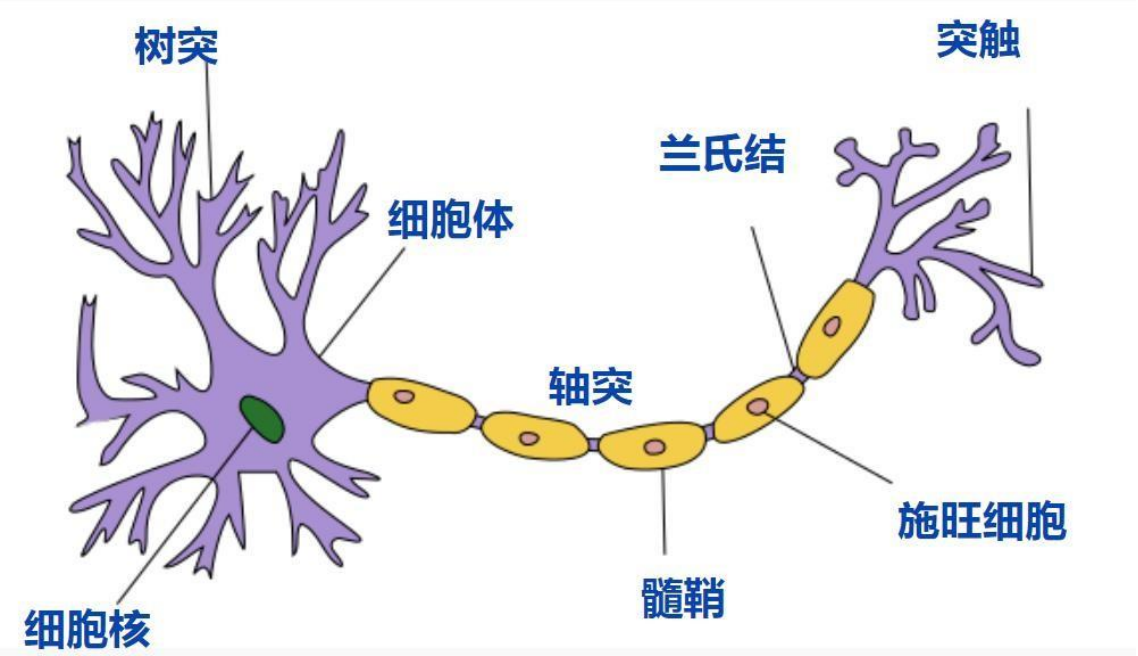
人工智能原理与技术

二、神经网络基础

1. 浅层神经网络

大家已经知道了，深度学习实际上是神经网络的一种，
那么相对应的，在深度神经网络之前有一种浅层神经网络，
我们来学习一下

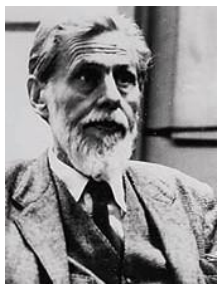
生物神经元



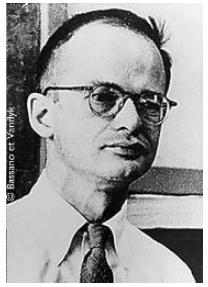
当前的大多数研究都是受生物启发，上面这个图大家没必要看懂，但是要知道它有下面四个特点：

- 每个神经元都是一个**多输入单输出**的信息处理单元；
- 神经元具有**空间整合**和**时间整合**特性；
- 神经元输入分**兴奋性输入**和**抑制性输入**两种类型；
- 神经元具有**阈值特性。**

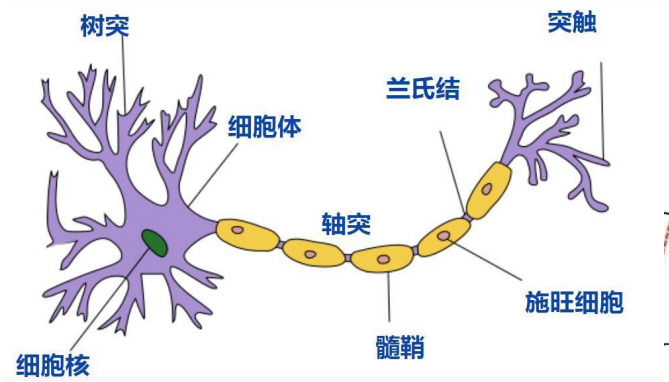
M-P神经元

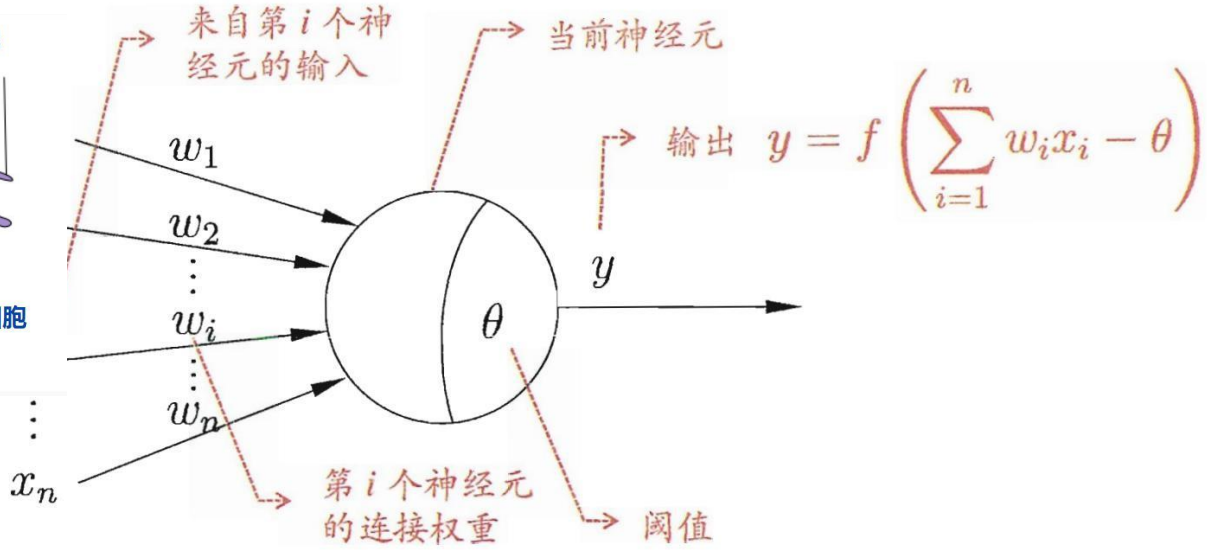


McCulloch



Pitts





- 多输入单输出
- 空间整合

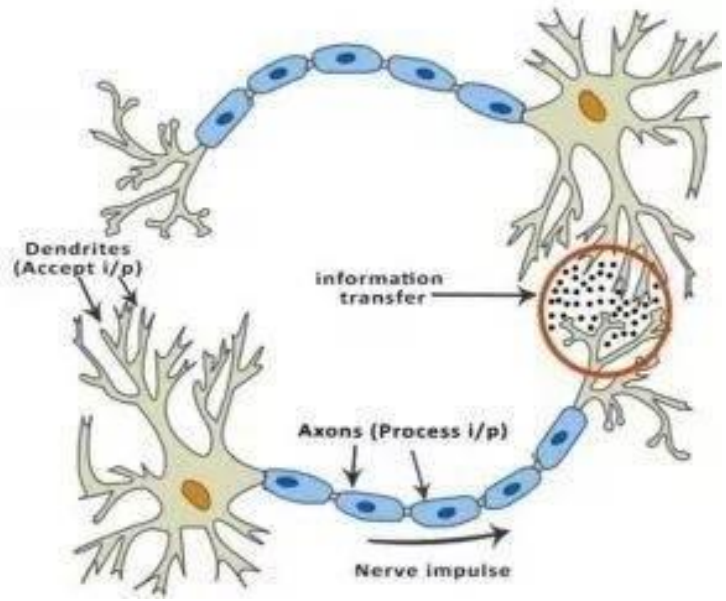
⇨ 多输入信号进行累加 $\Sigma_i x_i$

- 兴奋性/抑制性输入 ⇨ 权值w_i正负模拟兴奋\抑制，大小模拟强度
- 阈值特性 ⇨ 输入和超过阈值θ，神经元被激活(fire)

为什么需要激活函数

■ 激活函数*f*

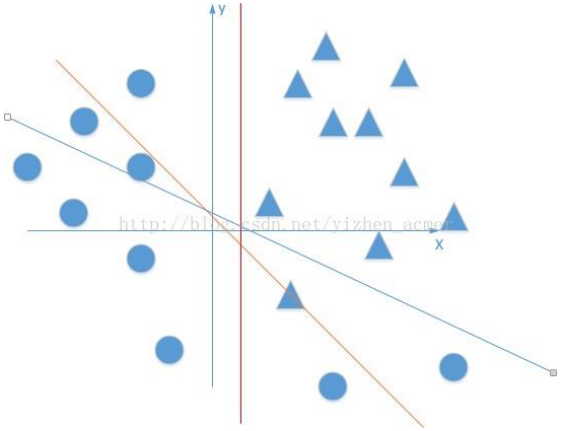
- 神经元继续传递信息、产生新连接的概率（超过阈值被激活，但不一定传递）



早上起床的闹钟，把你吵醒

- 没有激活函数相当于矩阵相乘
  - 多层和一层一样
  - 只能拟合线性函数

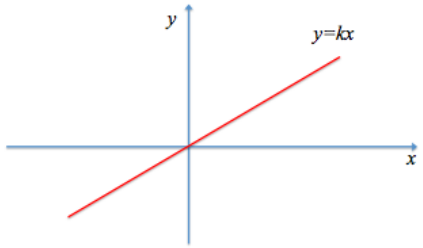
$$x^T W_1 \cdots W_n = x^T \prod_{k=1}^{n} W_k$$



激活函数举例

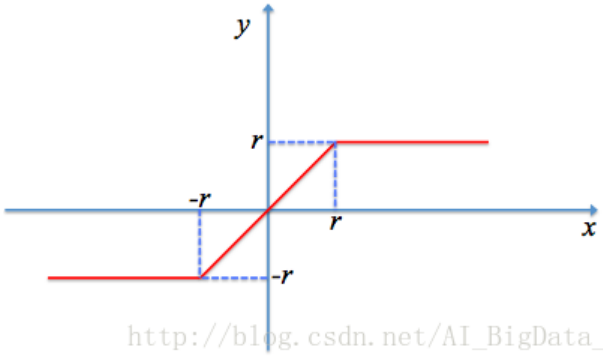
- 线性函数(e.g.,恒等函数)

$$f(x) = k * x + c$$



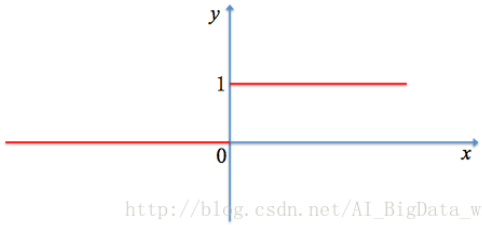
- 斜面函数

$$f(x) = \begin{cases} T, & x > c \\ k * x, & |x| \leq c \\ -T, & x < -c \end{cases}$$



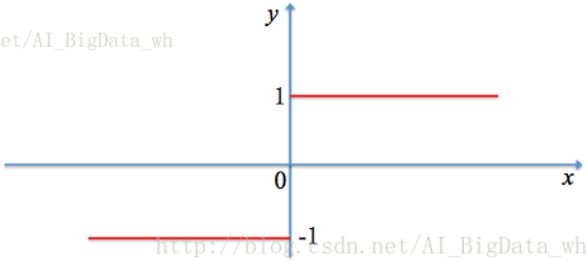
- 阈值函数（e.g.,阶跃函数）

$$f(x) = \begin{cases} 1, x \geq c \\ 0, x < c \end{cases}$$



- 符号函数

$$y = F(x) = \begin{cases} 1, & \text{当} x > 0 \\ -1, & \text{当} x \leq 0 \end{cases}$$

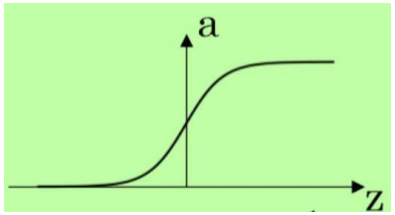


激活函数举例

问题：输入为特别大的负值，跳不出去

S性函数(sigmoid)

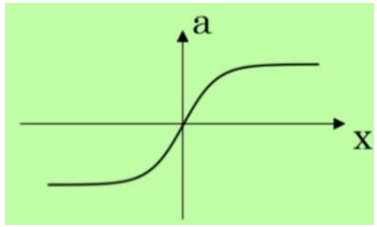
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$\sigma(z)' = \sigma(z)(1-\sigma(z))$$

问题1，容易饱和
问题2，输出不对称

双极S性函数(tanh)



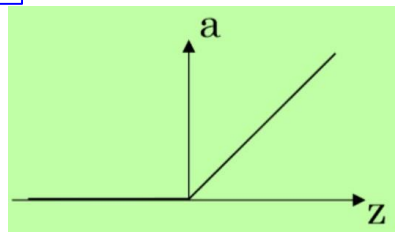
$$\tanh(x) = 2sigmoid(2x) - 1 = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

$$g(z)' = 1 - g(z)^2$$

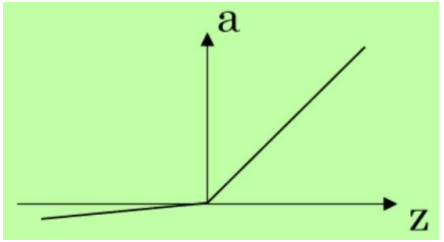
ReLU修正线性单元

$$relu(z) = max(0, z)$$

- ✓ 当z为正的时候，导数恒为1，
- ✓ 当z为负的时候，导数恒为0。



Leaky ReLU

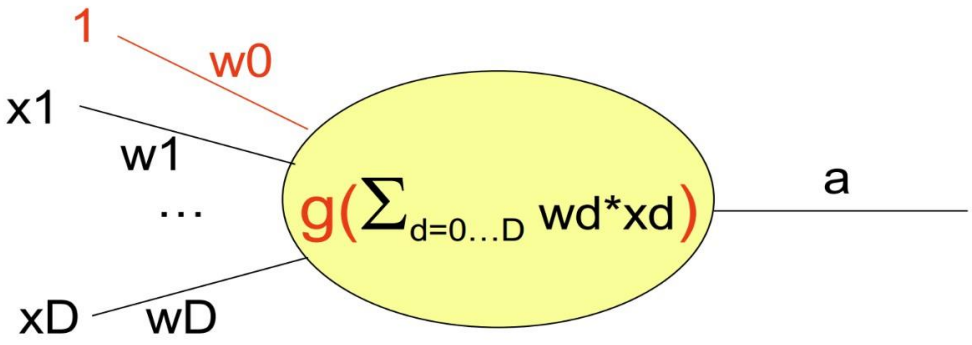


$$leakyrelu(z) = max(0.01z, z)$$

单层感知器

- M-P神经元的权重预先设置，无法学习
- 单层感知器是首个可以学习的人工神经网络

Rosenblat与感知器

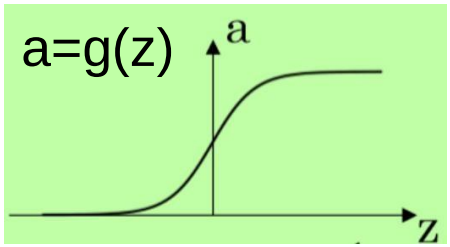


线性激活函数: g(z)=z

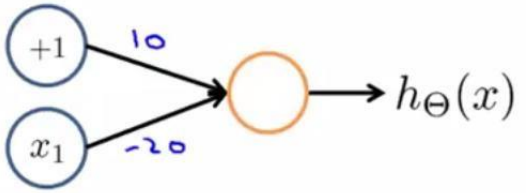
a=W * x;

单层感知器

- 非线性激活函数



逻辑非



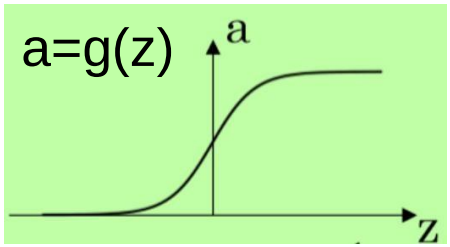
$h_\Theta(x) = g(10 - 20x_1)$

同学们计算，
当 x = 0 的时候，（10-20x）是？最终输出是？
当 x = 1 的时候，（10-20x）是？最终输出是？

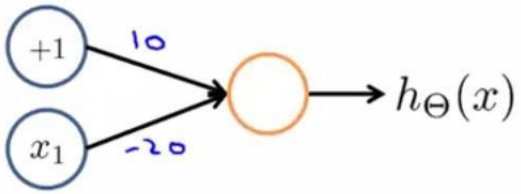
当 x = 0 ，（10-20x）是10，输出是1
当 x = 1 ，（10-20x）是-10，输出是0
实际上进行了逻辑非运算

单层感知器

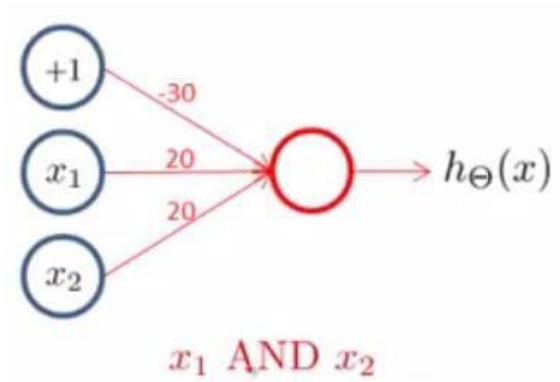
- 非线性激活函数

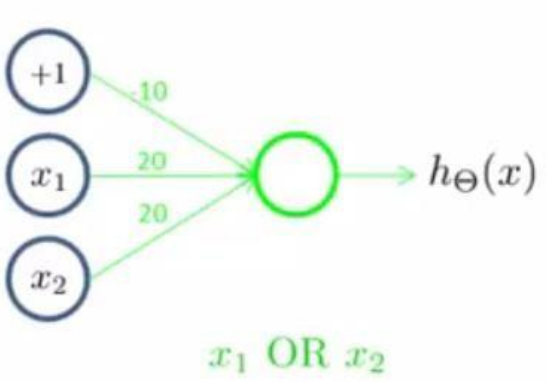


逻辑非



$h_\Theta(x) = g(10 - 20x_1)$

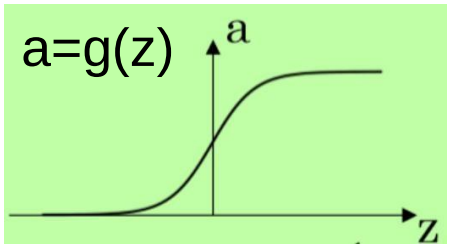




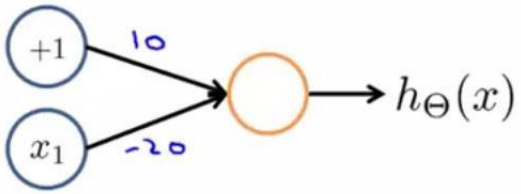
x1=0, x2=0 时，-30+20x1+20x2=-30，结果为0
x1=0, x2=1 时，-30+20x1+20x2=-10，结果为0
x1=1, x2=0 时，-30+20x1+20x2=-10，结果为0
x1=1, x2=1 时，-30+20x1+20x2=10，结果为1
实际上进行了逻辑与运算

单层感知器

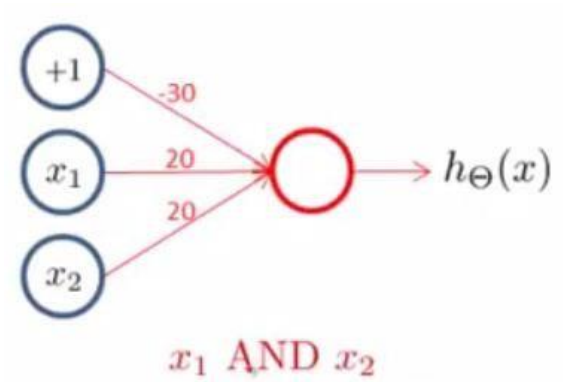
- 非线性激活函数

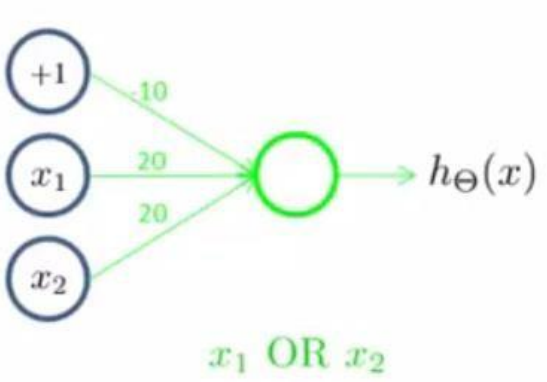


逻辑非



$h_\Theta(x) = g(10 - 20x_1)$

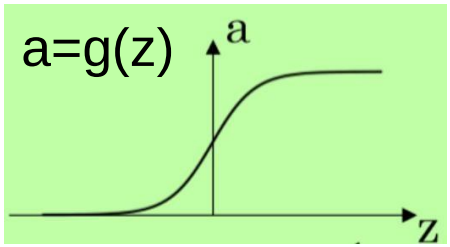




x1=0, x2=0 时，-10+20x1+20x2=-10，结果为0
x1=0, x2=1 时，-10+20x1+20x2=10，结果为1
x1=1, x2=0 时，-10+20x1+20x2=10，结果为1
x1=1, x2=1 时，-10+20x1+20x2=30，结果为1
实际上进行了逻辑或运算

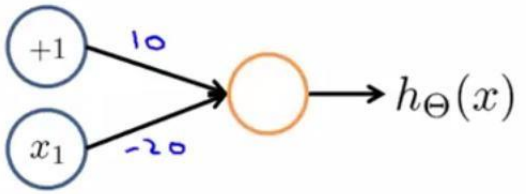
单层感知器

- 非线性激活函数

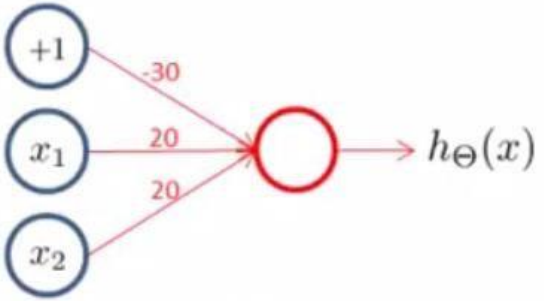


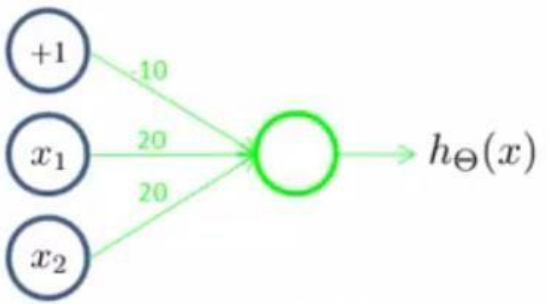
放在一个分类问题里：

逻辑非

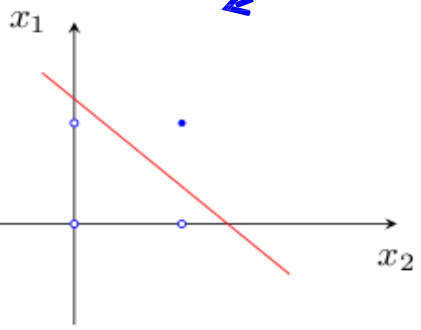


$h_\Theta(x) = g(10 - 20x_1)$

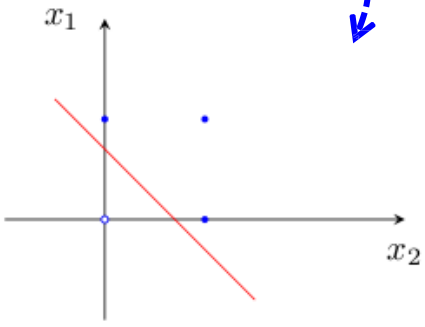




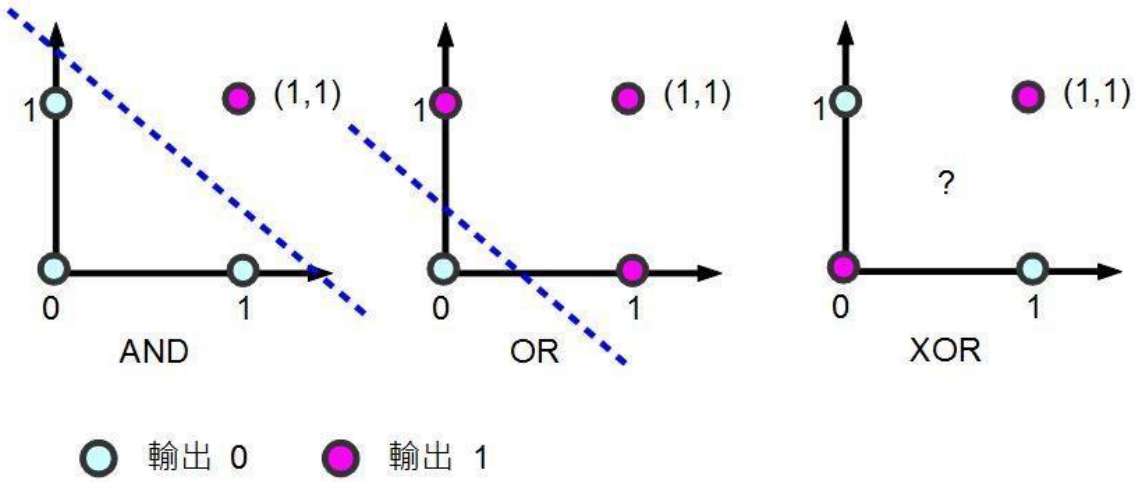
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



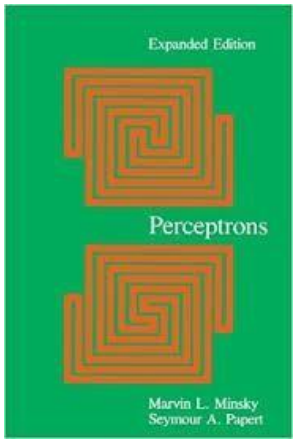
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



单层→多层感知器



感知器陷入低谷是因为明斯基指出感知器无法解决异或问题



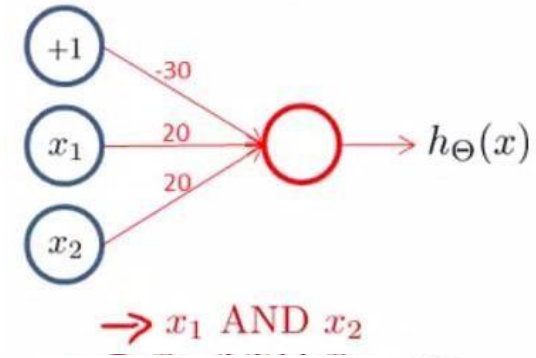
扫码有证明

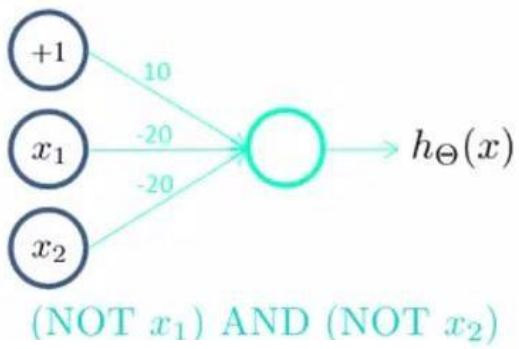
多层感知器

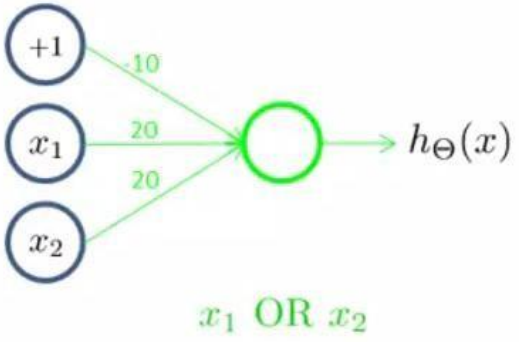
■ 三层感知器实现同或门

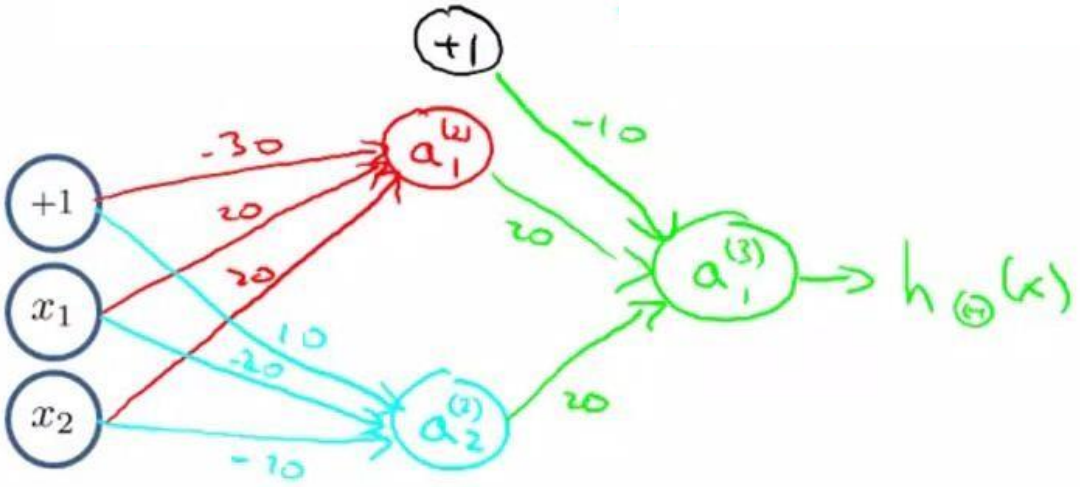
a XOR b = ((NOT a) AND b) OR (a AND (NOT b));

a XNOR b = NOT (a XOR b) = (a AND b) OR ((NOT a) AND (NOT b))





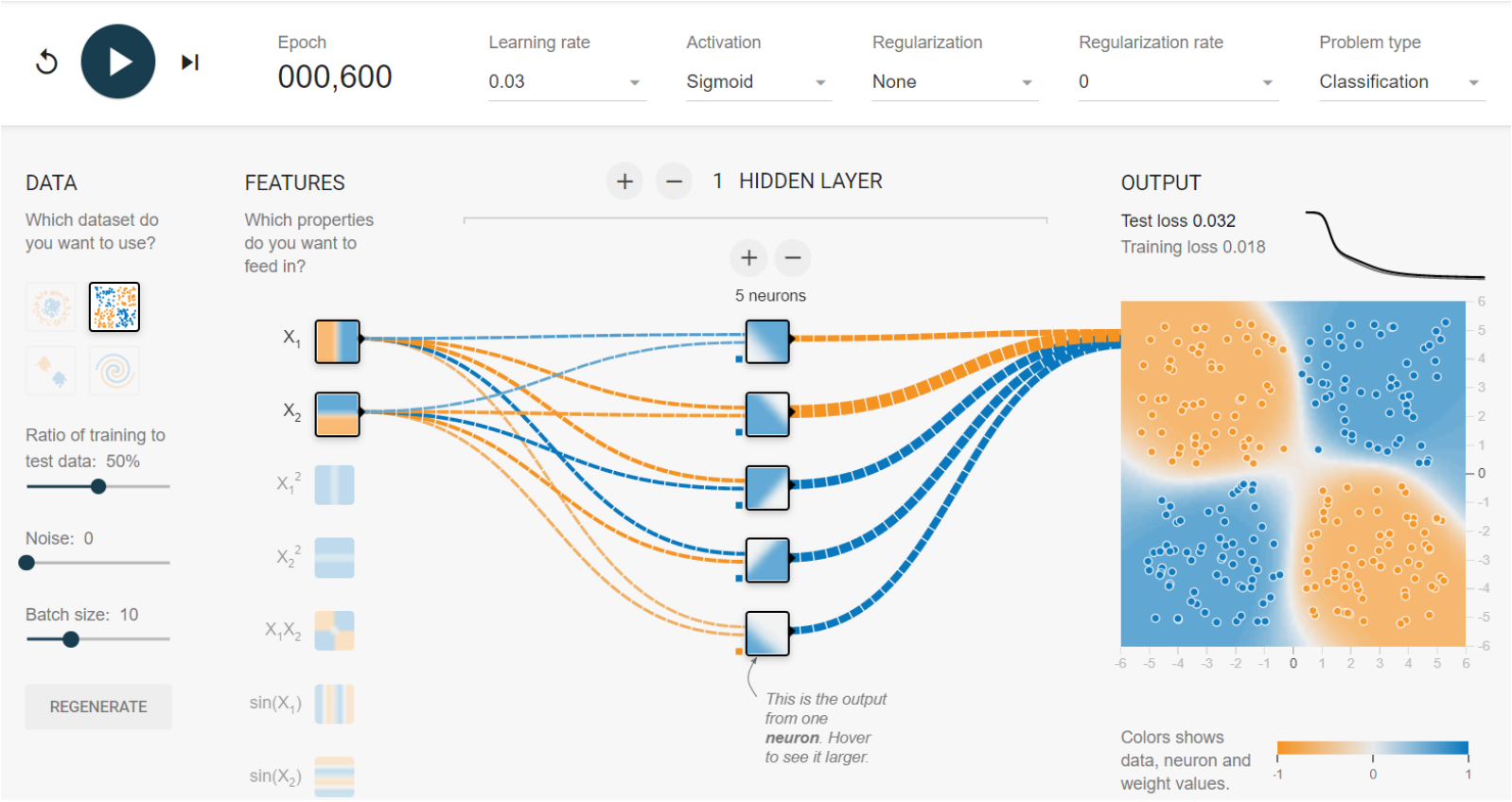




x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

单隐层神经网络可视化

http://playground.tensorflow.org/

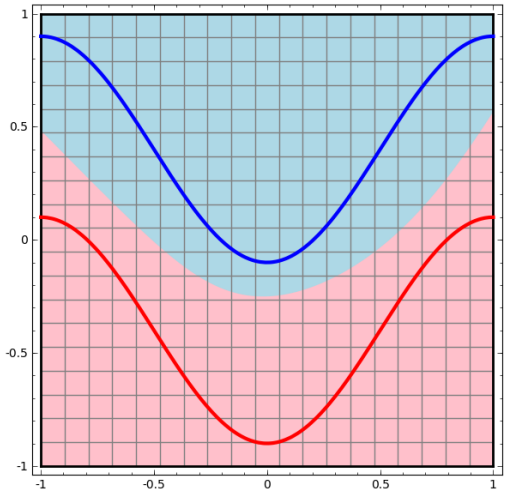


演示：单隐层，可以解决二分类问题

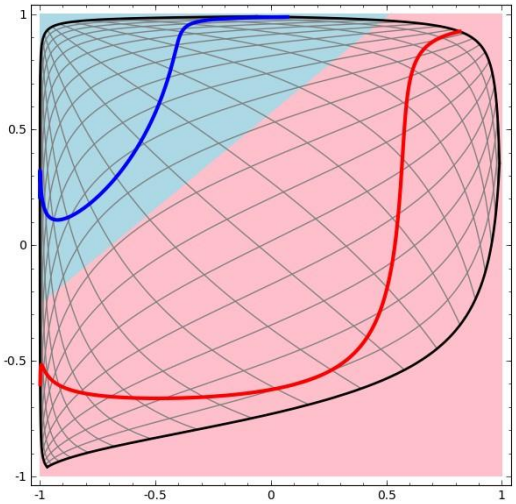
万有逼近定理

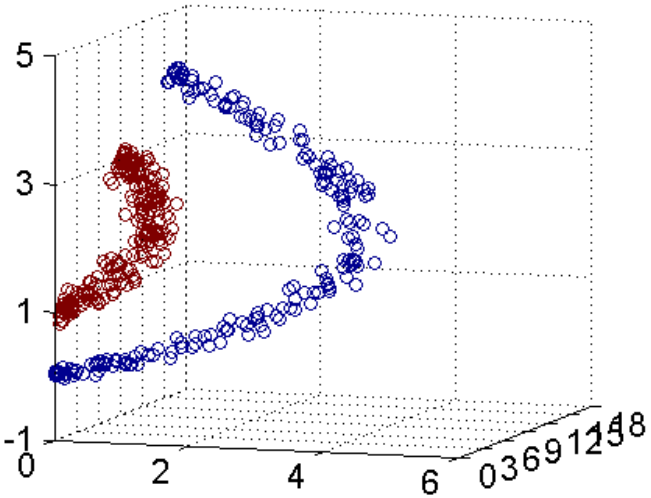
■ 万有逼近定理

如果一个隐层包含足够多的神经元，三层前馈神经网络（输入-隐层-输出）能以任意精度逼近任意预定的连续函数。



为什么线性分类任务组合后可以解决非线性分类任务？

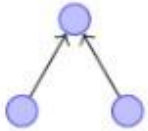
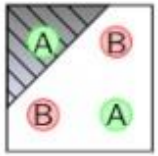
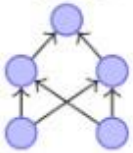
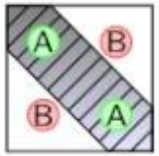
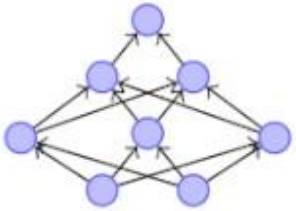
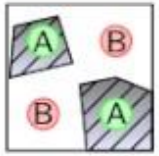




万有逼近定理

■ 双隐层感知器逼近非连续函数

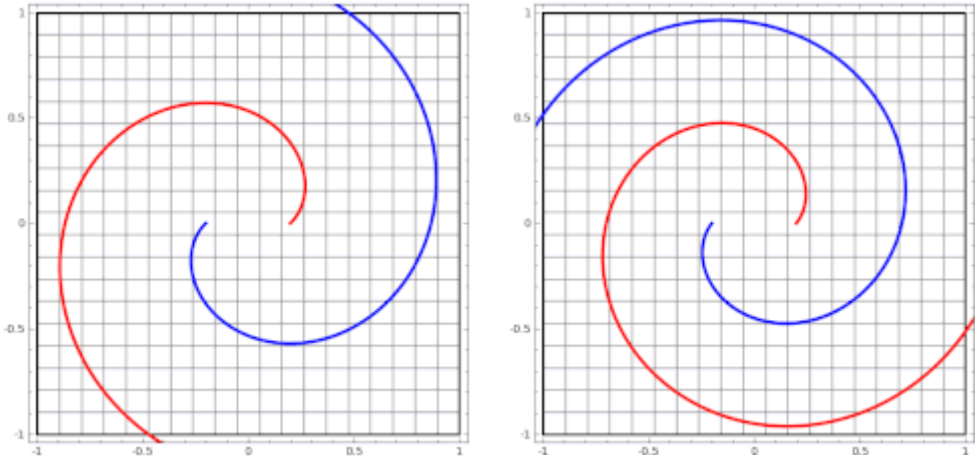
当隐层足够宽时，双隐层感知器（输入-隐层1-隐层2-输出）可以逼近任意非连续函数：可以解决任何复杂的分类问题。

结构	决策区域类型	区域形状	异或问题
无隐层 	由一超平面分成两个		
单隐层 	开凸区域或闭凸区域		
双隐层 	任意形状（其复杂度由单元数目确定）		

神经网络每一层的作用

- 每一层数学公式 $\vec{y} = a(W \cdot \vec{x} + b)$
- 完成输入→输出空间变换
  - 升维/降维
  - 放大/缩小
  - 旋转

  } $W \cdot x$
  - 平移 → $+b$
  - 弯曲 → $a(\cdot)$



Demo： https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html

■神经网络学习如何利用矩阵的线性变换加激活函数的非线性变换，将原始输入空间投影到线性可分的空间去分类/回归。

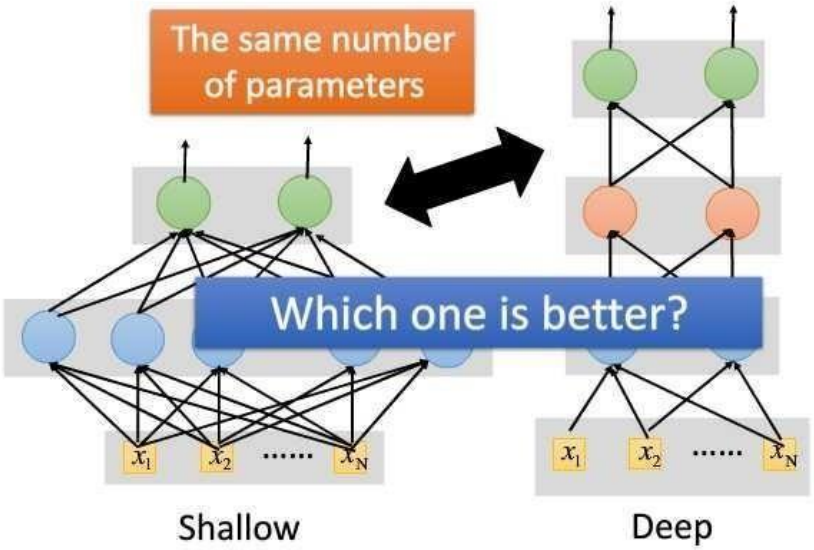
- **增加节点数：增加维度，即增加线性转换能力。**
- **增加层数：增加激活函数的次数，即增加非线性转换次数**

更宽or更深?

万有逼近定理说，一个网络中间隐层节点足够多就可以了。前面我们又看到神经网络需要有足够多的层数，这个矛盾我们如何来协调？到底是要瘦高的，还是要矮胖的？

Fat + Short v.s. Thin + Tall

Fat + Short v.s. Thin + Tall



Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcriptio
Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

更宽or更深?

■ 更宽还是更深? 更深!

- 在神经元总数相当的情况下，增加网络深度可以比增加宽度带来更强的网络表示能力：产生更多的线性区域[1]。

- 深度和宽度对函数复杂度的贡献是不同的，深度的贡献是指数增长的，而宽度的贡献是线性的[2]。

$$FC = \prod_{l=1}^{d} (\alpha_l \cdot \theta_l)^{\beta_l}$$

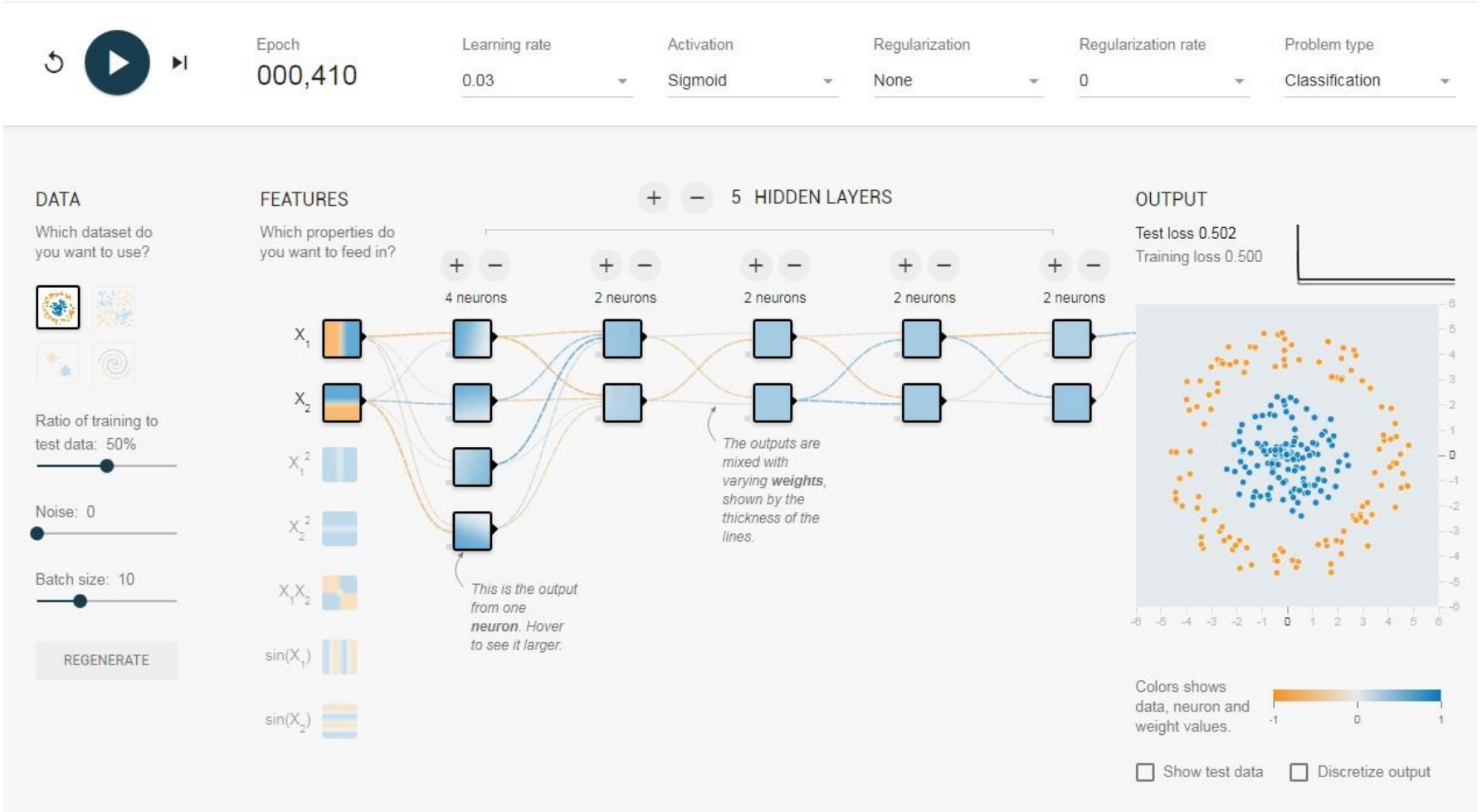
其中 α 表示参数每层参数对函数复杂度的贡献， θ 表示参数数量， β 表示深度对函数复杂度的贡献， α 和 β 都是一个区间即相同的参数在不同数值下仍然有不同的复杂度。 d 表示最大深度， l 表示第 l 层。

1 On the Number of Linear Regions of Deep Neural Networks.

2 Delalleau and Y. Bengio. Shallow vs. deep sum-product networks. In NIPS, 2011

多层神经网络的问题：梯度消失

http://playground.tensorflow.org/



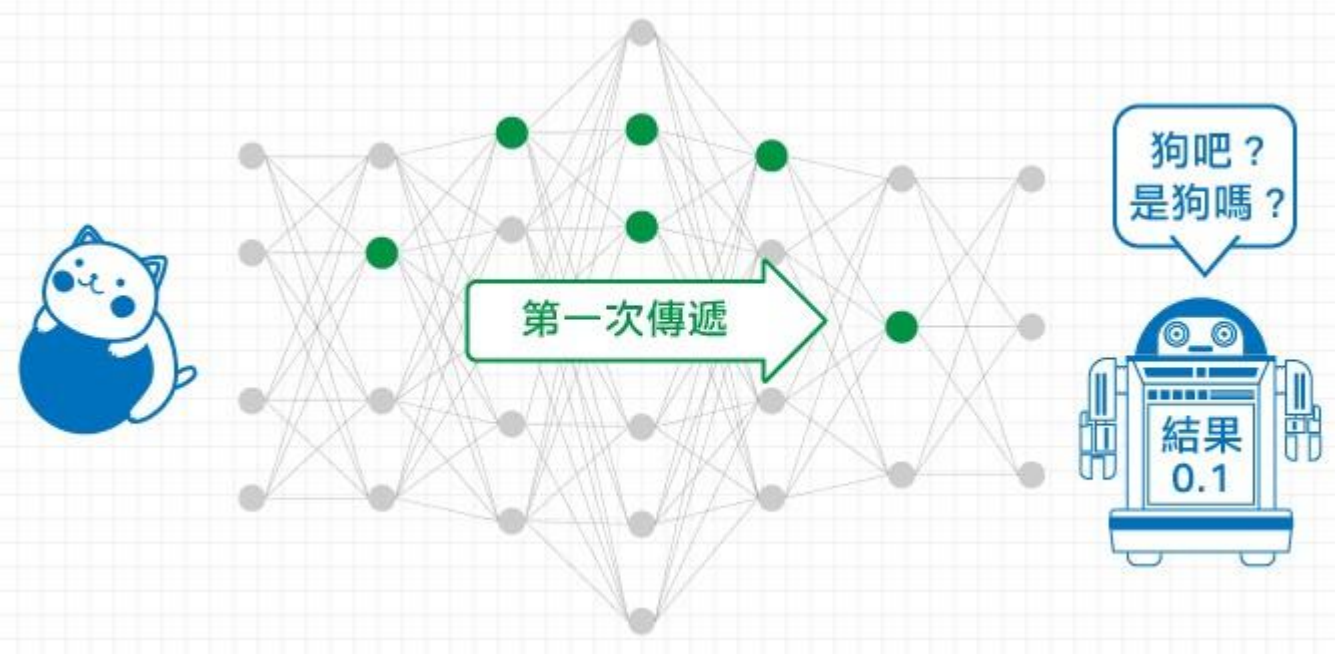
演示：多加几层以后，网络训不动了

神经网络的参数学习：误差反向传播

- 多层神经网络可看成是一个复合的非线性多元函数 $F(\cdot): X \to Y$

$$F(x) = f_n(\ldots f_3(f_2(f_1(x) * \theta_1 + b) * \theta_2 + b)\ldots)$$

- 给定训练数据$\{x^i, y^i\}_{i=1:N}$，希望损失$\sum_i loss(F(x^i), y^i)$尽可能小

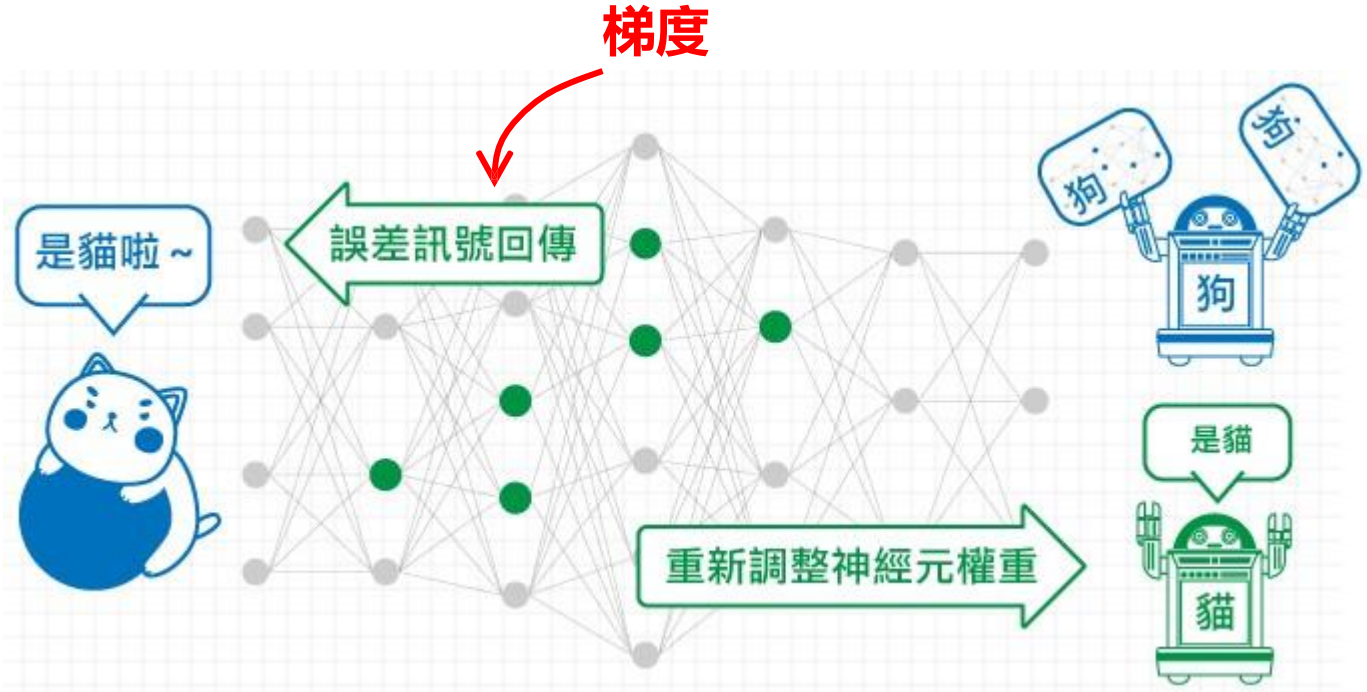


神经网络的参数学习：误差反向传播

- 多层神经网络可看成是一个复合的非线性多元函数 $F(\cdot): X \to Y$

$$F(x) = f_n(\ldots f_3(f_2(f_1(x) * \theta_1 + b) * \theta_2 + b)\ldots)$$

- 给定训练数据$\{x^i, y^i\}_{i=1:N}$，希望损失$\sum_i loss(F(x^i), y^i)$尽可能小

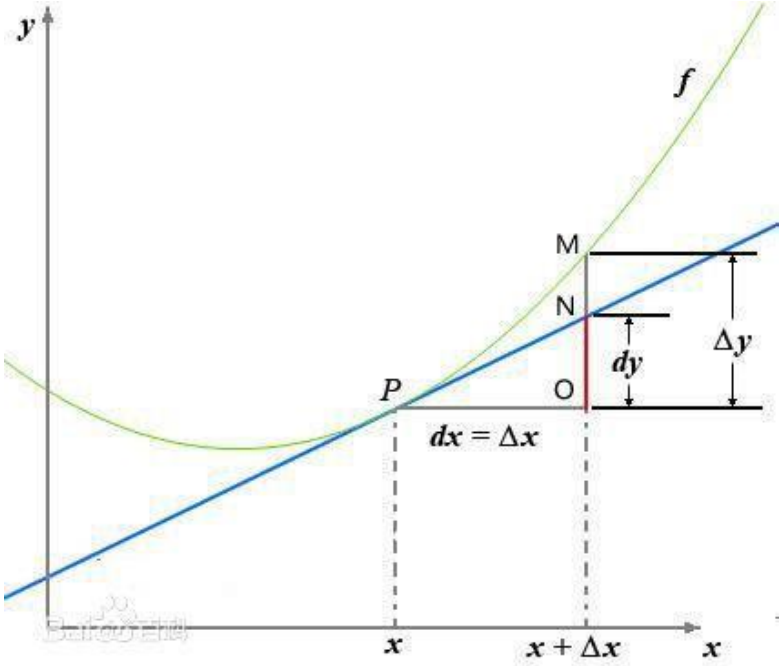


梯度和梯度下降

■导数：

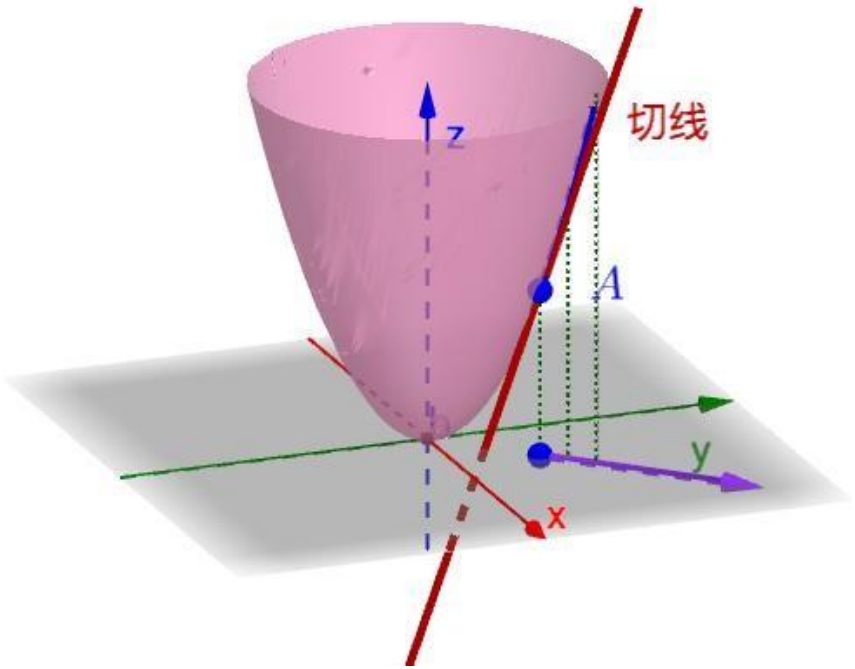
函数值在某一点沿自变量正方向的变化率

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$



■ 梯度：

- 多元函数$f(x,y)$在每个点可以有多个方向
- 每个方向都可以计算导数，称为方向导数

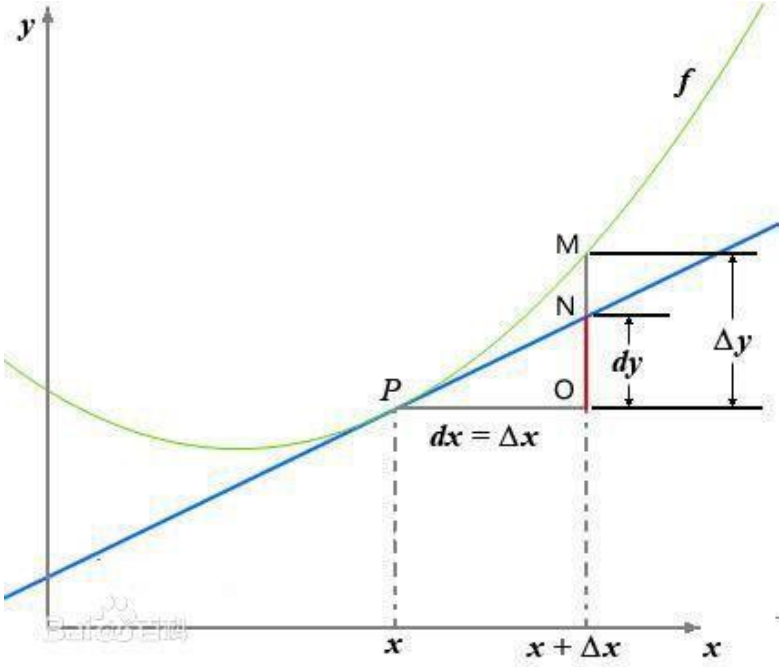


梯度和梯度下降

■导数：

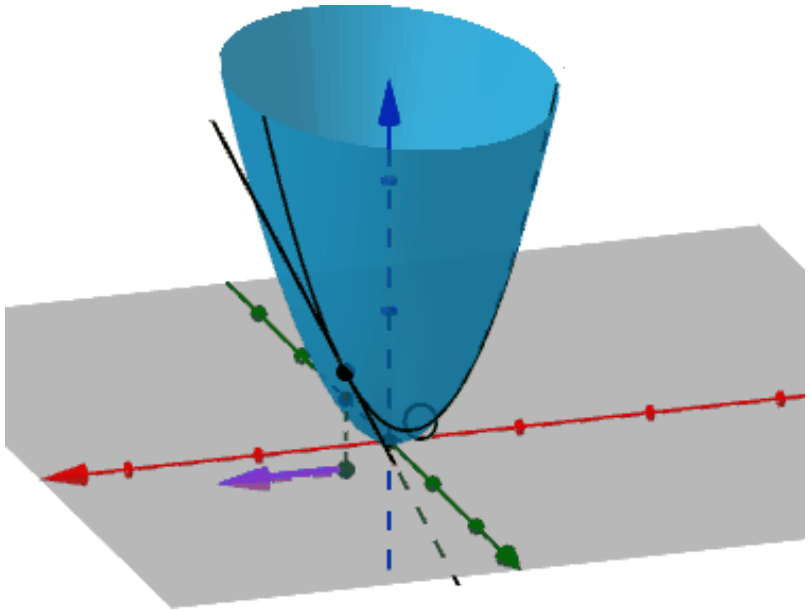
函数值在某一点沿自变量正方向的变化率

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$



■ 梯度：

- 多元函数$f(x,y)$在每个点可以有多个方向
- 每个方向都可以计算导数，称为方向导数



用马同学的DEMO演示

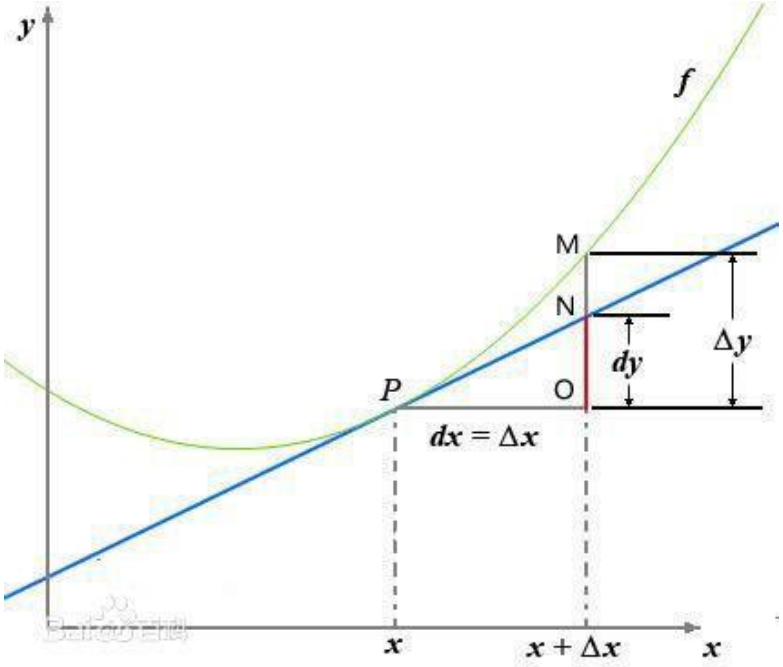
Demo: https://www.matongxue.com/madocs/222.html

梯度和梯度下降

■导数：

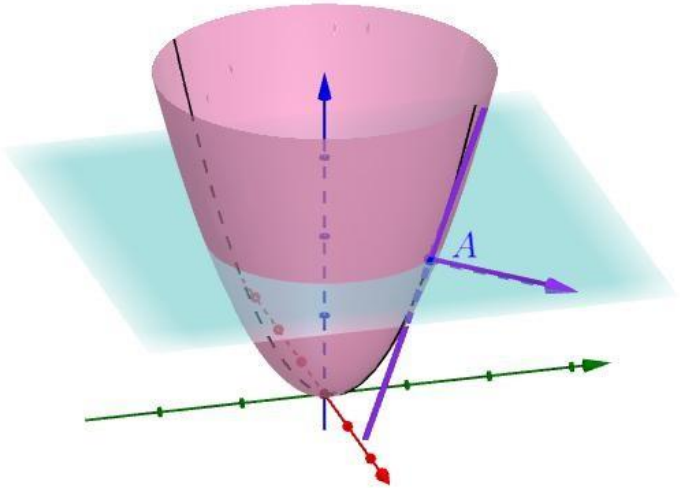
函数值在某一点沿自变量正方向的变化率

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

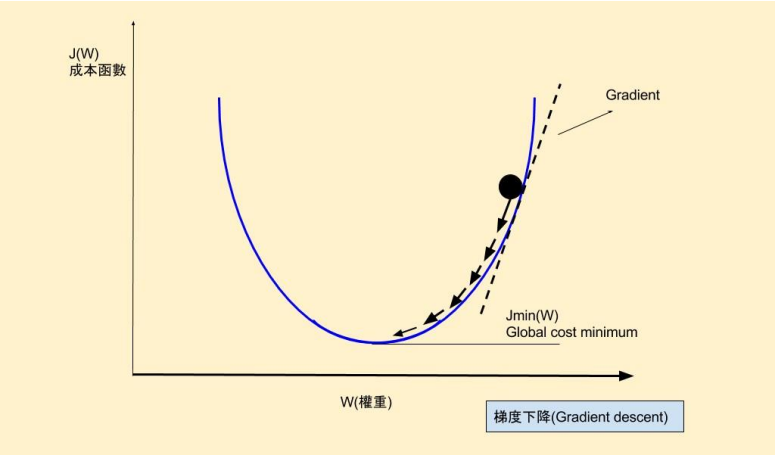


■ 梯度：

- 多元函数$f(x,y)$在每个点可以有多个方向
- 每个方向都可以计算导数，称为方向导数
- 梯度是一个向量
  - 方向是最大方向导数的方向
  - 模为方向导数的最大值



梯度和梯度下降

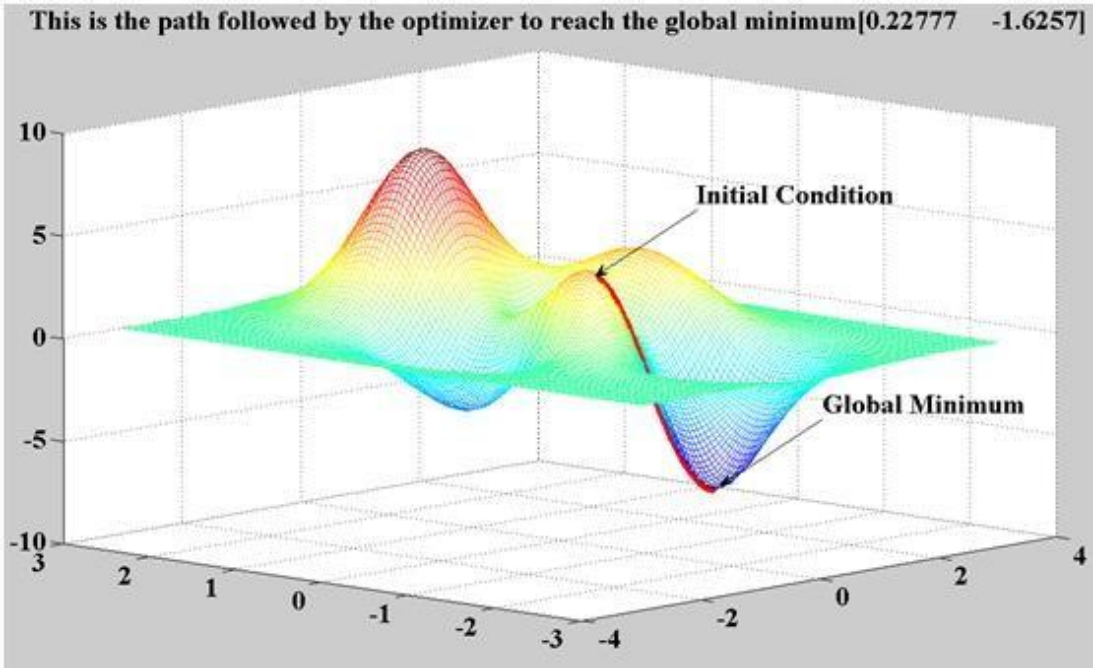


凸函数，只有一个最小值

- **无约束优化：梯度下降**

→ 参数沿负梯度方向更新可以使函数值下降

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

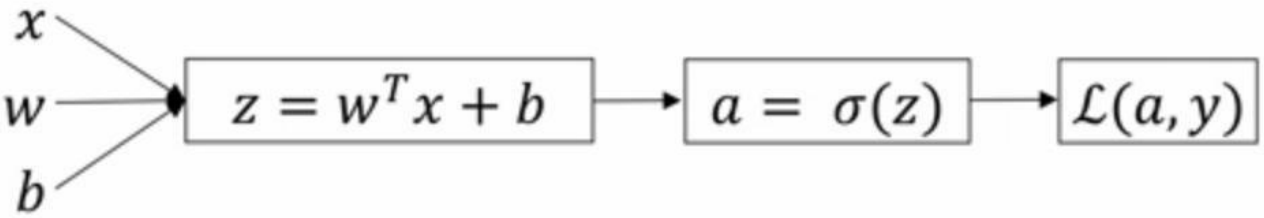


非凸函数，多个极小值，会非常依赖初始点的选择

神经网络的参数学习：误差反向传播

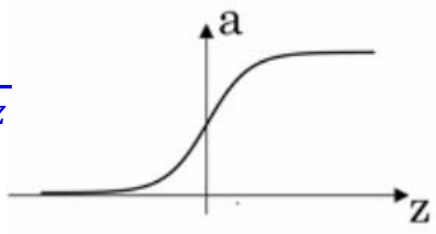
复合函数的链式求导

Logistic regression



$\frac{\partial L}{\partial w_i} = ?$ $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_i} \triangleq dw_i$

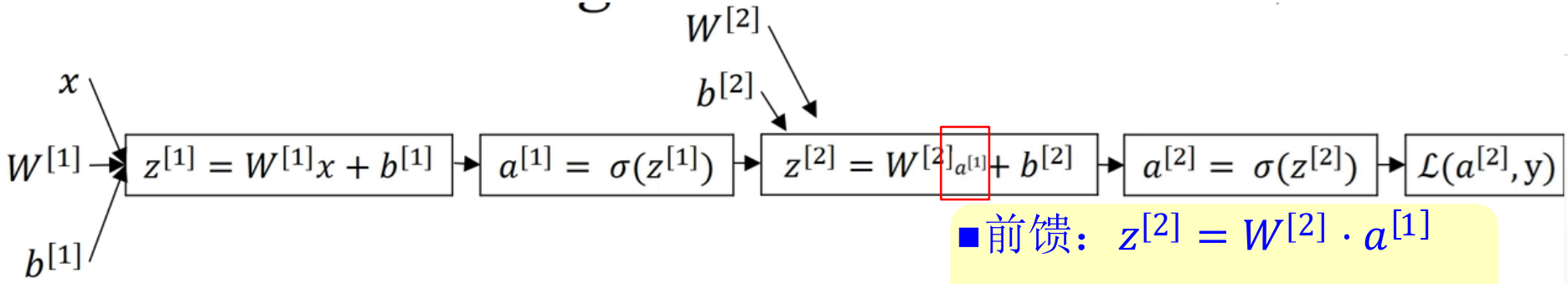
$L = \sum_{k=1}^{c} (a_k - y_k)^2$ $\sigma(z) = \frac{1}{1 + e^{-z}}$



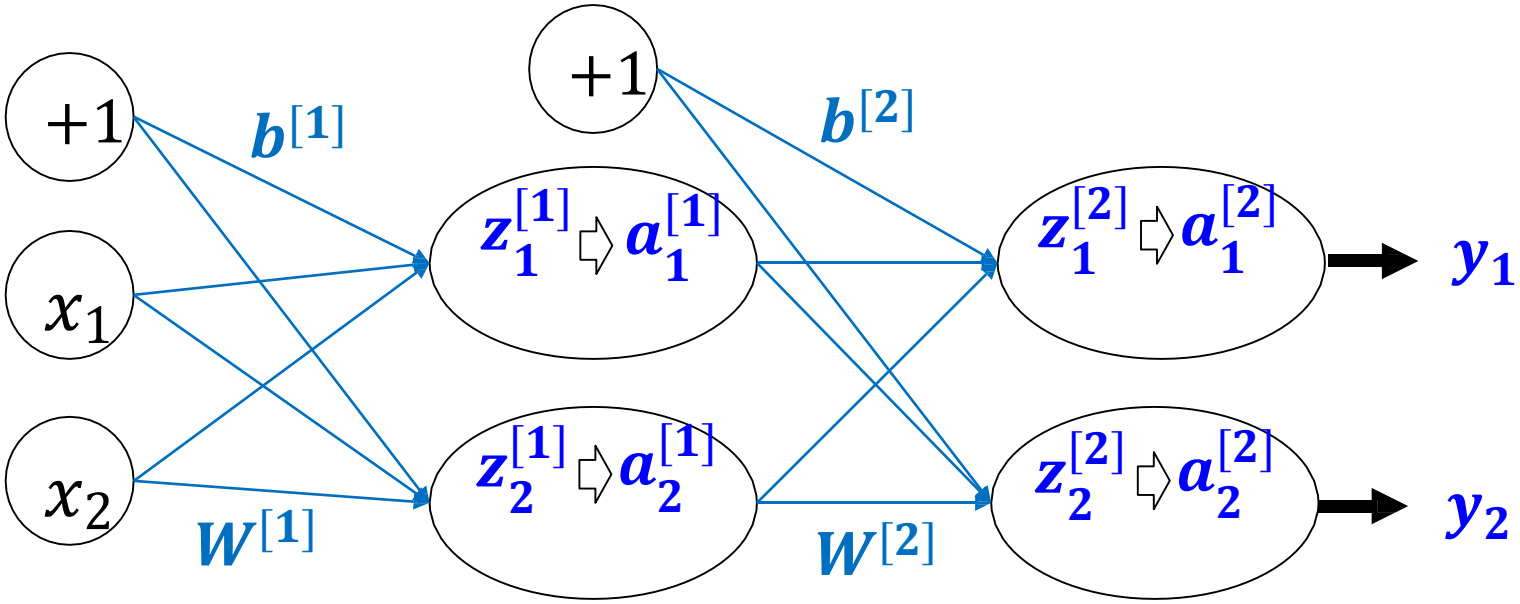
$$\frac{\partial L}{\partial w_i} = 2(a - y) \cdot a(1 - a) \cdot x$$

神经网络的参数学习：误差反向传播

■ 三层前馈神经网络的BP算法

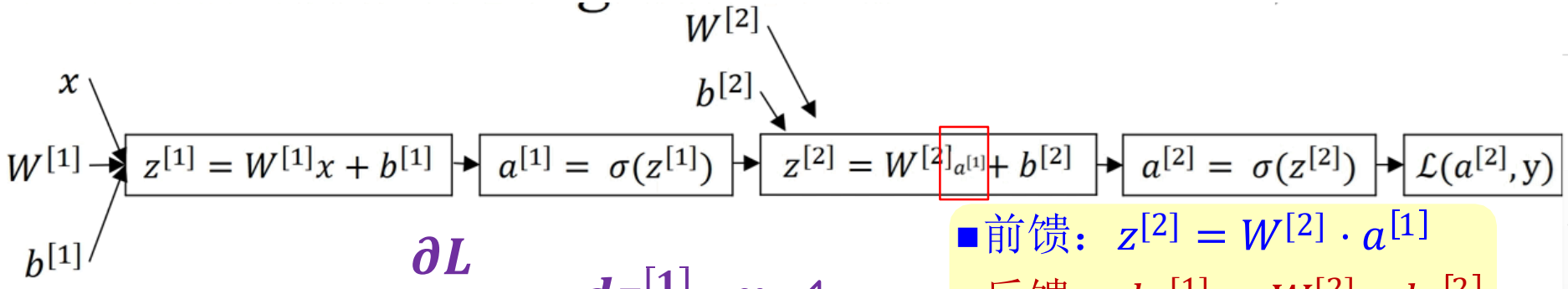


■ 前馈：$z^{[2]} = W^{[2]} \cdot a^{[1]}$



神经网络的参数学习：误差反向传播

■ 三层前馈神经网络的BP算法



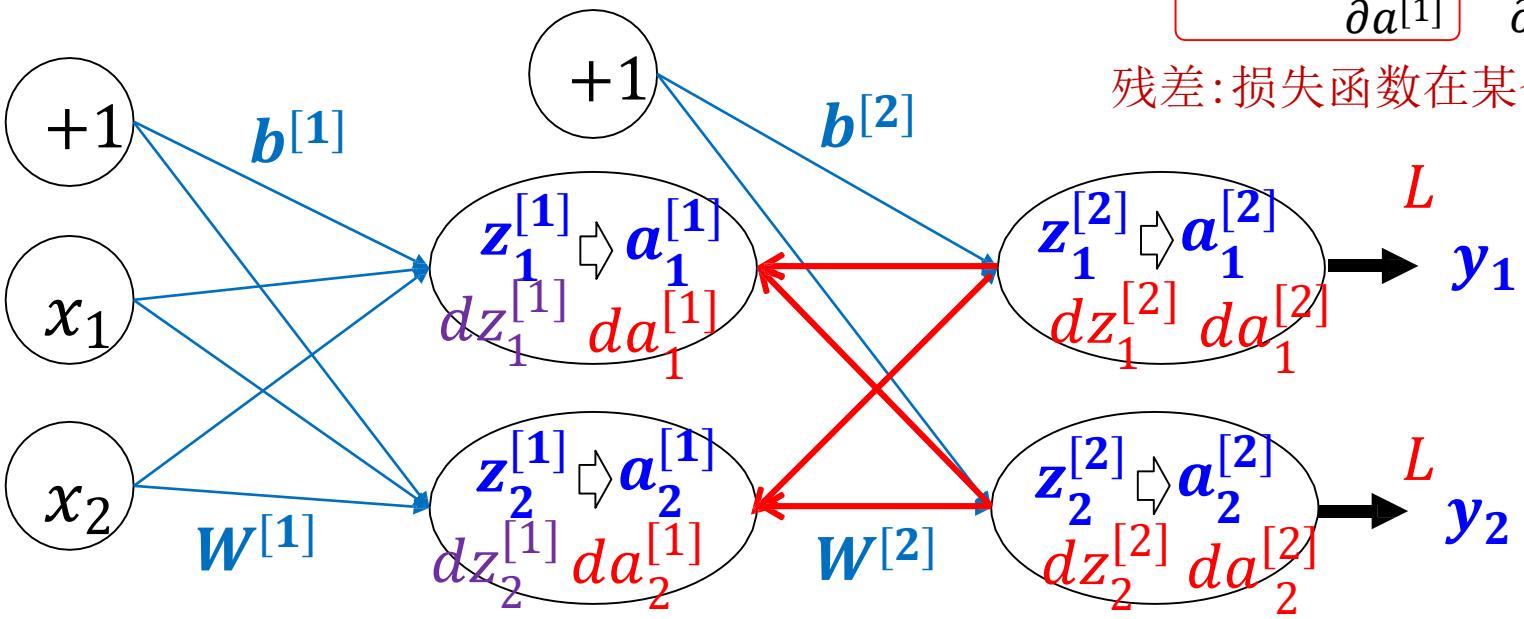
$$\frac{\partial L}{\partial W^{[1]}} = dz^{[1]} \cdot x \Longleftarrow$$

■前馈：$z^{[2]} = W^{[2]} \cdot a^{[1]}$

■反馈：$da^{[1]} = W^{[2]} \cdot dz^{[2]}$

$$da^{[1]} = \frac{\partial L}{\partial a^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \cdot W^{[2]}$$

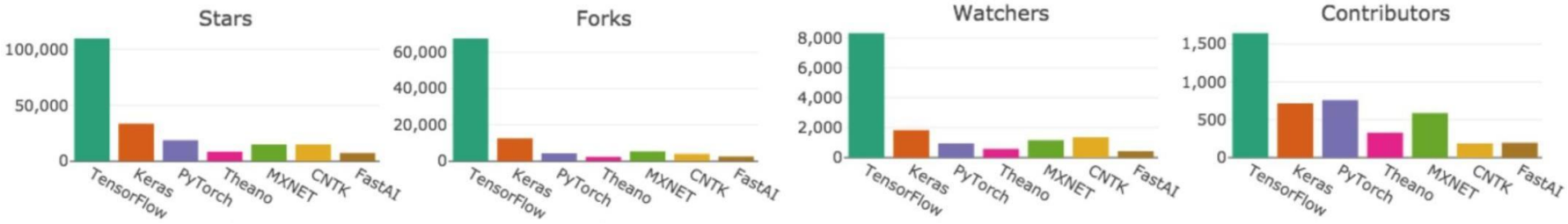
残差:损失函数在某个结点的偏导

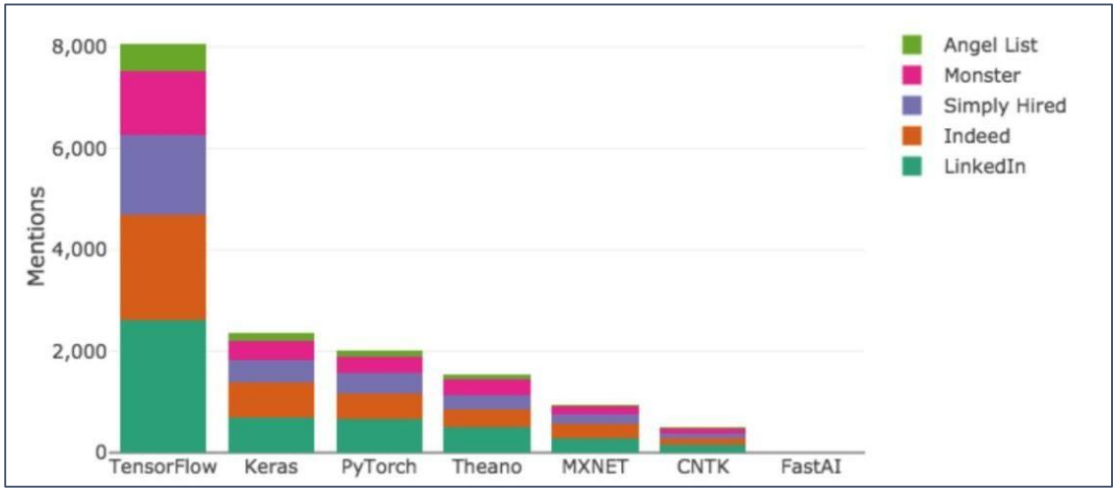


深度学习开发框架

各个大公司开发了一系列的框架，希望开发者能够在他们的平台上做这件事

GitHub





TensorFlow

Keras

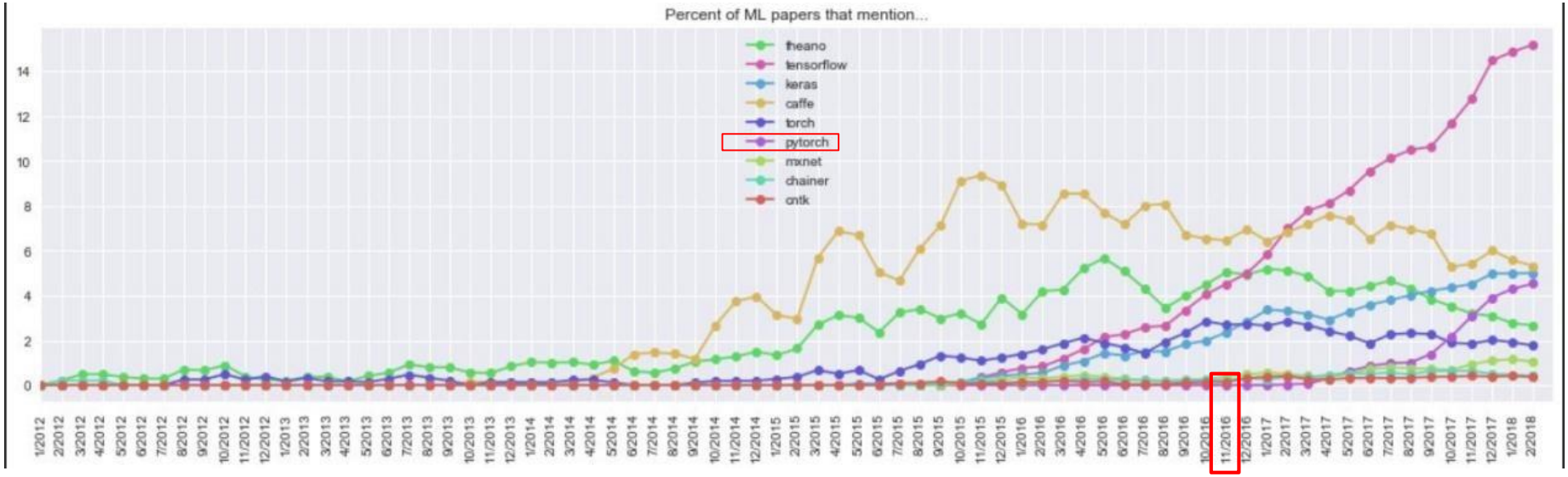
PyTorch

theano

APACHE mxnet

Microsoft CNTK

深度学习开发框架



		Change
1	MicrosoftDocs/azure-docs	4.7x
2	pytorch/pytorch	2.8x
3	godotengine/godot	2.2x
4	nuxt/nuxt.js	2.1x
5	ethereum/go-ethereum	2.0x
6	wix/react-native-navigation	1.9x
7	spyder-ide/spyder	1.8x
8	tensorflow/models	1.8x
9	home-assistant/home-assistant	1.6x
10	MarlinFirmware/Marlin	1.6x

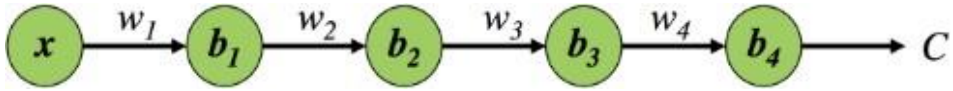
PyTorch 是2017年推出的，设计简洁，代码容易理解

许多新发表的论文在采用它为实现的工具，受到越来越多人的追捧

“Keep it simple, stupid”

深层神经网络的问题：梯度消失

- 前向传播

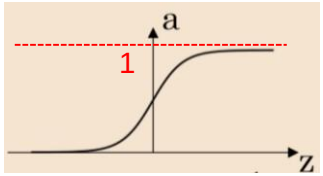


- 反向传播（误差通过梯度传播）

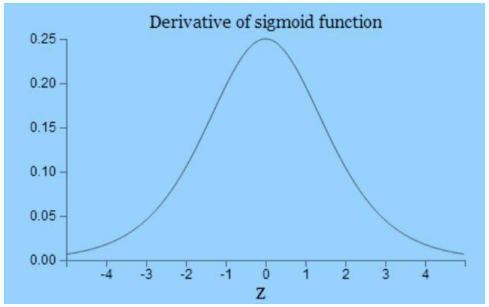
$$\frac{\partial C}{\partial b_1} = \sigma'(b_1) w_2 \sigma'(b_2) w_3 \sigma'(b_3) w_4 \sigma'(b_4) \frac{\partial C}{\partial b_4}$$

- Sigmoid激活函数

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

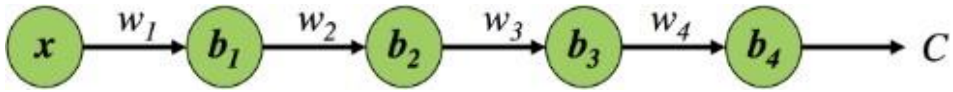


$$\sigma(z)' = \sigma(z)(1-\sigma(z))$$



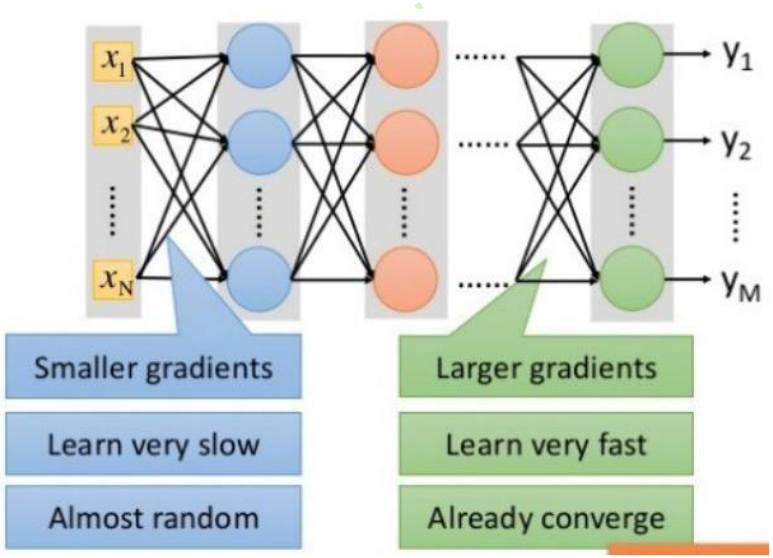
深层神经网络的问题：梯度消失

- 前向传播



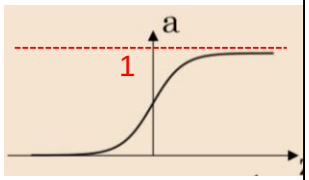
- 反向传播（误差通过梯度传播）

$$\frac{\partial C}{\partial b_1} = \sigma'(b_1) w_2 \sigma'(b_2) w_3 \sigma'(b_3) w_4 \sigma'(b_4) \frac{\partial C}{\partial b_4}$$

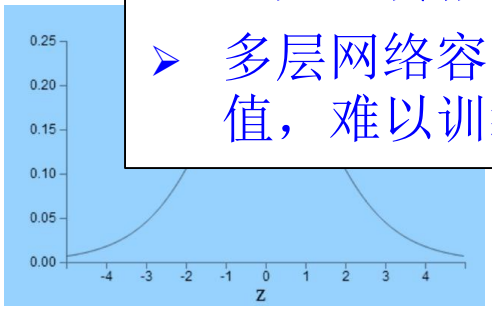


- Sigmoid激活函数

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



$$\sigma(z)' = \sigma(z)(1-\sigma(z))$$



- 增加深度会造成梯度消失(gradient vanishing)，误差无法传播；
- 多层网络容易陷入局部极值，难以训练。

⇨

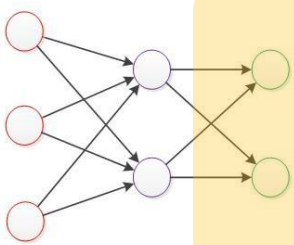
- 三层神经网络是主流
- 预训练、新激活函数使深度成为可能

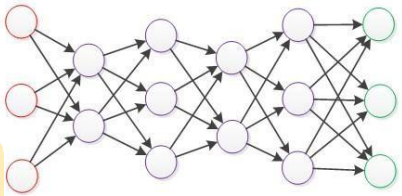
神经网络的“第二次落”

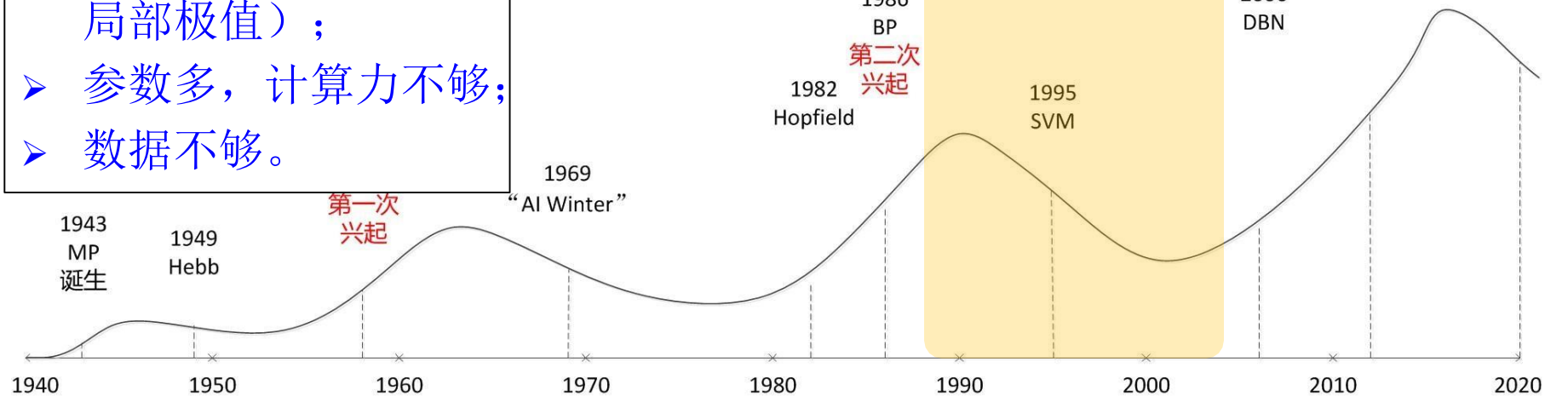
(Vapnik, 1995)

- 全局最优解（凸二次规划）；
- 无需调参；
- 基于支持向量，小样本训练。

- 训练困难（梯度消失、局部极值）；
- 参数多，计算力不够；
- 数据不够。







THANKS