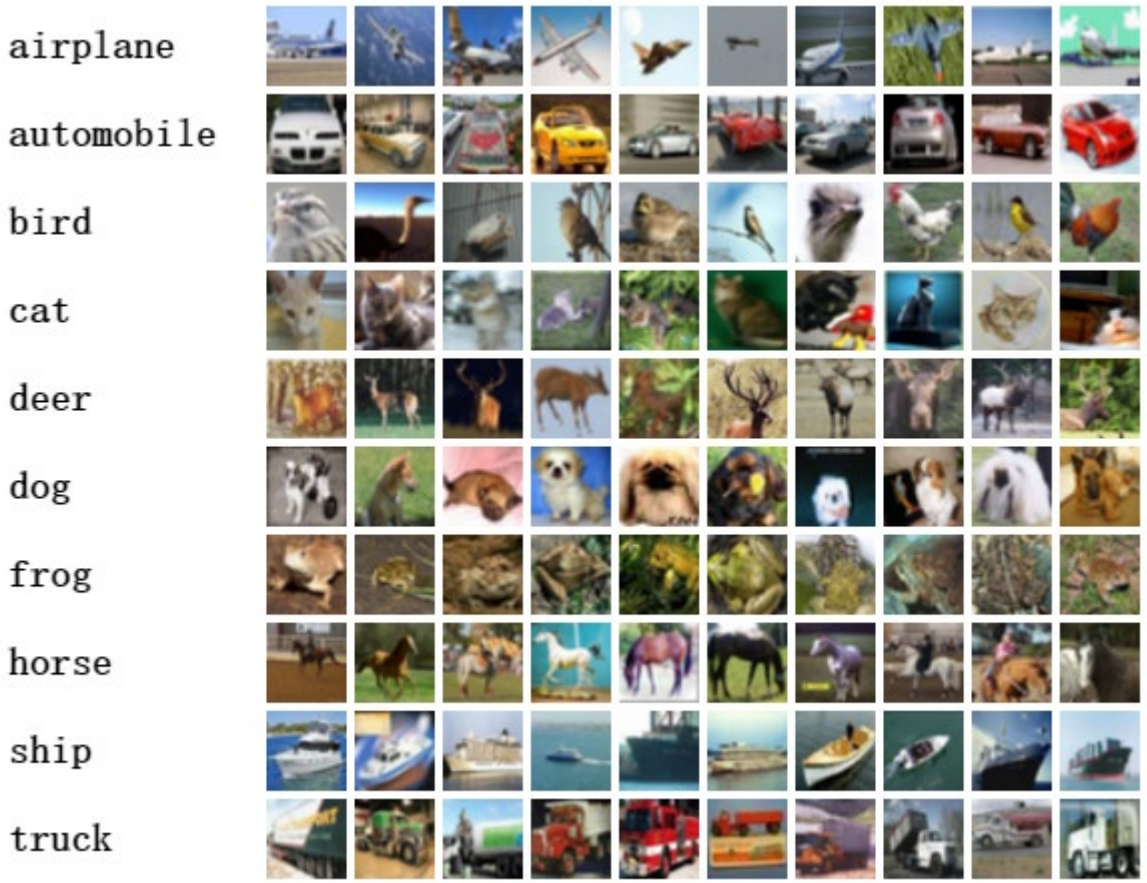
第2章
前馈神经网络

分类问题示例

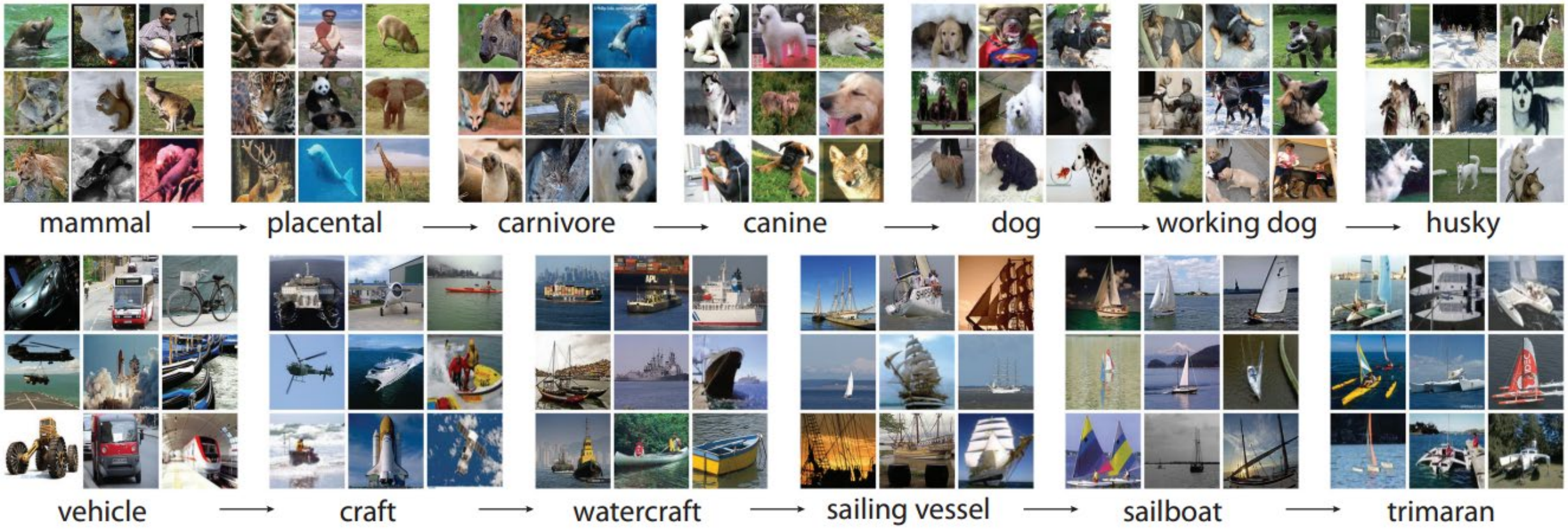
数据集：CIFAR-10

- 60000张32x32色彩图像，共10类，每类6000张图像。

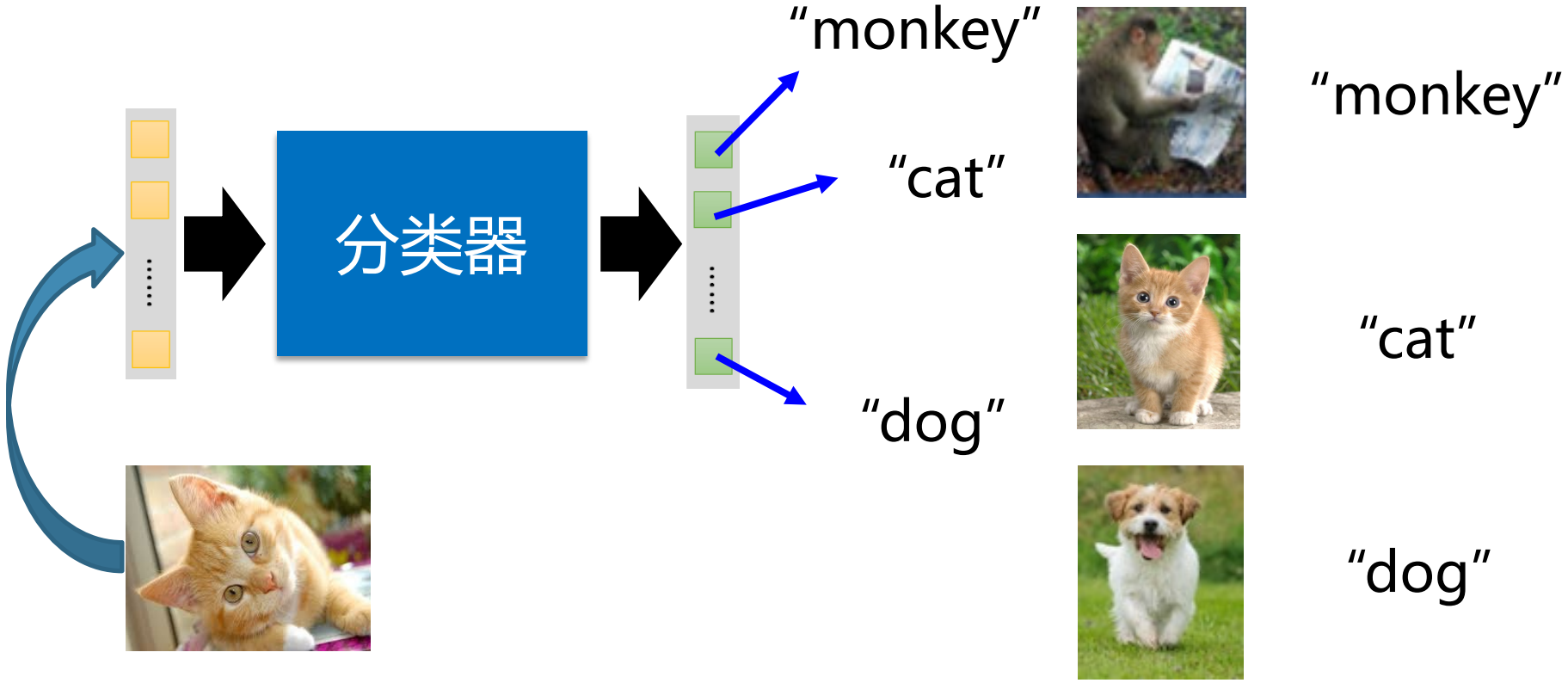


数据集：ImageNet

- 14,197,122 images, 21841 synsets



图像分类



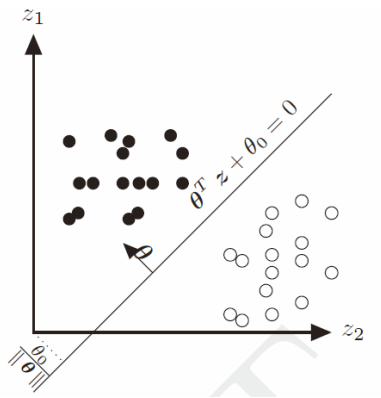
文本分类

根据文本内容来判断文本的相应类别

D_1：“我喜欢读书”

D_2：“我讨厌读书”

	我	喜欢	讨厌	读书
D_1	1	1	0	1
D_2	1	0	1	1

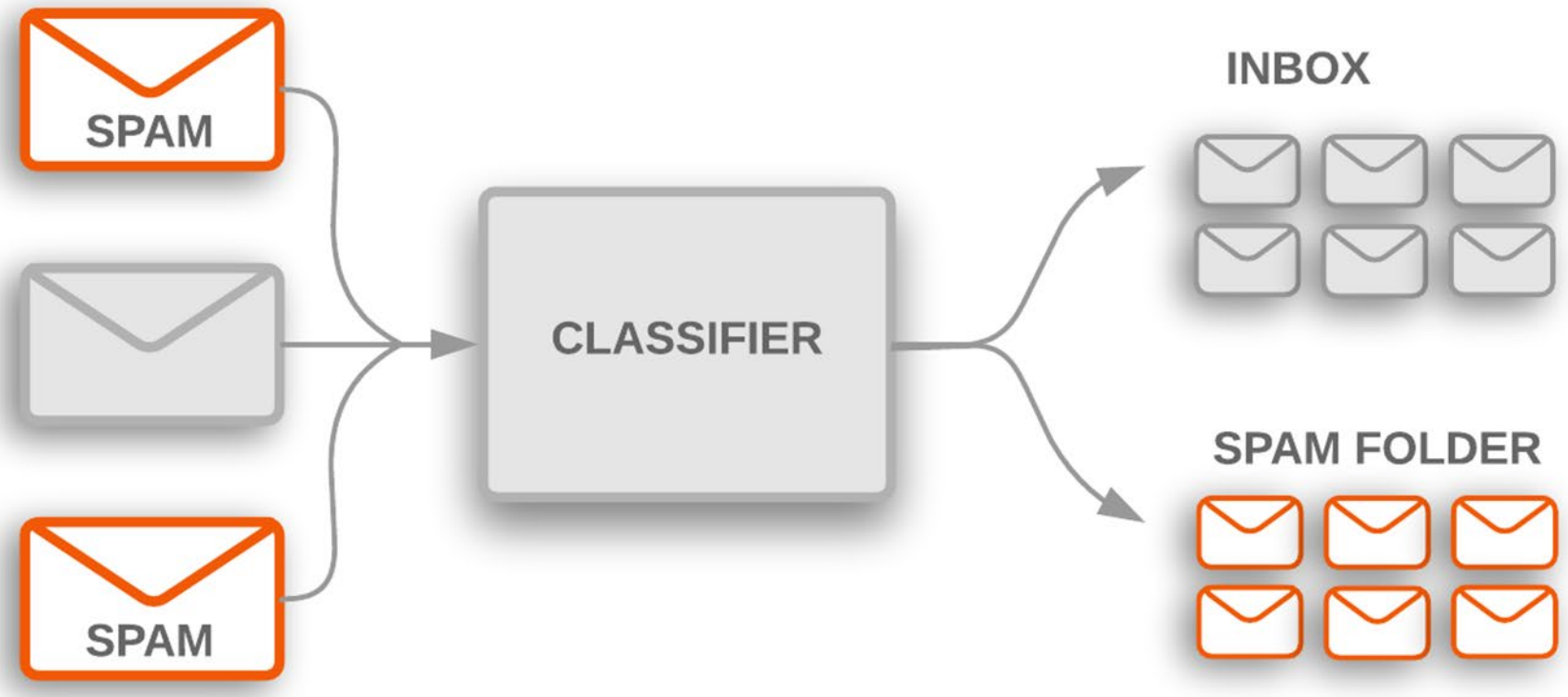


\+

\-

文档归类

垃圾邮件过滤



感知器

感知器

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

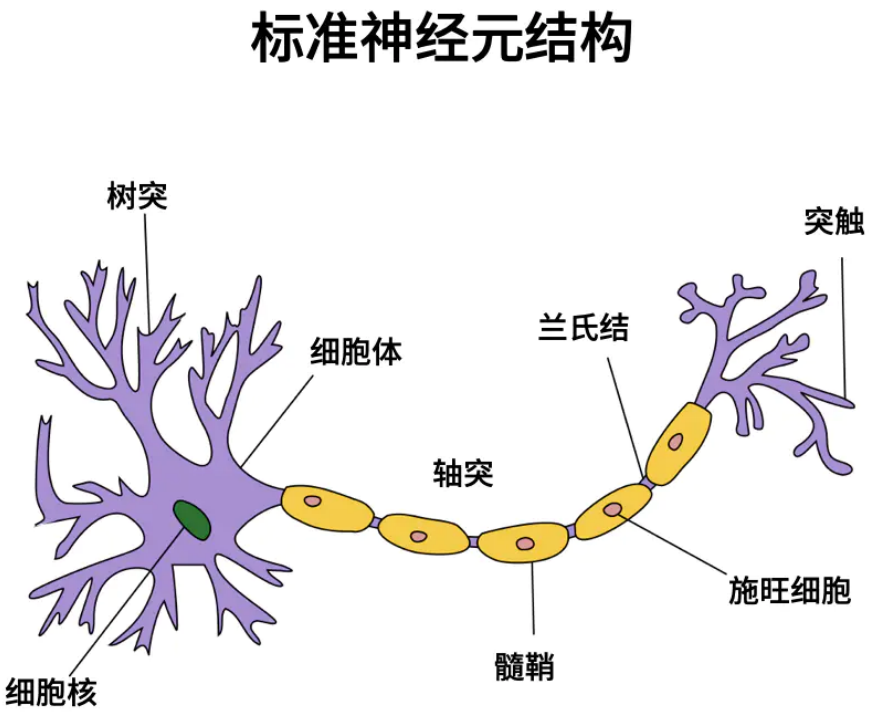
HAVING told you about the giant digital computer known as I.B.M. 704 and how it has been taught to play a fairly creditable game of chess, we'd like to tell you about an even more remarkable machine, the perceptron, which, as its name implies, is capable of what amounts to original thought. The first perceptron has yet to be built,

The New Yorker, December 6, 1958 P. 44

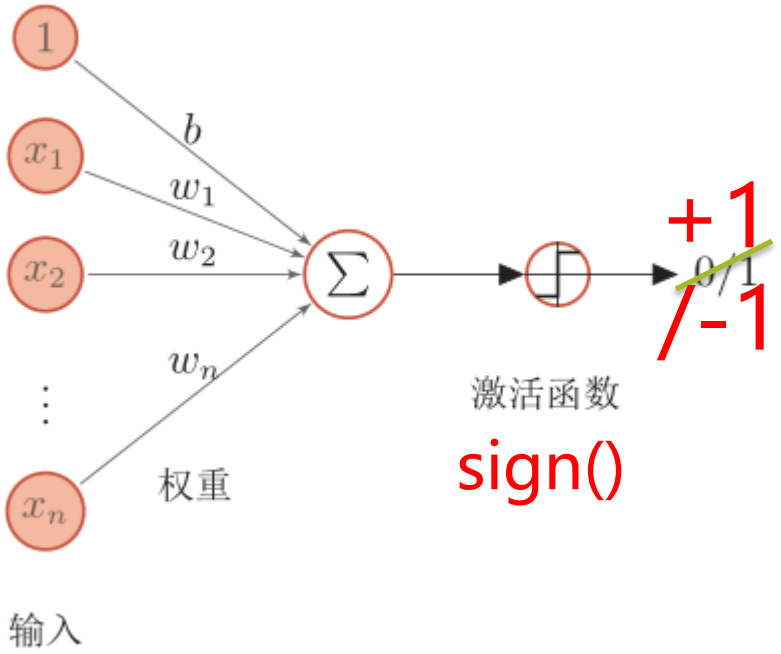
The IBM 704 computer

感知器

- 模拟生物神经元行为的机器，有与生物神经元相对应的部件，如权重（突触）、偏置（阈值）及激活函数（细胞体），输出为+1或-1。



$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \\ -1 & \text{当} \mathbf{w}^{\mathrm{T}}\mathbf{x} \leq 0 \end{cases},$$



感知器

- **学习算法**
  - 一种错误驱动的在线学习算法：
  - 先初始化一个权重向量$\boldsymbol{w} \leftarrow 0$（通常是全零向量）；
  - 每次分错一个样本$(\boldsymbol{x}, y)$时，即

$$y\boldsymbol{w}^T\boldsymbol{x} < \mathbf{0}$$

  - 用这个样本来更新权重

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$$

  - 根据感知器的学习策略，可以反推出感知器的损失函数为

$$\mathcal{L}(\boldsymbol{w}; \boldsymbol{x}, y) = \max(0, -y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$$

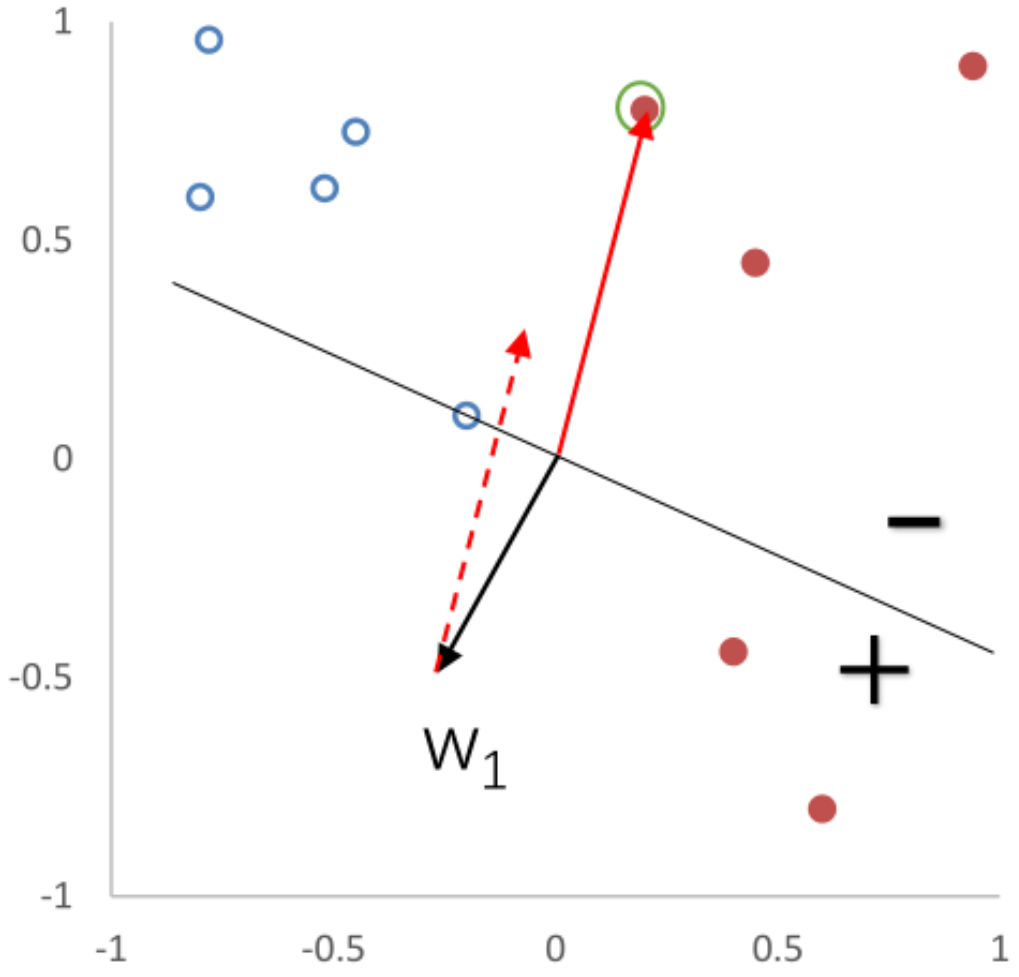
输入： 训练集：$(\mathbf{x}_i, y_i), i = 1, \cdots, N$，迭代次数：$T$

初始化：$\mathbf{w}_0 = 0$；
$k = 0$；
for $t = 1 \cdots T$ **do**
  for $i = 1 \cdots N$ **do**
    选取一个样本$(\mathbf{x}_i, y_i)$，**if** $\mathbf{w}^{\mathrm{T}}(y_i\mathbf{x}_i) < 0$ **then**
      $\mathbf{w}_{k+1} = \mathbf{w}_k + y_i\mathbf{x}_i$，；
      $k = k + 1$;
    end
  end
end
输出： $\mathbf{w}_k$

表示分错

感知器参数学习的更新过程

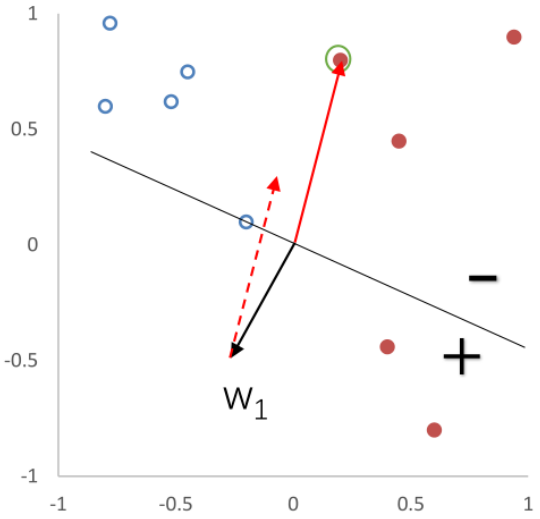
$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$$

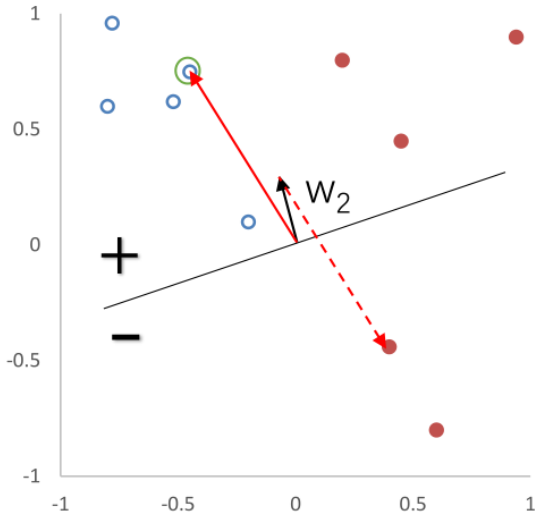


- 红色实心点为正例
- 蓝色空心点为负例
- 黑色箭头表示权重向量
- 红色虚线箭头表示权重的更新方向

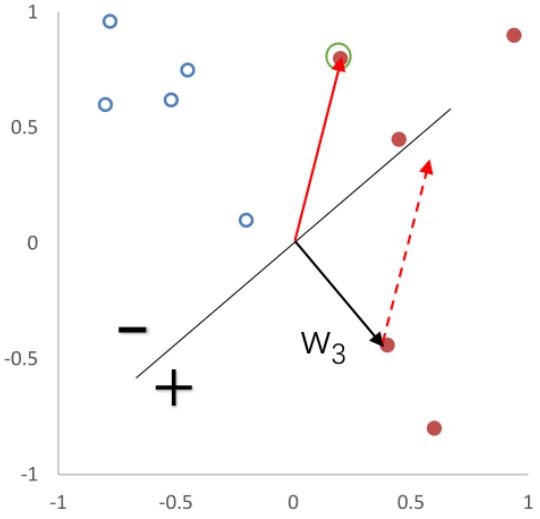
感知器参数学习的更新过程

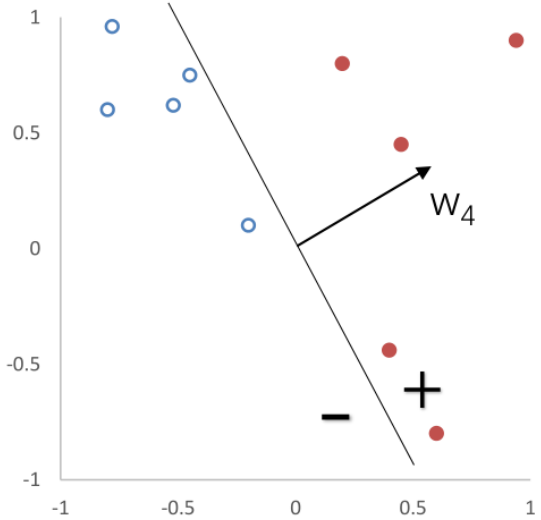
$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$$





- 红色实心点为正例
- 蓝色空心点为负例
- 黑色箭头表示权重向量
- 红色虚线箭头表示权重的更新方向





收敛性

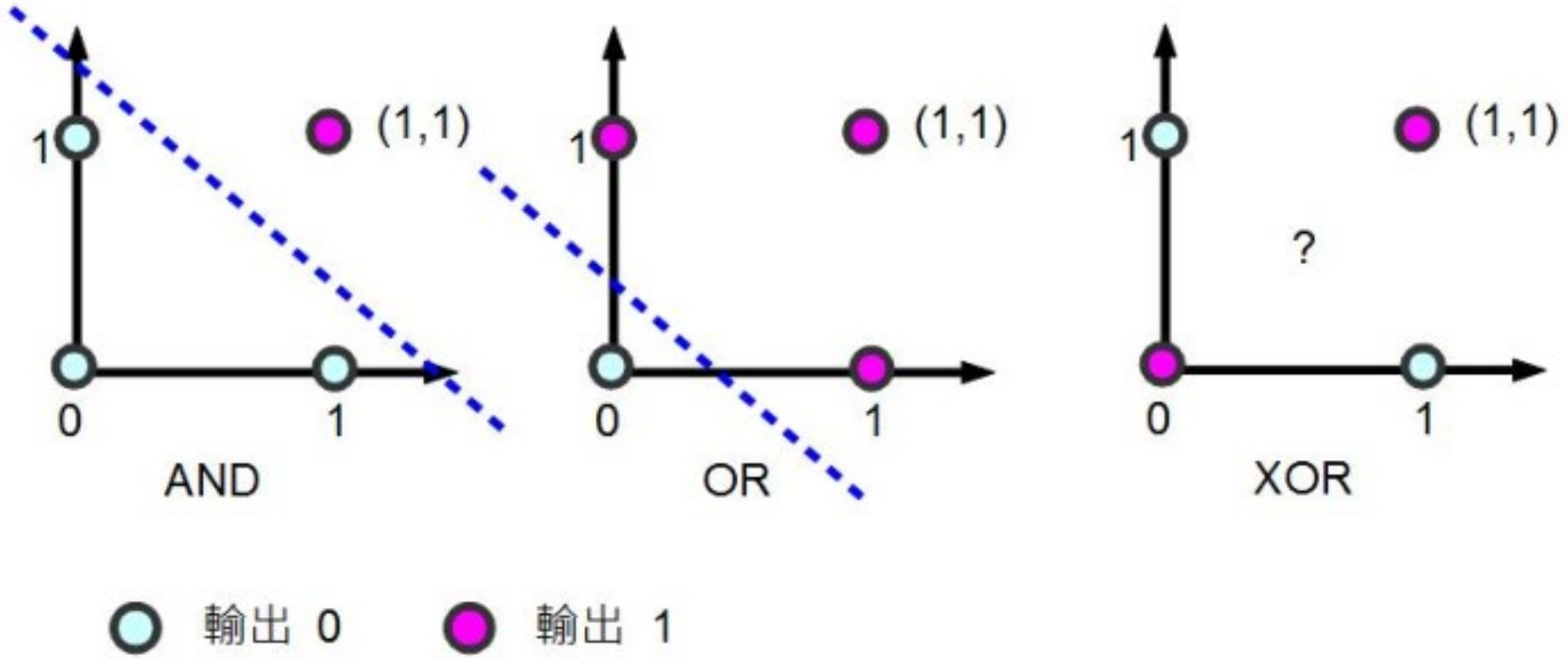
定义 3.1 – 两类线性可分： 对于训练集 $\mathcal{D} = \left\{(\mathbf{x}^{(n)}, y^{(n)})\right\}_{n=1}^{N}$，如果存在权重向量 $\mathbf{w}^*$，对所有样本都满足 $yf(\mathbf{x}; \mathbf{w}^*) > 0$，那么训练集 $\mathcal{D}$ 是线性可分的。

定理 3.1 – 感知器收敛性： 给定一个训练集 $\mathcal{D} = \left\{(\mathbf{x}^{(n)}, y^{(n)})\right\}_{n=1}^{N}$，假设 R 是训练集中最大的特征向量的模，

$$R = \max_n \|x^{(n)}\|.$$

如果训练集 $\mathcal{D}$ 线性可分，感知器学习算法3.1的权重更新次数不超过 $\frac{R^2}{\gamma^2}$。

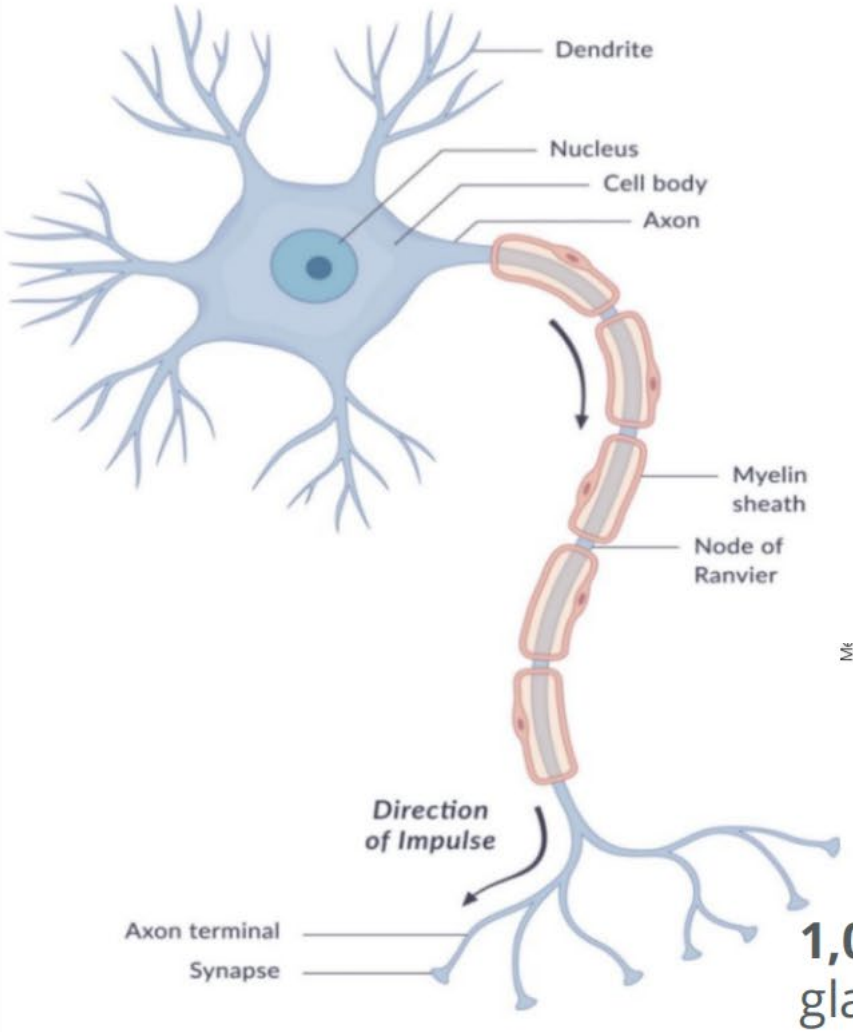
XOR问题



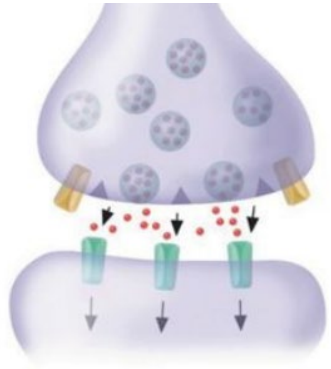
神经网络

生物神经元

1,000's of inputs (other neurons, sensory neurons e.g. taste buds from a salt or sugar molecule...)



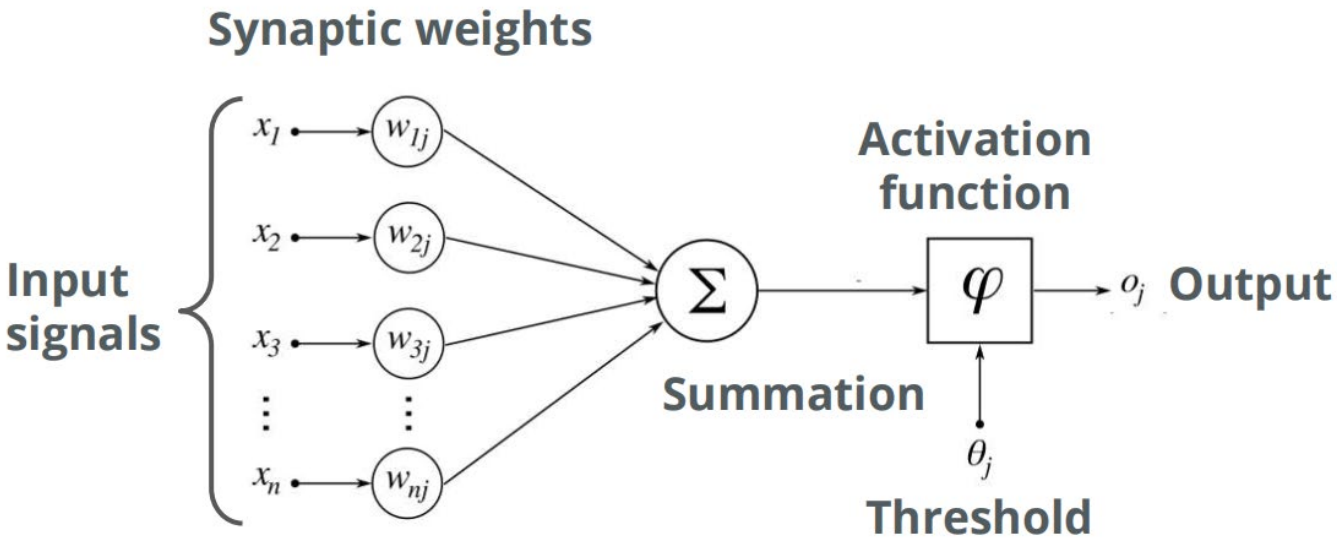
- 细胞体（Soma）中的神经细胞膜上有各种受体和离子通道，胞膜的受体可与相应的化学物质神经递质结合，引起离子通透性及膜内外电位差发生改变，产生相应的生理活动：兴奋或抑制。
- 细胞突起是由细胞体延伸出来的细长部分，又可分为树突和轴突。
  - 树突（Dendrite）可以接受刺激并将兴奋传入细胞体。每个神经元可以有一或多个树突。
  - 轴突 (Axons) 可以把自身的兴奋状态从胞体传送到另一个神经元或其他组织。每个神经元只有一个轴突。



1,000's of output targets (e.g. other neurons, muscle cells, gland cells, blood vessels to release hormones...)

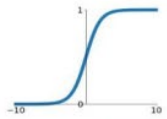
Figure by wetcake (left) and Andrej Kral (right)

人工神经元

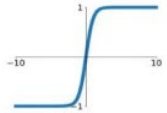


$$z = \sum_{i=1}^{d} w_i x_i + b$$
$$= \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b,$$

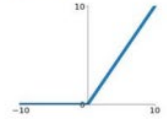
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



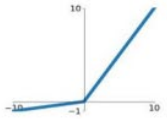
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

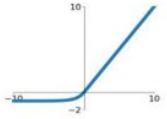


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

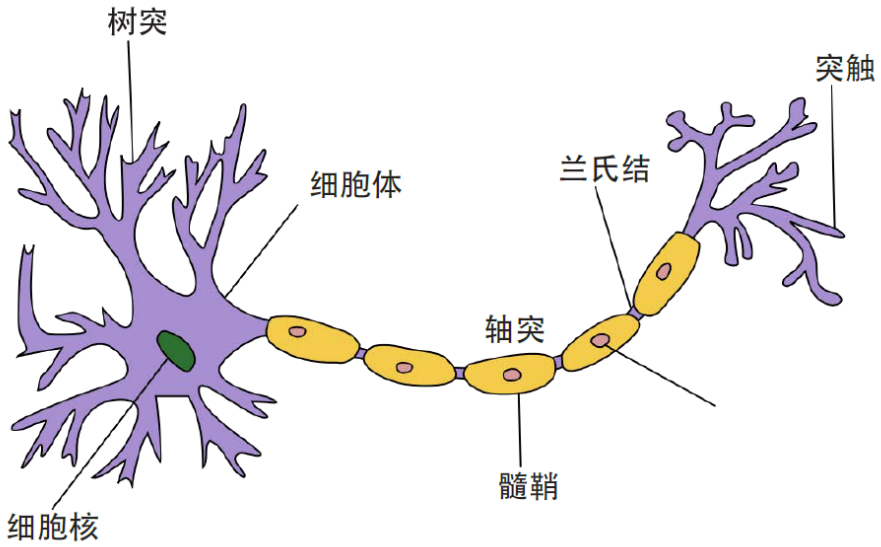
ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



$$a = f(z)$$

神经网络

- **分布式并行处理(Parallel Distributed Processing, PDP)网络**
  - 神经网络最早是作为一种主要的连接主义模型。
  - 20世纪80年代后期，最流行的一种连接主义模型
  - 其有3个主要特性：
    - （1）信息表示是分布式的（非局部的）；
    - （2）记忆和知识是存储在单元之间的连接上；
    - （3）通过逐渐改变单元之间的连接强度来学习新的知识。



- **引入误差反向传播来改进其学习能力之后，**
- **神经网络也越来越多地应用在各种机器学习任务上。**

复习

- **辨析下面的问题是回归问题还是分类问题：**
  - 工程师正在训练模型，用于自动驾驶汽车的制动系统。模型的输入是一组来自前视摄像头的实时画面。模型需要计算一个关键指标：当前车速下，本车与前方障碍物之间的“碰撞时间”。一旦系统计算出“碰撞时间”低于 1.5 秒，则触发红灯警报；如果低于 0.8 秒，则直接触发刹车。训练一个模型来直接输出这个“碰撞时间”数值，属于什么类型的问题？
  - 银行开发了一套实时交易监测系统。当一笔交易发生时，模型需要输出风险评级，以估计交易涉嫌欺诈的可能性。这属于哪种类型的问题？
- **什么是表示学习？**
- **深度学习与一般机器学习有何区别？**
- **人工神经元是如何模拟生物神经元的工作的？**
- **简述感知器的学习算法。**
- **神经网络与深度学习有何关联？**

激活函数的性质

连续可导

- 连续并可导（允许少数点上不可导）的非线性函数。
- 可导的激活函数可以直接利用数值优化的方法来学习网络参数。

函数简单

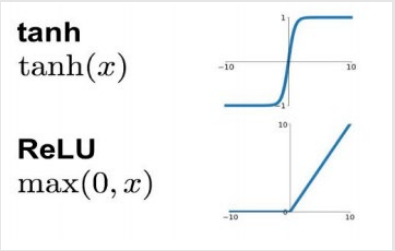
- 激活函数及其导函数要尽可能的简单
- 有利于提高网络计算效率。

值域有限

- 激活函数的导函数的值域要在一个合适的区间内
- 不能太大也不能太小，否则会影响训练的效率和稳定性。

单调递增?

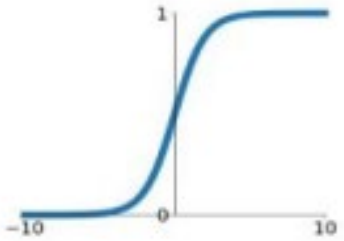
- ?



常见激活函数

Sigmoid

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



tanh

$$\tanh(x)$$

ReLU

$$\max(0, x)$$

Leaky ReLU

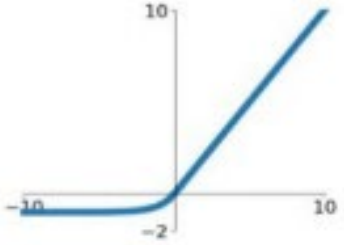
$$\max(0.1x, x)$$

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

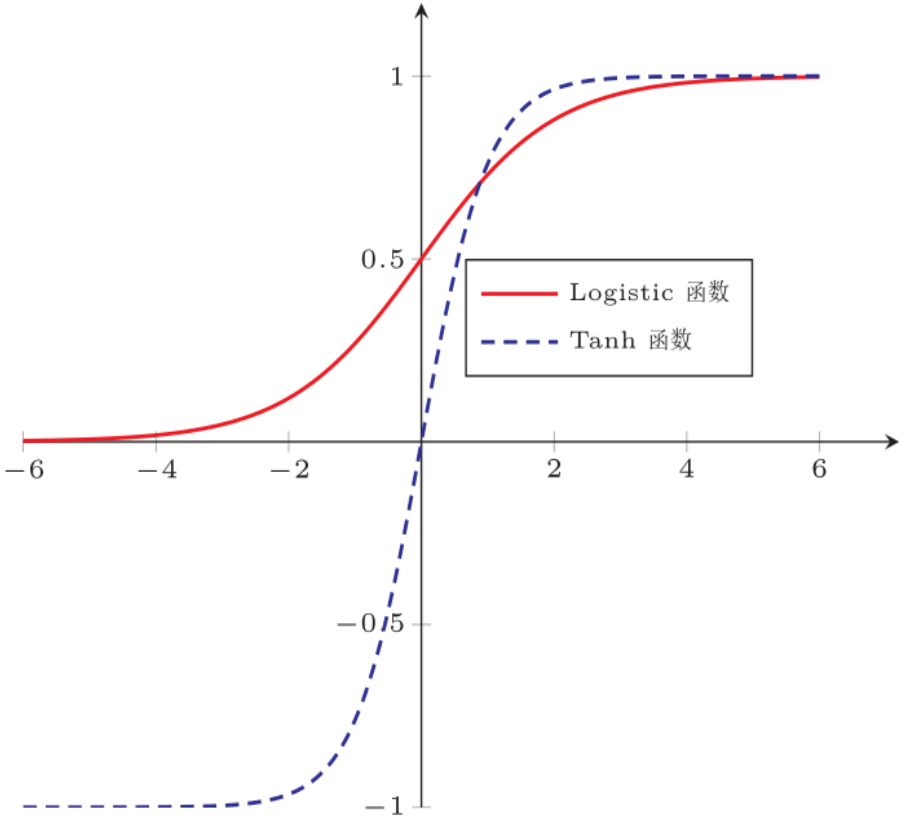


Sigmoid和Tanh函数

- **Logistic & Tanh**

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



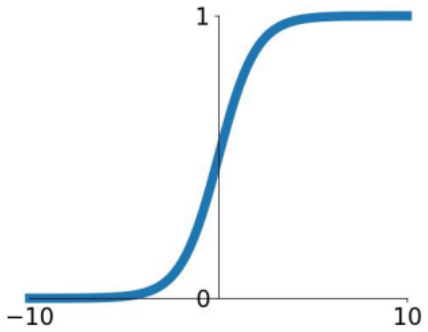
- **性质：**
  - 饱和函数
  - Tanh函数是零中心化的，而logistic函数的输出恒大于0

- **偏置偏移（bias shift）**
  - 非零中心化的输出会使得其后一层的神经元的输入发生，并进一步使得梯度下降的收敛速度变慢。

Sigmoid和Tanh函数

Activation Functions

$$\sigma(x) = 1/(1+e^{-x})$$



Sigmoid

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

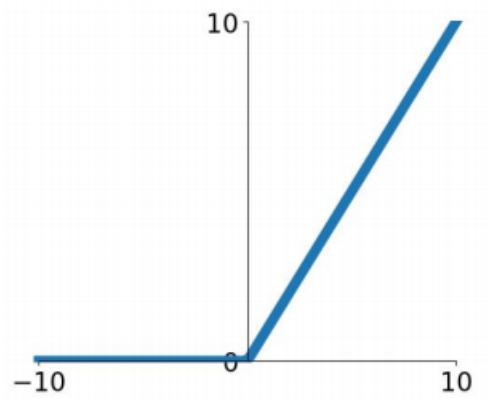
3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1+\exp(-x)}$	$f'(x) = f(x)\big(1 - f(x)\big)$

ReLU函数

Activation Functions



ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

[Krizhevsky et al., 2012]

$$\text{ReLU} \qquad f(x) = \max(0, x) \qquad f'(x) = I(x > 0)$$

ReLU函数

- **ReLU系列**

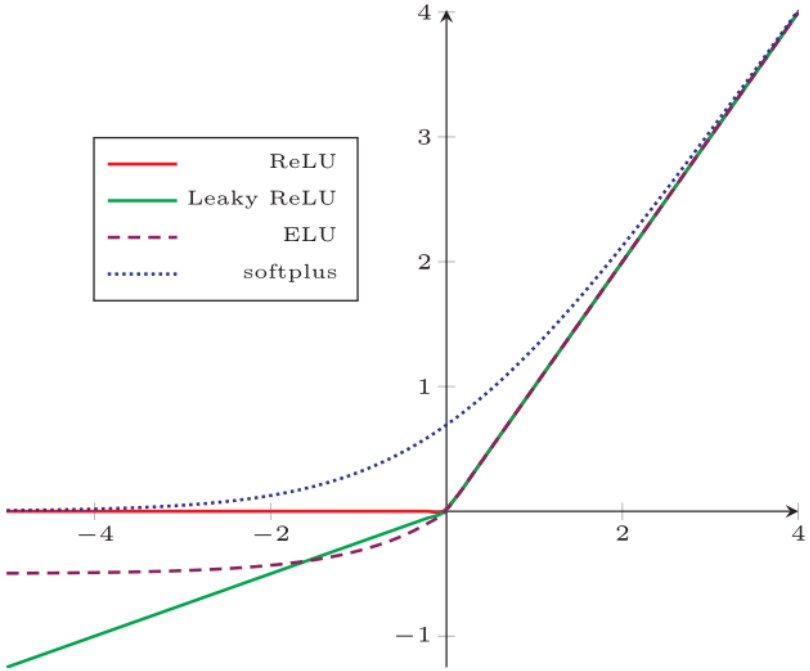
$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} = \max(0, x).$$

$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases} = \max(0, x) + \gamma \min(0, x)$$

$$\mathrm{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases} = \max(0, x) + \gamma_i \min(0, x)$$

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

$$\mathrm{softplus}(x) = \log(1 + \exp(x))$$



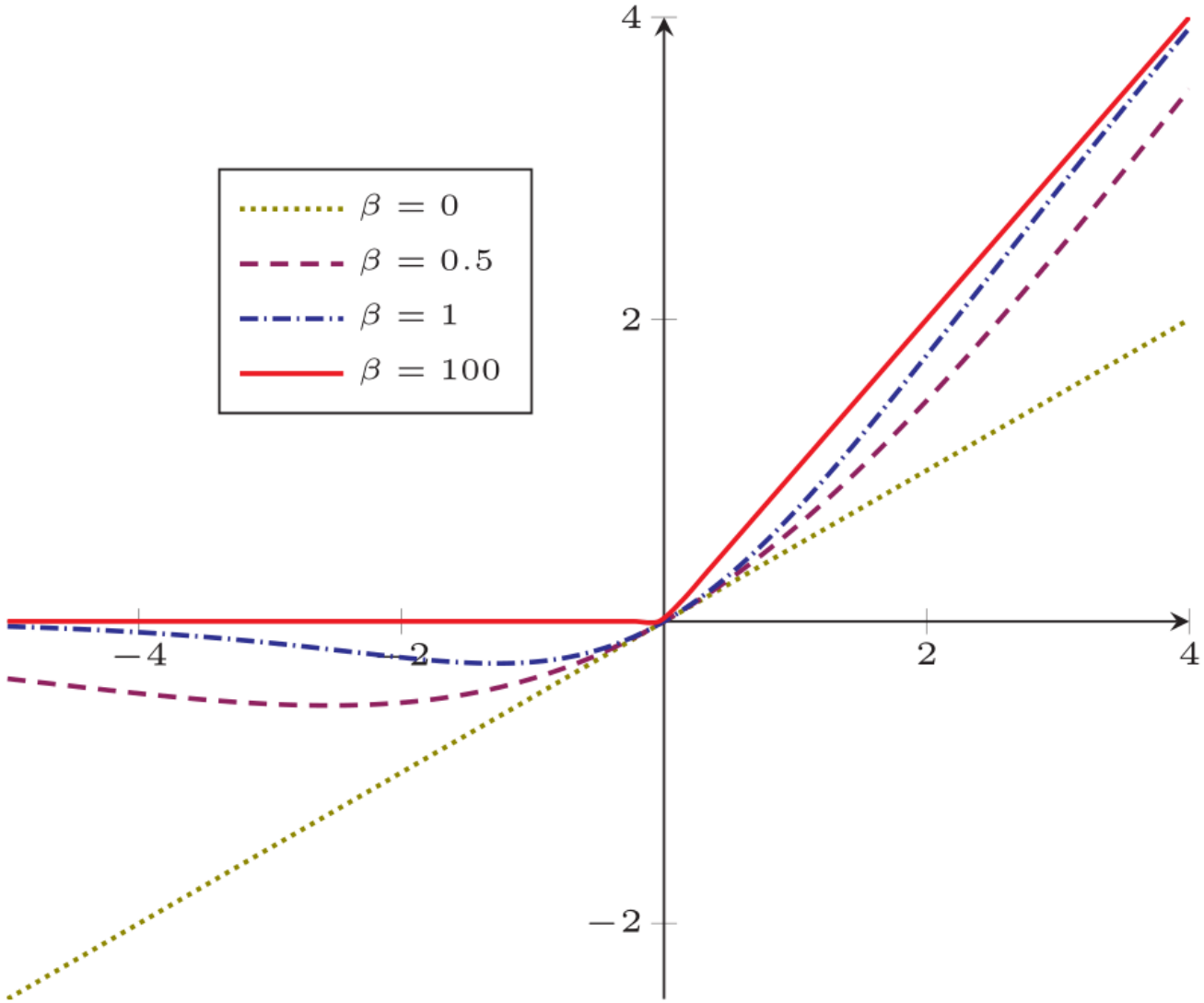
$$f'(x) = I(x > 0)$$

死亡ReLU问题（Dying ReLU Problem）

- 计算上更加高效。
- 生物上的解释性
  - 单侧抑制、宽兴奋边界
- 在一定程度上缓解梯度消失问题

Swish函数

$$\text{swish}(x) = x\sigma(\beta x)$$



常见激活函数及其导数

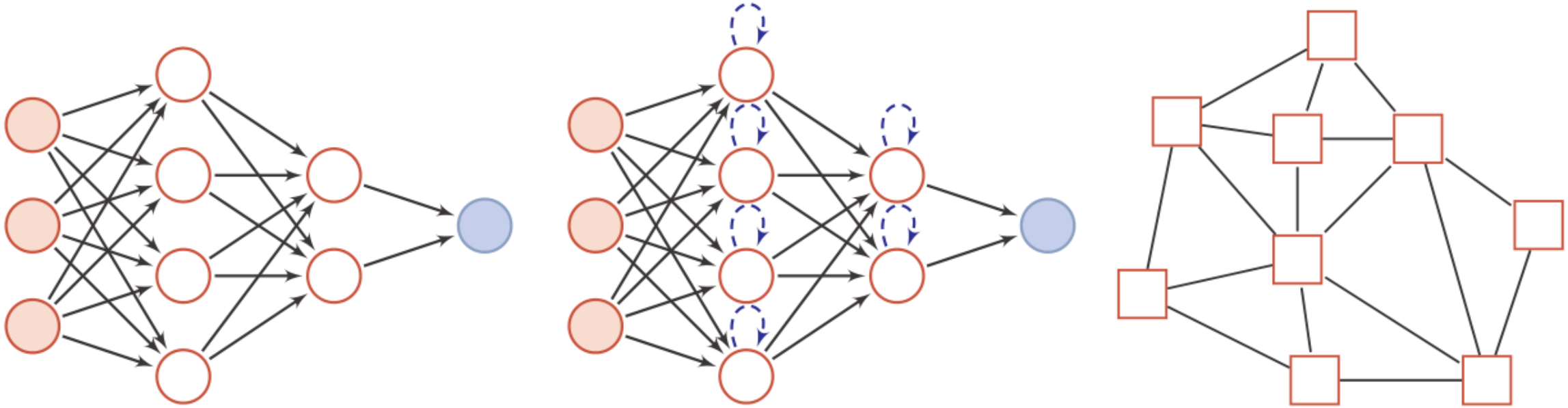
激活函数	函数	导数
Logistic 函数	$f(x)=\frac{1}{1+\exp(-x)}$	$f'(x)=f(x)\big(1-f(x)\big)$
Tanh 函数	$f(x)=\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x)=1-f(x)^2$
ReLU	$f(x)=\max(0,x)$	$f'(x)=I(x>0)$
ELU	$f(x)=\max(0,x)+\min\big(0,\gamma(\exp(x)-1)\big)$	$f'(x)=I(x>0)+I(x\leq 0)\cdot\gamma\exp(x)$
SoftPlus 函数	$f(x)=\log\big(1+\exp(x)\big)$	$f'(x)=\frac{1}{1+\exp(-x)}$

人工神经网络三要素

- **人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：**
  - 神经元的激活规则
    - 主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
  - 网络的拓扑结构
    - 不同神经元之间的连接关系。
  - 学习算法
    - 通过训练数据来学习神经网络的参数。

网络结构

- 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。



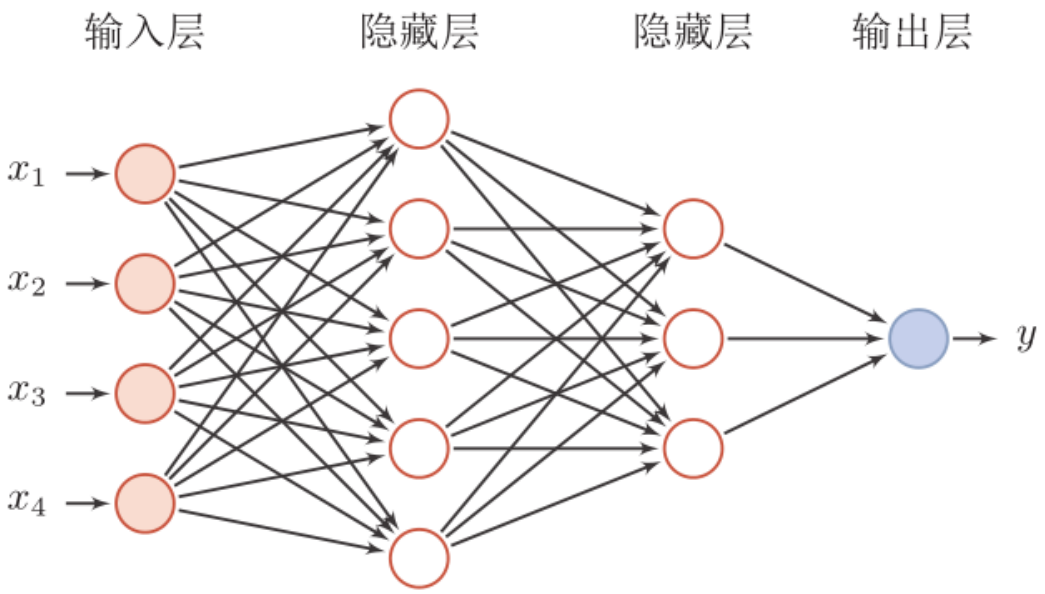
(a) 前馈网络　　(b) 记忆网络　　(c) 图网络

圆形节点表示一个神经元，方形节点表示一组神经元。

前馈神经网络

网络结构

- **前馈神经网络（全连接神经网络、多层感知器）**
  - 各神经元分别属于不同的层，层内无连接。
  - 相邻两层之间的神经元全部两两连接。
  - 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



- L：表示神经网络的层数；
- n^l：表示第 l 层神经元的个数；
- $f_l(\cdot)$：表示 l 层神经元的激活函数；
- $W^{(l)} \in \mathbb{R}^{n^l \times n^{l-1}}$：表示 $l-1$ 层到第 l 层的权重矩阵；
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n^l}$：表示 $l-1$ 层到第 l 层的偏置；
- $\mathbf{z}^{(l)} \in \mathbb{R}^{n^l}$：表示 l 层神经元的净输入（净活性值）；
- $\mathbf{a}^{(l)} \in \mathbb{R}^{n^l}$：表示 l 层神经元的输出（活性值）。

信息传递过程

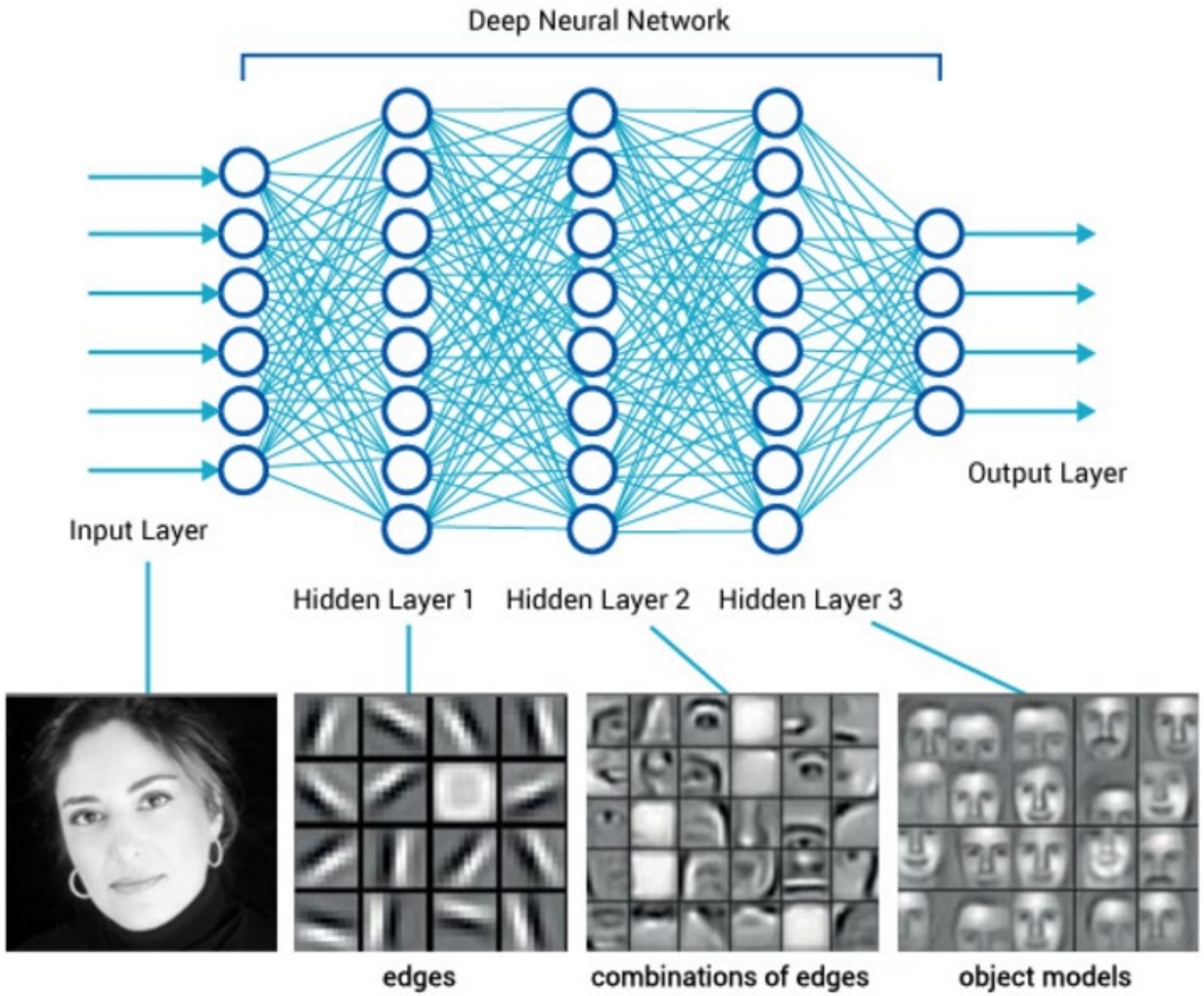
- 前馈神经网络通过下面公式进行信息传播。

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)},$$
$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)}).$$

- 前馈计算：

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)} = \phi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}))$$

深层前馈神经网络


Deep Neural Network
Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Output Layer
edges
combinations of edges
object models

通用近似定理

定理 4.1 – 通用近似定理（Universal Approximation Theorem）[Cybenko, 1989, Hornik et al., 1989]： 令 $\varphi(\cdot)$ 是一个非常数、有界、单调递增的连续函数，$\mathcal{I}_d$ 是一个 d 维的单位超立方体 $[0,1]^d$，$C(\mathcal{I}_d)$ 是定义在 $\mathcal{I}_d$ 上的连续函数集合。对于任何一个函数 $f \in C(\mathcal{I}_d)$，存在一个整数 m，和一组实数 $v_i, b_i \in \mathbb{R}$ 以及实数向量 $\mathbf{w}_i \in \mathbb{R}^d$，$i = 1, \cdots, m$，以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^{m} v_i \varphi(\mathbf{w}_i^{\mathrm{T}} \mathbf{x} + b_i), \tag{4.33}$$

作为函数 f 的近似实现，即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \tag{4.34}$$

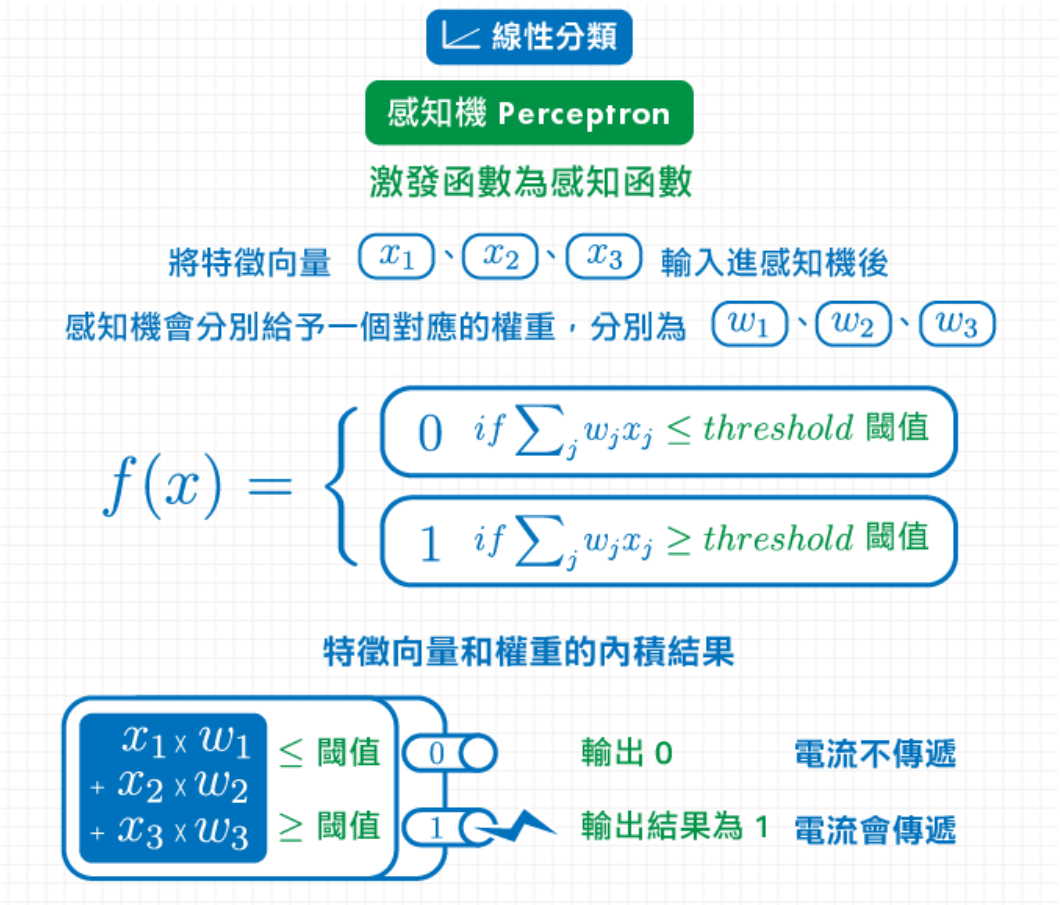
其中 $\epsilon > 0$ 是一个很小的正数。

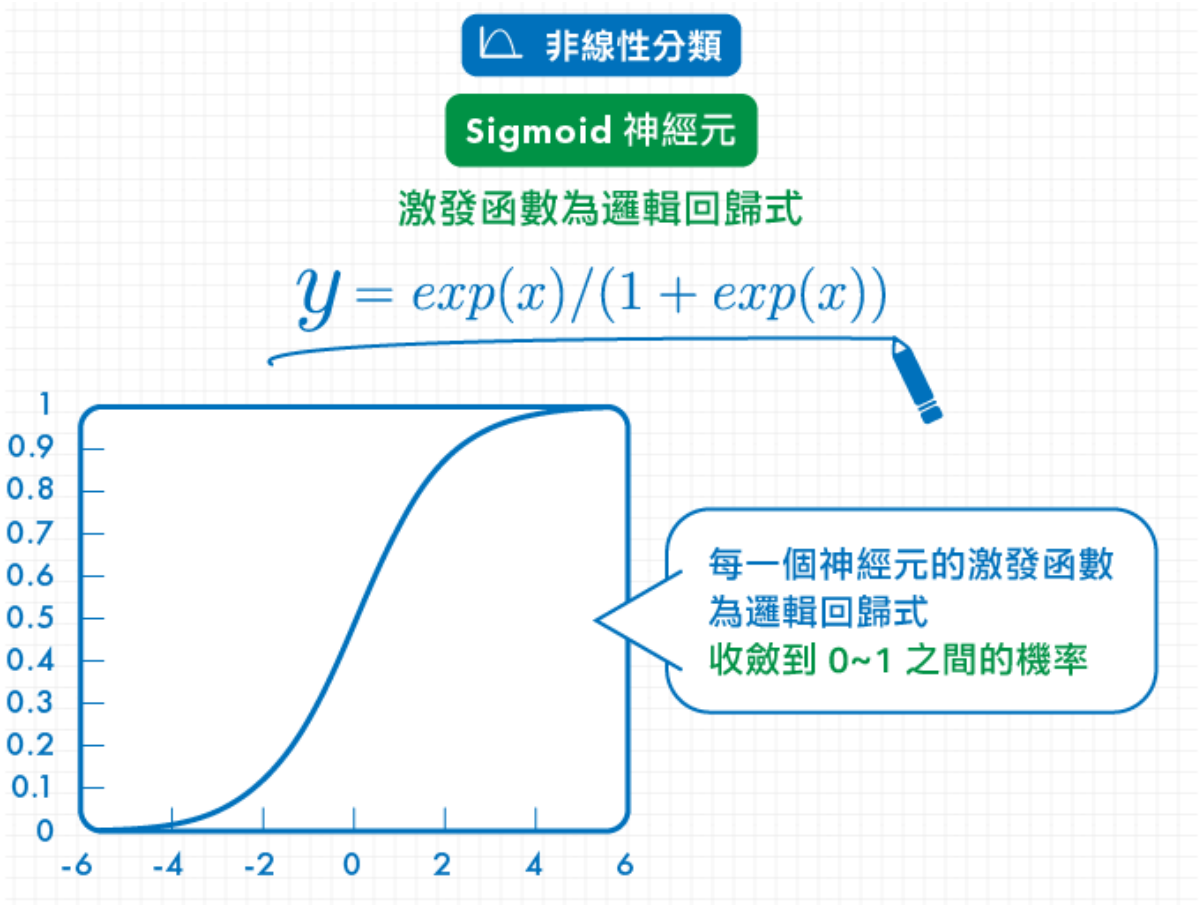
根据通用近似定理，对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络，只要其隐藏层神经元的数量足够，它可以以任意的精度来近似任何从一个定义在实数空间中的有界闭集函数。

应用到机器学习

- 神经网络可以作为一个“万能”函数来使用，可以用来进行复杂的特征转换，或逼近一个复杂的条件分布。

$$\hat{y} = g(\varphi(\mathbf{x}), \theta)$$





应用到机器学习

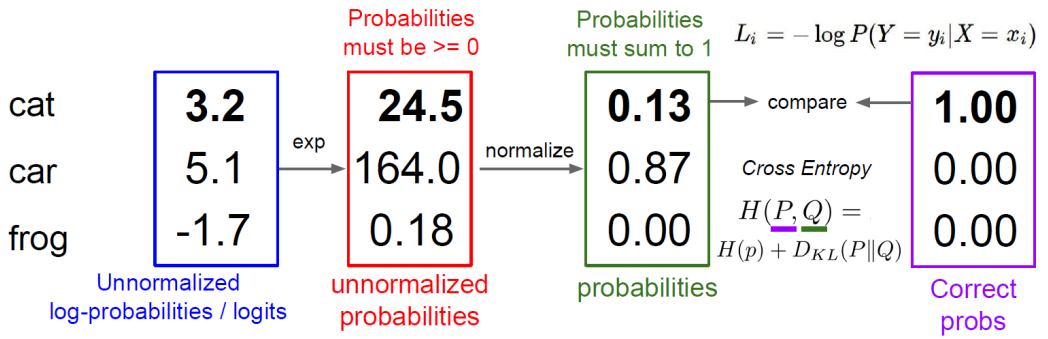
- **对于多类分类问题**
  - 如果使用softmax回归分类器，相当于网络最后一层设置C 个神经元，其输出经过softmax函数进行归一化后可以作为每个类的条件概率。

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(L)})$$

$$P(y = c|\mathbf{x}) = \text{softmax}(\mathbf{w}_c^{\mathrm{T}}\mathbf{x}) = \frac{\exp(\mathbf{w}_c^{\top}\mathbf{x})}{\sum_{i=1}^{C}\exp(\mathbf{w}_i^{\top}\mathbf{x})}.$$

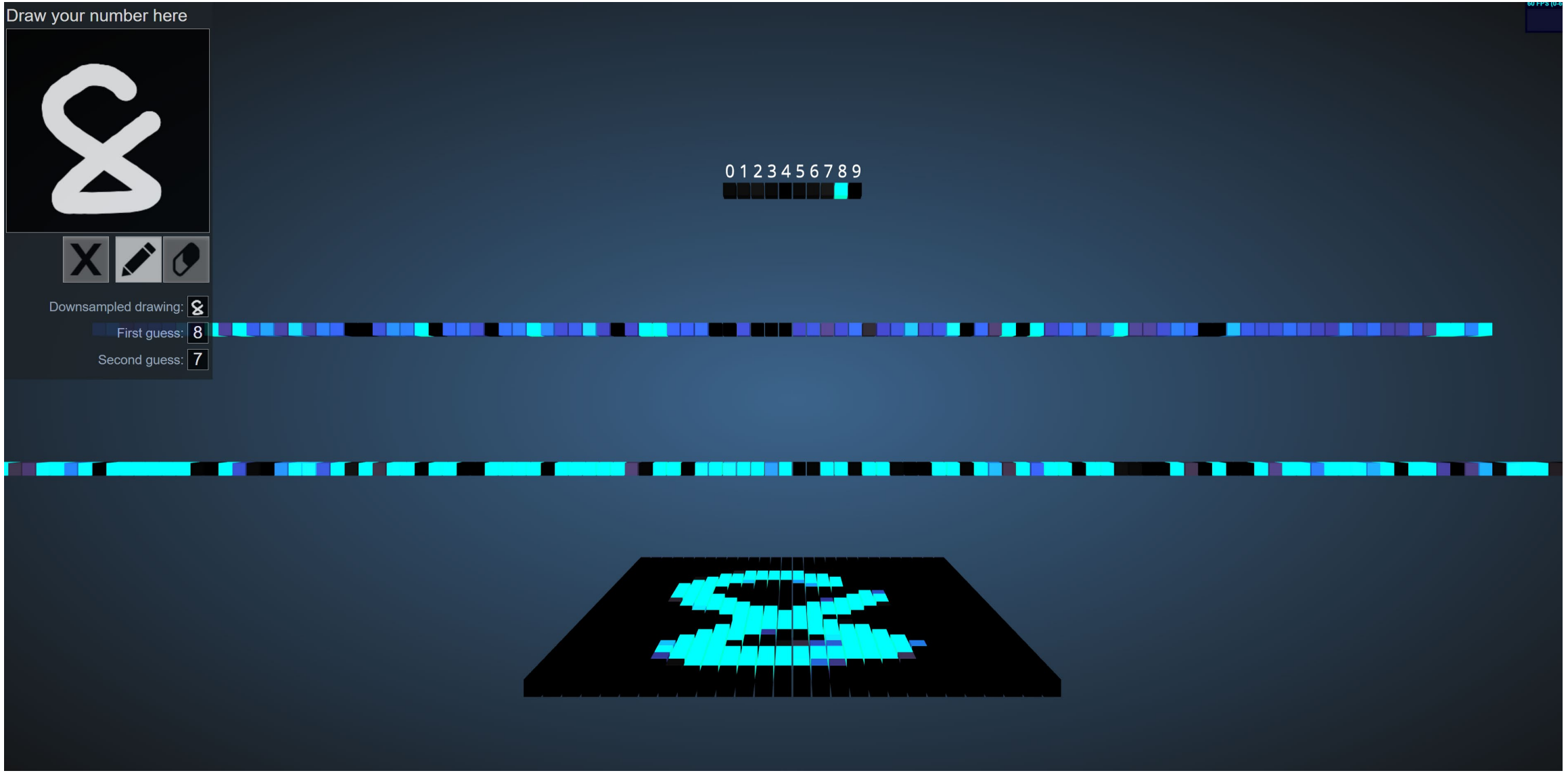
  - 采用交叉熵损失函数，对于样本(x,y)，其损失函数为

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\mathrm{T}} \log \hat{\mathbf{y}}$$



应用到机器学习

[3D Visualization of a Fully-Connected Neural Network (adamharley.com)]



参数学习

机器学习的四个要素

机器学习问题通常包含以下要素：

- **数据**
- **模型**
  - 线性方法： $f(\mathbf{x}, \theta) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$
  - 广义线性方法： $f(\mathbf{x}, \theta) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) + b$
    - 如果ϕ(x)为可学习的非线性基函数，f(x,θ)就等价于神经网络。
- **学习准则**
  - 期望风险： $\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}), y)],$

期望风险通常未知，可通过经验风险近似

$$\mathcal{R}_{\mathcal{D}}^{emp}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(y^{(n)}, f(x^{(n)}, \theta))$$

- **优化算法**
  - 梯度下降

参数学习

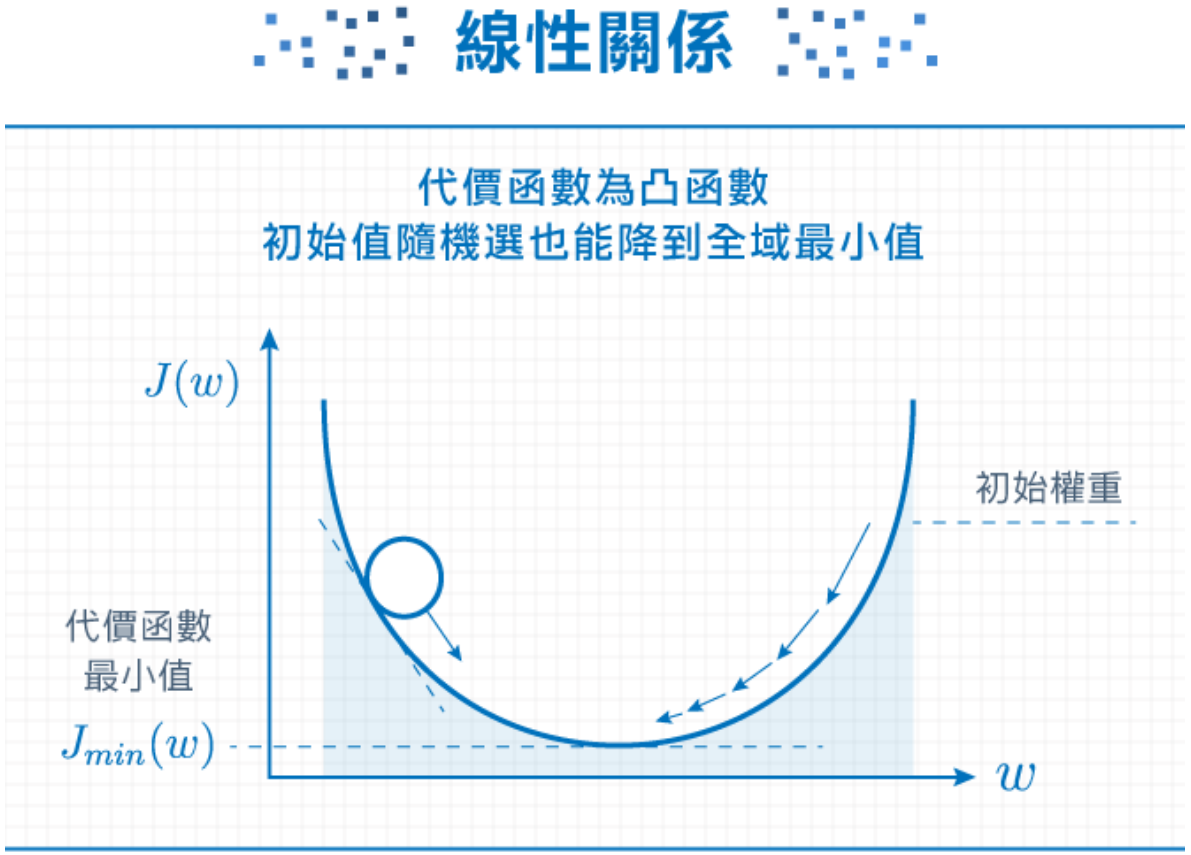
- 给定训练集为D，将每个样本输入给前馈神经网络，得到网络输出。其在数据集D上的结构化风险函数为：

$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2}\lambda\|W\|_F^2$$

- 梯度下降

$$W^{(l)} \leftarrow W^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$



损失函数

平方损失

- 定义：真实值与预测值的平方误差，也称L2 Loss；
- 形式：$L(y, f(x)) = \frac{1}{2}(y - f(x))^2$
- 应用：用于回归问题。

0-1损失

- 定义：预测值和真实值不相等时为1，否则为0；
- 形式：$L(Y, f(X)) = \begin{cases} 1, Y \neq f(x) \\ 0, Y = f(x) \end{cases}$
- 应用：用于二分类问题。

二元交叉熵损失函数

- 定义：对数损失函数，也称交叉熵损失，在二分类模型中，预测结果只有两种情况，对每个类别得到的概率分别p和1—p；

- 形式：$$L_i = L(p_i, y_i) = \begin{cases} -log(p_i), y_i = 1 \\ -log(1-p_i), y_i = 0 \end{cases} = -y_i log(p_i) - (1-y_i) log(1-p_i)$$

- 应用：用于二分类问题。

多元交叉熵损失函数

- **定义：是对二分类损失函数的扩展；**

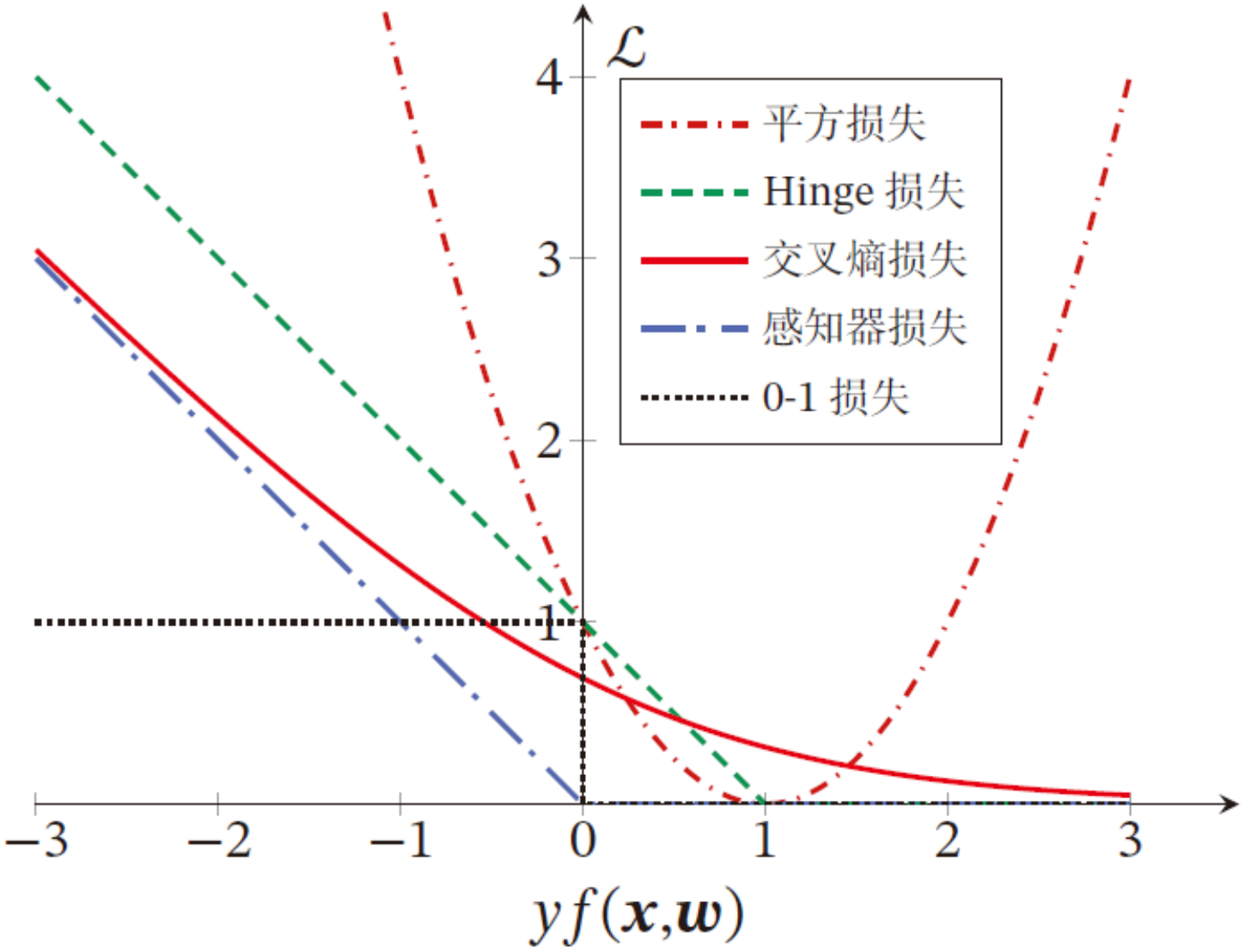
- **形式：** $L_i = -\sum_{c=1}^{K} y_{ic} log(p_{ic})$ 当属于某一类时，该类的y为1

- **应用：用于多分类问题。**

合页损失函数（Hinge Loss）

- 定义：样本被正确分类，则损失为0，否则为1-f•y(x)；
- 形式：$L(y, f(x)) = max(0, 1 - f \cdot y(x))$
- 应用：用于二分类问题，例如，SVM采用的就是合页损失函数。

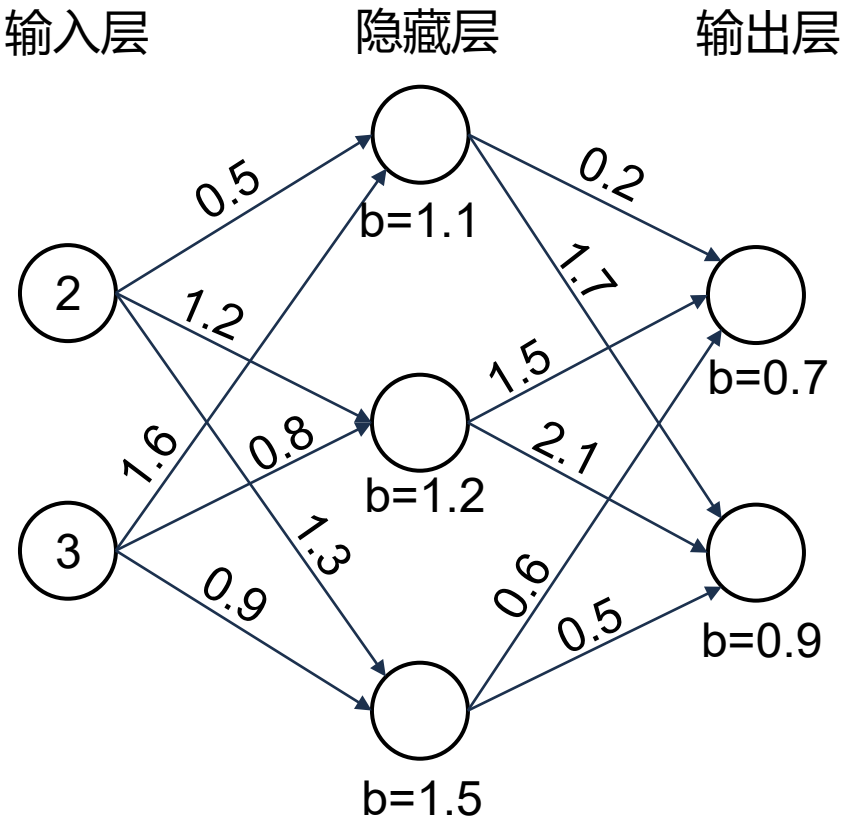
损失函数对比


$\mathcal{L}$
4
3
2
1
−3
−2
−1
0
1
2
3
平方损失
Hinge 损失
交叉熵损失
感知器损失
0-1 损失
$yf(\boldsymbol{x},\boldsymbol{w})$

复习

- **神经网络的设计需要考虑哪三方面?**
- **什么是损失函数? 以下哪个损失函数主要应用在回归问题?**

A. 0-1损失函数　B. 交叉熵损失函数　C. 平方损失函数　D. 合页损失函数

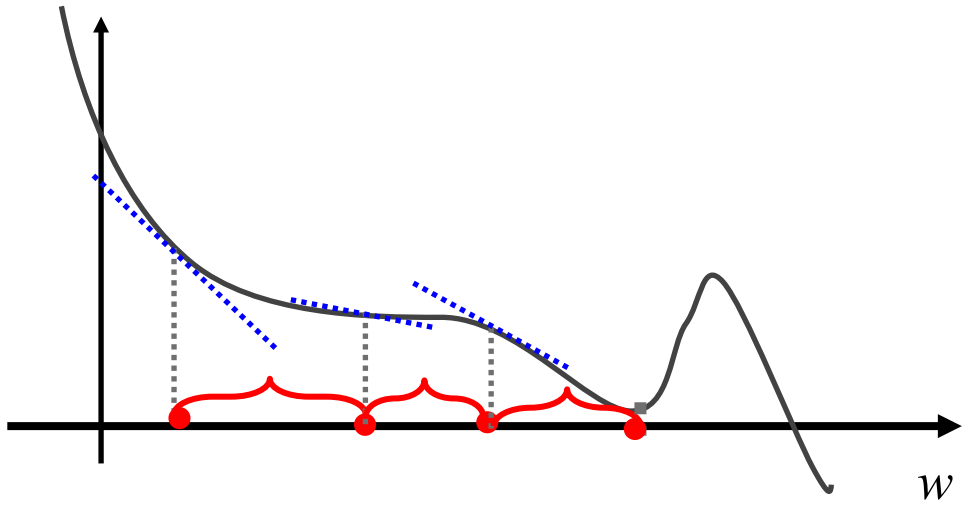


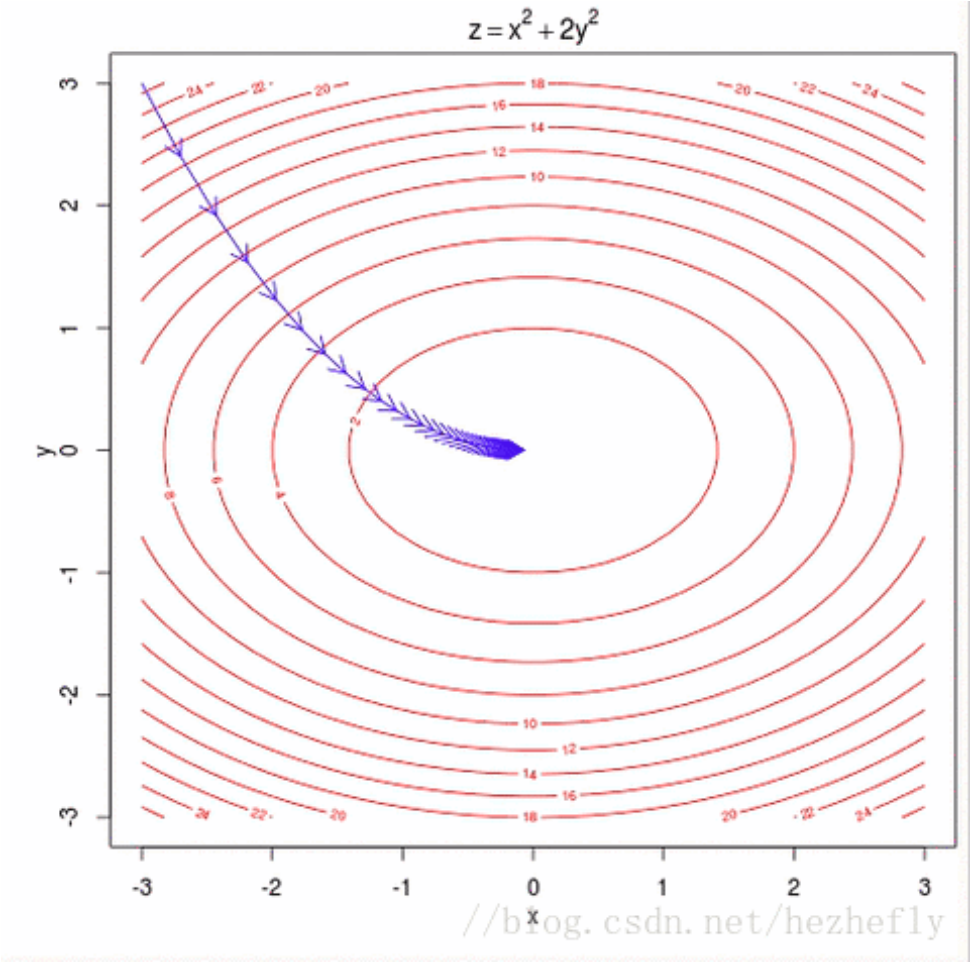
- **某前馈全连接神经网络的拓扑结构如图所示:**
  - 该网络包含1个输入层、1个隐藏层、1个输出层;
  - 某一次前向传播的权重和偏置已标注在图中适当位置;
  - 隐藏层使用ReLU激活函数，输出层不使用激活函数。
- **请将隐藏层、输出层的净活性值标注在图中的相应位置。**

优化算法

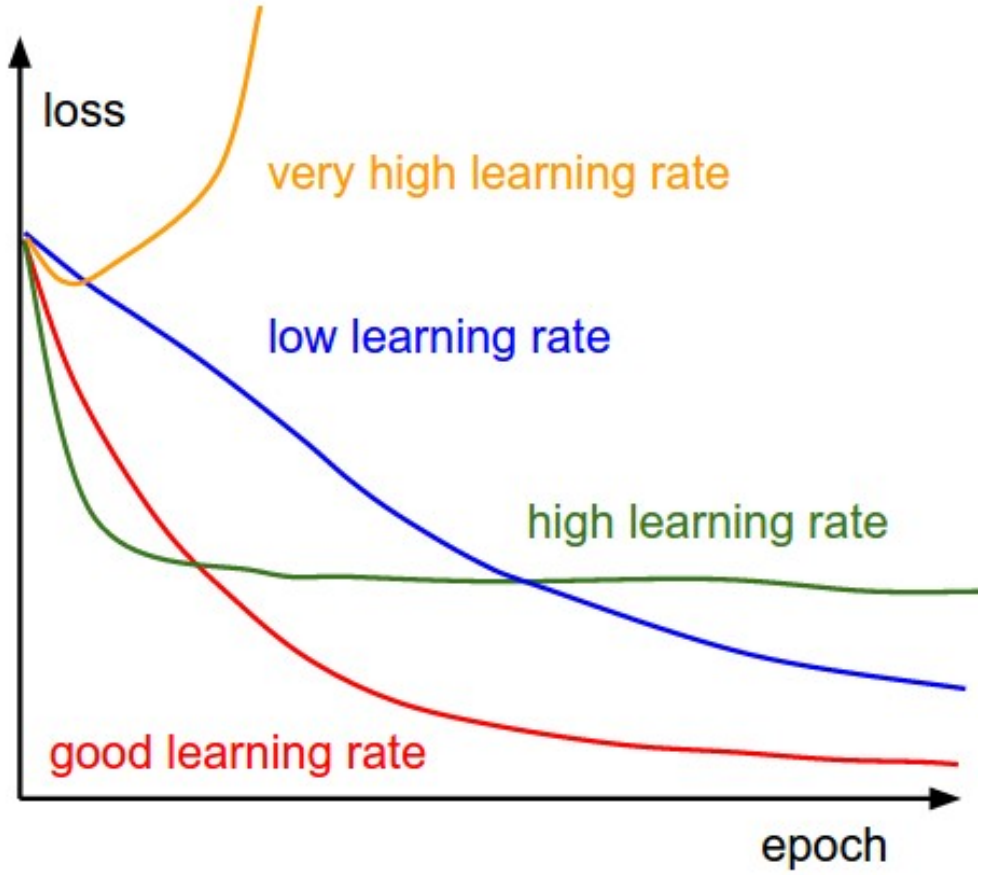
梯度下降

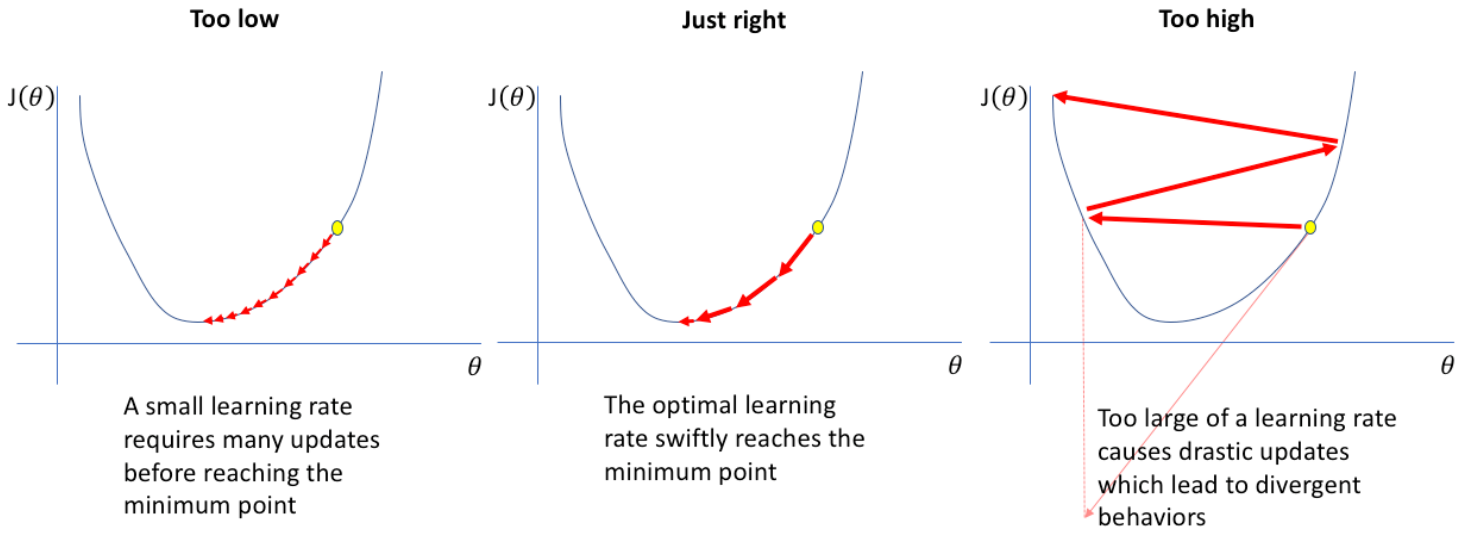
- **初始化网络参数w**
- **重复**
  - 计算梯度：$\partial L / \partial w$
  - 更新参数：$w \leftarrow w - \lambda \cdot \partial L / \partial w$





学习率的选择


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch


Too low
$J(\theta)$
θ
A small learning rate requires many updates before reaching the minimum point
Just right
$J(\theta)$
θ
The optimal learning rate swiftly reaches the minimum point
Too high
$J(\theta)$
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

随机梯度下降法

- 随机梯度下降法（Stochastic Gradient Descent，SGD）也叫增量梯度下降，每个样本都进行更新。

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{L}\left(\theta_t; x^{(t)}, y^{(t)}\right)}{\partial \theta},$$

算法 2.1: 随机梯度下降法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α

随机初始化 θ;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n = 1 \cdots N$ **do**

    从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$;

    // 更新参数

    $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}\left(\theta; x^{(n)}, y^{(n)}\right)}{\partial \theta}$;

  end

until 模型 $f(\mathbf{x}; \theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: θ

小批量（Mini-Batch）随机梯度下降法

- 实际用于深度学习的梯度下降算法介于以上两者之间，使用一个以上而又不是全部的训练样本。
- 在一次训练中，小批量（Mini-batch）的数量通常是固定的，也被称为Batch Size（通常在1到几百之间，且通常为2的n次方）。

反向传播算法

如何计算梯度?

- **神经网络为一个复杂的复合函数**
  - 链式法则

$$y = f^5(f^4(f^3(f^2(f^1(x))))) \rightarrow \frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial \mathrm{x}} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$

- **反向传播算法**
  - 根据前馈网络的特点而设计的高效方法

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W_{ij}^{(l)}} = \left(\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} \right)^{\mathrm{T}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}},$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \left(\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \right)^{\mathrm{T}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}.$$

- **一个更加通用的计算方法**
  - 自动微分（Automatic Differentiation，AD）

矩阵微积分

- **矩阵微积分（Matrix Calculus）**
  - 多元微积分的一种表达方式，即使用矩阵和向量来表示因变量每个成分关于自变量每个成分的偏导数。

- **标量关于向量的偏导数**

$$\frac{\partial y}{\partial \boldsymbol{x}} = [\frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_p}]^{\mathrm{T}}$$

- **向量关于向量的偏导数**

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

反向传播算法的问题定义

已知：

- **训练集：** $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$
- **前馈神经网络：**

L ——神经网络总层数

M_l ——第l层的神经元个数

$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$ ——第l-1层到第l层的权重矩阵

$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$ ——第l-1层到第l层的偏置

$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$ ——第l层神经元的净活性值

$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$ ——第l层神经元的活性值

- **损失函数：** $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$

反向传播算法的问题定义

已知：

- 训练集
- 前馈神经网络
- 损失函数

问题：

- $\mathcal{L}(y, \hat{y})$ 是如何指导权重 w 、偏置 b 的更新的?
- 也就是说，计算出了 $\mathcal{L}(y, \hat{y})$ ， w 、b 应如何更新?

反向传播算法

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

（1）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}$

只有 z_i 是与 $w_{i:}$ 相关的

$$\begin{aligned}\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right] \\ &= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\ &= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right] \\ &\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},\end{aligned}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

（3）计算误差项

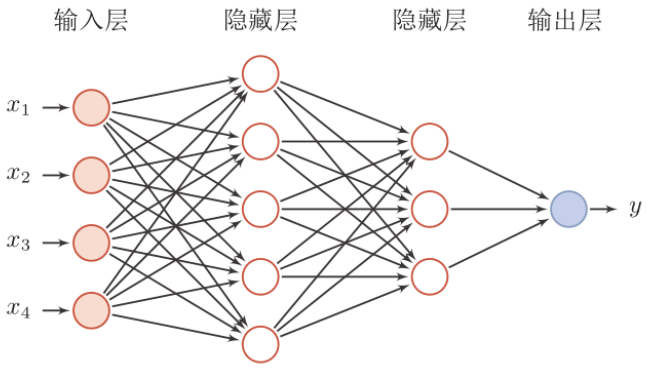
$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad \in \mathbb{R}^{m^{(l)}}$$

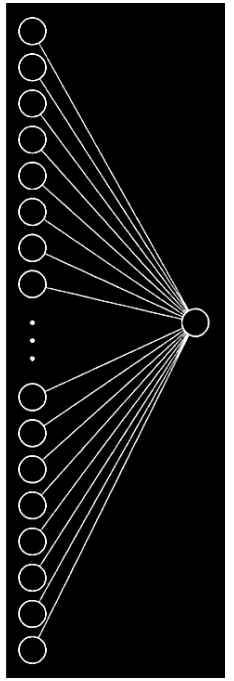
误差项

（2）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

$m^{(l)} \times m^{(l)}$ 的单位矩阵





反向传播算法

误差项 $\delta^{(l)}$ 来表示第 l 层神经元对最终损失的影响，也反映了最终损失对第 l 层神经元的敏感程度。误差项也间接反映了不同神经元对网络能力的贡献程度，从而比较好地解决了“贡献度分配问题”。

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$$

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

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^{\mathrm{T}}.$$

根据 $\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$，其中 $f_l(\cdot)$ 为按位计算的函数

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)})).$$

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

$$\begin{aligned} \delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\ &= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\ &= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right), \end{aligned}$$

反向传播：第 l 层的一个神经元的误差项（或敏感性）是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后，再乘上该神经元激活函数的梯度。

反向传播算法

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

（1）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}$

$$\begin{aligned}\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right] \\ &= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right] \\ &= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right] \\ &\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},\end{aligned}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

（3）计算误差项

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad \in \mathbb{R}^{m^{(l)}}$$

误差项

（2）计算偏导数 $\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}}$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

$m^{(l)} \times m^{(l)}$ 的单位矩阵

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

在计算出上面三个偏导数之后，

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}.$$

同理，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层偏置 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

反向传播算法

1. 前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$，直到最后一层；
2. 反向传播计算每一层的误差项 $\delta^{(l)}$；
3. 计算每一层参数的偏导数，并更新参数。

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\mathrm{T}}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$$

$$\begin{aligned}
\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\
&= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)} \right),
\end{aligned}$$

算法 4.1: 基于随机梯度下降的反向传播算法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α，正则化系数 λ，网络层数 L，神经元数量 $m^{(l)}$，$1 \leq l \leq L$.

随机初始化 $W, \mathbf{b}$；
repeat
  对训练集 $\mathcal{D}$ 中的样本随机重排序；
  for $n = 1 \cdots N$ **do**
    从训练集 $\mathcal{D}$ 中选取样本 $(\mathbf{x}^{(n)}, y^{(n)})$；
    前馈计算每一层的净输入 $\mathbf{z}^{(l)}$ 和激活值 $\mathbf{a}^{(l)}$，直到最后一层；
    反向传播计算每一层的误差 $\delta^{(l)}$； // 公式 (4.60)
    // 计算每一层参数的导数
    $\forall l, \quad \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\mathrm{T}}$； // 公式 (4.62)
    $\forall l, \quad \frac{\partial \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$； // 公式 (4.63)
    // 更新参数
    $W^{(l)} \leftarrow W^{(l)} - \alpha(\delta^{(l)} (\mathbf{a}^{(l-1)})^{\mathrm{T}} + \lambda W^{(l)})$；
    $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \delta^{(l)}$；
  end
until 神经网络模型在验证集 $\mathcal{V}$ 上的错误率不再下降；
输出: $W, \mathbf{b}$

计算图与自动微分

自动微分

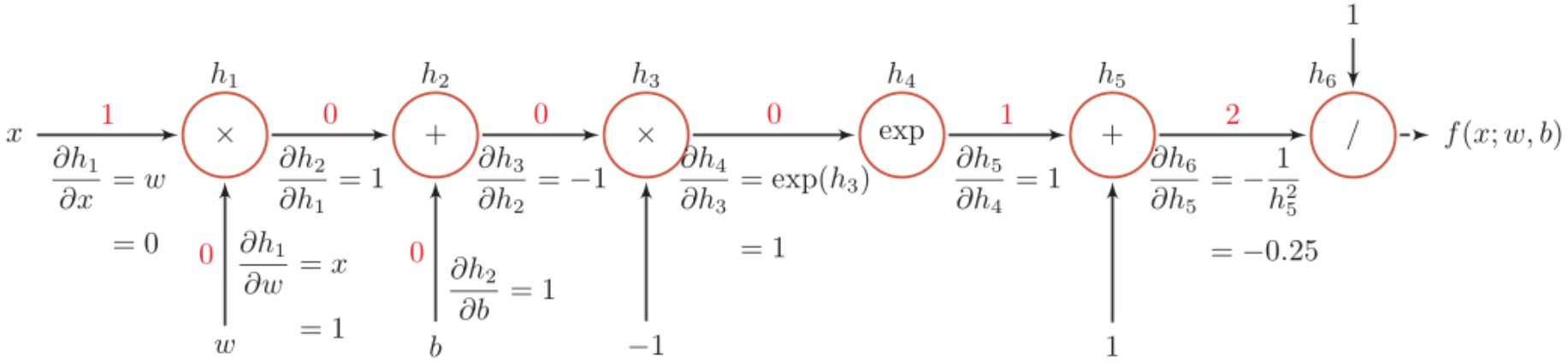
- **自动微分是利用链式法则来自动计算一个复合函数的梯度。**

$$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}.$$

函数	导数	
$h_1=x\times w$	$\frac{\partial h_1}{\partial w}=x$	$\frac{\partial h_1}{\partial x}=w$
$h_2=h_1+b$	$\frac{\partial h_2}{\partial h_1}=1$	$\frac{\partial h_2}{\partial b}=1$
$h_3=h_2\times -1$	$\frac{\partial h_3}{\partial h_2}=-1$	
$h_4=\exp(h_3)$	$\frac{\partial h_4}{\partial h_3}=\exp(h_3)$	
$h_5=h_4+1$	$\frac{\partial h_5}{\partial h_4}=1$	
$h_6=1/h_5$	$\frac{\partial h_6}{\partial h_5}=-\frac{1}{h_5^2}$	

计算图

- 计算图（Computational Graph）



$$\frac{\partial f(x;w,b)}{\partial w}\Big|_{x=1,w=0,b=0} = \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}$$

$$= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1$$

$$= 0.25.$$

自动微分

- **前向模式和反向模式**
  - 反向模式和反向传播的计算梯度的方式相同

- **如果函数和参数之间有多条路径，可以将这多条路径上的导数再进行相加，得到最终的梯度。**

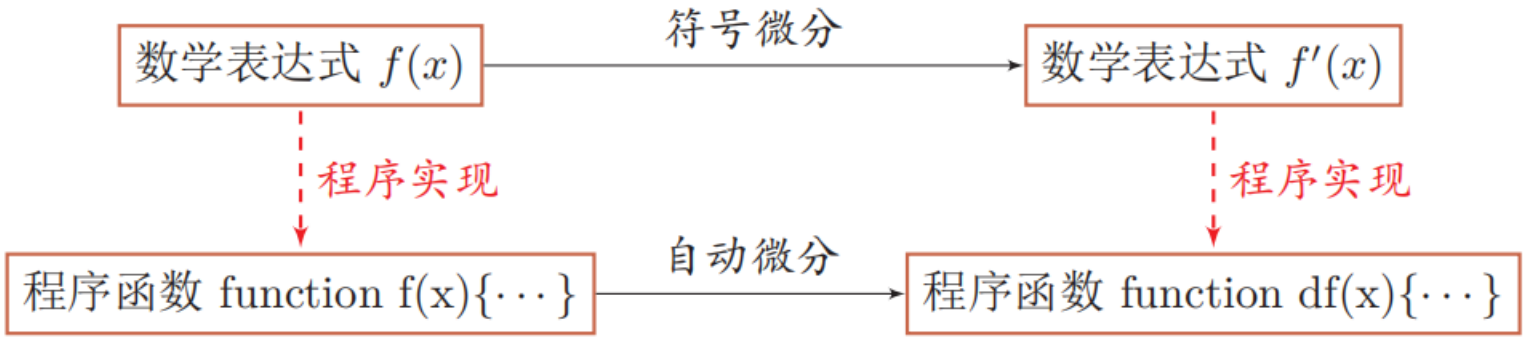


图 4.9 符号微分与自动微分对比

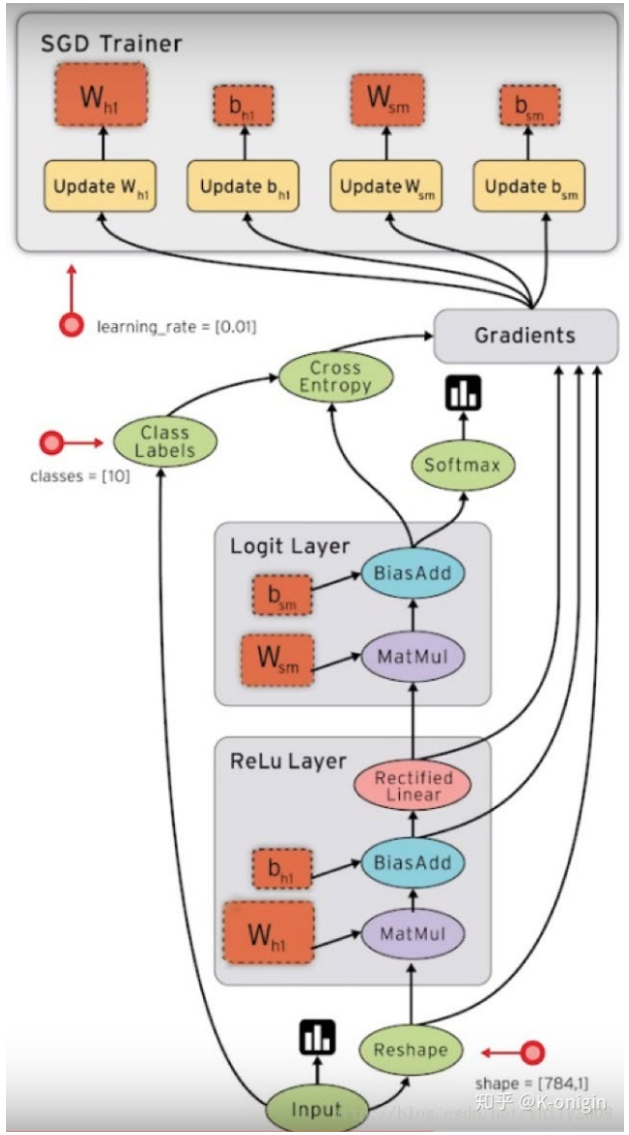
静态计算图和动态计算图

- **静态计算图**
  - 是在编译时构建计算图，计算图构建好之后在程序运行时不能改变。
  - Theano和Tensorflow1.0

```
import tensorflow as tf

#定义计算图
g = tf.Graph()
with g.as_default():
    #placeholder为占位符，执行会话时候指定填充对象
    x = tf.placeholder(name='x', shape=[], dtype=tf.string)
    y = tf.placeholder(name='y', shape=[], dtype=tf.string)
    z = tf.string_join([x,y],name = 'join',separator=' ')

#执行计算图
with tf.Session(graph = g) as sess:
    print(sess.run(fetches = z,feed_dict={}))
```

静态计算图和动态计算图

- **动态计算图**
  - 是在程序运行时动态构建。两种构建方式各有优缺点。
  - DyNet，Chainer，PyTorch，TensorFlow2.0

A graph is created on the fly


```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

Q&A