《神经网络与深度学习》

前馈神经网络

内容

- 神经网络
  - 神经元
  - 网络结构
- 前馈神经网络
  - 参数学习
  - 计算图与自动微分
  - 优化问题

神经网络

神经网络

- 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是分布式并行处理（Parallel Distributed Processing，PDP）网络，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**误差反向传播算法**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。

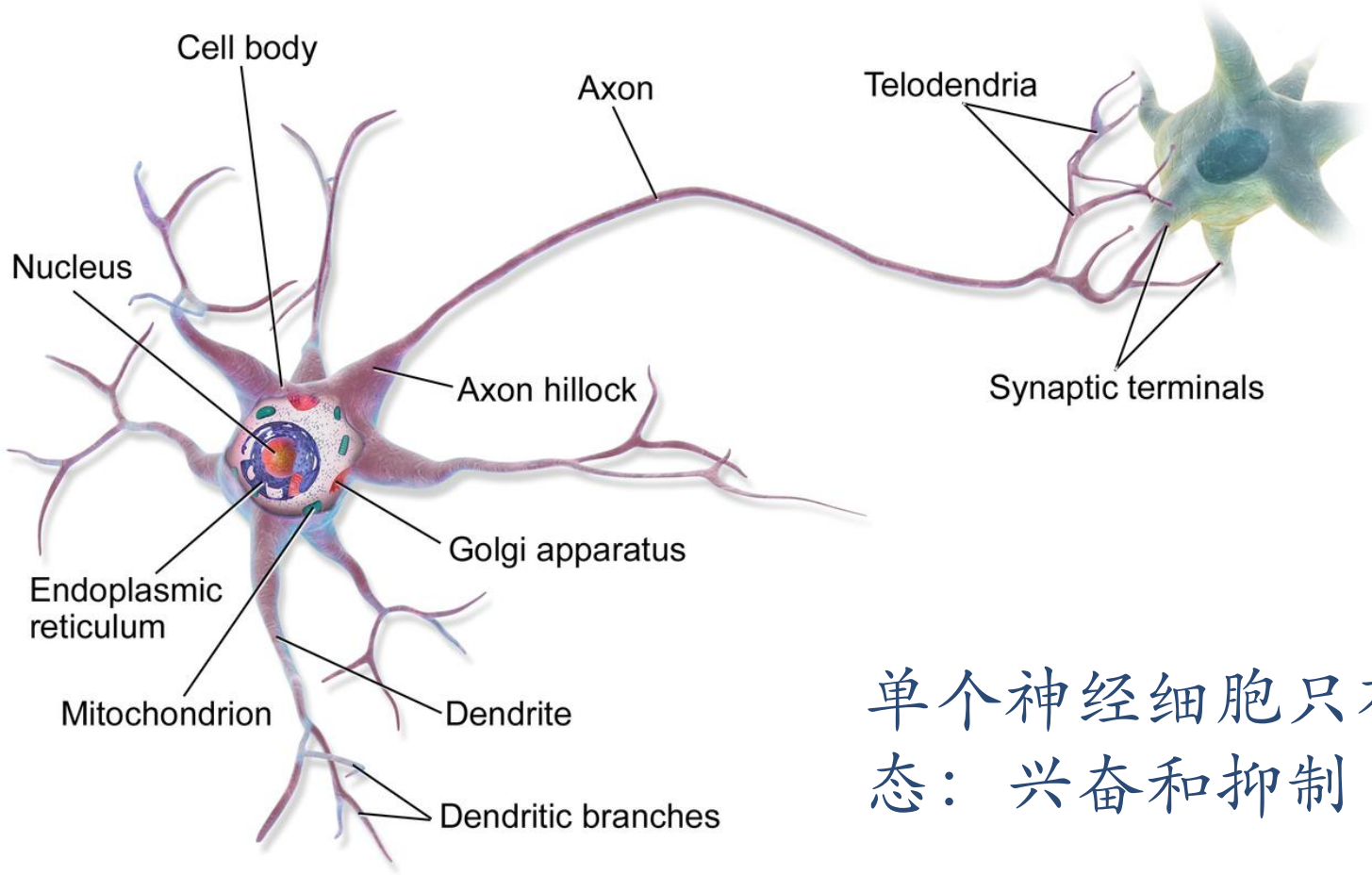
神经网络

- 分布式设计的好处
  - 编码冗余
  - 组合能力
  - 抽象和泛化
  - 学习和适应
  - 容错性
  - 计算效率
  - 灵活性和扩展性
  - 生物学合理性

神经元

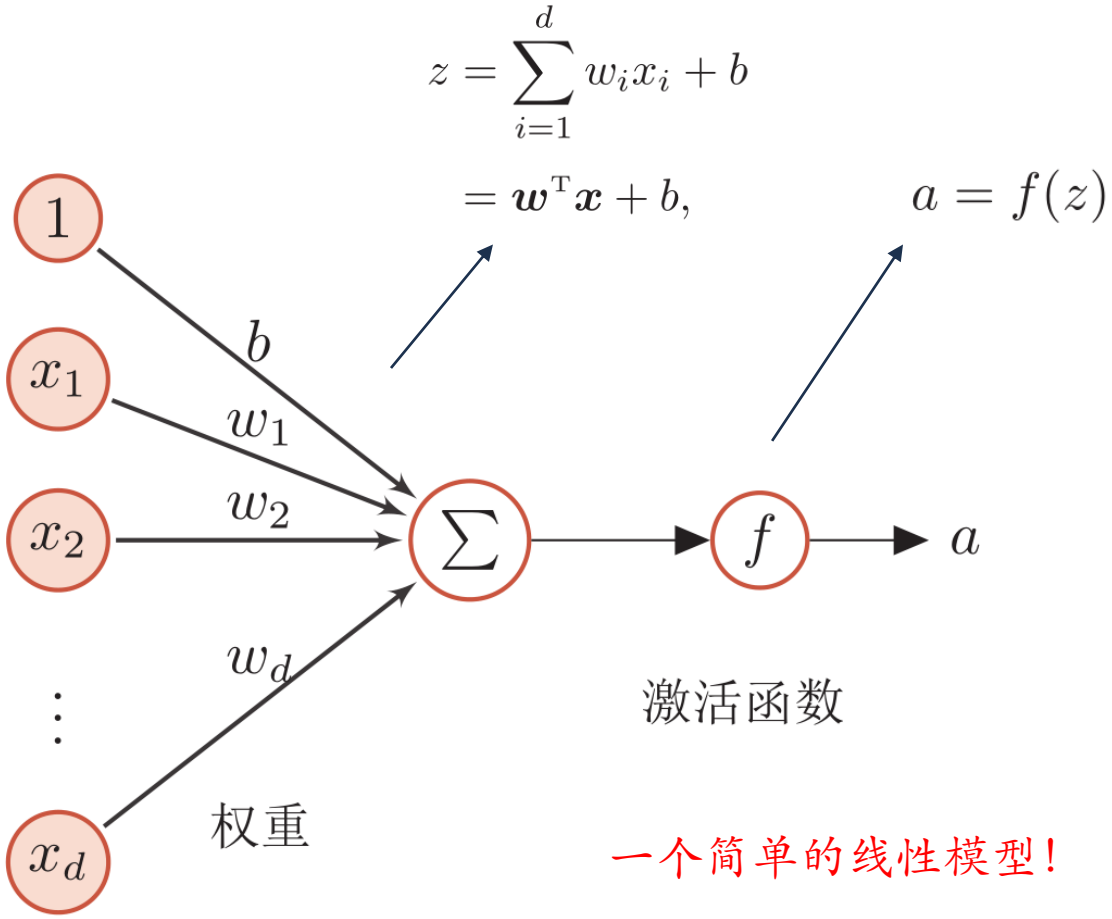
生物神经元

video: structure of brain



单个神经细胞只有两种状态：兴奋和抑制

人工神经元


$z = \sum_{i=1}^{d} w_i x_i + b$
$= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b,$
$a = f(z)$
1
x_1
x_2
x_d
b
w_1
w_2
w_d
$\sum$
f
a
激活函数
权重
一个简单的线性模型!

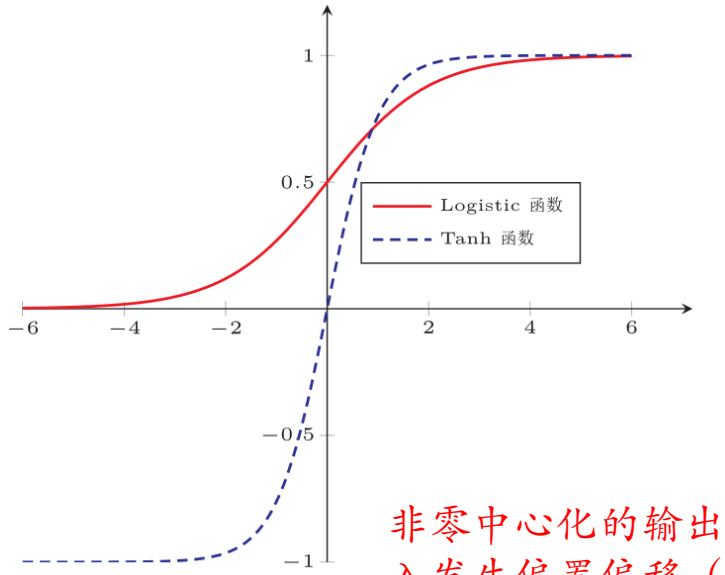
激活函数的性质

- 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。
- 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。
- 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。
- 单调性
  - 具有单调性的激活函数（如ReLU）可以确保网络学习到的权重在输入空间中保持一致的正向或负向影响，这有助于网络捕捉到更复杂的非线性关系。
  - 并不是所有的神经网络函数都需要单调性。

常见激活函数

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

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



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

- 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

常见激活函数

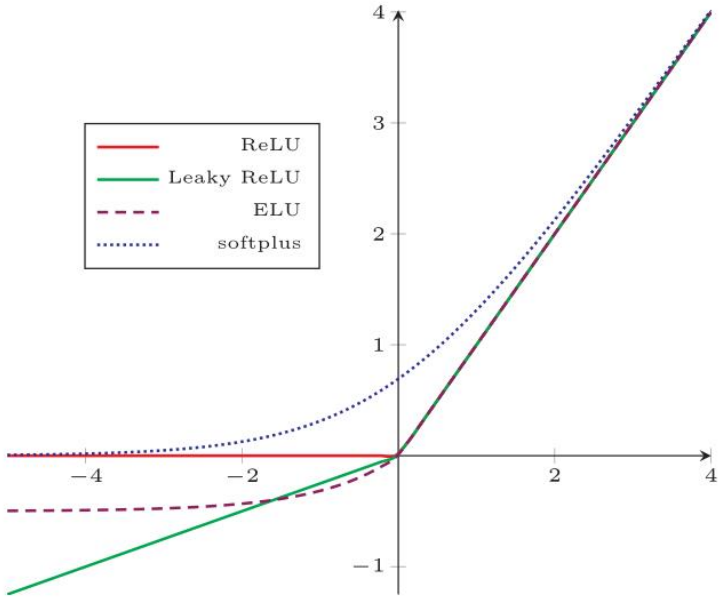
$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$= \max(0, x).$$

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$

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

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

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

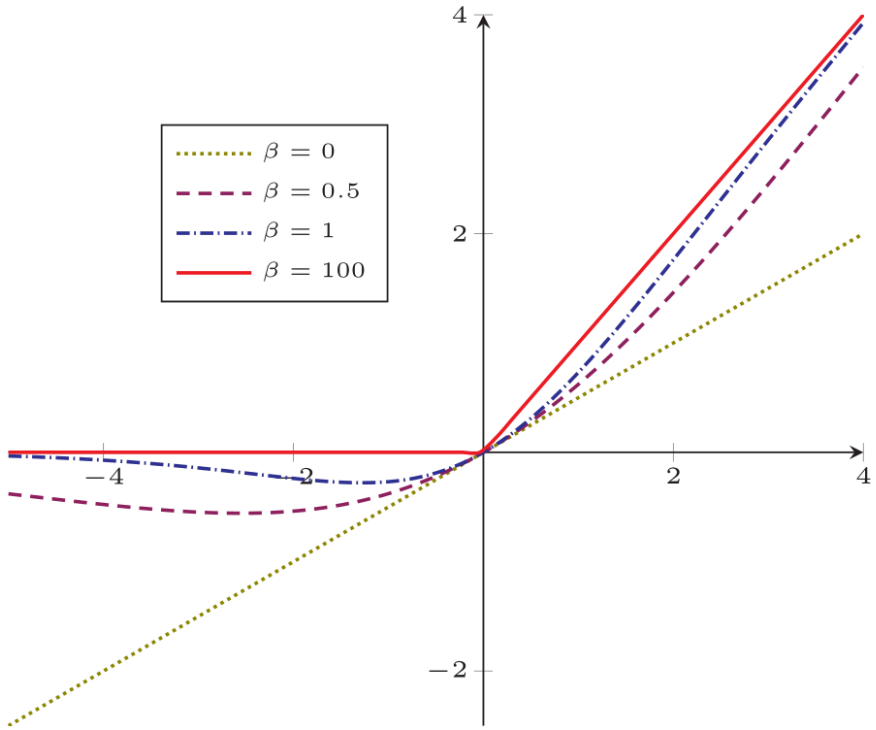


- 计算上更加高效
- 生物学合理性
  - 单侧抑制、宽兴奋边界
- 在一定程度上缓解梯度消失问题

死亡ReLU问题（Dying ReLU Problem）

常见激活函数

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



常见激活函数

- 高斯误差线性单元（Gaussian Error Linear Unit，GELU）

$$\mathrm{GELU}(x) = xP(X \leq x)$$

- 其中$P(X \leq x)$是高斯分布$N(\mu, \sigma^2)$的累积分布函数，其中μ, σ为超参数，一般设$\mu = 0, \sigma = 1$即可

- 由于高斯分布的累积分布函数为S型函数，因此GELU可以用Tanh函数或Logistic函数来近似

$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

或 $$\mathrm{GELU}(x) \approx x\sigma(1.702x).$$

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 函数	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU 函数	$f(x) = \max(0, x) + \min\Big(0, \gamma\big(\exp(x) - 1\big)\Big)$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log\big(1 + \exp(x)\big)$	$f'(x) = \frac{1}{1+\exp(-x)}$

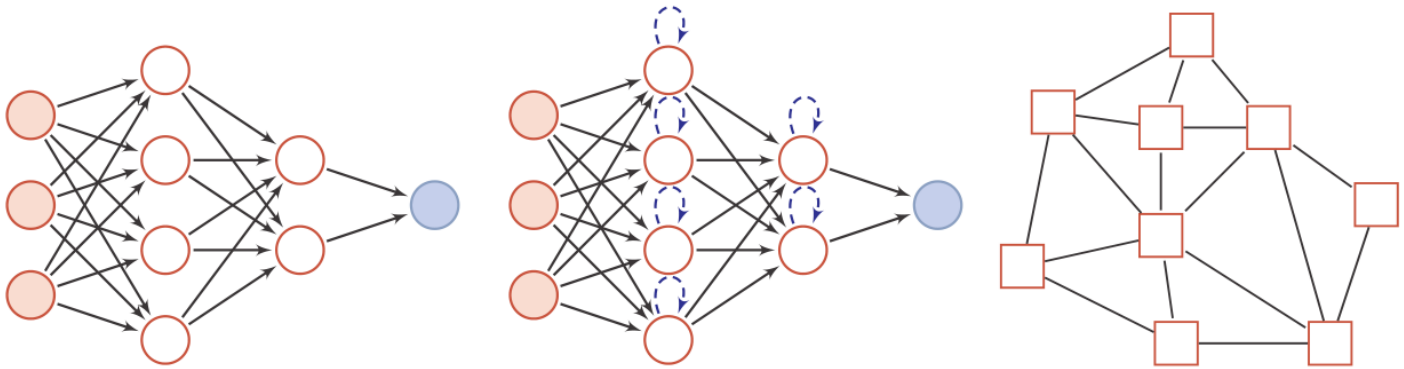
如何构建人工神经网络

- 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。



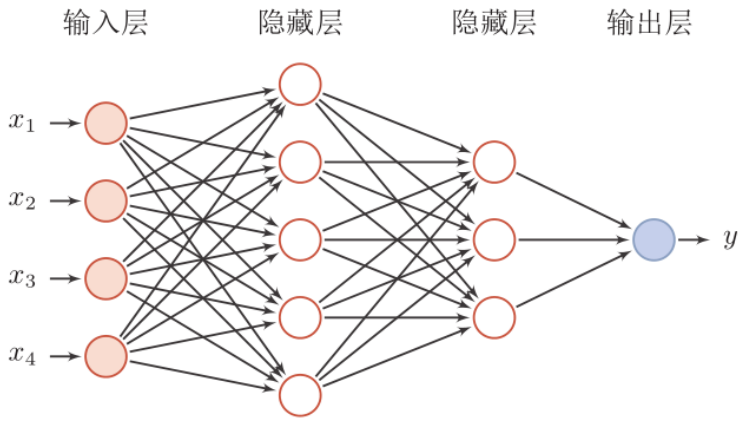
(a) 前馈网络　(b) 记忆网络　(c) 图网络

圆形节点表示一个神经元，方形节点表示一组神经元。

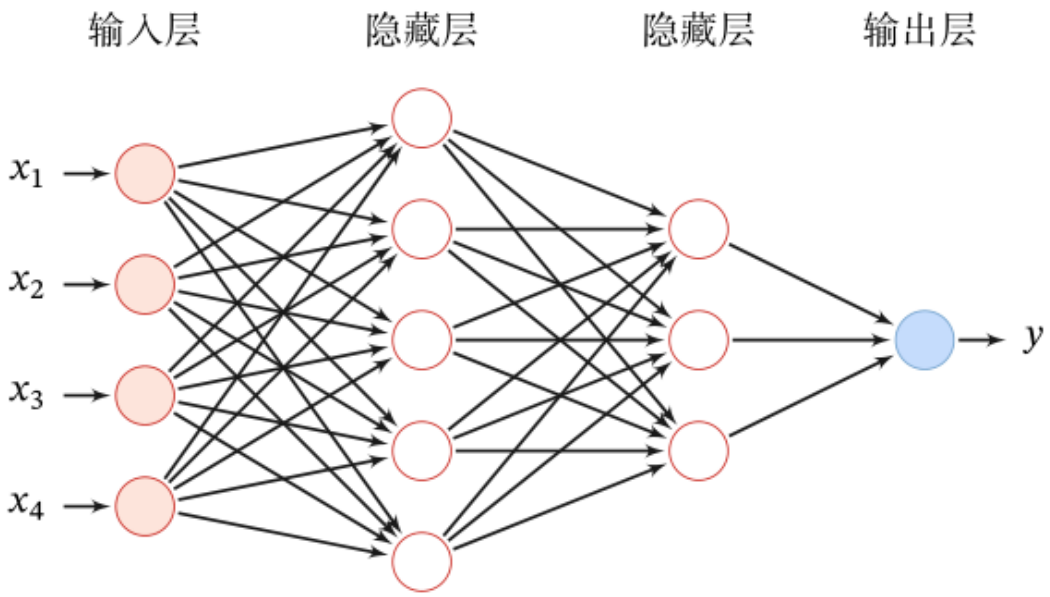
前馈神经网络

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。



前馈网络



给定一个前馈神经网络，用下面的记号来描述这样网络：

记号	含义
L	神经网络的层数
M_l	第 l 层神经元的个数
$f_l(\cdot)$	第 l 层神经元的激活函数
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	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 层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第 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：

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

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。

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

分类器　　神经网络

- 如果$g(\cdot)$为Logistic回归，那么Logistic回归分类器可以看成神经网络的最后一层。

参数学习

应用到机器学习

- 对于多分类问题
  - 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。

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

  - 采用交叉熵损失函数，对于样本(x,y)，其损失函数为

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

参数学习

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$输入给前馈神经网络，得到网络输出为$\hat{y}^{(n)}$，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

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

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

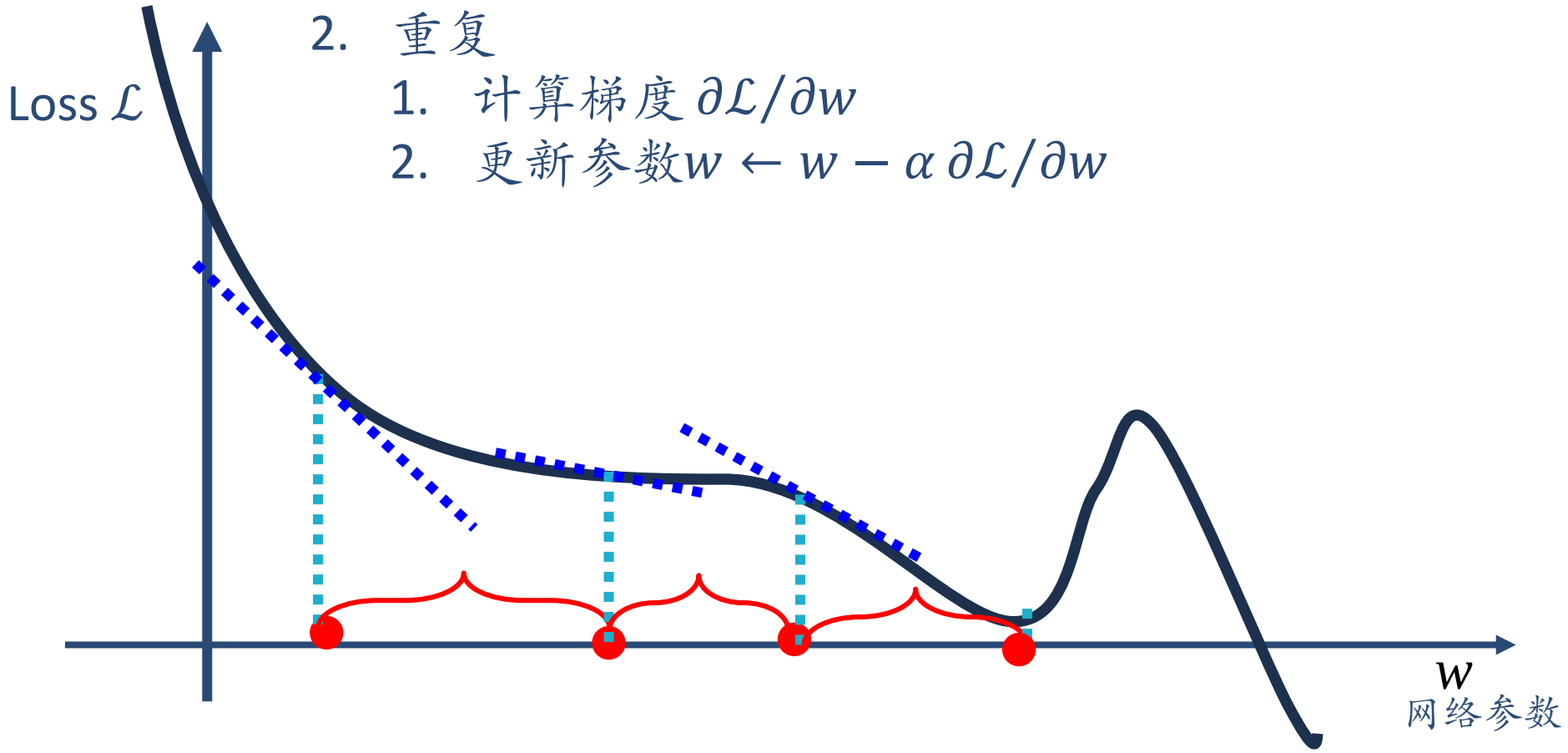
梯度下降

梯度下降

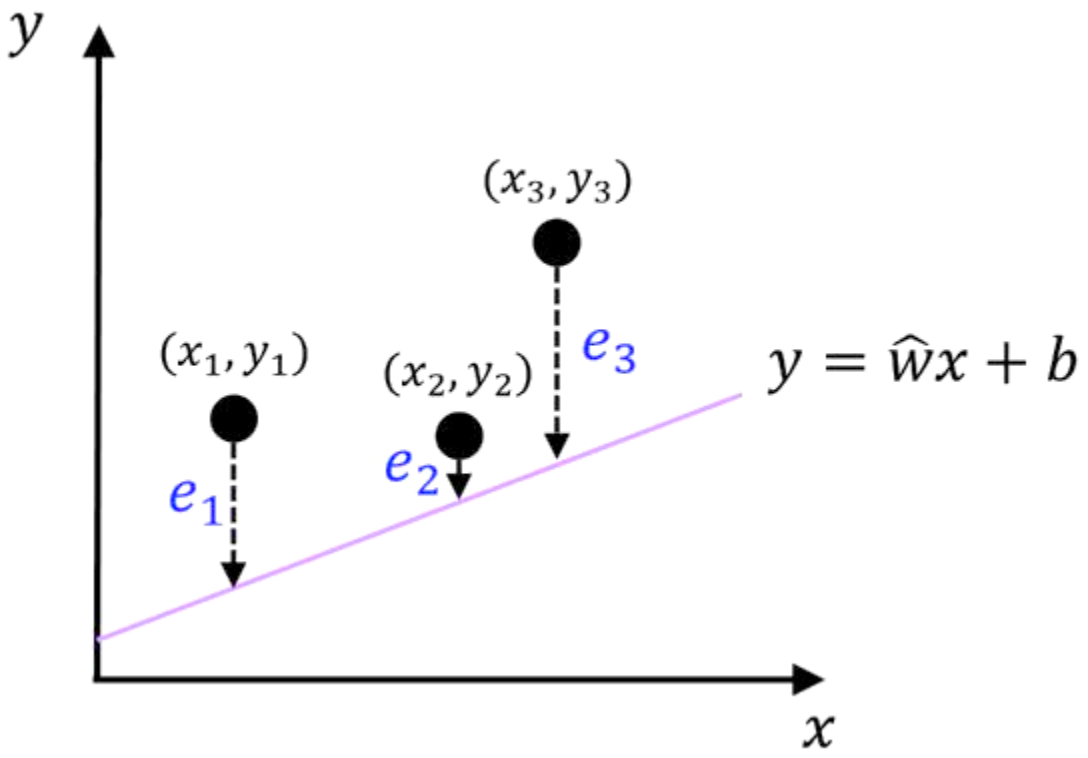
梯度下降

1. 初始化w
2. 重复
   1. 计算梯度 $\partial\mathcal{L}/\partial w$
   2. 更新参数 $w \leftarrow w - \alpha\,\partial\mathcal{L}/\partial w$

梯度：$\frac{\partial f(w)}{\partial w} = \lim_{\Delta w \to 0} \frac{f(w+\Delta w)}{\Delta w}$



梯度下降

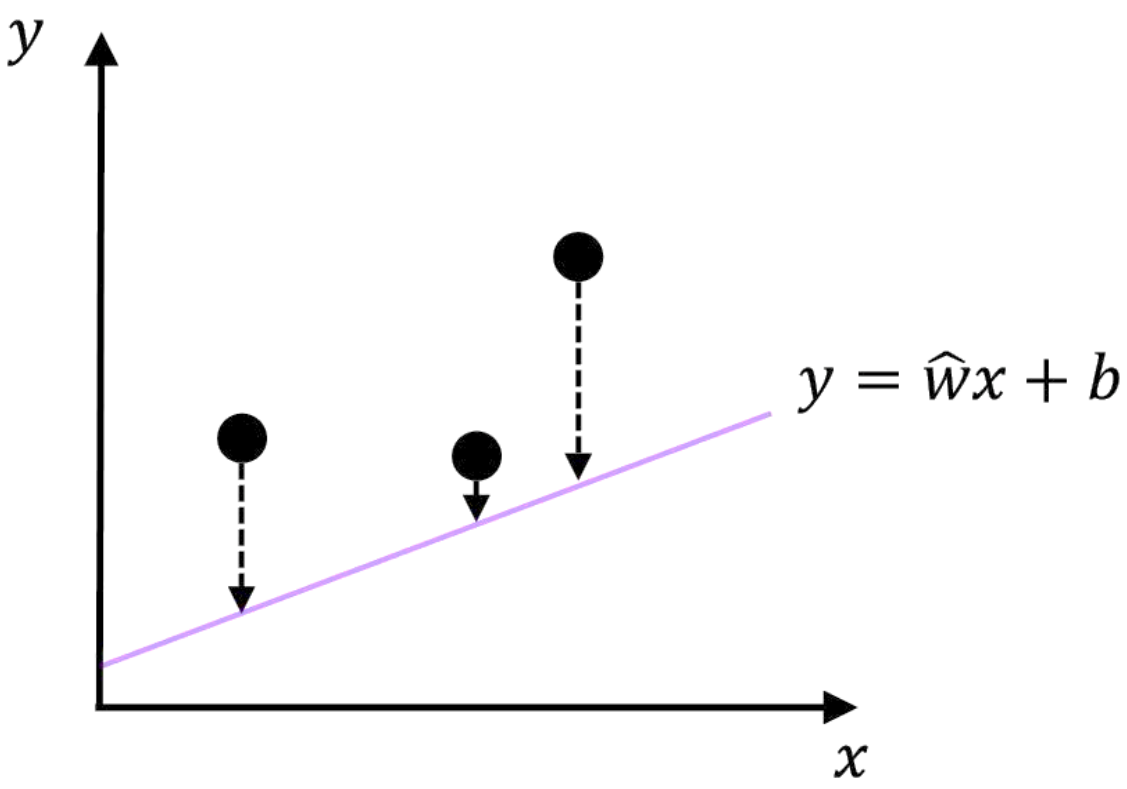


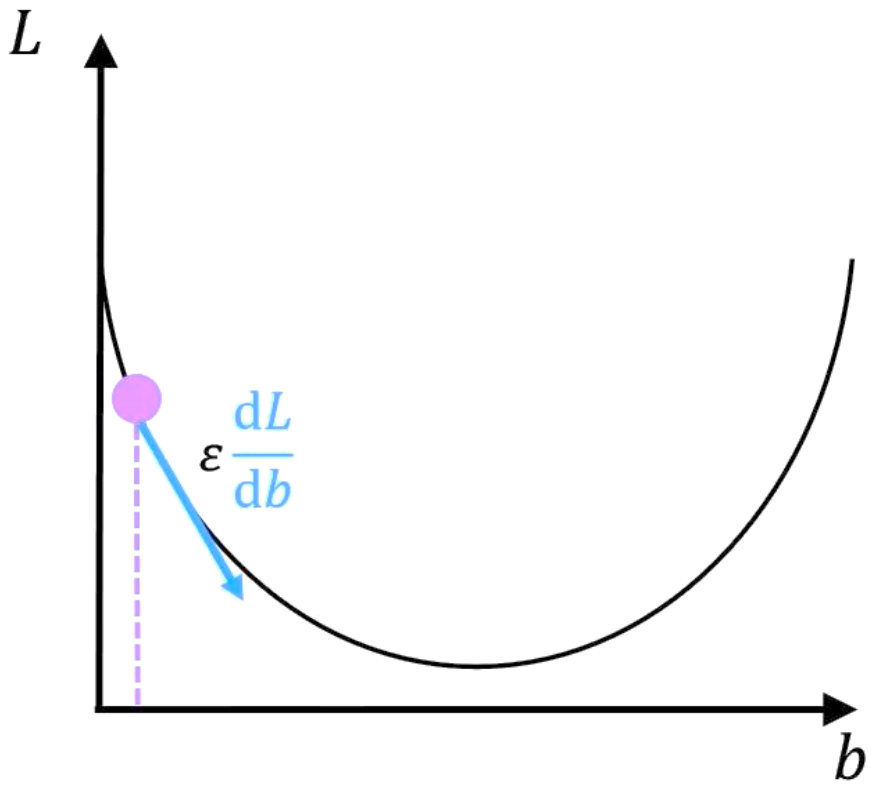
$$L = \frac{1}{2}\sum_{i=1}^{3} |e_i|^2 \quad \text{损失函数}$$

$$L = \frac{1}{2}\sum_{i=1}^{3} [y_i - (\hat{w}x_i + b)]^2$$

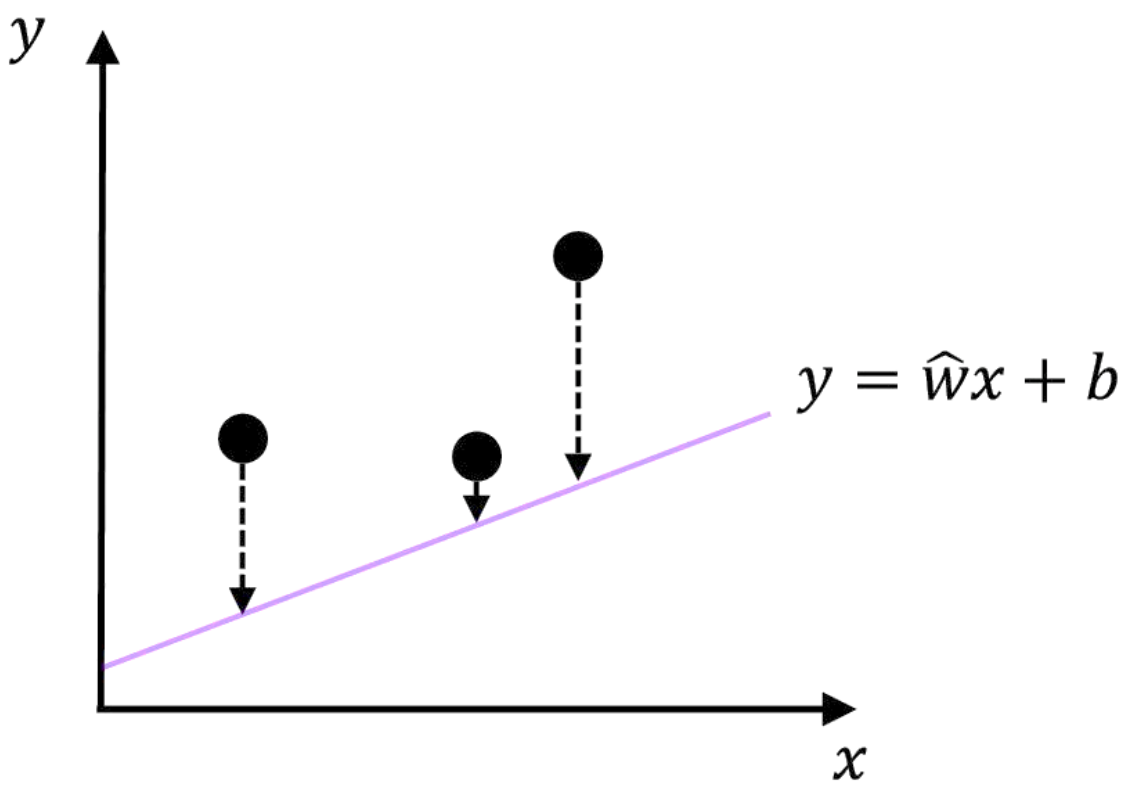
$$L = \frac{1}{2}\sum_{i=1}^{3} b^2 + 2(\hat{w}x_i - y_i)b + (y_i - \hat{w}x_i)^2$$

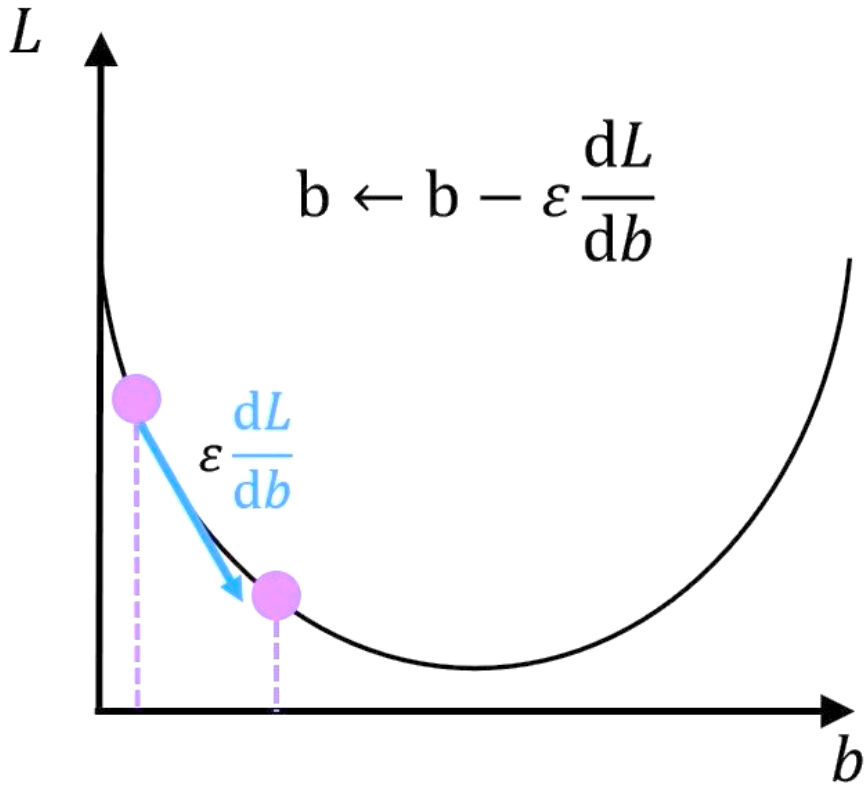
梯度下降



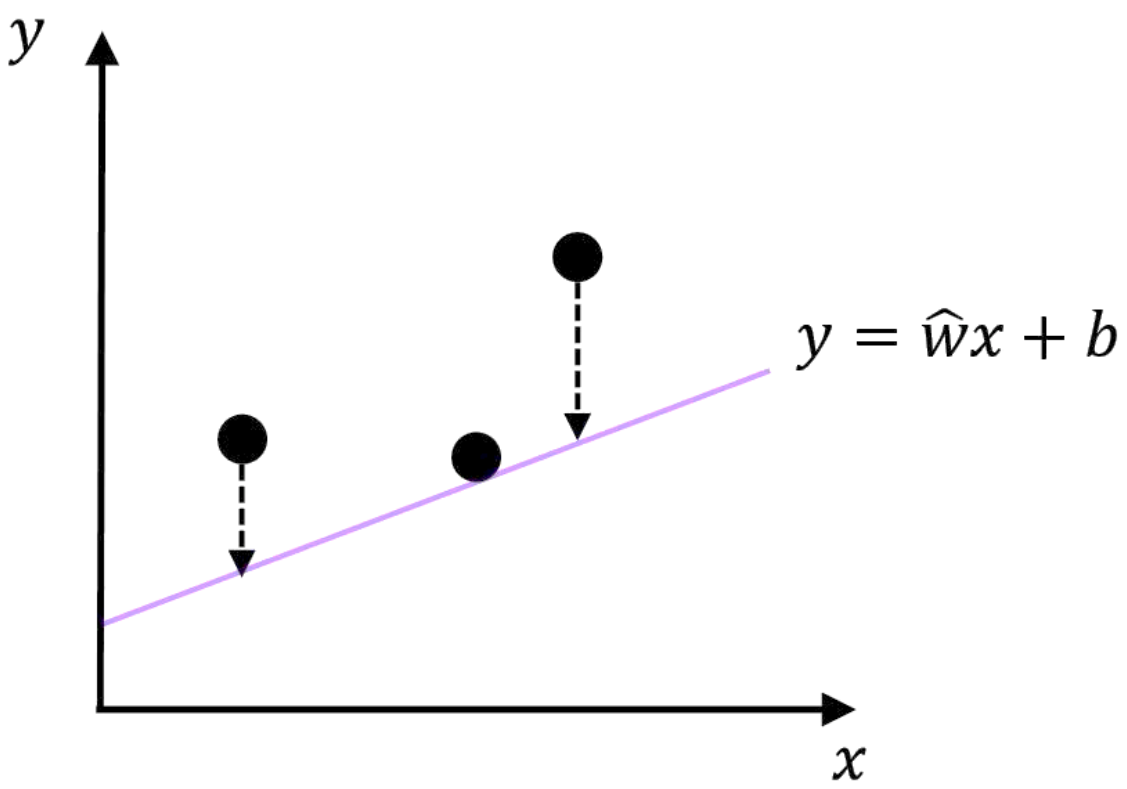


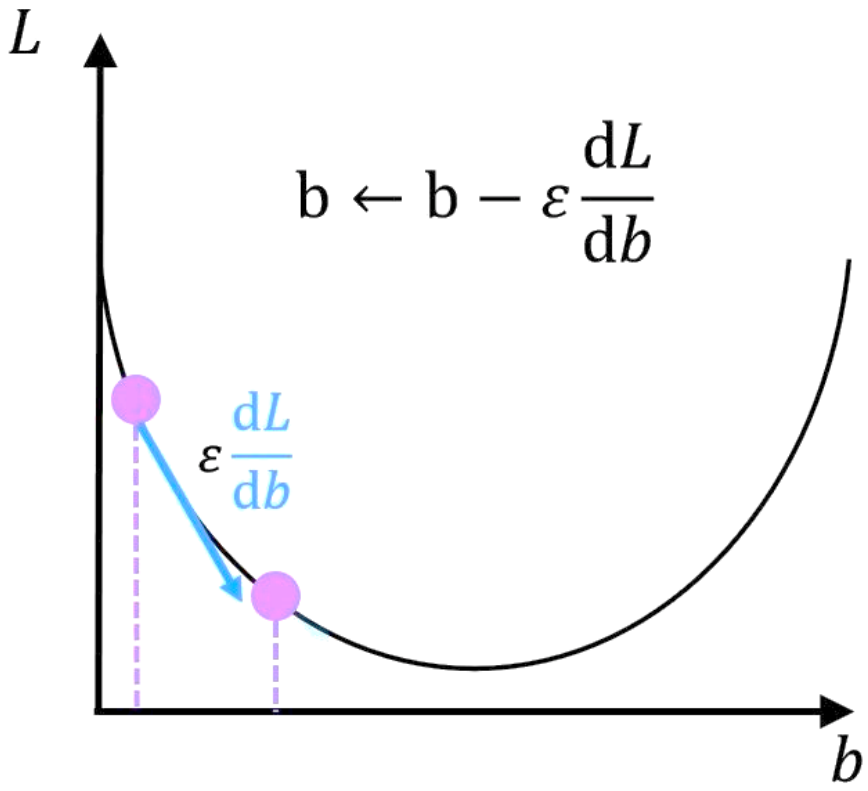
梯度下降


y
x
$y = \hat{w}x + b$

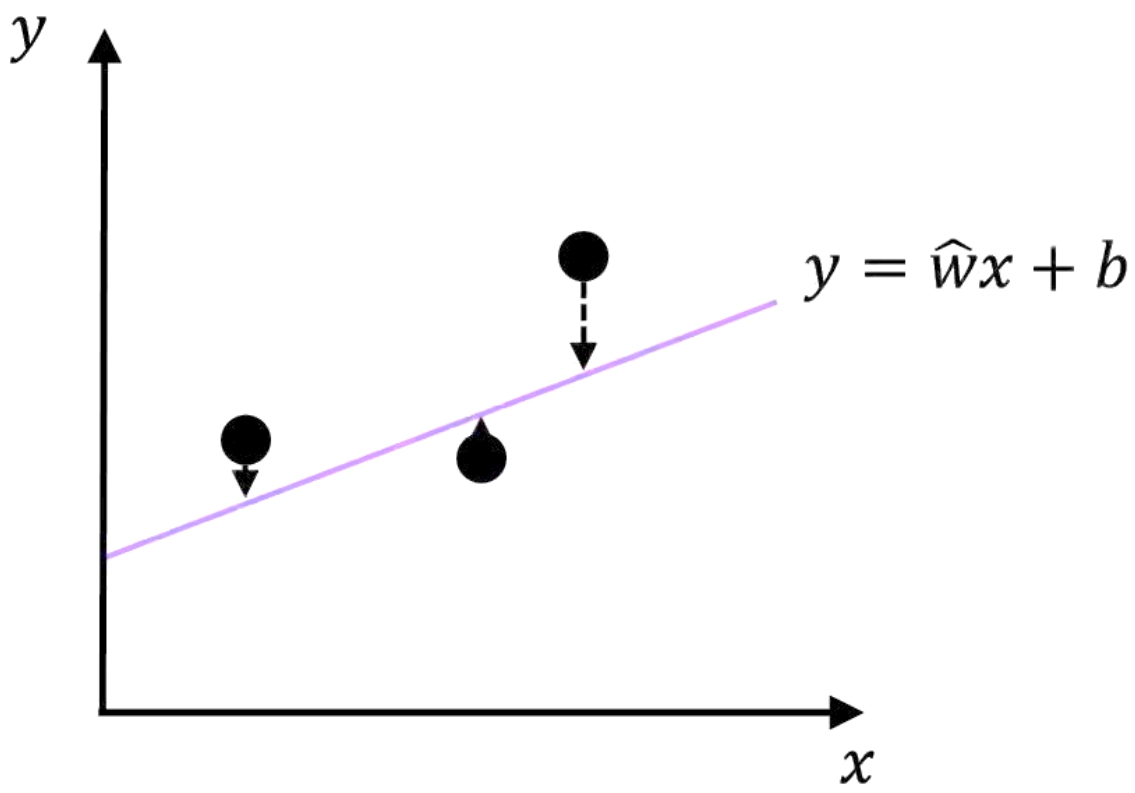

L
b
$\mathrm{b} \leftarrow \mathrm{b} - \varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$
$\varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$

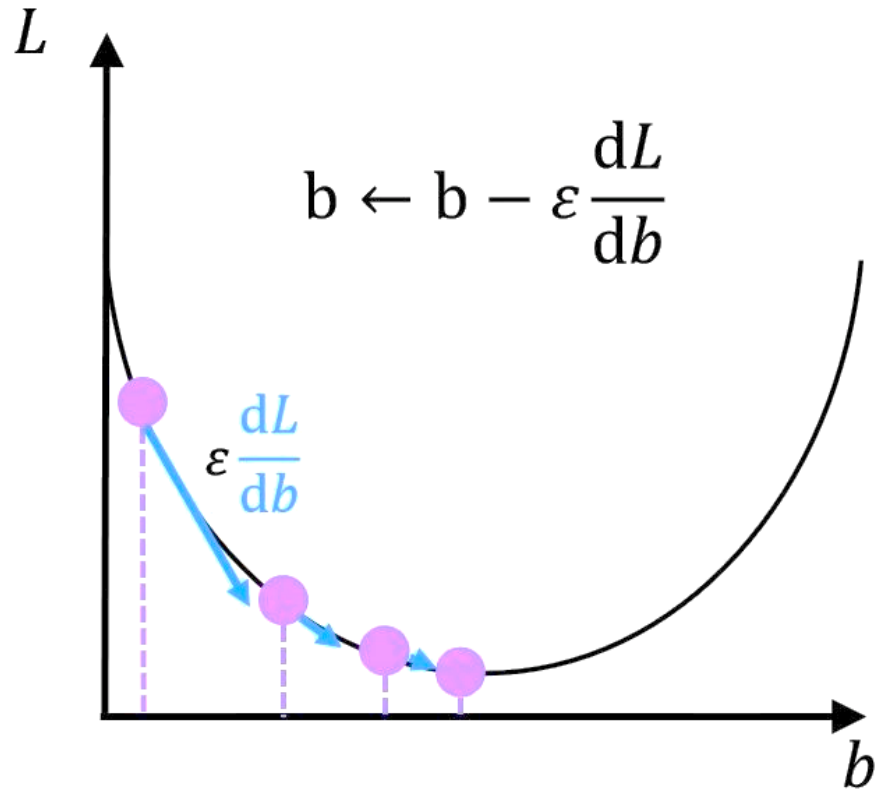
梯度下降


y
x
$y = \hat{w}x + b$

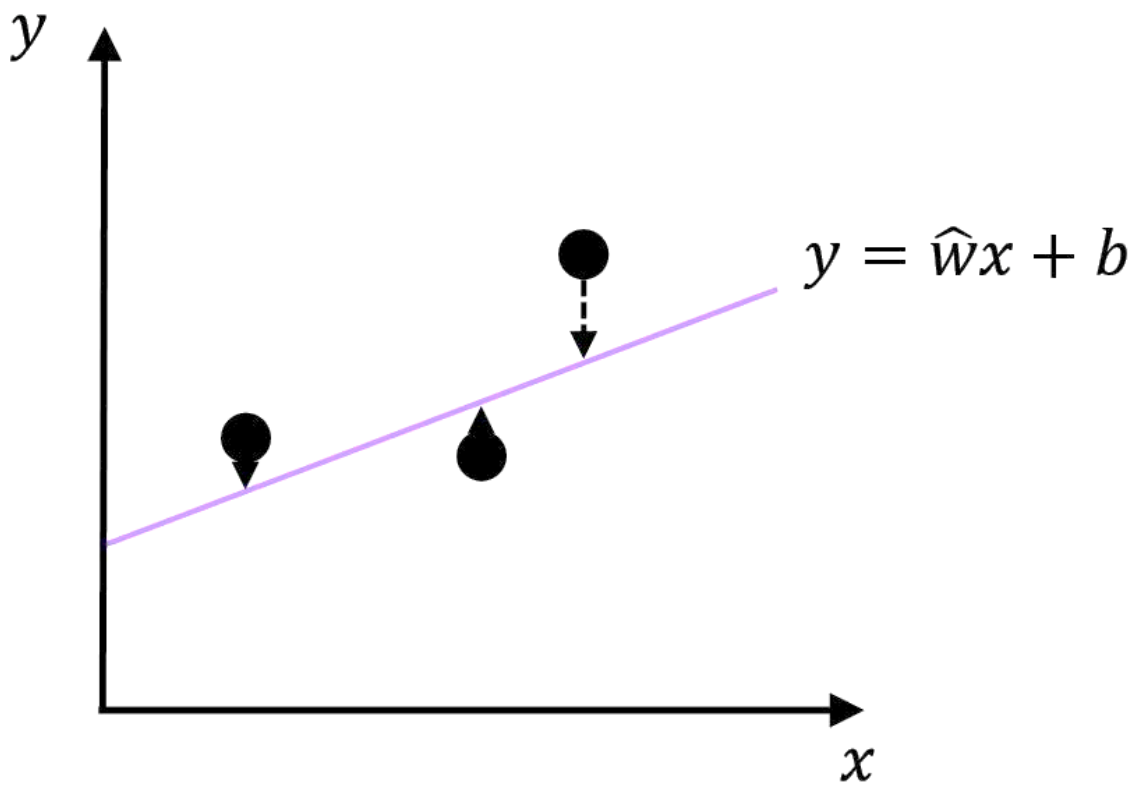

L
b
$b \leftarrow b - \varepsilon \frac{dL}{db}$
$\varepsilon \frac{dL}{db}$

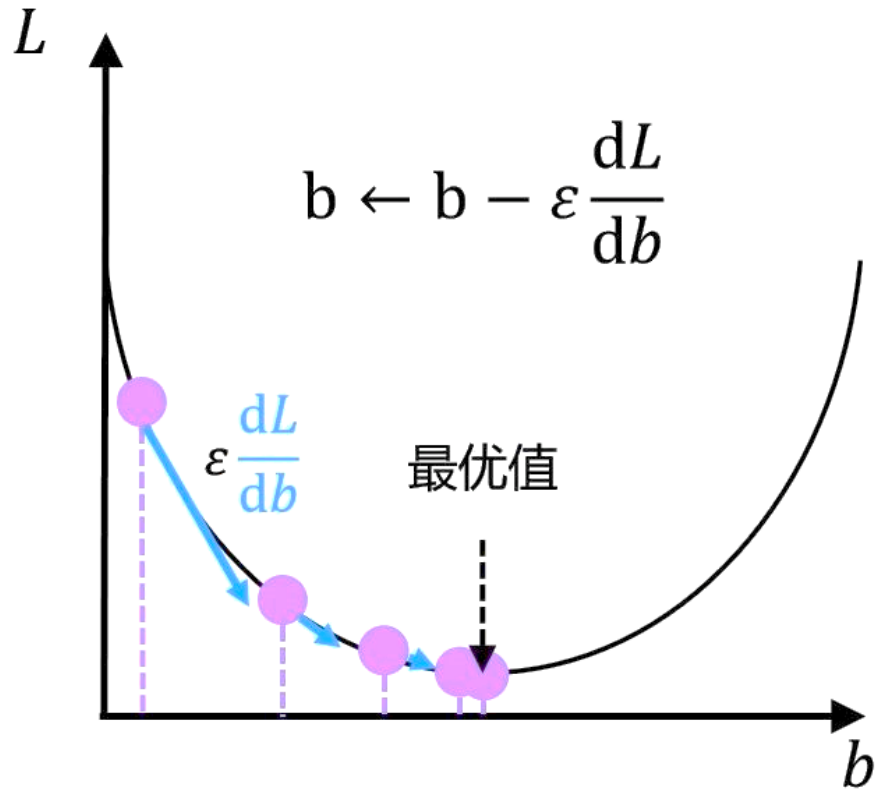
随机梯度下降


y
x
$y = \hat{w}x + b$

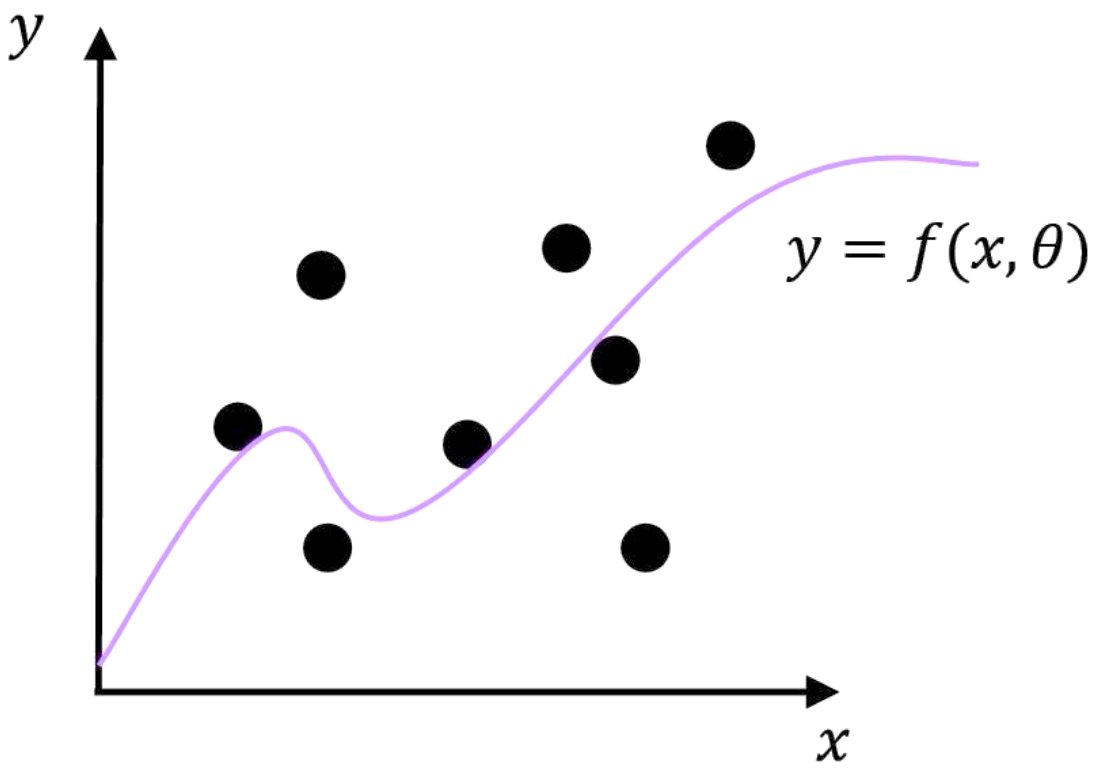

L
b
$\mathrm{b} \leftarrow \mathrm{b} - \varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$
$\varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$

梯度下降


y
x
$y = \hat{w}x + b$


L
b
$\mathrm{b} \leftarrow \mathrm{b} - \varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$
$\varepsilon \frac{\mathrm{d}L}{\mathrm{d}b}$
最优值

梯度下降



$L(f(x_i, \theta), y_i)$ 损失函数

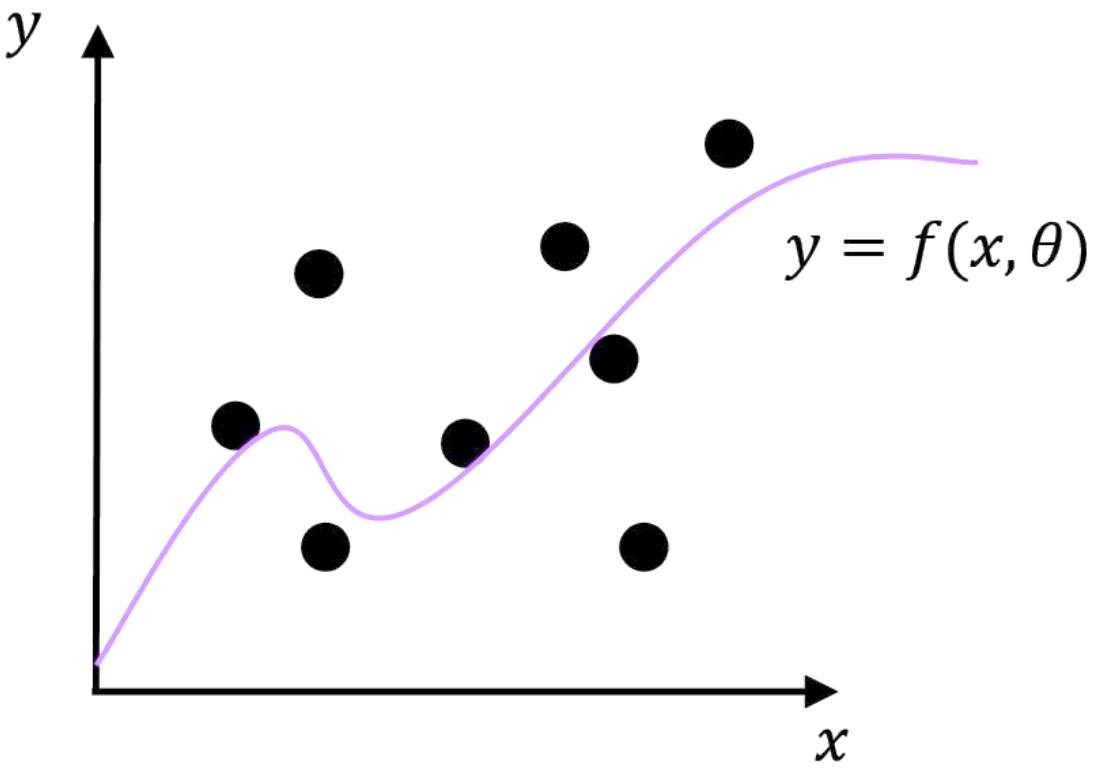
$\nabla_\theta L(f(x_i, \theta), y_i)$

$$g = \frac{1}{N} \nabla_\theta \sum_{i=1}^{N} L(f(x_i, \theta), y_i)$$

$$\theta \leftarrow \theta - \varepsilon g$$

学习率

随机梯度下降

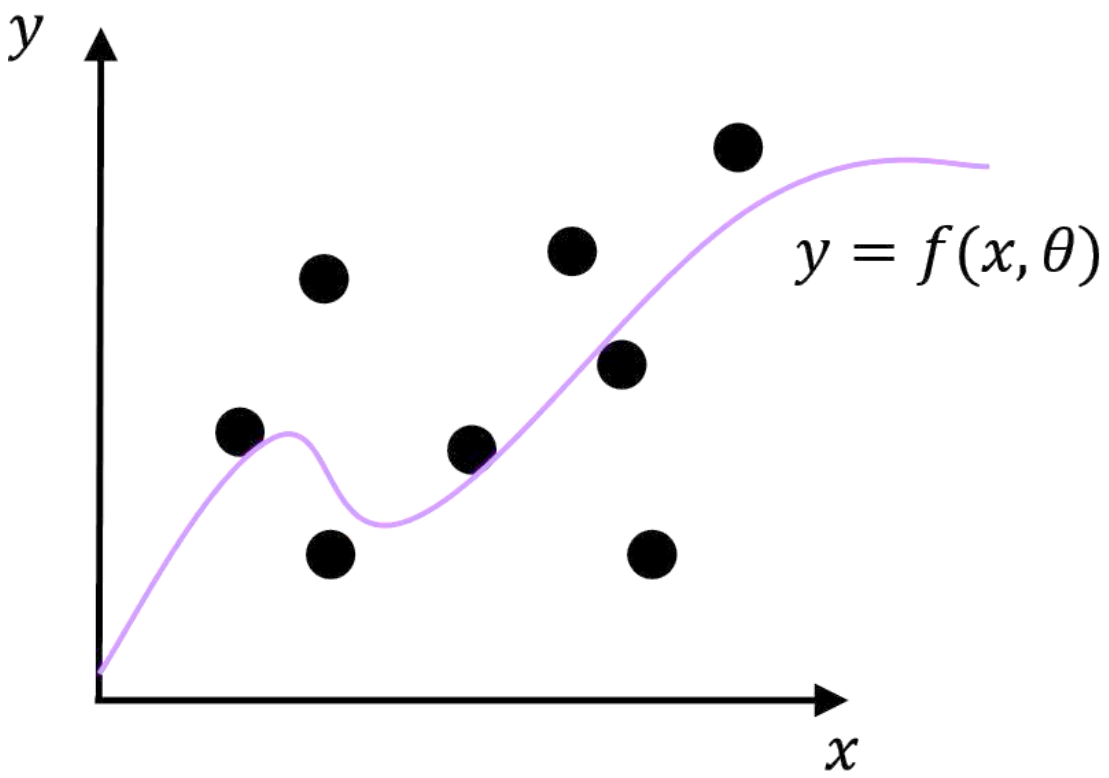


$$\nabla_\theta L(x_1)$$
$$\nabla_\theta L(x_2)$$
$$\nabla_\theta L(x_3)$$
$$\nabla_\theta L(x_4)$$
$$\nabla_\theta L(x_5)$$
$$\nabla_\theta L(x_6)$$
$$\nabla_\theta L(x_7)$$
$$\nabla_\theta L(x_8)$$

$$g = \frac{1}{N} \nabla_\theta \sum_{i=1}^{N} L(f(x_i, \theta), y_i)$$

$$\theta \leftarrow \theta - \varepsilon g$$

随机梯度下降



$$\nabla_\theta L(x_1)$$
$$\nabla_\theta L(x_2)$$
$$\nabla_\theta L(x_3)$$
$$\nabla_\theta L(x_4)$$
$$\nabla_\theta L(x_5)$$
$$\nabla_\theta L(x_6)$$
$$\nabla_\theta L(x_7)$$
$$\nabla_\theta L(x_8)$$

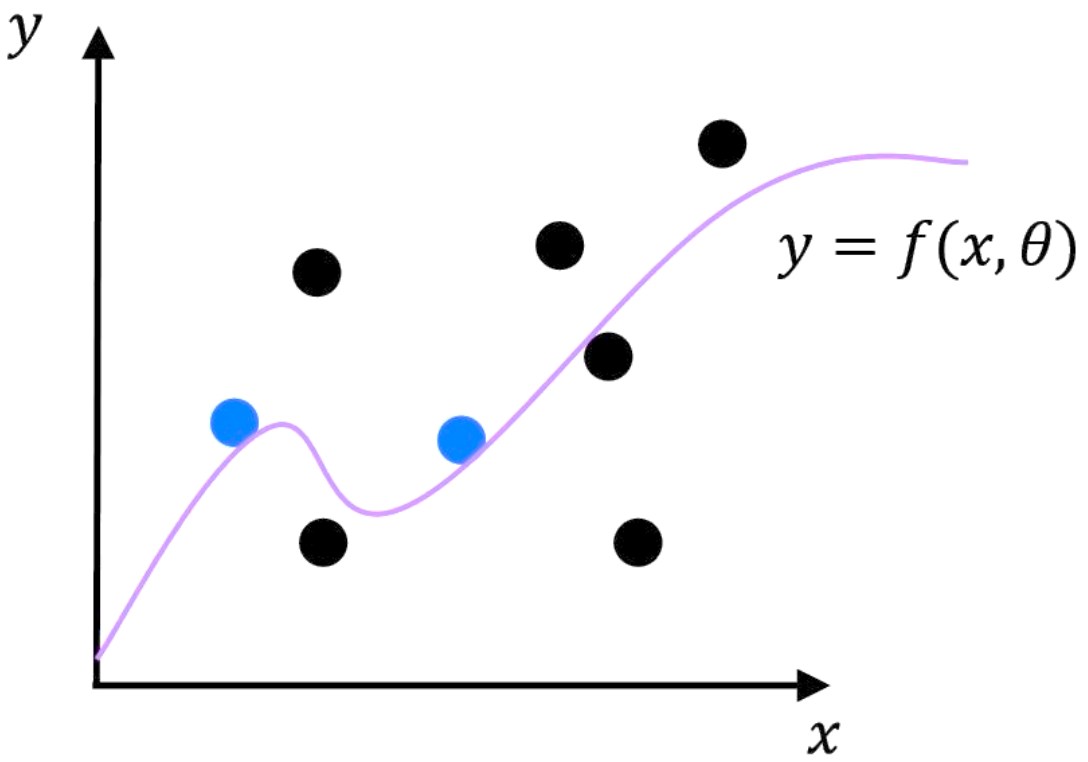
$$g = \frac{1}{N} \nabla_\theta \sum_{i=1}^{N} L(f(x_i, \theta), y_i)$$

$$\theta \leftarrow \theta - \varepsilon g$$

内存开销问题

更新速度慢

随机梯度下降

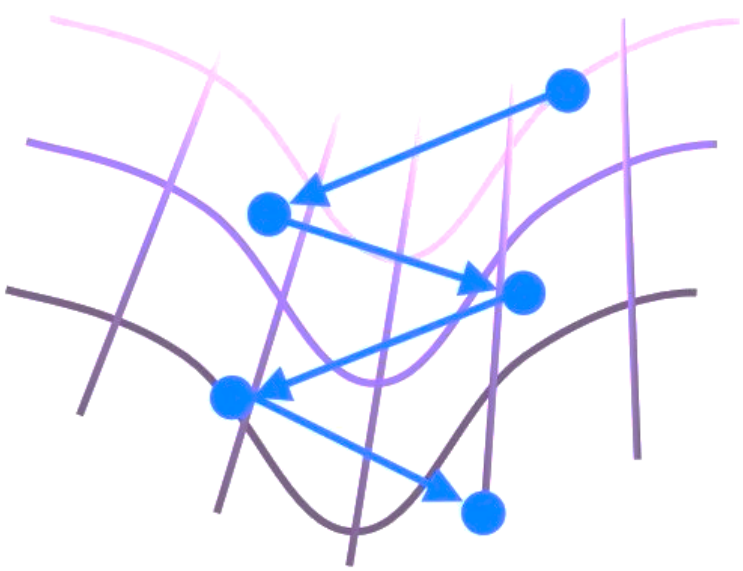


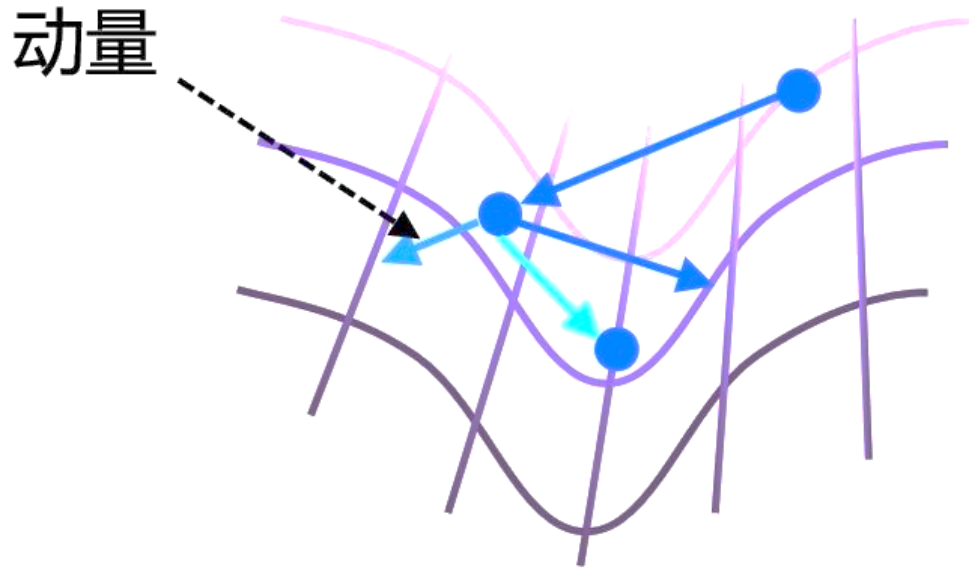
$$\nabla_\theta L(x_1)$$
$$\nabla_\theta L(x_2)$$
$$\nabla_\theta L(x_3)$$
$$\nabla_\theta L(x_4)$$
$$\nabla_\theta L(x_5)$$
$$\nabla_\theta L(x_6)$$
$$\nabla_\theta L(x_7)$$
$$\nabla_\theta L(x_8)$$

$$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

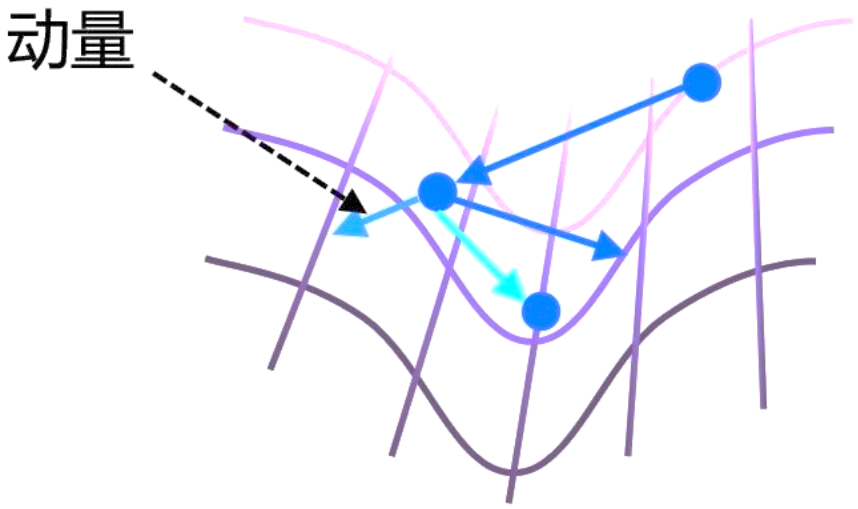
$$\theta \leftarrow \theta - \varepsilon g$$

动量梯度下降





动量梯度下降



$$g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

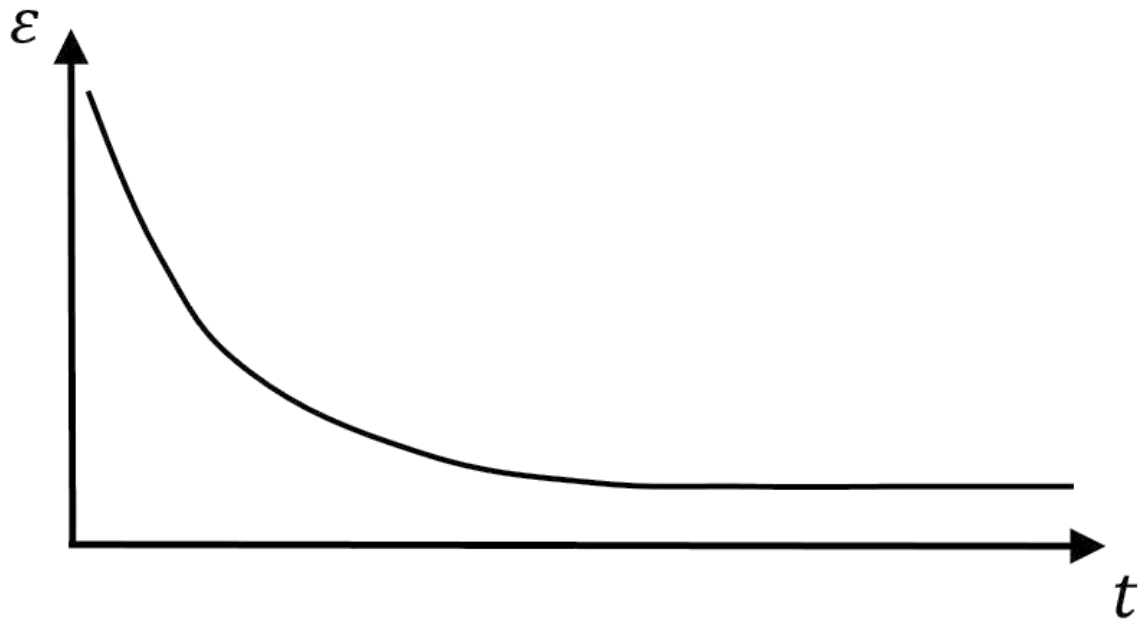
$v \leftarrow \alpha v - \varepsilon g$　　α控制动量

$\theta \leftarrow \theta + v$

自适应学习率

$$g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

$$\theta \leftarrow \theta - \varepsilon g$$



AdaGrad(2011)和RMSProp(2012)

$$g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

$$r \leftarrow r + g^2$$

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{r} + \delta} g$$

δ是一个小量，稳定数值计算

$$g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

$$r \leftarrow \rho r + (1 - \rho) g^2$$

$$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{r} + \delta} g$$

δ是一个小量，稳定数值计算

Adam(2014)

$$g = \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$$

$s \leftarrow \rho_1 s + (1-\rho_1) g$ ◄------- 自适应动量

$r \leftarrow \rho_2 r + (1-\rho_2) g^2$

$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$

$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$

$$\theta \leftarrow \theta - \frac{\varepsilon \hat{s}}{\sqrt{\hat{r}} + \delta} g$$

梯度下降

- 梯度下降算法(Gradient Descent,GD)，深度学习核心之一
- 随机梯度下降算法(Stochastic Gradient Descent,SGD)
- 动量(Moment)
- 自适应学习 AdaGrad和 RMSProp 算法
- 动量 & 自适应学习 Adam

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
  - 反向传播算法(Back-Propagation,BP)，深度学习核心之一
  - 加速计算参数梯度值的方法

矩阵微积分

- 矩阵微积分（Matrix Calculus）是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。
- 分母布局
  - 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

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

链式法则

- 链式法则（Chain Rule）是在微积分中求复合函数导数的一种常用方法。

（1）若 $x \in \mathbb{R}, \boldsymbol{u} = u(x) \in \mathbb{R}^s, \boldsymbol{g} = g(\boldsymbol{u}) \in \mathbb{R}^t$，则

$$\frac{\partial \boldsymbol{g}}{\partial x} = \frac{\partial \boldsymbol{u}}{\partial x} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} \quad \in \mathbb{R}^{1 \times t}.$$

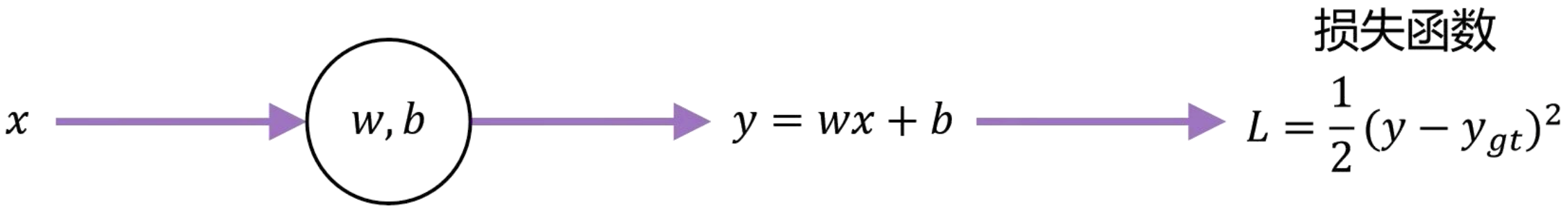
（2）若 $\boldsymbol{x} \in \mathbb{R}^p, \boldsymbol{y} = g(\boldsymbol{x}) \in \mathbb{R}^s, \boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^t$，则

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \quad \in \mathbb{R}^{p \times t}.$$

（3）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$\boldsymbol{y} = g(X) \in \mathbb{R}^s, z = f(\boldsymbol{y}) \in \mathbb{R}$，则

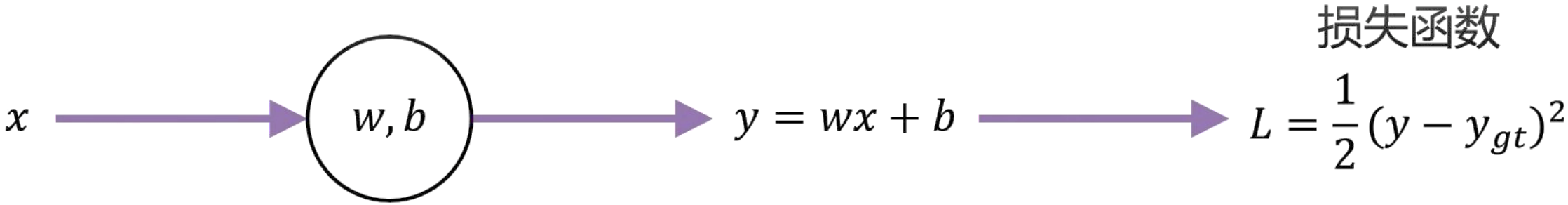
$$\frac{\partial z}{\partial X_{ij}} = \frac{\partial \boldsymbol{y}}{\partial X_{ij}} \frac{\partial z}{\partial \boldsymbol{y}} \quad \in \mathbb{R}.$$

反向传播算法



x	w	b	y	L	y_{gt}
1.5	0.8	0.2	$0.8 \times 1.5 + 0.2 = 1.4$	0.18	0.8

反向传播算法

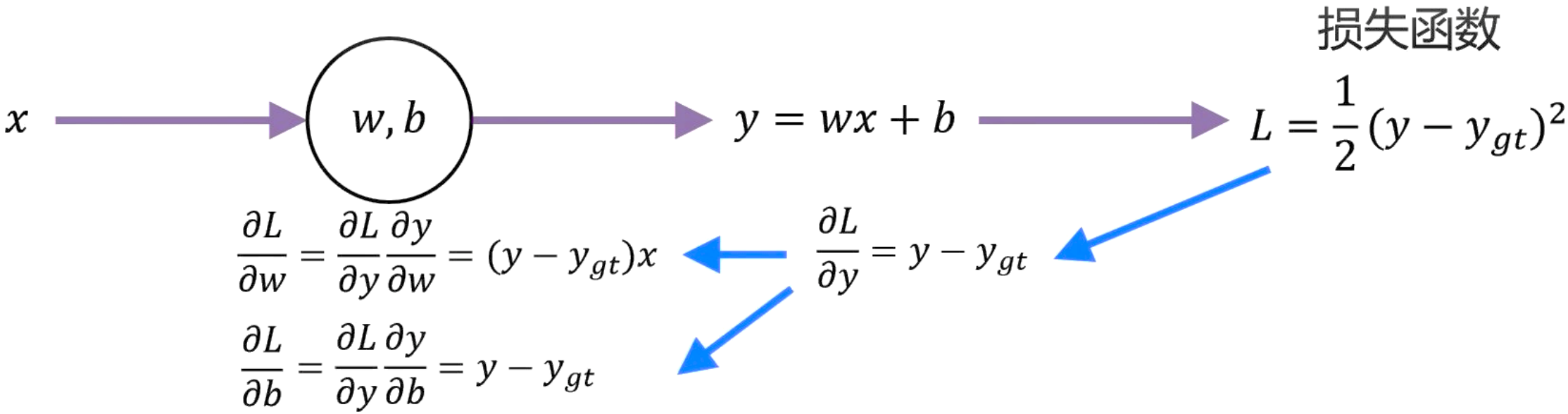


$$\frac{\partial L}{\partial w} \quad w \leftarrow w - \varepsilon \frac{\partial L}{\partial w}$$

$$\frac{\partial L}{\partial b} \quad b \leftarrow b - \varepsilon \frac{\partial L}{\partial b}$$

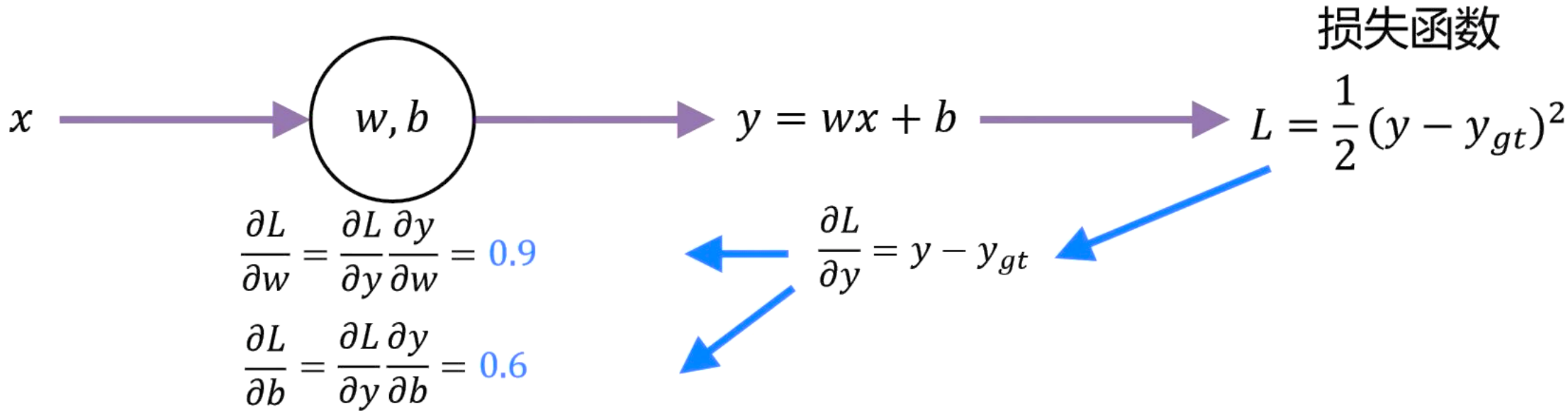
x	w	b	y	L	y_{gt}
1.5	0.8	0.2	$0.8 \times 1.5 + 0.2 = 1.4$	0.18	0.8

反向传播算法



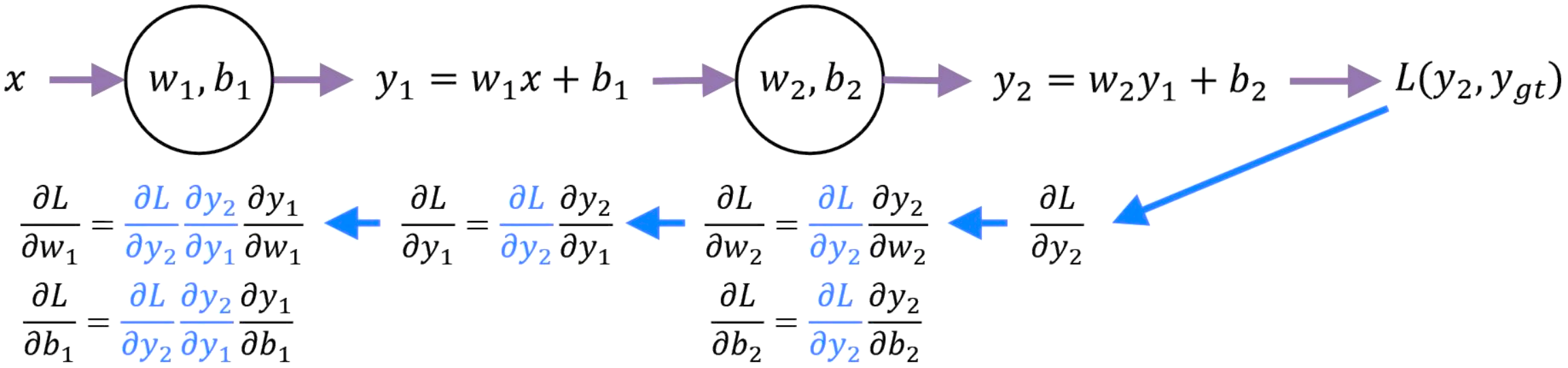
x	w	b	y	L	y_{gt}
1.5	0.8	0.2	$0.8 \times 1.5 + 0.2 = 1.4$	0.18	0.8

反向传播算法

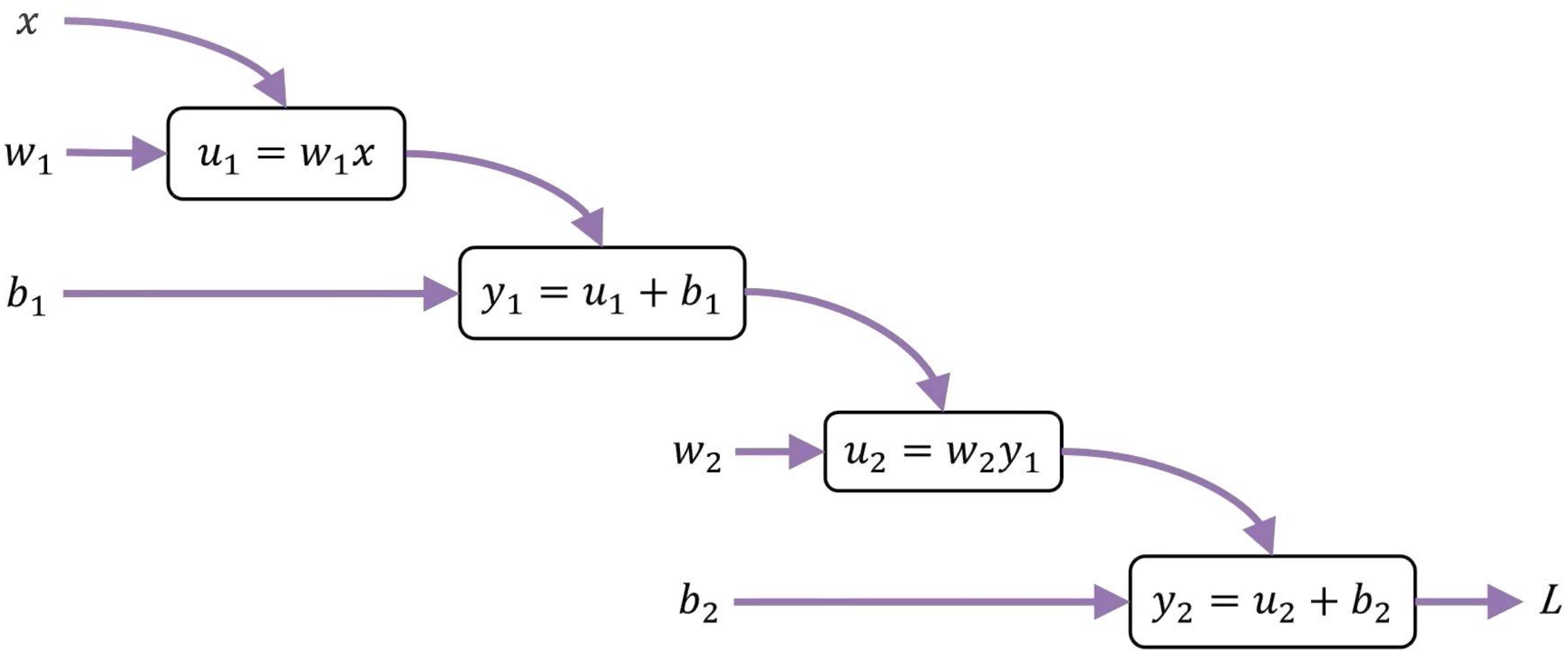


ε	x	w	b	y	L	y_{gt}
	1.5	0.8	0.2	$0.8 \times 1.5 + 0.2 = 1.4$	0.18	0.8
0.1	1.5	0.71	0.14	$0.71 \times 1.5 + 0.14 = 1.205$	0.082	0.8

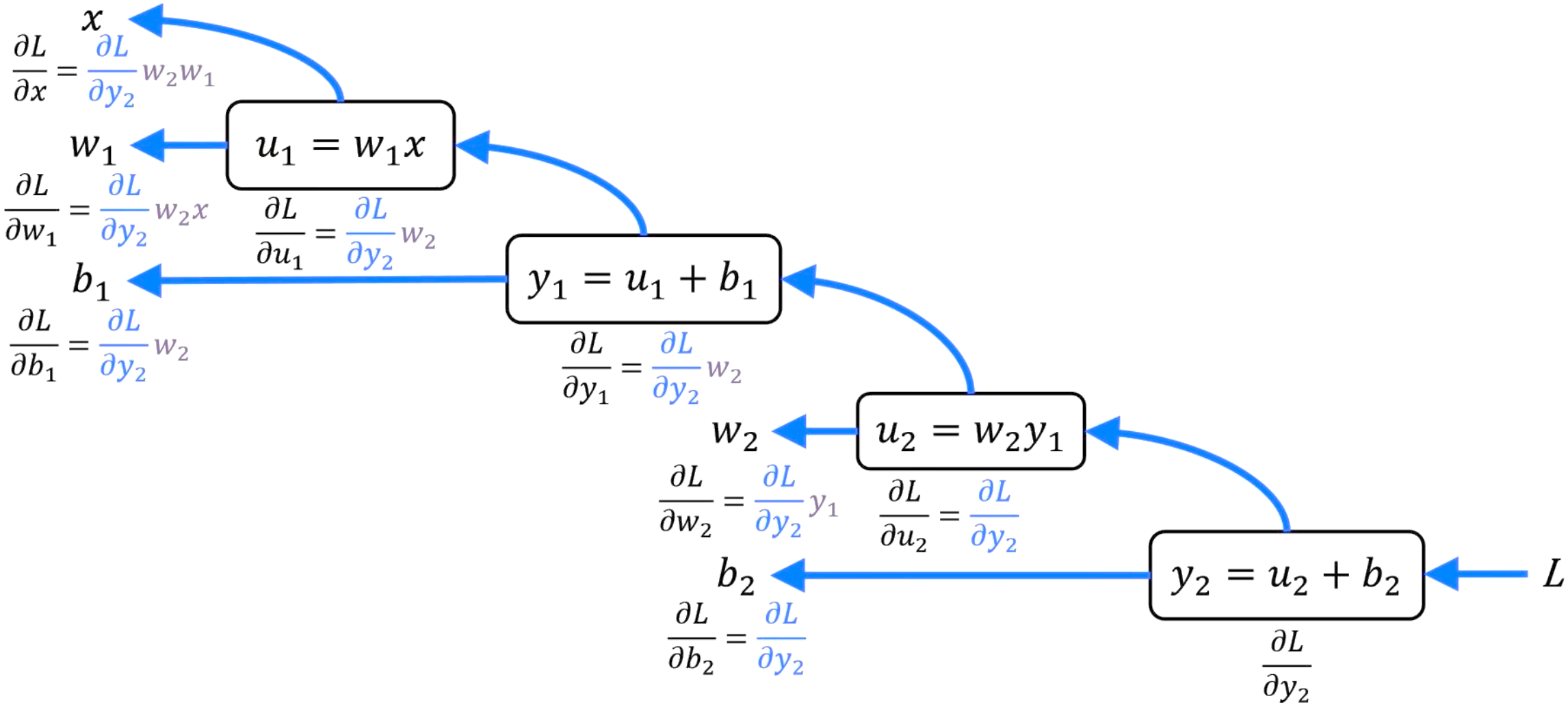
反向传播算法


x
w_1, b_1
$y_1 = w_1 x + b_1$
w_2, b_2
$y_2 = w_2 y_1 + b_2$
$L(y_2, y_{gt})$
$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial y_1}\frac{\partial y_1}{\partial w_1}$
$\frac{\partial L}{\partial y_1} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial y_1}$
$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial w_2}$
$\frac{\partial L}{\partial y_2}$
$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial y_1}\frac{\partial y_1}{\partial b_1}$
$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial b_2}$

反向传播算法：计算图和自动微分



反向传播算法：计算图和自动微分

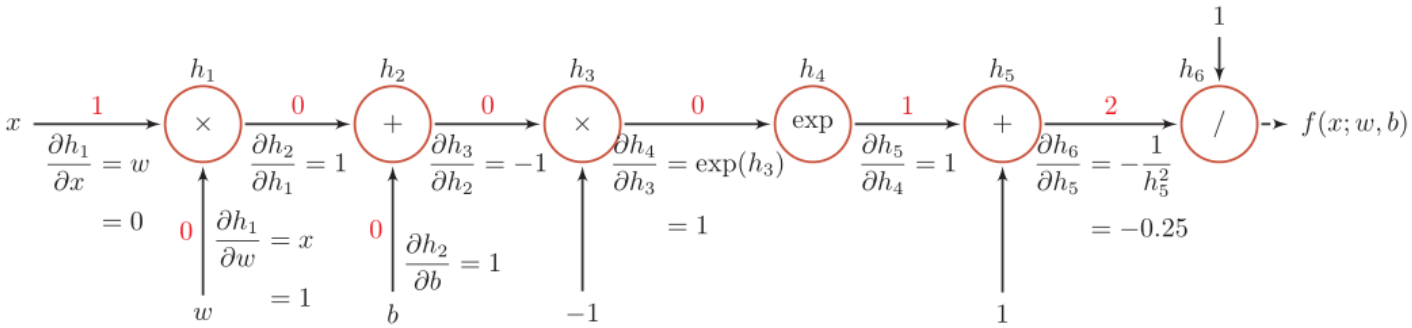


反向传播算法：计算图和自动微分

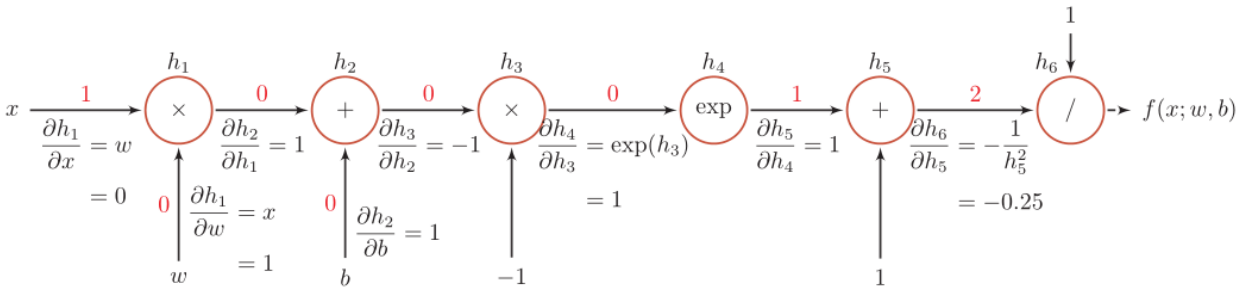
- 自动微分是利用链式法则来自动计算一个复合函数的梯度。

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

- 计算图



计算图



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

当x = 1,w = 0,b = 0时，可以得到

$$\begin{aligned}\frac{\partial f(x;w,b)}{\partial w}|_{x=1,w=0,b=0} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} \\ &= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1 \\ &= 0.25.\end{aligned}$$

反向传播算法：计算图和自动微分

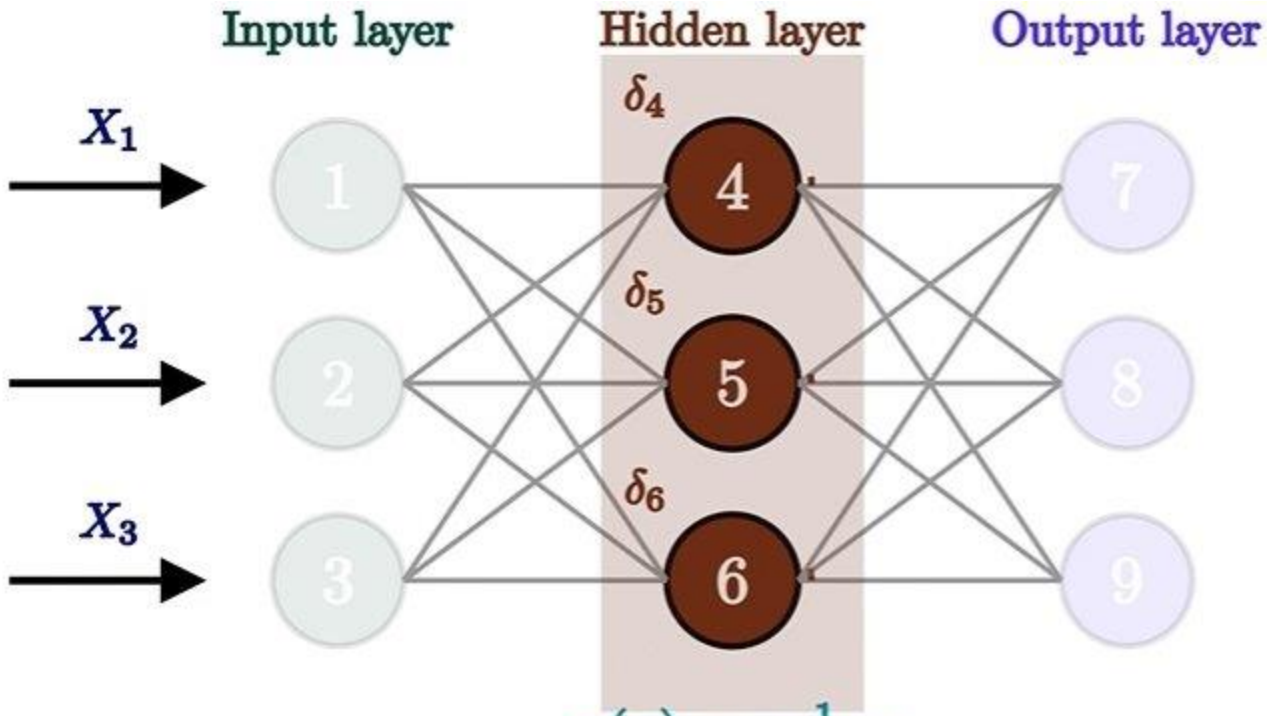
```python
class Matmul:
    def __init__(self):
        self.mem = {}

    def forward(self, x, W):
        h = np.matmul(x, W)
        self.mem={'x': x, 'W':W}
        return h

    def backward(self, grad_y):
        x = self.mem['x']
        W = self.mem['W']

        grad_W = np.matmul(x.T,grad_y)
        grad_x = np.matmul(grad_y,W.T)

        return grad_x, grad_W
```

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Tensorflow
- 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和PyTorch

- 静态计算图在构建时可以进行优化，并行能力强，但灵活性比较差低。动态计算图则不容易优化，当不同输入的网络结构不一致时，难以并行计算，但是灵活性比较高。

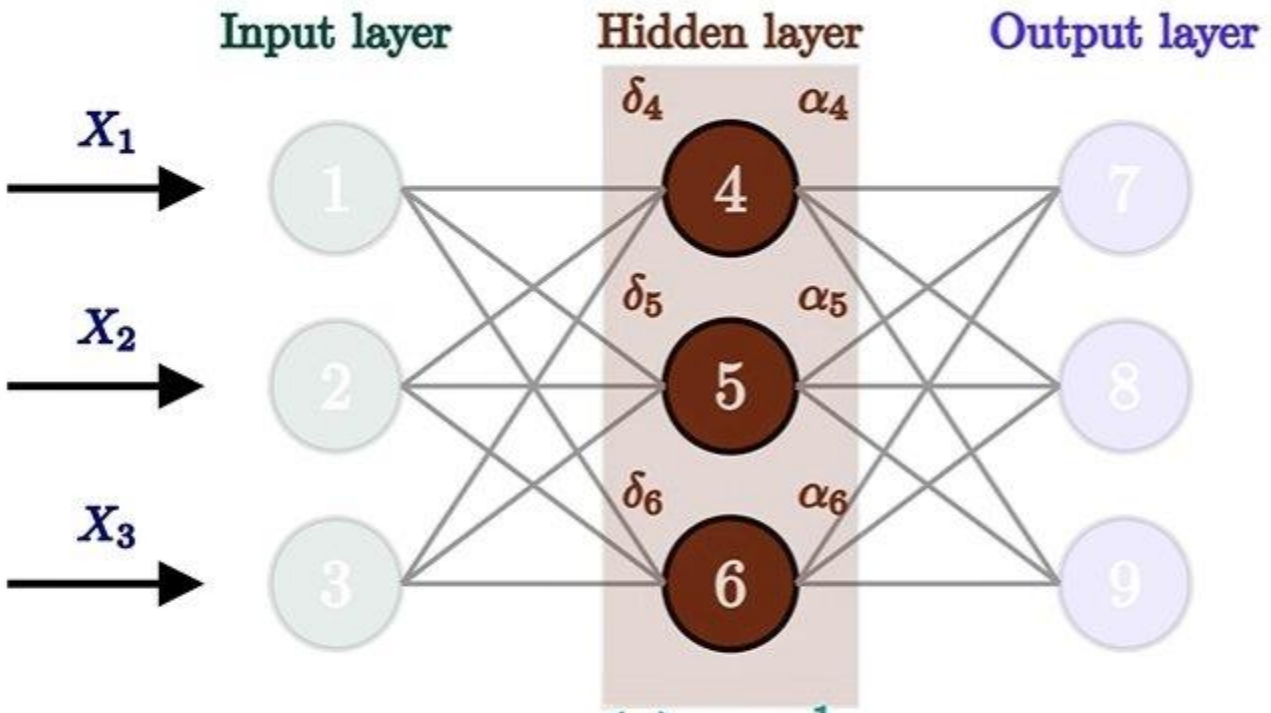
神经网络梯度计算



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

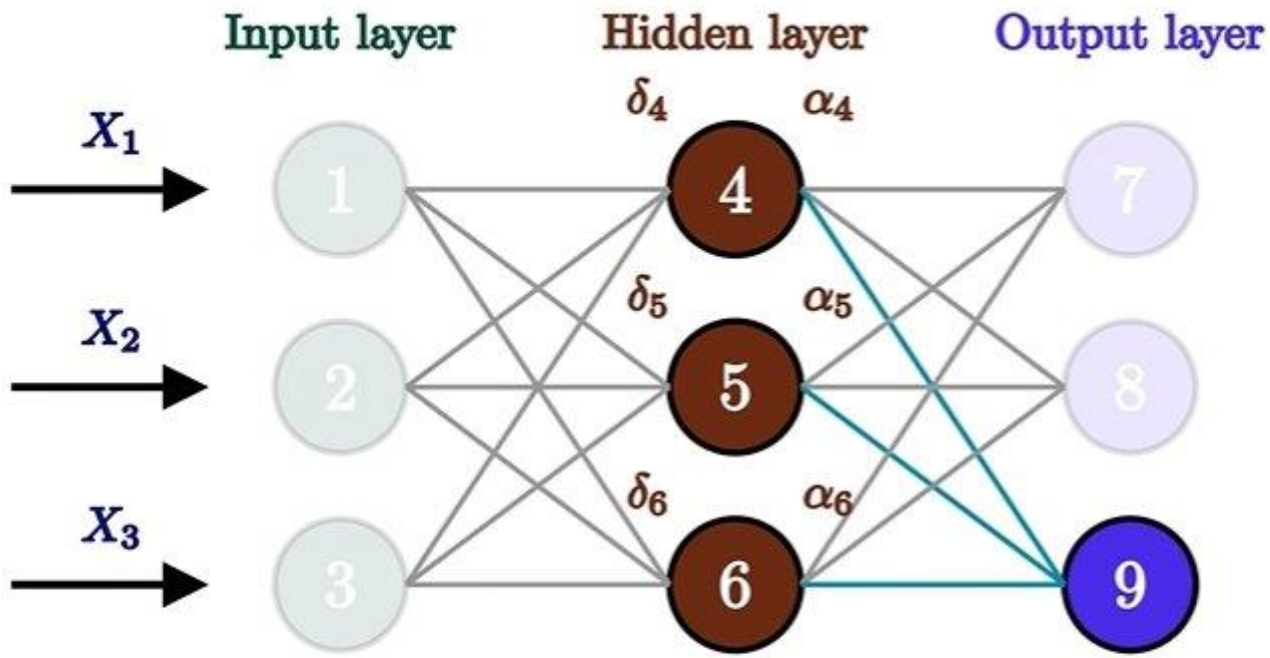
$$\delta_4 = \begin{bmatrix} \omega_{14} & \omega_{24} & \omega_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \delta_5 = \begin{bmatrix} \omega_{15} & \omega_{25} & \omega_{35} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \delta_6 = \begin{bmatrix} \omega_{16} & \omega_{26} & \omega_{36} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

神经网络梯度计算



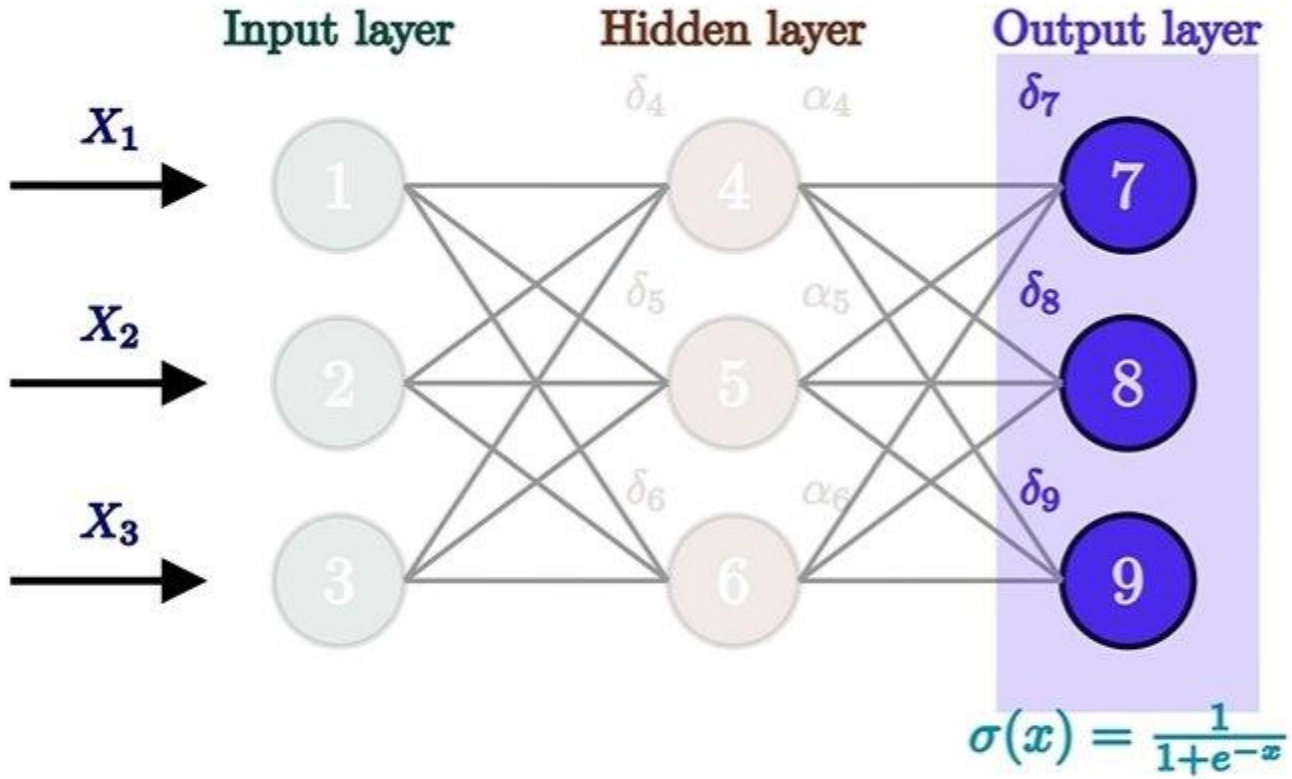
$$\alpha_4 = sigmoid(\delta_4)\ \alpha_5 = sigmoid(\delta_5)\ \alpha_6 = sigmoid(\delta_6)$$

神经网络梯度计算



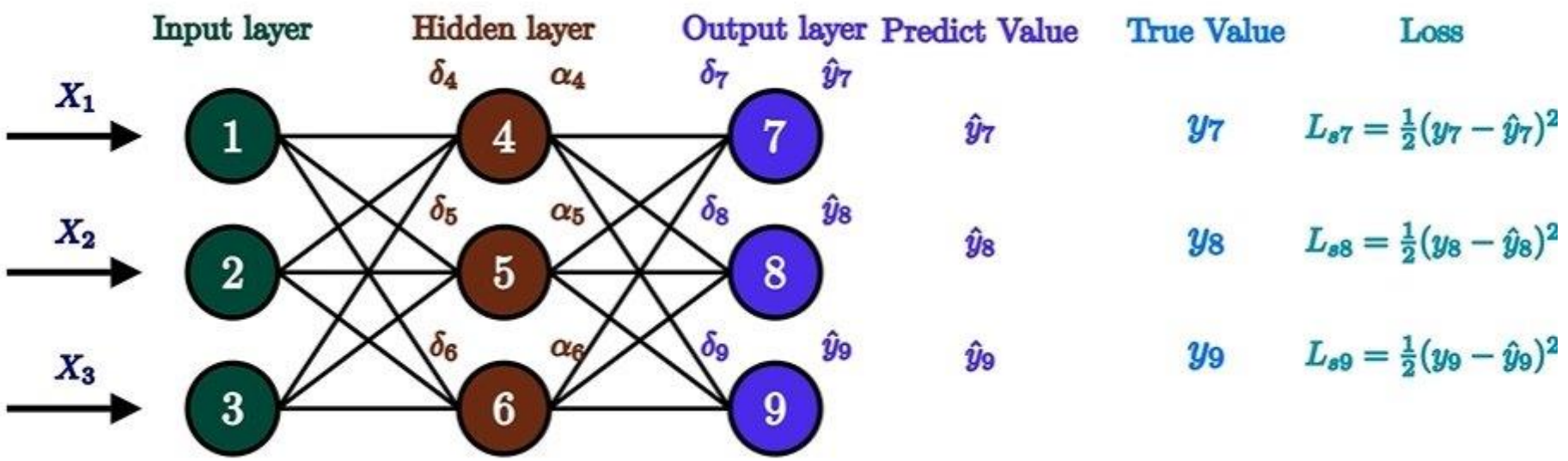
$$\delta_7 = \begin{bmatrix} \omega_{47} & \omega_{57} & \omega_{67} \end{bmatrix} \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} \quad \delta_8 = \begin{bmatrix} \omega_{48} & \omega_{58} & \omega_{68} \end{bmatrix} \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} \quad \delta_9 = \begin{bmatrix} \omega_{49} & \omega_{59} & \omega_{69} \end{bmatrix} \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}$$

神经网络梯度计算

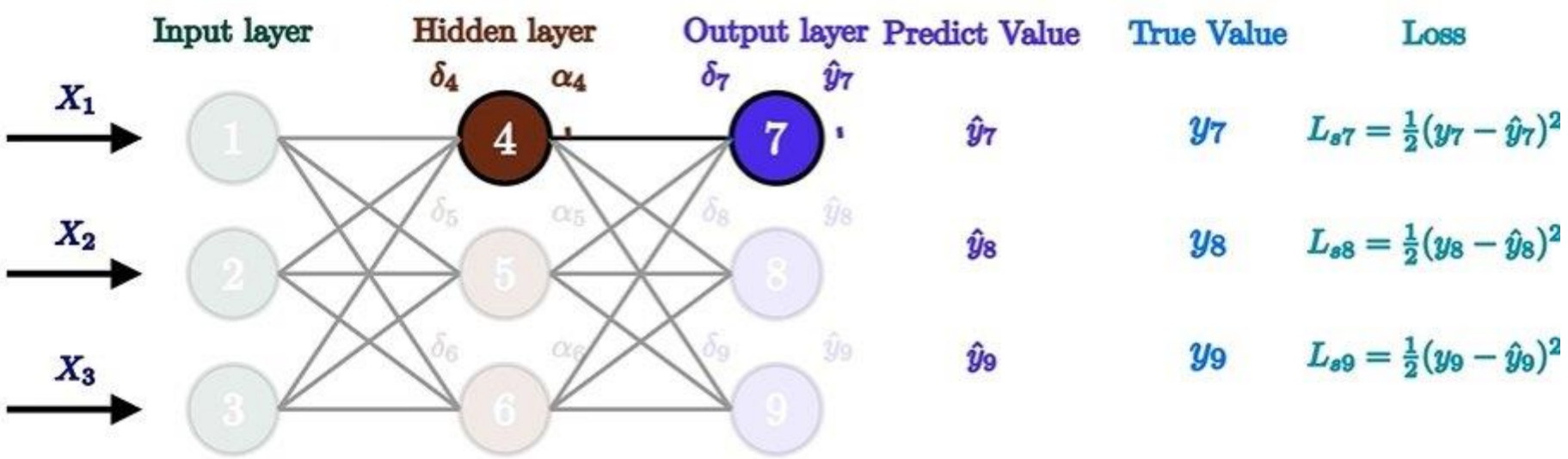


$$\hat{y}_7 = sigmoid(\delta_7)\ \hat{y}_8 = sigmoid(\delta_8)\ \hat{y}_9 = sigmoid(\delta_9)$$

神经网络梯度计算


Input layer
Hidden layer
Output layer
Predict Value
True Value
Loss
X_1
X_2
X_3
1
2
3
4
5
6
7
8
9
δ_4
α_4
δ_5
α_5
δ_6
α_6
δ_7
$\hat{y}_7$
δ_8
$\hat{y}_8$
δ_9
$\hat{y}_9$
$\hat{y}_7$
y_7
$L_{s7} = \frac{1}{2}(y_7 - \hat{y}_7)^2$
$\hat{y}_8$
y_8
$L_{s8} = \frac{1}{2}(y_8 - \hat{y}_8)^2$
$\hat{y}_9$
y_9
$L_{s9} = \frac{1}{2}(y_9 - \hat{y}_9)^2$

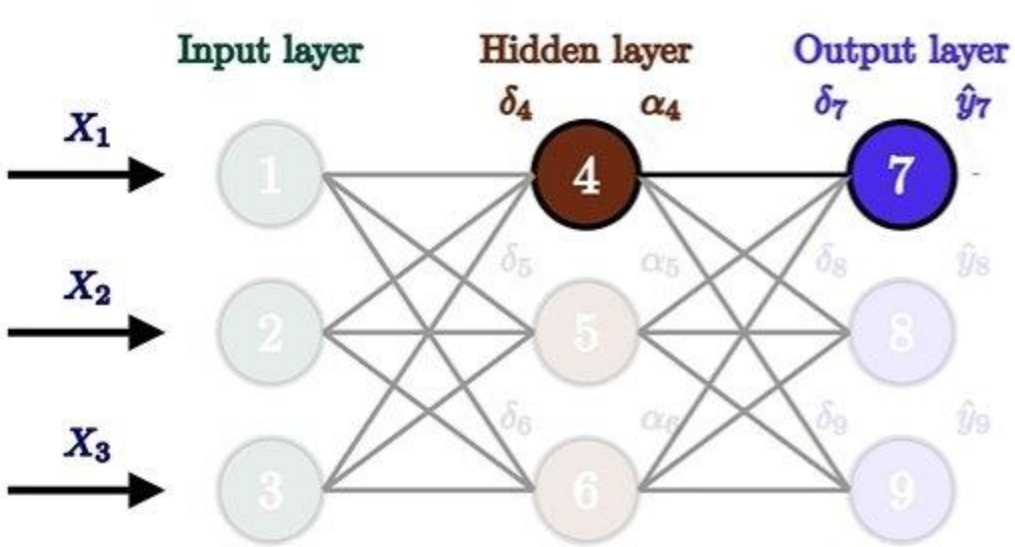
神经网络梯度计算



$$\frac{\partial L_{s7}}{\partial \omega_{47}} = \frac{\partial L_{s7}}{\partial \hat{y}_7} \frac{\partial \hat{y}_7}{\partial \delta_7} \frac{\partial \delta_7}{\partial \omega_{47}}$$

$$\frac{\partial \frac{1}{2}(y_7 - \hat{y}_7)^2}{\partial y_7} \qquad \frac{\partial\left(\frac{1}{1+e^{-\delta_7}}\right)}{\partial \delta_7} \qquad \frac{\partial \begin{bmatrix} \omega_{47} & \omega_{57} & \omega_{67} \end{bmatrix} \begin{bmatrix} \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix}}{\partial \omega_{47}}$$

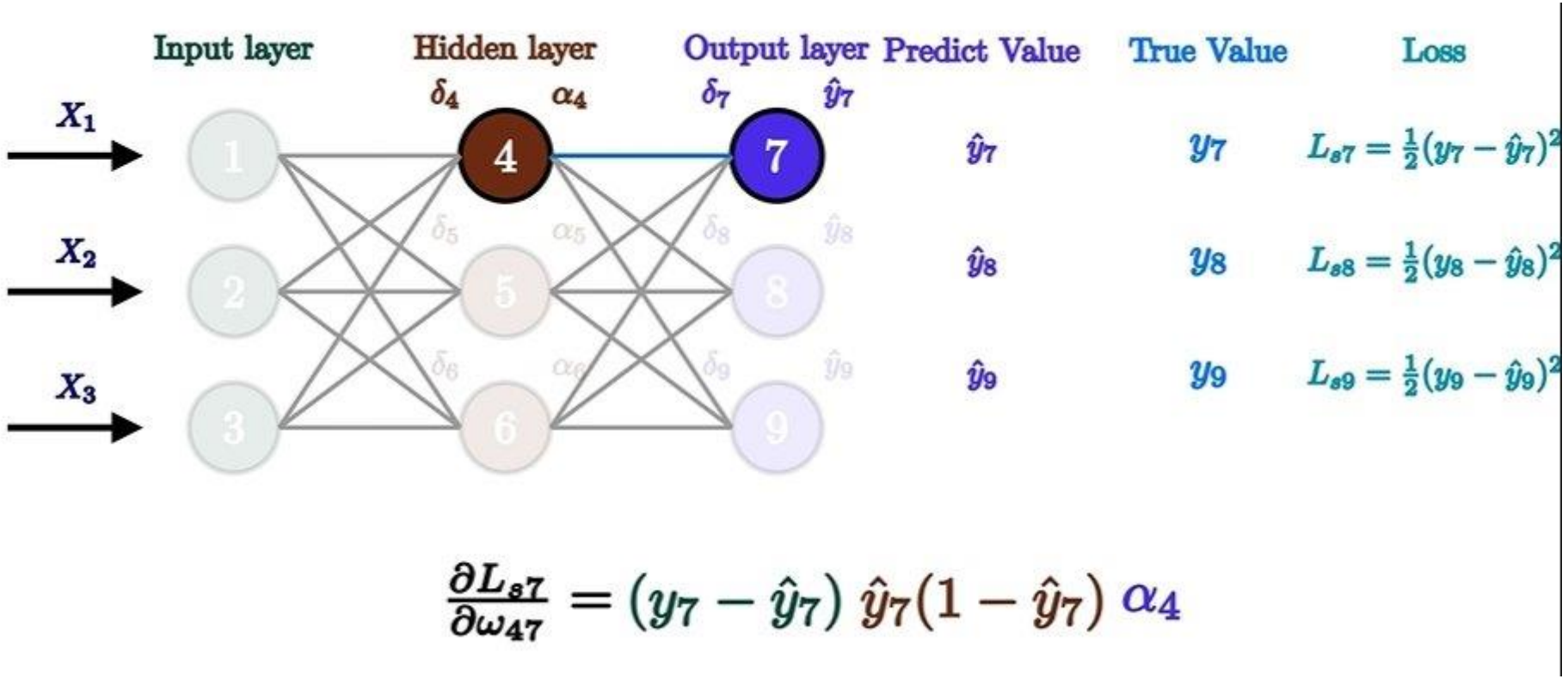
神经网络梯度计算



$$\frac{\partial L_{s7}}{\partial \omega_{47}} = \frac{\partial L_{s7}}{\partial \hat{y}_7} \frac{\partial \hat{y}_7}{\partial \delta_7} \frac{\partial \delta_7}{\partial \omega_{47}}$$

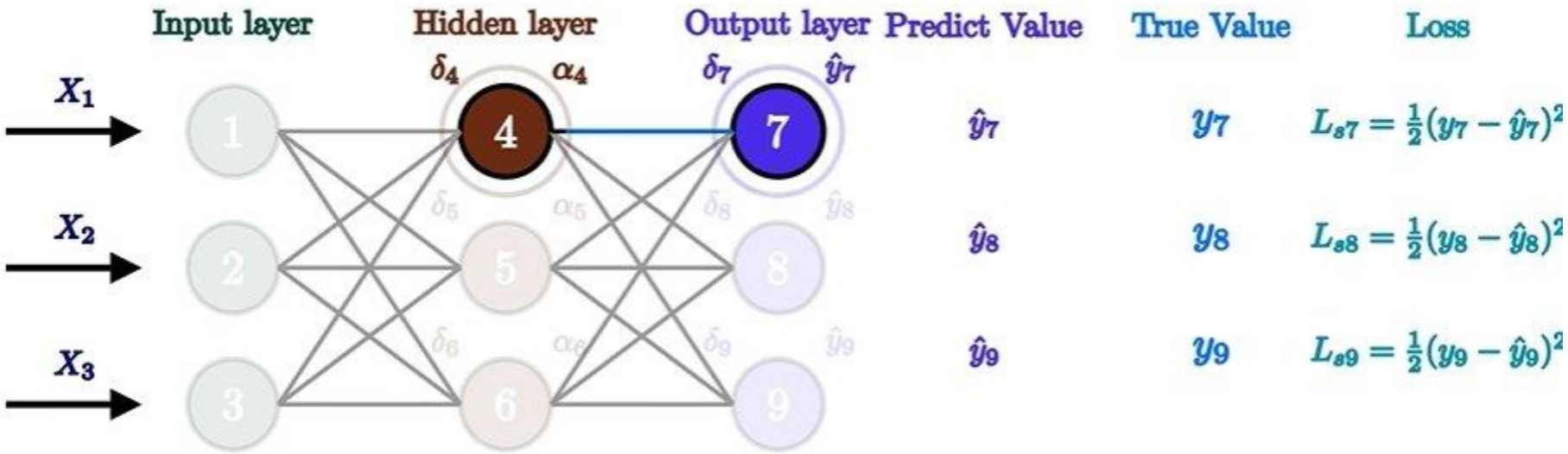
$$(y_7 - \hat{y}_7) \qquad \hat{y}_7(1 - \hat{y}_7) \qquad \alpha_4$$

神经网络梯度计算



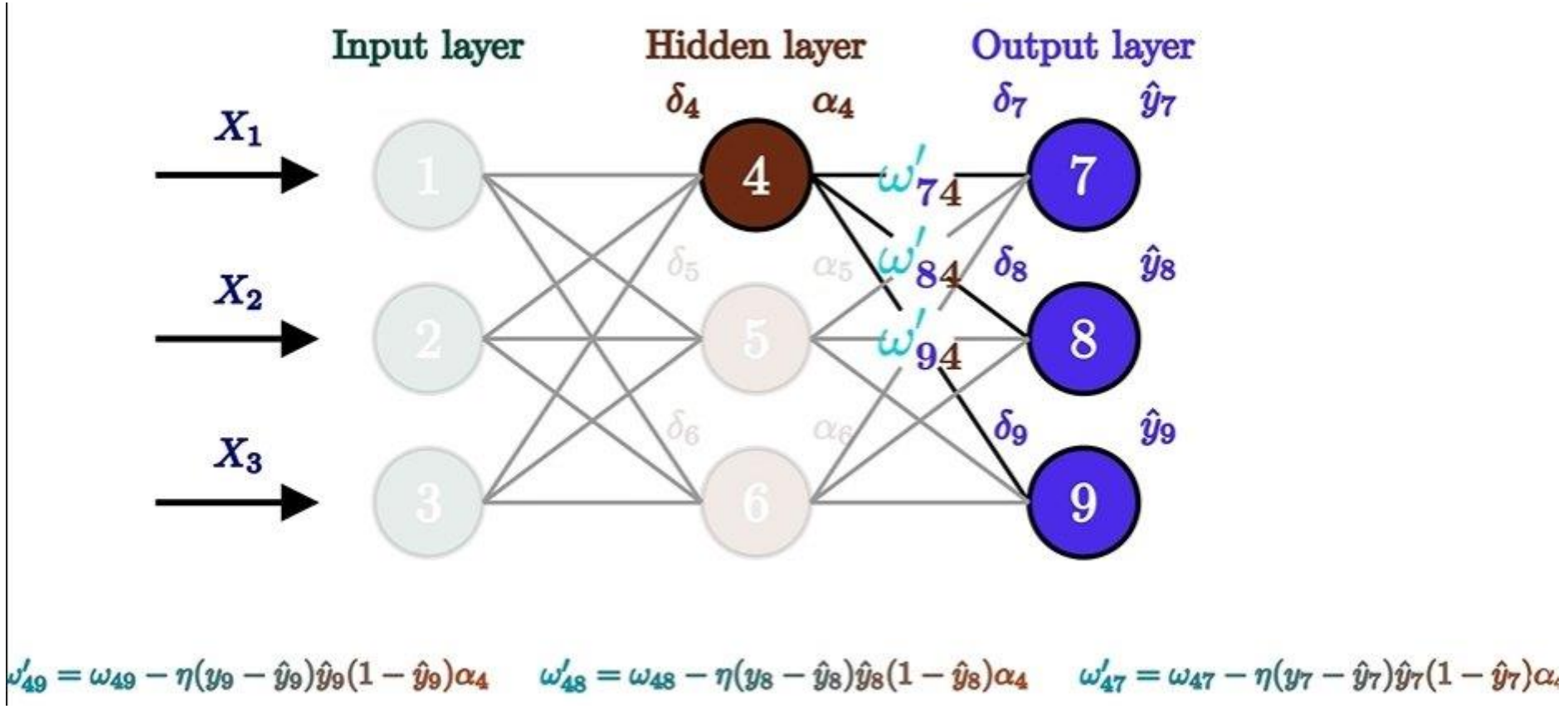
$$\frac{\partial L_{s7}}{\partial \omega_{47}} = (y_7 - \hat{y}_7)\,\hat{y}_7(1 - \hat{y}_7)\,\alpha_4$$

神经网络梯度计算



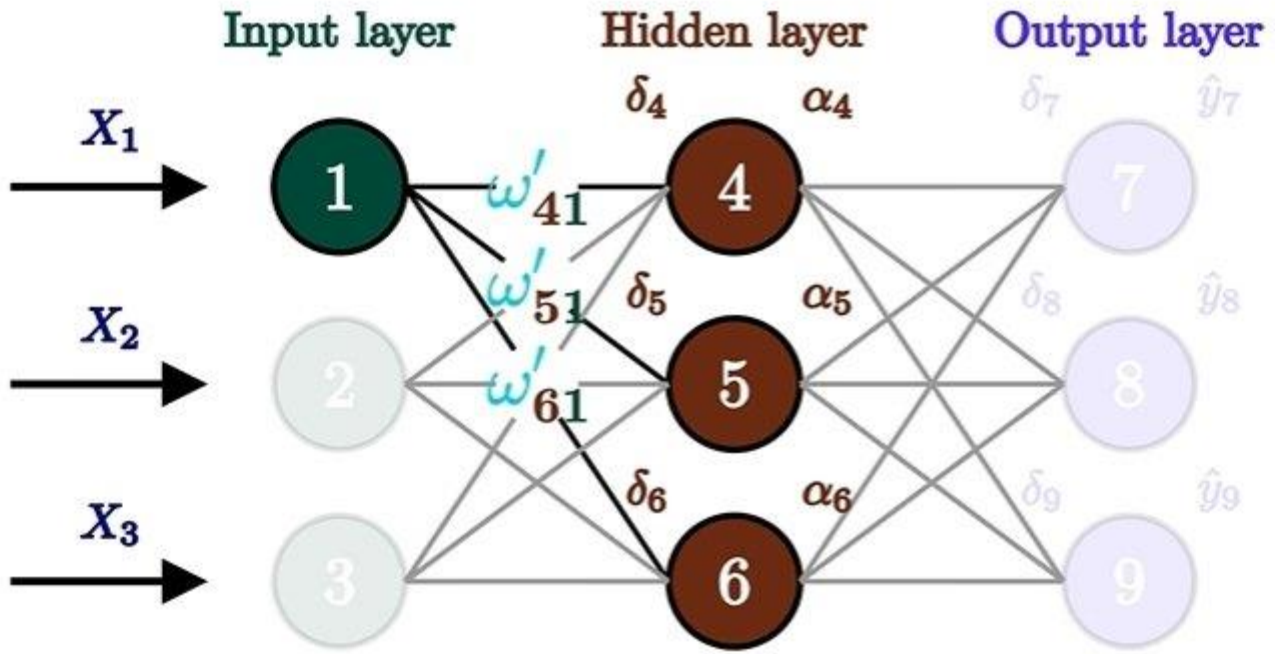
$$\omega'_{47} = \omega_{47} - \eta \frac{\partial L_{s7}}{\partial \omega_{47}} = (y_7 - \hat{y}_7)\,\hat{y}_7(1 - \hat{y}_7)\,\alpha_4$$

神经网络梯度计算



$\omega'_{49} = \omega_{49} - \eta(y_9 - \hat{y}_9)\hat{y}_9(1 - \hat{y}_9)\alpha_4$ $\omega'_{48} = \omega_{48} - \eta(y_8 - \hat{y}_8)\hat{y}_8(1 - \hat{y}_8)\alpha_4$ $\omega'_{47} = \omega_{47} - \eta(y_7 - \hat{y}_7)\hat{y}_7(1 - \hat{y}_7)\alpha_4$

神经网络梯度计算



$$\omega'_{14} = \omega_{14} - \eta \sum_1^n \frac{\partial L_{si}}{\partial \omega_{14}} \qquad \omega'_{15} = \omega_{15} - \eta \sum_1^n \frac{\partial L_{si}}{\partial \omega_{15}} \qquad \omega'_{16} = \omega_{16} - \eta \sum_1^n \frac{\partial L_{si}}{\partial \omega_{16}}$$

计算图与自动微分

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。

反向传播算法 (自动微分的反向模式)

- 前馈神经网络的训练过程可以分为以下三步
  - 前向计算每一层的状态和激活值，直到最后一层
  - 反向计算每一层的参数的偏导数
  - 更新参数

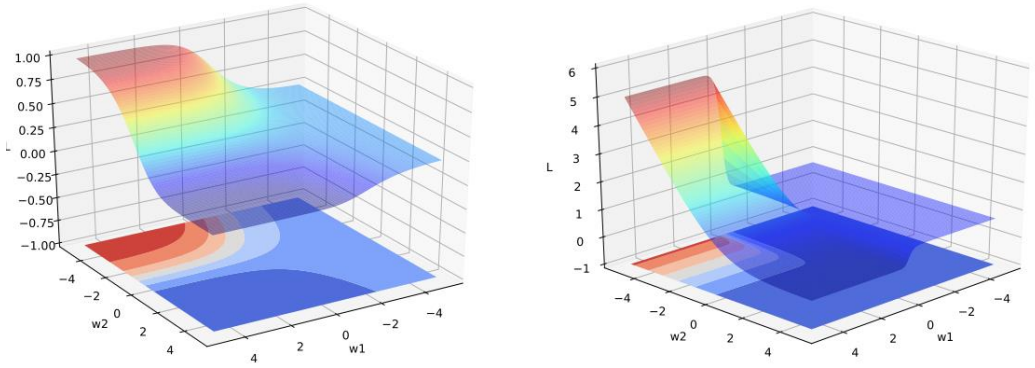
优化问题

优化问题

- 难点
  - 参数过多，影响训练
  - 非凸优化问题：即存在局部最优而非全局最优解，影响迭代
  - 梯度消失问题，下层参数比较难调
  - 参数解释起来比较困难
- 需求
  - 计算资源要大
  - 数据要多
  - 算法效率要好：即收敛快

优化问题

- 非凸优化问题



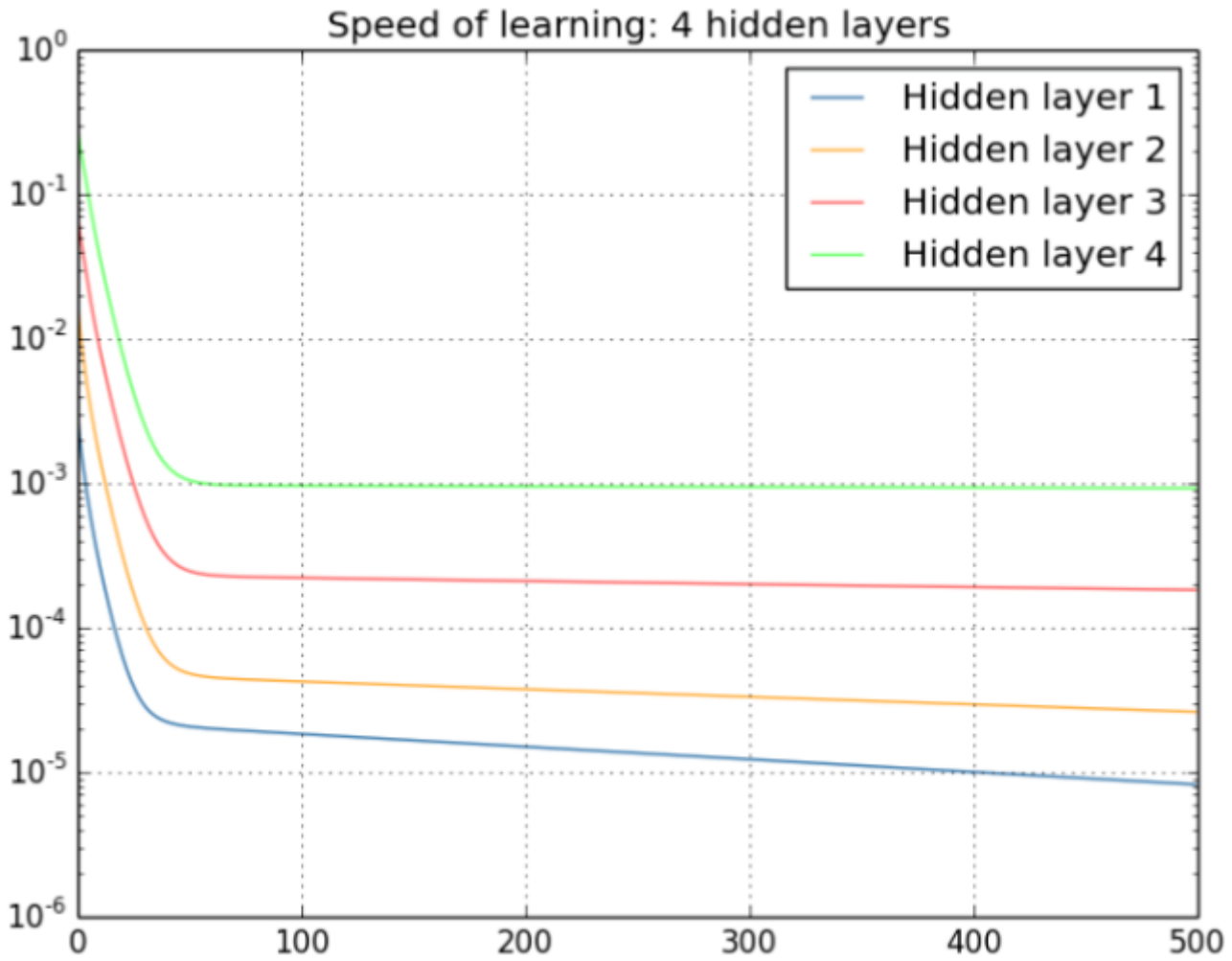
(a) 平方误差损失　　(b) 交叉熵损失

图 4.9 神经网络 $y = \sigma(w_2\sigma(w_1x))$ 的损失函数

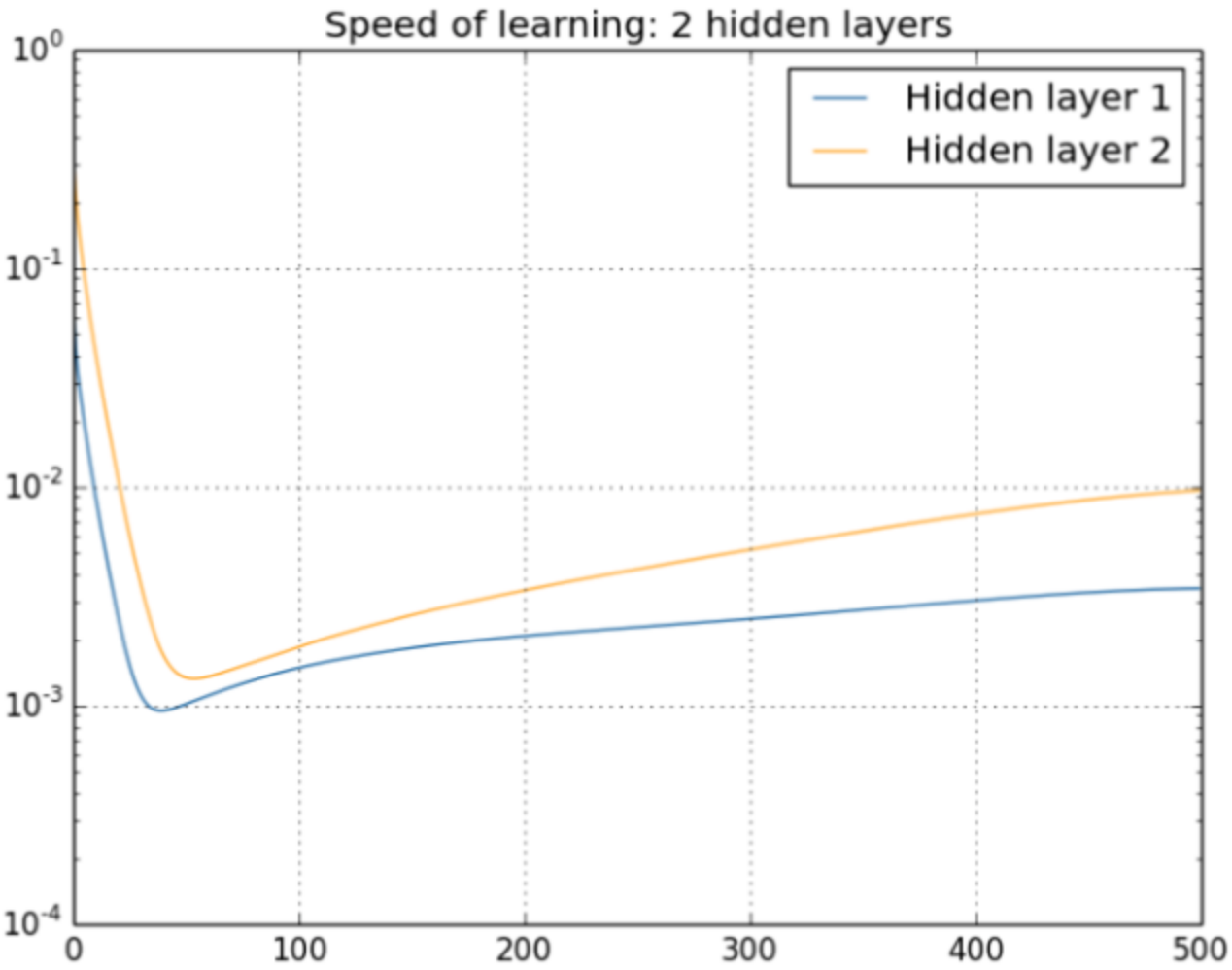
梯度消失和梯度爆炸

- 梯度消失：
  - 在神经网络中，当前面隐藏层的学习速率低于后面隐藏层的学习速率，即随着隐藏层数目的增加，分类准确率反而下降了。这种现象叫梯度消失。
- 梯度爆炸：
  - 在神经网络中，当前面隐藏层的学习速率低于后面隐藏层的学习速率，即随着隐藏层数目的增加，分类准确率反而下降了。这种现象叫梯度爆炸。
- 其实梯度消失和梯度爆炸是一回事，只是表现的形式，以及产生的原因不一样。

梯度消失

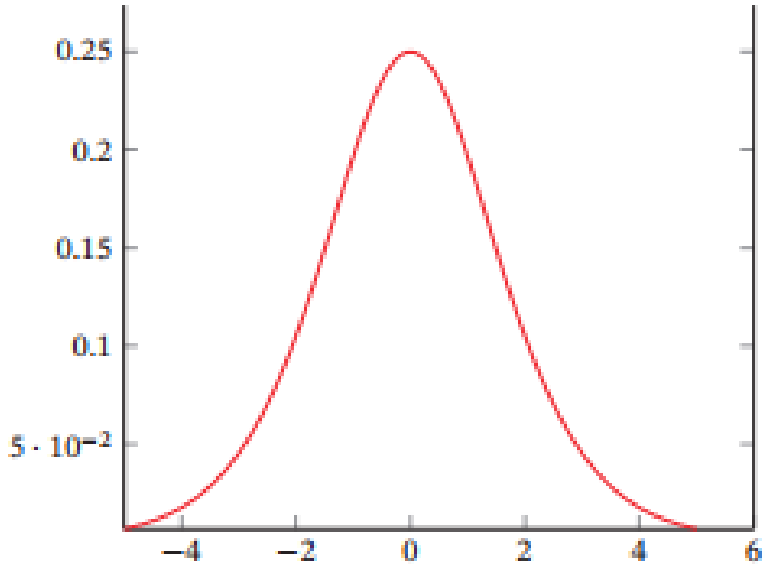

Speed of learning: 4 hidden layers
Hidden layer 1
Hidden layer 2
Hidden layer 3
Hidden layer 4
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
100
200
300
400
500

梯度爆炸

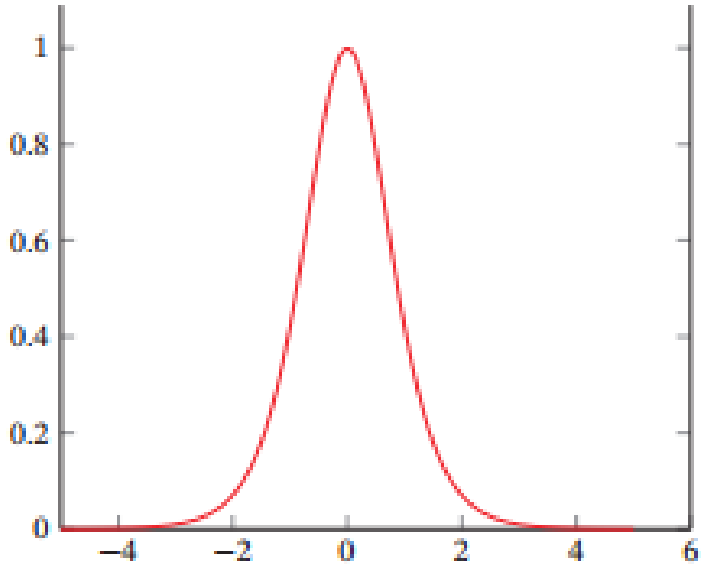

Speed of learning: 2 hidden layers
Hidden layer 1
Hidden layer 2
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
100
200
300
400
500

梯度消失

- 由于 Sigmoid 型函数的饱和性，饱和区的导数更是接近于 0. 这样，误差经过每一层传递都会不断衰减. 当网络层数很深时，梯度就会不停衰减，甚至消失，使得整个网络很难训练.



(a) Logistic 函数的导数



(b) Tanh 函数的导数

梯度消失和梯度爆炸

- 梯度消失:
  - (1) 隐藏层的层数过多;
  - (2) 采用了不合适的激活函数(更容易产生梯度消失，但是也有可能产生梯度爆炸)
  - (3) 在深度神经网络中， 减轻梯度消失问题的方法有很多种． 一种简单有效的方式是使用导数比较大的激活函数， 比如ReLU等．

- 梯度爆炸:
  - (1) 隐藏层的层数过多;
  - (2) 权重的初始化值过大