第五周 人工神经网络

李泽榉，复旦大学生物医学工程与技术创新学院

Schedule

Week	Date	Plan
5	10/10/2025	Traditional AI
6	10/17/2025	Artifical Neural Networks
11	11/21/2025	Transformer
13	12/5/2025	Reinforcement Learning
14	12/12/2025	Ethics and Safety
16	12/26/2025	Open Discussion

目录

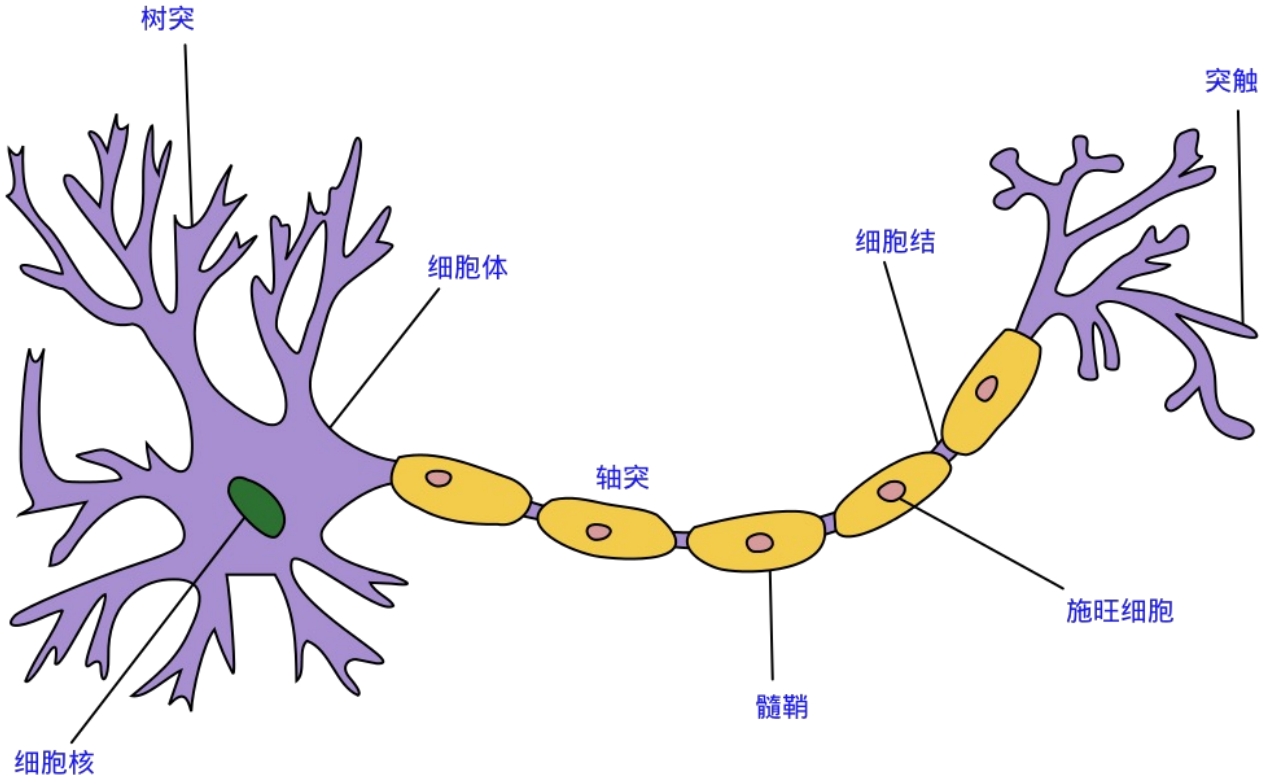
1 发展历史

2 单层神经网络

3 深度神经网络

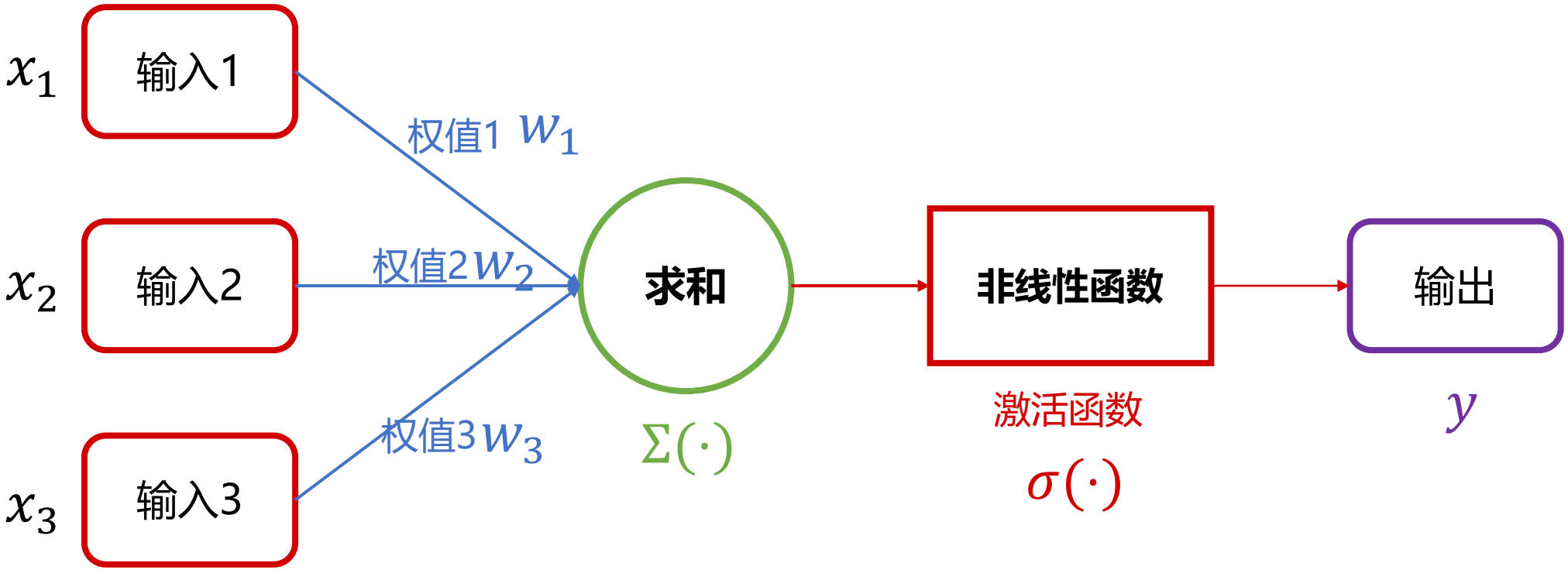
4 反向传播与梯度下降

生物神经元



通常情况下，一个神经元通常具有多个**树突**，主要用来接受传入信息；而**轴突**只有一条，轴突尾端有许多轴突末梢可以给其他多个神经元传递信息。轴突末梢跟其他神经元的树突产生连接，从而传递信号。这个连接的位置在生物学上叫做“**突触**”。

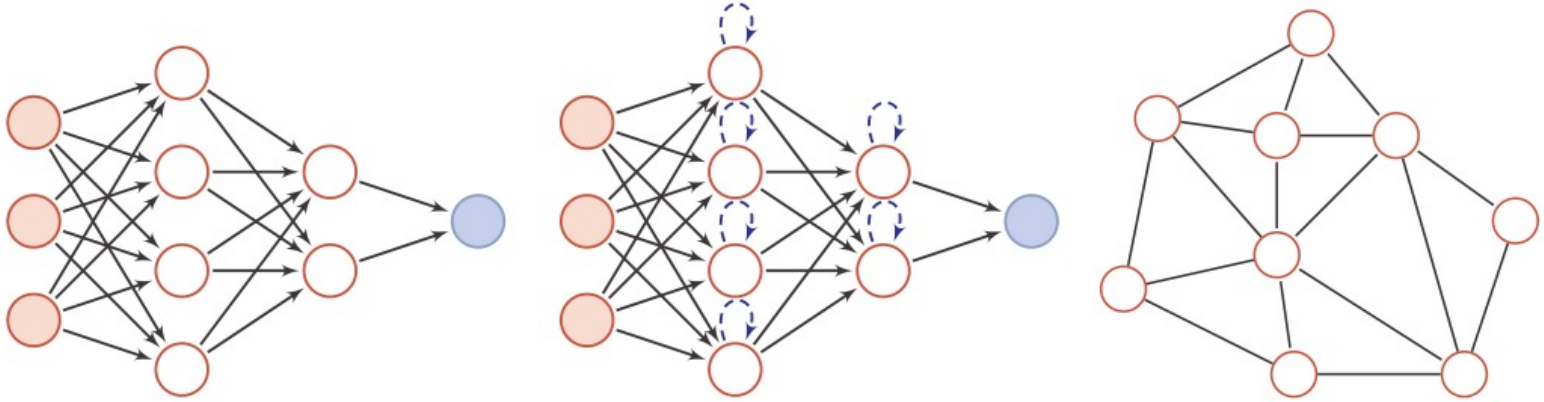
人工神经元



$$y = \sigma\left(\sum_{i=1}^{3} w_i \cdot x_i\right)$$

人工神经网络

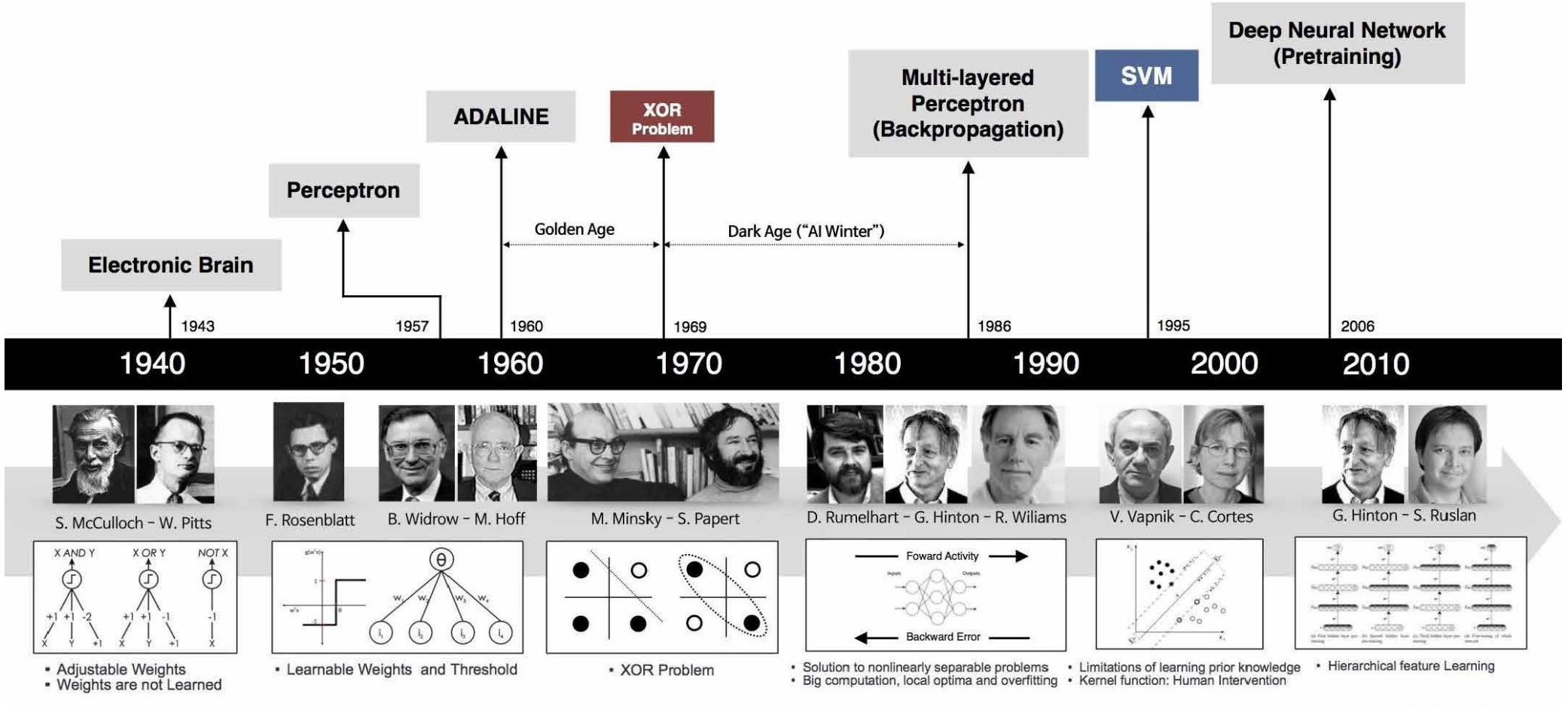
- 人工神经网络由神经元模型构成，这种由许多神经元组成的信息处理网络具有并行分布结构。
  - 虽然这里将神经网络结构大体上分为三种类型，但是大多数网络都是**复合型结构**，即一个神经网络中包括多种网络结构



(a) 前馈网络　(b) 记忆网络　(c) 图网络

神经网络发展史 | 深度学习的崛起

- 神经网络的发展大致经过五个阶段。



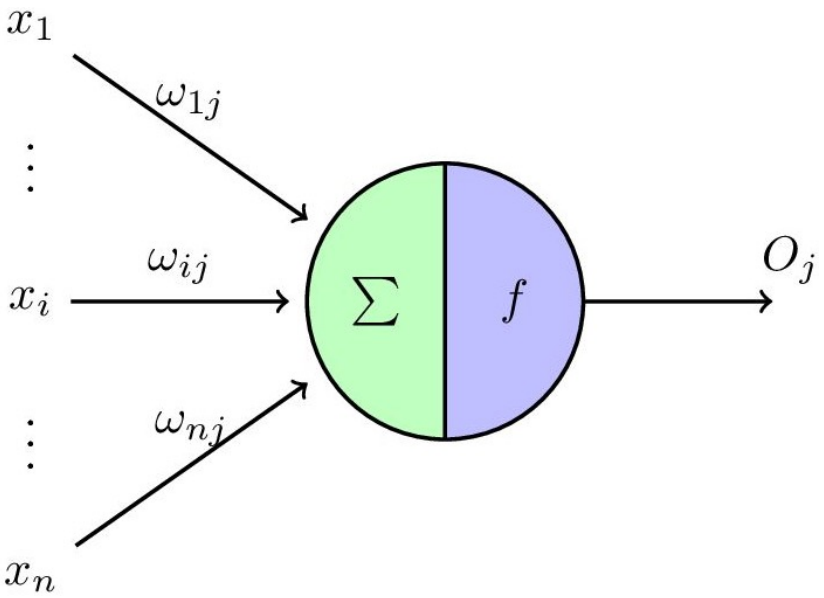
至今

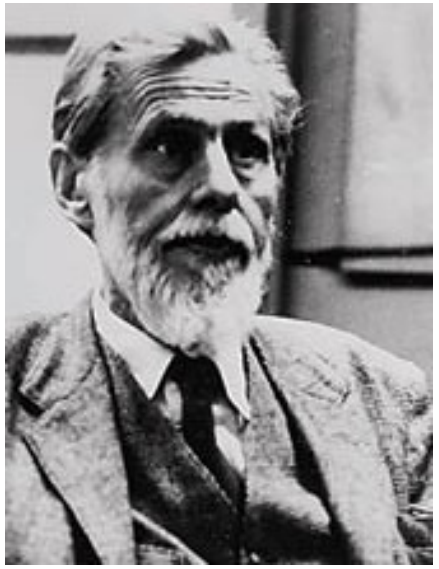
第一阶段 第二阶段 第三阶段 第四阶段 第五阶段

神经网络发展史 | 模型提出

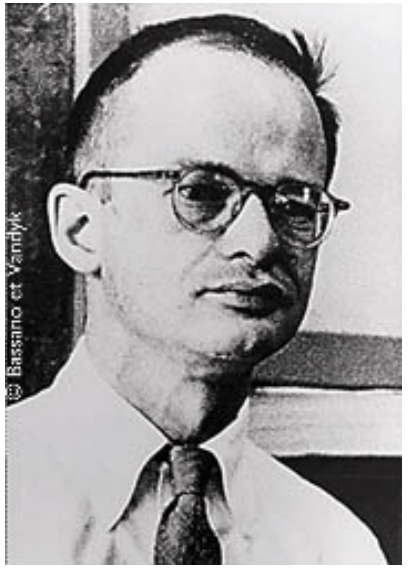
➢第一阶段：**模型提出**

- 在1943年，沃伦·麦卡洛克(Warren **M**cCulloch)和沃尔特·皮茨(Walter **P**itts)最早描述了一种理想化的人工神经网络模型：**M-P**模型。





沃伦·麦卡洛克



沃尔特·皮茨

神经网络发展史 | 模型提出

- 艾伦·图灵在1948年的论文中描述了一种“B型无组织机器”

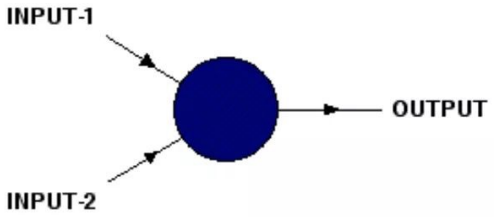
Intelligent Machinery

A. M. Turing
[1912—1954]

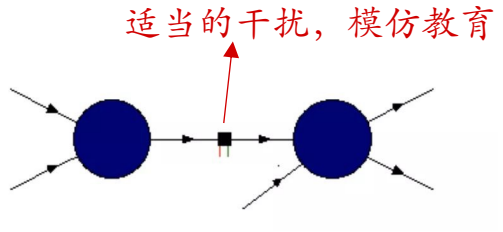
Abstract

The possible ways in which machinery might be made to show intelligent behaviour are discussed. The analogy with the human brain is used as a guiding principle. It is pointed out that the potentialities of the human intelligence can only be realized if suitable education is provided. The investigation mainly centres round an analogous teaching process applied to machines. The idea of an unorganized machine is defined, and it is suggested that the infant human cortex is of this nature. Simple examples of such machines are given, and their education by means of rewards and punishments is discussed. In one case the education process is carried through until the organization is similar to that of an ACE.

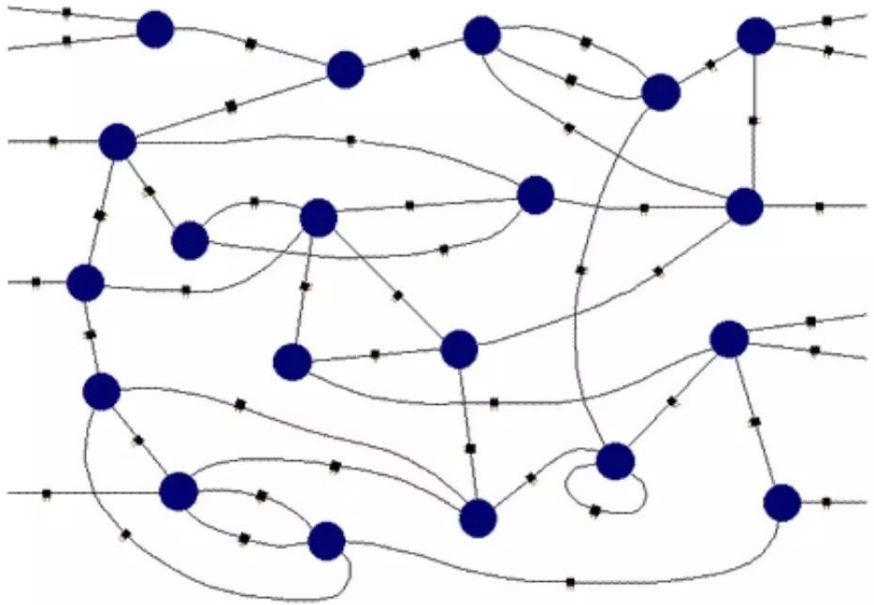
论文《**智能机器**》



单个神经元(与非门)



B 型网络中的两个神经元



大型初始随机 B 型网络的一部分

神经网络发展史 | 模型提出

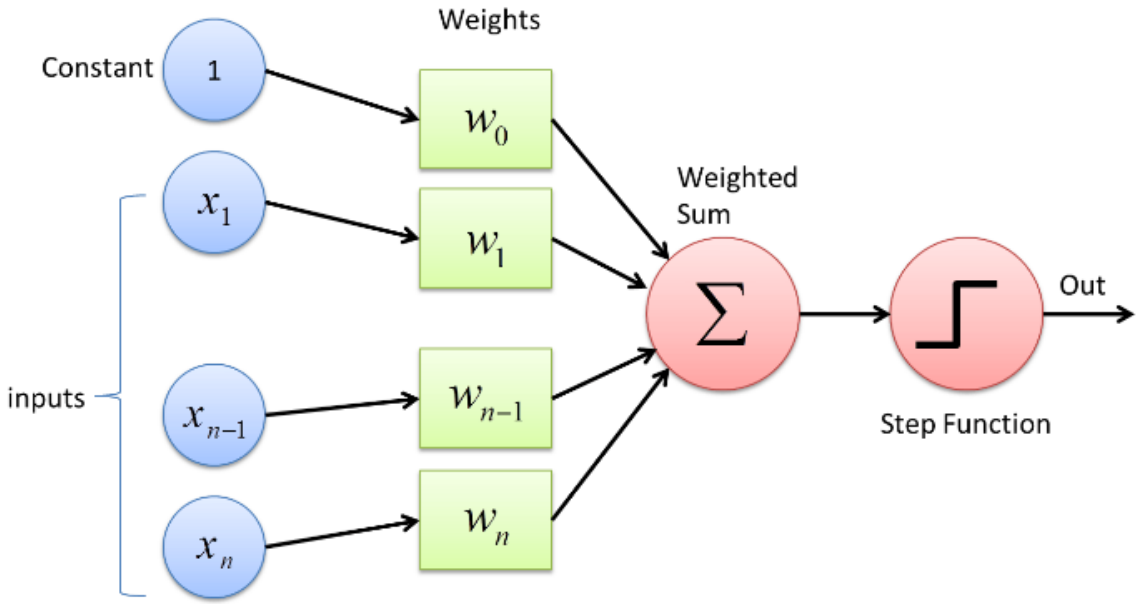
- 1951年，麦卡洛克和皮茨的学生马文·明斯基(Marvin Minsky)建造了第一台神经网络机，称为SNARC

一个SNARC神经元的元件

马文·明斯基

神经网络发展史 | 模型提出

- 1957年，弗兰克·罗森布拉特(Frank Rosenblatt)最早提出可以模拟人类感知能力的神经网络模型：感知器（Perceptron），**并提出了一种接近于人类学习过程（迭代、试错）的学习算法。**

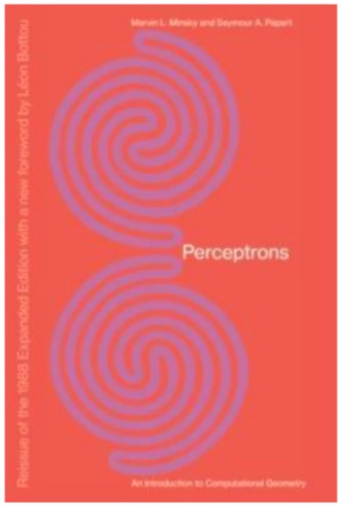


感知器

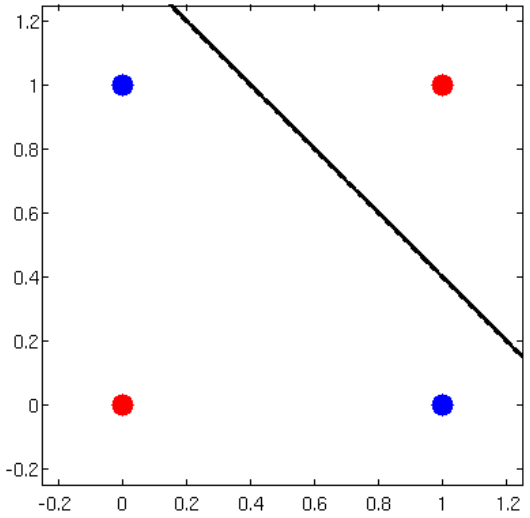
弗兰克·罗森布拉特

神经网络发展史 | 冰河期

- 第二阶段：**冰河期**
  - 1969年，Marvin Minsky出版《感知机》一书，书中论断直接将神经网络打入冷宫，导致神经网络十多年的“冰河期”。他们发现了神经网络的两个关键问题
    - 第一个是基本感知机无法处理异或回路
    - 第二个重要的问题是电脑没有足够的能力来处理大型神经网络所需要的很长的计算时间



《感知机》

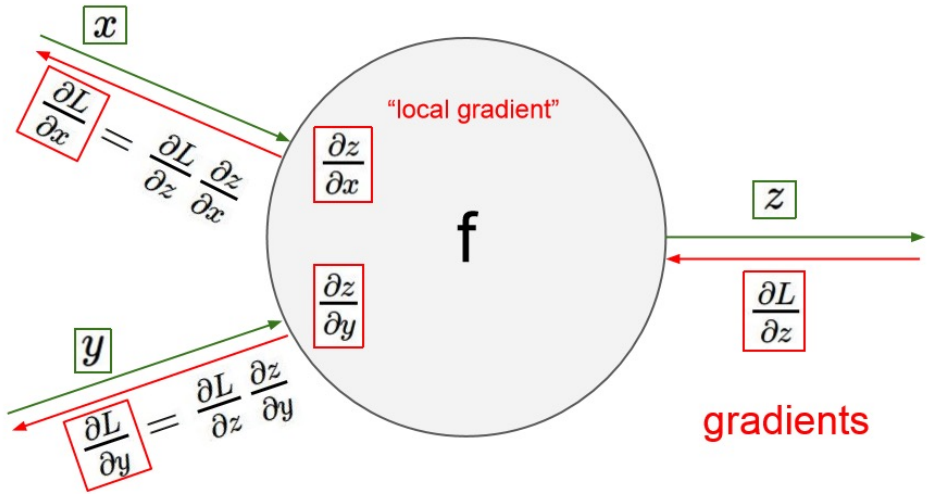


无法解决异或问题

神经网络发展史 | 冰河期

- 1974年，哈佛大学的保罗·沃伯斯(Paul Werbos)发明**反向传播算法**，但当时未受到应有的重视。
- 1980年，福岛邦彦(Kunihiko Fukushima)提出了一种带卷积和子采样操作的多层神经网络：新知机（Neocognitron）。

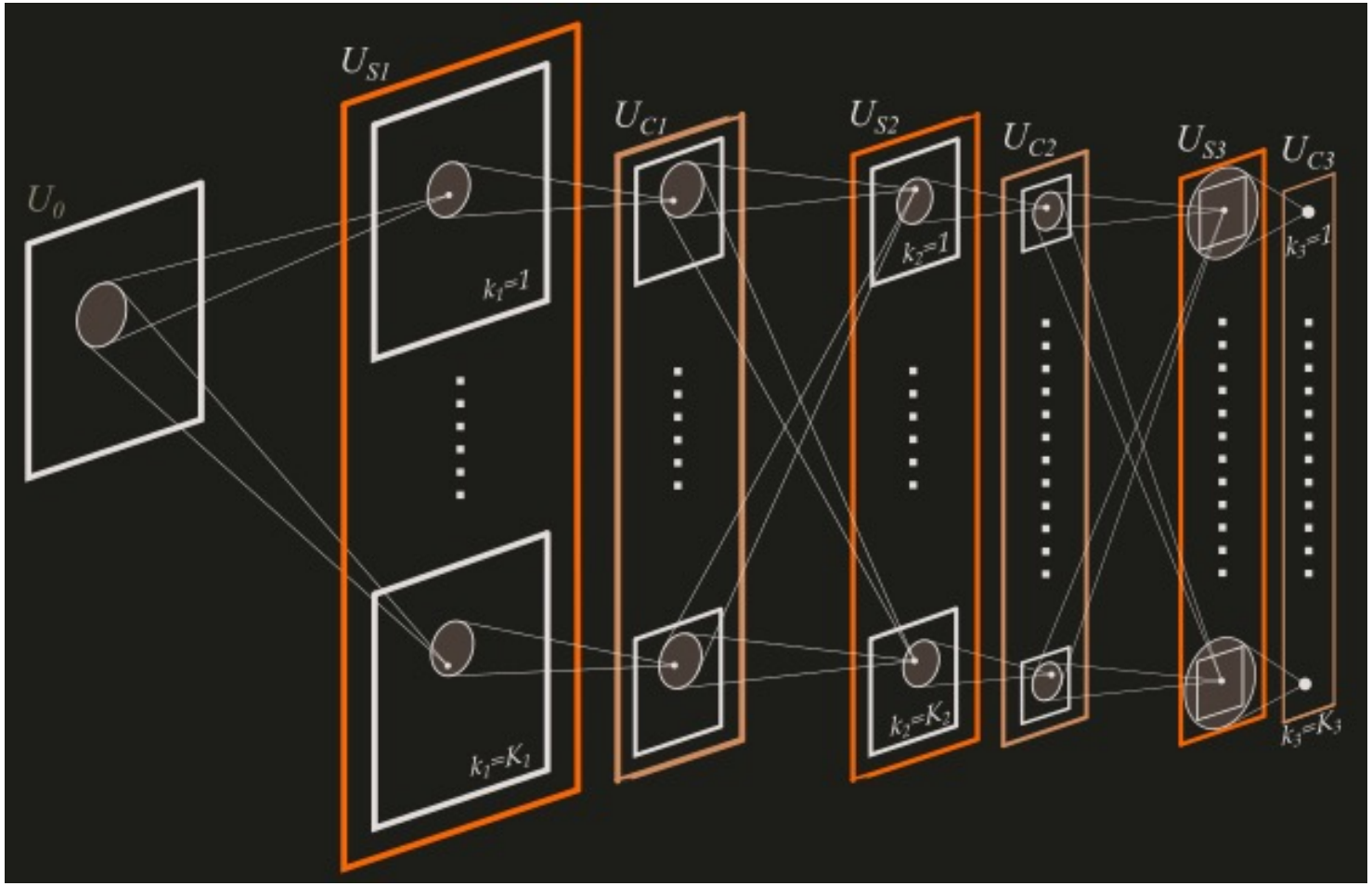
保罗·沃伯斯



反向传播算法

福岛邦彦

神经网络发展史 | 冰河期

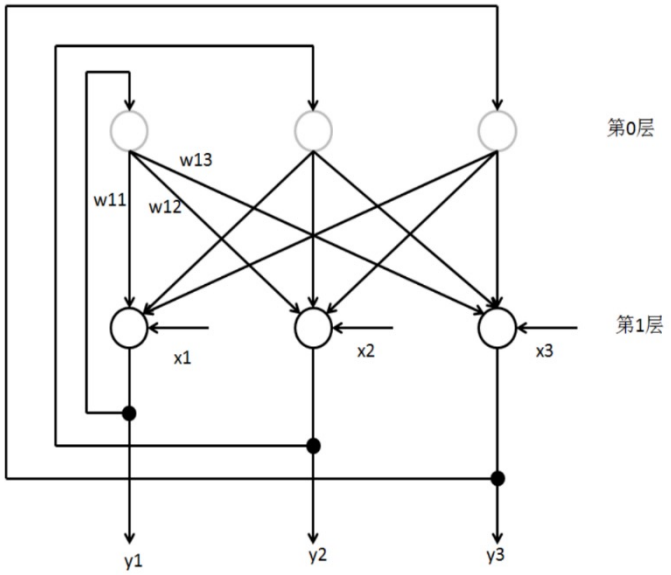


新知机结构　　**Neocognitron**

神经网络发展史 | 反向传播引起的复兴

➢第三阶段：**反向传播算法引起的复兴**

- 1983年，加州理工学院的物理学家约翰·霍普菲尔德(John Hopfield)对神经网络引入能量函数的概念，并提出了用于联想记忆和优化计算的网络（称为Hopfield网络），在旅行商问题上获得当时最好结果，引起轰动

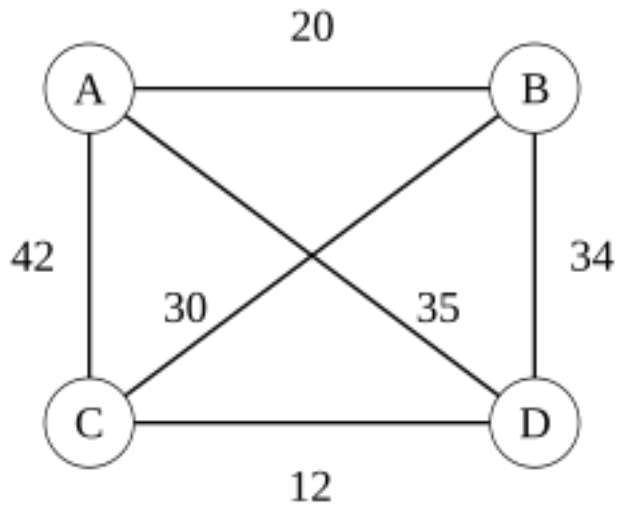


Hopfield网络

约翰·霍普菲尔德

旅行商问题：NP问题

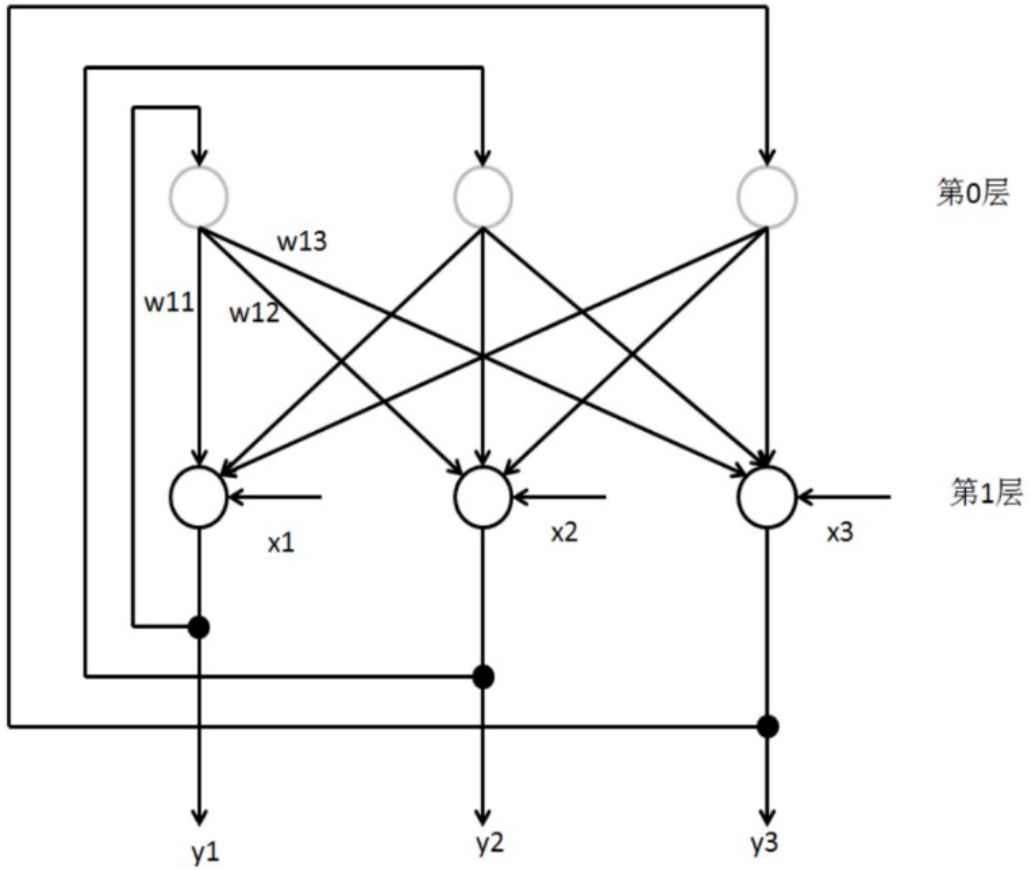
- 旅行商问题：找到一个闭合的环路（从起点出发最后回到起点），访问每个城市且仅访问一次，并且使得这个环路的总距离（或总成本）最小。
  - 4个城市，有 3 条路线。
  - 5个城市，有 12 条路线。
  - 10个城市，就有 181,440 条可能的路线。
  - 20个城市，路线数量就达到了约 60,822,550,204,416,000 条。



- 是经典NP（Nondeterministic Polynomial Time）问题，容易验证，但可能难以求解：
  - 验证：如果我给你一条路线，你可以非常快速地将这条路线上的所有距离加起来，看看是否小于某个值K。这个验证过程是多项式时间的。
  - 求解：但要找到这条最短路线，目前最好的算法需要尝试几乎所有的排列组合，所需时间是随着城市数量n呈阶乘级增长的，非常慢。

旅行商问题：NP问题

- 将路径用神经元的**状态矩阵**来表示
- 用第i层的X_c表示“城市c在第i个顺序被访问”。如果这个神经元激活（输出接近1），就代表这个命题为真。
- 约束项：确保每个城市只被访问一次，每个顺序只能有一个城市。违反约束会导致能量升高。
- 目标项：代表路径的总长度。路径越长，能量越高。



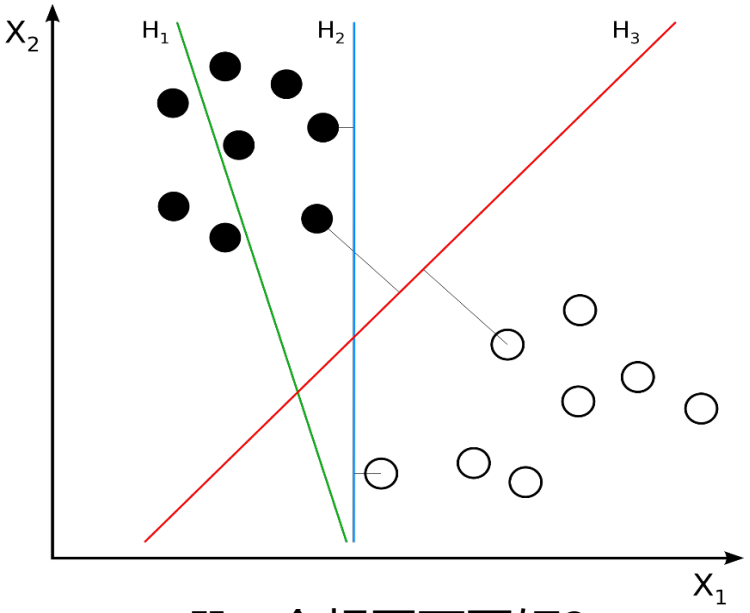
Hopfield网络

神经网络发展史 | 反向传播引起的复兴

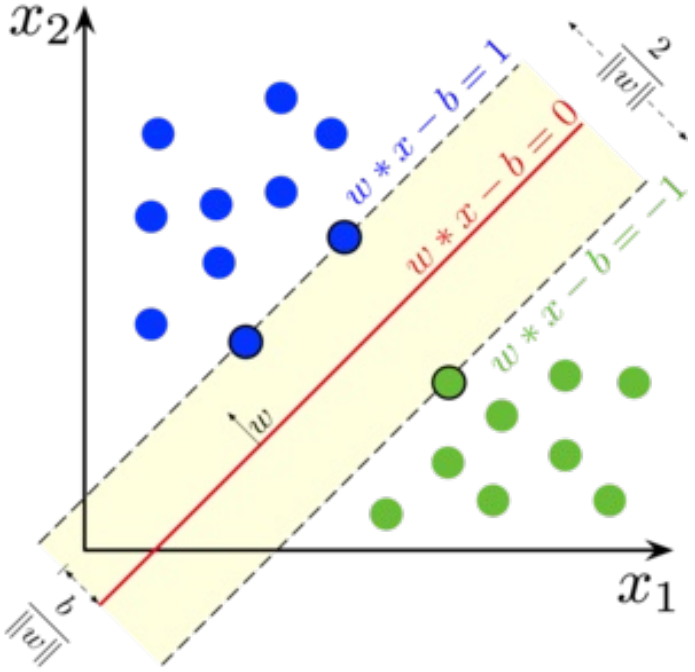
- 1984年，Geoffrey Hinton提出一种随机化版本的Hopfield网络，即玻尔兹曼机。
- 1986年， David Rumelhart和James McClelland对于联结主义在计算机模拟神经活动中的应用提供了全面的论述，并重新发明了反向传播算法。
- Geoffrey Hinton等人将反向传播算法引入到多层感知器
- LeCun等人将反向传播算法引入了卷积神经网络，并在手写体数字识别上取得了很大的成功。

神经网络发展史

- 第四阶段：**流行度降低**
  - 在20世纪90年代中期，统计学习理论和以**支持向量机（support vector machines，SVM）**为代表的机器学习模型开始兴起。
  - 相比之下，神经网络的理论基础不清晰、优化困难、可解释性差等缺点更加凸显，神经网络的研究又一次陷入低潮。



哪一个超平面更好?

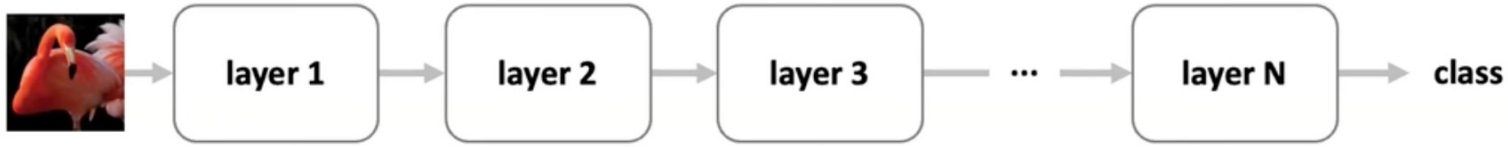


神经网络发展史 | 深度学习的崛起

➢ 第五阶段：**深度学习的崛起**

- 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。

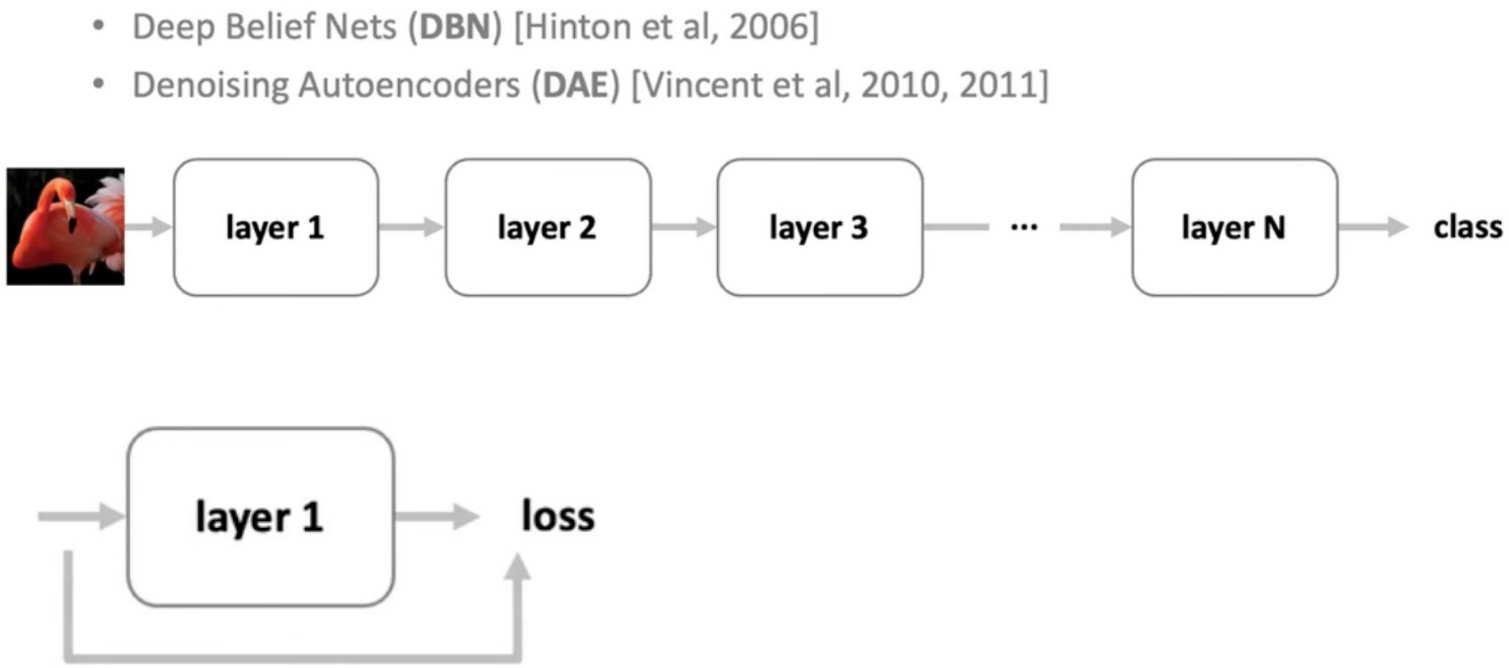
- Deep Belief Nets (**DBN**) [Hinton et al, 2006]
- Denoising Autoencoders (**DAE**) [Vincent et al, 2010, 2011]



神经网络发展史 | 深度学习的崛起

➢ 第五阶段：**深度学习的崛起**

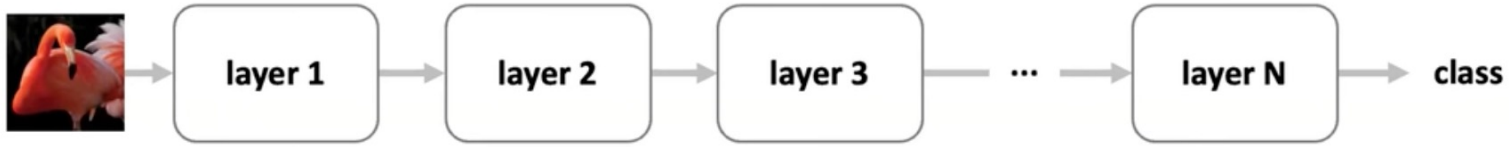
- 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。

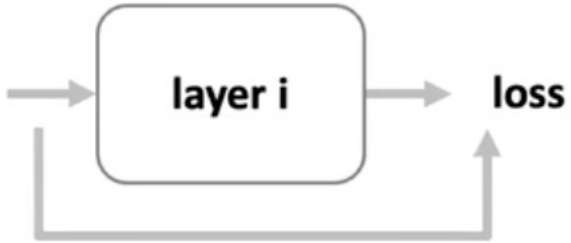


神经网络发展史 | 深度学习的崛起

- 第五阶段：**深度学习的崛起**
  - 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。

- Deep Belief Nets (**DBN**) [Hinton et al, 2006]
- Denoising Autoencoders (**DAE**) [Vincent et al, 2010, 2011]

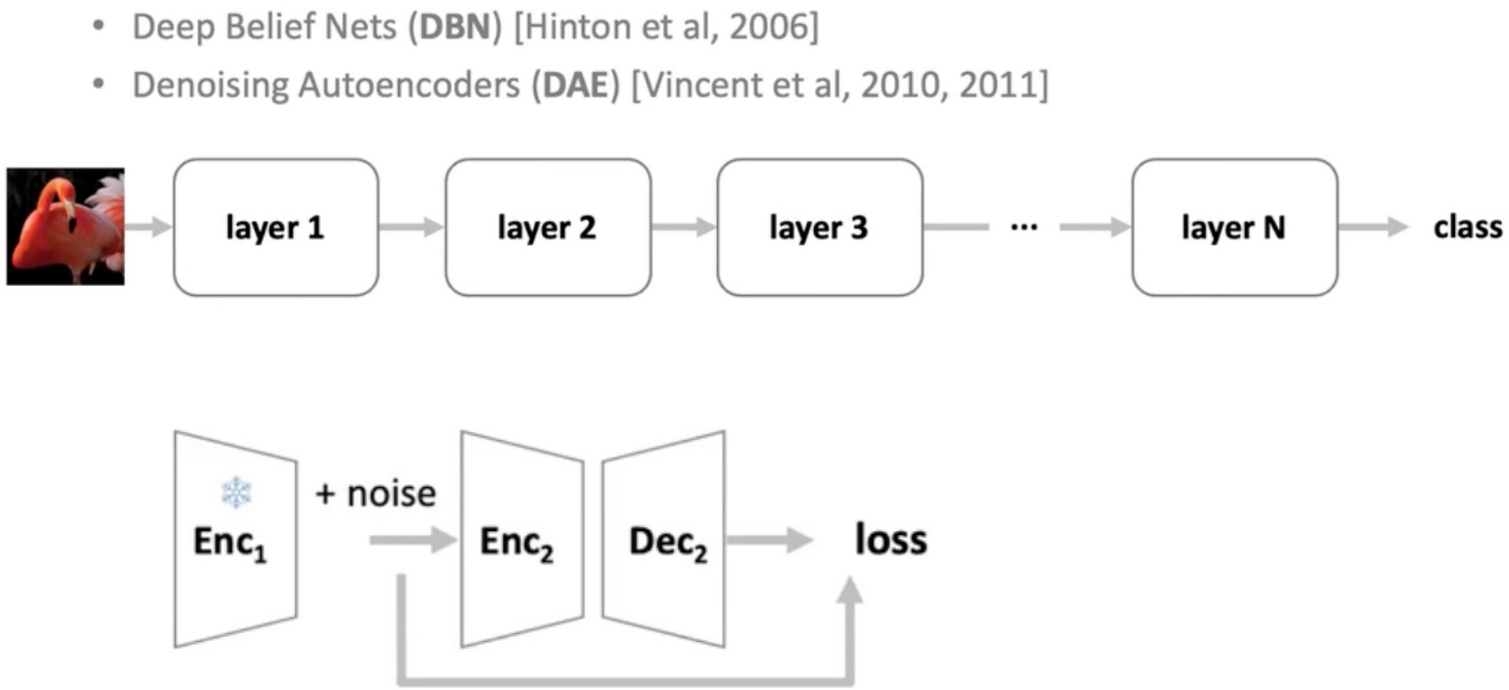




神经网络发展史 | 深度学习的崛起

➢ 第五阶段：**深度学习的崛起**

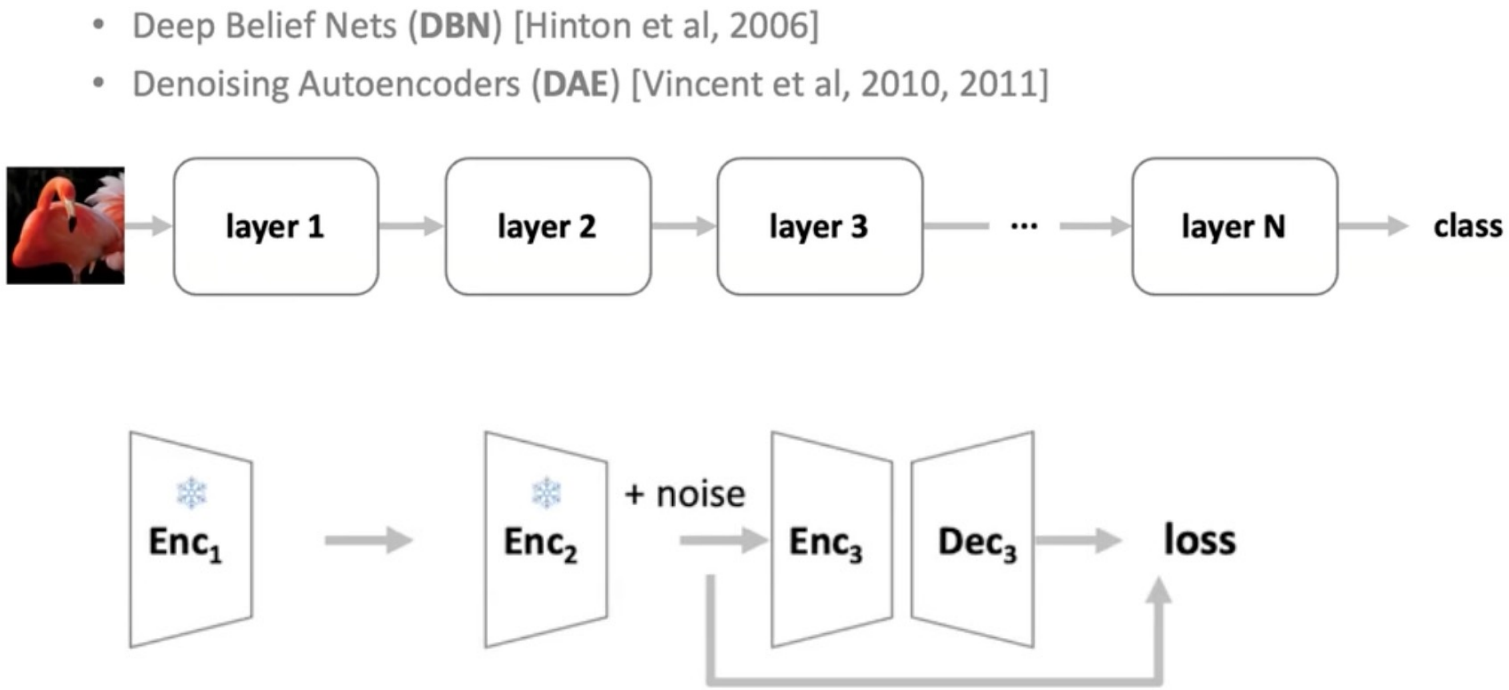
- 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。



神经网络发展史 | 深度学习的崛起

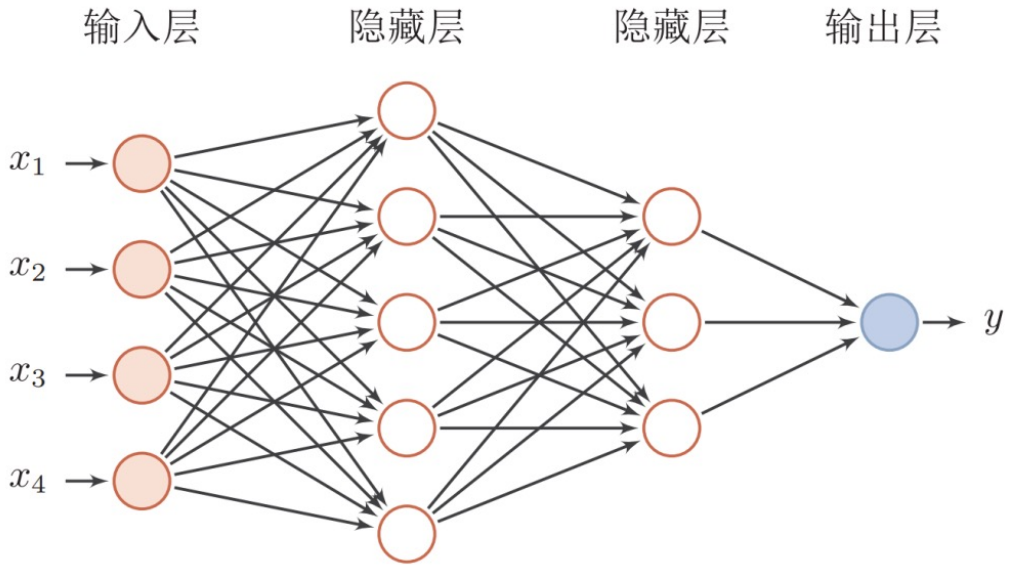
➢ 第五阶段：**深度学习的崛起**

- 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。



神经网络发展史 | 深度学习的崛起

- 第五阶段：**深度学习的崛起**
  - 2006年，Hinton等人发现多层前馈神经网络可以先通过**逐层预训练**，再用反向传播算法进行精调的方式进行有效学习。
    - 深度神经网络在语音识别和图像分类等任务上的巨大成功。



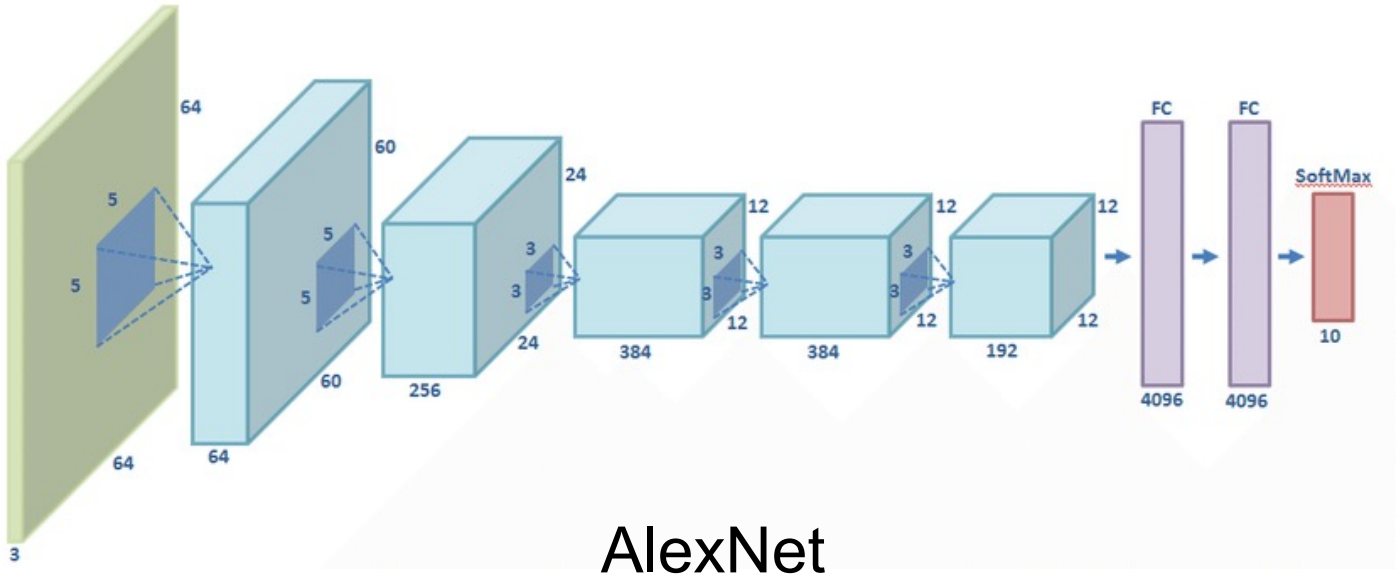
多层前馈神经网络/多层感知器

深度神经网络

“狗”

神经网络发展史 | 深度学习的崛起

- 2012年，AlexNet：第一个现代深度卷积网络模型，是深度学习技术在图像分类上取得真正突破的开端。在ImageNet挑战赛上一举成名
- **算力（GPU）与数据规模的提升**使得训练大规模人工神经网络成为可能



AlexNet

ImageNet

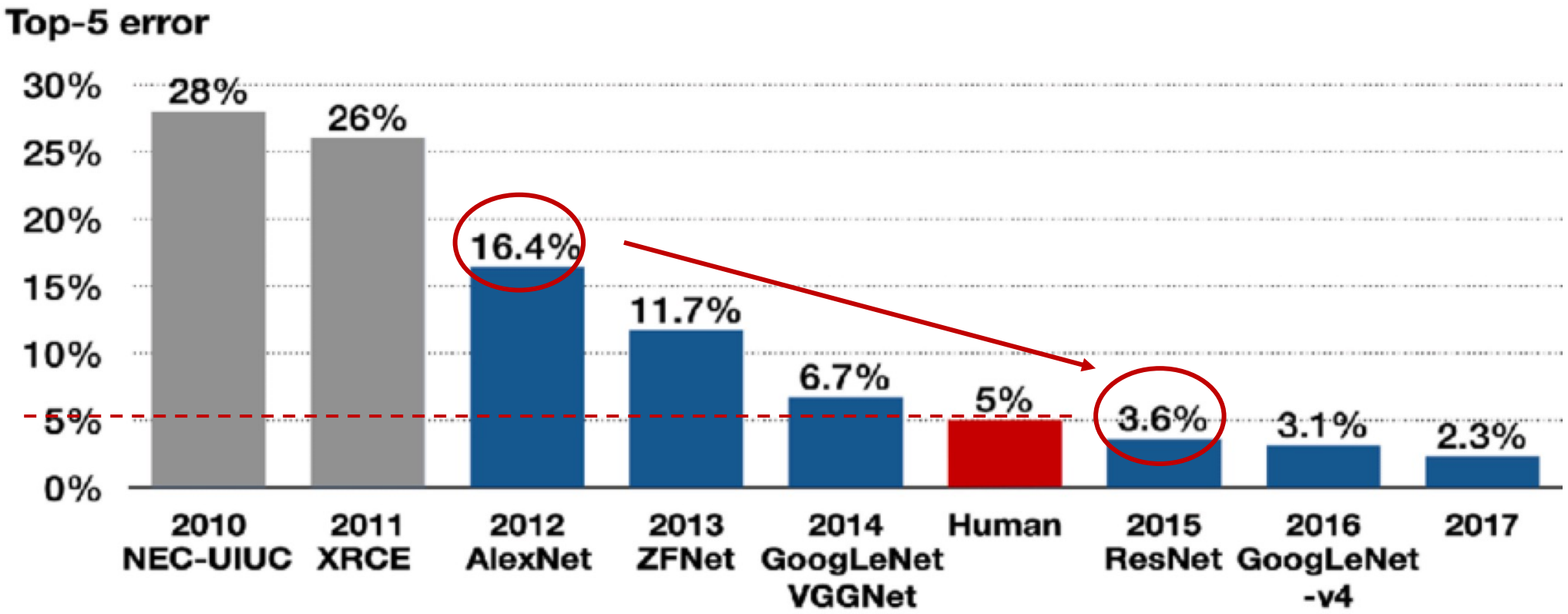
李飞飞 | 数据信仰

李飞飞，1976年出生于中国北京，美国国家工程院院士、美国国家医学院院士、美国艺术与科学院院士，美国斯坦福大学首位红杉讲席教授，以人为本人工智能研究院（HAI）院长，AI4ALL联合创始人及主席，Twitter公司董事会独立董事。

2006年，李飞飞发现整个学术圈和人工智能行业都在苦心研究同一个概念：通过更好的算法来制定决策，但却并不关心数据。她认为：**如果使用的数据无法反映真实世界的状况，即便是最好的算法也无济于事。**

她的解决方案是构建一个更好的数据集！

深度学习模型在图像识别任务上超越人类



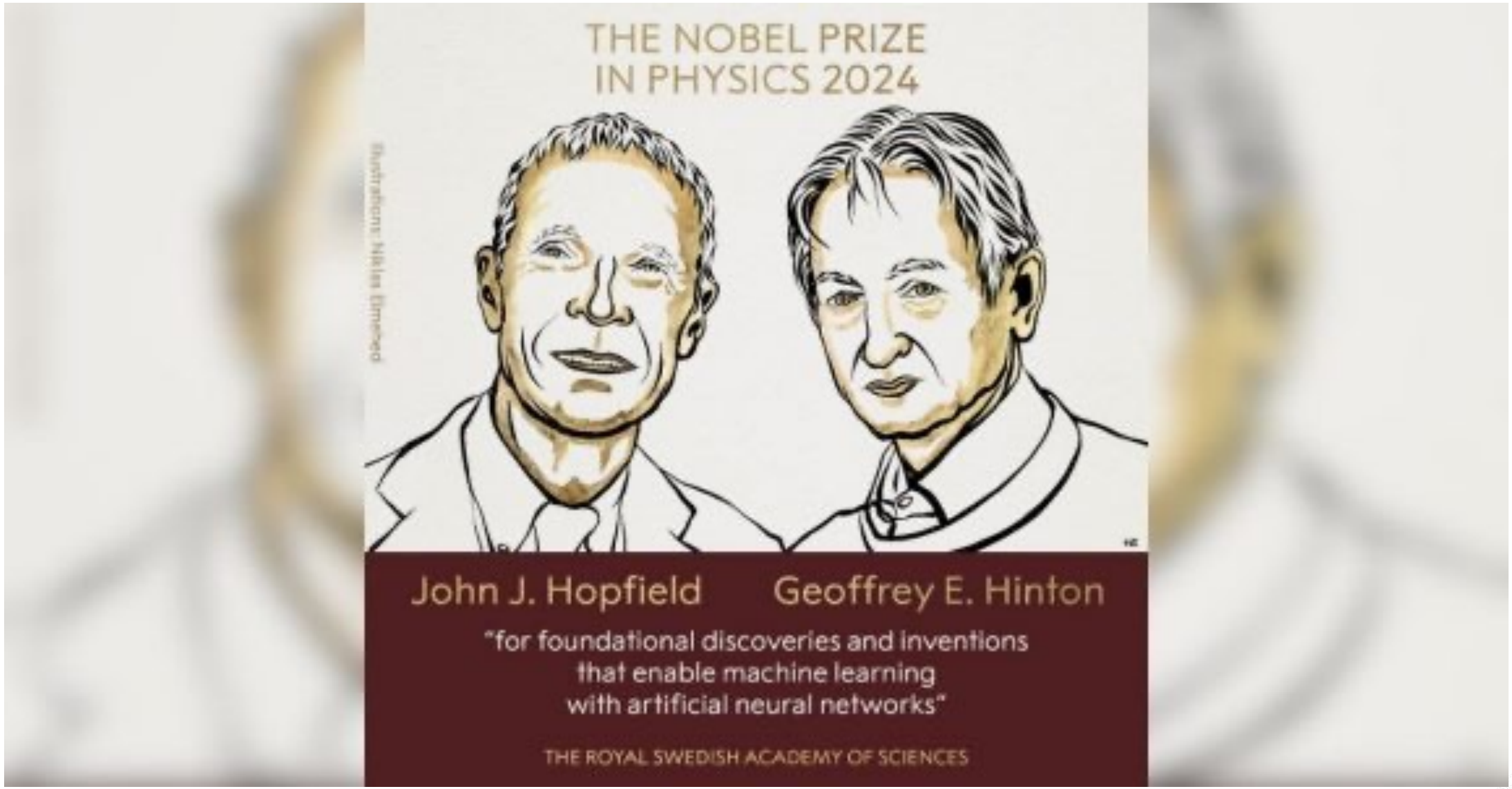
不同模型在ImageNet图像识别挑战赛任务上的错误率

三位图灵奖得主

2019年3月27日，ACM（国际计算机学会）宣布，三位“深度学习之父”杨立昆(Yann LeCun)、杰弗里·辛顿(Geoffrey Hinton)和约书亚·本吉奥(Yoshua Bengio) 共同获得了2018年图灵奖。

Yann LeCun　Geoffrey Hinton　Yoshua Bengio

2024年诺贝尔奖


THE NOBEL PRIZE
IN PHYSICS 2024
John J. Hopfield
Geoffrey E. Hinton
"for foundational discoveries and inventions
that enable machine learning
with artificial neural networks"
THE ROYAL SWEDISH ACADEMY OF SCIENCES

目录

1 人工神经网络概述

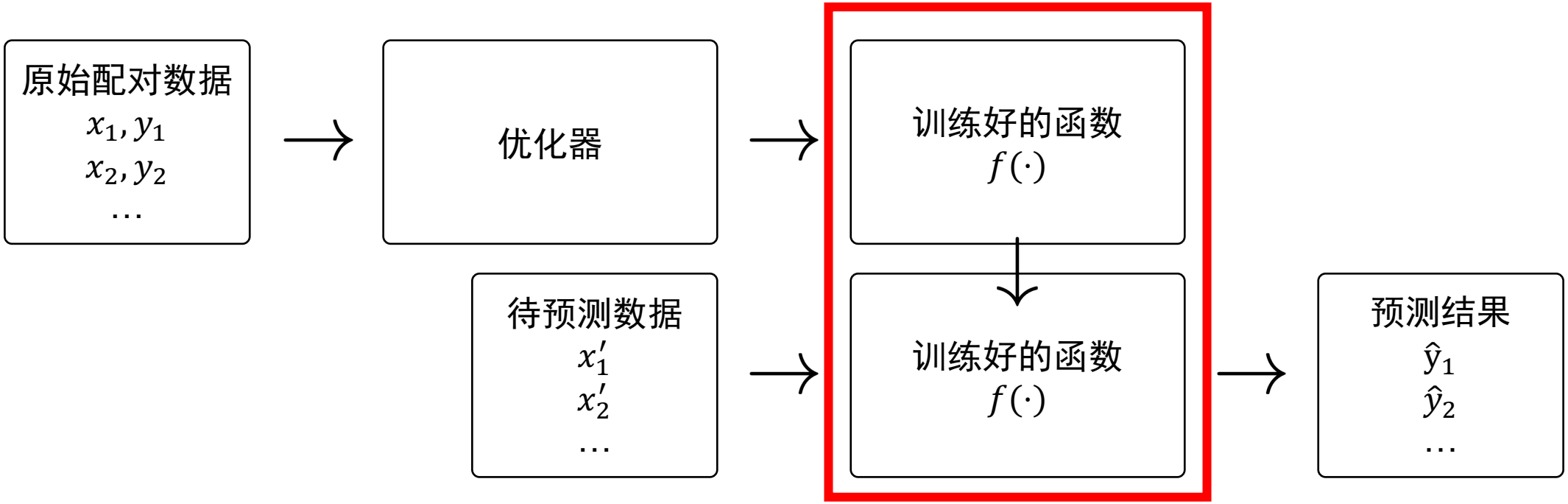
2 单层神经网络

3 深度神经网络

4 反向传播与梯度下降

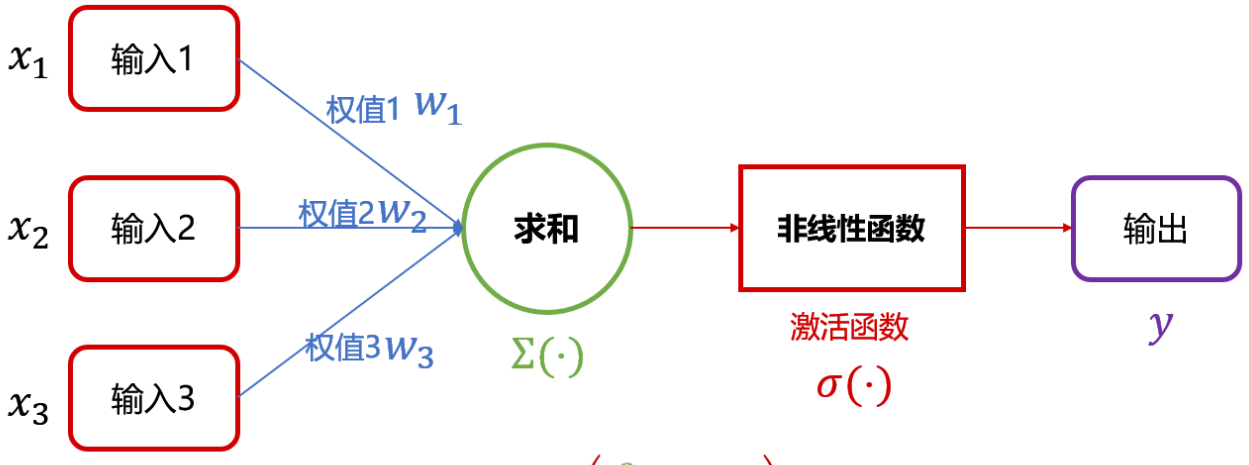
深度学习的核心问题

- **模型：需要构建出表达能力足够丰富的函数$f(x)$**
- 优化目标：需要有能够评价$f(x)$的输出能力的函数$L(w)$
- 优化器：需要有能够根据$L(w)$来调整$f(x)$的通用方法



单层神经网络

- 神经元模型是人工神经网络的基本成分，它从神经科学中获得灵感，但工作方式与大脑神经元并不相同
- 神经元模型对输入进行简单的加权求和，再通过非线性激活函数得到神经元的输出.
- 神经元模型也被称为感知器

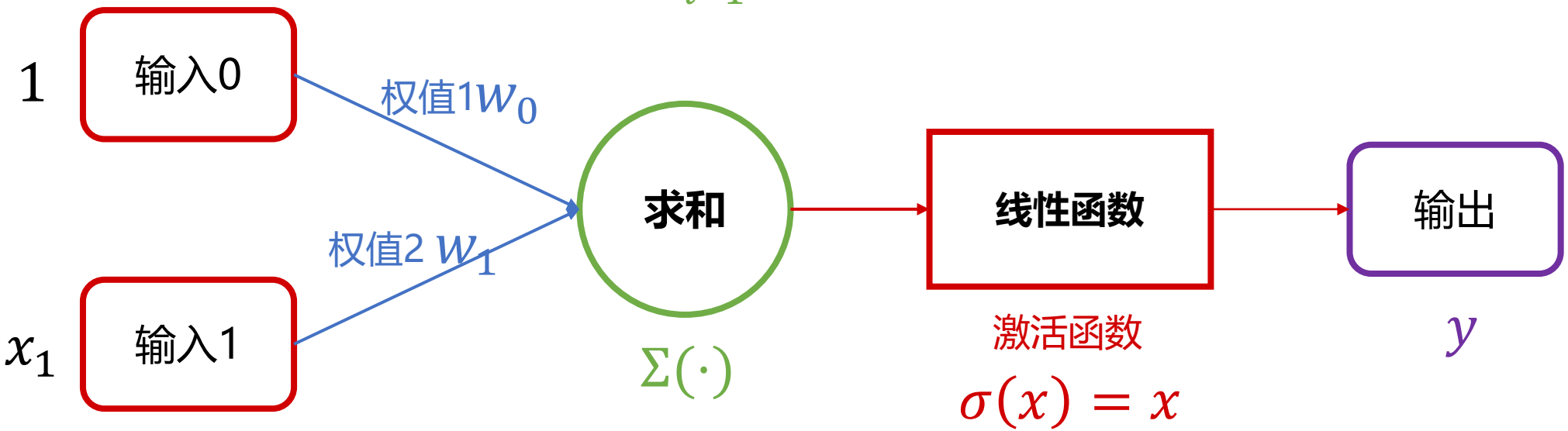


$$y = \sigma\left(\sum_{i=1}^{3} w_i \cdot x_i\right)$$

单层神经网络的特例

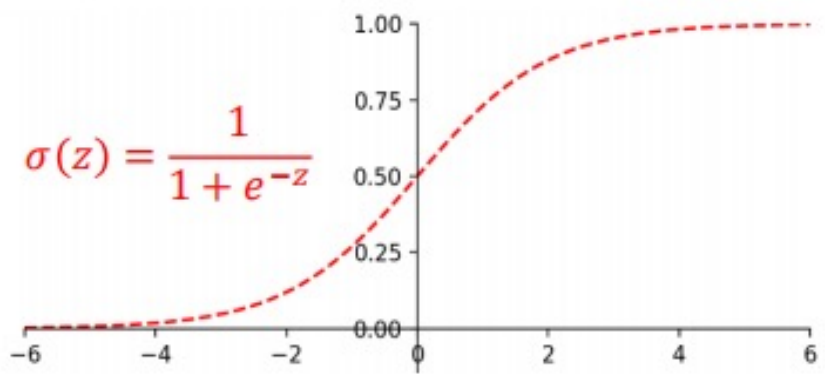
- 假设我们有许多配对好的数据$\{x, y\}$
- 我们希望预测新的一个$\{x', \hat{y}\}$
- 可以构建一个单个神经元来完成这个预测任务

$$y = \sigma\left(\sum_{i=1}^{3} w_i \cdot x_i\right) = w_1 x_1 + w_0$$



单层神经网络

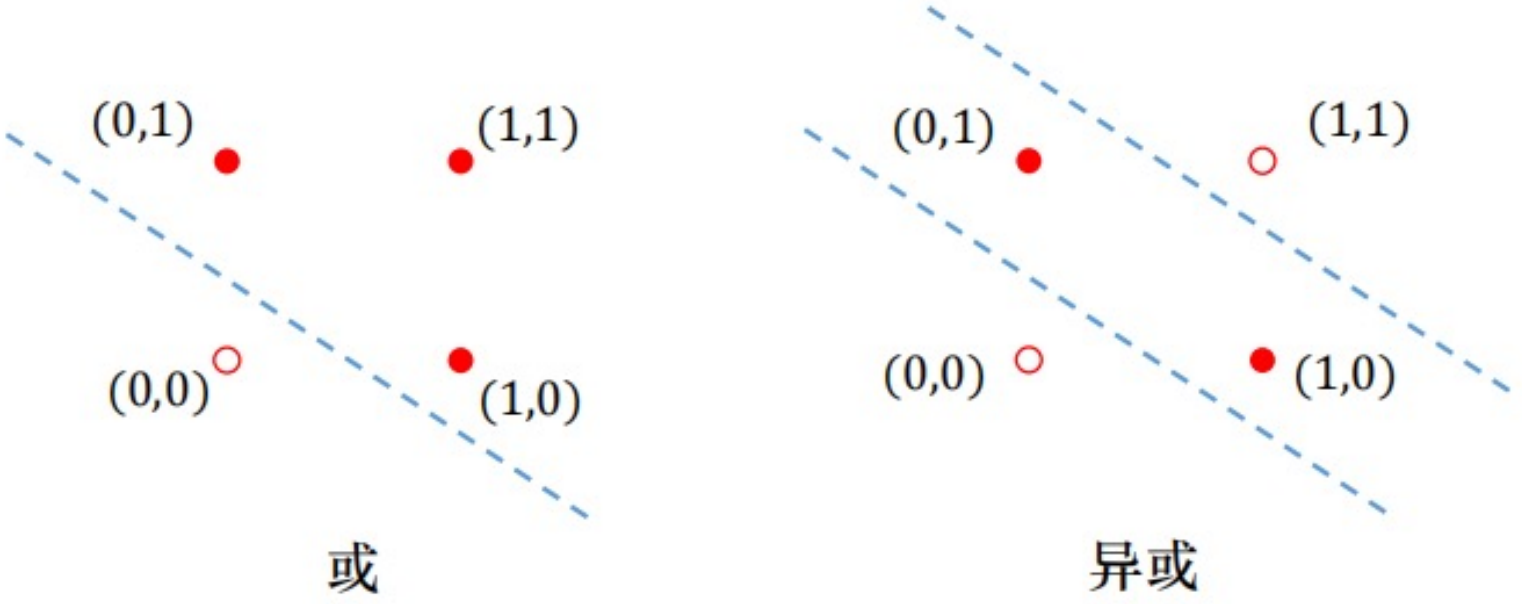
- 用感知器可以实现一些简单的运算
  - 令$\sigma(z) = (1 + e^{-z})^{-1}$，$w_0 = b = -10$，$w_1 = 20$，$w_2 = 20$　$z = w^T x = [w_0\ w_1\ w_2][1\ x_1\ x_2]^{\mathrm{T}}$



$x_1 \setminus x_2$	0	1
0	≈ 0	≈ 1
1	≈ 1	≈ 1

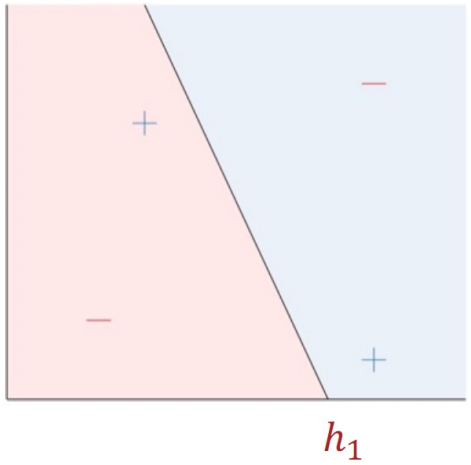
单层神经网络

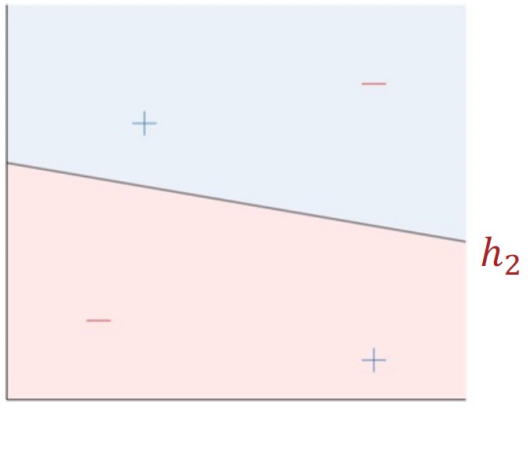
- 单层感知器的局限性
  - 只能处理线性可分的问题（如或运算）
  - 1969年，Marvin Minsky和Seymour Papert 发表《Perceptrons: An introduction to computational geometry》一书，从数学的角度证明了单层感知器无法处理简单的异或问题
  - 要处理线性不可分的问题，人们引入了多层感知器

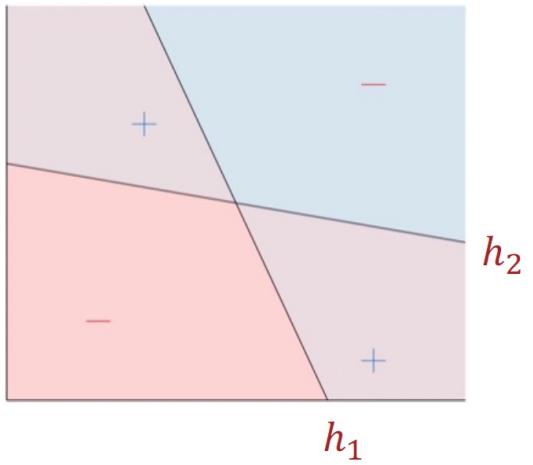


单层神经网络

- 多个线性模型







$$h(\boldsymbol{x}) = \begin{cases} +1 \text{ if } (h_1(\boldsymbol{x}) = +1 \text{ and } h_2(\boldsymbol{x}) = -1) \text{ or } (h_1(\boldsymbol{x}) = -1 \text{ and } h_2(\boldsymbol{x}) = +1) \\ -1 \text{ otherwise} \end{cases}$$

目录

1 人工神经网络概述

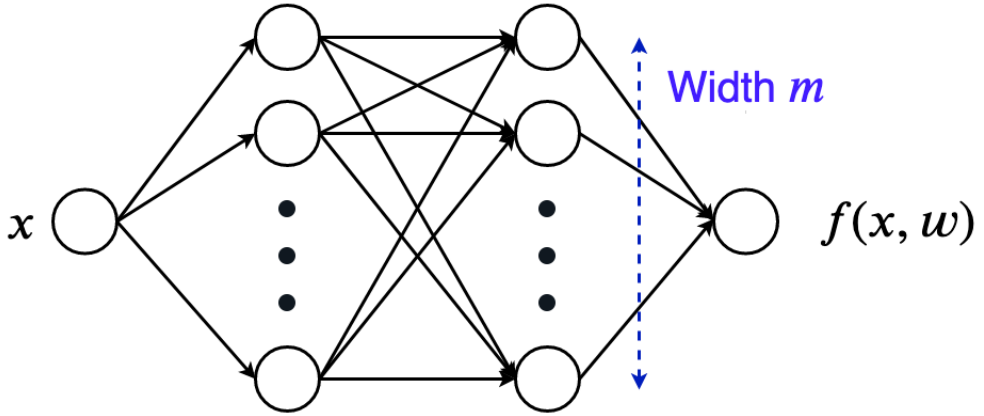
2 单层神经网络

3 深度神经网络

4 反向传播与梯度下降

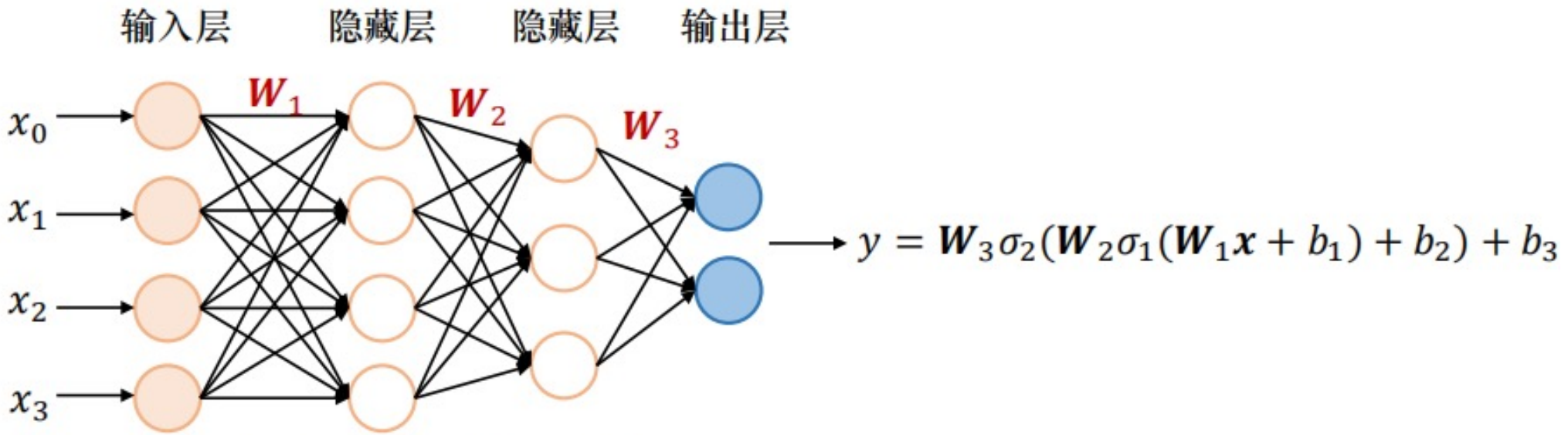
多层感知器

- 为什么要多层?
- 在相当宽泛的条件下，一个无穷宽的单层神经网络可以成为通用函数逼近器
- 在实践中，由于我们无法拥有无限的宽度，我们通过增加深度来以一种更参数高效、泛化能力更强的方式，去逼近甚至超越无限宽网络的理论能力。



多层感知器

- 多层感知器
  - 结构：输入层、隐藏层、输出层
  - 相邻两层中所有的“神经元”两两相连，也叫“全连接”
  - 数据沿着输入层单向传播到输出层



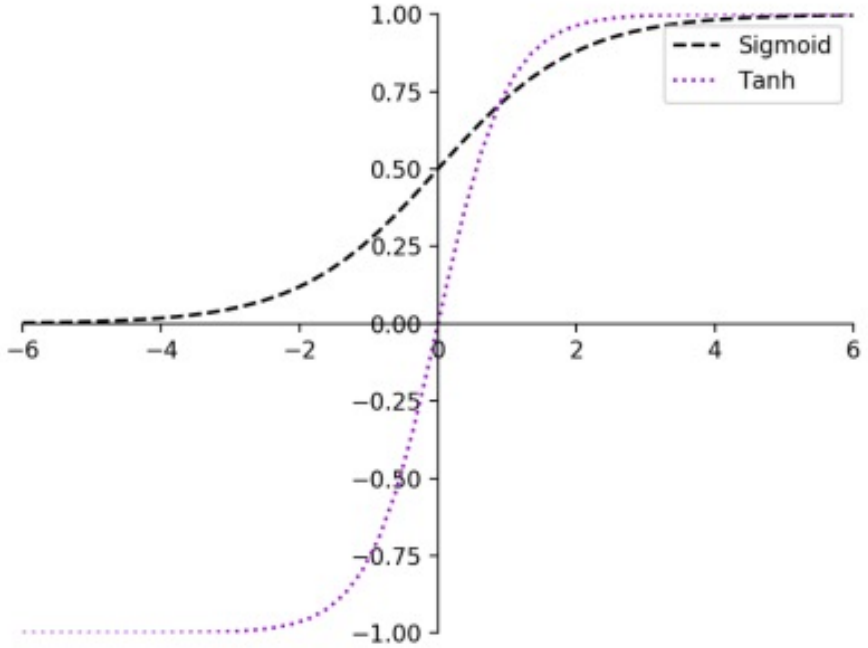
含有两层隐含层的多层感知机

激活函数

- 如果没有激活函数，多层感知机的堆叠和单层感知机等价
- 常见的激活函数：Sigmoid函数、双曲正切(tanh)函数
  - 优点：处处连续、处处可导
  - 缺点：在正负无穷处，它们的导数都趋近于0 → 梯度消失

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

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

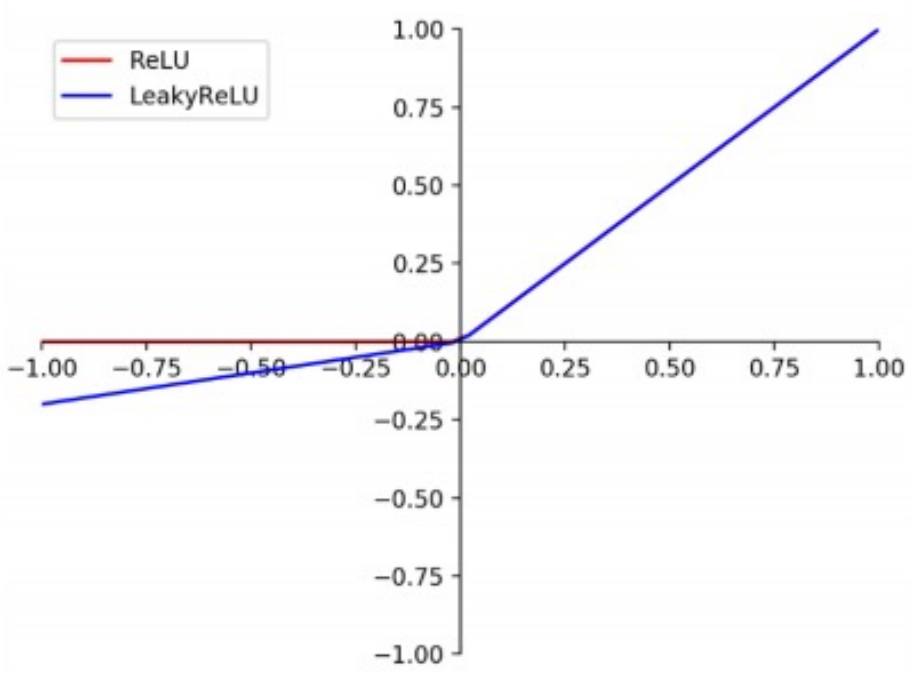


激活函数

- 常见的激活函数：ReLU函数、LeakyReLU函数
  - 优点：分段线性，计算量小，易于优化
  - 缺点：需要处理不连续点的导数
  - LeakyReLU可以解决ReLU函数在负半轴区域的“硬饱和”问题

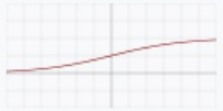
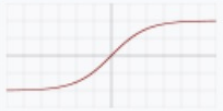
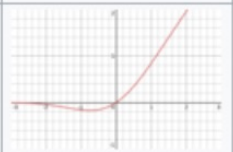
$$\mathrm{ReLU}(z) = \max(0, z)$$

$$\mathrm{LeakyReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \lambda z & \text{if } z \leq 0 \end{cases}$$



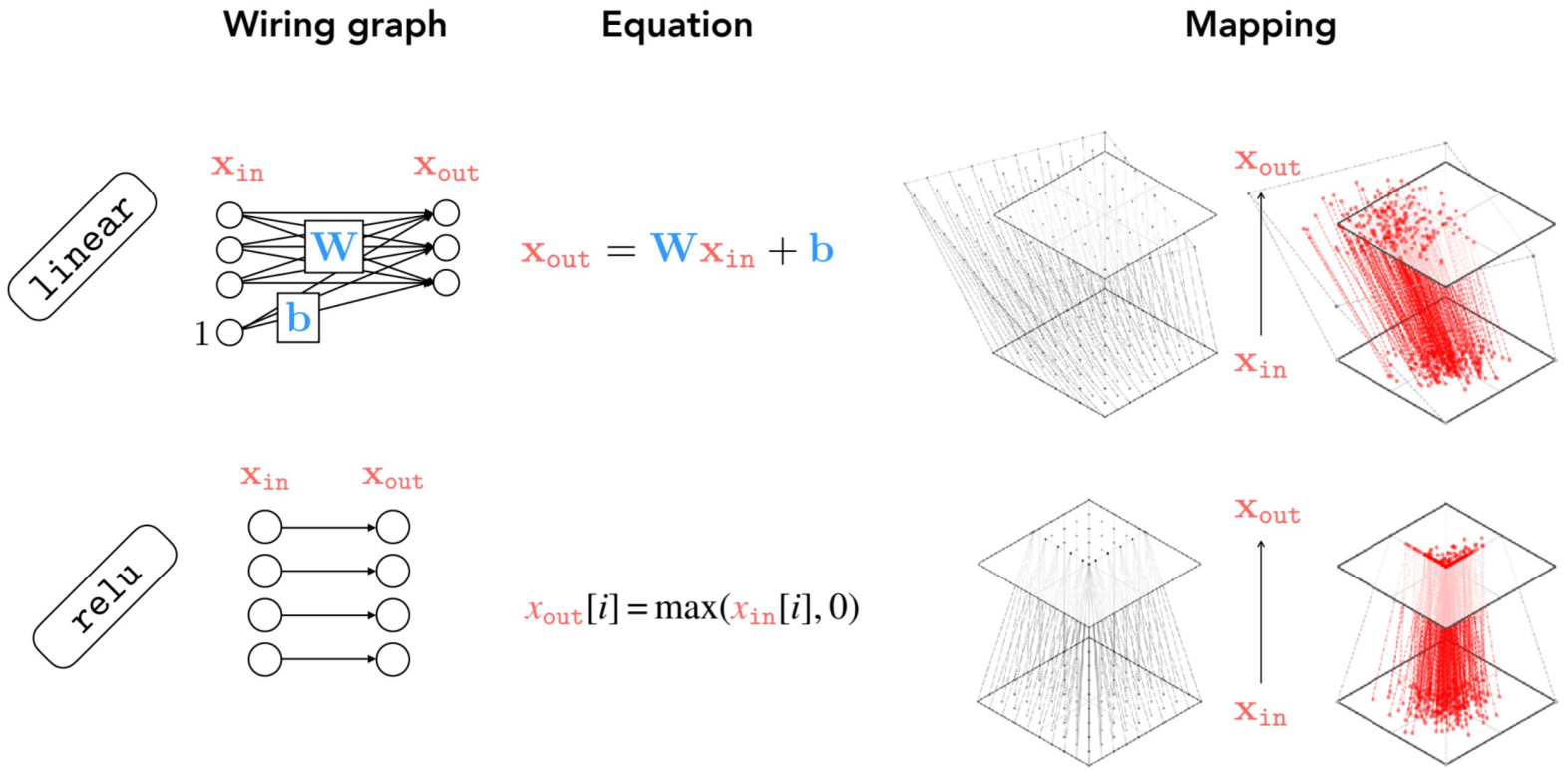
激活函数

- 其他激活函数

Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1+\operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[8]		$\ln(1+e^{x})$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^{x}-1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parametric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α

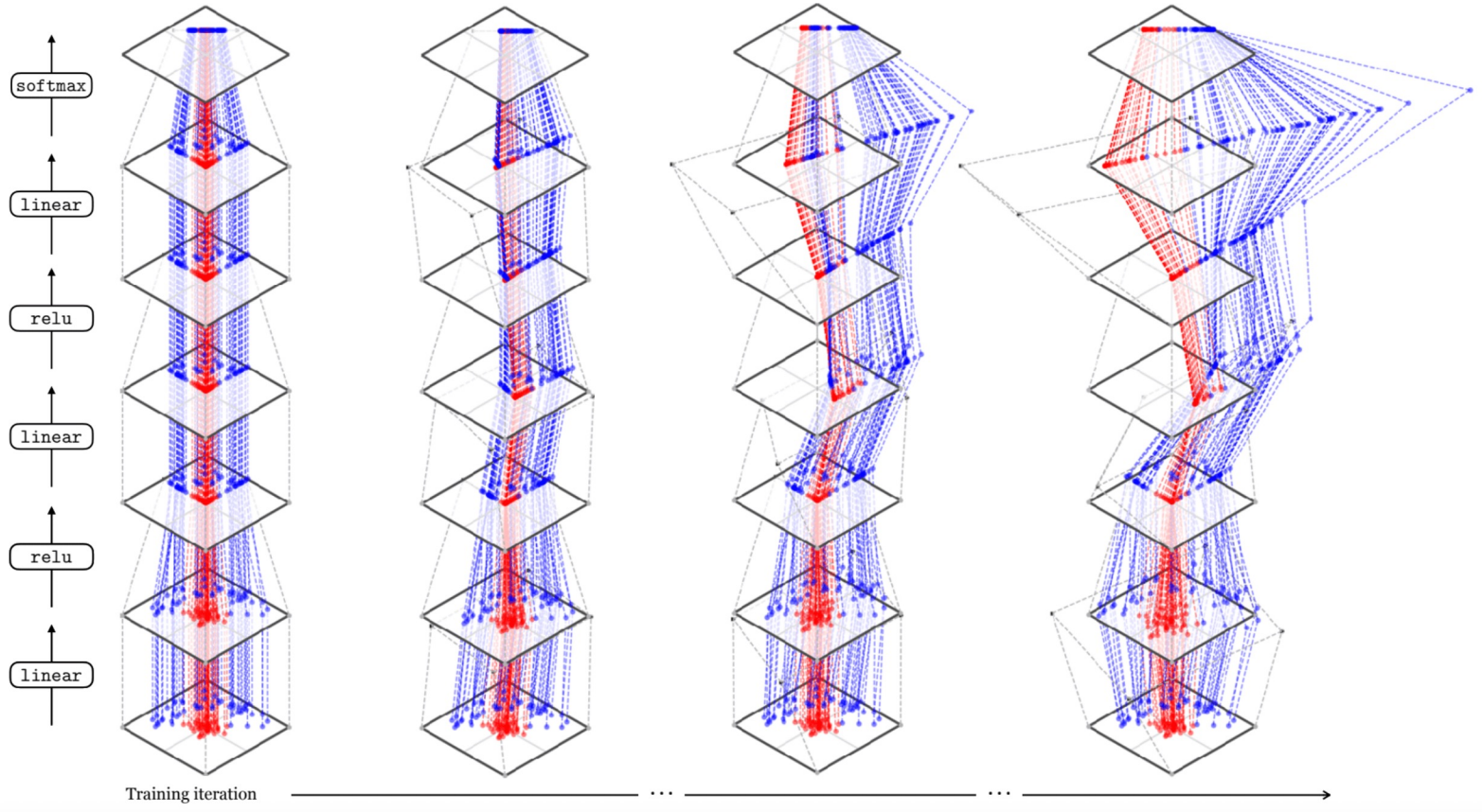
激活函数

- 神经网络运算可视化



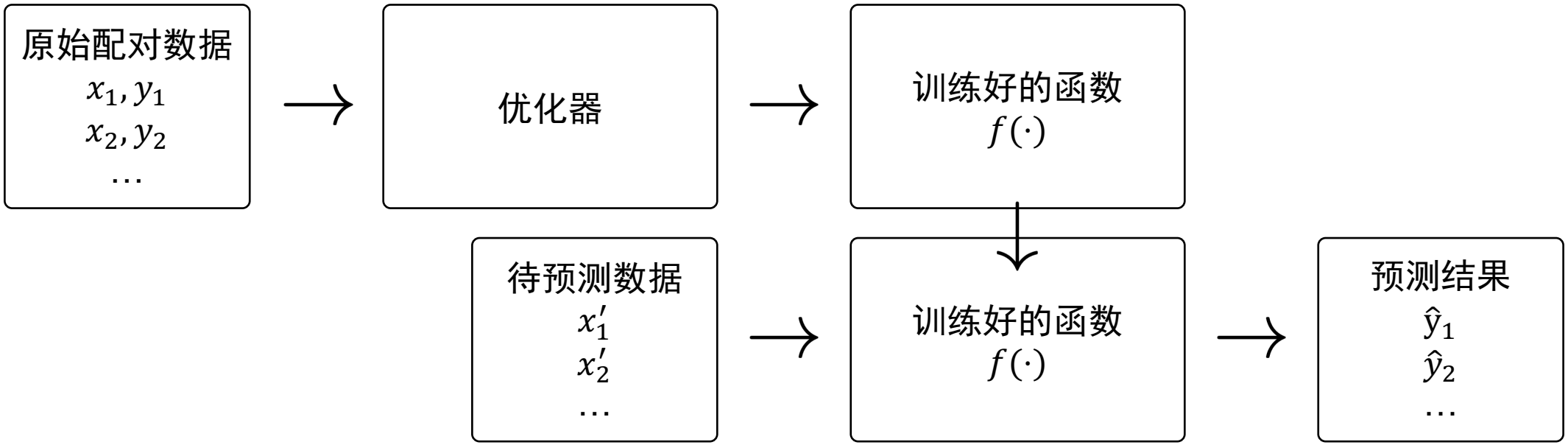
激活函数

- 训练过程可视化



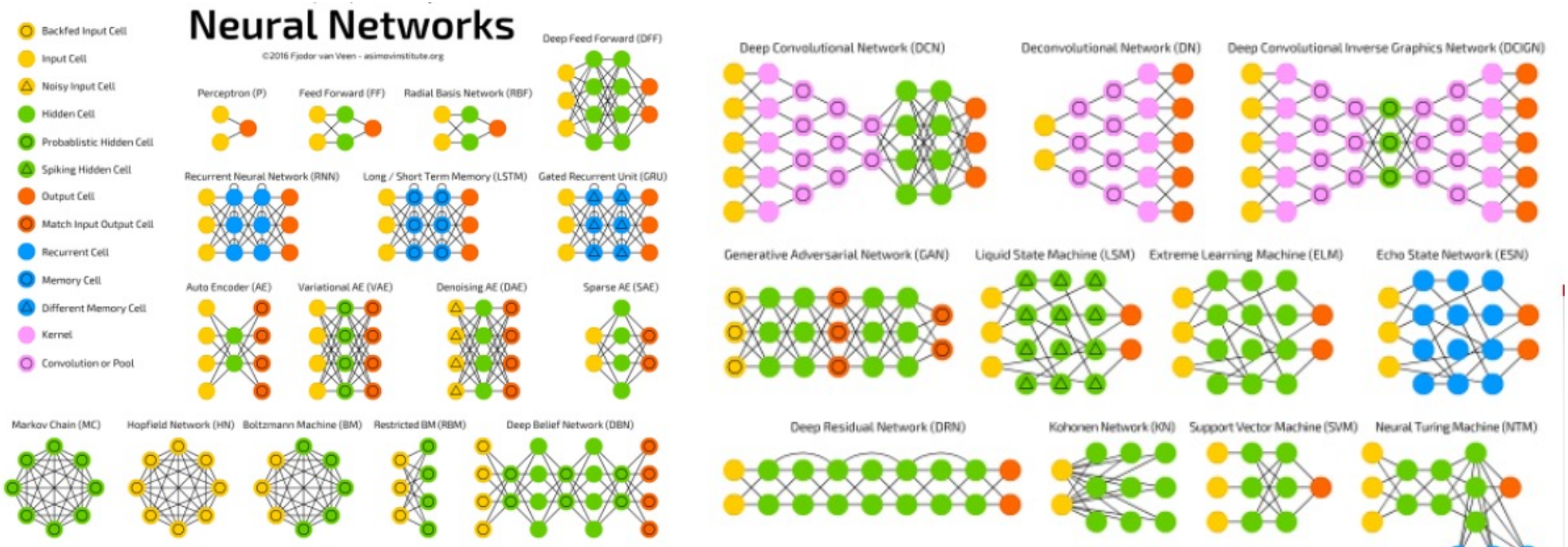
多层感知机 = 很复杂的函数$f(\cdot)$

- 经过多个激活函数和加权求和的处理，多层感知机变成了一个很难展开的复杂的函数，可以完成复杂任务
- 对于m维的输入，n维的输出，多层感知机$f(\cdot)$完成的是一个$f(\cdot): R^m \mapsto R^n$的映射过程
- 在所有权值W通过某种方式（即训练）确定后，通过对输入和输出的定义，可以完成不同种类的任务，如分类、回归...



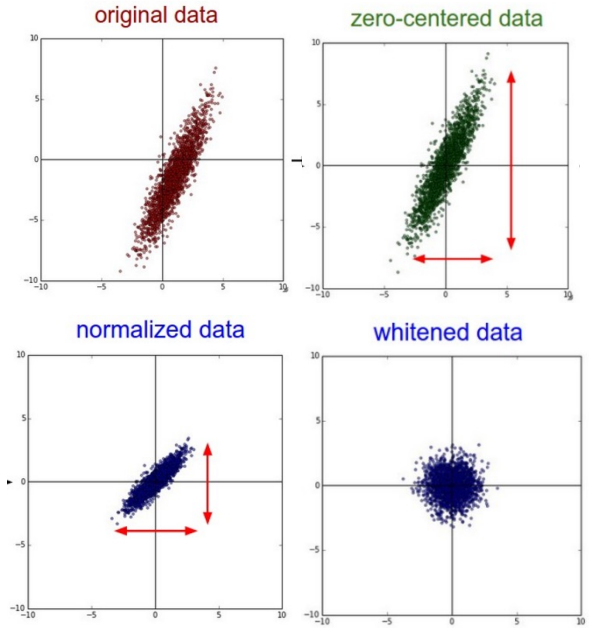
深度神经网络类型

类型多样

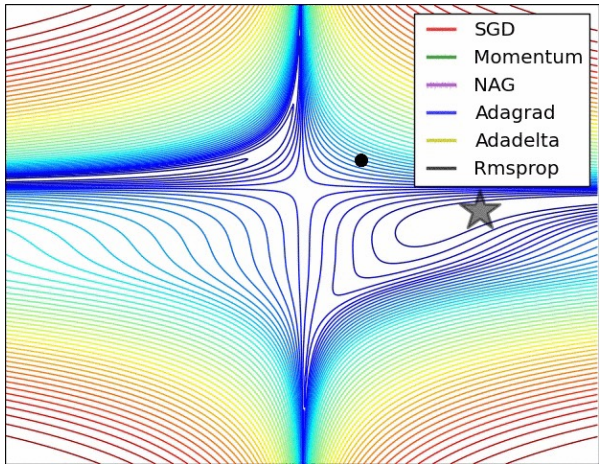


深度学习的流程

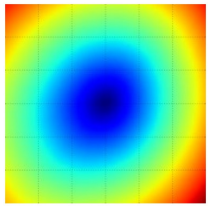
准备训练数据



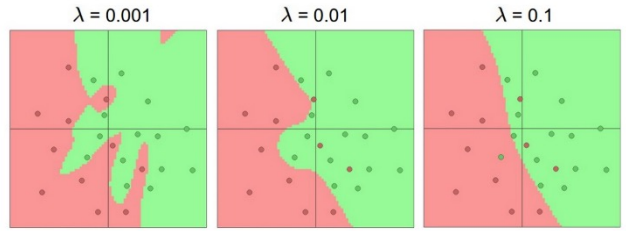
设置优化器



设置损失函数



landscape of a loss function, it varies w.r.t. data, the function itself



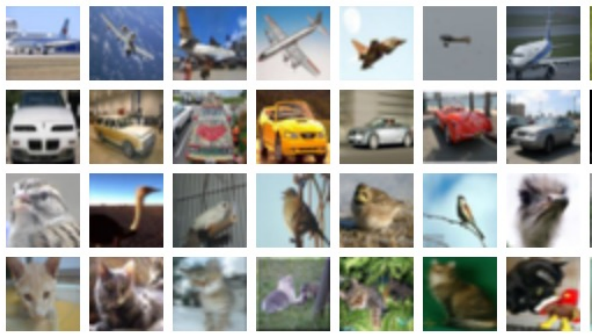
regularization makes decision region smoother

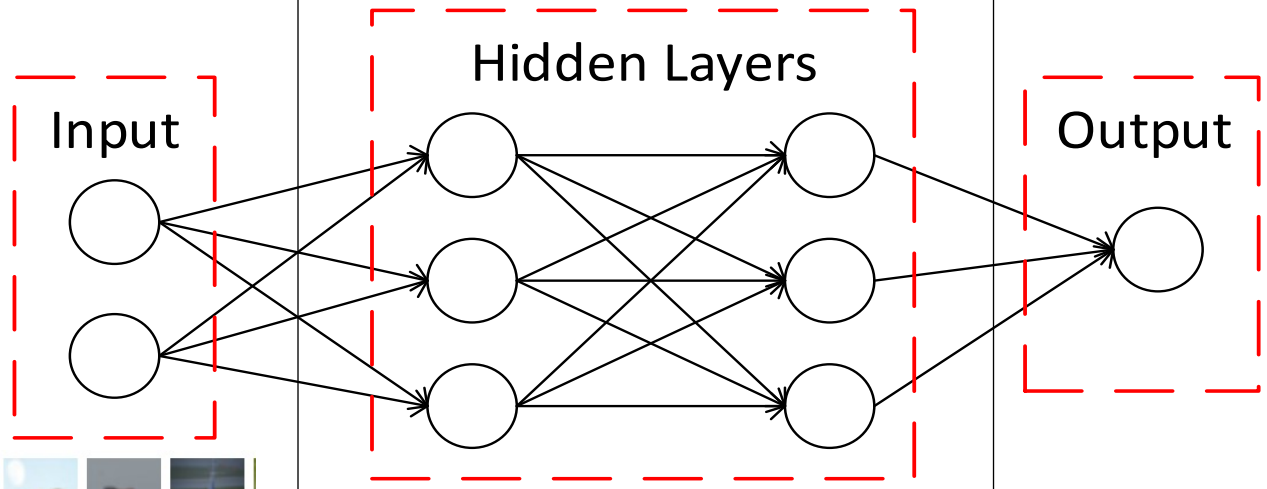
深度学习的流程

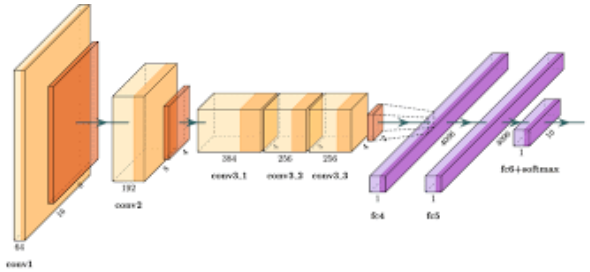
准备训练数据

设置优化器

设置损失函数







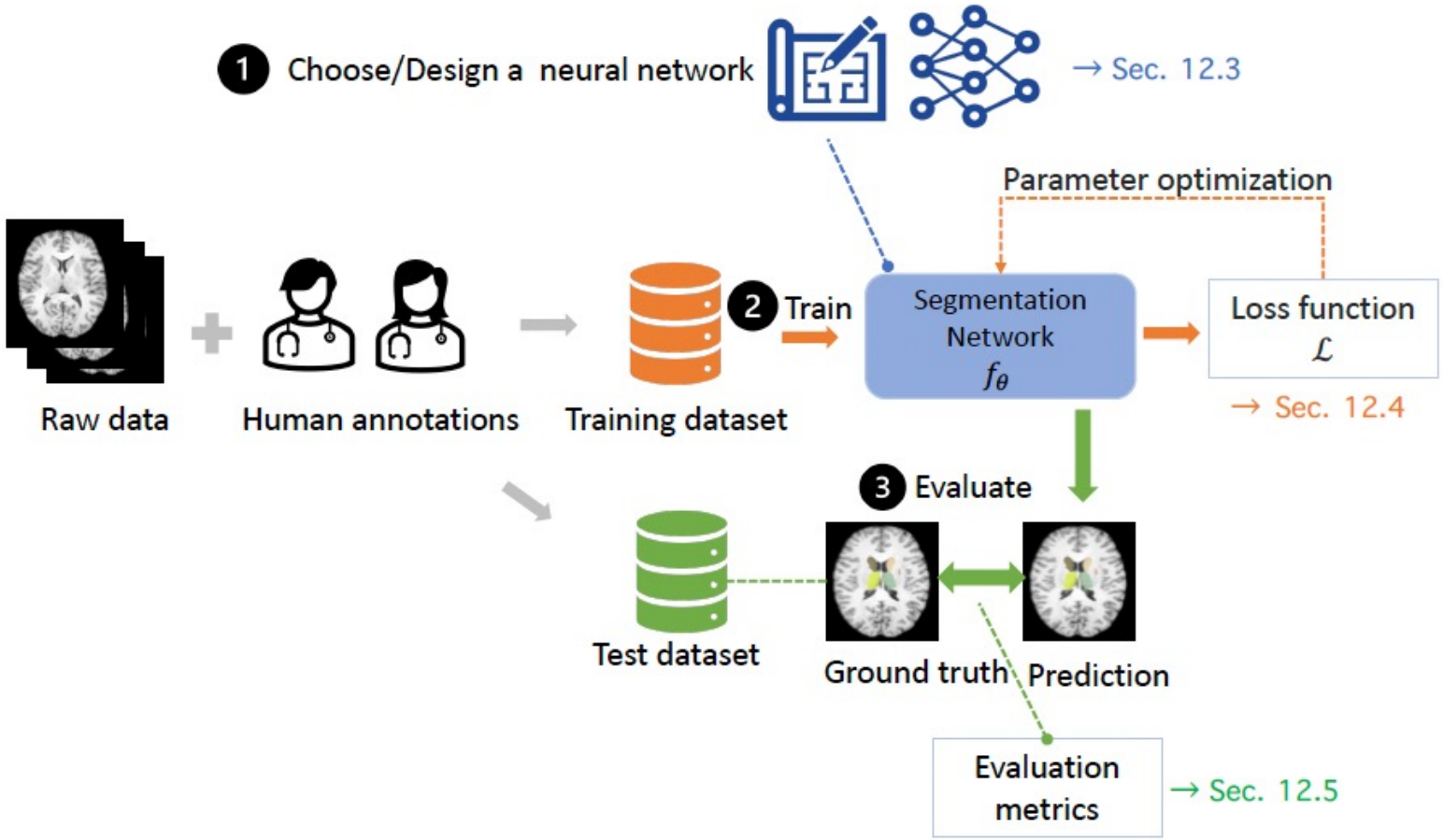
$$\min_{\theta} \mathbb{E}_D[L(f_\theta(x_i), y_i)]$$

Cross entropy:

$$L(f_\theta(x_i), y_i) = \sum_{j=1}^{C} y_{ij} \log f_\theta^j(x_i)$$

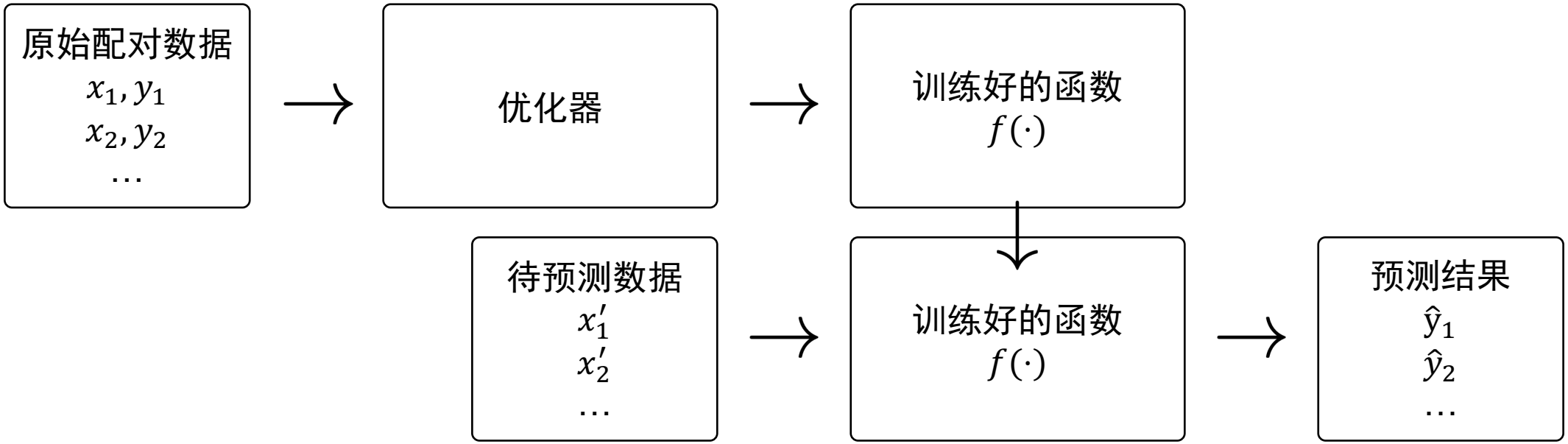
深度学习的流程

- 医学图像分割算法的开发



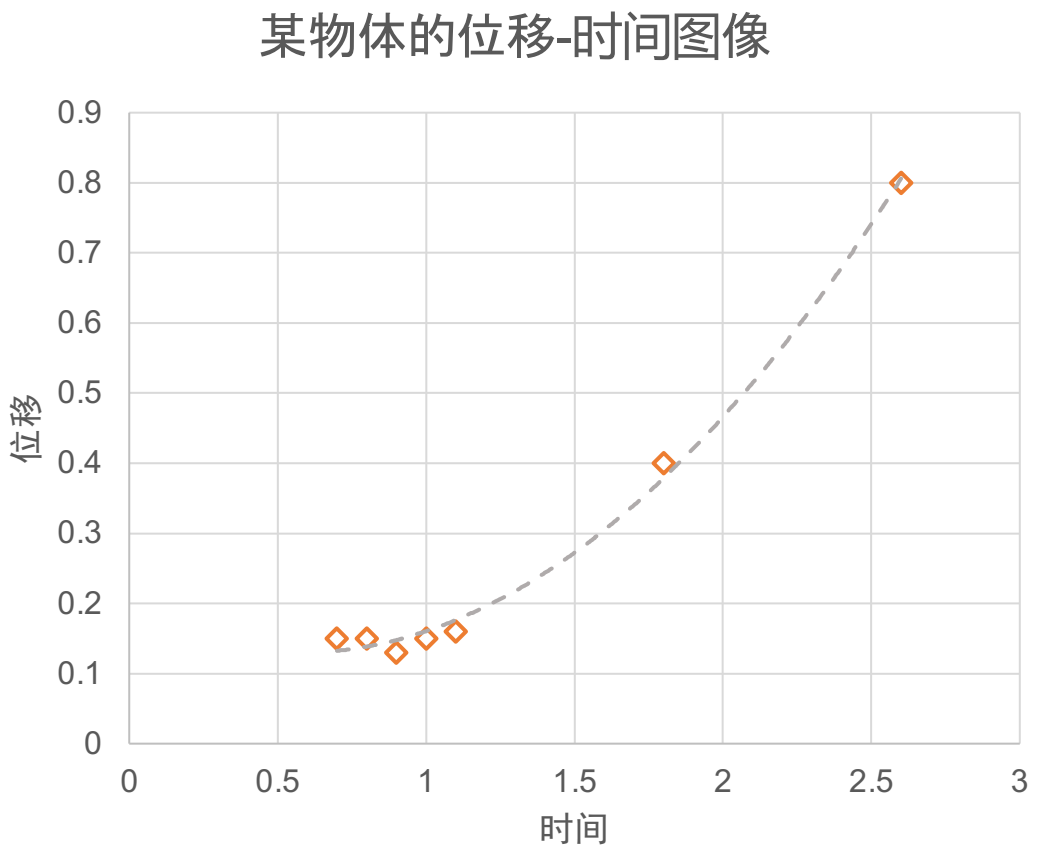
损失函数

- 模型：需要构建出表达能力足够丰富的函数$f(x)$
- **优化目标：需要有能够评价$f(x)$的输出能力的函数$L(w)$**
- 优化器：需要有能够根据$L(w)$来调整$f(x)$的通用方法



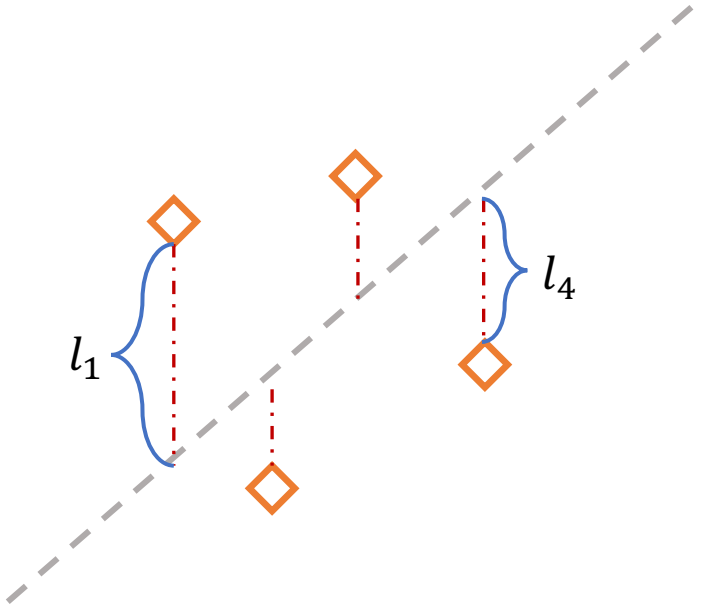
损失函数

- 背景：某物体做匀变速直线运动
- 模型：$x = v_0 t + \frac{1}{2} \cdot a \cdot t^2$
- 在有观测误差的情况下，如何求得**最可能的**初速度和加速度？
- 如何定义**最好**？



损失函数

- 完美的曲线：所有点都在线上
- 好的曲线：所有点尽可能在线附近
- …也即所有点到线的误差最小
- 总误差：$L = \sum_i^n l_i$ 也叫做损失函数
- **损失函数越小，模型表现越好**

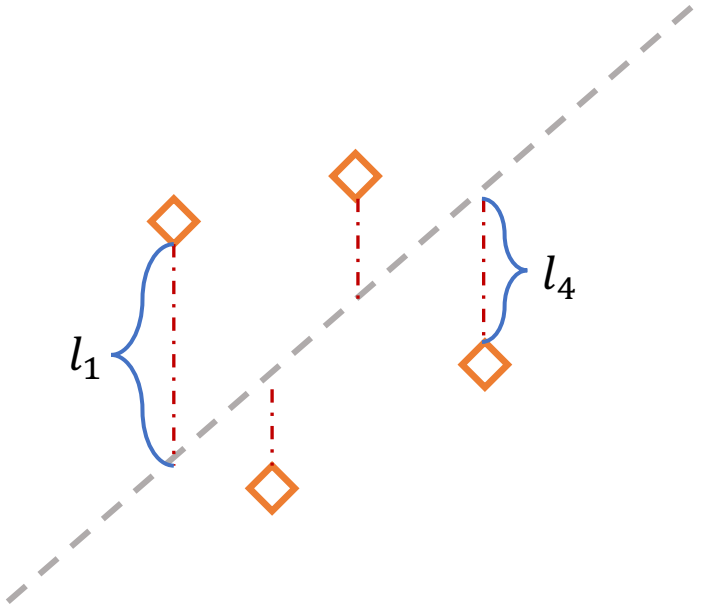


常见损失函数

- 损失函数：$L(\theta) = \sum_i^n l_i(\theta)$
- 直接作差：$l_i(\theta) = y_{real} - y(\theta)$
- 取绝对值：$l_i(\theta) = |y_{real} - y(\theta)|$
- 取平方项：$l_i(\theta) = \left(y_{real} - y(\theta)\right)^2$

$$L = \sum_i^n \left(y_i - y(x_i;\theta)\right)^2$$

Mean Square Error, MSE
均方误差损失



常见损失函数

- 回归问题常用平方损失函数
  - 其中$\boldsymbol{h}^L = \boldsymbol{W}\boldsymbol{h}^{L-1} + \boldsymbol{b}$

$$\ell(\boldsymbol{y}, \boldsymbol{h}^L) = \|\boldsymbol{y} - \boldsymbol{h}^L\|^2$$

- 多类别分类问题常用交叉熵损失函数
  - 其中h_i^L是经过softmax之后第i类的得分

$$\ell(\boldsymbol{y}, \boldsymbol{h}^L) = \sum_i -y_i \log(h_i^L)$$

张量（Tensor）计算：深度神经网络中数据计算的方式

标量 1

向量 [1 2]

矩阵 $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

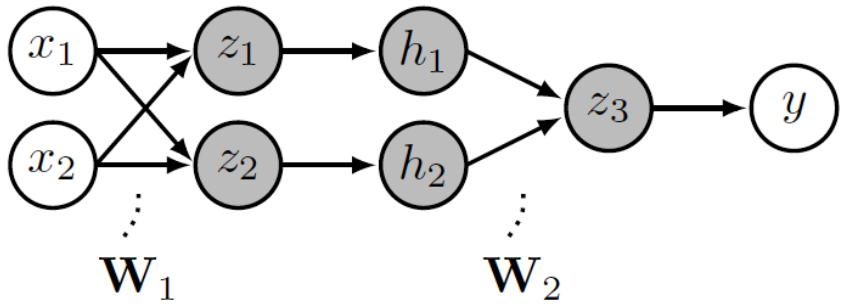
张量 $\left[\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}\right]$

特点	张量	numpy
数据类型	TensorFlow中的数据结构	Python科学计算库
支持的操作	深度学习相关操作，自动求导、神经网络的前向和反向传播等	科学计算和矩阵运算
并行计算	可以存储在GPU中进行并行计算	只能在CPU上进行计算
兼容性	与其他深度学习框架（如PyTorch、Keras等）进行无缝集成	与其他深度学习框架的兼容性相对较差
适用场景	深度学习任务	一般的科学计算和矩阵运算

张量是深度学习架构中一种数据结构，在numpy基础上进一步封装而成，通常用于表示高维矩阵，储存于GPU上，具有并行计算，自动求导的能力，适用于深度学习任务

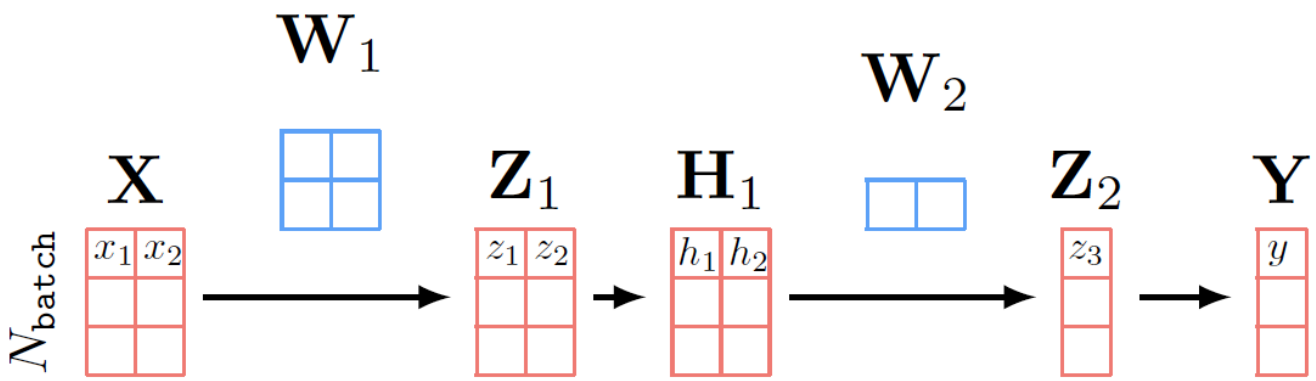
张量计算

对于batch大小为3的2维向量数据，可以作为3*2大小的张量进行处理：



$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h} = g(\mathbf{z})$$
$$z_3 = \mathbf{W}_2\mathbf{h} + b_2$$
$$y = 1(z_3 > 0)$$

Tensor processing with batch size = 3:



目录

1 人工神经网络概述

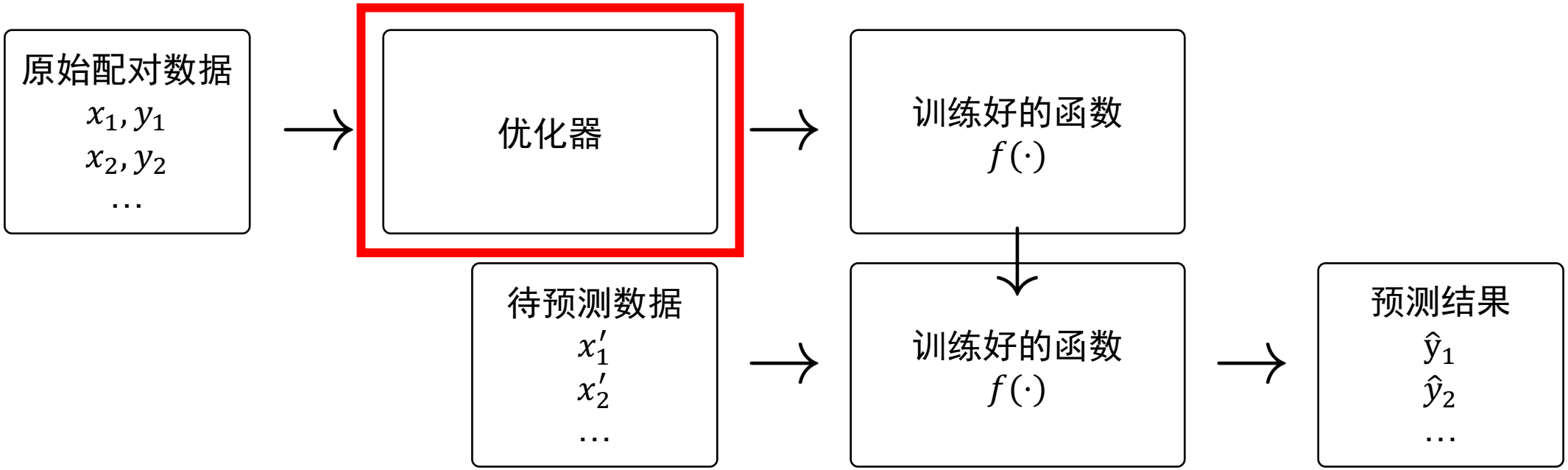
2 单层神经网络

3 深度神经网络

4 反向传播与梯度下降

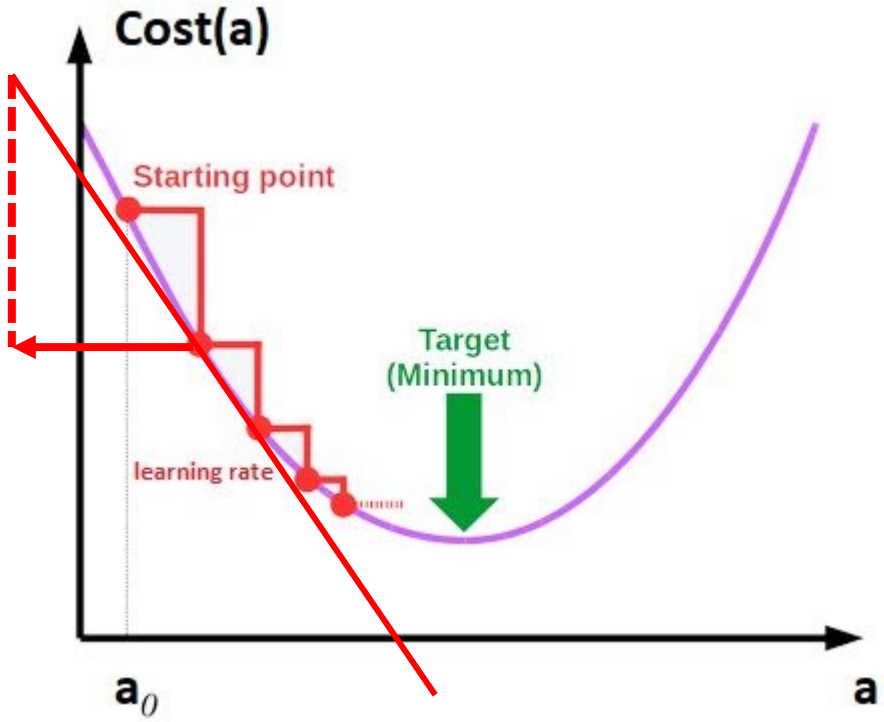
优化器

- 模型：需要构建出表达能力足够丰富的函数$f(x)$
- 优化目标：需要有能够评价$f(x)$的输出能力的函数$L(w)$
- **优化器：需要有能够根据$L(w)$来调整$f(x)$的通用方法**



梯度下降——沿函数变化最快的方向修改变量

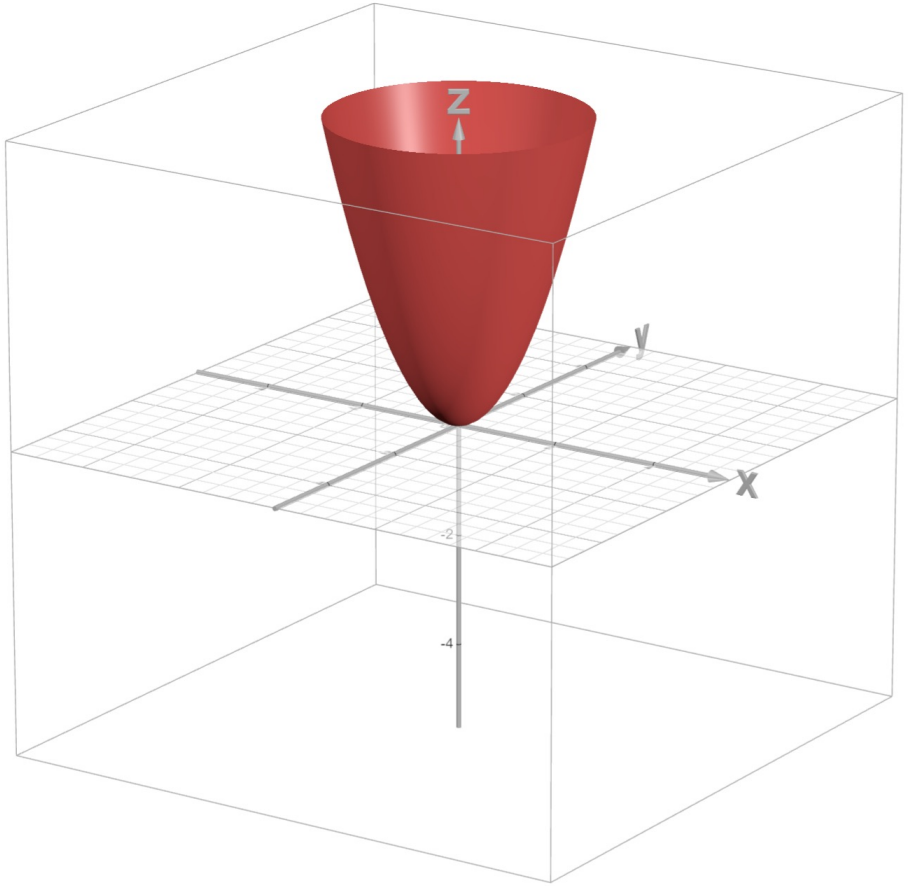
- 给定一个初始状态
- 已知一个优化目标
- 每一步向**负梯度方向**走一点
- 重复上述过程，直到收敛
- 梯度就是偏导数在对应轴的投影



例子：二元函数的梯度如何求?

- 求：点$(1,2)$处$f(x,y)$的梯度

1. 求$f(x,y)$对x,y的偏导数
2. 将x,y的值代入偏导数
3. 将偏导数组合成向量



$$f(x,y)=x^2+y^2$$

例子：二元函数的梯度如何求?

- 求：点$(1,2)$处$f(x,y)$的梯度

1. 求$f(x,y)$对x,y的偏导数

$$\nabla_x(f) = 2x$$

$$\nabla_y(f) = 2y$$

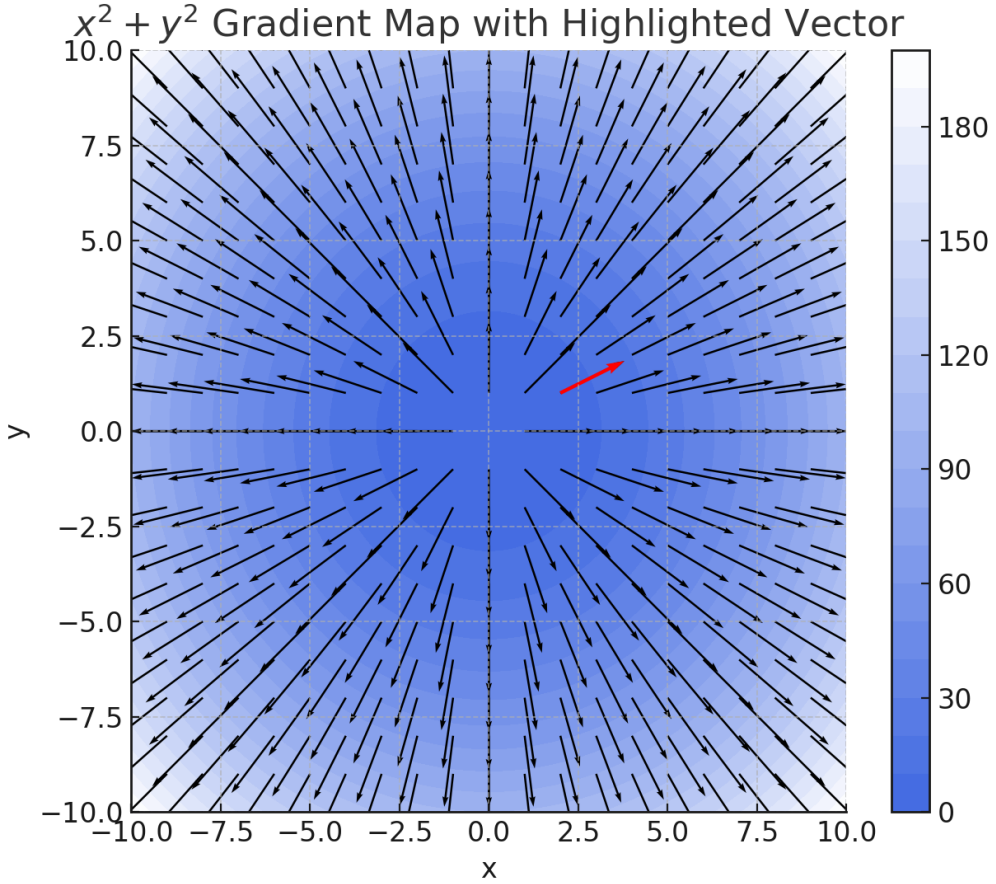
2. 将x,y的值代入偏导数

$$\nabla_x\big(f(1)\big) = 2\times 1 = 2$$

$$\nabla_y\big(f(2)\big) = 2\times 2 = 4$$

3. 将偏导数组合成向量

$$[2,4]^T$$



$$f(x,y) = x^2 + y^2$$

用梯度下降法进行学习

- 优化目标：调整网络中的权值W，使损失函数L最小
- 策略：迭代，每一步都向L减小的方向移动
- $W^{(t+1)} = W^{(t)} - \eta \cdot \nabla_W L$
- η：学习率，即每一步要走多远

输入：学习率ϵ、模型初始化参数θ

while 未收敛 do

计算损失函数对于模型参数的梯度$\boldsymbol{g}$;

计算更新：$\Delta\theta = -\epsilon \odot \boldsymbol{g}$;

更新参数：$\theta = \theta + \Delta\theta$;

end

关键问题：计算机如何求复杂函数的梯度？

背景知识：导数的向量表达

- 对于R^n上的列向量$\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$
- 定义自变量为$\boldsymbol{x}$的函数$y = f(\boldsymbol{x})$
- 如果y是标量，则导数$\frac{\partial y}{\partial \boldsymbol{x}} = \left(\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n}\right)$，是$(1 \times n)$的行向量
- 如果y是R^m上的列向量，则

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \in R^m \times R^n$$

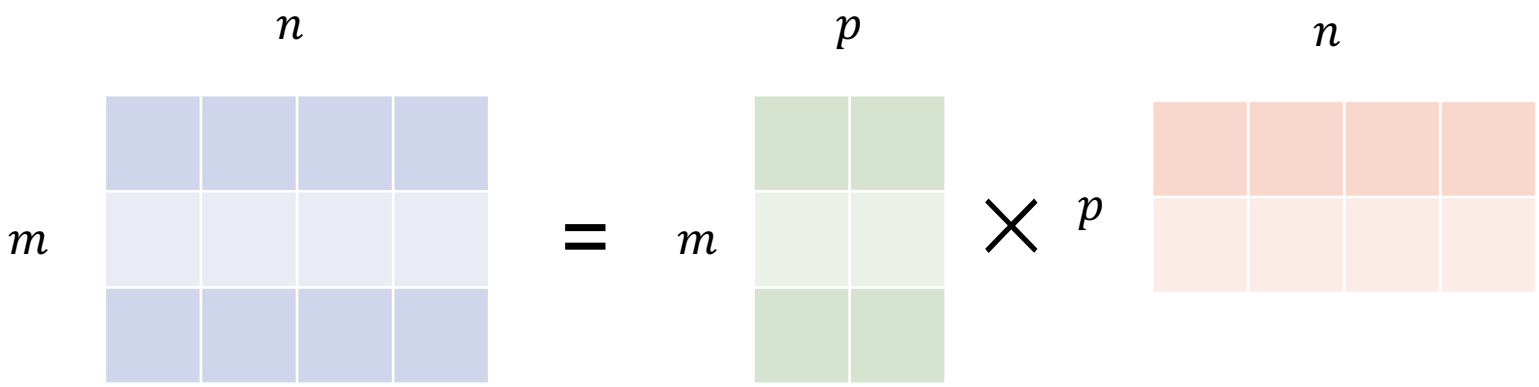
背景知识：链式法则（Chain Rule）

- 对于函数$h(\boldsymbol{x}) = f\big(g(\boldsymbol{x})\big)$，它的导数为

$$h'(\boldsymbol{x}) = f'\big(g(\boldsymbol{x})\big)g'(\boldsymbol{x}) = f'(\boldsymbol{u})u'(\boldsymbol{x})$$

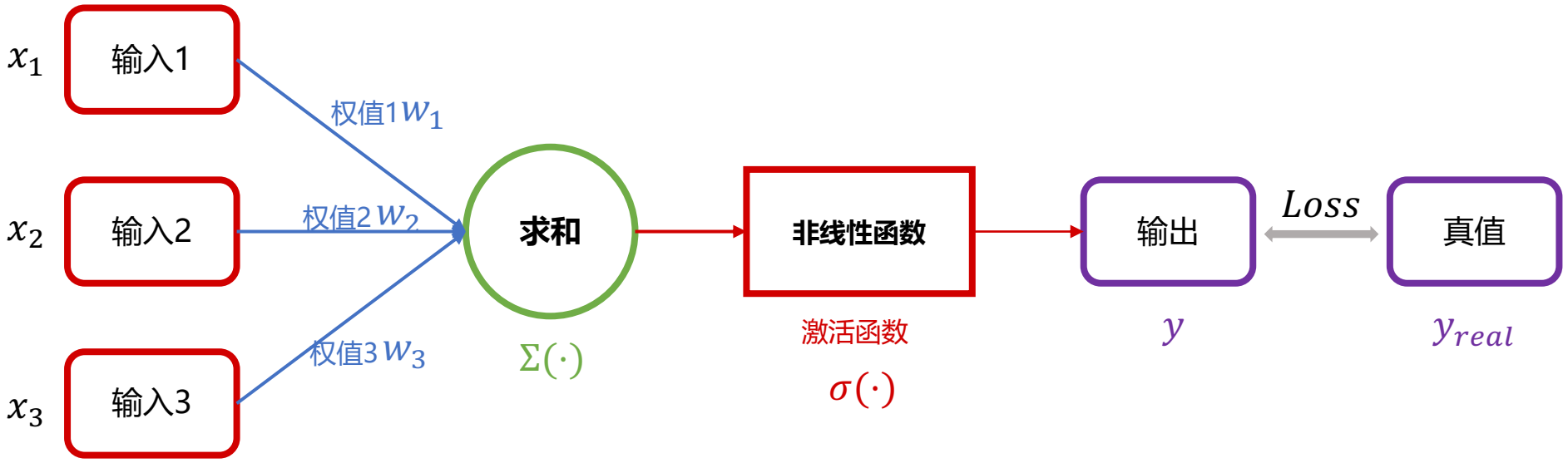
- 设$\boldsymbol{z} = f(\boldsymbol{u}) \in R^m$，$\boldsymbol{u} = g(\boldsymbol{x}) \in R^p$，$\boldsymbol{x} \in R^n$，根据同样的法则：

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}_{\boldsymbol{x}=\boldsymbol{a}} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{u}}_{u=g(\boldsymbol{a})} \cdot \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}_{\boldsymbol{x}=\boldsymbol{a}}$$



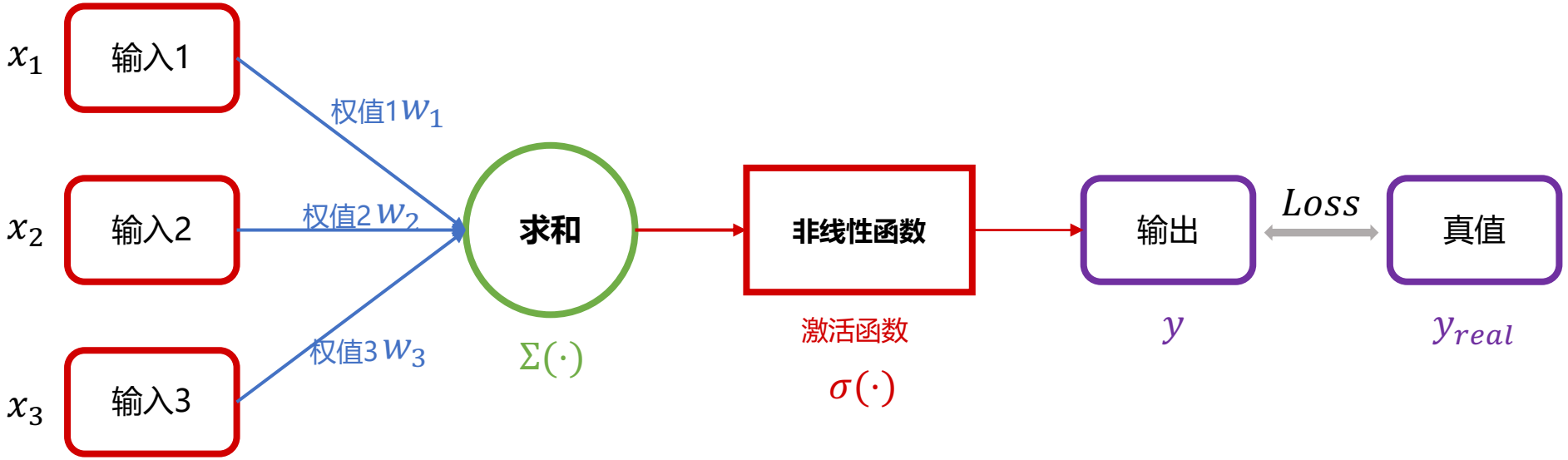
正向传播——计算loss函数的值

- 正向传播的过程就是从左到右的计算过程
- 在这个例子中：$y = \sigma(w_1x_1 + w_2x_2 + w_3x_3)$
- $loss = (y - y_{real})^2$
- 这个过程就是单个神经元的正向传播过程



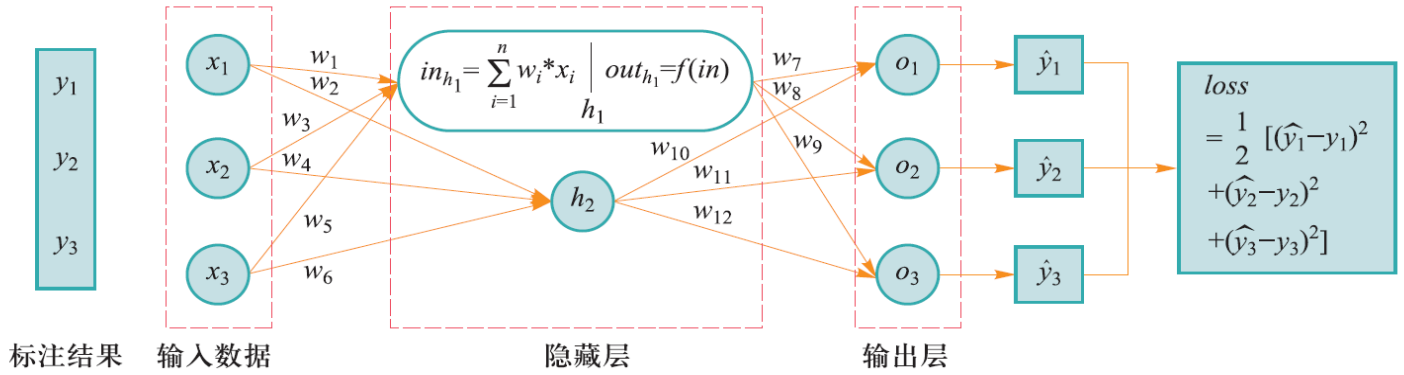
反向传播——逐层计算梯度

- 以待调整参数w_1为例，更新规则为$w_1^{new} = w_1^{old} - \eta \cdot \boldsymbol{\nabla_{w_1} L}$
- 根据链式法则：$\nabla_{w_1} L = \frac{dL}{d\sigma} \cdot \frac{d\sigma}{d\Sigma} \cdot \frac{d\Sigma}{dw_1}$
- 同理可以使用这种方法计算损失函数L对任意一个权值w_i的梯度$\nabla_{w_i} L$
- $w_i^{new} = w_i^{old} - \eta \cdot \nabla_{w_i} L$

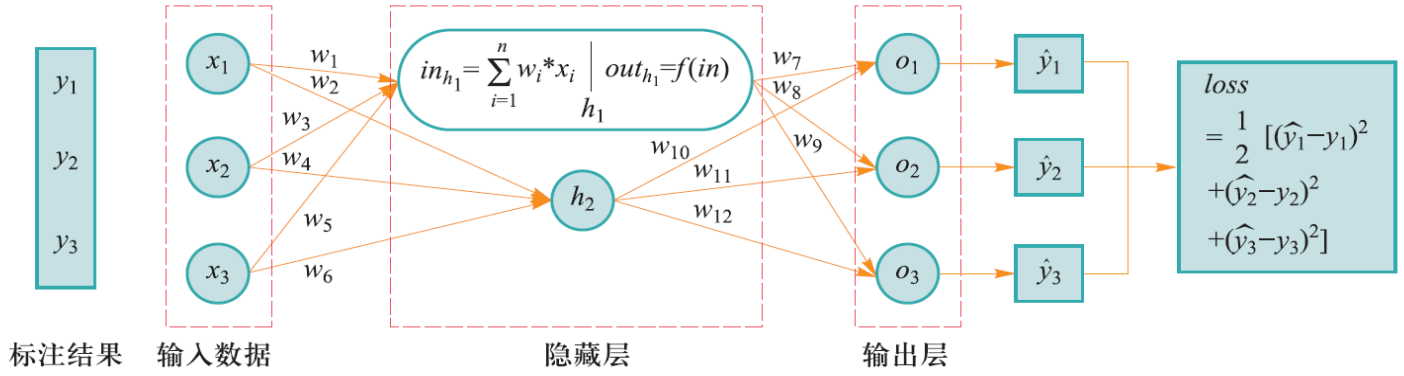


反向传播——逐层计算梯度

- 由多个神经元可以组成多层的神经网络，中间的层就叫做隐藏层
- 以3层为例，中间有1层隐藏层，图中的隐藏层有2个神经元
- 三分类问题：
  - 输入：$\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}$
  - 输出：$\hat{y}_1, \hat{y}_2, \hat{y}_3 \in \{(1,0,0), (0,1,0), (0,0,1)\}$
  - 损失函数：均方损失函数(MSE)

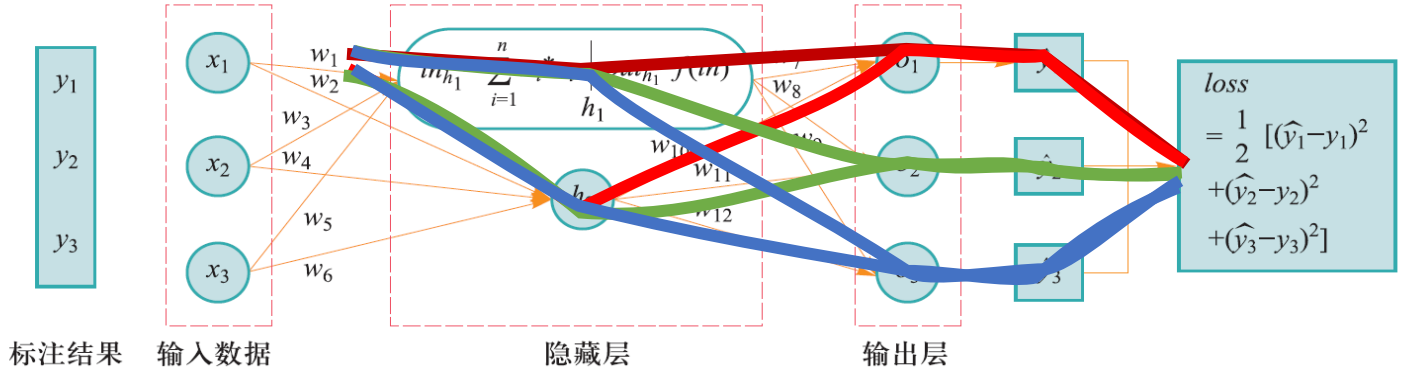


反向传播——逐层计算梯度



单元	相邻前序神经元所传递信息的线性累加	非线性变换
h_1	$\mathrm{In}_{h_1} = w_1 \times x_1 + w_3 \times x_2 + w_5 \times x_3$	$\mathrm{Out}_{h_1} = \mathrm{sigmoid}(\mathrm{In}_{h_1})$
h_2	$\mathrm{In}_{h_2} = w_2 \times x_1 + w_4 \times x_2 + w_6 \times x_3$	$\mathrm{Out}_{h_2} = \mathrm{sigmoid}(\mathrm{In}_{h_2})$
o_1	$\mathrm{In}_{o_1} = w_7 \times \mathrm{Out}_{h_1} + w_{10} \times \mathrm{Out}_{h_2}$	$\hat{y_1} = \mathrm{Out}_{o_1} = \mathrm{sigmoid}(\mathrm{In}_{o_1})$
o_2	$\mathrm{In}_{o_2} = w_8 \times \mathrm{Out}_{h_1} + w_{11} \times \mathrm{Out}_{h_2}$	$\hat{y_2} = \mathrm{Out}_{o_2} = \mathrm{sigmoid}(\mathrm{In}_{o_2})$
o_3	$\mathrm{In}_{o_3} = w_9 \times \mathrm{Out}_{h_1} + w_{12} \times \mathrm{Out}_{h_2}$	$\hat{y_3} = \mathrm{Out}_{o_3} = \mathrm{sigmoid}(\mathrm{In}_{o_3})$

反向传播——逐层计算梯度



计算$loss$对w_1的梯度：

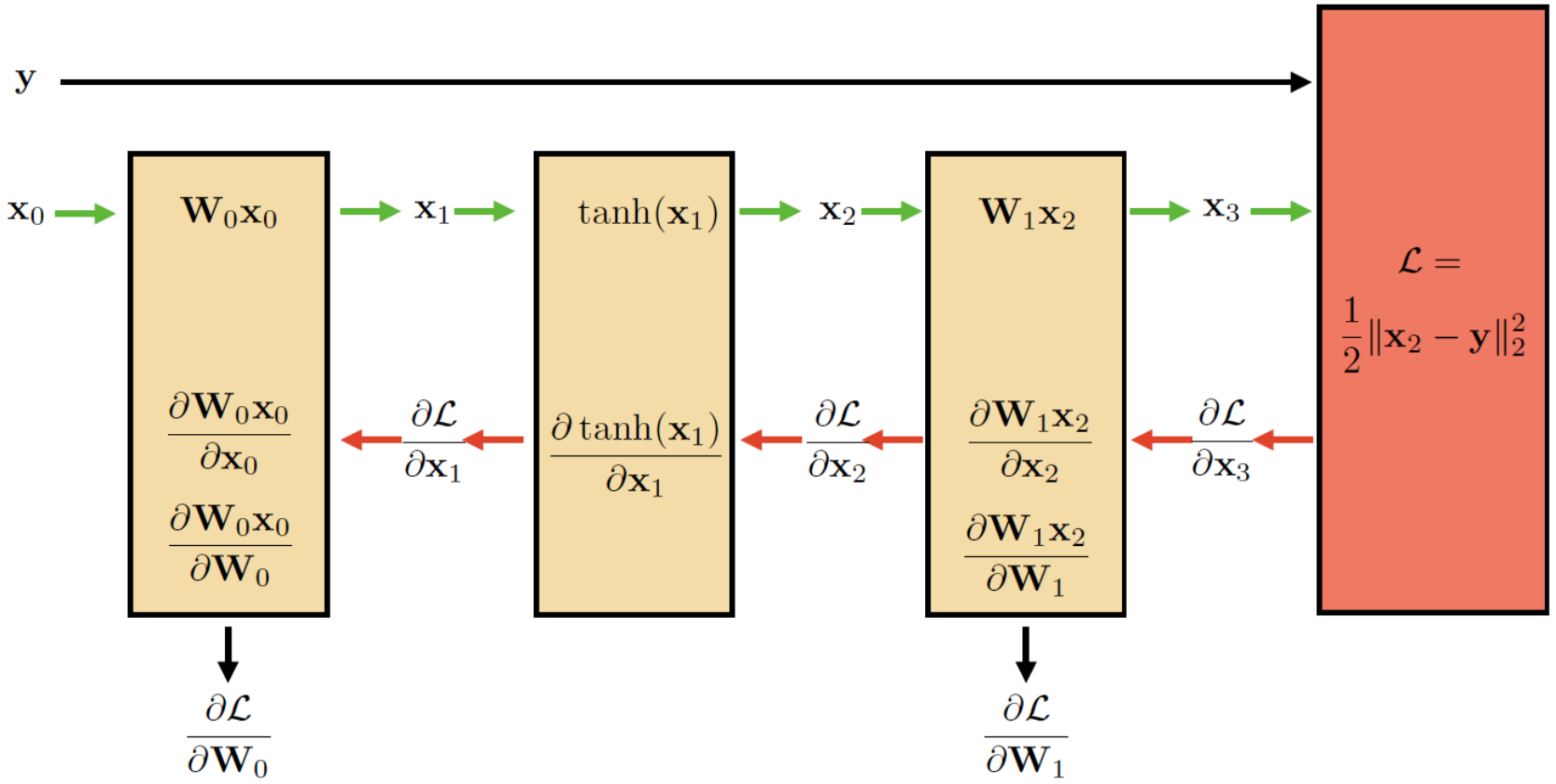
$$\nabla_{w_1} L = \frac{dL}{dw_1} = \frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dw_1} + \frac{dL}{d\hat{y}_2} \cdot \frac{d\hat{y}_2}{dw_1} + \frac{dL}{d\hat{y}_3} \cdot \frac{d\hat{y}_3}{dw_1}$$

$$\frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dw_1} = \boldsymbol{\frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dIn_{o_1}}} \cdot \frac{dIn_{o_1}}{dOut_{h_1}} \cdot \frac{dOut_{h_1}}{dIn_{h_1}} \cdot \frac{dIn_{h_1}}{dw_1} + \boldsymbol{\frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dIn_{o_1}}} \cdot \frac{dIn_{o_1}}{dOut_{h_2}} \cdot \frac{dOut_{h_2}}{dIn_{h_2}} \cdot \frac{dIn_{h_2}}{dw_1}$$

$$\nabla_{w_7} L = \frac{dL}{dw_7} = \boldsymbol{\frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dIn_{o_1}}} \cdot \frac{dIn_{o_1}}{dw_7}; \quad \nabla_{w_8} L = \frac{dL}{dw_8} = \boldsymbol{\frac{dL}{d\hat{y}_1} \cdot \frac{d\hat{y}_1}{dIn_{o_1}}} \cdot \frac{dIn_{o_1}}{dw_8}$$

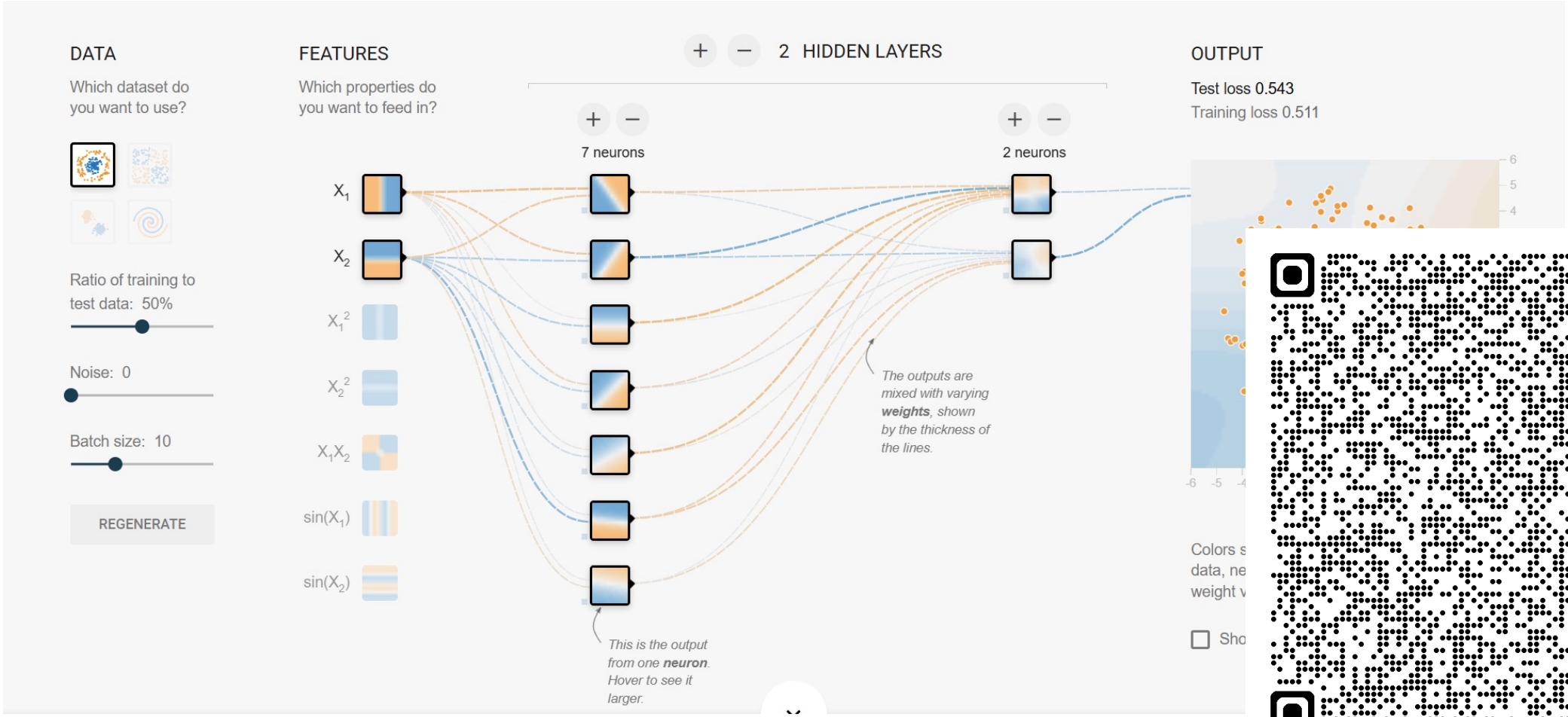
正向传播和反向传播

通过正向传播和反向传播，可以实现复杂梯度的迭代计算，并最终用于梯度下降！



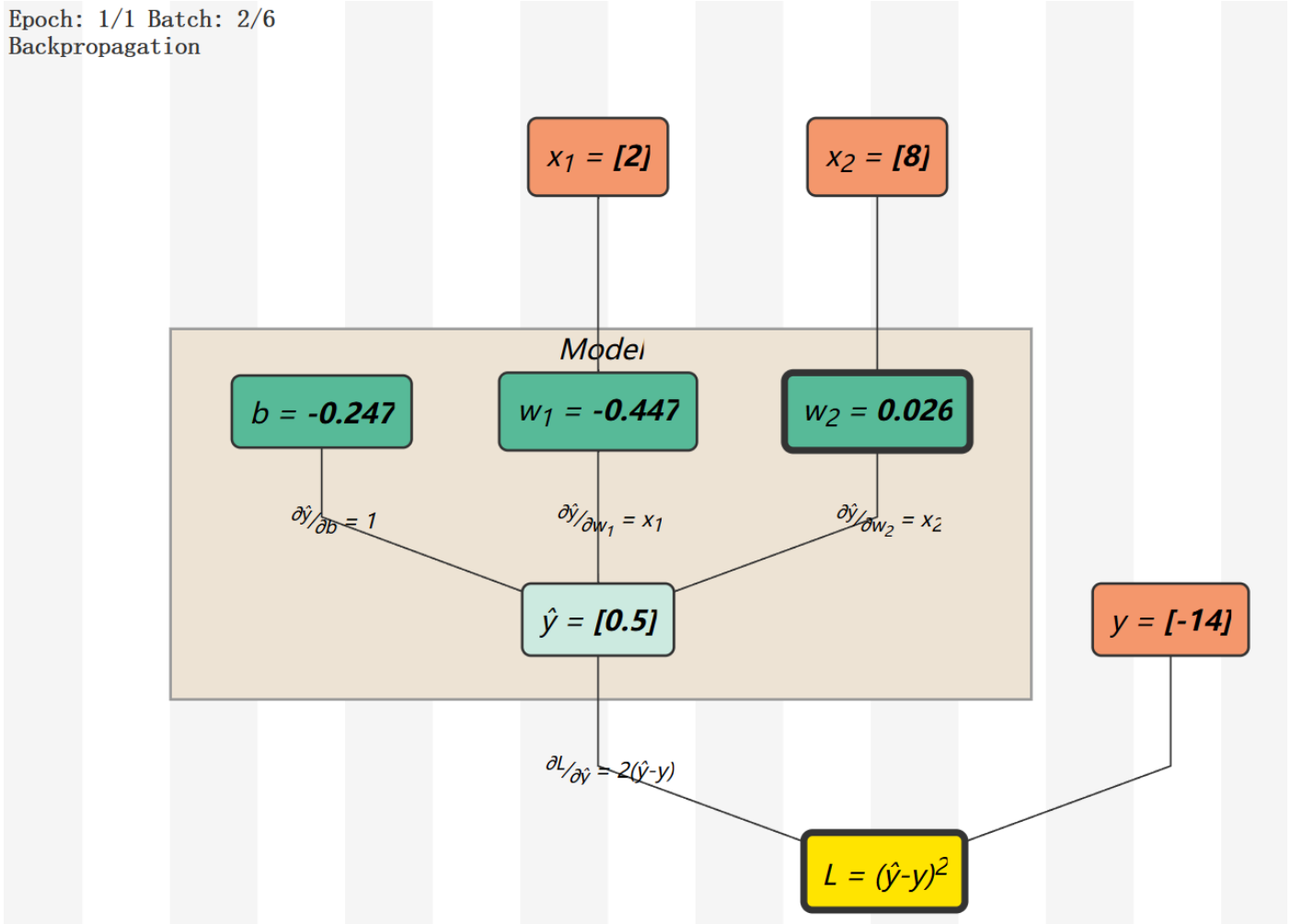
神经网络训练Demo

- [A Neural Network Playground]



反向传播Demo

- Backpropagation Demo



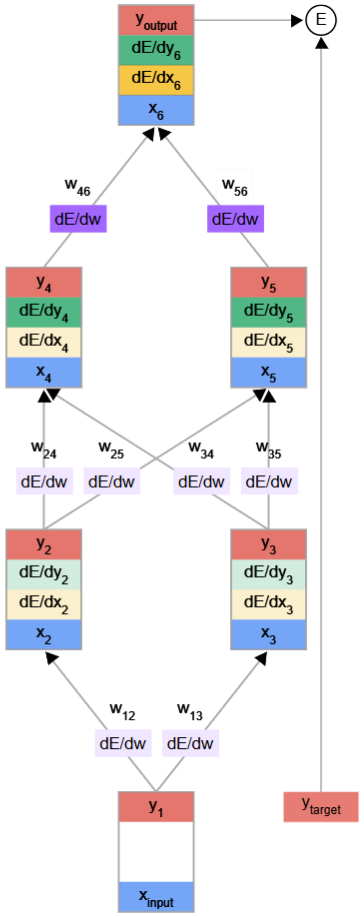
反向传播演示

- 反向传播演示

反向传播

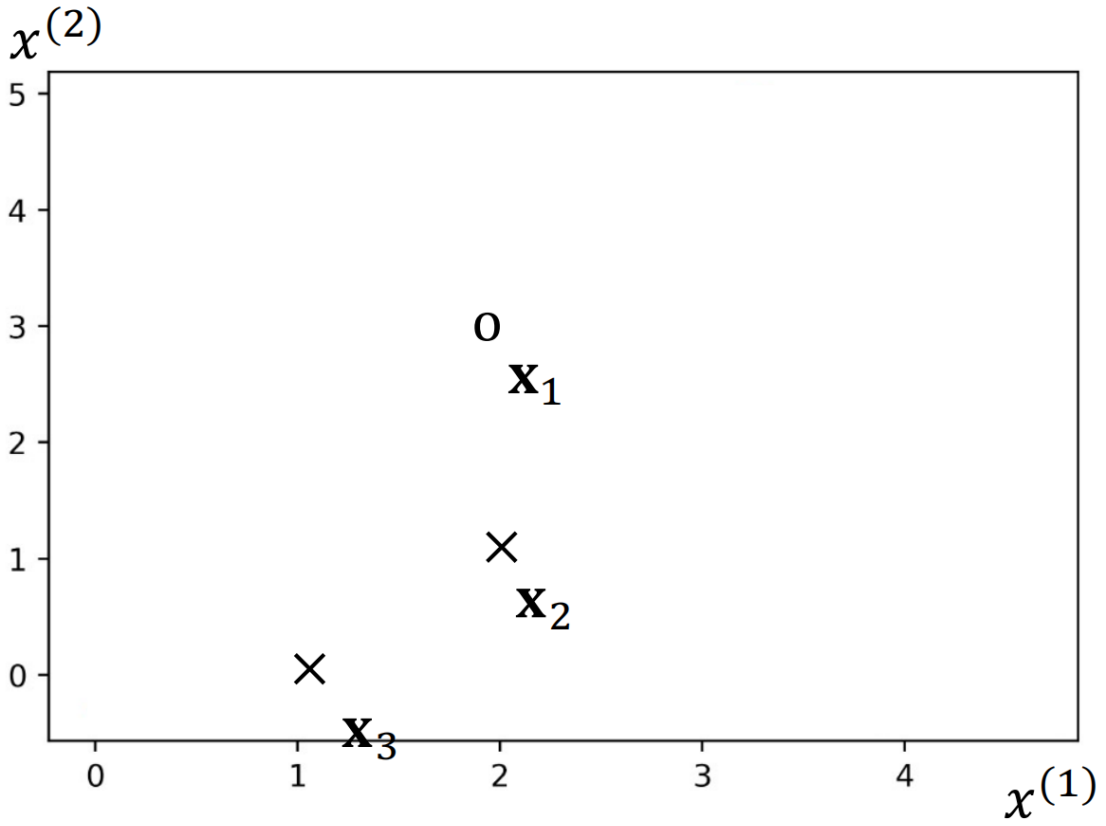
根据链式法则，我们还可以根据上一层得出 $\frac{dE}{dy}$。此时，我们形成了一个完整的循环。

$$\frac{\partial E}{\partial y_i} = \sum_{j \in out(i)} \frac{\partial x_j}{\partial y_i} \frac{\partial E}{\partial x_j} = \sum_{j \in out(i)} w_{ij} \frac{\partial E}{\partial x_j}$$



例题——单层感知器分类问题

给定如图所示的训练数据集，其正实例点是$\mathbf{x}_1 = [2;3]$，负实例点是$\mathbf{x}_2 = [2;1]$，$\mathbf{x}_3 = [1;0]$，用感知机学习算法的原始形式求感知机模型$f(x) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$。设学习率$\eta = 1$



例题——单层感知器分类问题

- 取初值$\mathbf{w}^{(0)}=[0;0]$，$b^{(0)}=0$，得到初始感知机模型
- 对$\mathbf{x}_1=[2;3]$，$y_1\left(\left[\mathbf{w}^{(0)}\right]^{\mathrm{T}}\mathbf{x}_1+b^{(0)}\right)=0$，分类错误，更新$\mathbf{w}$与$b$

$$\mathbf{w}^{(1)}=\mathbf{w}^{(0)}+y_1\mathbf{x}_1=[2;3]$$

$$b^{(1)}=b^{(0)}+y_1=1$$

- 对$\mathbf{x}_2$，$y_2\left(\left[\mathbf{w}^{(1)}\right]^{\mathrm{T}}\mathbf{x}_2+b^{(1)}\right)<0$，分类错误，

$$\mathbf{w}^{(2)}=\mathbf{w}^{(1)}+y_2\mathbf{x}_2=[0;2]$$

$$b^{(2)}=b^{(1)}+y_2=0$$

- 对$\mathbf{x}_2$，分类错误，继续更新

$$\mathbf{w}^{(3)}=\mathbf{w}^{(2)}+y_2\mathbf{x}_2=[-2;1]$$

$$b^{(3)}=b^{(2)}+y_2=-1$$

例题——单层感知器分类问题

- 对$\mathbf{x}_2$， 分类正确，对$\mathbf{x}_3$，正确，对$\mathbf{x}_1$，错误

$$\mathbf{w}^{(4)} = \mathbf{w}^{(3)} + y_1\mathbf{x}_1 = [0; 4]$$

$$b^{(4)} = b^{(3)} + y_1 = 0$$

- 对$\mathbf{x}_1$， 分类正确，对$\mathbf{x}_3$，错误

$$\mathbf{w}^{(5)} = \mathbf{w}^{(4)} + y_3\mathbf{x}_3 = [-1; 4]$$

$$b^{(5)} = b^{(4)} + y_3 = -1$$

- 对$\mathbf{x}_1$， 分类正确，对$\mathbf{x}_2$，错误

$$\mathbf{w}^{(6)} = \mathbf{w}^{(5)} + y_2\mathbf{x}_2 = [-3; 3]$$

$$b^{(6)} = b^{(5)} + y_2 = -2$$

例题——单层感知器分类问题

- 对$\mathbf{x}_2$， 分类正确，对$\mathbf{x}_2$，正确，对$\mathbf{x}_3$，正确

$$\mathbf{w}^{(6)} = \mathbf{w}^{(5)} + y_2\mathbf{x}_2 = [-3; 3]$$

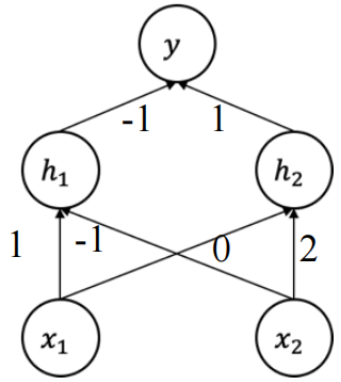
$$b^{(6)} = b^{(5)} + y_2 = -2$$

- 最终模型为：

$$f(\mathrm{x}) = \mathrm{sign}([-3,3]^{\mathrm{T}}\mathbf{x} - 2)$$

梯度推导例题——多层感知器

下图为具有单个隐藏层的多层感知机，模型输入为$\mathbf{x} = [x_1, x_2]^\top$，隐藏层包含两个神经元$h_1, h_2$，模型输出为$y$，忽略偏置项：



隐藏层的激活函数为ReLU，输出层不加激活函数，损失函数为$\ell(y,t) = \frac{1}{2}(y-t)^2$，$t$是输出$y$的目标值，输入层和隐藏层的权重用$W$和$V$表示。

假设初始化$W = \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}$，$V = [-1 \quad 1]$，输入$\boldsymbol{x} = [1 \quad -1]^\top$，目标值$t = 1$，计算损失函数对权重$W$的梯度。

梯度推导例题——多层感知器

$y=-2$。令$r=V\sigma(Wx)$，$s=Wx$，计算中间变量$h=\sigma(Wx)=[2 \quad 0]^T$

$$s=Wx=[2 \quad -2]^T$$

ReLU激活函数的梯度为

$$\sigma^g(x)=\begin{cases}0, & x\leq 0\\ 1, & x>0\end{cases}$$

根据链式法则，有

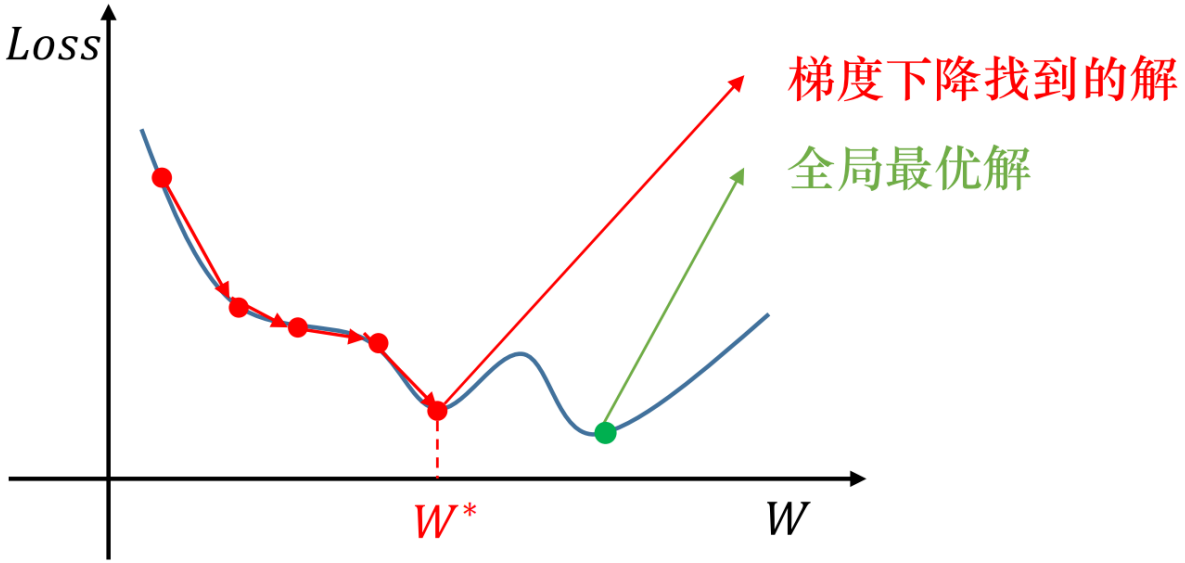
$$\frac{\partial L}{\partial V}=\left[\frac{\partial L}{\partial y}\times\frac{\partial y}{\partial r}\times\frac{\partial r}{\partial V_0} \quad \frac{\partial L}{\partial y}\times\frac{\partial y}{\partial r}\times\frac{\partial r}{\partial V_1}\right]=[-6 \quad 0]$$

$$\frac{\partial L}{\partial W}=\begin{bmatrix}\frac{\partial L}{\partial h_0}\times\frac{\partial h_0}{\partial s_0}\times\frac{\partial s_0}{\partial W_{00}} & \frac{\partial L}{\partial h_0}\times\frac{\partial h_0}{\partial s_0}\times\frac{\partial s_0}{\partial W_{01}}\\ \frac{\partial L}{\partial h_1}\times\frac{\partial h_1}{\partial s_1}\times\frac{\partial s_1}{\partial W_{10}} & \frac{\partial L}{\partial h_1}\times\frac{\partial h_1}{\partial s_1}\times\frac{\partial s_1}{\partial W_{11}}\end{bmatrix}$$

$$=\begin{bmatrix}3 & -3\\ 0 & 0\end{bmatrix}$$

局部最优解

- MLP的损失函数是非凸的，所以梯度下降很难找到全局最优解
- 通常局部最优解的效果已经可以接受



权重初始化

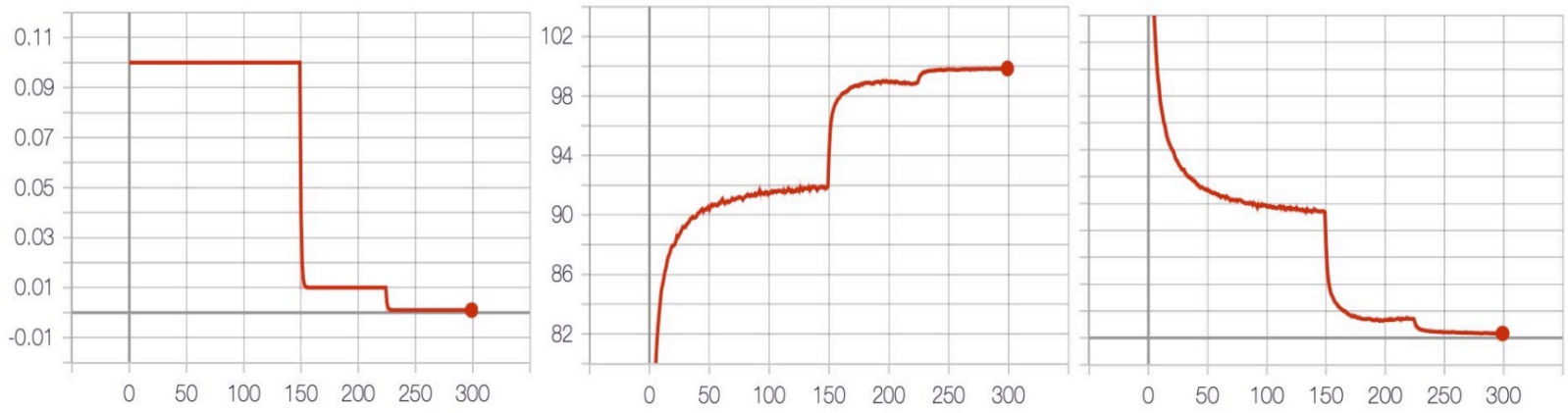
- 梯度下降对参数的初始值较为敏感，好的初始值可以带来更快更好的收敛效果

- 在训练MLP时需要将所有的权重初始化为小随机数，当初始化权重太大时，计算的梯度会累积多个大的权重，导致“梯度爆炸”

- 通常使用从高斯分布或均匀分布中随机抽取的值来初始化权重

学习率设置策略

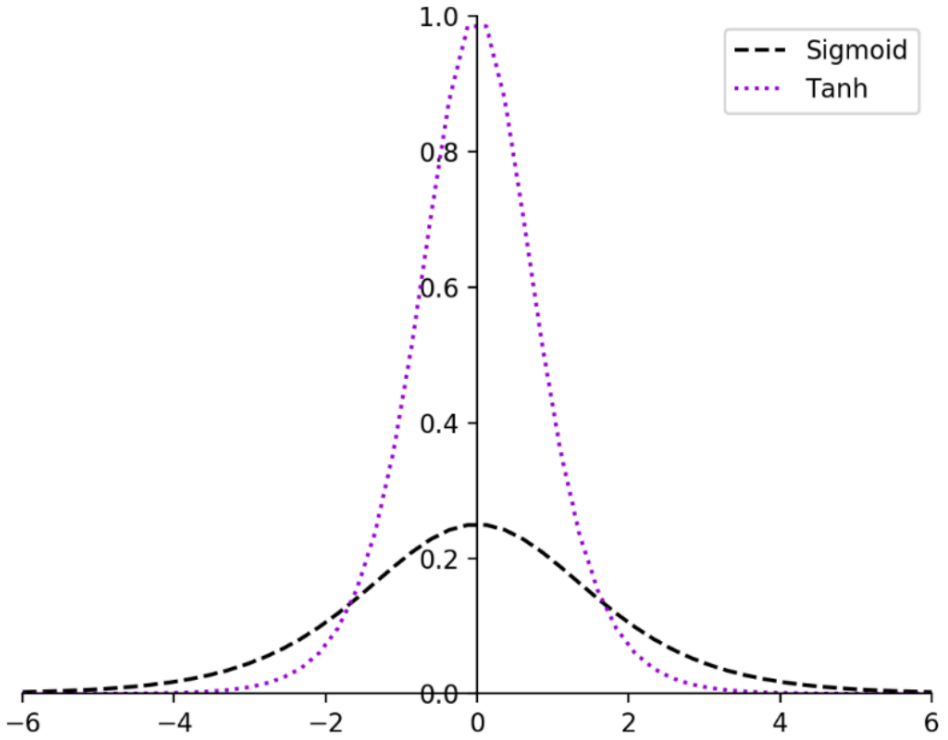
■ 模型训练初期需要较大的学习率来让它快速找到损失较小的区域，之后则需要恰当时机减小学习率，来让损失继续下降



学习率　　训练集准确率　　训练损失

存在的问题

- 梯度消失
  - sigmoid和tanh激活函数的导数取值如图，反向传播时多个导数相乘，导致梯度接近于0
- 解决方法
  - 使用ReLU系列激活函数

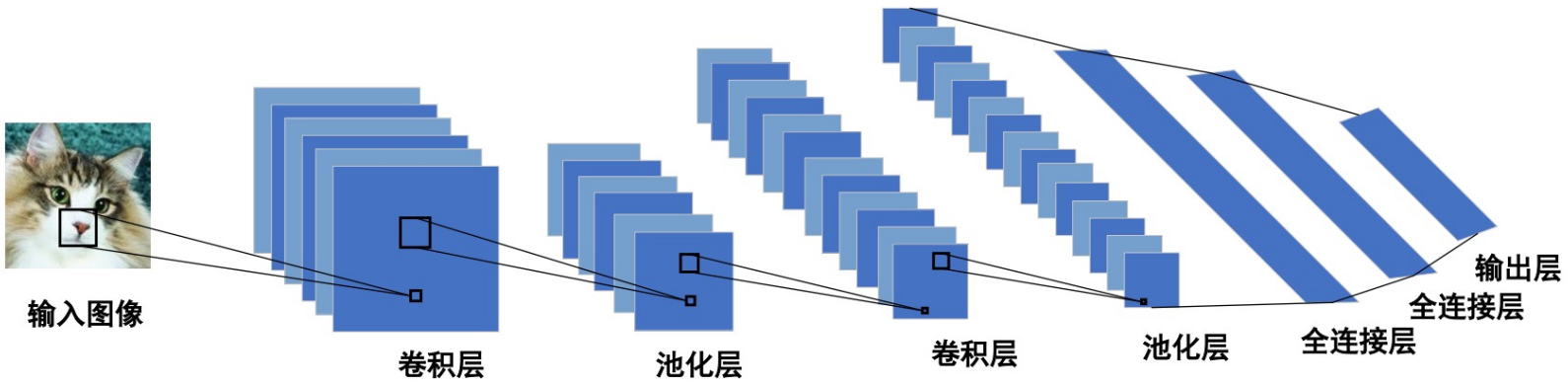


存在的问题

- 过拟合问题
  - 早停止策略：将数据划分为训练集和验证集，验证集用来估计误差；若训练集误差降低，验证集误差升高，则停止训练
  - 正则化策略：在损失函数中增加描述模型复杂度的部分，例如权重的平方和，以此来限制模型的学习能力，缓解过拟合
  - 增加训练数据：增加训练数据可以有效防止过拟合，因为大数据量可以抵消模型拟合能力过强的问题

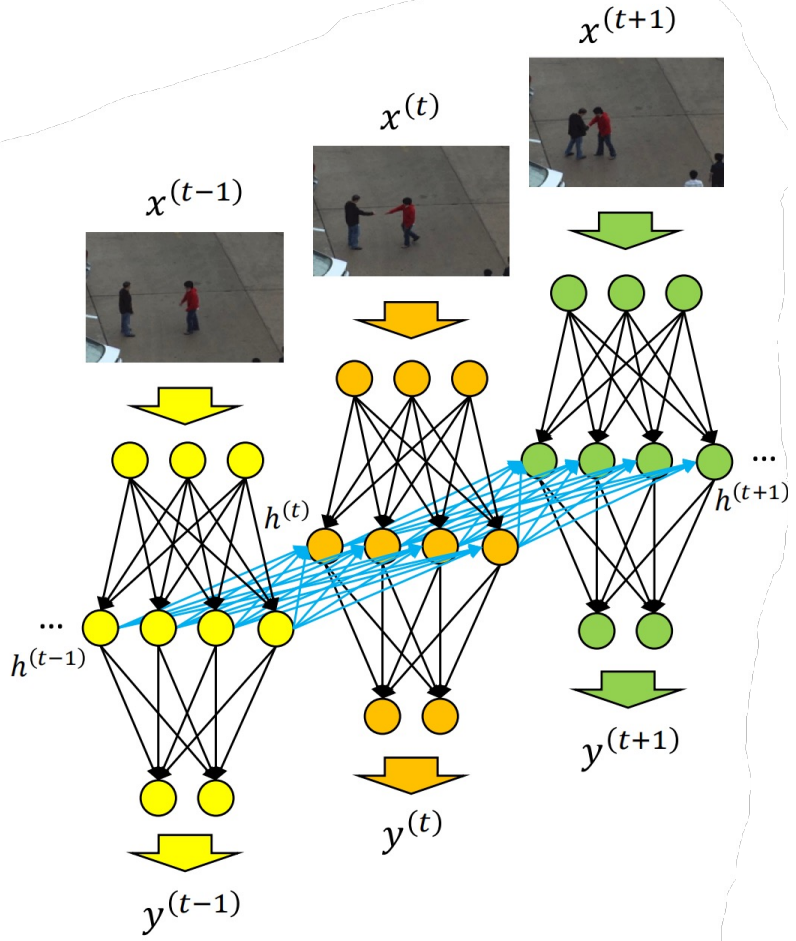
其他类型神经网络

- 卷积神经网络（Convolutional Neural Networks，简称CNN）
  - CNN是专门用于处理网格化数据（图片等）的网络，在图像分类中首次取得重大成功
  - 不同于全连接神经网络，CNN的卷积核每次只计算局部的特征，因此卷积网络能更多地关注图片的局部特性
  - CNN大大减少了全连接神经网络的参数量



其他类型神经网络

- 循环神经网络（Recurrent Neural Network，简称RNN）
  - 是一类用于处理序列数据的神经网络
  - 允许网络出现环形结构，当前的输出不仅与当前的输入有关还与上一步的网络状态有关
  - 最终的输出结合了每个时间步的信息



小节

- **起源：**人工神经网络**的灵感来源于**生物神经网络。
- **历史：**神经网络的发展**是AI历史的核心脉络**，至今已**经历了超过80年的起伏。**
- **核心训练技术：**模型训练的关键在于使用**反向传播** 这一高效算法。
- **深度学习的本质：**通过**堆叠**多层感知器（MLP）来增强模型的**函数拟合能力**与泛化能力，**这种方法构成了深度学习的基础。**
- **三要素：**深度学习的三大核心要素是：**训练数据、优化算法（优化器）和损失函数。**

谢谢！