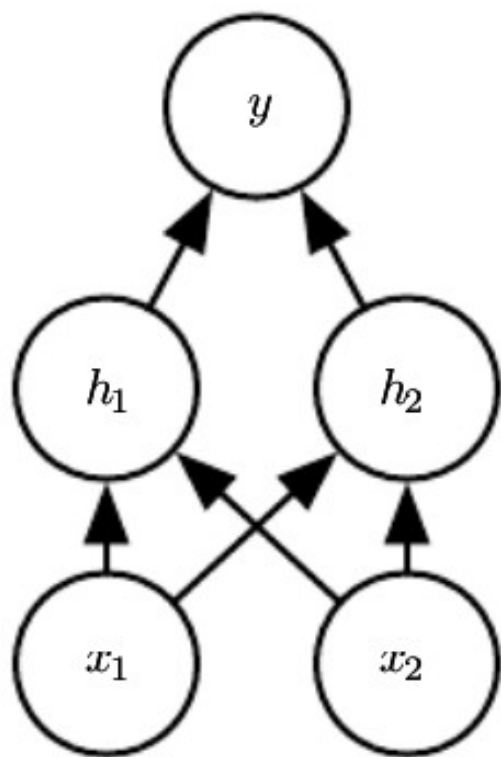


$$w_1 = w_2 = 0$$

$$b = \frac{1}{2}$$

无法求解!

# 前馈网络

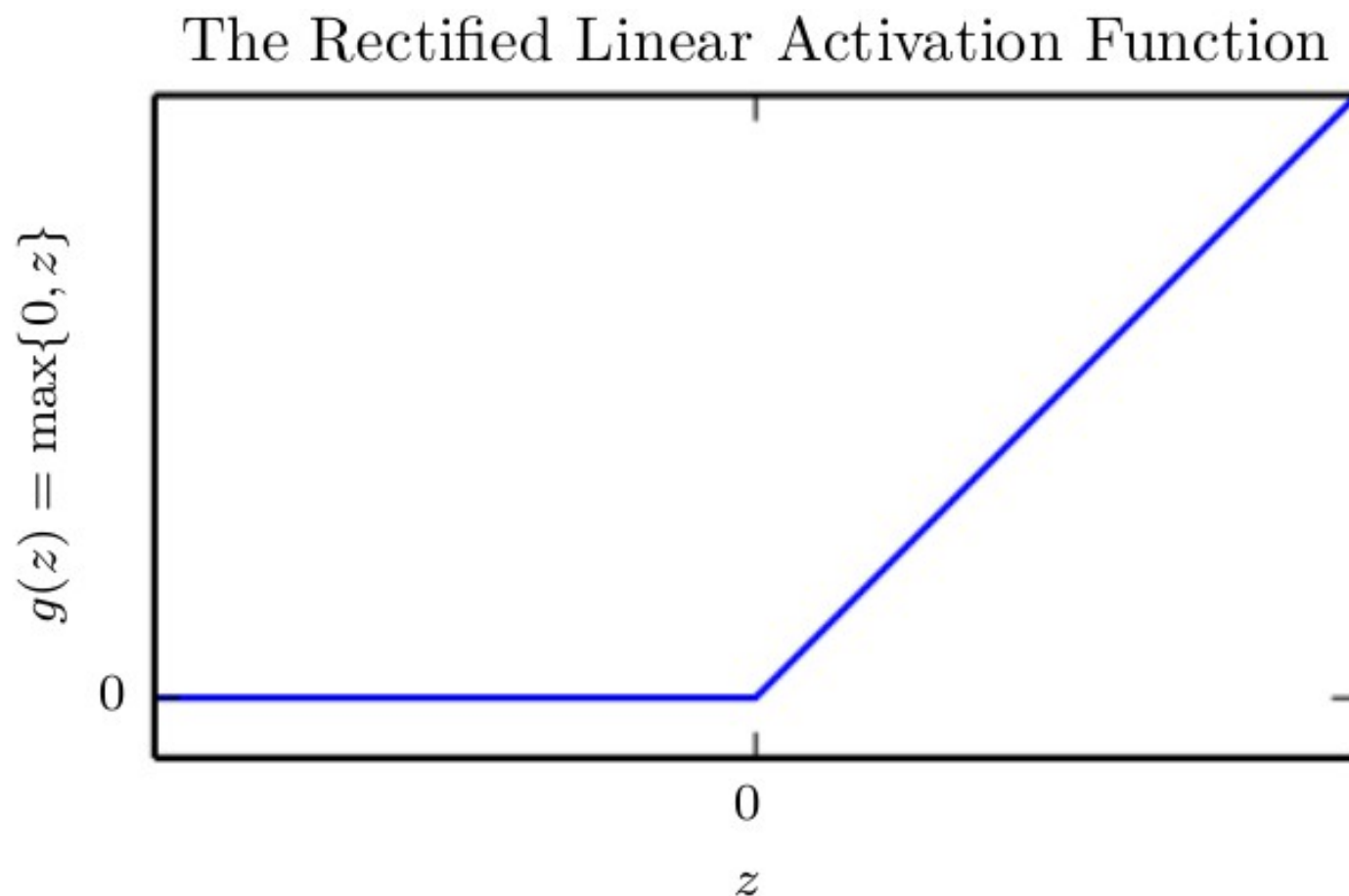


$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c})$$

$$y = f^{(2)}(\mathbf{h}; \mathbf{w}, b)$$

$$\begin{aligned} f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) &= f^{(2)}(f^{(1)}(\mathbf{x})) \\ &= \mathbf{w}^\top \max\{0, \mathbf{W}^\top \mathbf{x} + \mathbf{c}\} + b. \end{aligned}$$

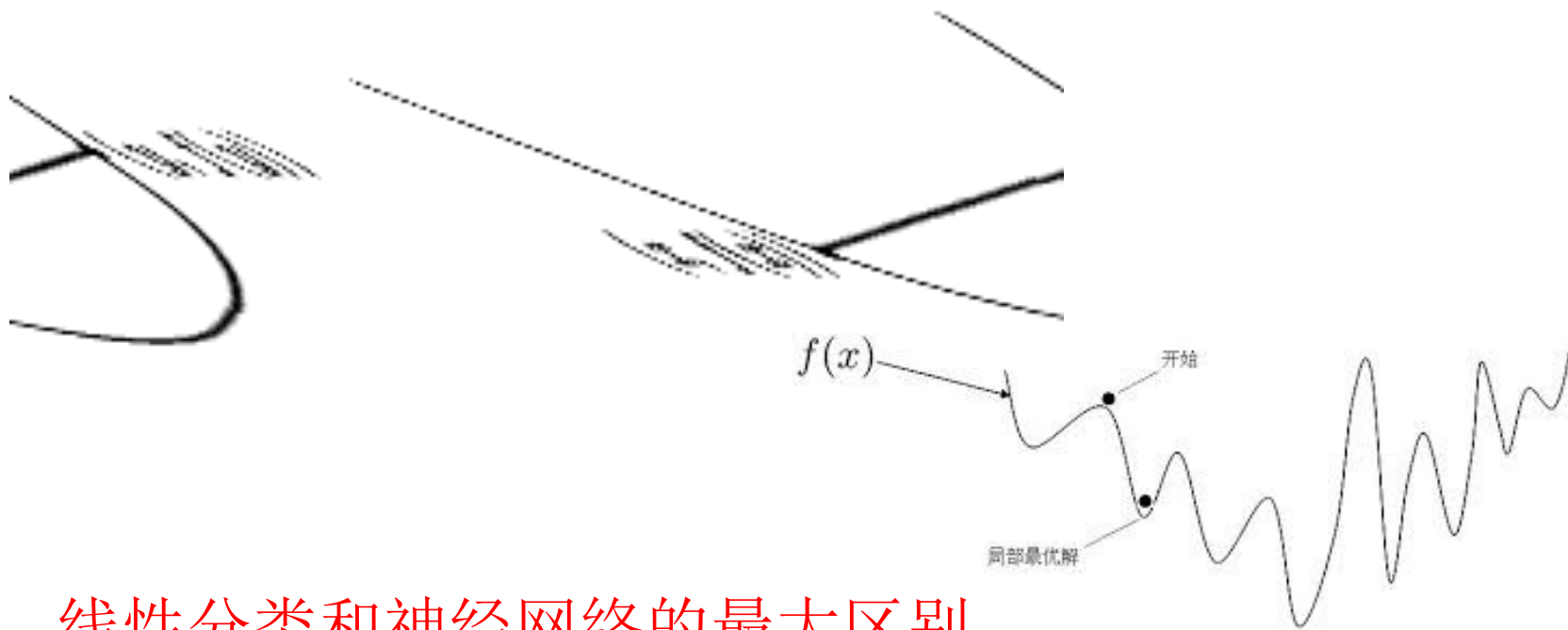
# 整流（rectify）函数、激活函数



普适的非线性函数！

多姿多彩的  
非线性 = 普适的  
非线性 + 普适的  
线性 + 神经网络

# 凸函数



线性分类和神经网络的最大区别  
在于损失函数是非凸函数；

非凸函数优化的初值敏感性问题；

# 凸 ( convex ) 优化

	凸优化	非凸优化
机器学习方法	线性回归、逻辑回归、支持向量机	深度学习
收敛性	有保证	无保证
初值敏感性	不敏感	敏感 ( $w$ 初始化为小的随机量, $b$ 初始化为小的正数 )

# 梯度下降法

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{E}}{\partial w_j}$$

$$w_j \leftarrow w_j - \alpha \frac{1}{N} \sum_{i=1}^N x_j (y^{(i)} - t^{(i)})$$

所有机器学习算法都  
采用类似的优化方法，  
如何高效计算梯度？



**BP 算法！**

**Learning representations  
by back-propagating errors**

David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

第二次神经  
网络热潮



# 复习：最大似然原理

已知  $f_D$  和  $\theta$  可得抽样概率： $\mathbb{P}(x_1, x_2, \dots, x_n) = f_D(x_1, \dots, x_n \mid \theta)$

问题：已知抽样  $X_n$  和分布  $f_D$ ，如何得到  $\theta$ ？

定义似然函数  $like(\theta)$



最大化似然函数



$$\Theta_{ML} = \operatorname{argmax}_{\theta} like(\theta)$$

# 最大似然和交叉熵等价

型学习就是在给定的训练数据条件下对模型进行极大似然估计或正则化的极大似然估计。

证明：

[http://www.52caml.com/head\\_first\\_ml/ml-chapter2-entropy-based-family/](http://www.52caml.com/head_first_ml/ml-chapter2-entropy-based-family/)

# 负对数价格（cost）函数

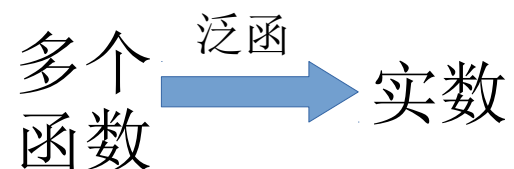
$$p(\mathbf{y} \mid \mathbf{x}) \xrightarrow[\text{似然}]{\text{最大}} \log p(\mathbf{y} \mid \mathbf{x})$$

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{y} \mid \mathbf{x})$$

$$J(\theta) = \frac{1}{2} \mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} \|\mathbf{y} - f(\mathbf{x}; \boldsymbol{\theta})\|^2 + \text{const}$$

通过梯度下降，确定  $\theta$  值，  
从而确定模型  $f(\mathbf{y}; \theta)$ 。

# 价格函数，价格泛函？



“学习”：从多个函数中学习出一个函数，因此是一个泛函的变分最小化的过程；

# 其它价格函数

$$C_{MST}(W, B, S^r, E^r) = 0.5 \sum_j (a_j^L - E_j^r)^2$$

$$C_{CE}(W, B, S^r, E^r) = - \sum_j [E_j^r \ln a_j^L + (1 - E_j^r) \ln (1 - a_j^L)]$$

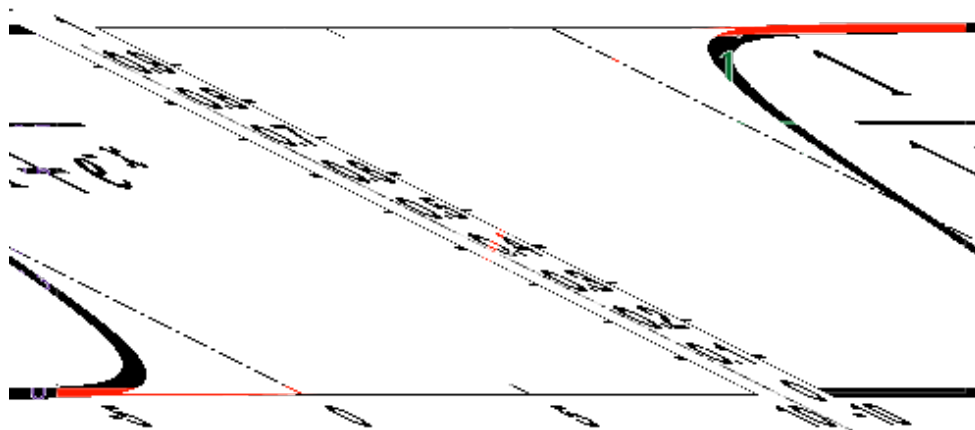
$$C_{EXP}(W, B, S^r, E^r) = \tau \exp\left(\frac{1}{\tau} \sum_j (a_j^L - E_j^r)^2\right)$$

$$C_{HD}(W, B, S^r, E^r) = \frac{1}{\sqrt{2}} \sum_j (\sqrt{a_j^L} - \sqrt{E_j^r})^2$$

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

理想与现实之间  
距离一个统计！

# 优化：梯度先行



负对数价格函数可防止由于激活函数饱和而带来的梯度消失问题。

# 神经网络设计

输出层 + 隐藏层 + 网络结构

# 输出神经元与分布函数

线性

$$\hat{\mathbf{y}} = \mathbf{W}^\top \mathbf{h} + b$$

$$p(\mathbf{y} | \mathbf{x}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, \mathbf{I})$$

二分类

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{h} + b)$$

$$P(y) = \sigma((2y - 1)z)$$

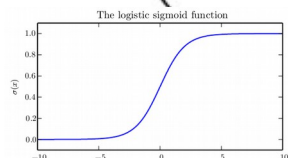


Figure 3.3: The logistic sigmoid function.

多分类

$$\hat{y}_i = P(y = i | \mathbf{x})$$

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

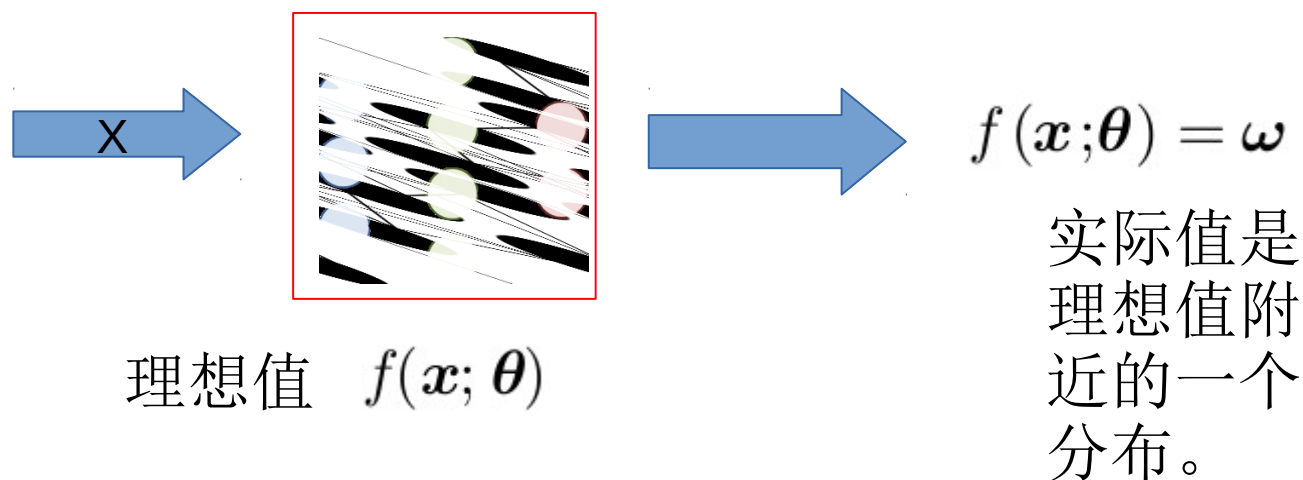
最大似然原则：提供一种通用的构造  
价格函数的方法。

$$p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta}) \longrightarrow -\log p(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta})$$

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$$



# 输出层：理解和推广



---

概率分布  $p(y | x; \theta)$

损失函数  $-\log p(y | x; \theta)$

价格函数：损失函数对训练样本的算术平均；

深度学习中的输出神经元和其它机器学习模型一致；

深度学习的核心在于隐藏层的设计；

# 神经网络设计

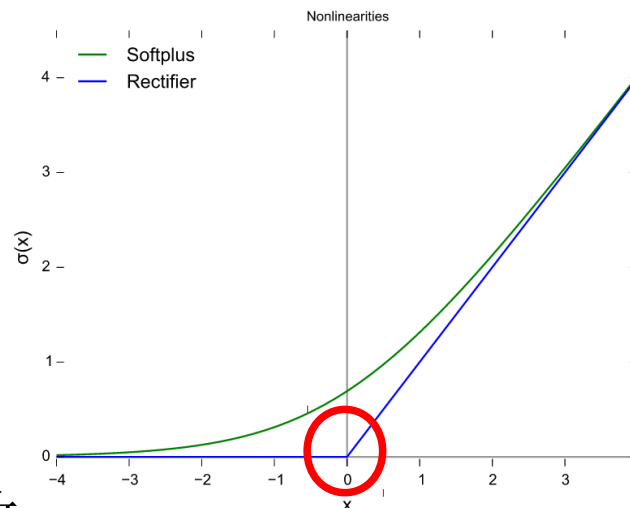
输出层 + 隐藏层 + 网络结构

# 通往深度网络： 隐藏层

## 1、线性函数

$$z = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$$

## 2、非线性激活函数



不连续性和梯度算法，忽略！一般不会达到该点。

ReLU

$$g(z) = \max\{0, z\}$$

Softplus

$$f(x) = \ln(1 + e^x)$$

RBF

$$h_i = \exp\left(-\frac{1}{\sigma_i^2} \|\mathbf{W}_{:,i} - \mathbf{x}\|^2\right)$$

Hard tanh

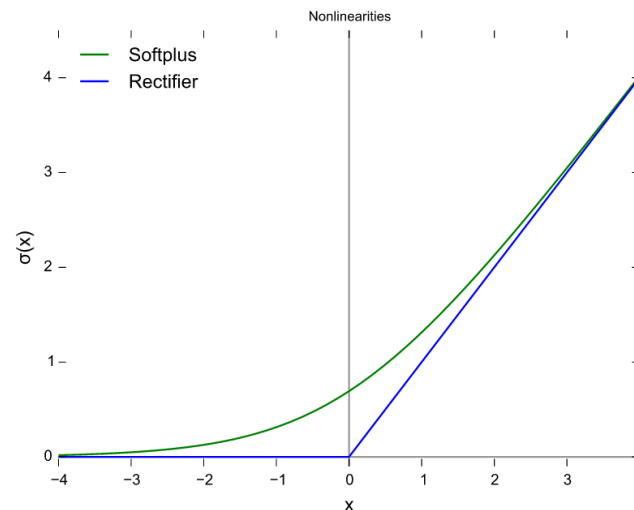
$$g(a) = \max(-1, \min(1, a))$$

哪一个最好？

# ReLU 神经元及其推广（一）

$$h = g(\mathbf{W}^\top \mathbf{x} + \mathbf{b})$$

初始化:  
 $\mathbf{b}=0.1$   
(经验)



$$z_i < 0: h_i = g(\mathbf{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

推广:  $\alpha_i = -1$   $g(z) = |z|$  物体识别

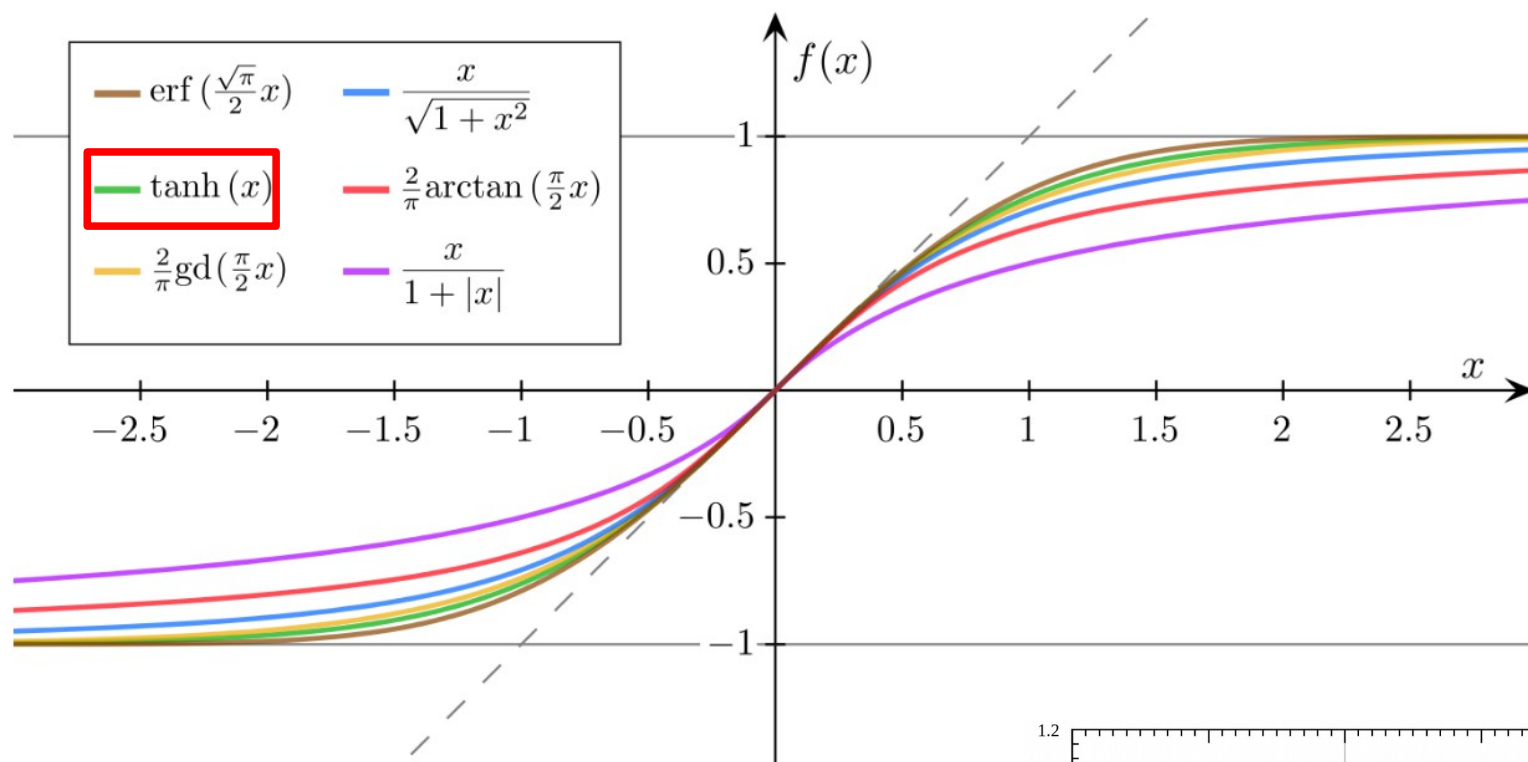
$\alpha_i = 0.01$  Leaky ReLU

$\alpha$  是可学习参数, Parametric ReLU

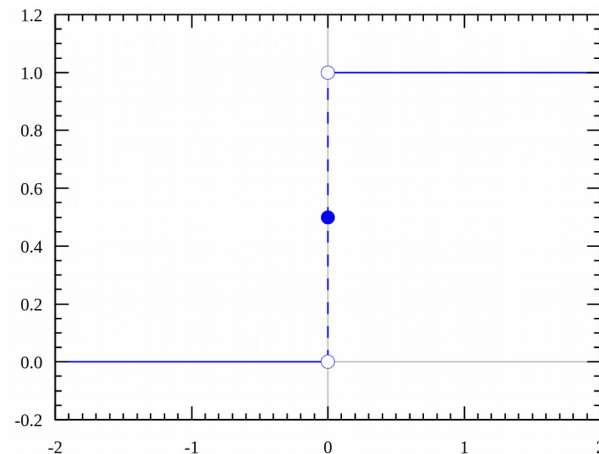
# ReLU 神经元及其推广（二）

Maxout :  $g(\mathbf{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$      $\mathbb{G}^{(i)}$  是对输入样本分组

# 隐藏层：激活（activation）函数



类 S 函数在大多数时候趋于饱和，梯度趋于 0，不利于梯度算法，因此一般不用于隐藏层，输出层可以通过取对数的方式避免饱和。



# 神经网络设计

输出层 + 隐藏层 + 网络结构



# 网络结构

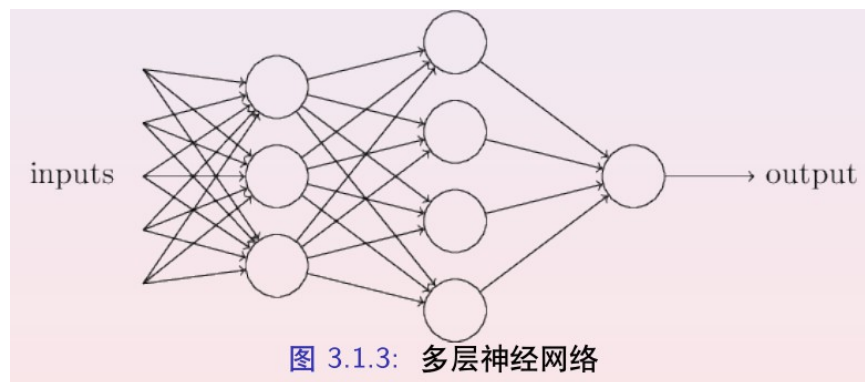
- 1、网络层数，每层神经元个数；
- 2、连接方式，例如，链式结构；

$$h^{(1)} = g^{(1)} \left( \mathbf{W}^{(1)\top} \mathbf{x} + \mathbf{b}^{(1)} \right)$$

↓

$$h^{(2)} = g^{(2)} \left( \mathbf{W}^{(2)\top} h^{(1)} + \mathbf{b}^{(2)} \right)$$

法无定法，依赖测试！



# 神经网络和非线性

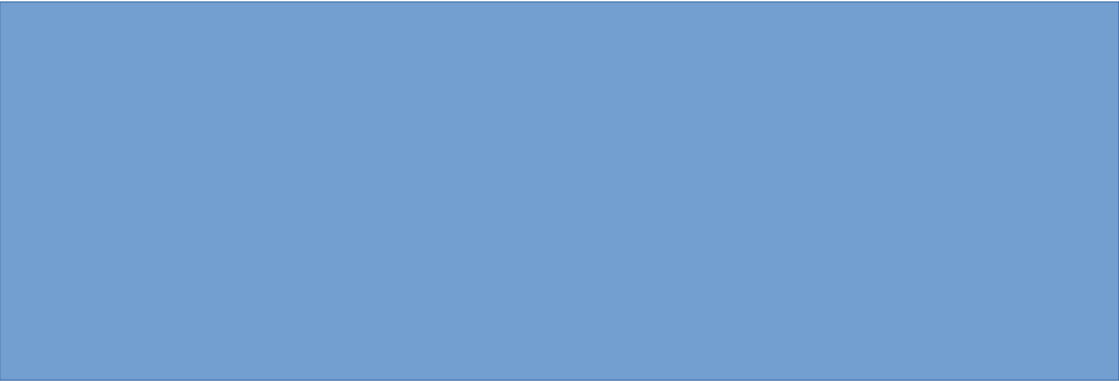
线性与非线性：线性模型最终会得到一个凸的损失函数；

非线性问题的损失函数一般为非凸；

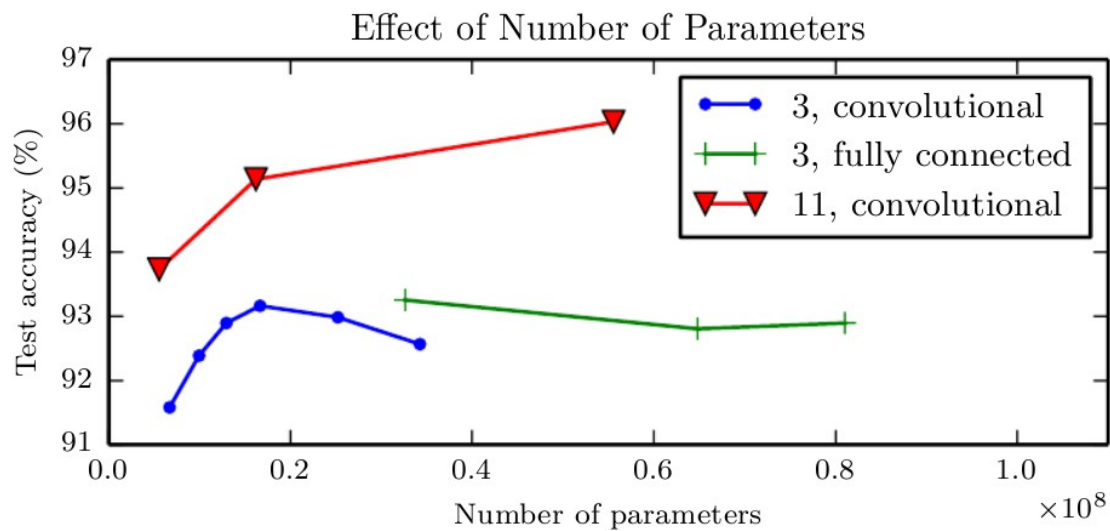
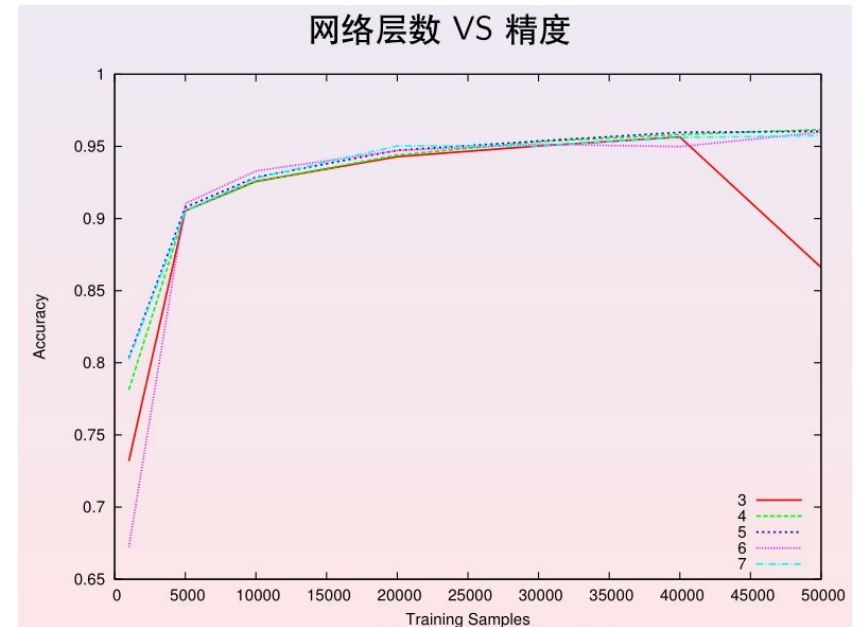
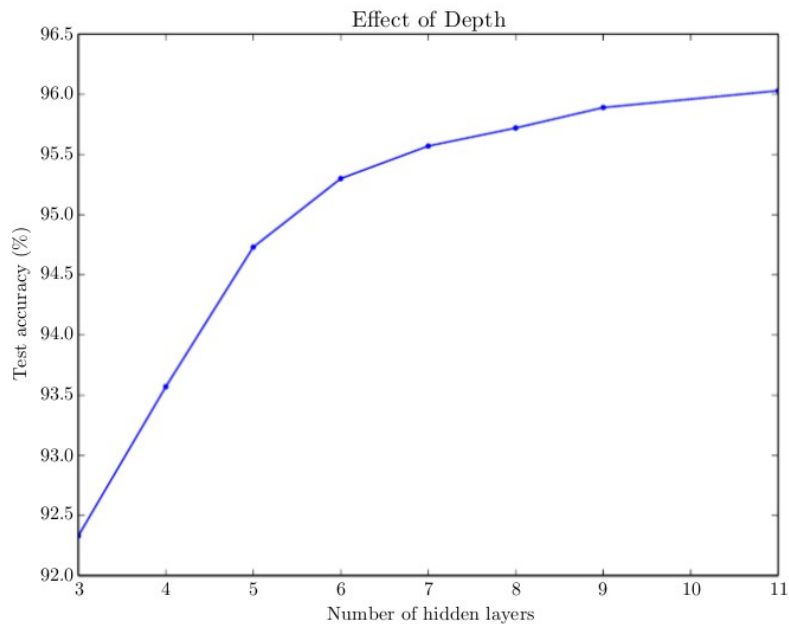
普适与非普适：前馈网络可以用来描述任意非线性函数，

但是优化算法未必能够得到正确的网络参数。

# 神经元数目还是层数？

- 
- 1、规模够大的单层神经网络可以描述任意函数，主要困难是规模过大、难以训练和推广；
  - 2、多层网络可以规避以上缺点；  
( how and why? )
  - 3、网络深度存在临界值；

# 网络结构和精度



CSRC-2017

# 总结：神经网络结构

- 1、深度；
- 2、宽度；
- 3、连接方式；

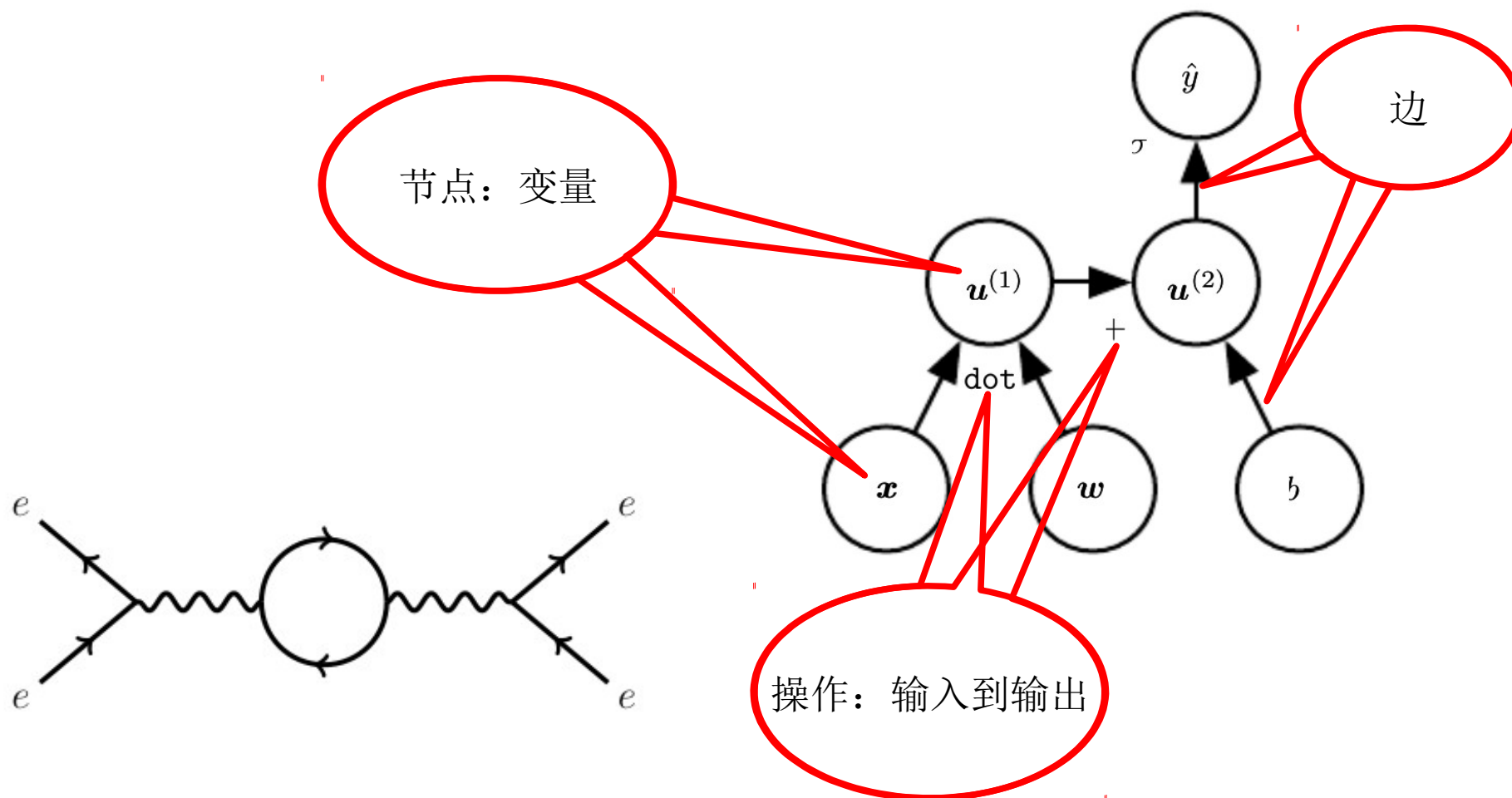
# BP 算法和梯度

常见的误解： BP 算法不是学习算法。

BP 算法是求梯度的通用方法；

SGD 是学习算法；

# 计算图方法



# 微分链式法则

$$z = f(g(x)) = f(y)$$

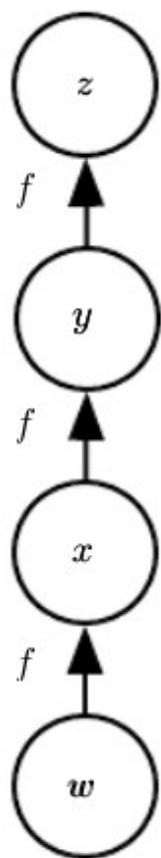
$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

$$\nabla_{\mathbf{x}} z = \left( \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^\top \nabla_{\mathbf{y}} z = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad \nabla_{\mathbf{x}} z = \sum_j (\nabla_{\mathbf{x}} Y_j) \frac{\partial z}{\partial Y_j}$$

雅可比  
行列式



# 链式求导：CPU 和内存



$$\begin{aligned}\frac{\partial z}{\partial w} &= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \\ &= f'(y) f'(x) f'(w) \\ &= f'(f(f(w))) f'(f(w)) f'(w)\end{aligned}$$

待计算子串的数目随着  
图的复杂度指数增加！

# 前馈算法

```
for  $i = 1, \dots, n_i$  do  
     $u^{(i)} \leftarrow x_i$   
end for  
for  $i = n_i + 1, \dots, n$  do  
     $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$   
     $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$   
end for  
return  $u^{(n)}$ 
```

算法初始化：输入矢量  $x$

计算节点  $n_i+1$  的所有父节点  
计算节点  $n_i+1$

父节点

# 反馈算法

Run forward propagation (Algorithm 6.1 for this example) to obtain the activations of the network

Initialize `grad_table`, a data structure that will store the derivatives that have been computed. The entry `grad_table[u(i)]` will store the computed value of  $\frac{\partial u^{(n)}}{\partial u^{(i)}}$ .

`grad_table[u(n)] ← 1`

`for j = n - 1 down to 1 do`

The next line computes  $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$  using stored values:

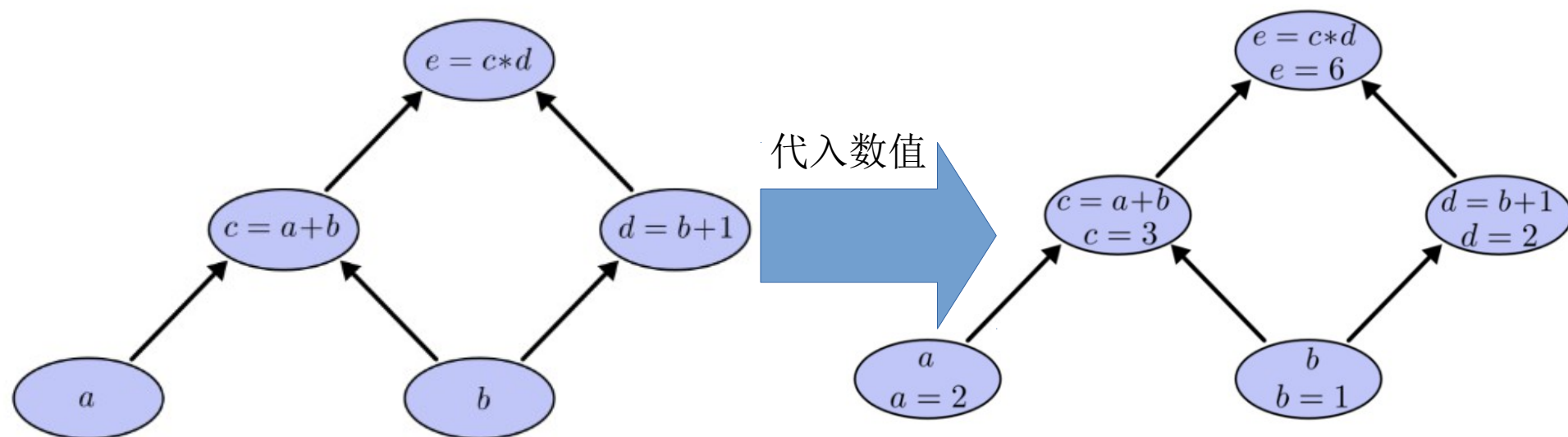
`grad_table[u(j)] ←  $\sum_{i:j \in Pa(u^{(i)})} \text{grad\_table}[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$`

`end for`

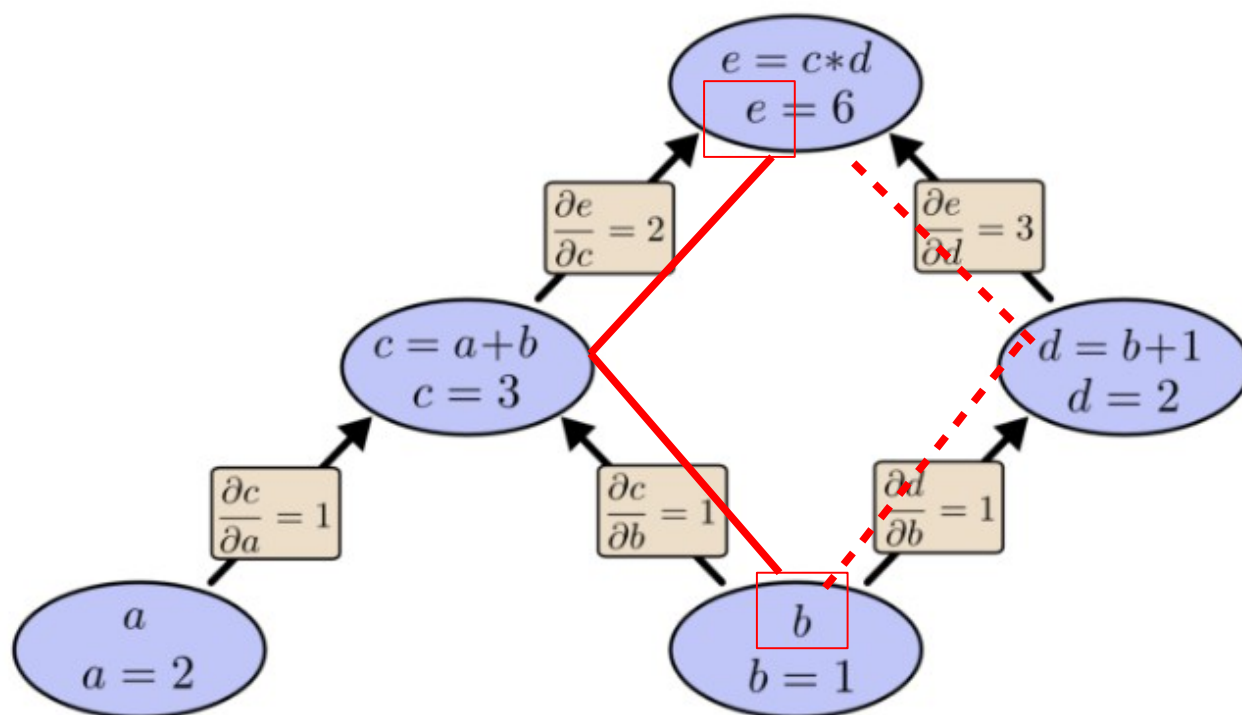
`return {grad_table[u(i)] | i = 1, ..., ni}`

# 例：BP 算法，与深度学习无关

$$e = (a + b) * (b + 1)$$



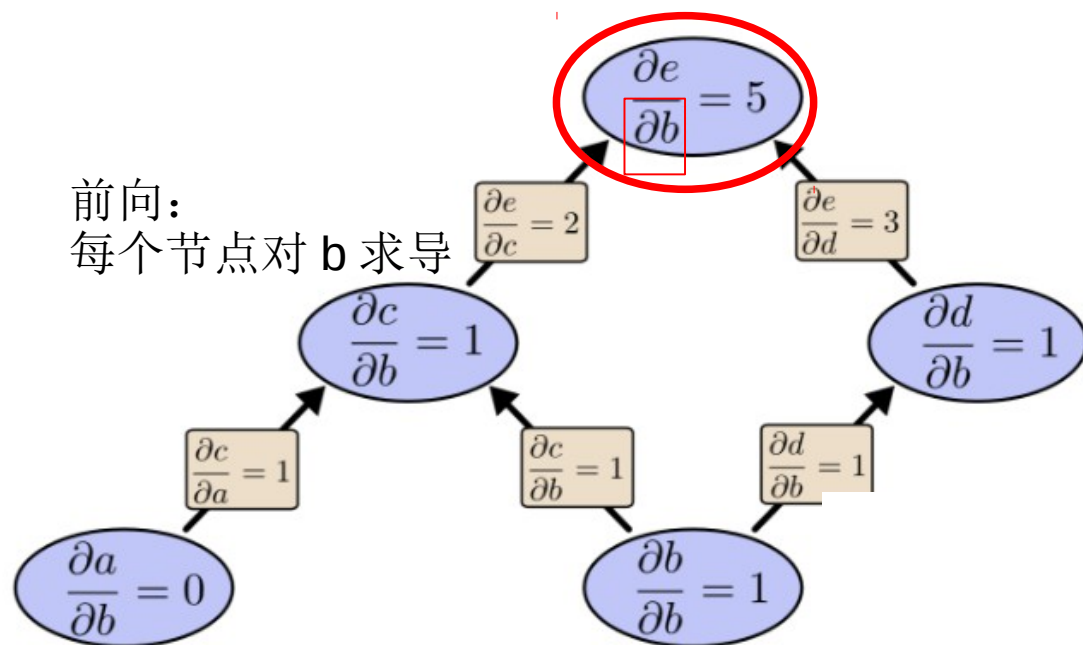
# 求导：路径求和



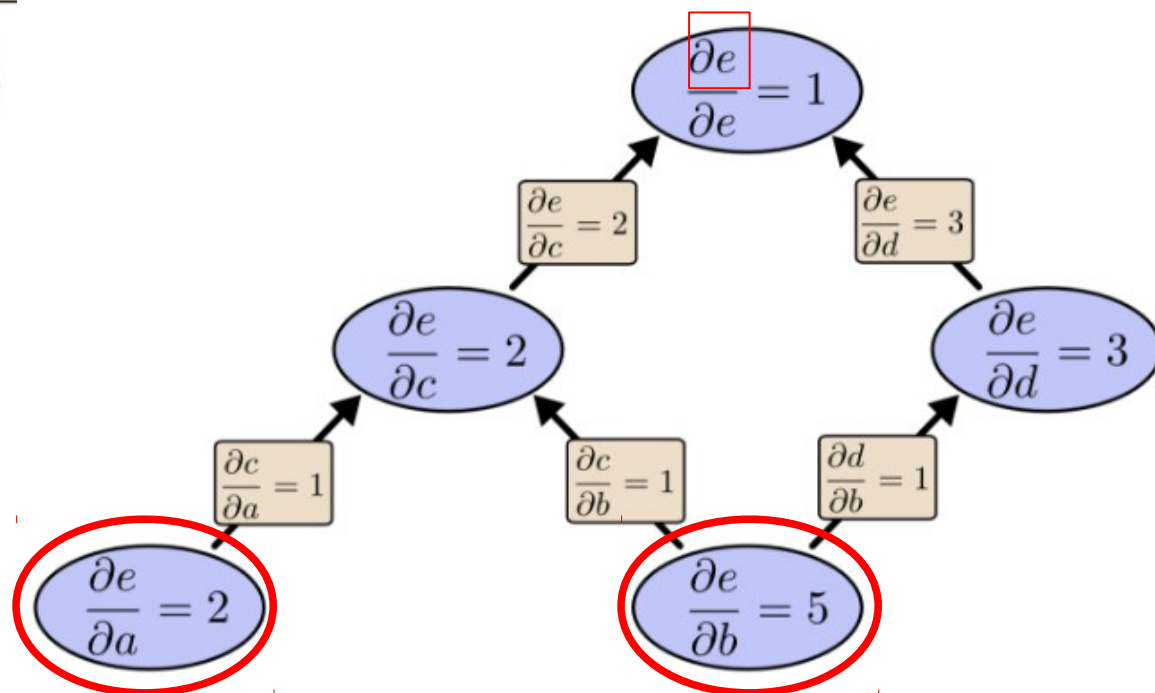
$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3$$

# 求导：前向和反向

前向：  
每个节点对  $b$  求导



反向：  
 $e$  对每个节点求导



加速倍数  
等于输入  
参数个  
数！

# WHO INVENTED THE REVERSE MODE OF DIFFERENTIATION?

ANDREAS GRIEWANK

Seppo Linnainmaa (Lin76) of Helsinki says the idea came to him on a sunny afternoon in a Copenhagen park in 1970. He used it as a tool for estimating the effects of arithmetic rounding errors on the results of complex expressions. Gerardi Ostrowski (OVB71) discovered and used it some five years earlier in the context of certain process models in chemical engineering. Here and throughout references that are not listed in the present bibliography are noted in parentheses and can be found in the book [7].

Also in the sixties Hachtel et al. [6] considered the optimization of electronic circuits using the costate equation of initial value problems and its discretizations to compute gradients in the reverse mode for explicitly time-dependent problems. Here we see, possibly for the first time, the close connection between the reverse mode of discrete evaluation procedures and continuous adjoints of differential equations. In the 1970s Iri analyzed the properties of dual and adjoint networks. In the 1980s he became one of the key researchers on the reverse mode.

被反复发明十余次，海森堡并不孤独！

# 深度前馈网络

**Require:** Network depth,  $l$

**Require:**  $\mathbf{W}^{(i)}, i \in \{1, \dots, l\}$ , the weight matrices of the model

**Require:**  $\mathbf{b}^{(i)}, i \in \{1, \dots, l\}$ , the bias parameters of the model

**Require:**  $\mathbf{x}$ , the input to process

**Require:**  $\mathbf{y}$ , the target output

$$\mathbf{h}^{(0)} = \mathbf{x}$$

**for**  $k = 1, \dots, l$  **do**

$$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}$$

$$\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$$

**end for**

$$\hat{\mathbf{y}} = \mathbf{h}^{(l)}$$

$$J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda \Omega(\theta)$$

---



# (深度) 反向传播 (BP) 算法

---

After the forward computation, compute the gradient on the output layer:

$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, y)$  从最后一层开始

for  $k = l, l - 1, \dots, 1$  do

Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if  $f$  is element-wise):

$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$  从后往前逐层求导，相当于把总误差函数逐层分配。

Compute gradients on weights and biases (including the regularization term, where needed):

$$\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g} + \lambda \nabla_{\mathbf{b}^{(k)}} \Omega(\theta)$$

$$\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g} \mathbf{h}^{(k-1)\top} + \lambda \nabla_{\mathbf{W}^{(k)}} \Omega(\theta)$$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

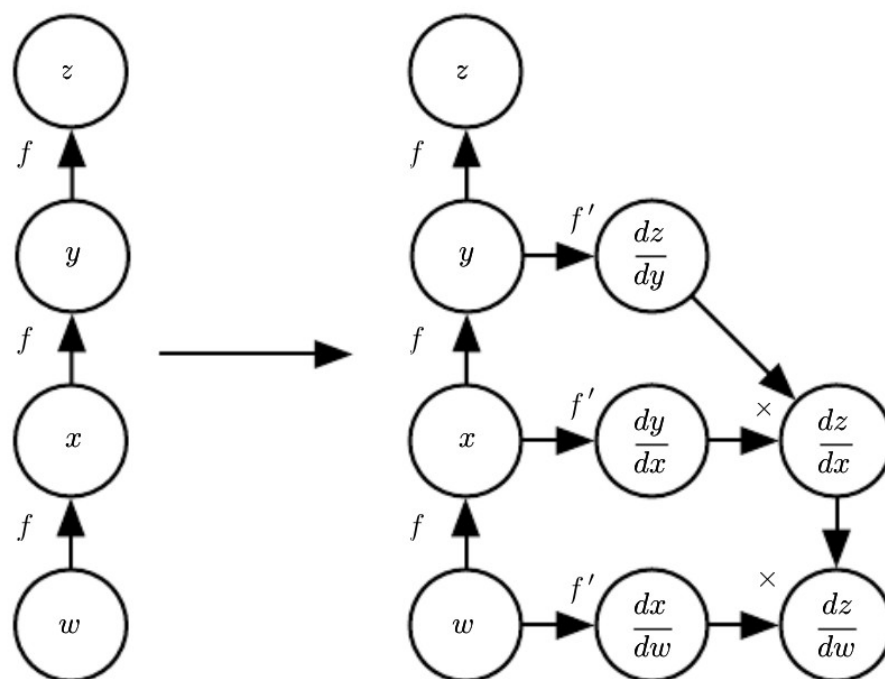
$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = \mathbf{W}^{(k)\top} \mathbf{g}$$

end for

---

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix} \quad \frac{\partial z}{\partial w} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w}$$

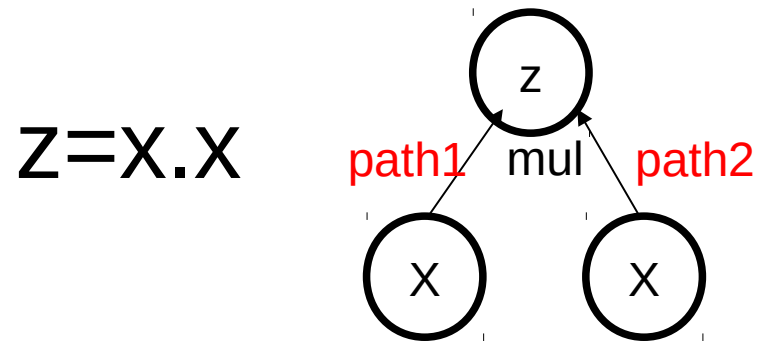
# 符号导数和数值导数



数值导数: Torch、Caffe  
图、符号导数: Theano、Tensorflow

# BP 算法：推广

$$\text{op.bprop}(\text{inputs}, \mathbf{X}, \mathbf{G}) = \sum_i (\nabla_{\mathbf{X}} \text{op.f}(\text{inputs})_i) G_i$$



$$\frac{\partial z}{\partial X} = \frac{\partial x}{\partial x} x + x \frac{\partial x}{\partial x} = 2x$$

# 深度学习中的 BP 算法：刚需！

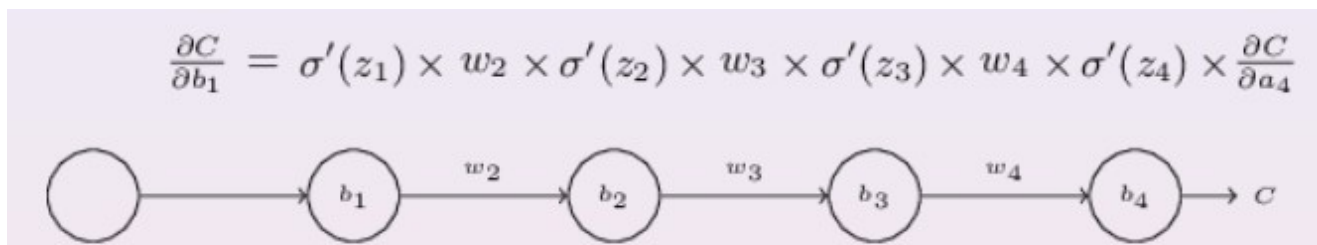


图 3.3.1: 多层单神经网络

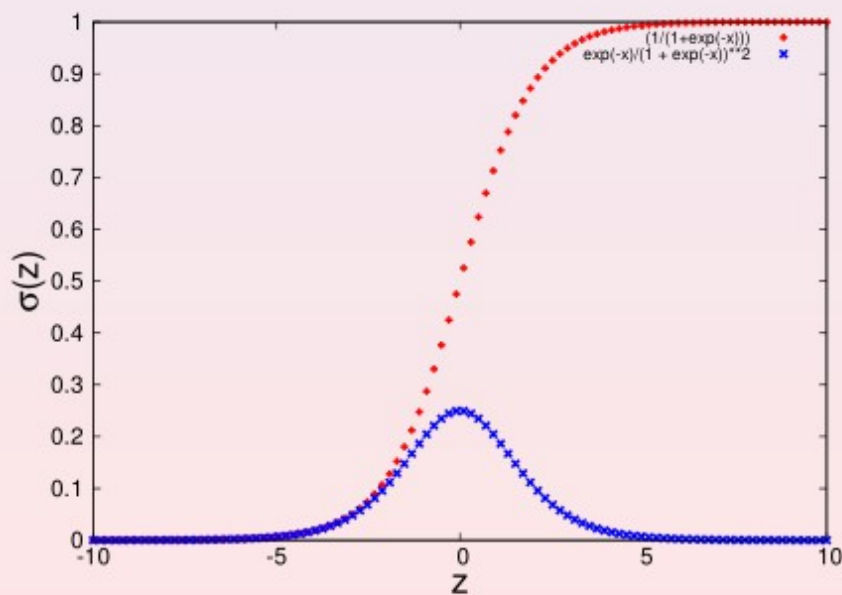
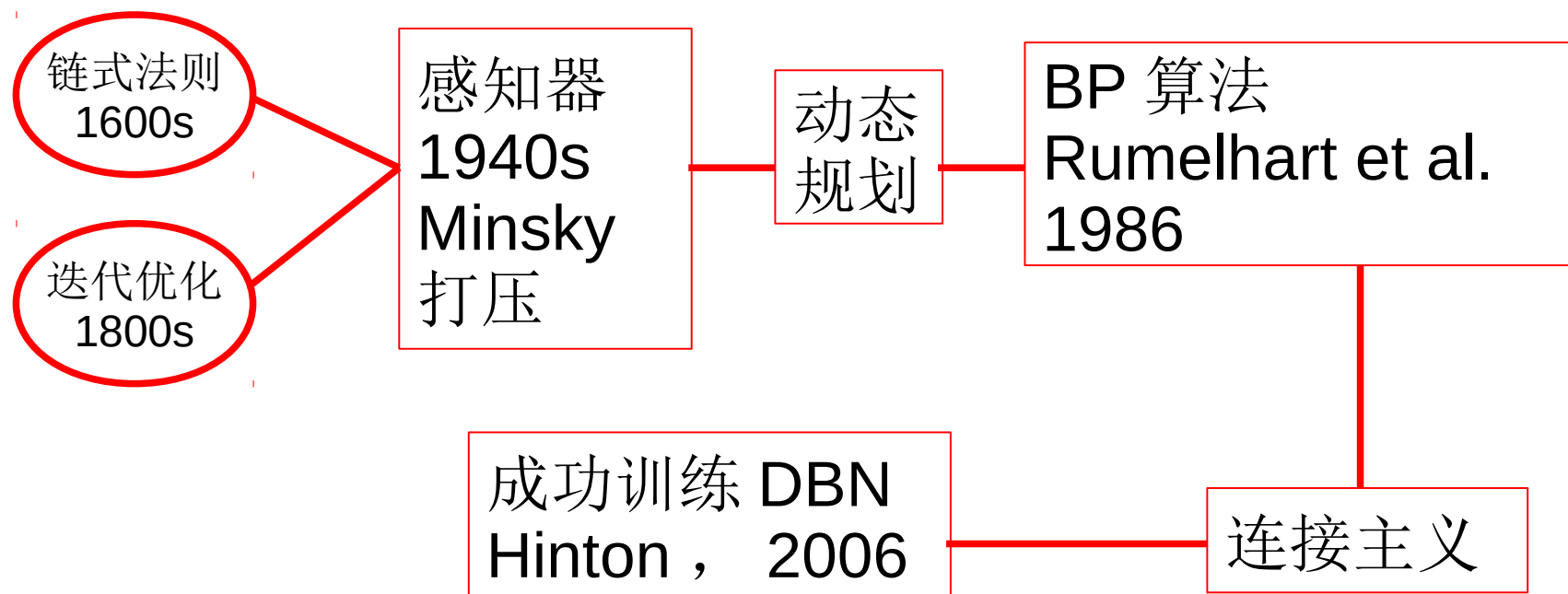


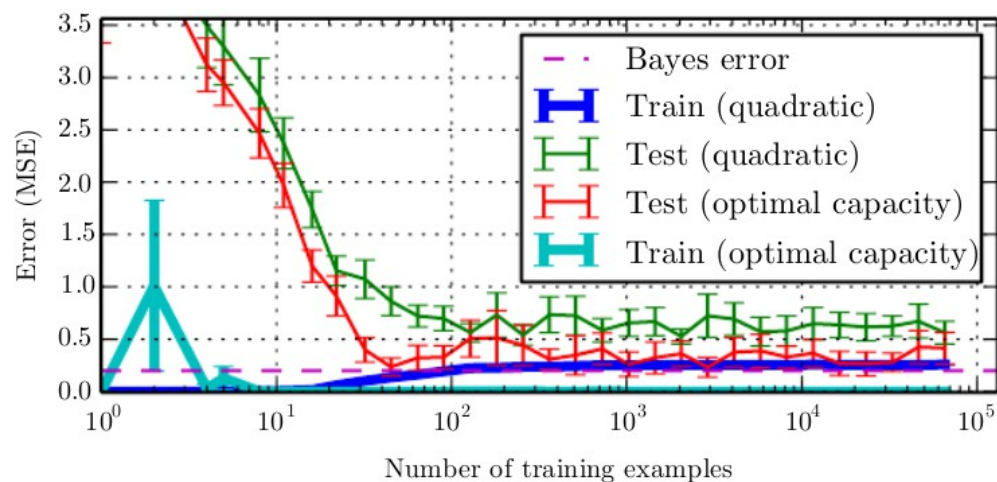
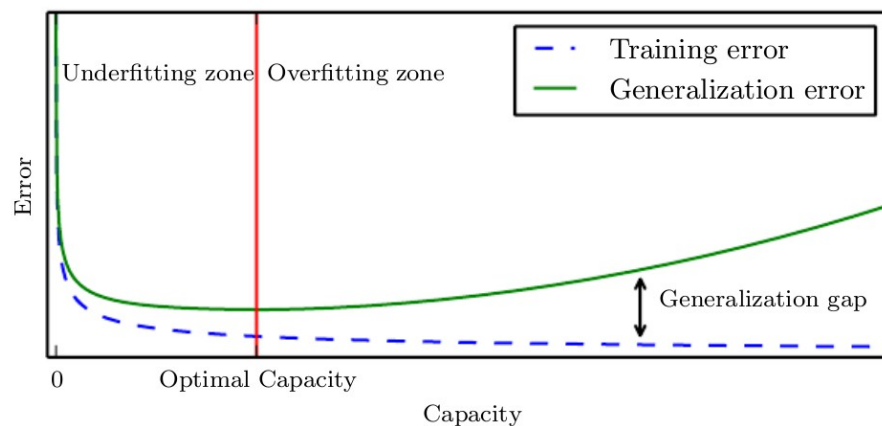
图 3.3.2: 激活函数及其导数

# 神经网络的历史轨迹



Feedforward networks can be seen as efficient nonlinear function approximators based on using gradient descent to minimize the error in a function approximation. From this point of view, the modern feedforward network is the culmination of centuries of progress on the general function approximation task.

# 回顾：训练误差和测试误差



# 深度学习中的正则化

深度学习中用来减少测试误差的方法统称为正则化方法。

限制参数取值范围；

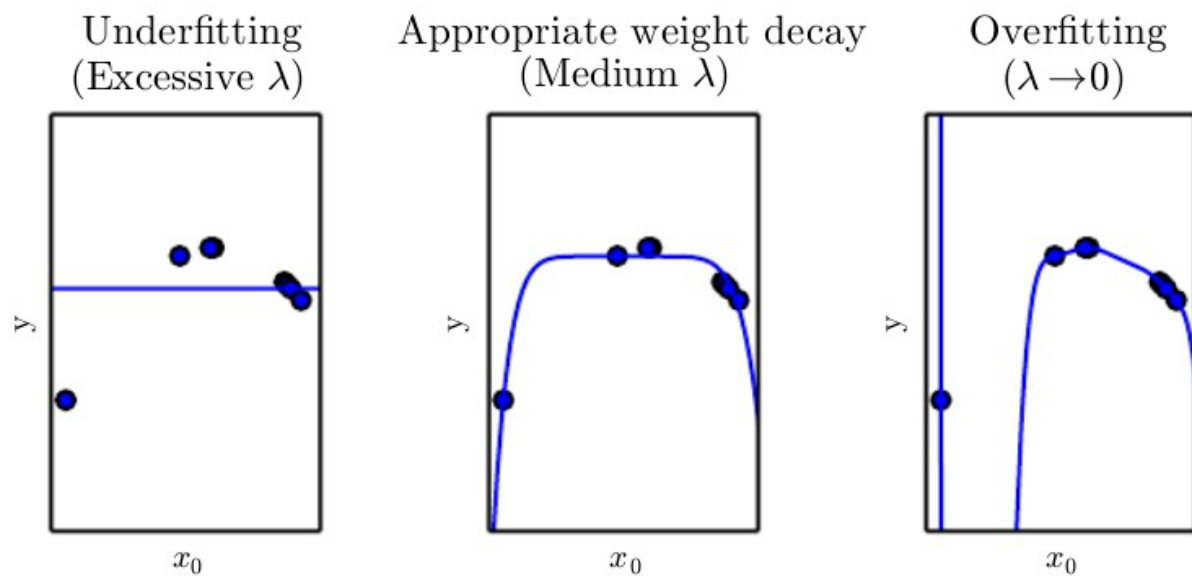
修订目标函数的形式；

纳入已知的经验；

（对深度学习的效果影响很大）

.....

# 正则化的作用





# 优化目标：损失函数

$$\tilde{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) = J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) + \alpha \Omega(\boldsymbol{\theta})?$$

最小二乘法

$$L(Y, f(X)) = \sum_{i=1}^n (Y - f(X))^2$$

支持向量机 (SVM)

$$L(y) = \max(0, 1 - y\tilde{y}), y = \pm 1$$

(自适应) 增强学习  
(Adaboost)

$$L(y, f(x)) = \frac{1}{n} \sum_{i=1}^n \exp[-y_i f(x_i)]$$

逻辑回归

$$L(y, P(Y = y|x)) = \begin{cases} \log(1 + \exp\{-f(x)\}) & , y = 1 \\ \log(1 + \exp\{f(x)\}) & , y = 0 \end{cases}$$

# Lp 球

$$x = \operatorname{argmin}_{x:Ax=b} \|x\|_0$$

$$x^* := \operatorname{argmin}_{x:Ax=b} \|x\|_1.$$

$$x^\# = \operatorname{argmin}_{x:Ax=b} \|x\|_2.$$

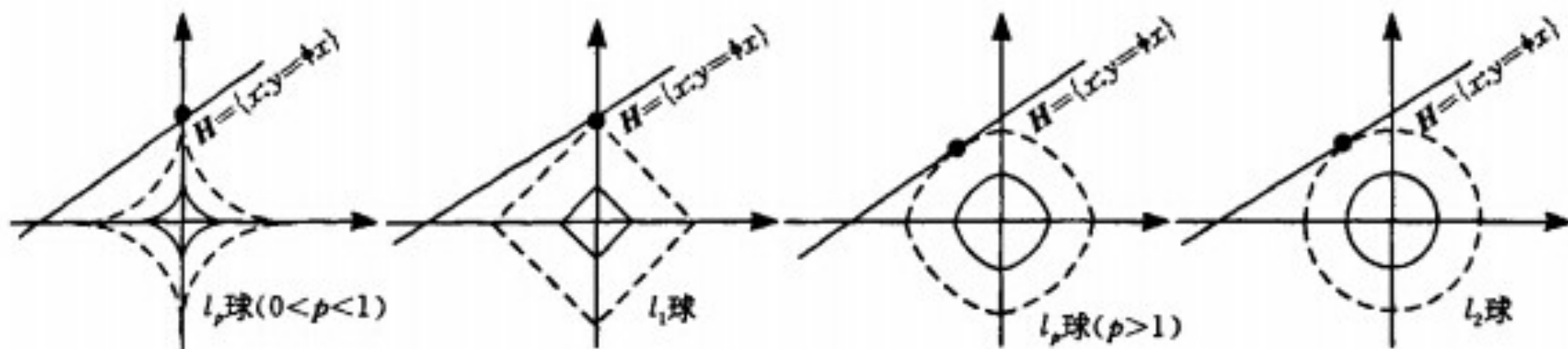
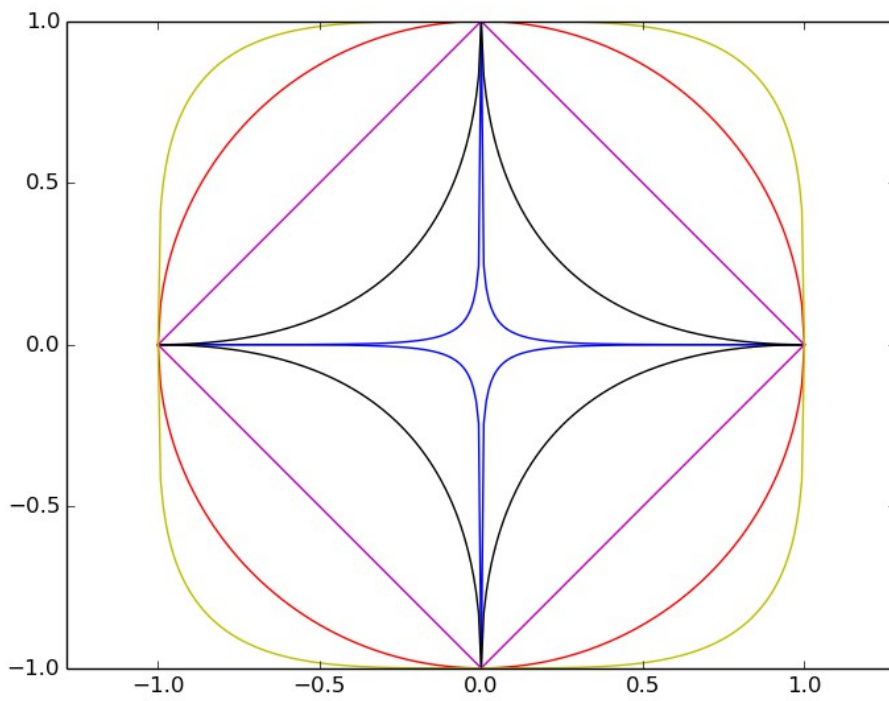


图 2 极小化  $l_1$  范数导致稀疏解的几何说明

# Lp 球

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}$$



二维情况下

# Lp 范数

L0 范数

非 0 元素的个数：稀疏化

L1 范数

各元素绝对值的和：稀疏化

L2 范数

欧氏向量模：避免过拟合

核范数

矩阵奇异值的和

# L0 范数和压缩感知

$$x = \sum_{i=1}^N \theta_i \psi_i \quad \theta_i = \langle x, \psi_i \rangle = \psi_i^T x$$

系数向量  $\theta$  为  $k$  稀疏,  $k \ll N$ 。(如何实现  $k$  稀疏?  
最小化  $L_0$  范数! ——  $>NP$  难题)

引入新的线性变换:  $\Phi: M \times N (M \ll N)$

$$y = \Phi x \quad y = \Phi \Psi \theta = A^{CS} \theta$$

将标准的线性代数问题转换为一个最优化问题!

—— 陶哲轩、Emmanuel Candes

# L1 范数和稀疏化

优势：（凸）近似等价于 L0 范数；可优化；

自动化特征选择，自学习过滤掉无用特征；

稀疏化导致可解释性；

# L2 范数和过拟合

过拟合：训练误差小，而测试误差大；

（典型的应试教育！）

L2 范数用来减少权重系数  $w$  的相互差异，从而达到“惩罚”的目的；

$$\kappa(A) = \|A\| \|A^{-1}\|,$$

Ridge Regression

或者

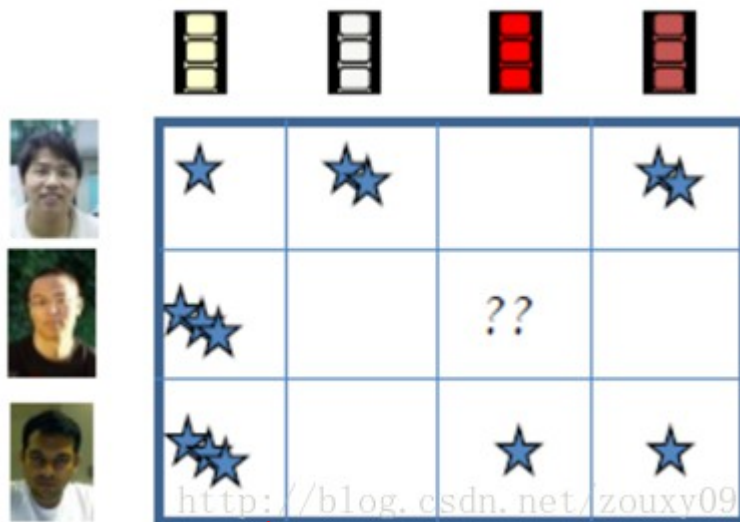
Weight decay

equations	solution	equations	solution
$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 7.999 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4.001 \\ 7.998 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -3.999 \\ 4.000 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4.001 \\ 7.001 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1.999 \\ 1.001 \end{bmatrix}$
$\begin{bmatrix} 1.001 & 2.001 \\ 2.001 & 3.998 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 7.999 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3.994 \\ 0.001388 \end{bmatrix}$	$\begin{bmatrix} 1.001 & 2.001 \\ 2.001 & 3.001 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$	$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2.003 \\ 0.997 \end{bmatrix}$

# 核范数

核范数：矩阵奇异值的和；（秩：矩阵非 0 奇异值的和；核范数是秩的凸近似）

用处：



去马塞克

$$\text{observation} = \text{low-rank} + \text{sparse}$$
$$Y = X + E$$

推荐系统



# 深度学习中的正则化 和超参数（hyperparameter）选择

$$w^* = \arg \min_w \sum_i L(y_i, f(x_i; w)) + \lambda \Omega(w)$$

超参数定义了不同的模型种类；

困难：无法通过简单最优化的方法确定超参数；

如何确定超参数：

经验；交叉验证；

坑：

调参复杂度远甚于 DFT ！

集成库（TF 等）里面往往隐藏了超参数！

# 线性回归 +L2 范数正则化

$$\begin{array}{ccc} (Xw - y)^\top (Xw - y) & \longrightarrow & (Xw - y)^\top (Xw - y) + \frac{1}{2}\alpha w^\top w \\ \downarrow & & \downarrow \\ w = (X^\top X)^{-1} X^\top y & & w = (X^\top X + \alpha I)^{-1} X^\top y \end{array}$$

# 线性回归 +L1 范数正则化

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \alpha \|\mathbf{w}\|_1 + J(\mathbf{w}; \mathbf{X}, \mathbf{y})$$



$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \alpha \text{sign}(\mathbf{w}) + \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w})$$



近似

$$\hat{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}^*; \mathbf{X}, \mathbf{y}) + \sum_i \left[ \frac{1}{2} H_{i,i} (\mathbf{w}_i - \mathbf{w}_i^*)^2 + \alpha |\mathbf{w}_i| \right]$$



$$\mathbf{w}_i = \text{sign}(\mathbf{w}_i^*) \max \left\{ |\mathbf{w}_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

$$\mathbf{w}_i^* > 0 \quad \mathbf{w}_i^* \leq \frac{\alpha}{H_{i,i}} \longrightarrow \mathbf{w}_i = 0 \quad \text{稀疏性}$$

# 正则化：人工扩充数据

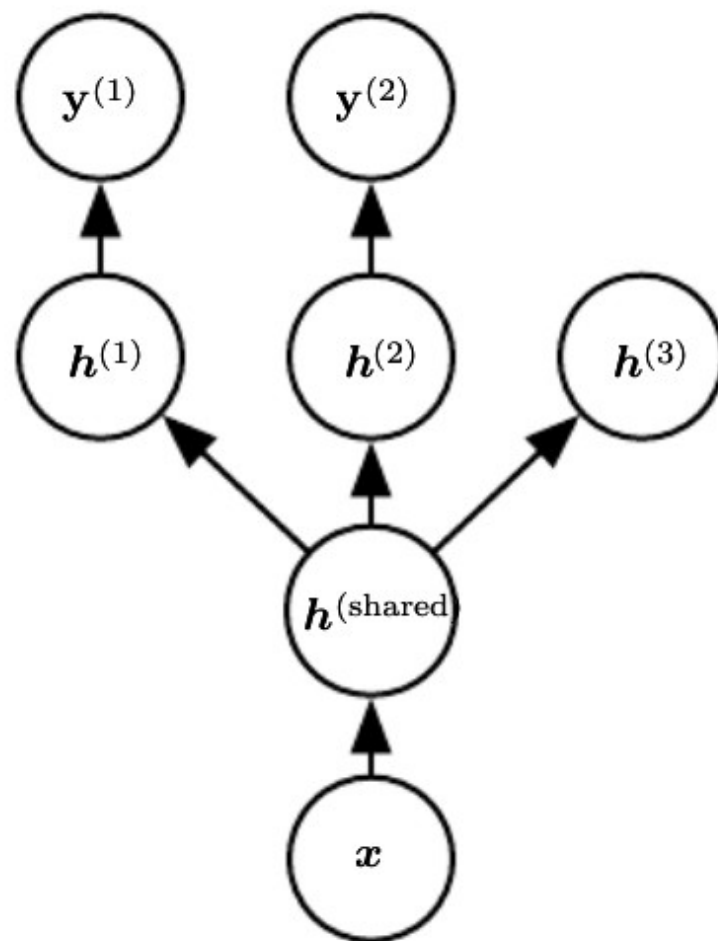
原因：训练样本不够；

方法：对已有样本的输入向量  $x$  做线性变换，保证分类不变（例如 6 和 9 不能做 180 度旋转）；

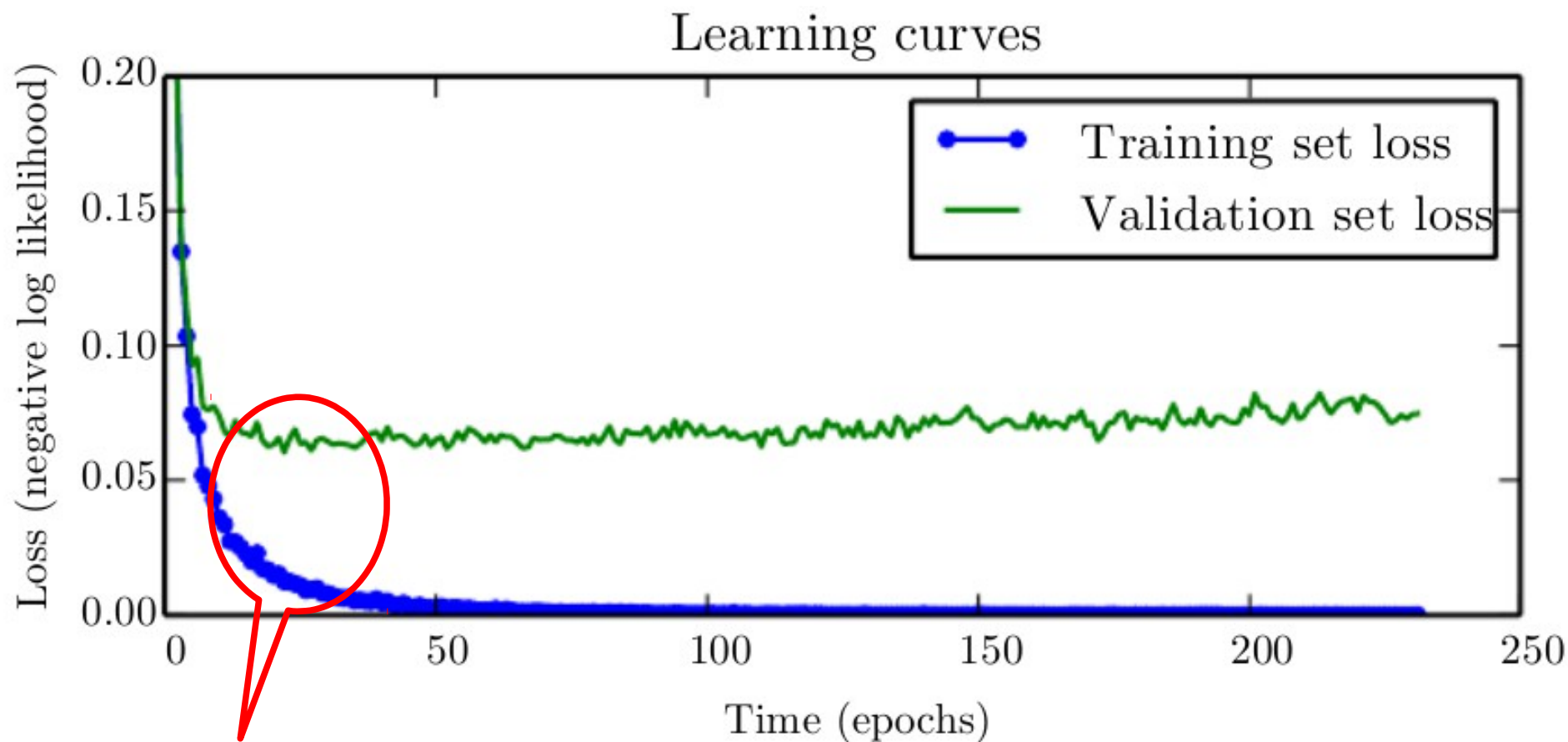
优点：常常可以在不改变模型超参数的情况，提升模型的预测能力；

用处：几何图形识别

# 正则化：多任务学习



# 正则化：提前停止 ( early stoping )



停止点

优化过程提前停止：通常的优化过程会在一个局域最小值附近停止。

# 正则化：权值共享

$$\hat{y}^{(A)} = f(\mathbf{w}^{(A)}, \mathbf{x})$$
$$\hat{y}^{(B)} = g(\mathbf{w}^{(B)}, \mathbf{x})$$

CNN：

- 1、紧束缚——有限大小的过滤器；
- 2、平移不变性——过滤器处处相同；

# 稀疏表示

模型稀疏化

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\mathbf{y} \in \mathbb{R}^m$        $\mathbf{A} \in \mathbb{R}^{m \times n}$        $\mathbf{x} \in \mathbb{R}^n$

$$\begin{bmatrix} -14 \\ 1 \\ 1 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\mathbf{y} \in \mathbb{R}^m$        $\mathbf{B} \in \mathbb{R}^{m \times n}$        $\mathbf{h} \in \mathbb{R}^n$

表示稀疏化



# bootstrap aggregating

从训练样本集中抽取多个子集，分别训练模型；

分别采用各个模型对新样本进行预测；

结果：分类问题票决出结果，

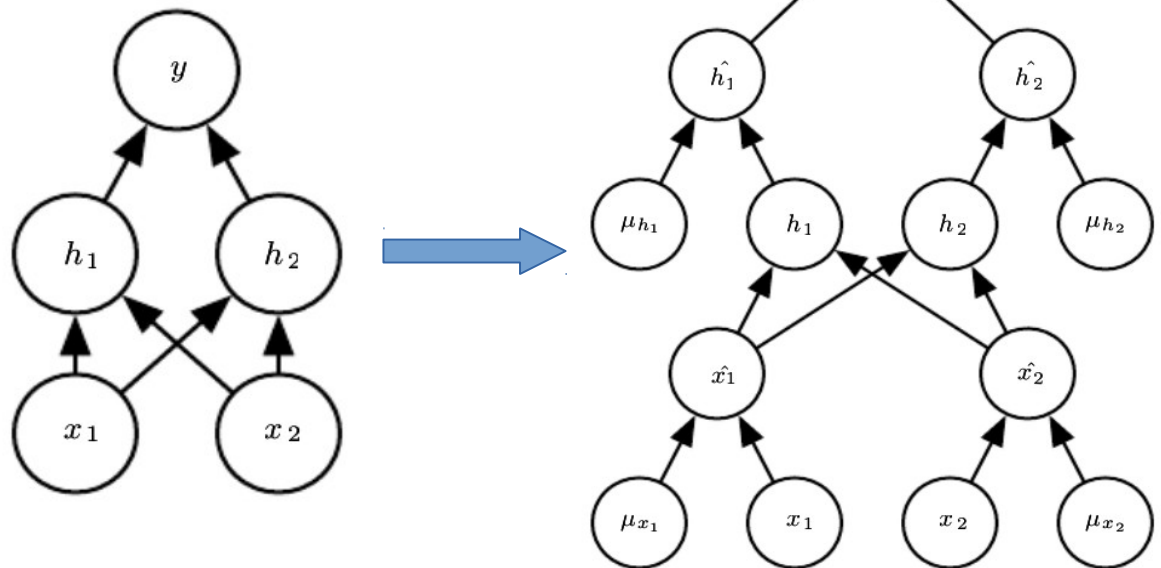
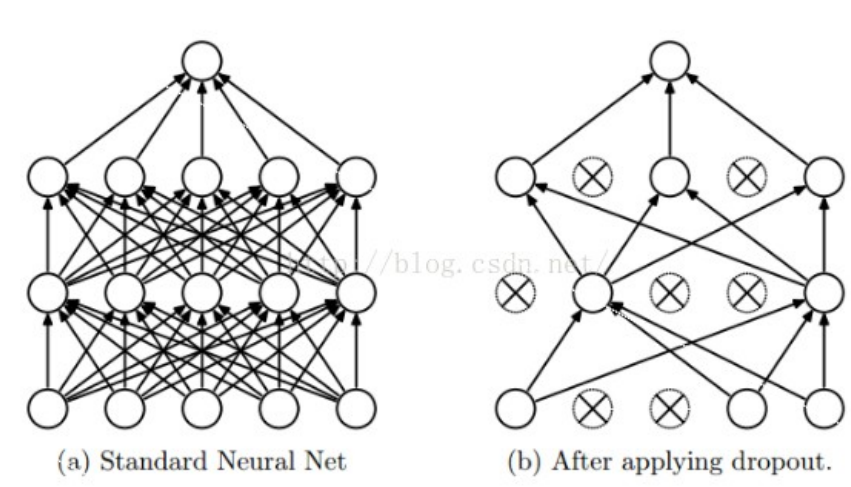
回归问题则取平均值。

# 增强学习： boosting

区别于 bagging：

- 1、训练集的选择独立：重点关注训练失败的样本；
- 2、预测效果好的预测函数权重较大（ bagging 等权重）；
- 3、 bagging 易于并行，精度较差，收敛较慢； boosting 在并行函数间无法并行，但一般收敛较快。

# dropout



# 对抗样本



+ .007 ×



=



$\mathbf{x}$

$y = \text{"panda"}$   
w/ 57.7%  
confidence

$\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$

"nematode"  
w/ 8.2%  
confidence

$\mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$   
"gibbon"  
w/ 99.3 %  
confidence

$$\tilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon \text{sign}(\nabla_x J(\theta, x, y)))$$

# 生成对抗网络原理

生成网络 **G** : 制造假币;

判别网络 **D** : 验钞机;

两组模型分别同步训练;

应用:

文字——> 图像

高清图像

# 生成对抗网络： 创造力和想象力



# “学习”和优化

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{\text{data}}} L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{\text{data}}} L(f(\mathbf{x}; \boldsymbol{\theta}), y)$$

假如  $(\mathbf{x}, y)$  来源于真实的概率分布函数，  
则为标准的优化问题；否则就需要通过  
“学习”来得到参数  $\Theta$ 。

# “提前停止”的启示

“学习”算法通常会在训练算法进入某个局域极小值之前就停止，从而达到广义误差最小，而不是训练误差最小。

是否能用这种方法来进行全局搜索？



# minibatch 和随机梯度下降

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}, y; \boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(\mathbf{x}, y; \boldsymbol{\theta})$$

$$\text{SE}(\hat{\mu}_m) = \sqrt{\text{Var}\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

- 1、minibatch 是划算的！
- 2、可并行；省内存；
- 3、batchsize 和 learning rate 正相关；  
需要仔细选择。

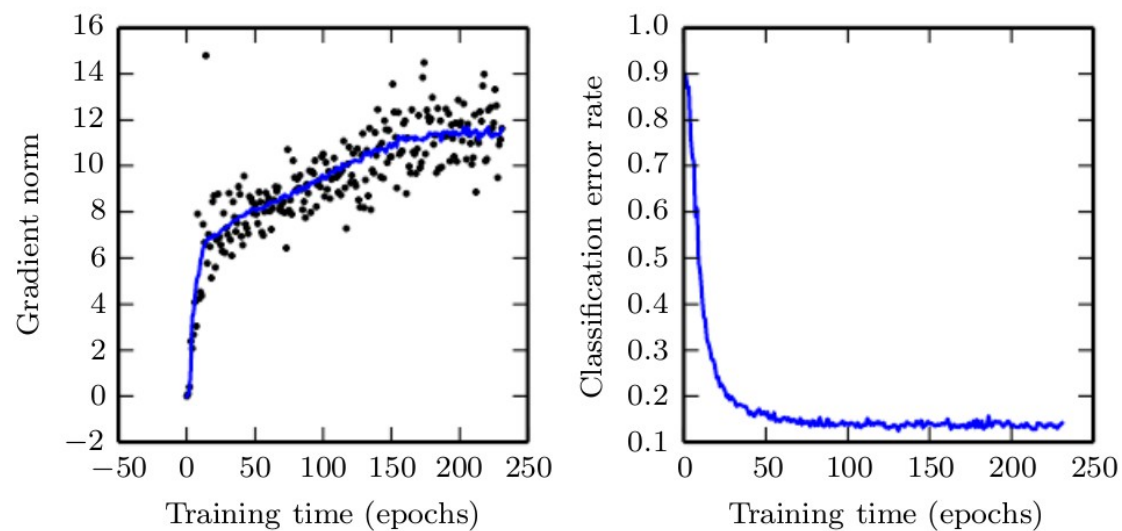
# Hessian 矩阵和条件数

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

梯度算法的收敛速度:  $\frac{\lambda_{\max}}{\lambda_{\min}} = \frac{\lambda_1}{\lambda_n} = \kappa$

# 病态 Hessian 矩阵

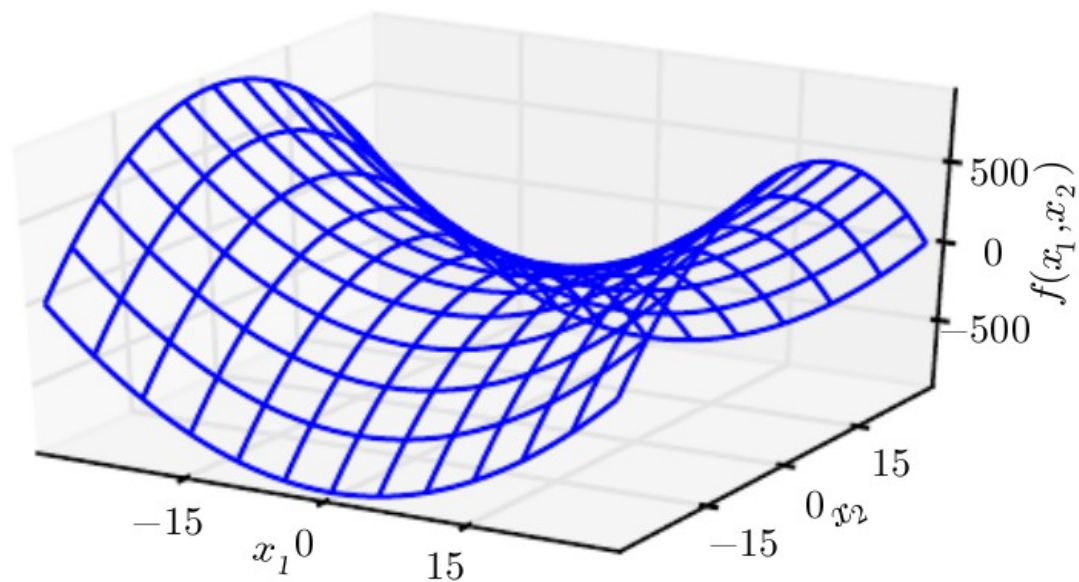
二阶项大于一阶项:  $\frac{1}{2}\epsilon^2 \mathbf{g}^\top \mathbf{H} \mathbf{g} - \epsilon \mathbf{g}^\top \mathbf{g}$



# 局域极小陷阱： 凸优化与非凸优化

- 1、凸优化拥有全局极小值；
- 2、NN 拥有大量的局域极小值；
- 3、大多数时候，局域极小值不具备破坏性；
- 4、如果局域极小值对应的 **cost func** 较大，则容易造成问题；
- 5、.....

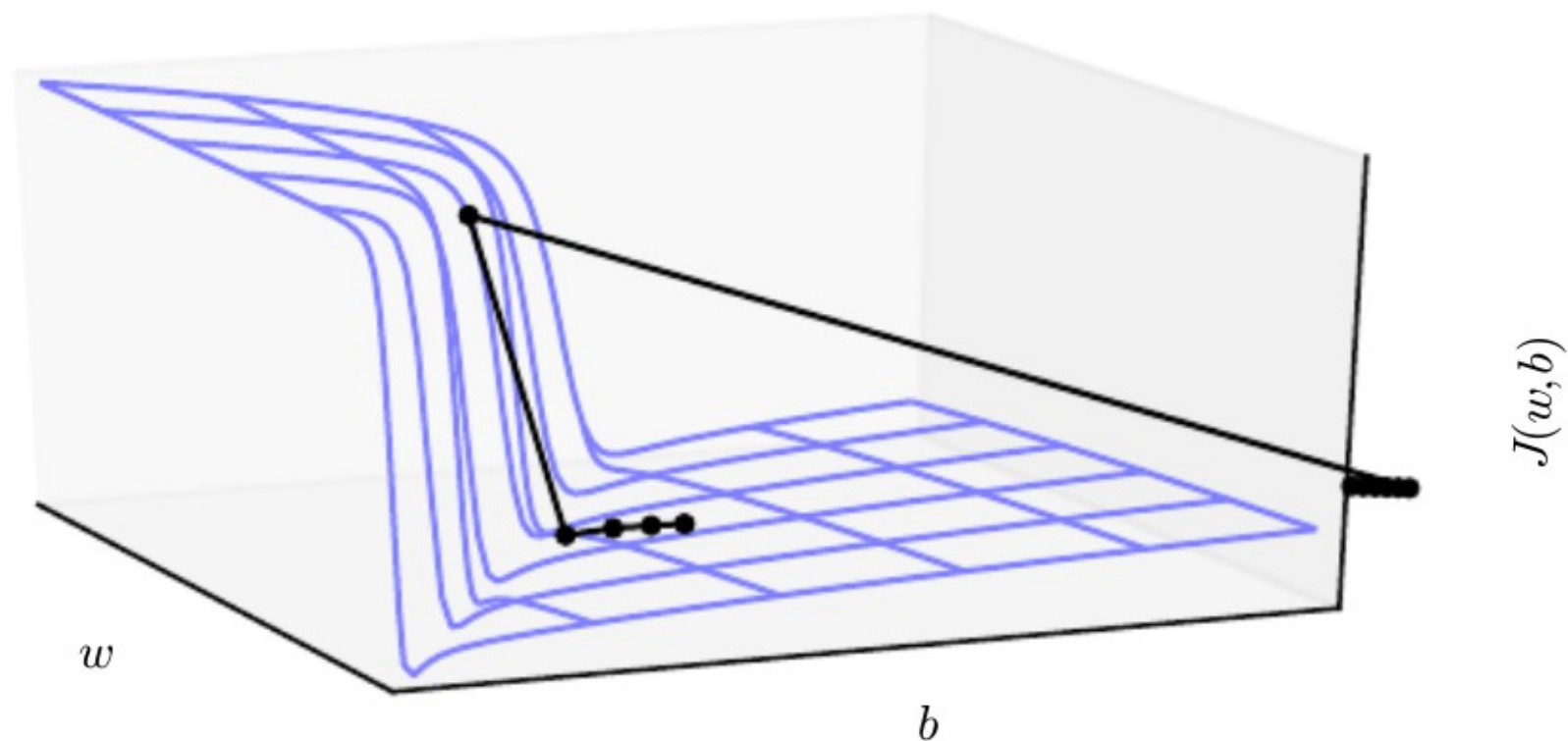
# 鞍点



随着  $n$  的增加，鞍点  
数远大于局域极小值：

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

# 权重系数和梯度爆炸



$$W = W1 * W2 * W3 \dots$$

# 激活（S）函数和梯度消失

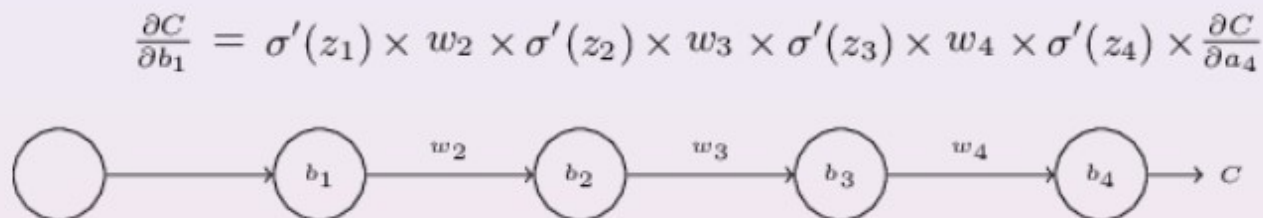


图 3.3.1: 多层单神经网络

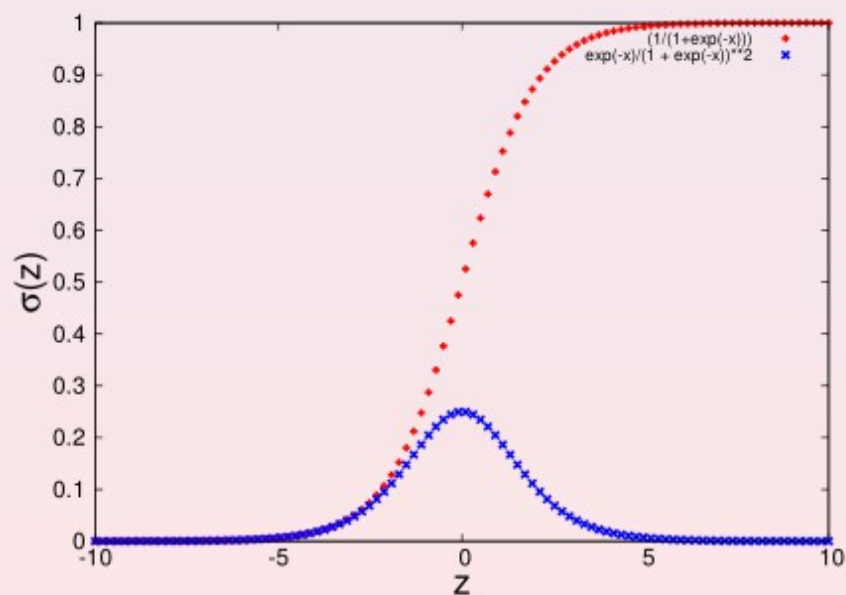


图 3.3.2: 激活函数及其导数

# 随机梯度下降法

---

**Algorithm 8.1** Stochastic gradient descent (SGD) update at training iteration  $k$

---

**Require:** Learning rate  $\epsilon_k$ .

**Require:** Initial parameter  $\theta$ .

递减!

**while** stopping criterion not met **do**

Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

Compute gradient estimate:  $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Apply update:  $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

**end while**

---

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

1、 $\epsilon_\tau$  通常为  $\epsilon_0$  的 1% ;

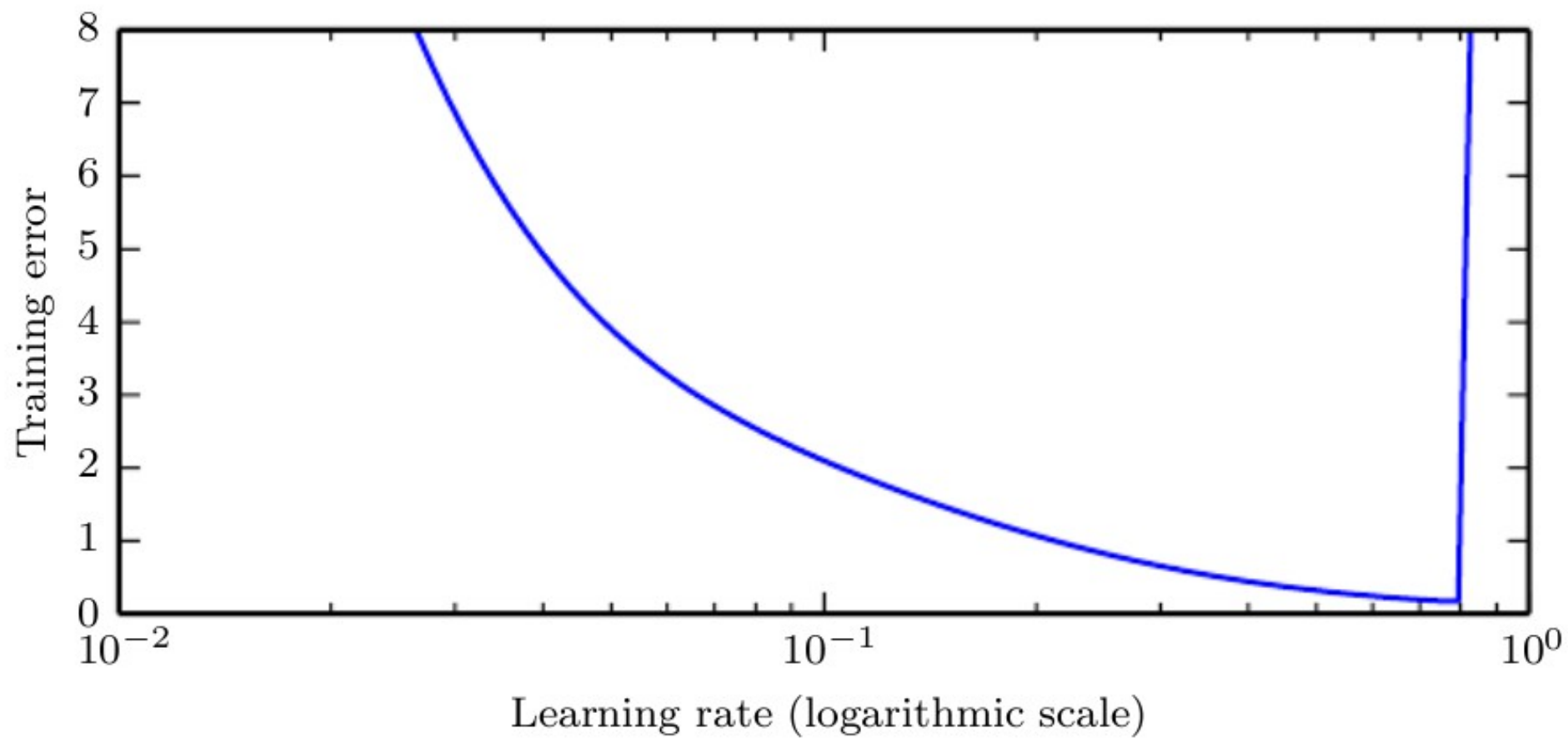
2、 $\epsilon_0$  通过试错法确定;

优点: 单次迭代的计算量

不随样本数目增加;

缺点: 收敛慢;





# SGD : 动量修正

---

**Algorithm 8.2** Stochastic gradient descent (SGD) with momentum

---

**Require:** Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .

**Require:** Initial parameter  $\theta$ , initial velocity  $v$ .

**while** stopping criterion not met **do**

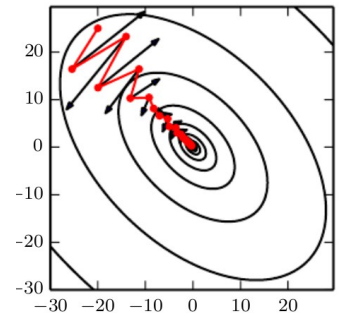
    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient estimate:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Compute velocity update:  $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

    Apply update:  $\theta \leftarrow \theta + \mathbf{v}$

**end while**



# Nesterov 修正

---

**Algorithm 8.3** Stochastic gradient descent (SGD) with Nesterov momentum

---

**Require:** Learning rate  $\epsilon$ , momentum parameter  $\alpha$ .

**Require:** Initial parameter  $\theta$ , initial velocity  $v$ .

**while** stopping criterion not met **do**

Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding labels  $\mathbf{y}^{(i)}$ .

Apply interim update:  $\tilde{\theta} \leftarrow \theta + \alpha v$

Compute gradient (at interim point):  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$

Compute velocity update:  $v \leftarrow \alpha v - \epsilon \mathbf{g}$

Apply update:  $\theta \leftarrow \theta + v$

**end while**

---

对完整梯度下降法的收敛性有改观，而对随机梯度下降法无改进。

# 初始化：w & b

- 1、bias 取固定值，通常 1；
- 2、W 按照高斯 / 均匀分布取值（why？？）；
- 3、初始化 W 过小，导致无效神经元；W 过大，导致梯度爆炸，或者激活函数到达饱和状态；

# 学习率： 自适应取值

---

**Algorithm 8.4** The AdaGrad algorithm

---

**Require:** Global learning rate  $\epsilon$

**Require:** Initial parameter  $\theta$

**Require:** Small constant  $\delta$ , perhaps  $10^{-7}$ , for numerical stability

Initialize gradient accumulation variable  $\mathbf{r} = \mathbf{0}$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

    Compute update:  $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ . (Division and square root applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta\theta$

**end while**

适用于凸优化；

---

**Algorithm 8.5** The RMSProp algorithm

---

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$ .

**Require:** Initial parameter  $\boldsymbol{\theta}$

**Require:** Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers.

Initialize accumulation variables  $\mathbf{r} = 0$

**while** stopping criterion not met **do**

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

    Compute parameter update:  $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{\delta + \mathbf{r}}}$  applied element-wise)

    Apply update:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

**end while**

---

适用于非凸优化;

---

**Algorithm 8.6** RMSProp algorithm with Nesterov momentum

---

**Require:** Global learning rate  $\epsilon$ , decay rate  $\rho$ , momentum coefficient  $\alpha$ .

**Require:** Initial parameter  $\boldsymbol{\theta}$ , initial velocity  $\boldsymbol{v}$ .

Initialize accumulation variable  $\boldsymbol{r} = \mathbf{0}$

**while** stopping criterion not met **do**

Sample a minibatch of  $m$  examples from the training set  $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$  with corresponding targets  $\boldsymbol{y}^{(i)}$ .

Compute interim update:  $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Compute gradient:  $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Accumulate gradient:  $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute velocity update:  $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ . ( $\frac{1}{\sqrt{\boldsymbol{r}}}$  applied element-wise)

Apply update:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

**end while**

---

# 牛顿法

---

**Algorithm 8.8** Newton's method with objective  $J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$ .

---

**Require:** Initial parameter  $\boldsymbol{\theta}_0$

**Require:** Training set of  $m$  examples

**while** stopping criterion not met **do**

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

    Compute Hessian:  $\mathbf{H} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}}^2 \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$

    Compute Hessian inverse:  $\mathbf{H}^{-1}$

    Compute update:  $\Delta\boldsymbol{\theta} = -\mathbf{H}^{-1}\mathbf{g}$

    Apply update:  $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end while**

---

计算量极大,  $O(3)$



# 其它优化算法

CG ;

BFGS ;

L-BFGS ;

.....

# 超参数和模型容量

超参数	如何增加容量	原因	注意事项
隐藏神经元数目	增加	直接增加模型表现力	增加计算和存储需求
学习速率	动态（最优）调整		过大过小均会导致优化失败
卷积核的大小	增加	模型参数个数增加 直接增加模型表现力	核越大，计算和存储需求越大

超参数	如何增加容量	原因	注意事项
0 填充	增加	增加表示的大小	增加计算和存储需求
权重衰减系数	减少	在同一个拉格朗日量里，当减少衰减因子，相当于增加模型大小	
dropout 比率	减少	减少 dropout 意味着增加模型的神经元数目	

算法设计时，应该尽量减少超参数。

# 表示（表象）学习

类似于坐标系选择；

简化学习过程；

精心选择，经验积累；

表象选择的系统方法？？

谢谢！