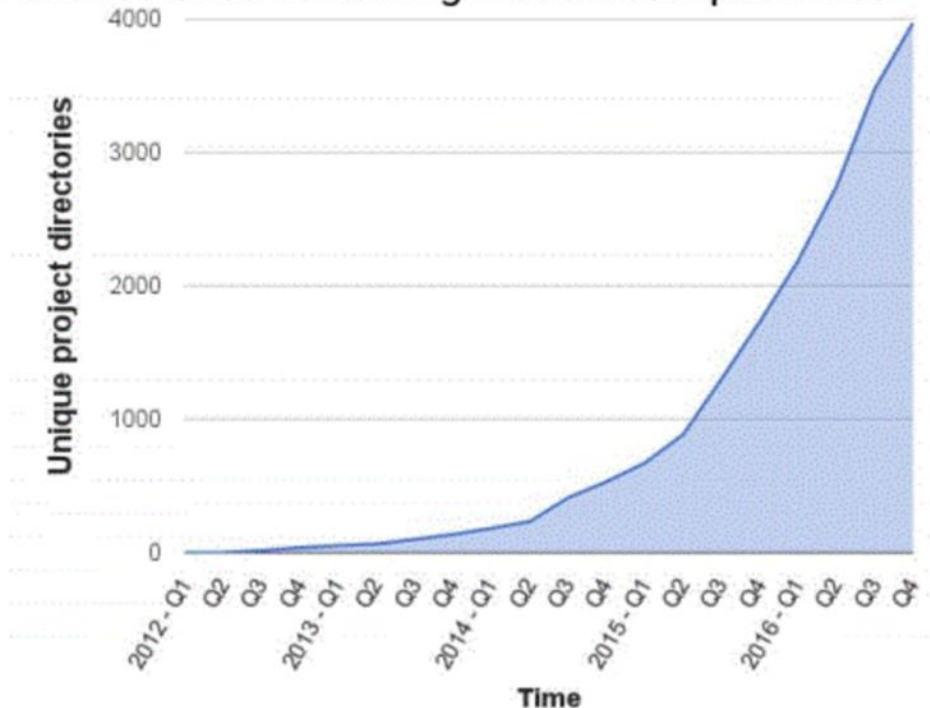


第14章 深度学习

Growing Use of Deep Learning at Google

of directories containing model description files

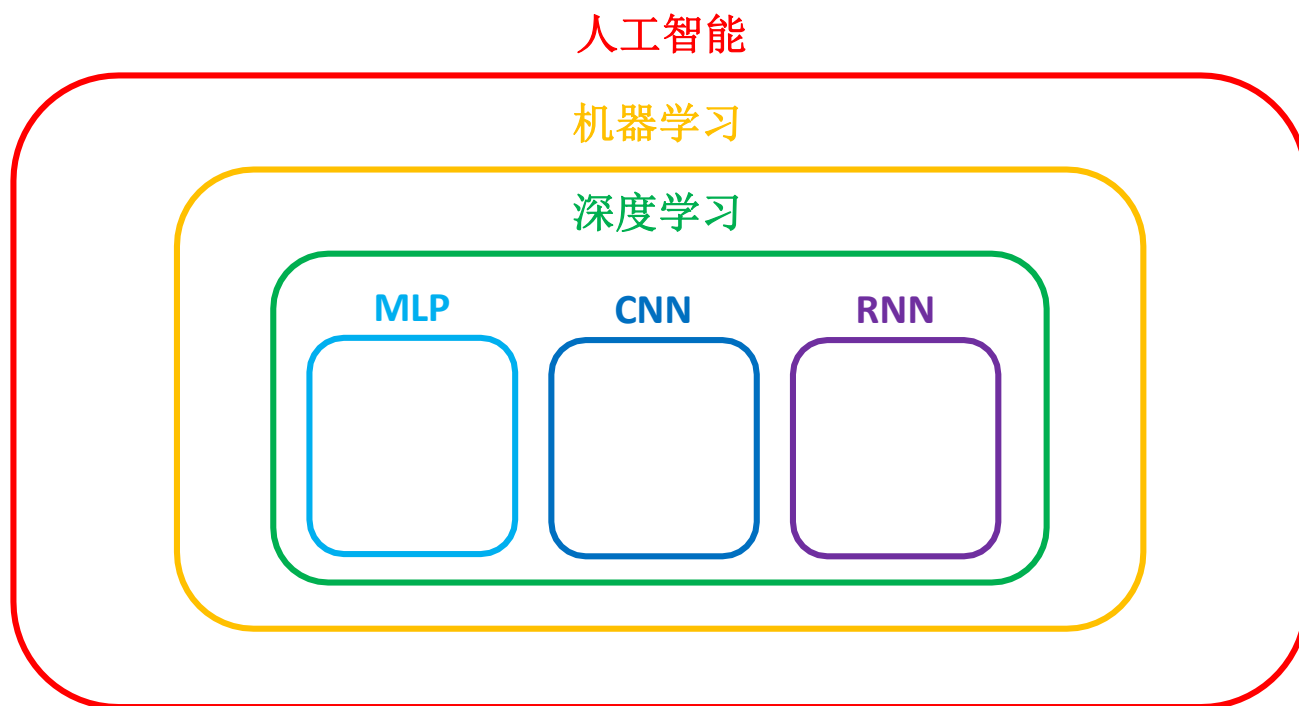


Across many products/areas:

Android
Apps
drug discovery
Gmail
Image understanding
Maps
Natural language understanding
Photos
Robotics research
Speech
Translation
YouTube
... many others ...

什么是深度学习？

- 【Wikipedia的定义】 **Deep learning** is part of a broader family of machine learning methods based on **artificial neural networks**.



深度学习发展简史

起源：

- **1943年**，神经学家Warren McCulloch（麦卡洛克）和数学家Walter Pitts（皮茨）提出第一个**脑神经元**的抽象模型（M-P模型）

深度学习发展简史

第一次浪潮：

- **1958年**，计算机科学家Frank Rosenblatt（罗森布拉特）基于M-P神经元模型提出第一个**感知机**模型
- **1969年**，MIT的Marvin Minsky（明斯基）和Seymour Papert（帕尔特）出版了《Perceptron》一书，分析了感知机模型的局限性，神经网络研究进入“冰河期”

深度学习发展简史

第二次浪潮：

- **1974年**，Paul Werbos（沃博斯）发明了BP算法，因正值“冰河期”，未受到重视
- **1986年**，D.E. Rumelhart，G.E. Hinton 和 R.J. Williams发表论文，阐释并推广了**反向传播算法**（Back Propagation, BP），并采用Sigmoid函数，有效解决了非线性分类问题
- **1991年**，BP算法被指出存在梯度消失问题

深度学习发展简史

第三次浪潮：

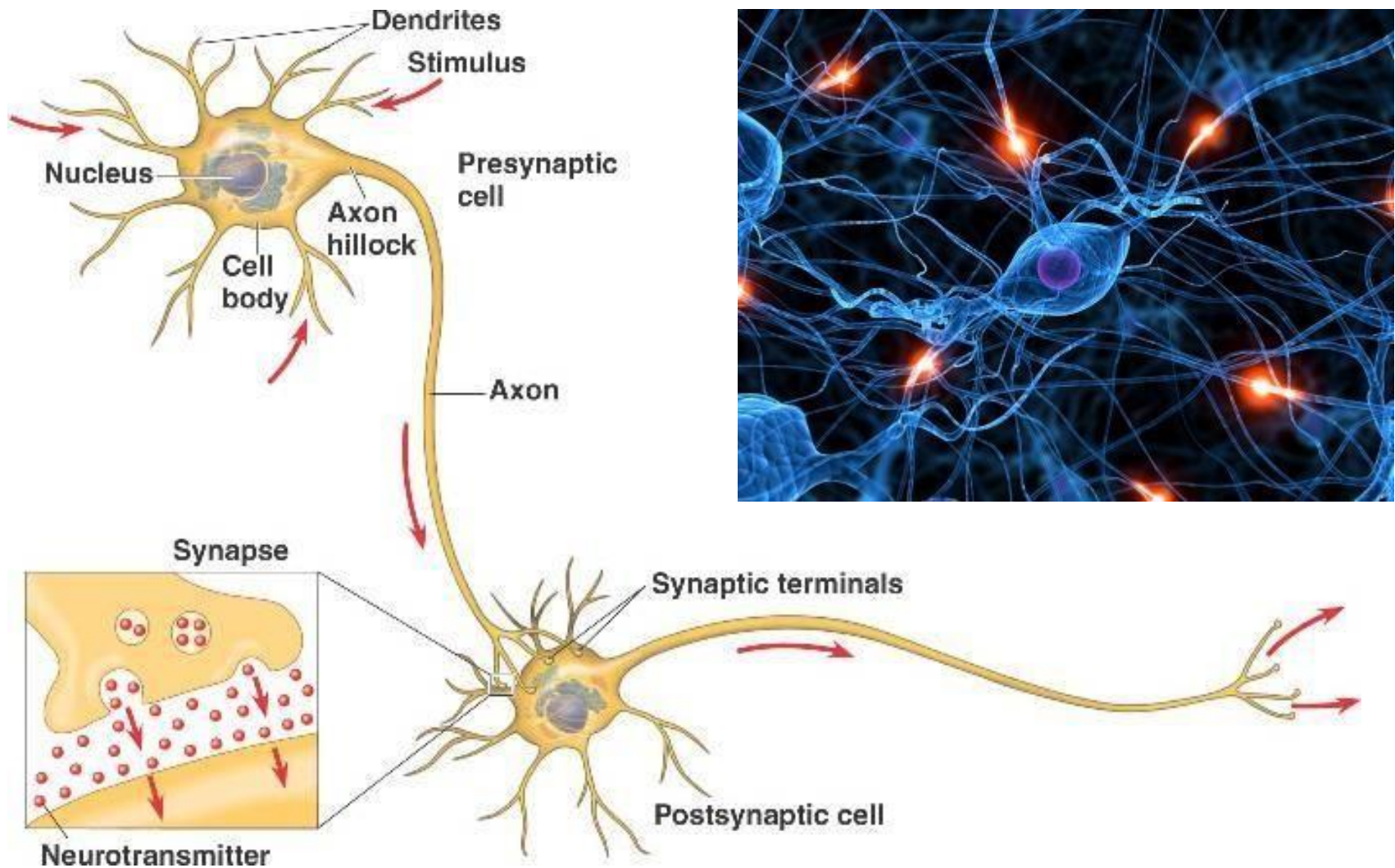
- **2006年**，Hinton等人提出一种可以实现快速学习的**深度**信念网络（DBN）
- **2009年**，**GPU**被用于训练深度神经网络
- **2011年**，深度学习用于语音辨识
- **2012年**，欣顿课题组首次使用深度学习在ImageNet**图像识别**竞赛中夺冠，高出第二名（SVM）10个百分点

深度学习发展简史

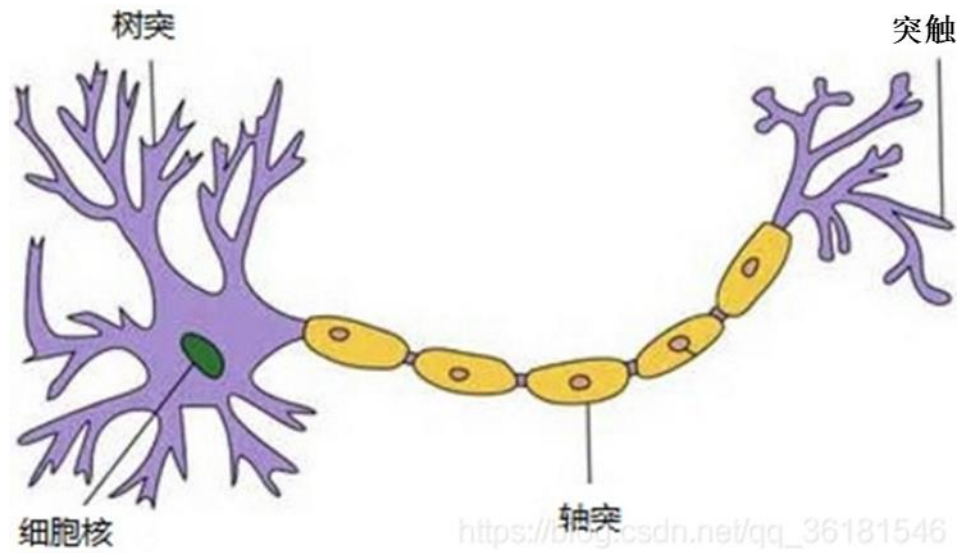
第三次浪潮：

- **2015年**，微软的ResNet以96.43%的Top5精度，超过人类水平（94.9%）
- **2016年**，AlphaGo击败**围棋**世界冠军Lee Sedol
- **2017年**，AlphaGo Zero以100:0击败AlphaGo
- **2017年**，IBM将**语音识别**错误率降到5.5%（人类5.1%）

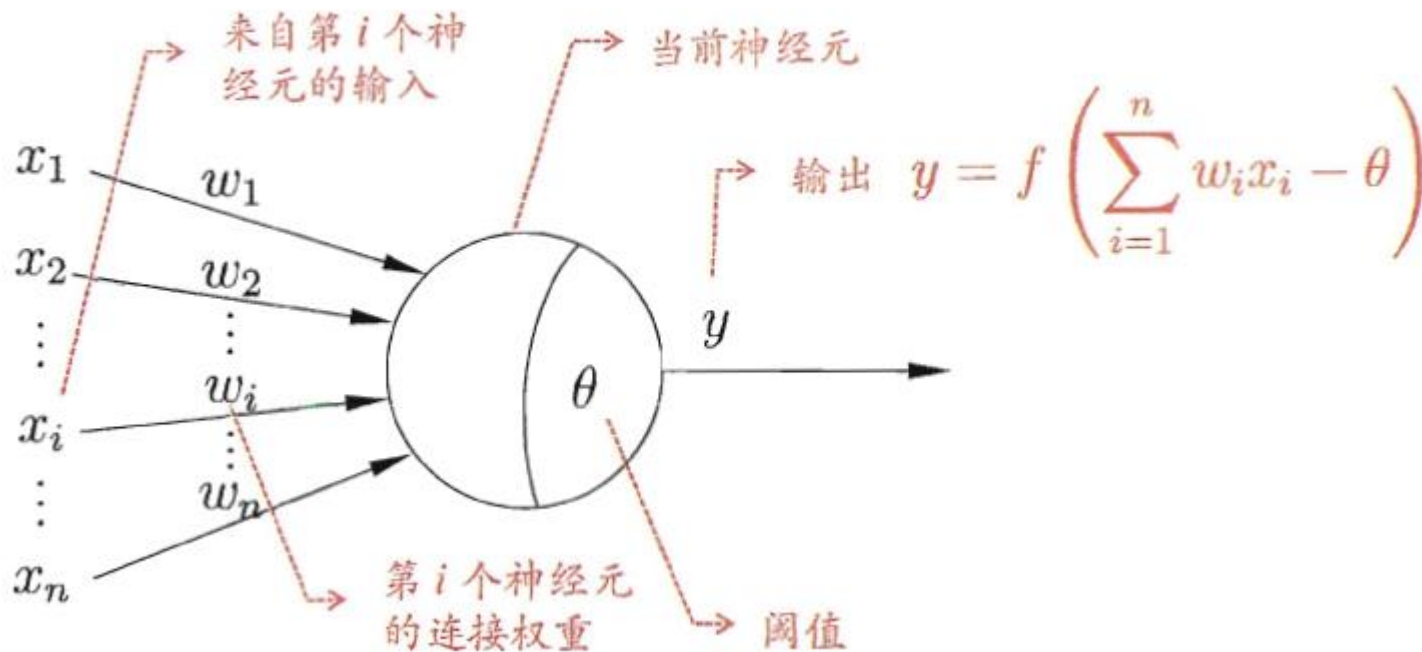
人脑神经元



神经元模型

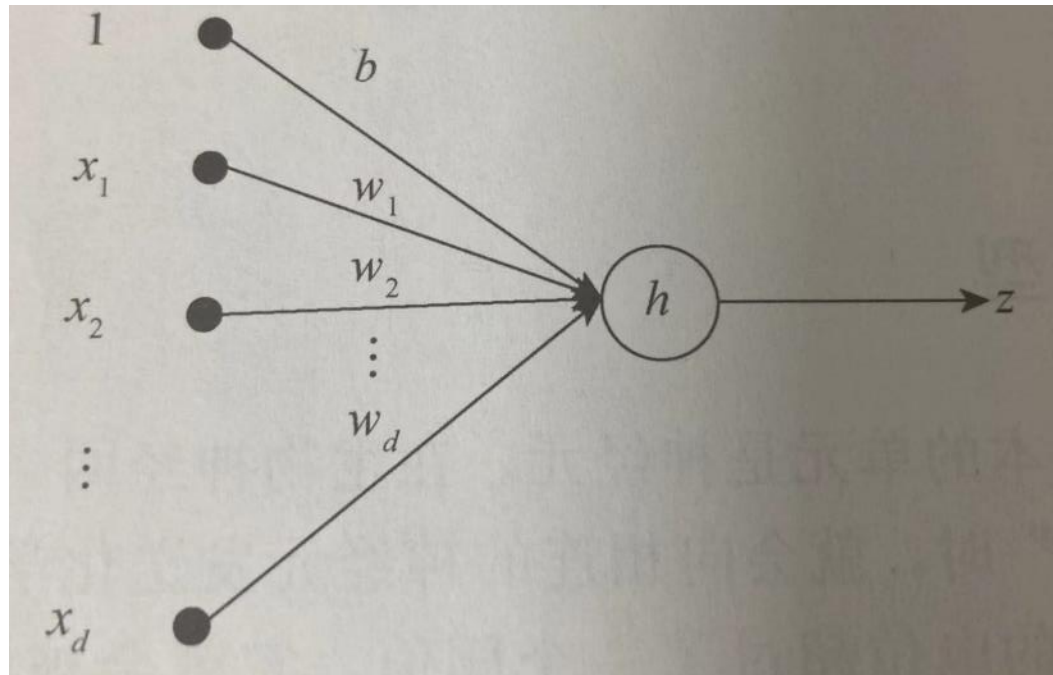


神经元模型



M-P神经元模型

神经元模型

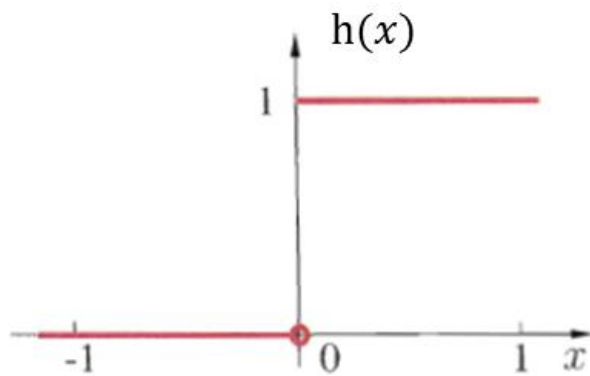


人工神经元模型

激活函数

施加在线性加权求和之上的函数。

1. 阶跃函数

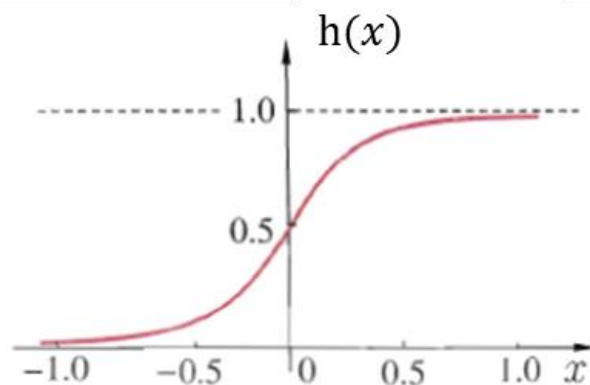


$$\text{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

激活函数

施加在线性加权求和之上的函数。

2. Sigmoid函数



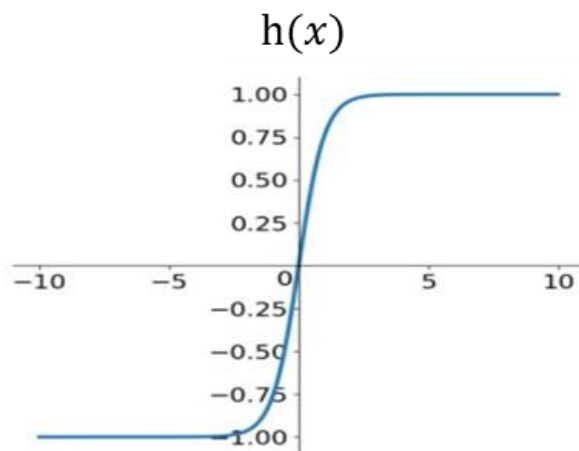
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

$$h'(x) = h(x)(1 - h(x))$$

激活函数

施加在线性加权求和之上的函数。

3. 双曲正切函数

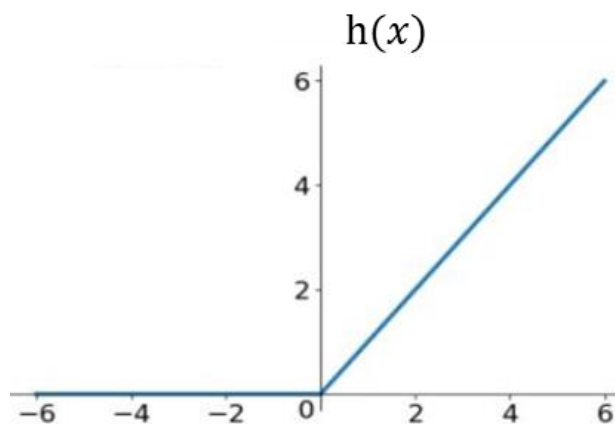


$$\text{Tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

激活函数

施加在线性加权求和之上的函数。

4. ReLU函数



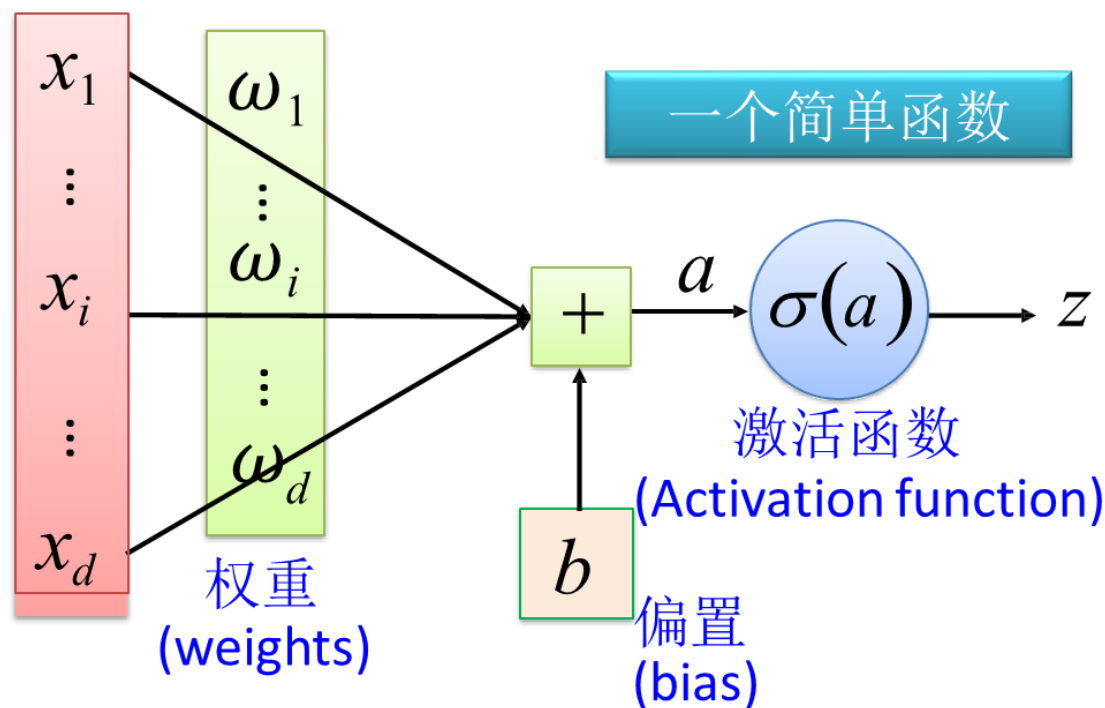
$$h(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

感知机与多层感知机

——人工神经网络 (ANN)

神经元 (Neuron)

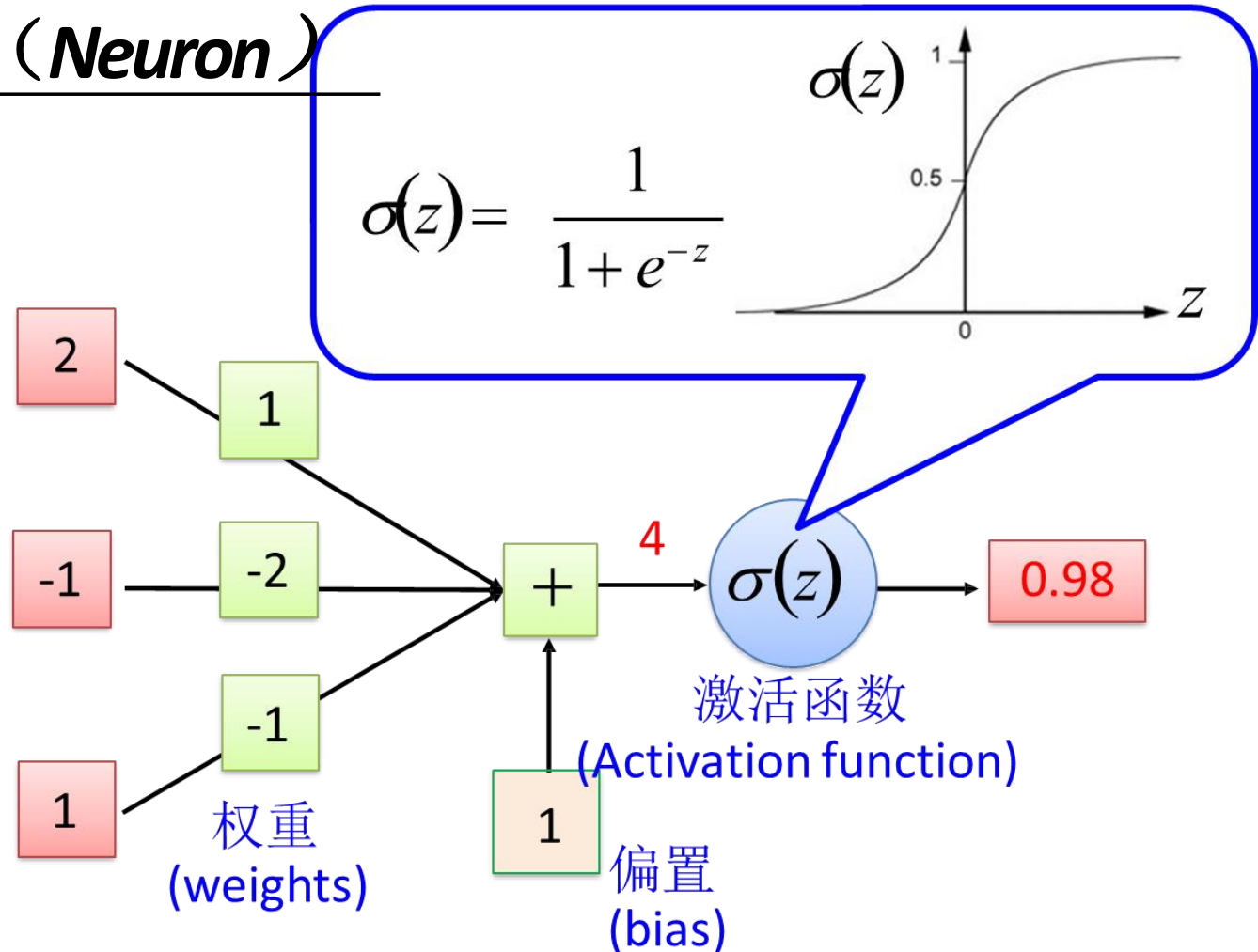
$$z = h(x_1\omega_1 + x_2\omega_2 + \cdots + x_d\omega_d + b)$$



感知机与多层感知机

——人工神经网络 (ANN)

神经元 (Neuron)

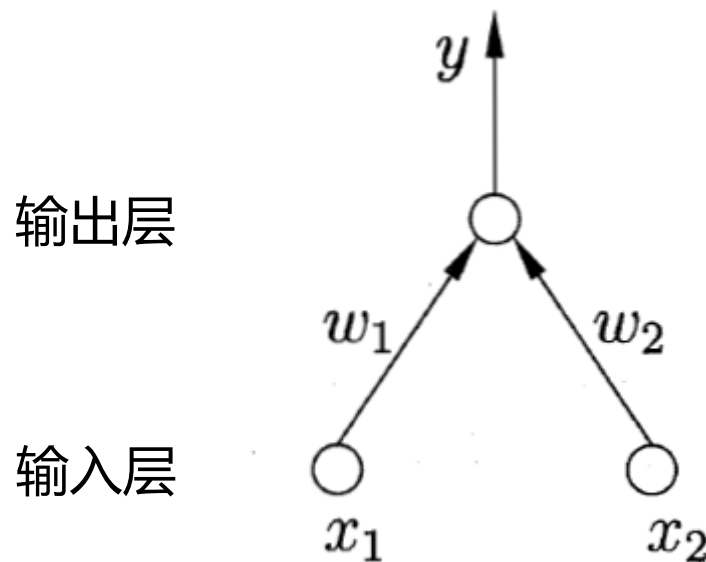


感知机与多层感知机

——人工神经网络（ANN）

感知机（Perceptron）

由两层神经元组成，输入层接收外界信号后传递给输出层，输出层是M-P神经元，也叫阈值逻辑单元



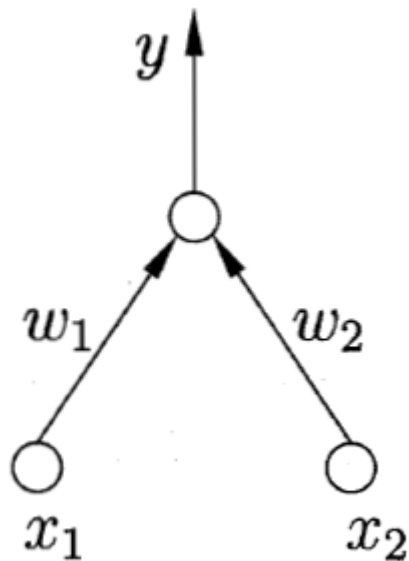
$$y = h(\sum x_i \omega_i - \theta)$$

感知机与多层感知机

——人工神经网络 (ANN)

感知机 (Perceptron)

能容易地实现与、或、非运算



与：令 $\omega_1 = \omega_2 = 1$, $\theta = 2$

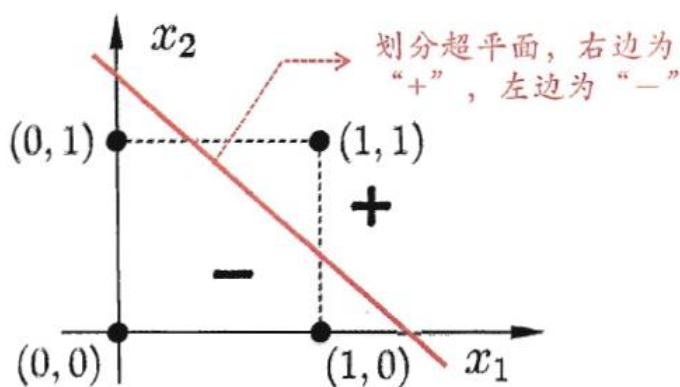
或：令 $\omega_1 = \omega_2 = 1$, $\theta = 0.5$

非：令 $\omega_1 = -0.6$, $\omega_2 = 0$,
 $\theta = -0.5$

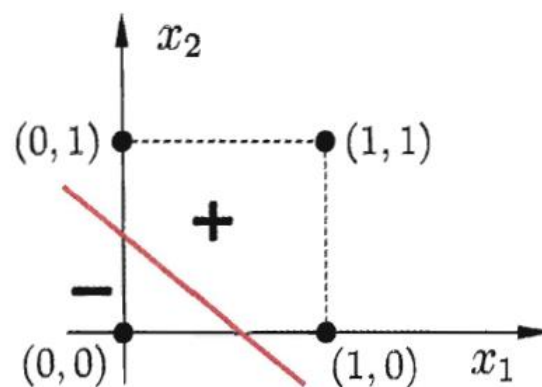
感知机与多层感知机

——人工神经网络 (ANN)

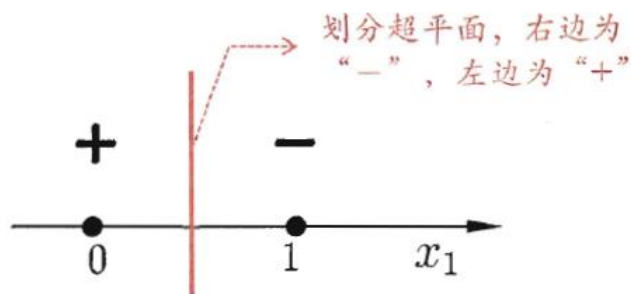
感知机 (Perceptron)



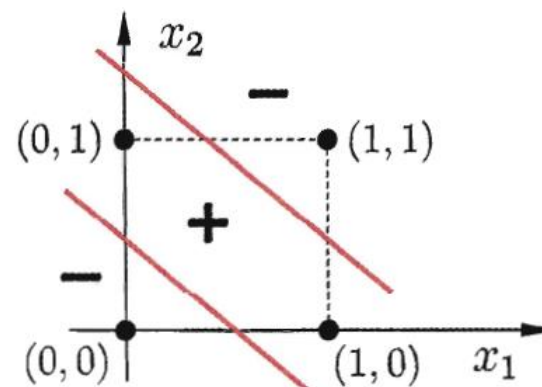
与



或



与

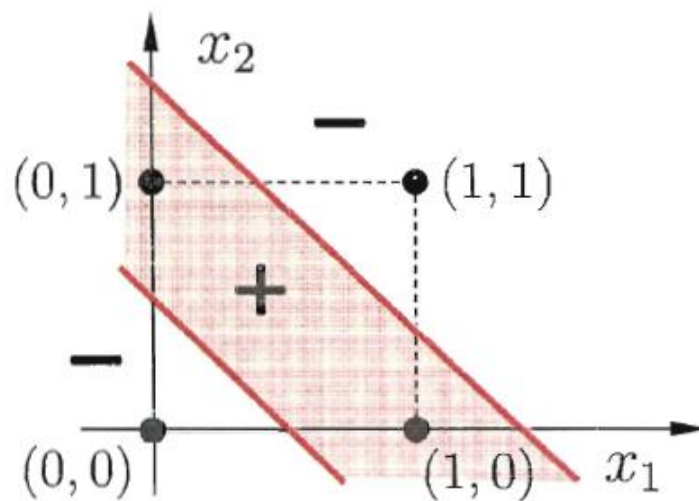
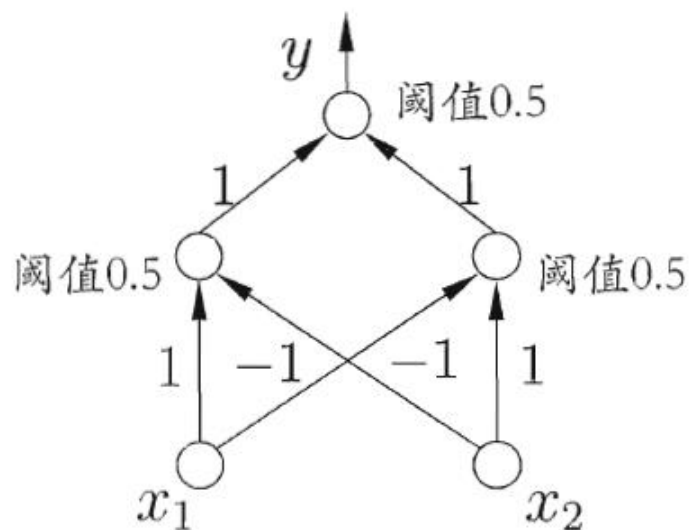


异或

感知机与多层感知机

——人工神经网络 (ANN)

多层感知机 (Multi Layer Perceptron)

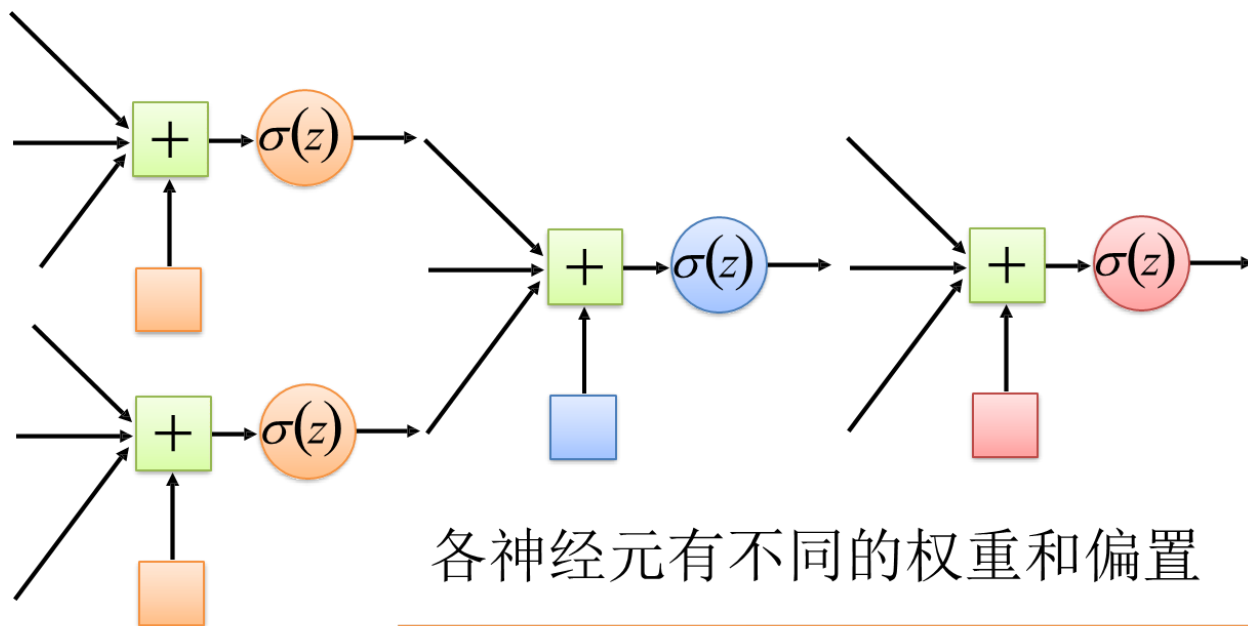


感知机与多层感知机

——人工神经网络 (ANN)

多层感知机 (Multi Layer Perceptron)

不同的连接将产生不同的网络结构




各神经元有不同的权重和偏置

权重和偏置是神经网络的参数 θ

机器学习 \approx 寻找一个函数


- 语音识别

$$f(\text{ $$

- 图像识别

$$f(\text{ $$

- 下围棋

$$f(\text{ 

(下一步落子)$$

- 人机会话

$$f(\text{"Hi"} \quad \text{(用户说)}) = \text{"Hello"} \quad \text{(系统回应)}$$

深度学习三部曲

第一步：
构建神经网络

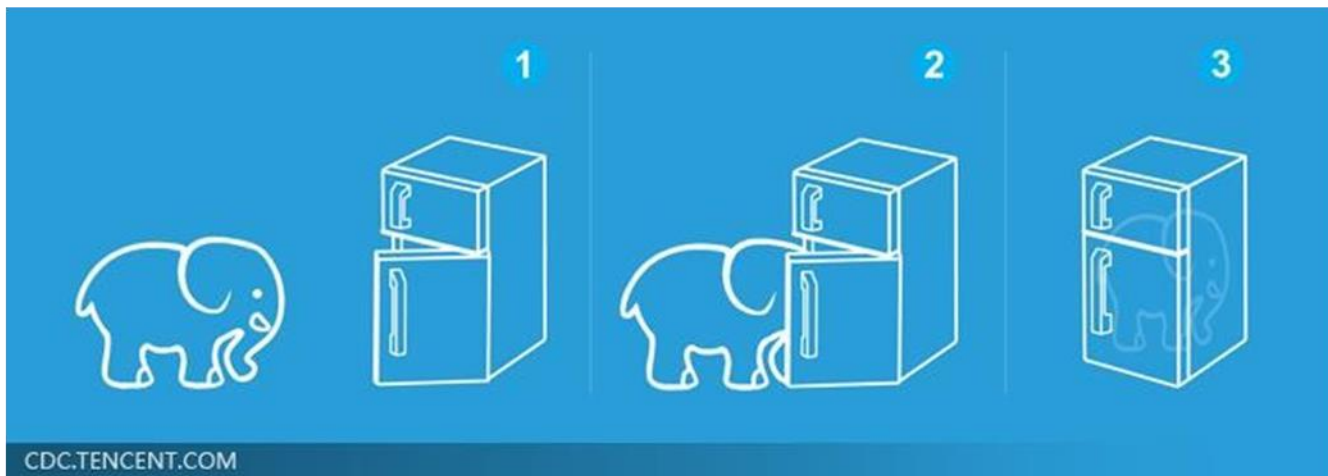


第二步：
确定学习目标

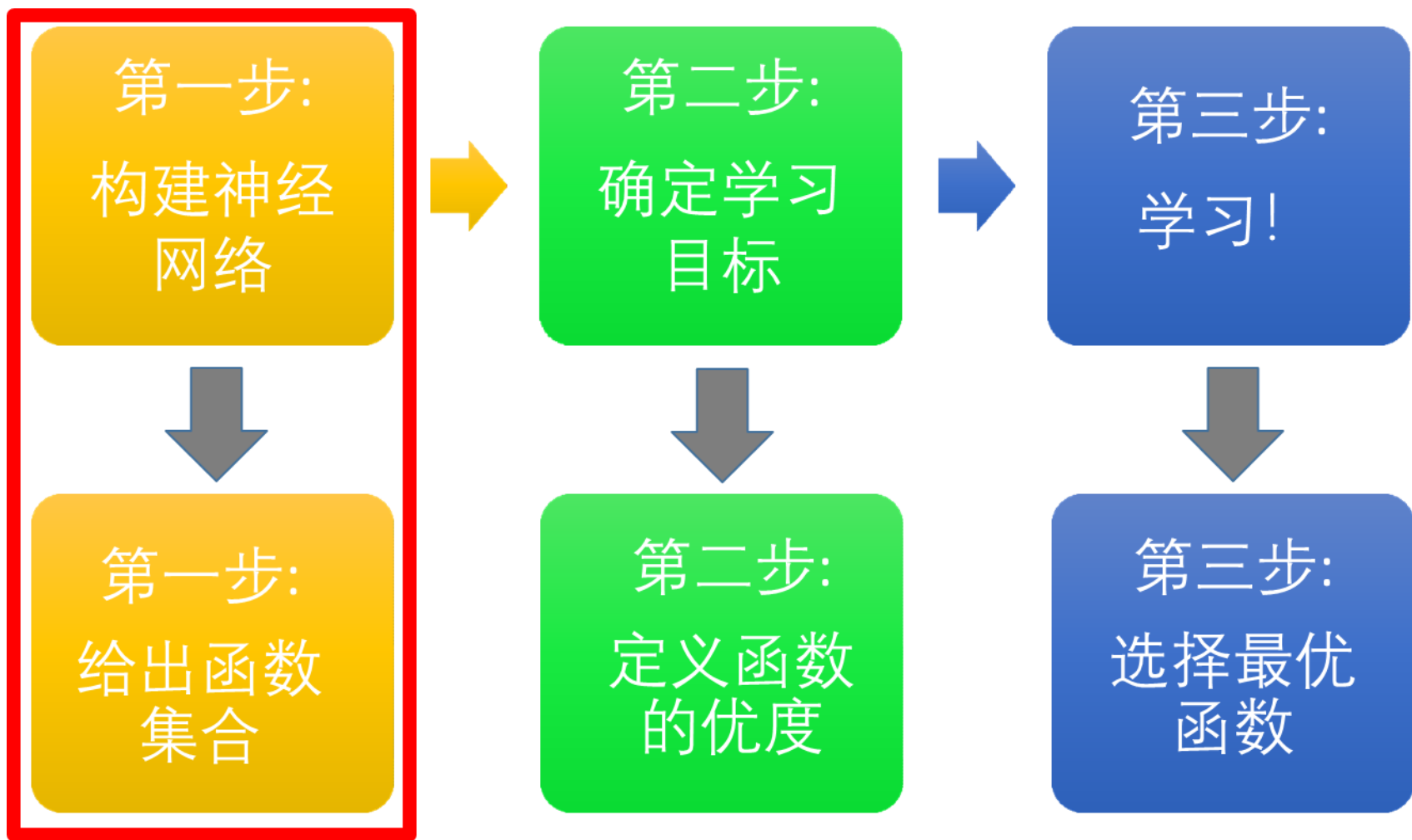


第三步：
学习!

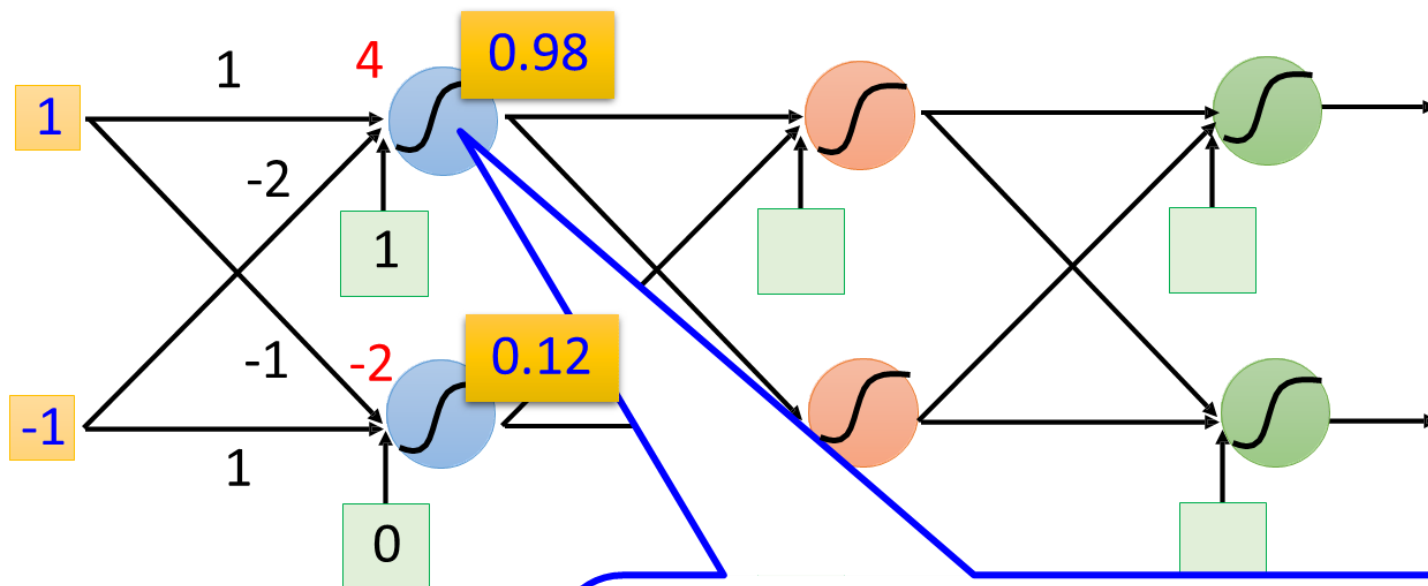
深度学习很简单



深度学习三部曲

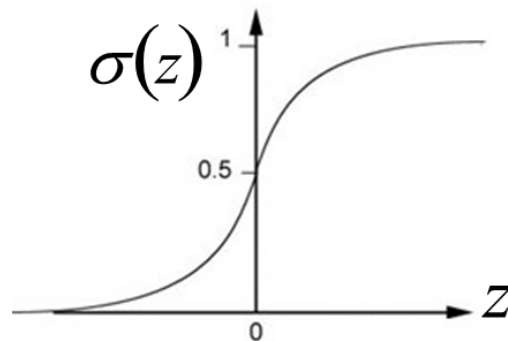


全连接前馈网络

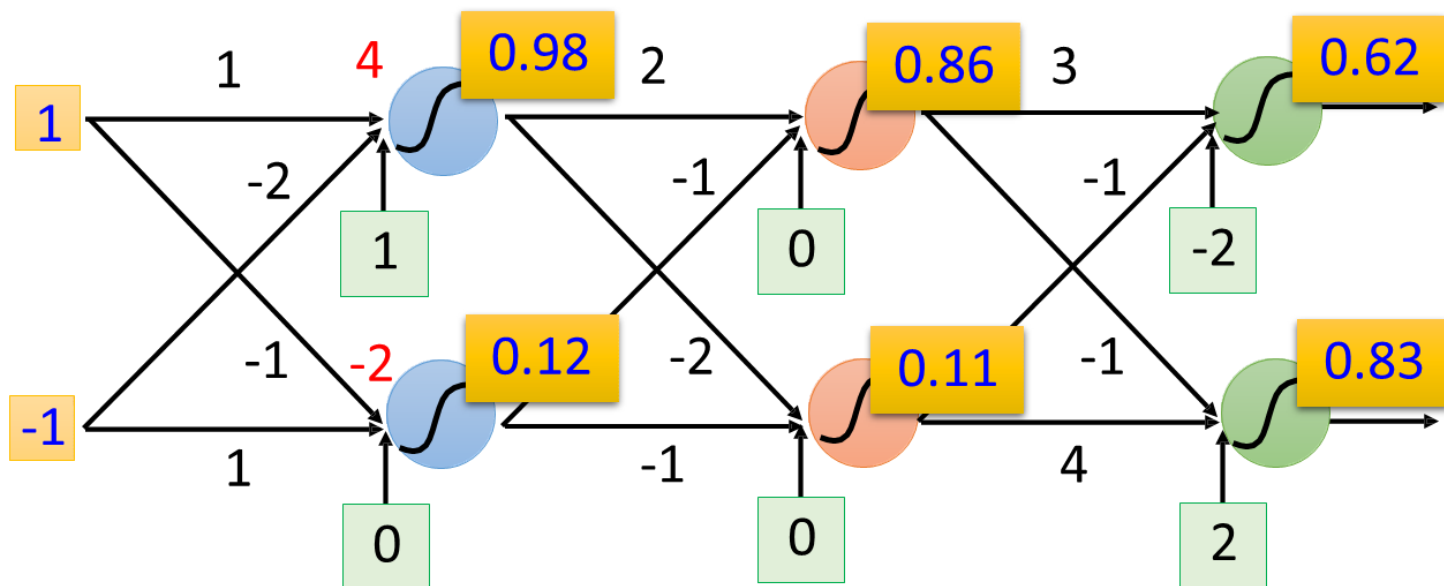


Sigmoid 函数

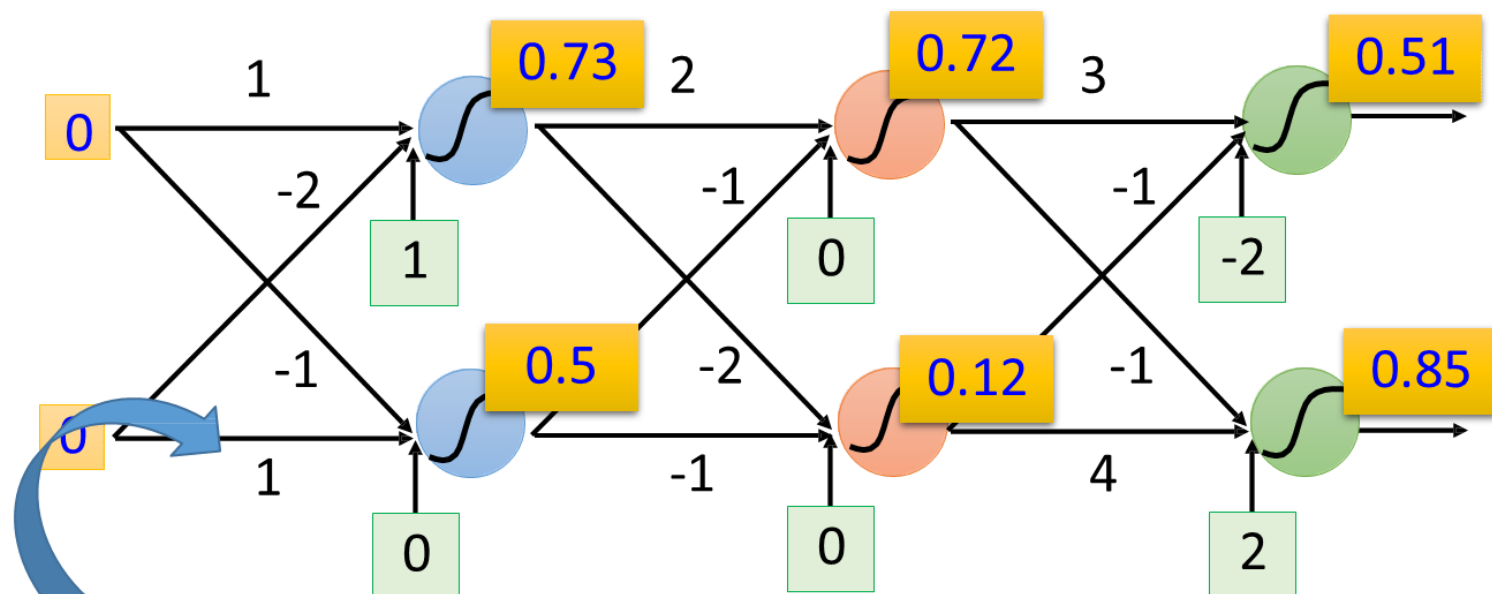
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



全连接前馈网络



全连接前馈网络

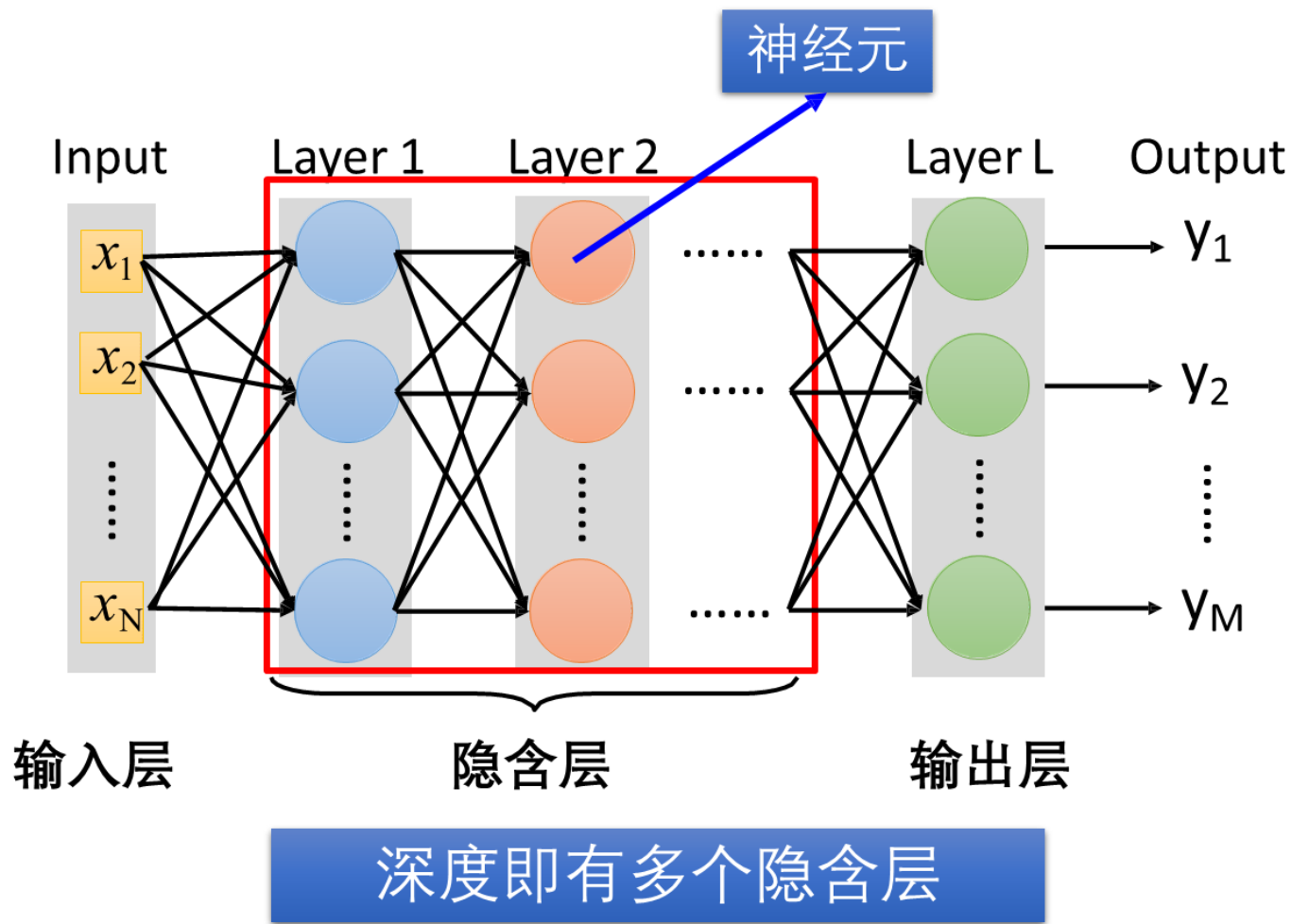


这是一个函数.
输入向量, 输出向量

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

给定网络结构, 就定义了一组函数

全连接前馈网络



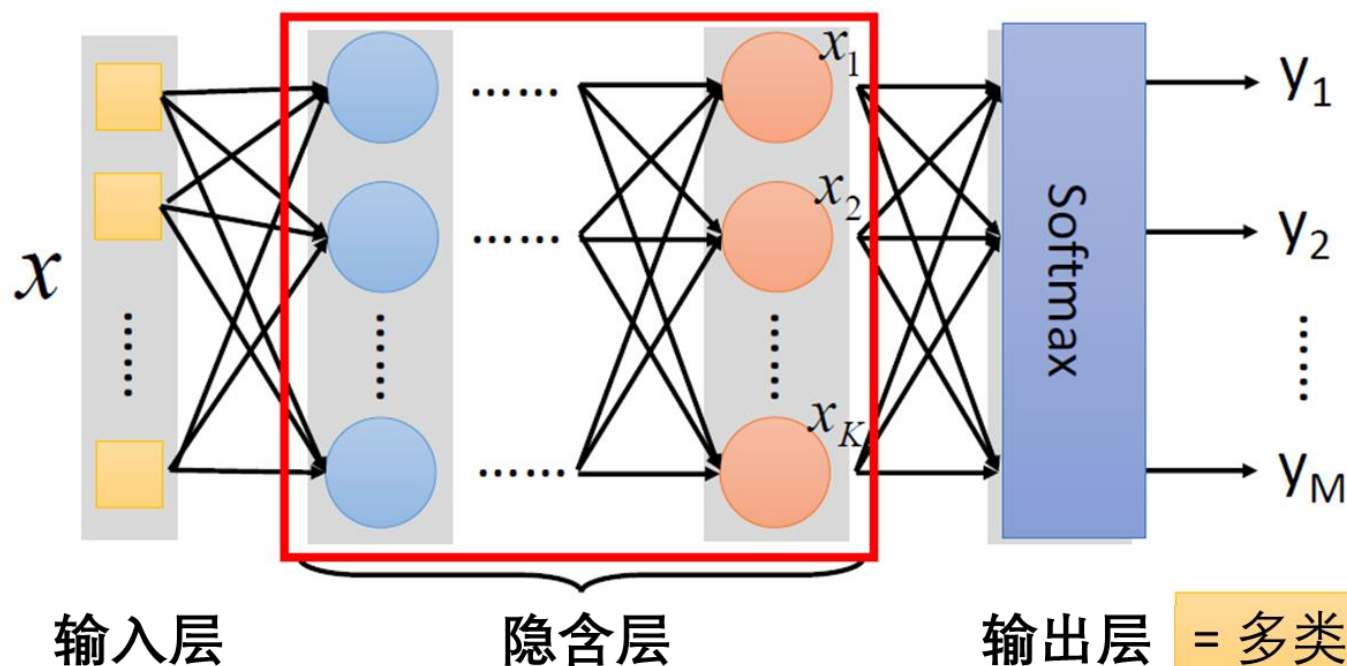
输出层（多类分类器）

特征抽取替代了特征工程

Probability:

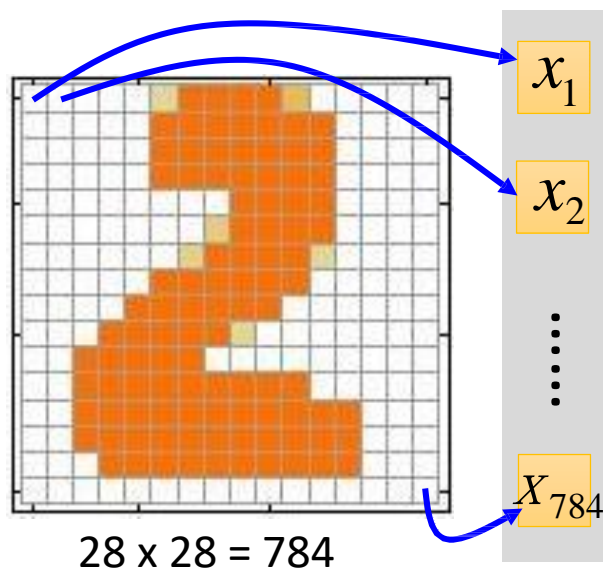
■ $1 > y_i > 0$

■ $\sum_i y_i = 1$



案例应用：手写数字识别

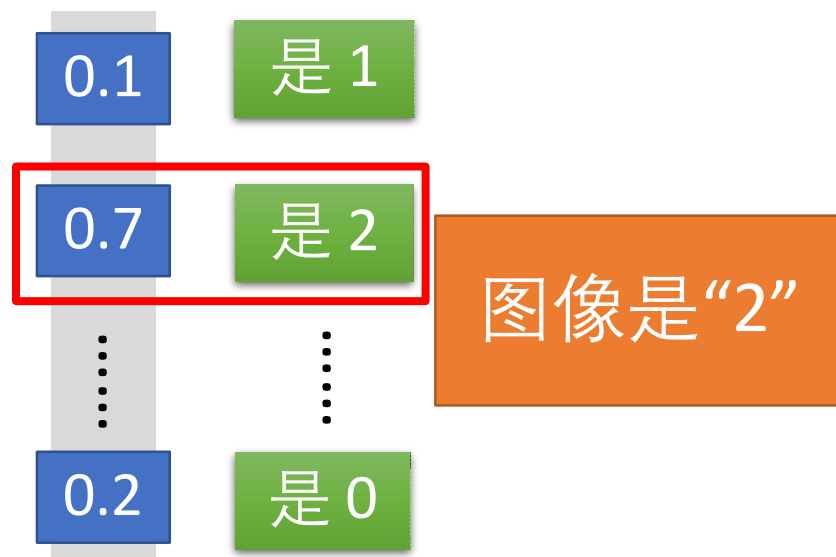
输入



墨水 $\rightarrow 1$

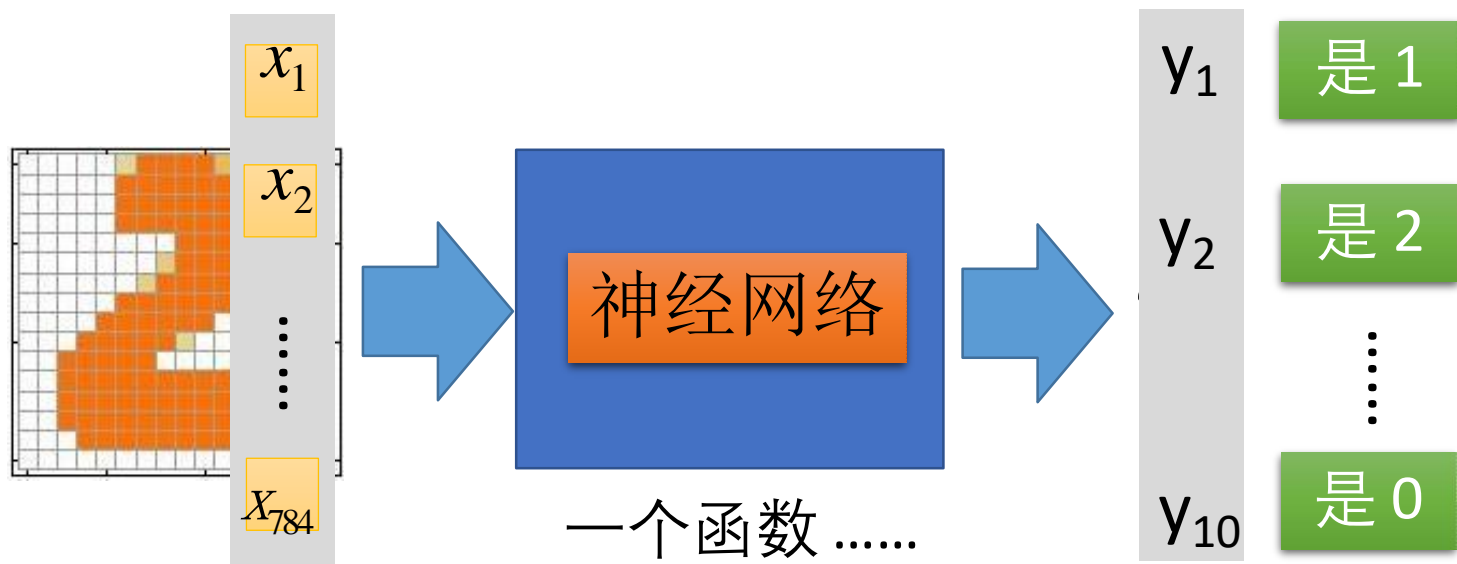
无墨水 $\rightarrow 0$

输出



每一维表示预测为一个数字的概率

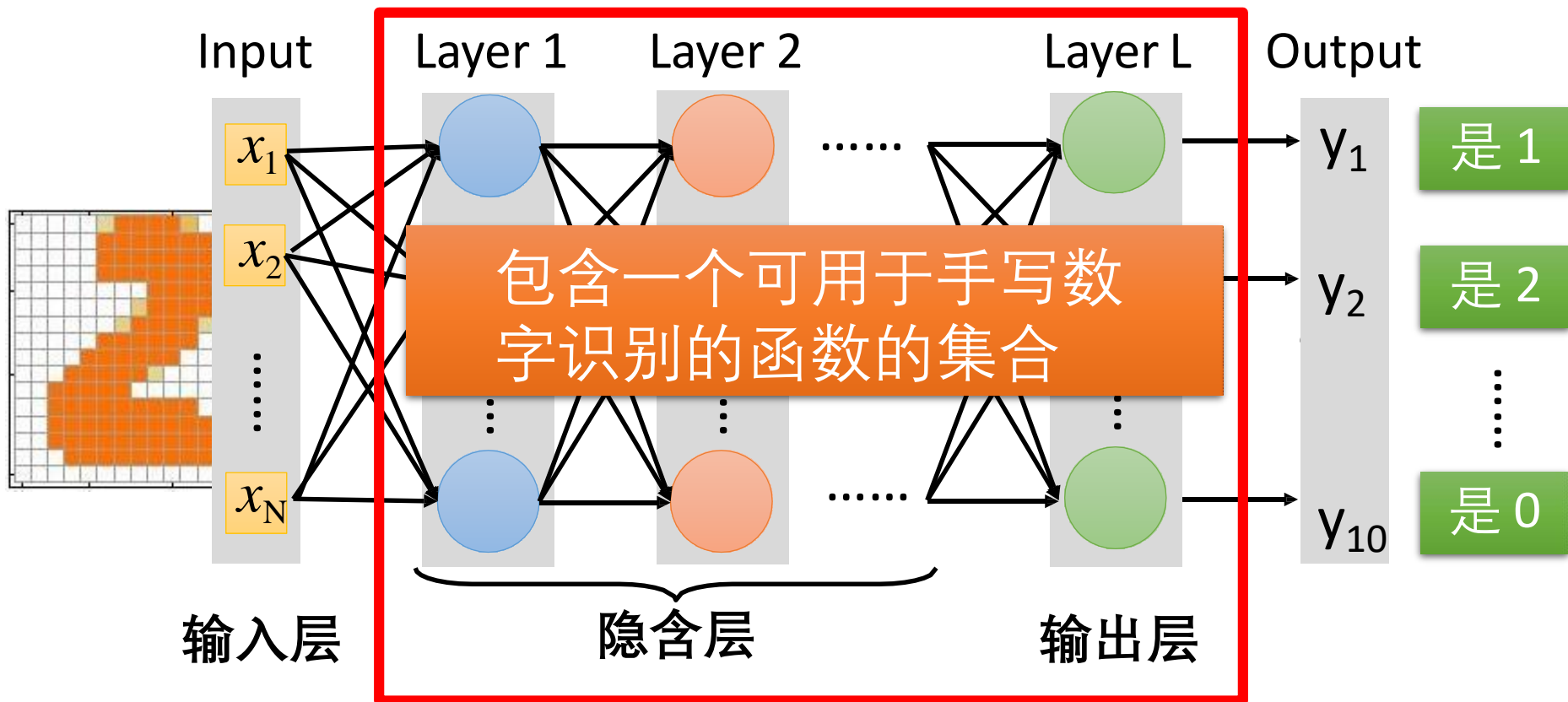
案例应用：手写数字识别



输入：
784维向量

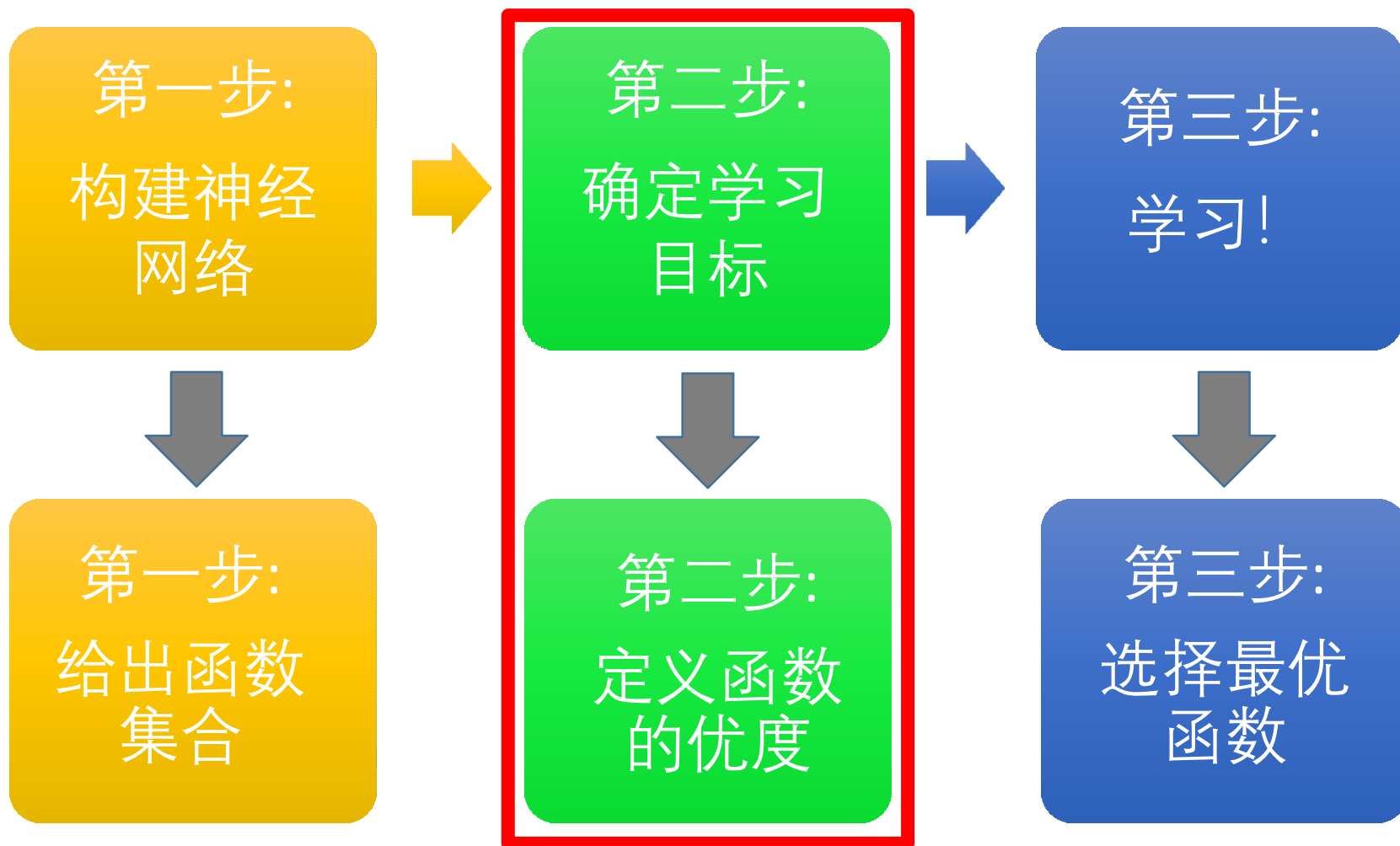
输出：
10维向量

案例应用：手写数字识别



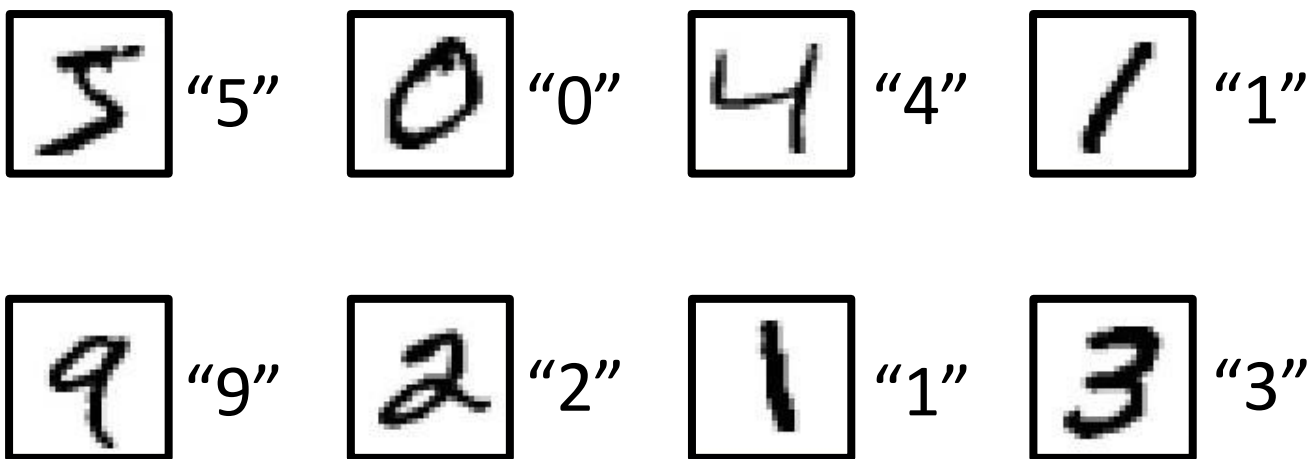
你需要构建一个神经网络结构以包含一个好的可用于手写数字识别的函数.

深度学习三部曲



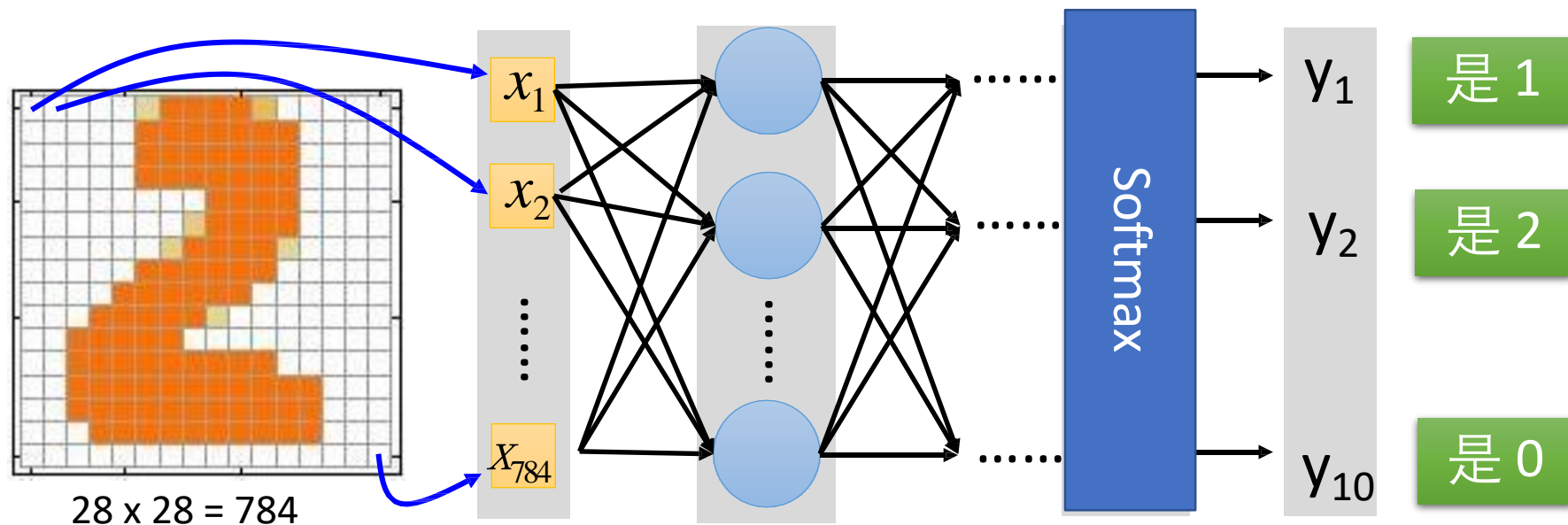
训练数据

- 准备训练数据: 图像及其标签




学习目标定义在训练数据之上


学习目标



墨水 \rightarrow 1
无墨水 \rightarrow 0

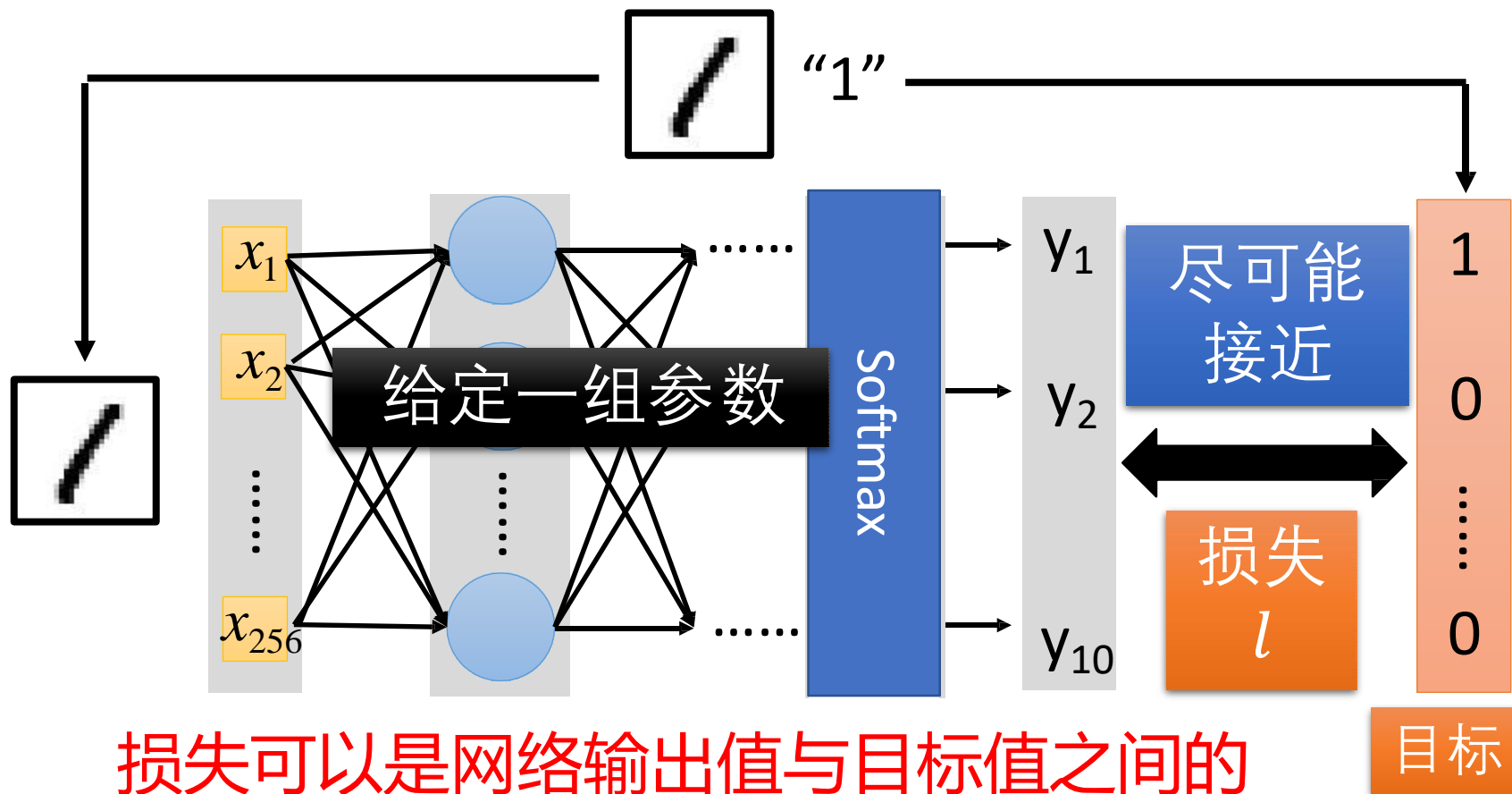
学习目标是:

输入:  $\rightarrow y_1$ 值最大

输入:  $\rightarrow y_2$ 值最大

损失

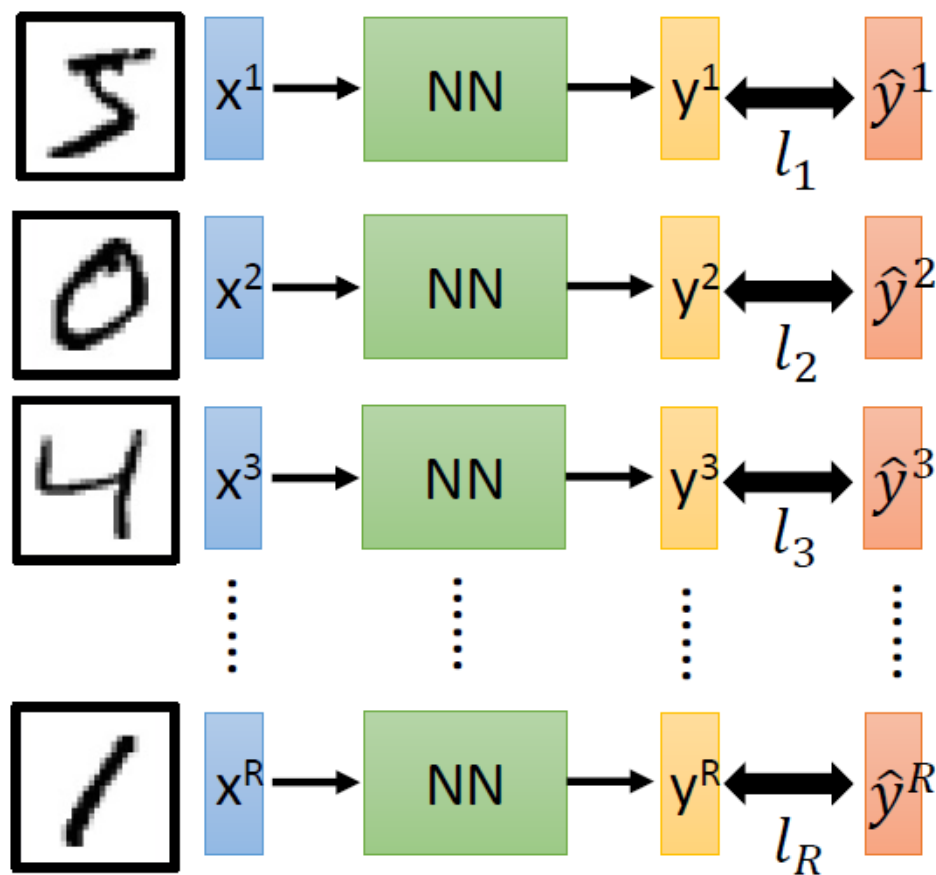
一个好的函数应该使所有的训练样例产生的损失尽可能小.



损失可以是网络输出值与目标值之间的
均方误差或交叉熵

整体损失

对于所有的训练数据...



整体损失:

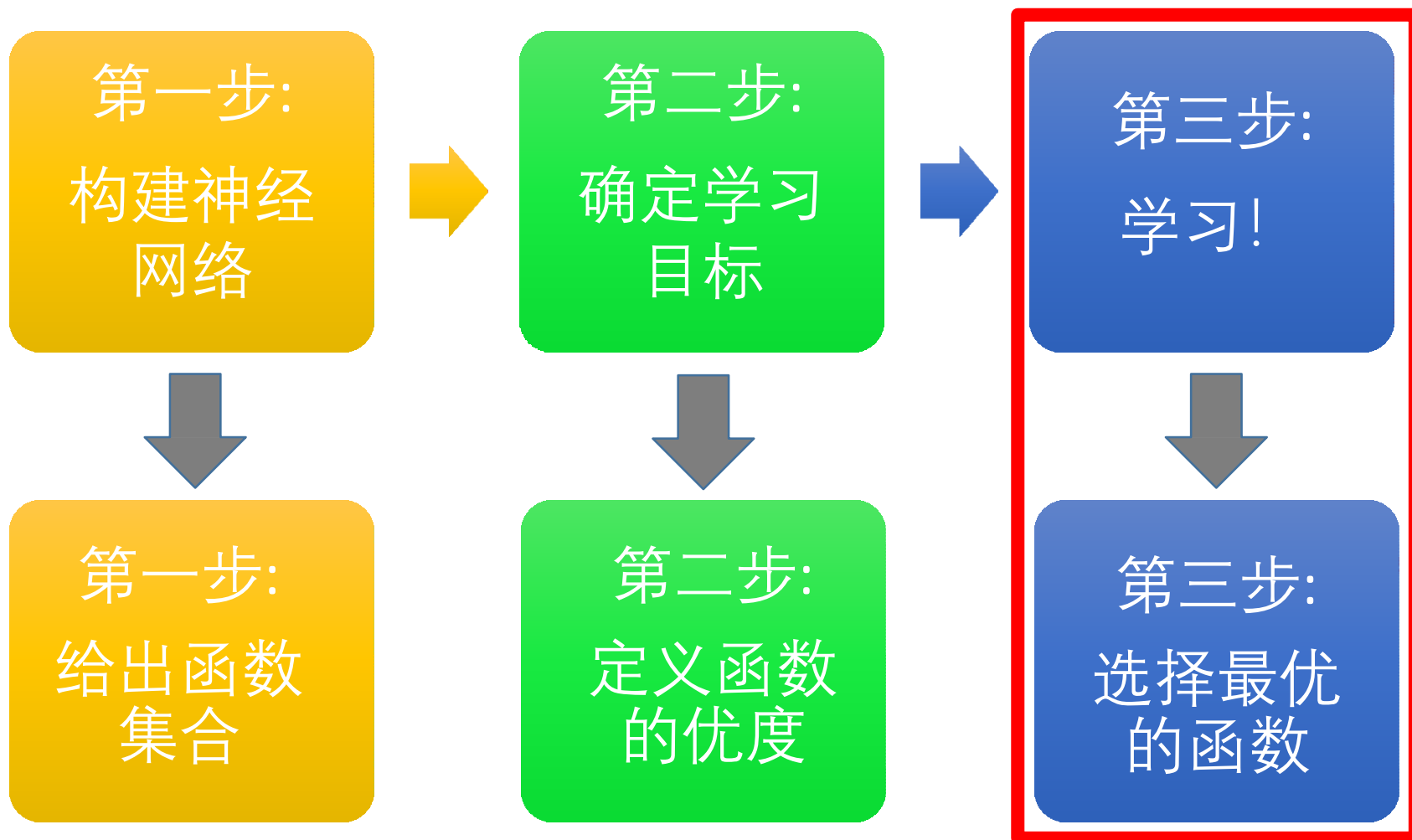
$$L = \sum_{r=1}^R l_r$$

尽可能小

在函数集中找到使整体损失L最小的一个函数

找到能最小化整体损失L的一组网络参数 θ^*

深度学习三部曲



如何选择最优的函数

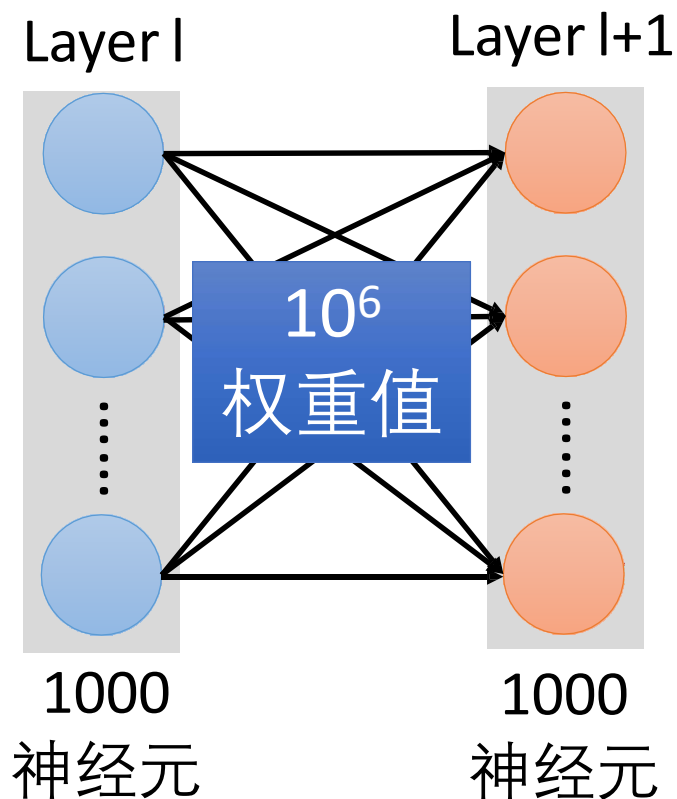
找到能最小化整体损失 L 的一组网络参数 θ^*

枚举所有可能的参数值

网络参数 θ
 $= \{w_1, w_2, \dots, w_n, b_2, b_3, \dots\}$

成百上千个参数

例如,
语音识别: 8层, 1000个 神经元/层

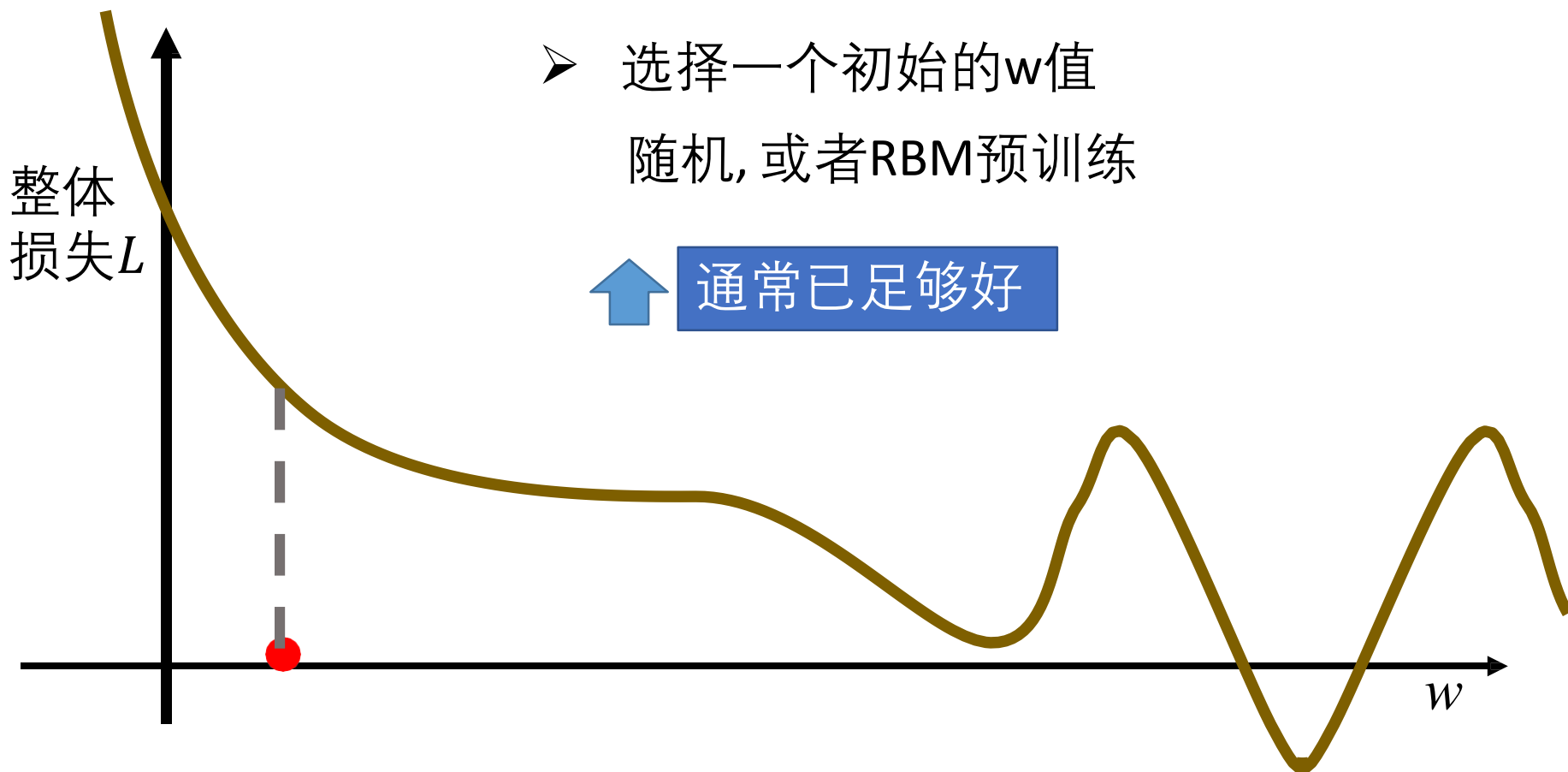


梯度下降法

网络参数

$$\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$$

找到能最小化整体损失 L 的一组网络参数 θ^*

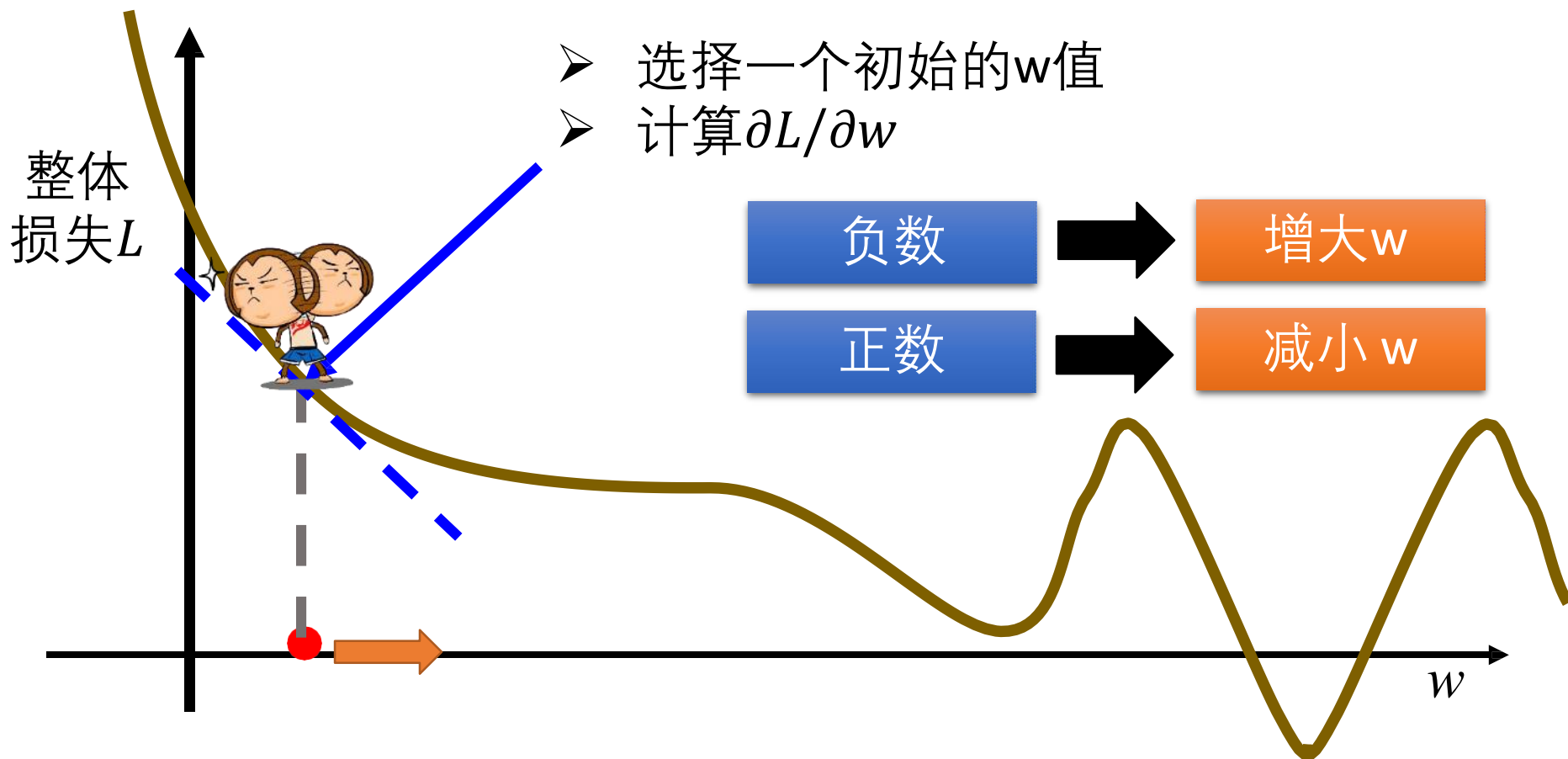


梯度下降法

网络参数 θ

$$= \{w_1, w_2, \dots, b_1, b_2, \dots\}$$

找到能最小化整体损失 L 的一组网络参数 θ^*

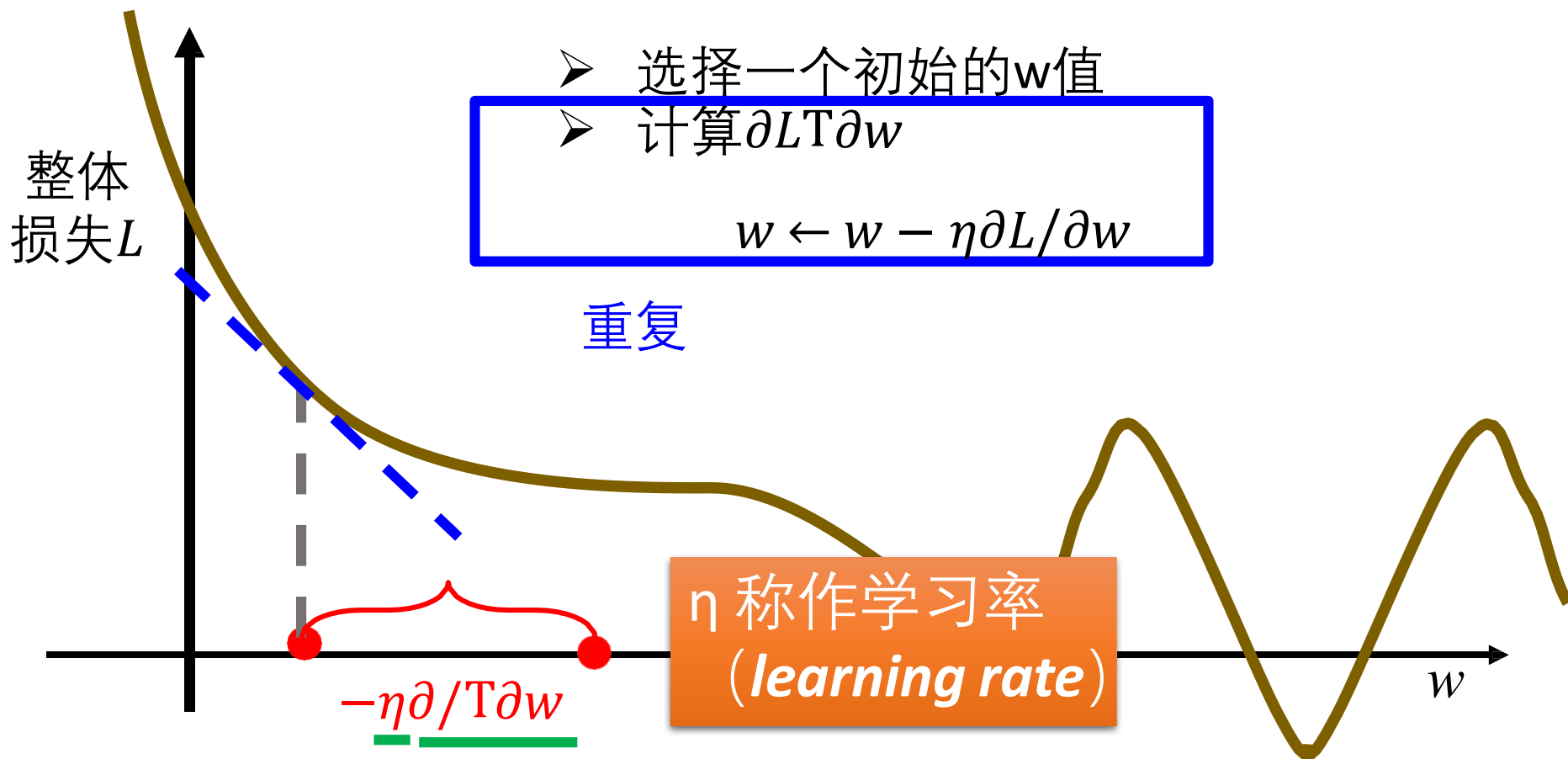


梯度下降法

网络参数 θ

$$= \{w_1, w_2, \dots, b_1, b_2, \dots\}$$

找到能最小化整体损失 L 的一组网络参数 θ^*

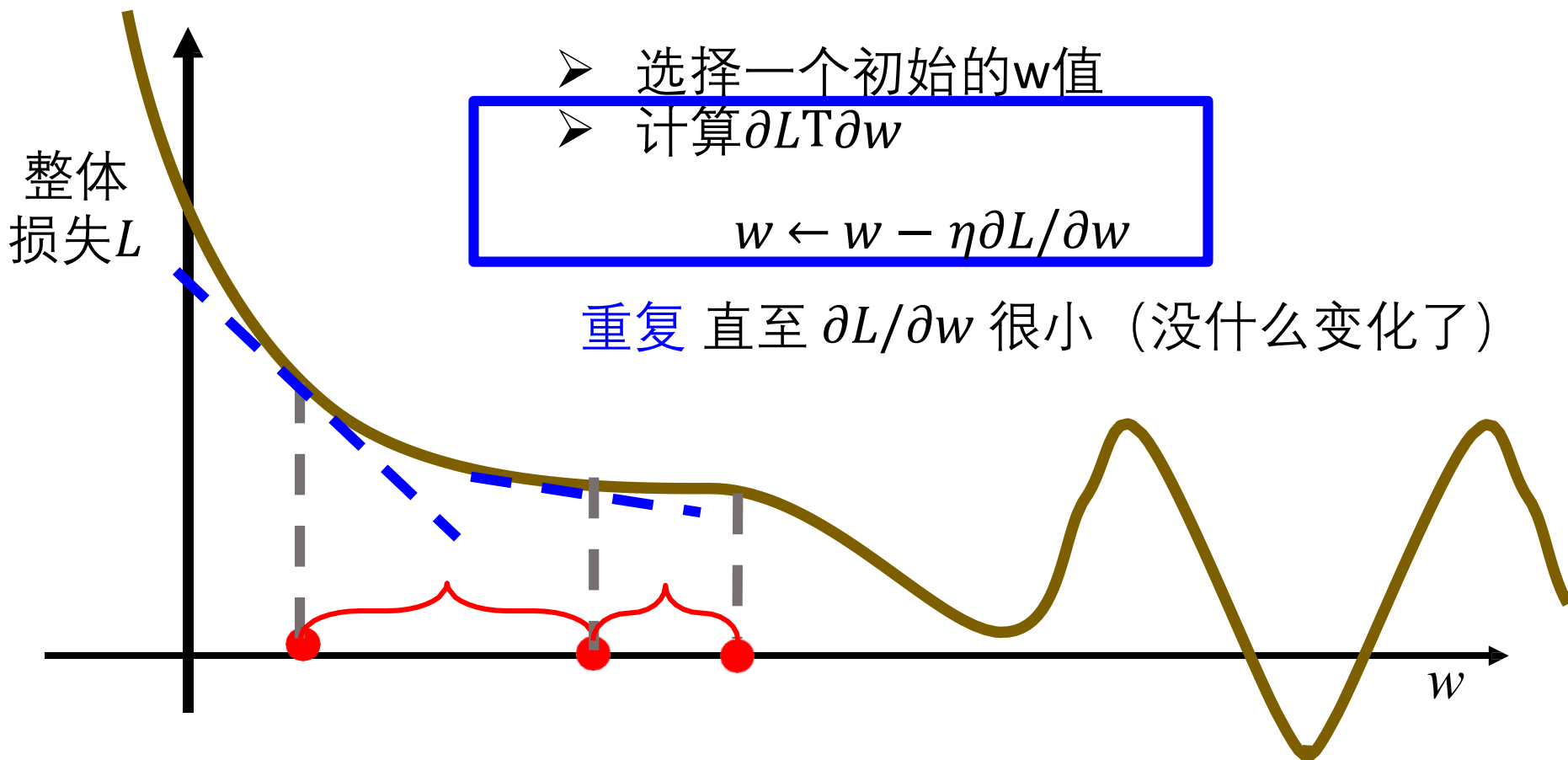


梯度下降法

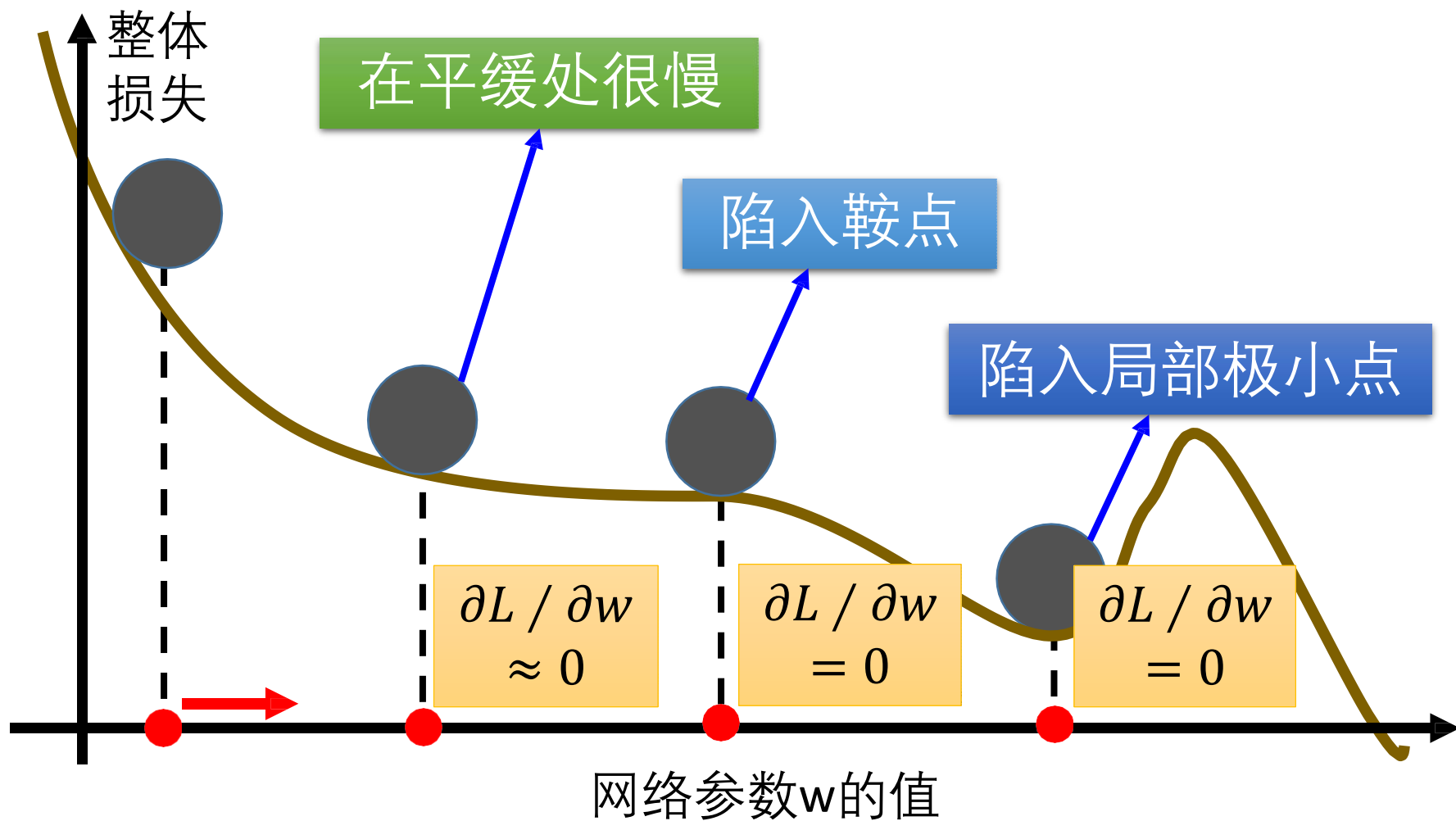
网络参数 θ

$$= \{w_1, w_2, \dots, b_1, b_2, \dots\}$$

找到能最小化整体损失 L 的一组网络参数 θ^*

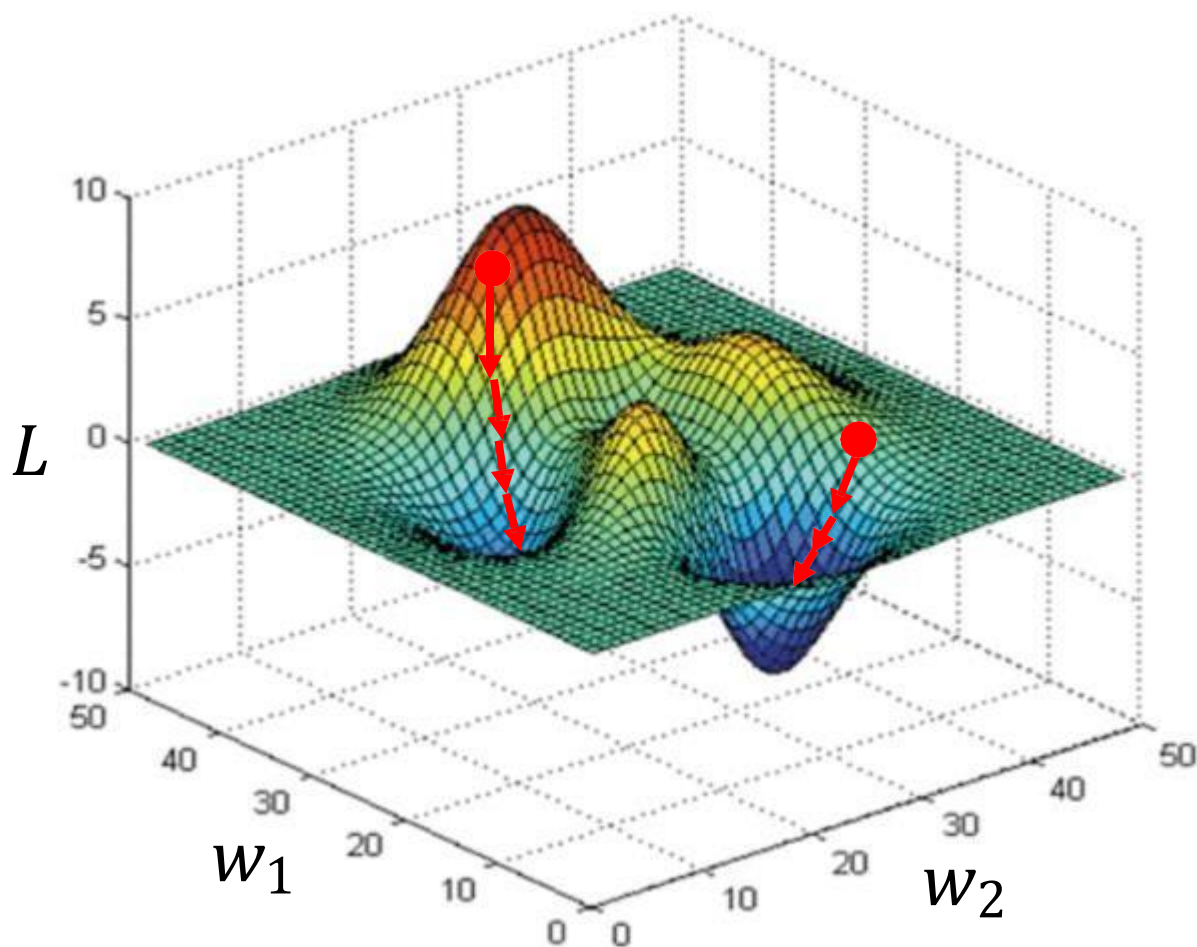


局部极小点

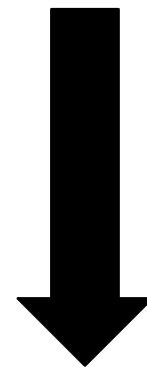


局部极小点

- 梯度下降法无法保证全局最小点



不同的起点



到达不同的极小点,
即不同的结果

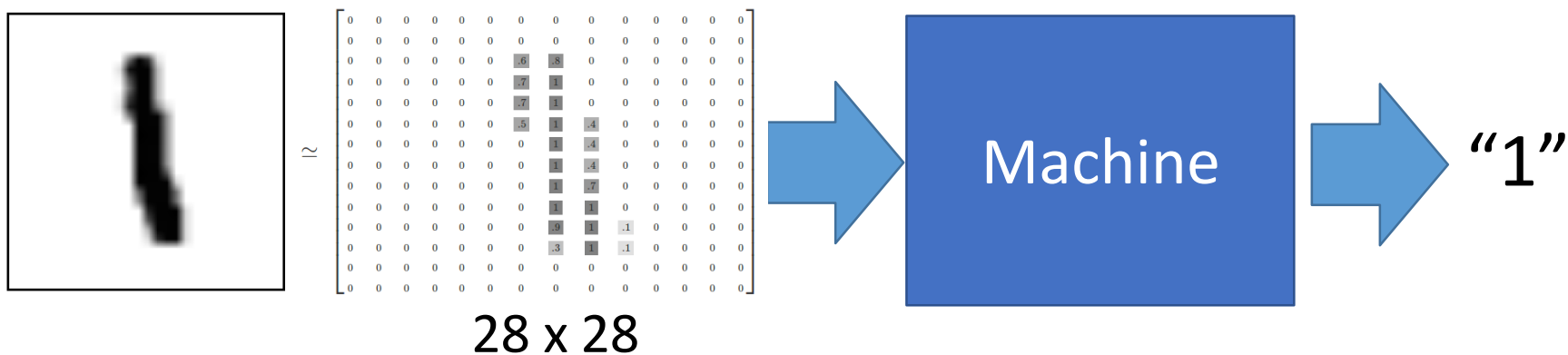
反向传播算法 (Backpropagation)

- 一个神经网络有成百上千万个参数，反向传播算法是一种高效计算所有梯度 $\partial L / \partial w$ 的算法。
- 许多已有的深度学习框架可以自动计算梯度。



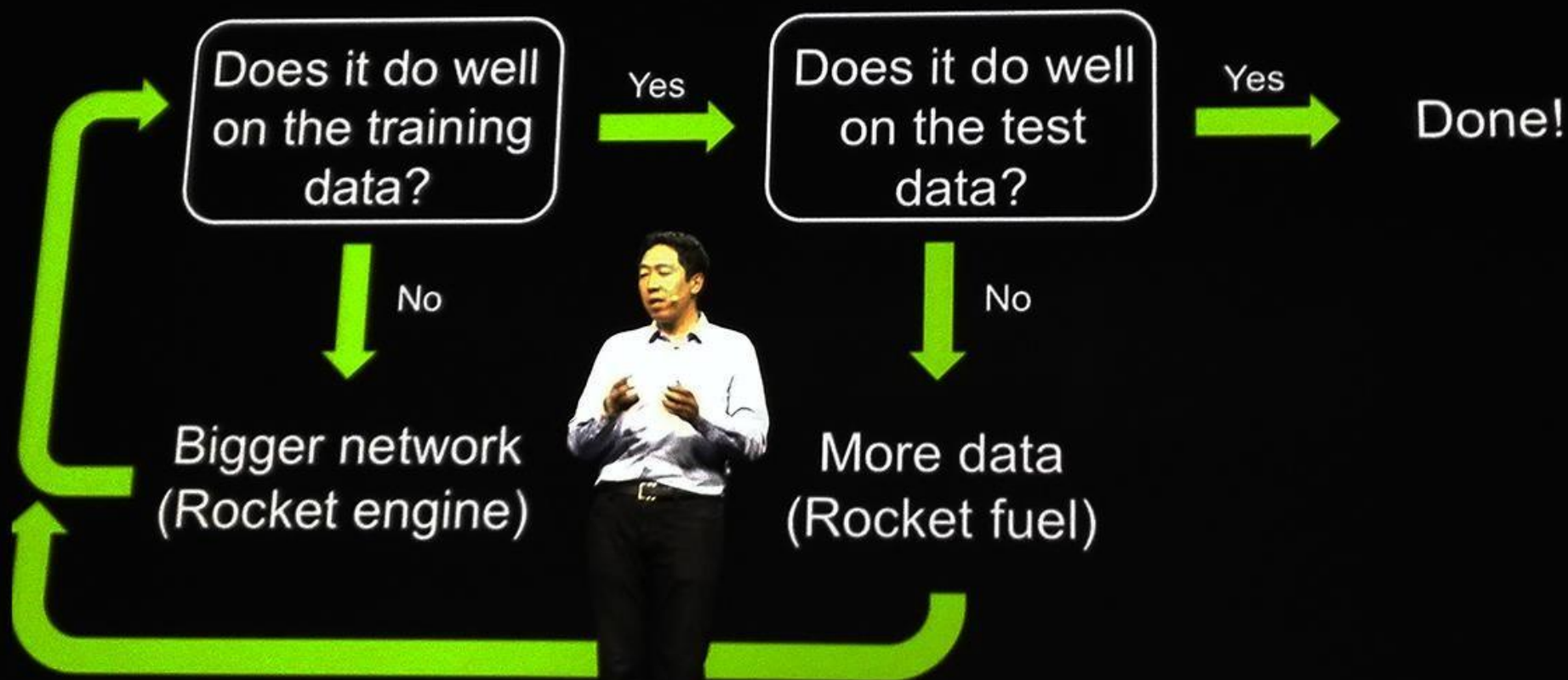
theano

案例应用：手写数字识别



MNIST 数据集: <http://yann.lecun.com/exdb/mnist/>

深度学习技巧

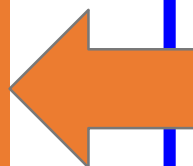


深度学习技巧



YES

NO

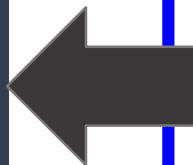


Good Results on
Testing Data?



YES

NO



Good Results on
Training Data?

Dropout

正则化 (Regularization)

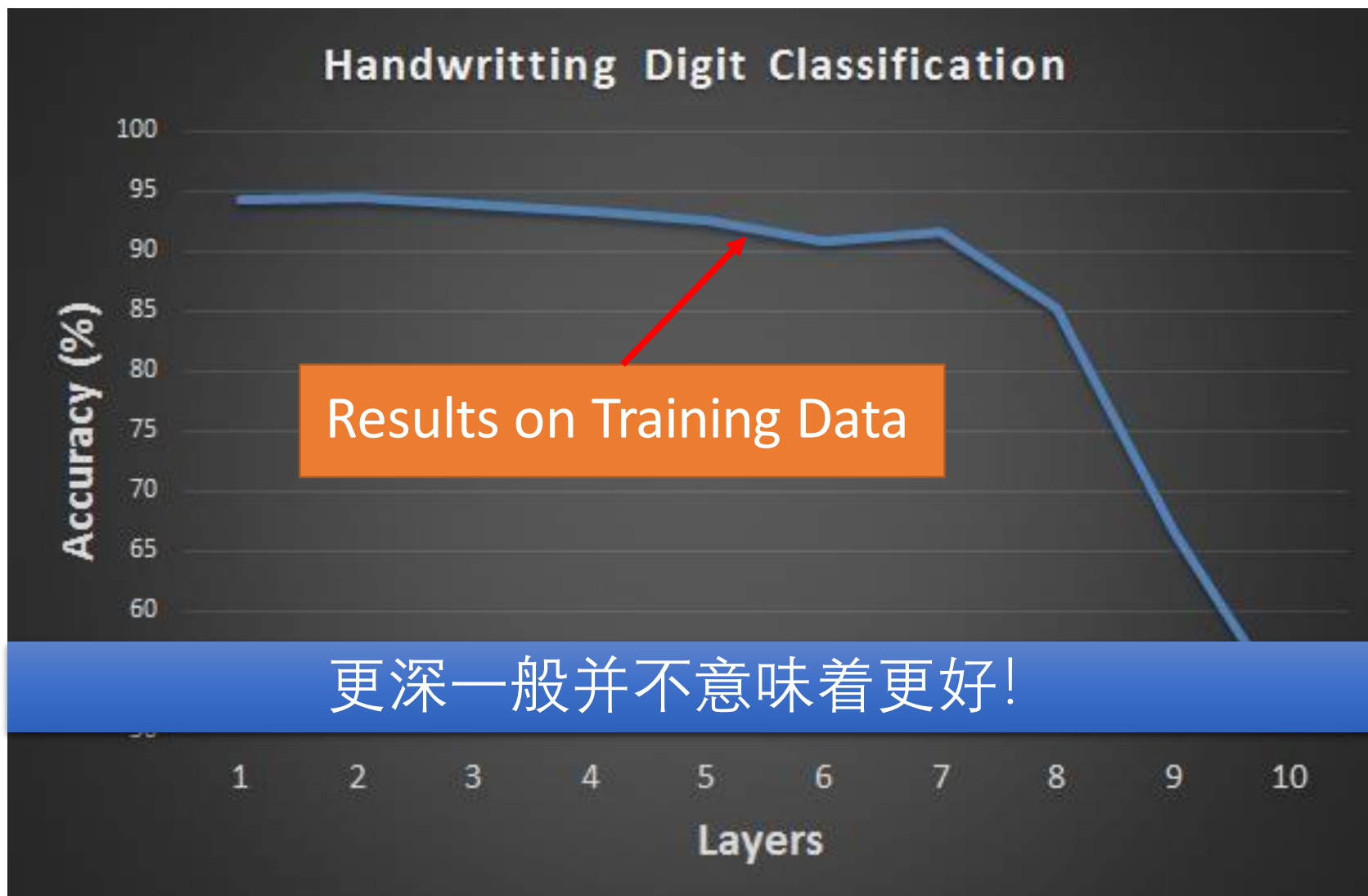
提前终止 (Early Stopping)

新的激活函数

合适的损失函数

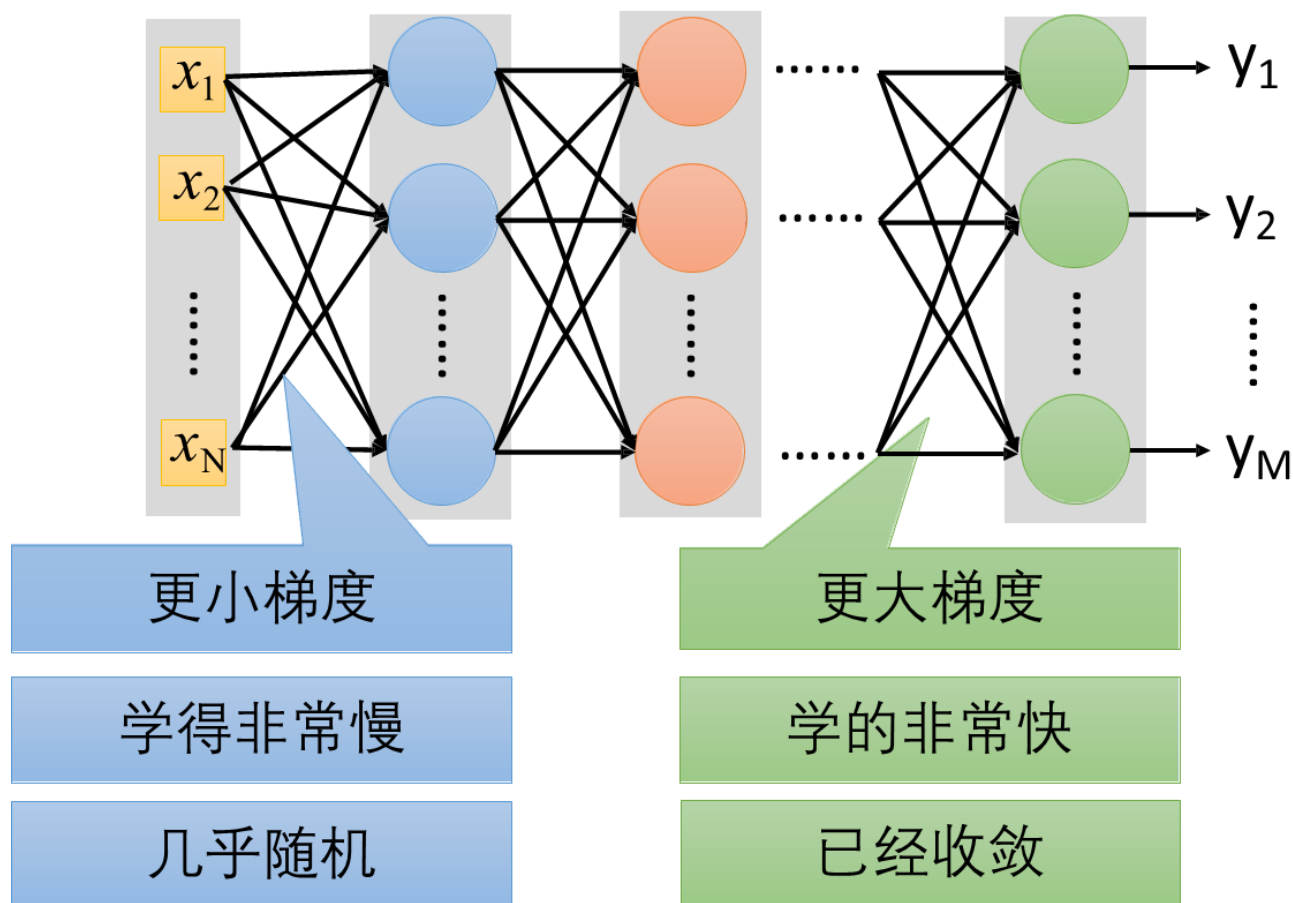
自适应的学习率、动量

Hard to get the power of Deep ...

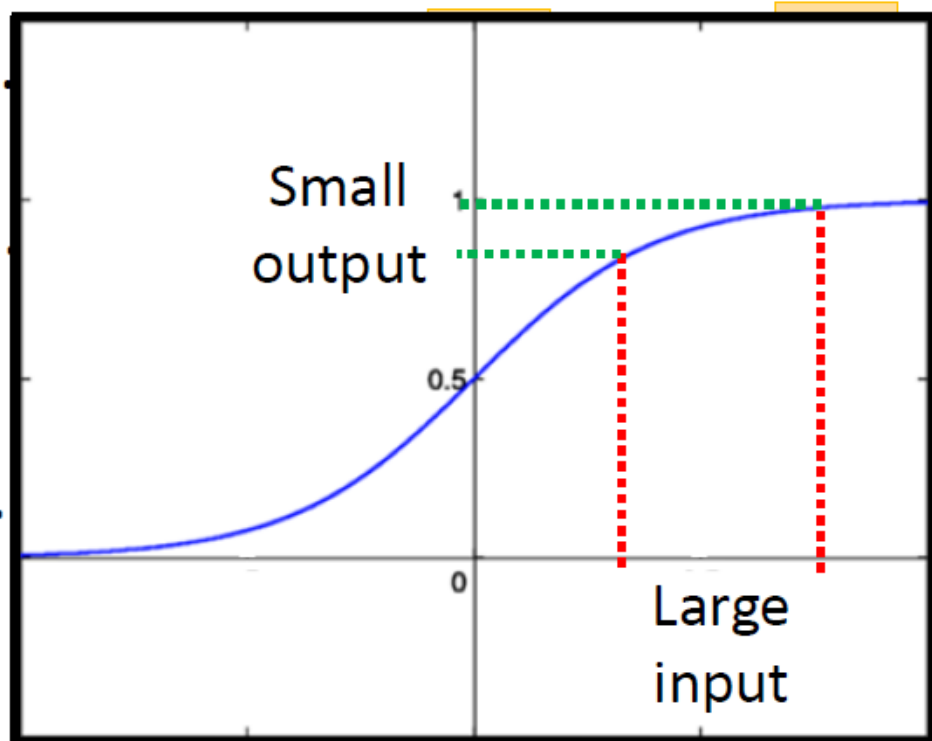
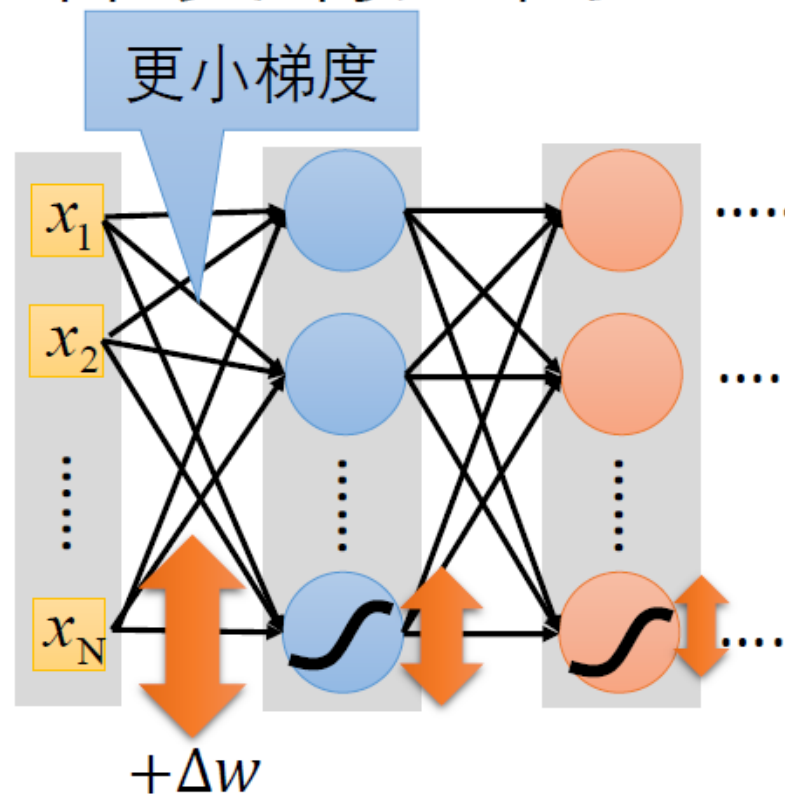


更深一般并不意味着更好!

梯度消失问题



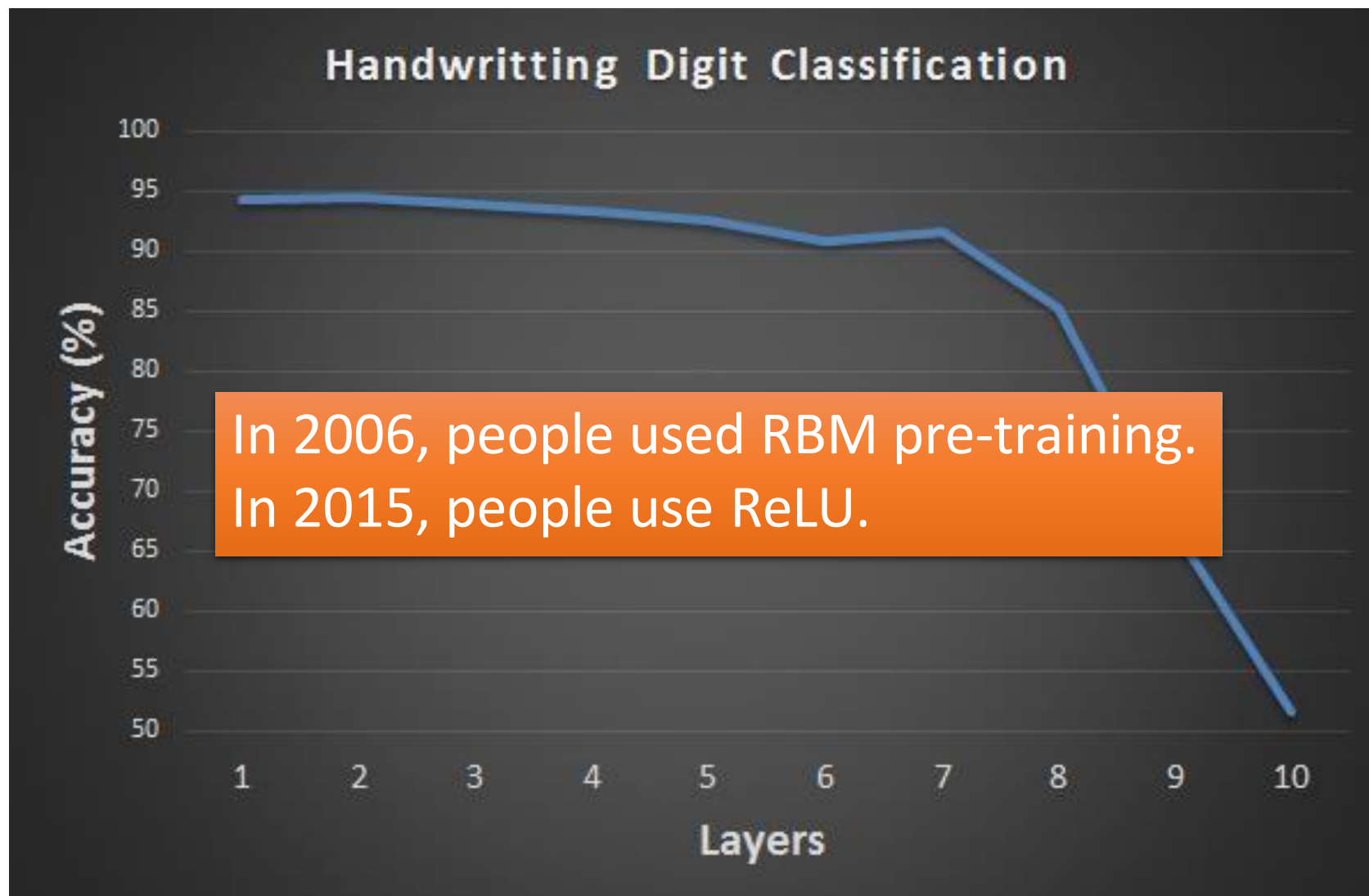
梯度消失问题



计算导数的直观方法 ...

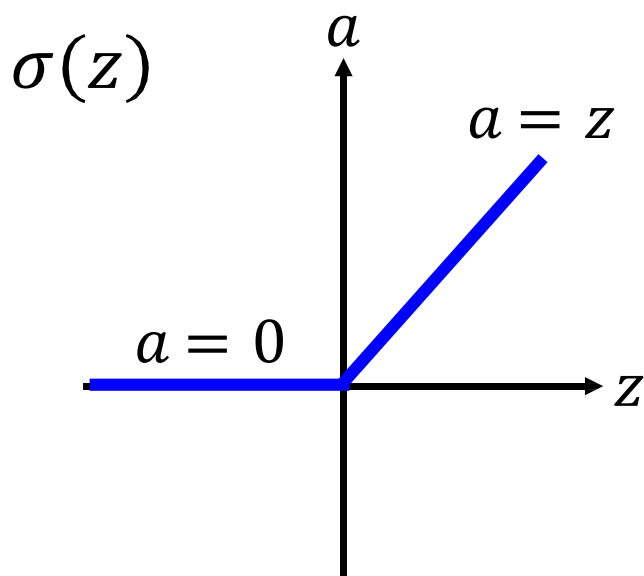
$$\frac{\partial l}{\partial w} = ? \quad \frac{\Delta l}{\Delta w}$$

Hard to get the power of Deep ...



新的激活函数

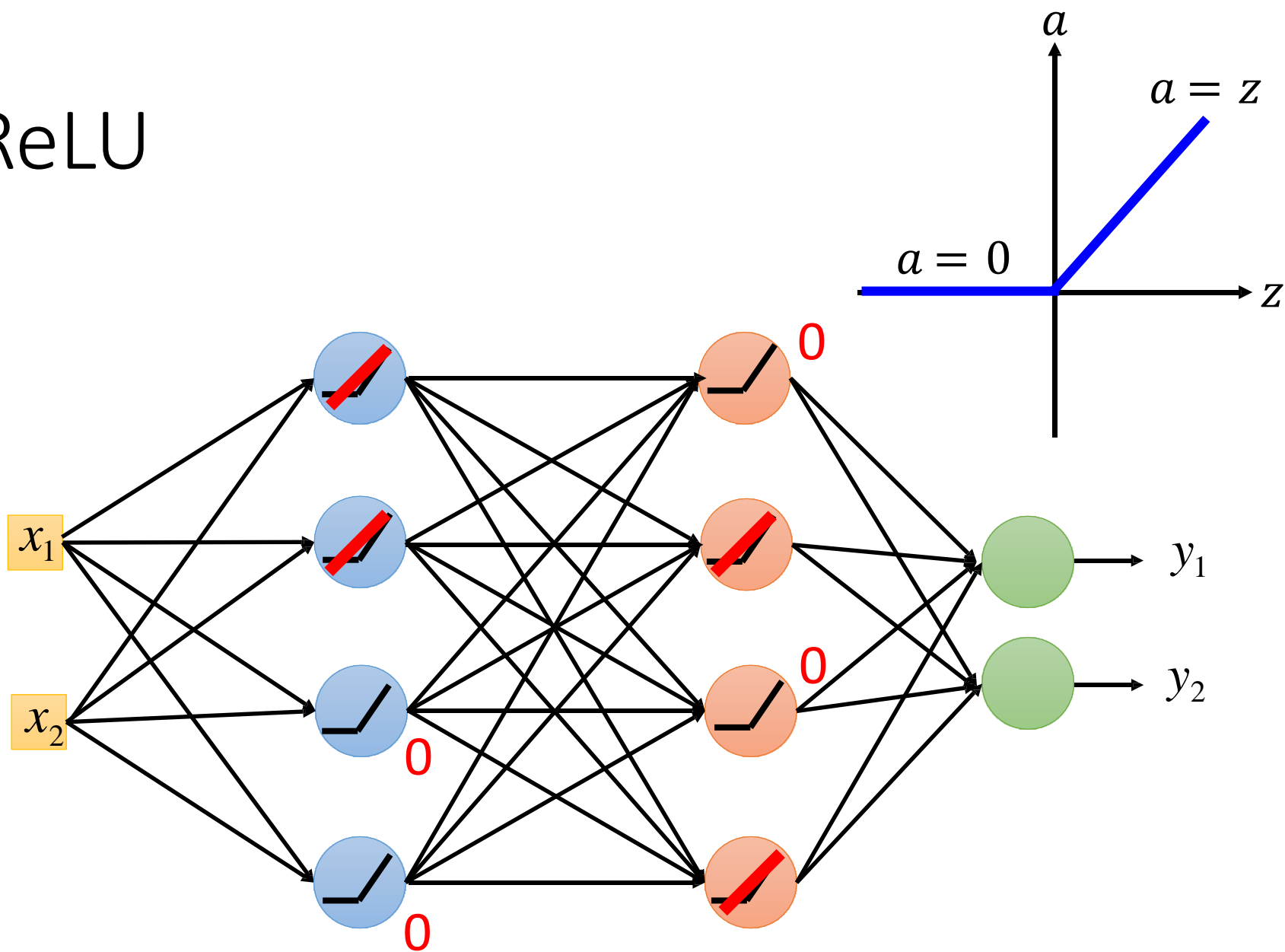
- Rectified Linear Unit (ReLU)



- 原因:

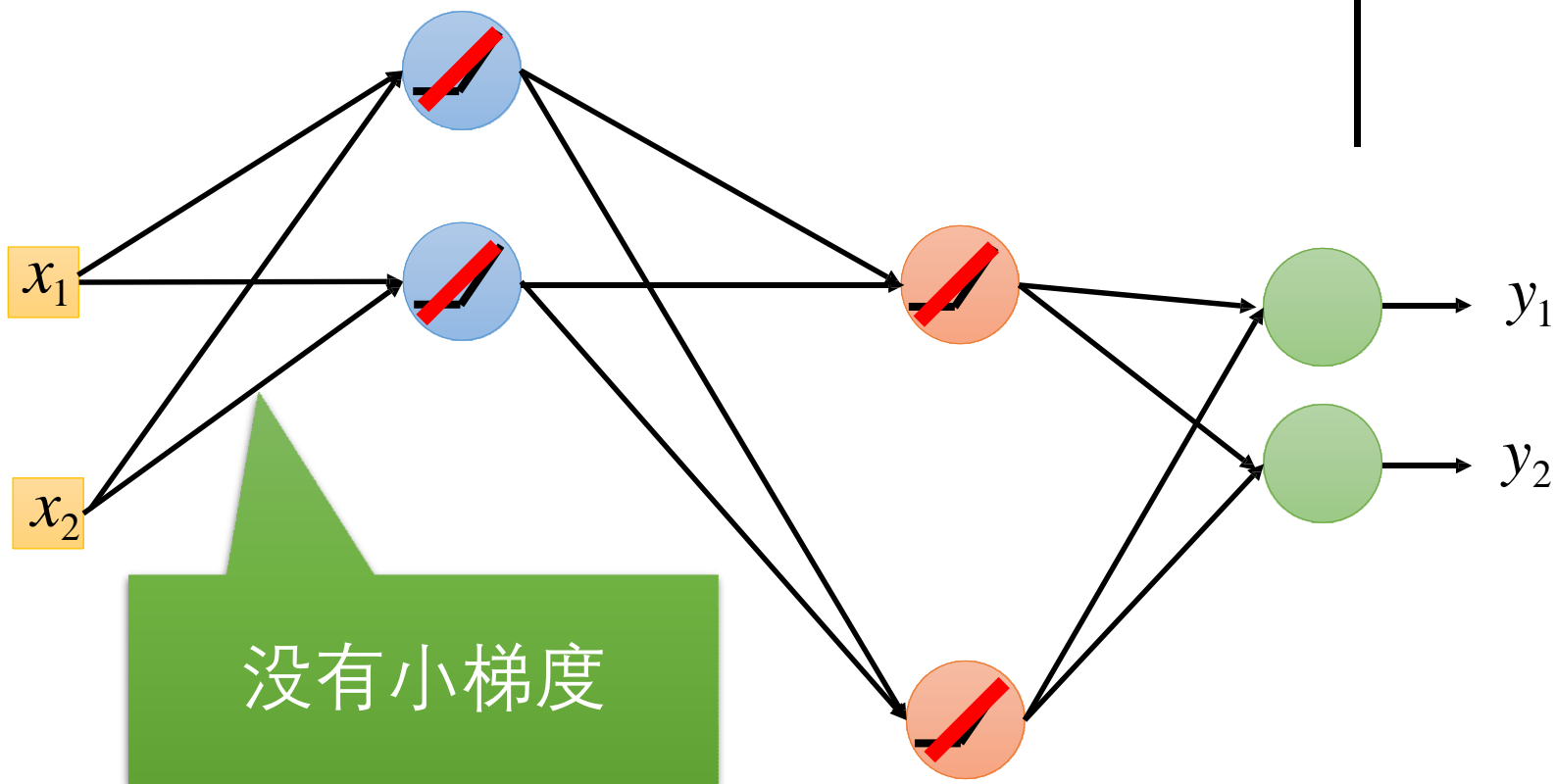
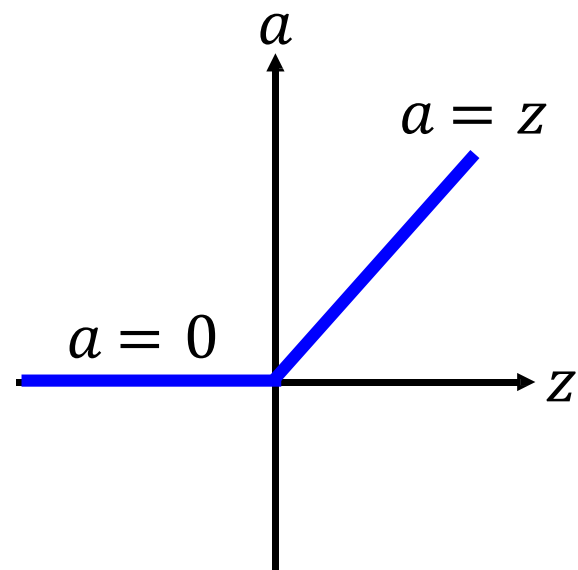
1. 快速计算
2. 符合生物神经元特征
3. Infinite sigmoid with different biases
4. 解决梯度消失问题

ReLU



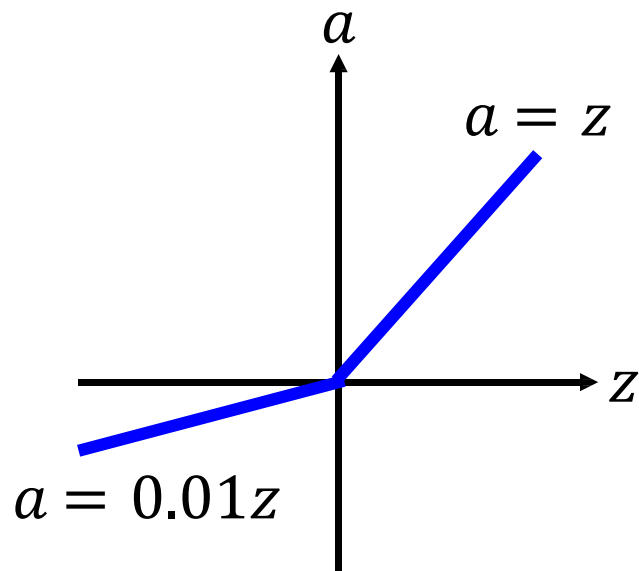
ReLU

更瘦的线性网络

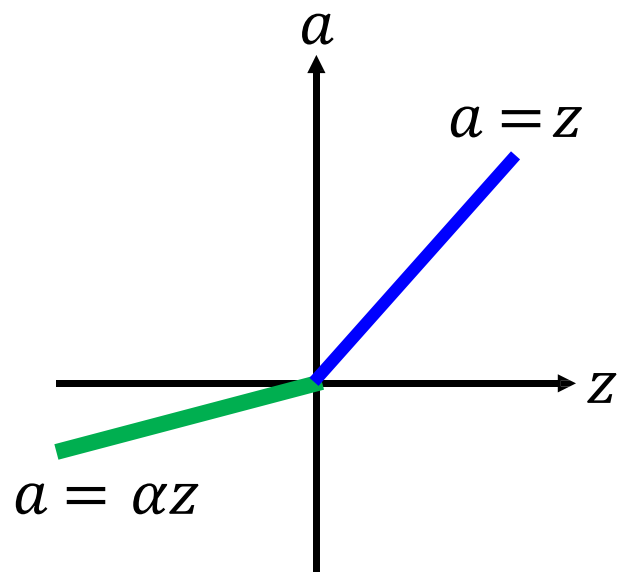


ReLU – 变体

Leaky ReLU

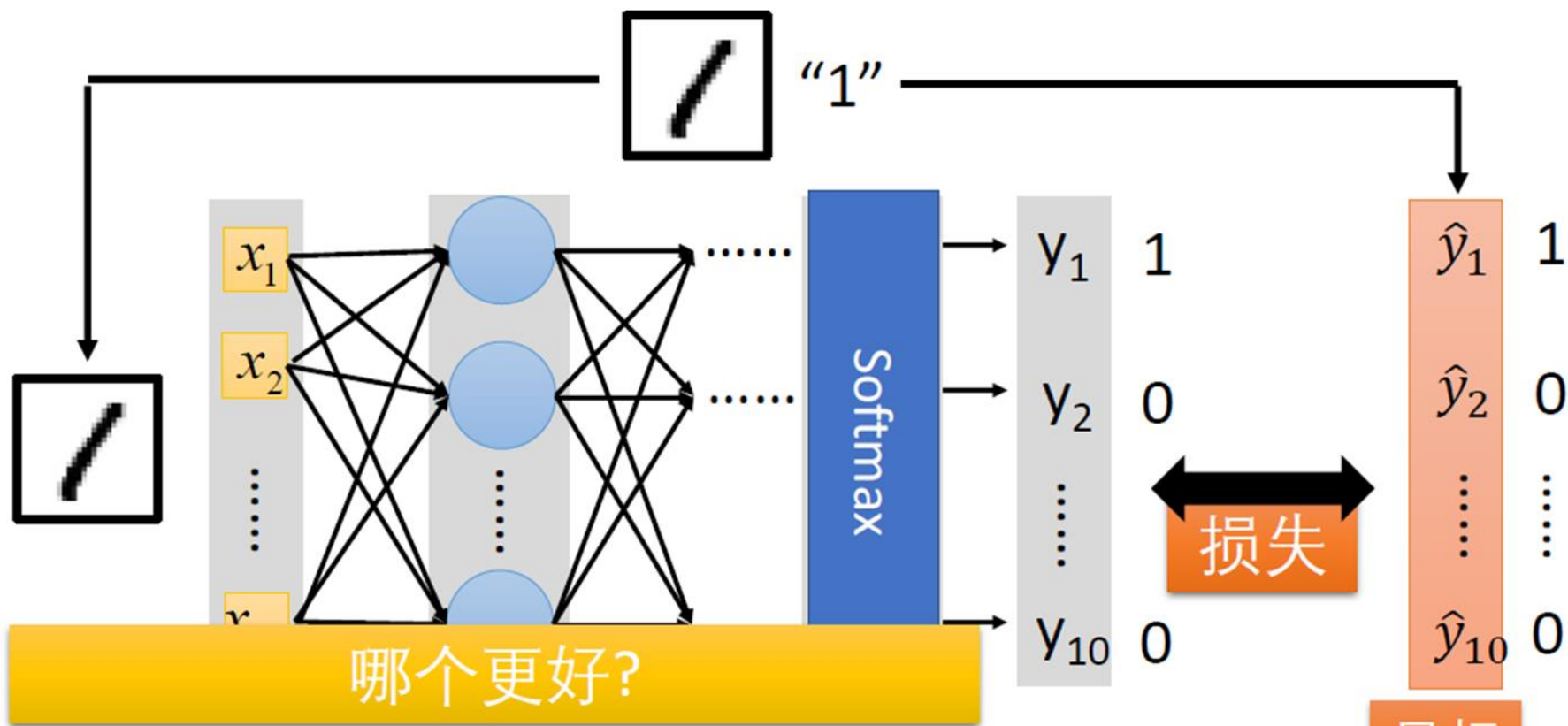


Parametric ReLU



α 也用梯度下降法学习

合适的损失函数



均方误差

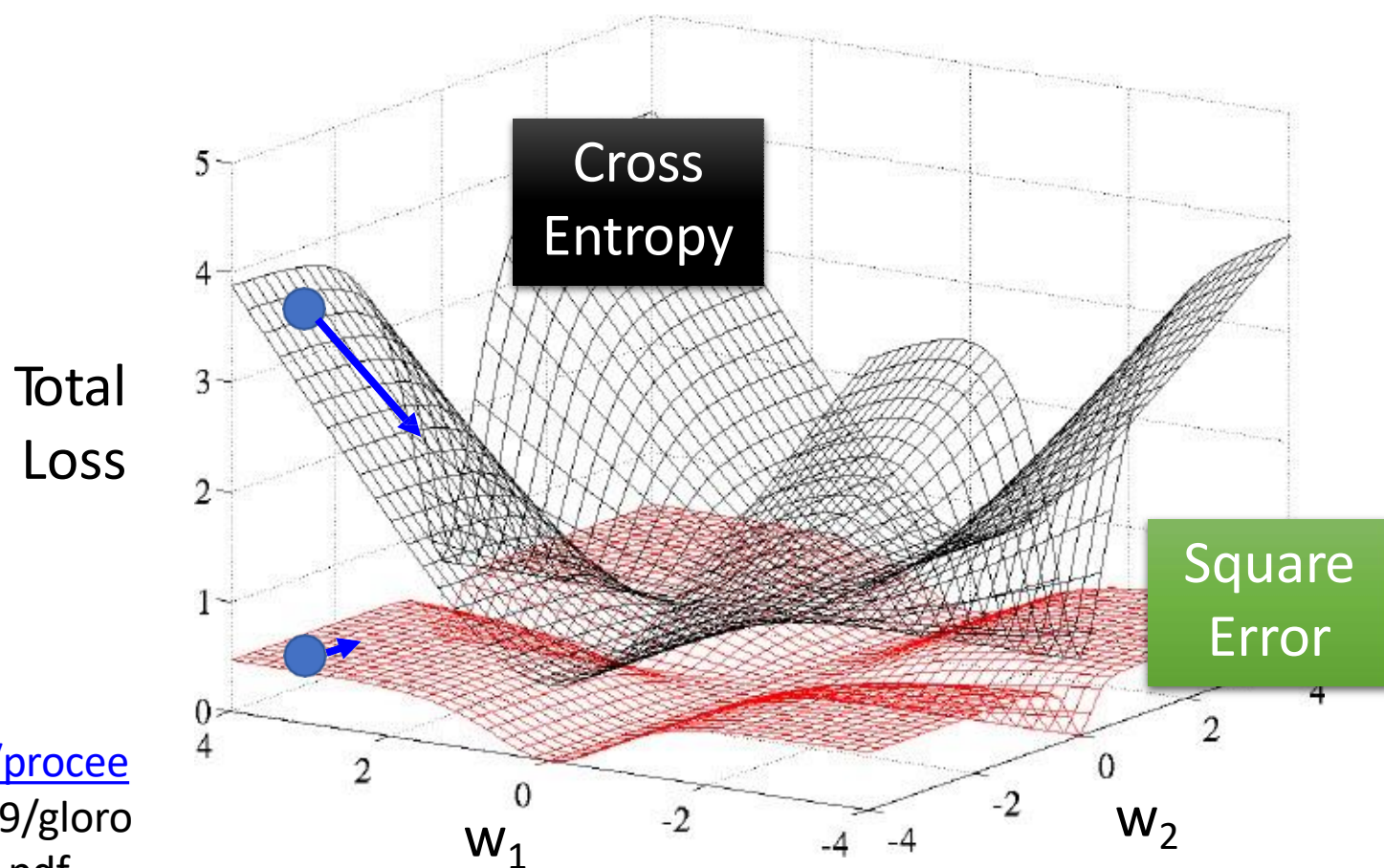
$$\sum_{i=1}^{10} (y_i - \hat{y}_i)^2 = 0$$

交叉熵

$$-\sum_{i=1}^{10} \hat{y}_i \ln y_i = 0$$

选择合适的损失函数

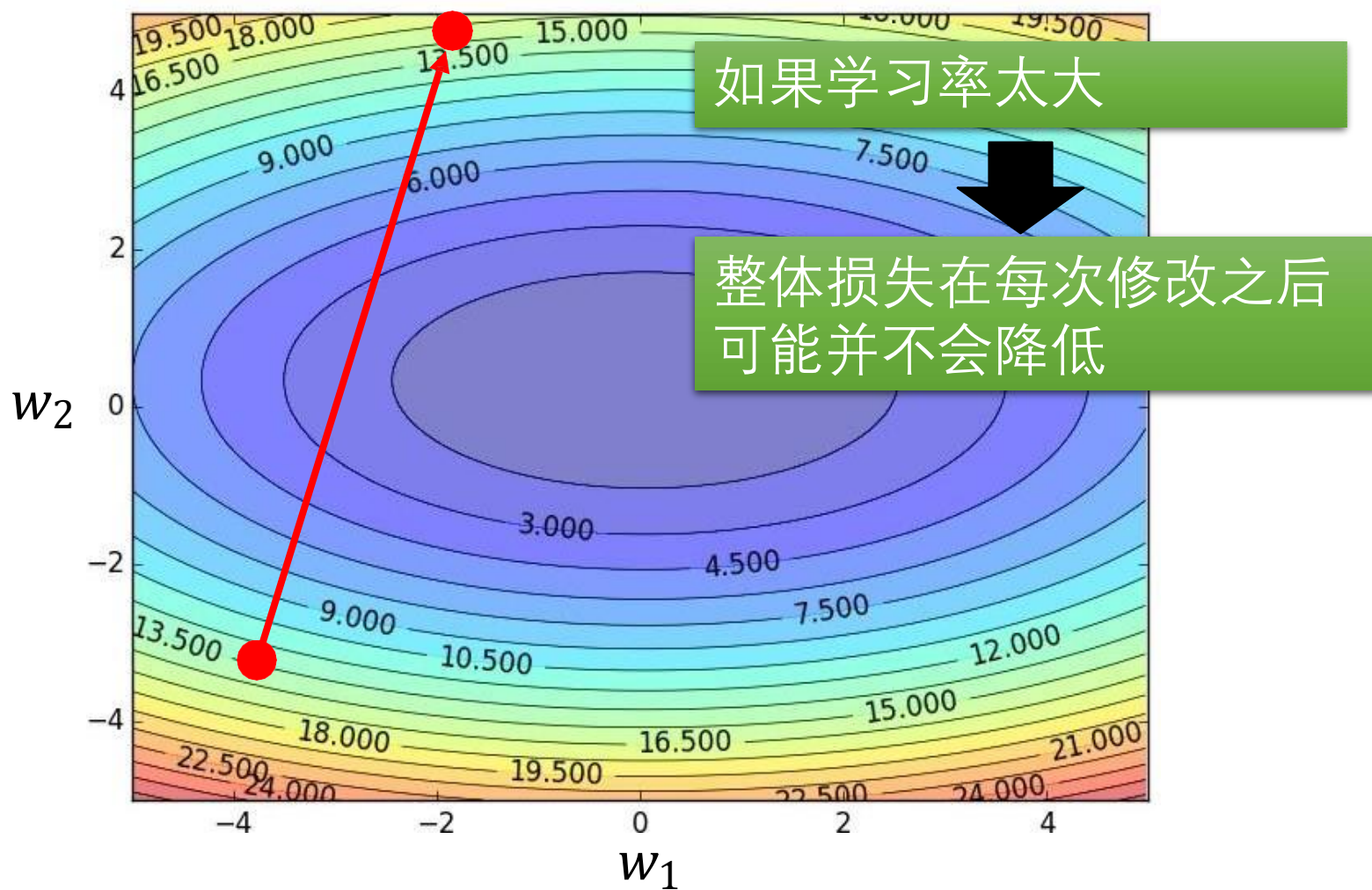
当使用softmax输出层时，使用交叉熵损失函数



<http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf>

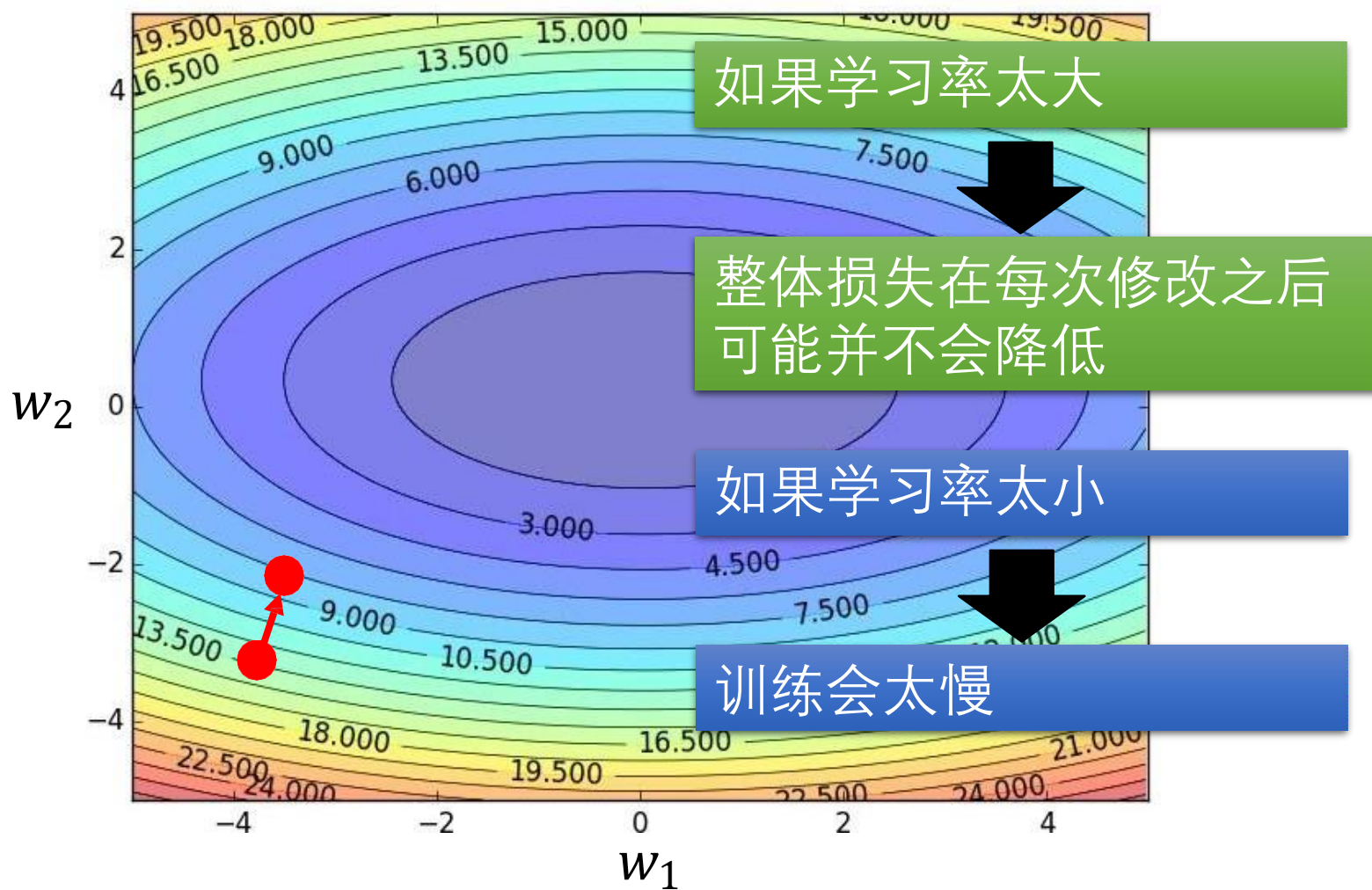
学习率

小心设置学习率 η



学习率

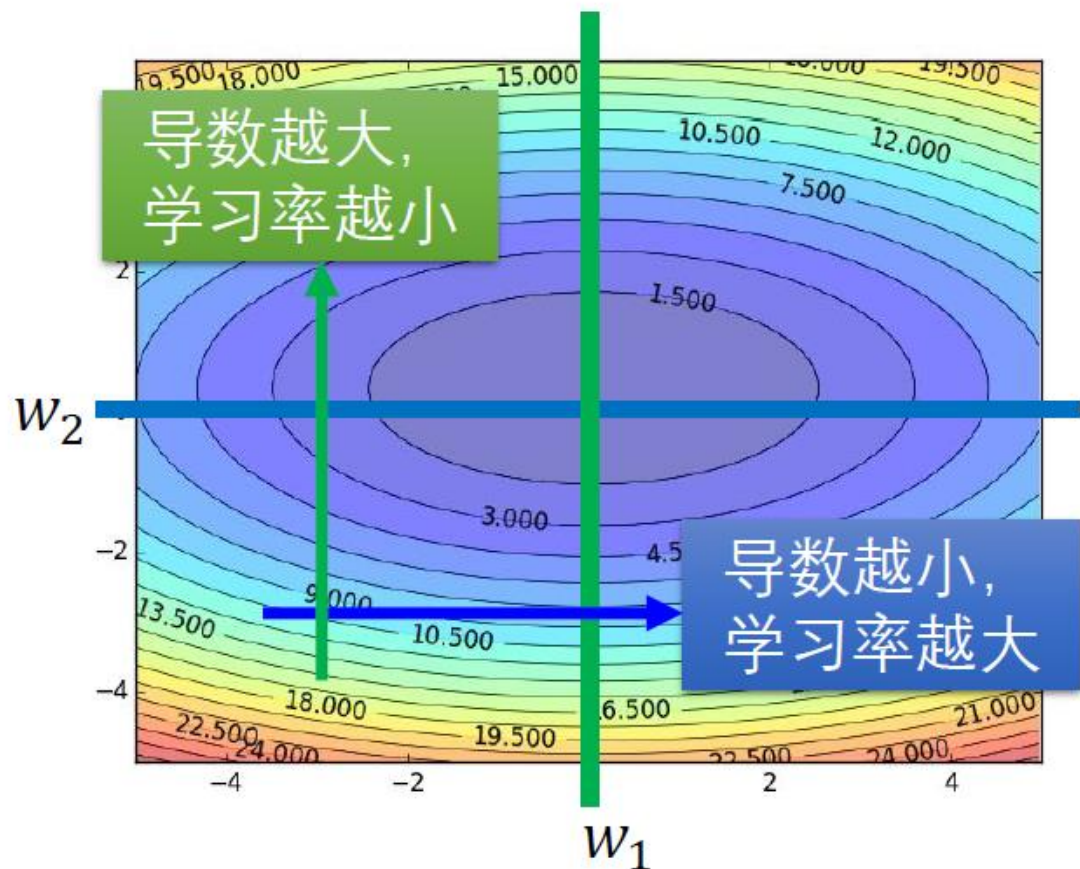
小心设置学习率 η



学习率

- 简单常用的想法：每几个epoch就将学习率减小一定幅度
 - 一开始，我们离目标很远，可以使用较大的学习率
 - 训练几个epoch后，我们接近目标了，就减小学习率
如： $\eta^t = \eta / \sqrt{t+1}$
- 学习率不可能是one-size-fits-all
 - 不同的参数应该有不同的学习率

自适应的 学习率

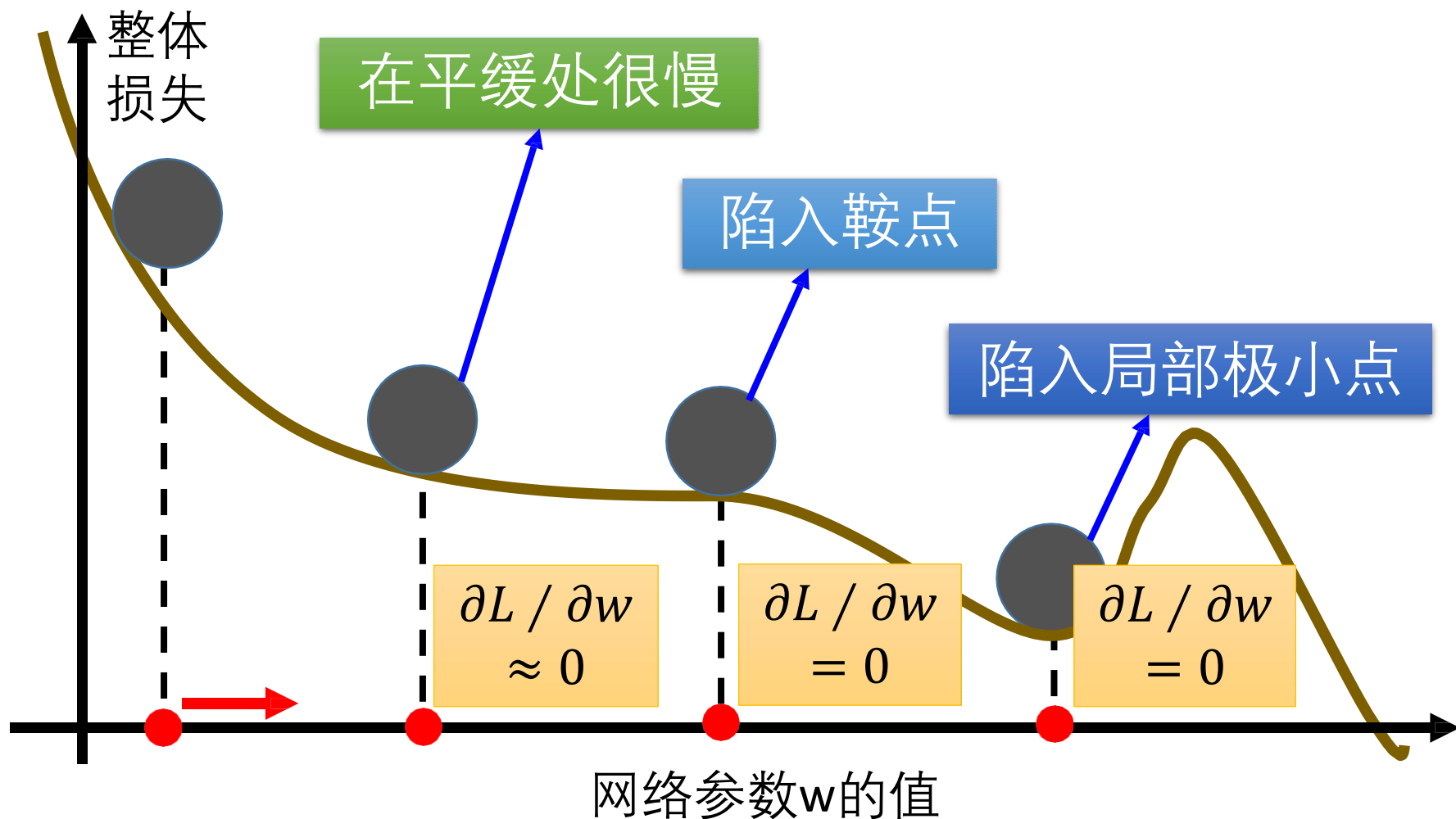


Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}} g^t$$

Adam: RMSProp (改进的Adagrad) + 动量

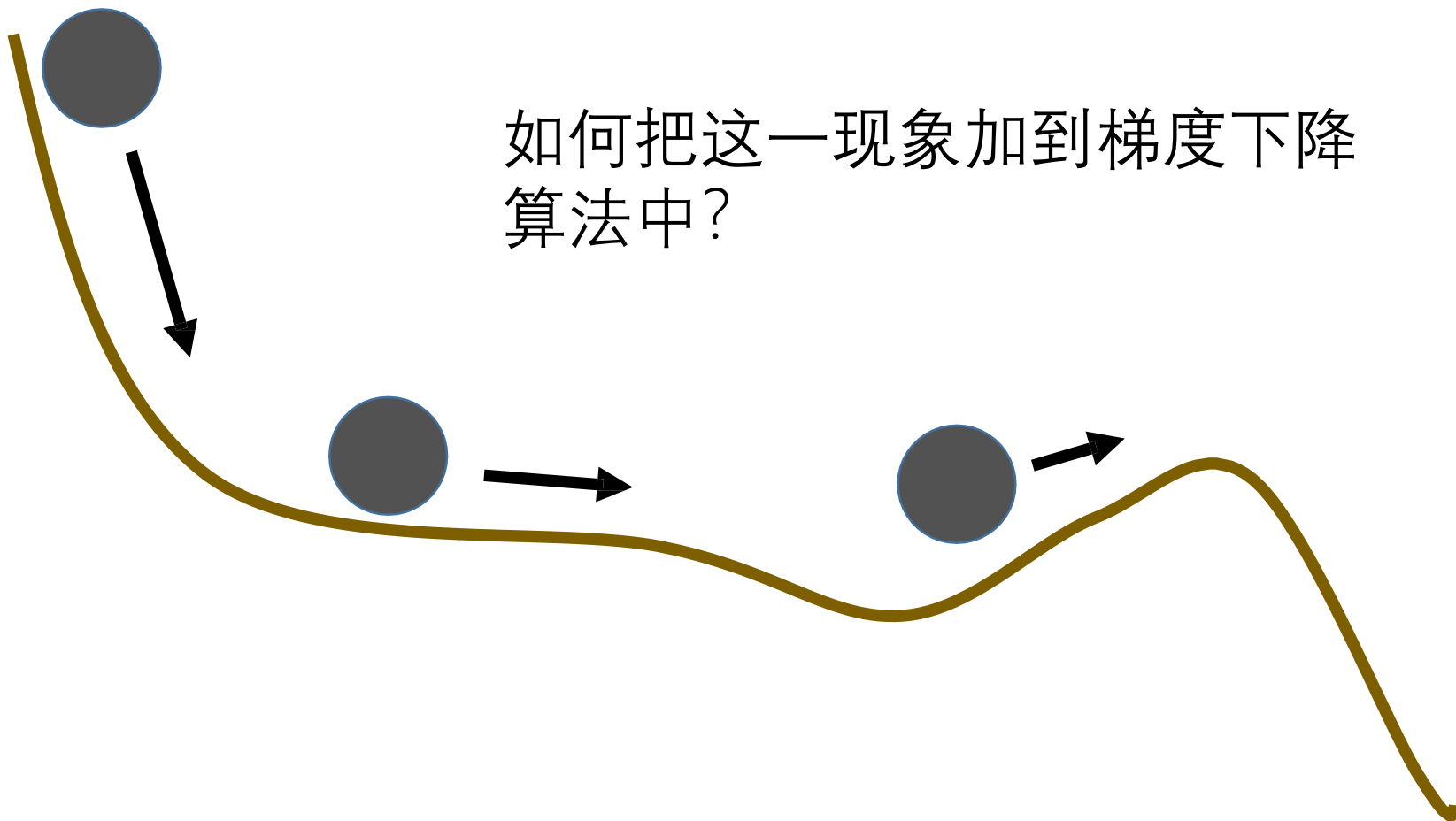
难以发现最优的网络参数



在物理世界中.....

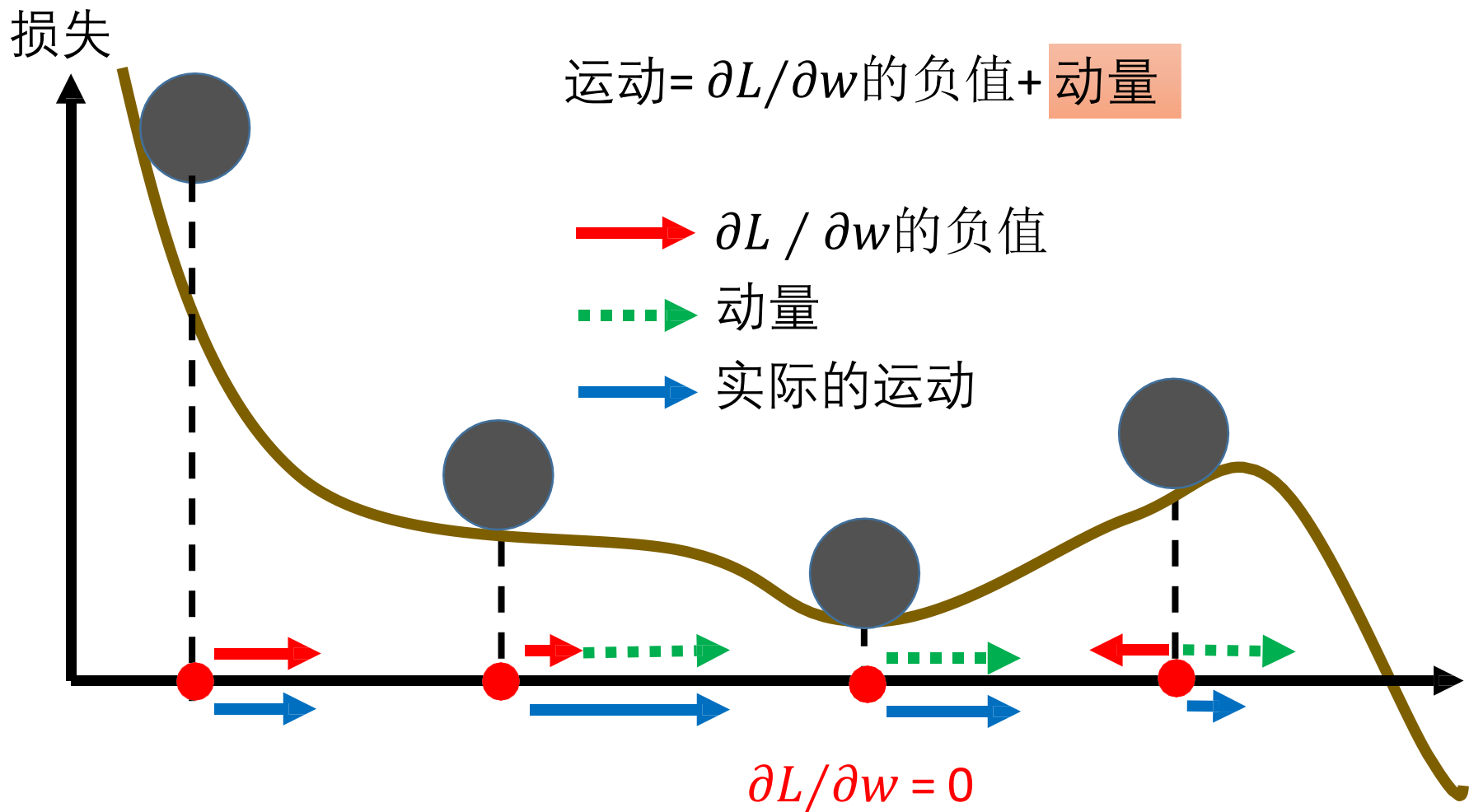
- 动量

如何把这一现象加到梯度下降算法中？



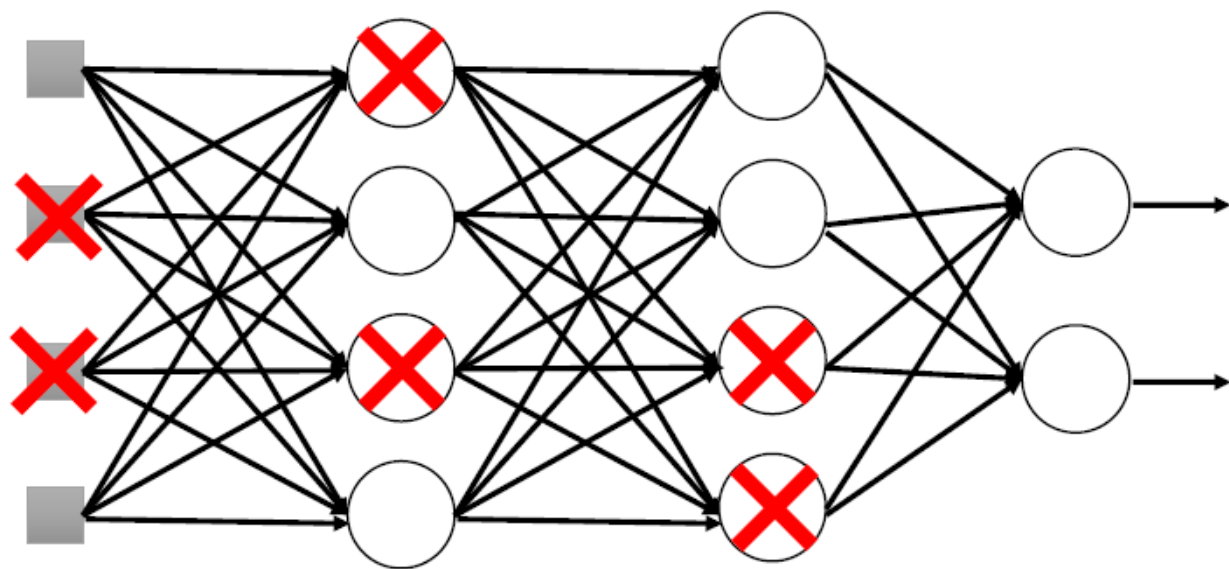
动量 (Momentum)

仍然不能保证到达全局最优，
但增加了一些希望.....



Dropout

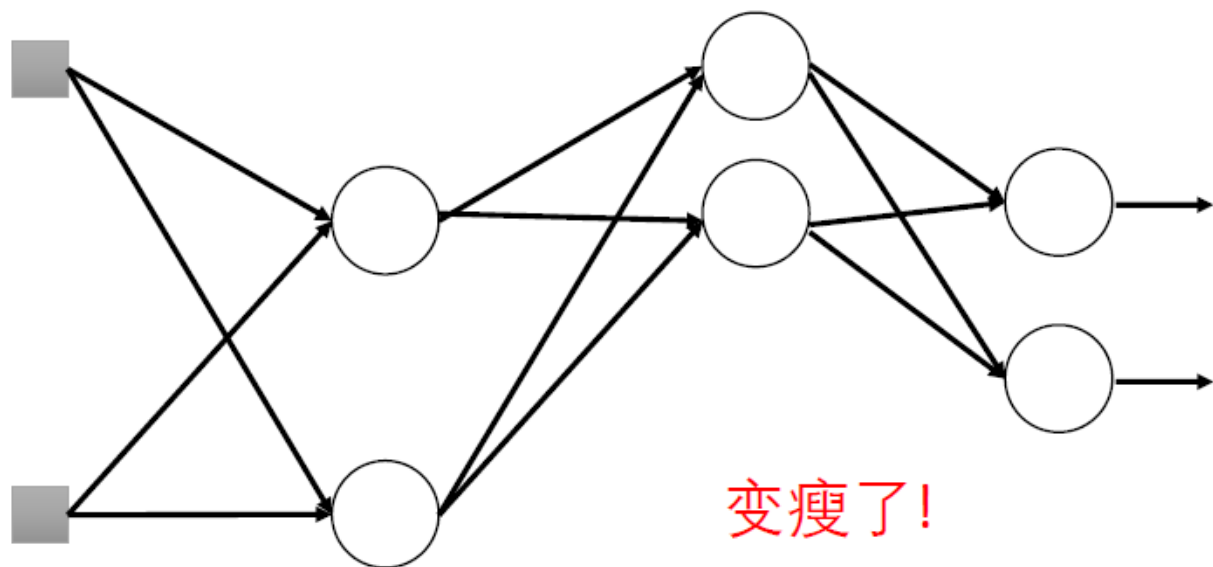
训练:



- 每次修改参数值前
 - 每个神经元有 $p\%$ 的概率退出

Dropout

训练:



➤ 每次修改参数值前

- 每个神经元有 $p\%$ 的概率退出



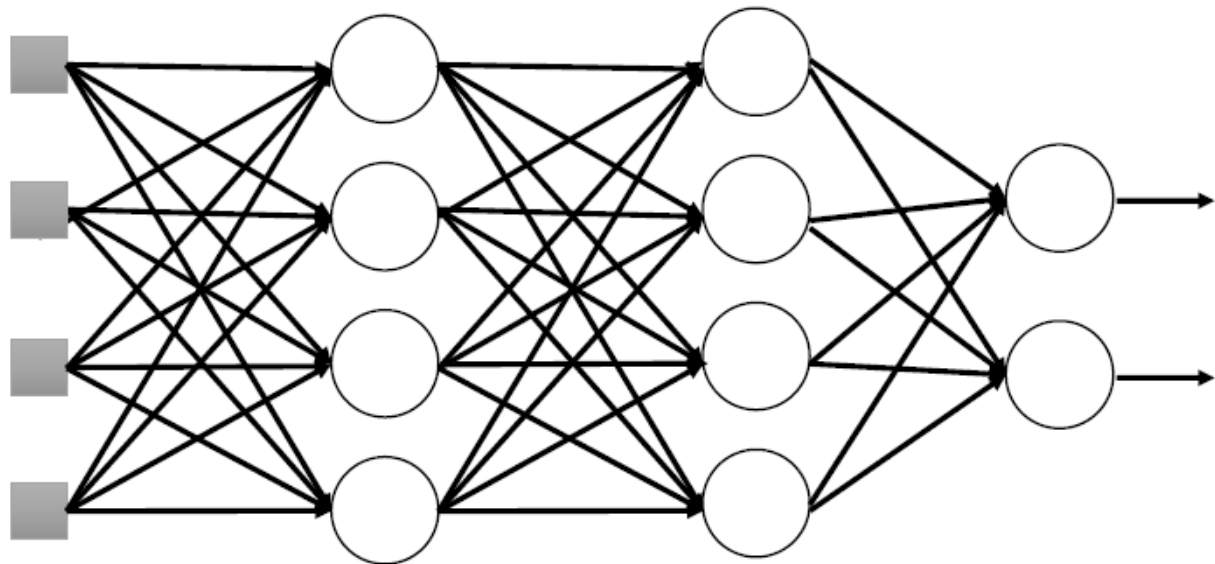
网络的结构变了

- 训练新的网络

对每个mini-batch, 我们重新随机选取退出的神经元

Dropout

测试:



➤ 没有dropout

- 如果训练时的退出概率是 $p\%$, 则所有的权重都乘以 $(1-p\%)$
- 假设退出概率是50%,
如果一个权重 w 训练得到的值是1, 则测试时将 w 的值设为0.5

Dropout – 直观理解

Training

Dropout (腳上綁重物)



Testing

No dropout
(拿下重物后就变很强)

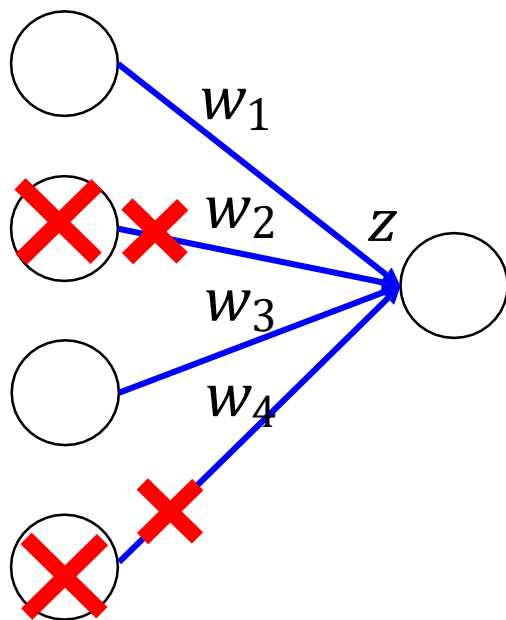


Dropout – 直观理解

- 为什么测试时网络的权重应该乘以 $(1-p)\%$ (dropout rate)?

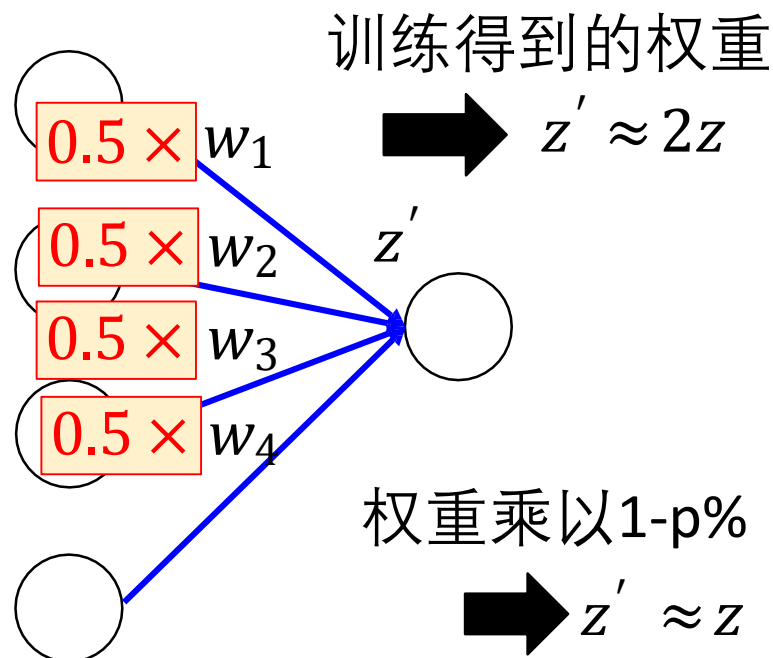
Training of Dropout

假设退出率是50%



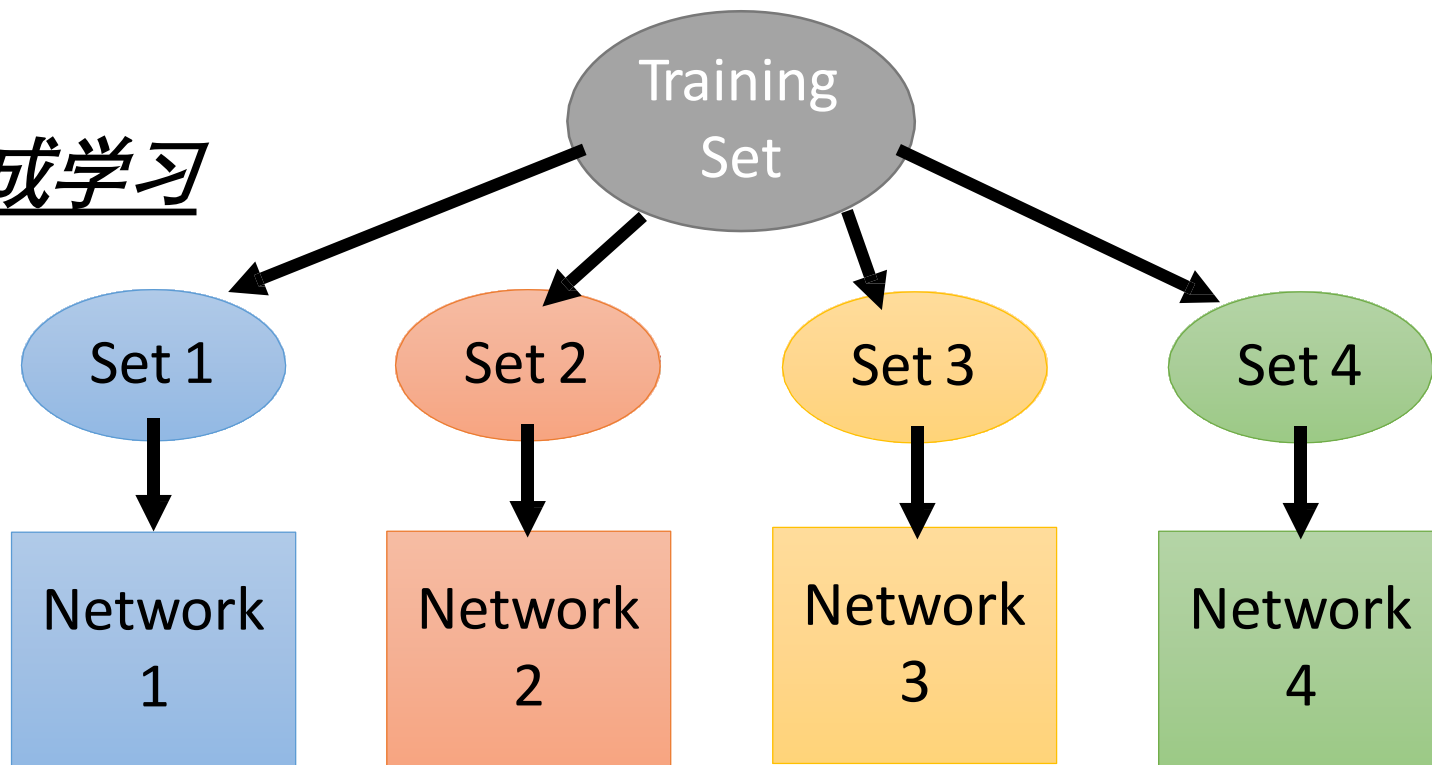
Testing of Dropout

没有退出



Dropout是一种集成学习

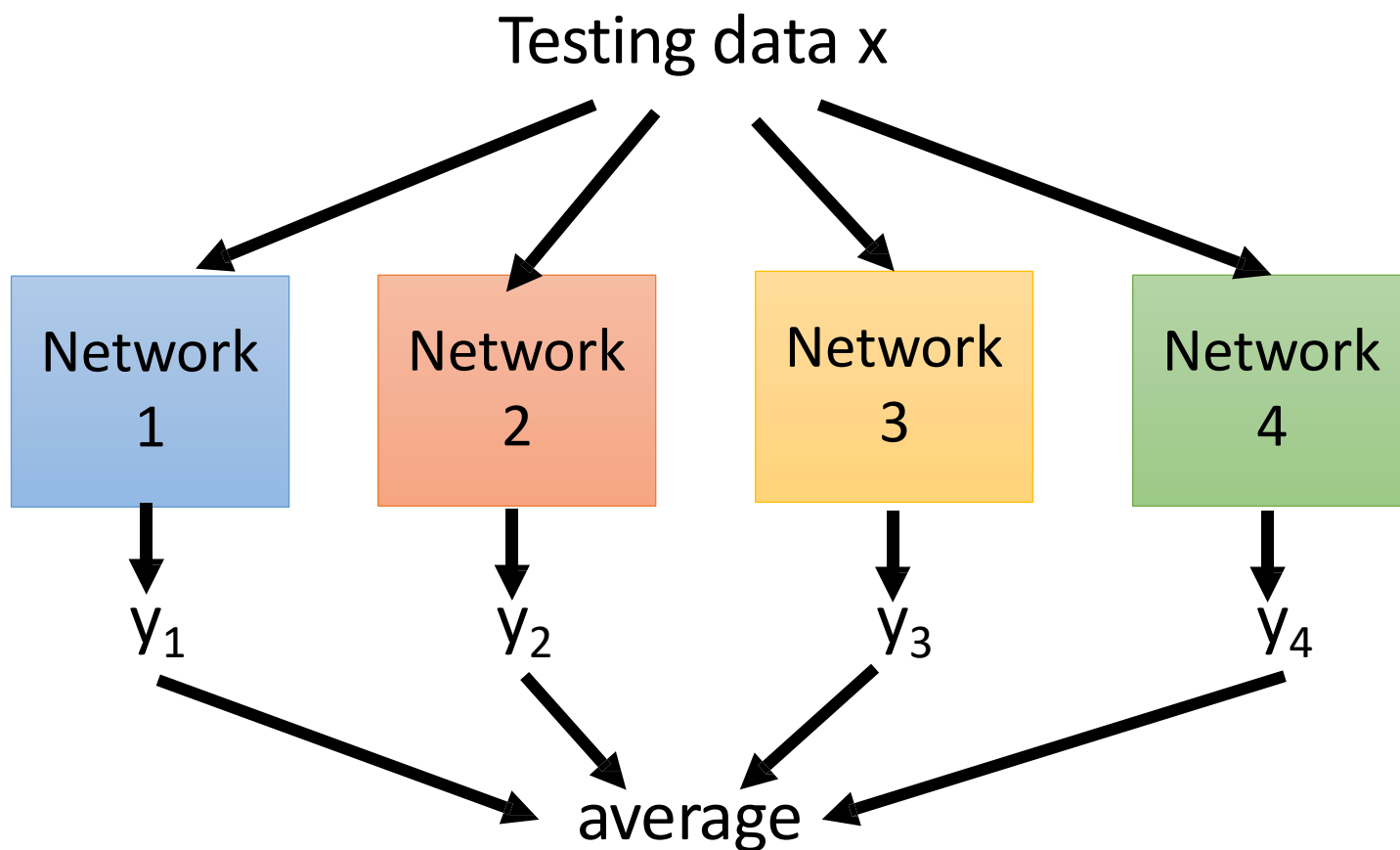
集成学习



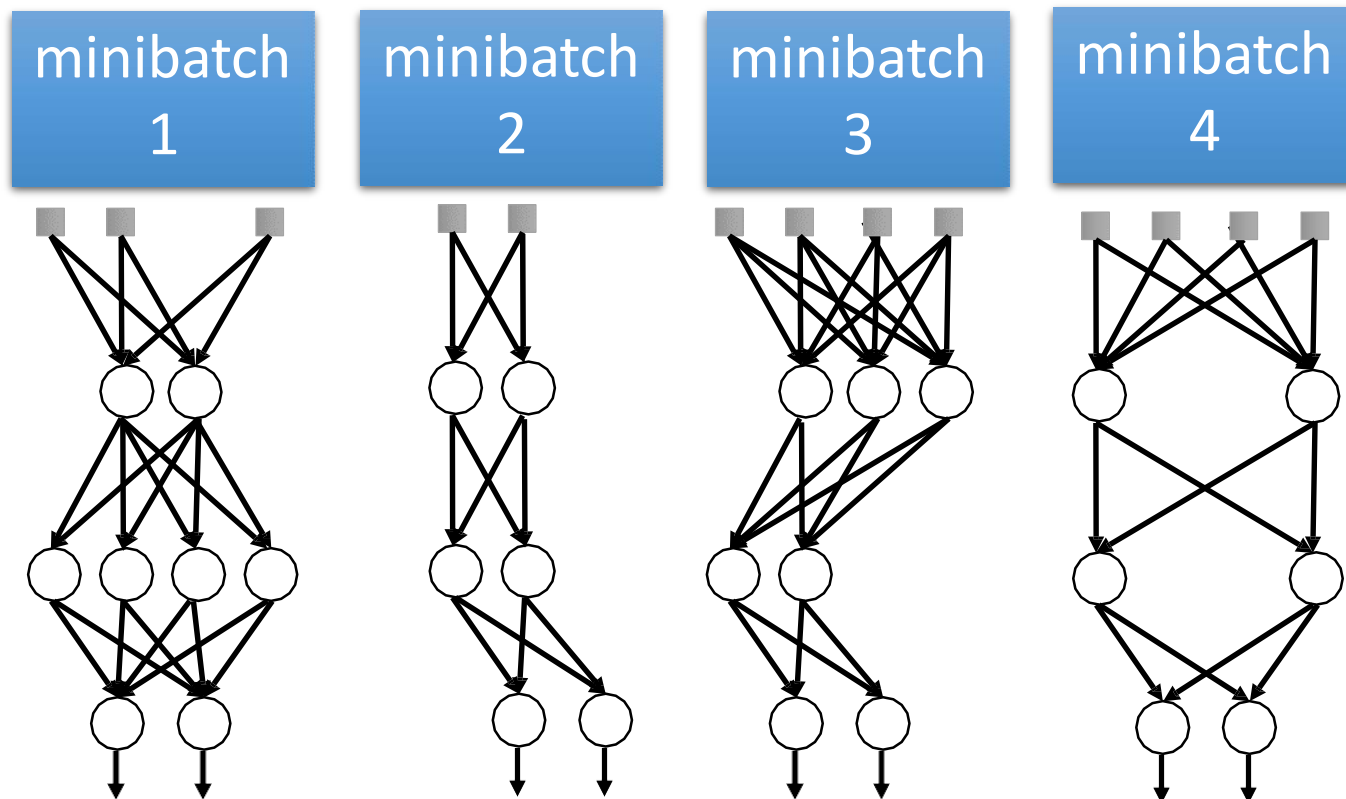
训练一组不同结构的网络

Dropout是一种集成学习

集成学习



Dropout是一种集成学习



训练过程

M 神经元

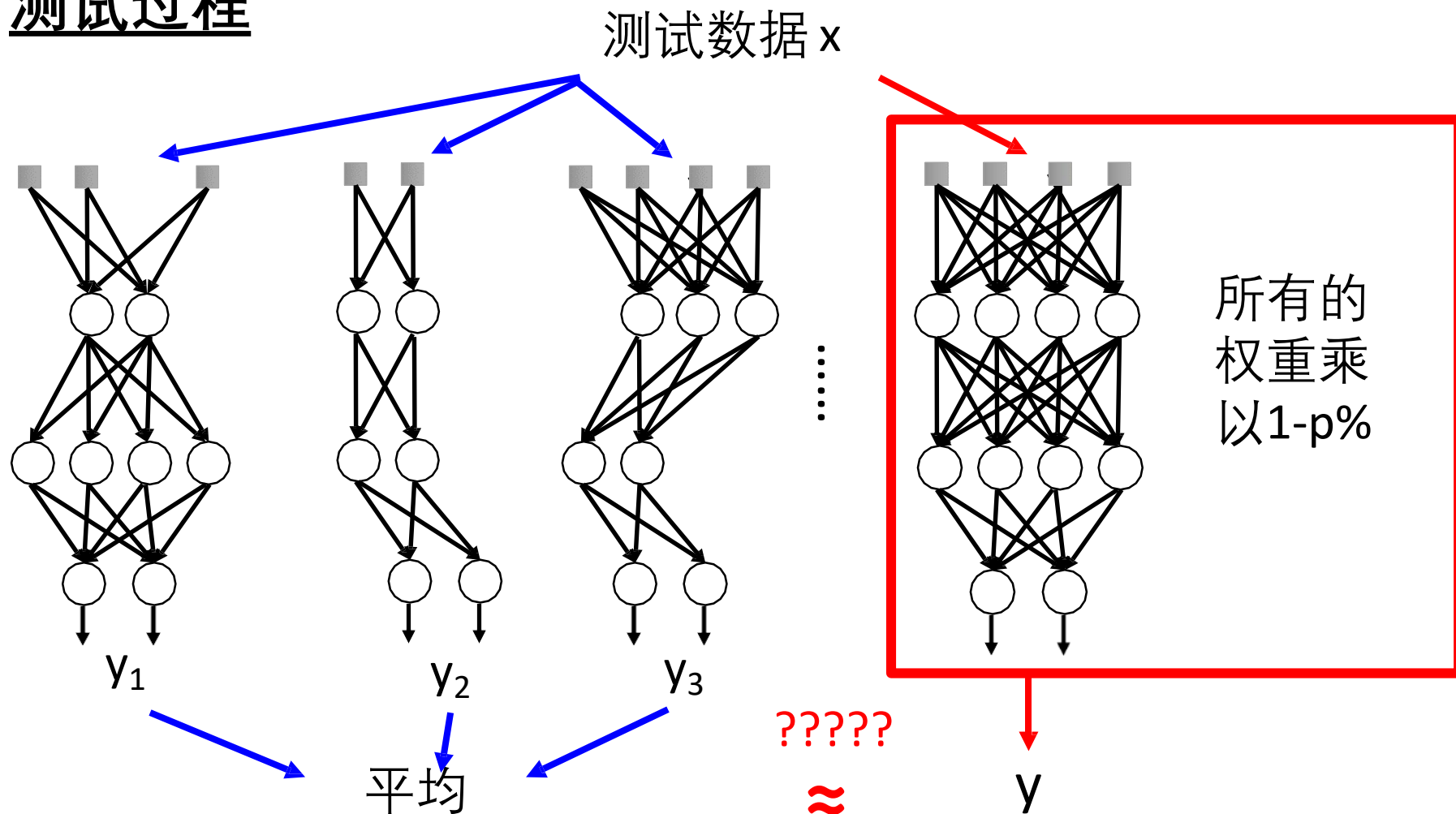


2^M 可能的
网络

- 用一个mini-batch训练一个网络
- 网络中的一些参数是共享的

Dropout是一种集成学习

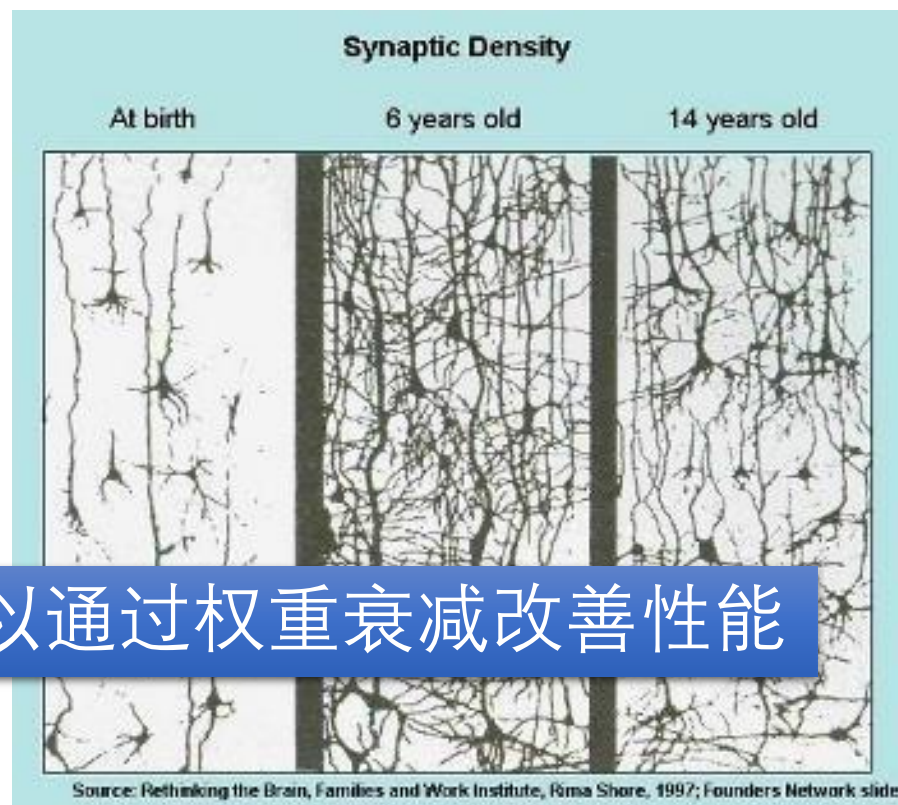
测试过程



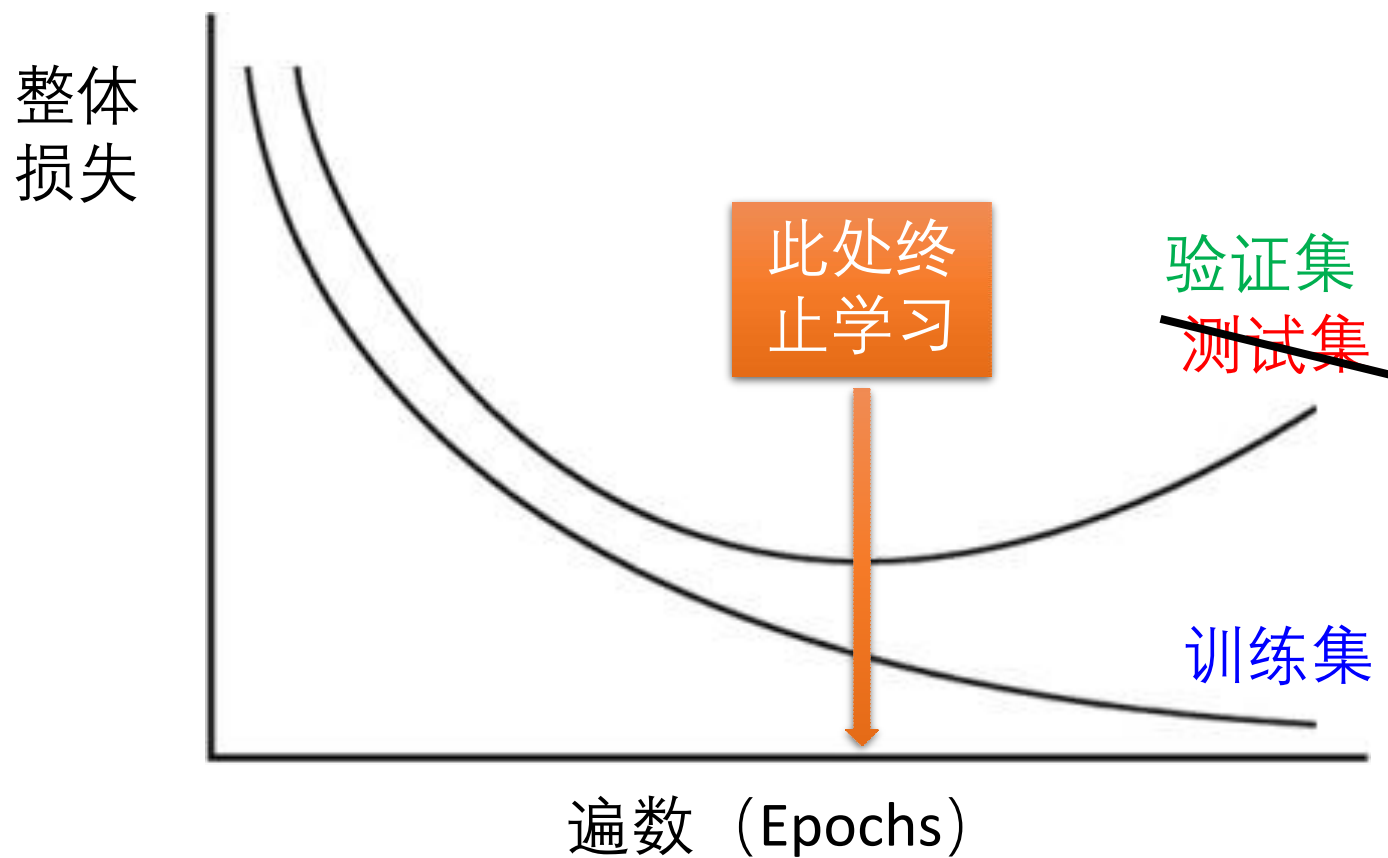
正则化－权重衰减

- 正则化即在损失函数中加入一个正则项。
- 直观地，正则化的作用是让网络偏好学习更小的权值，即复杂度低的网络。

人脑会将神经元之间
没用的连接滤去



提前终止



增加样本量

- 增加样本量可以减轻过拟合的风险
 - 过拟合本质上是模型对数据分布的学习过头了，以至于记住了数据的点点滴滴，这样过分的记忆导致模型在测试集上表现不好
 - 增加样本量使得模型记住数据全部细节变得困难
- 增加样本量能够更加直接地避免过拟合