《人工智能数学原理与算法》
第 3 章：神经网络基础

3.2 神经网络优化

连德富
liandefu@ustc.edu.cn

目录

目录

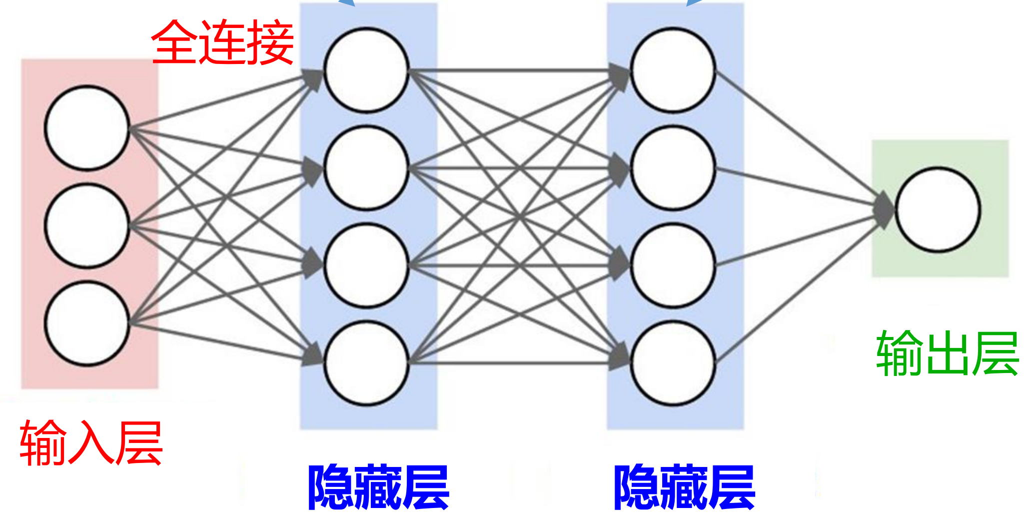
前馈神经网络（多层感知机）

口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

口整个网络中无反馈，信号从输入层向输出层单向传播

$$\boldsymbol{a}^{(1)} = f(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$$

$$\boldsymbol{a}^{(2)} = f(\boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} + b^{(2)})$$



一般层数L只考虑
隐藏层和输出层

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）

梯度下降（Gradient Descent）

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$输入给前馈神经网络，得到网络输出为 $\hat{\boldsymbol{y}}^{(n)}$，其在数据集D上的结构化风险函数为：

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right) + \frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2$$

- 梯度下降 (Gradient Descent, GD)

$$\boldsymbol{\theta}^{(l+1)} \leftarrow \boldsymbol{\theta}^{(l)} - \alpha\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{\theta}^{(l)}}$$

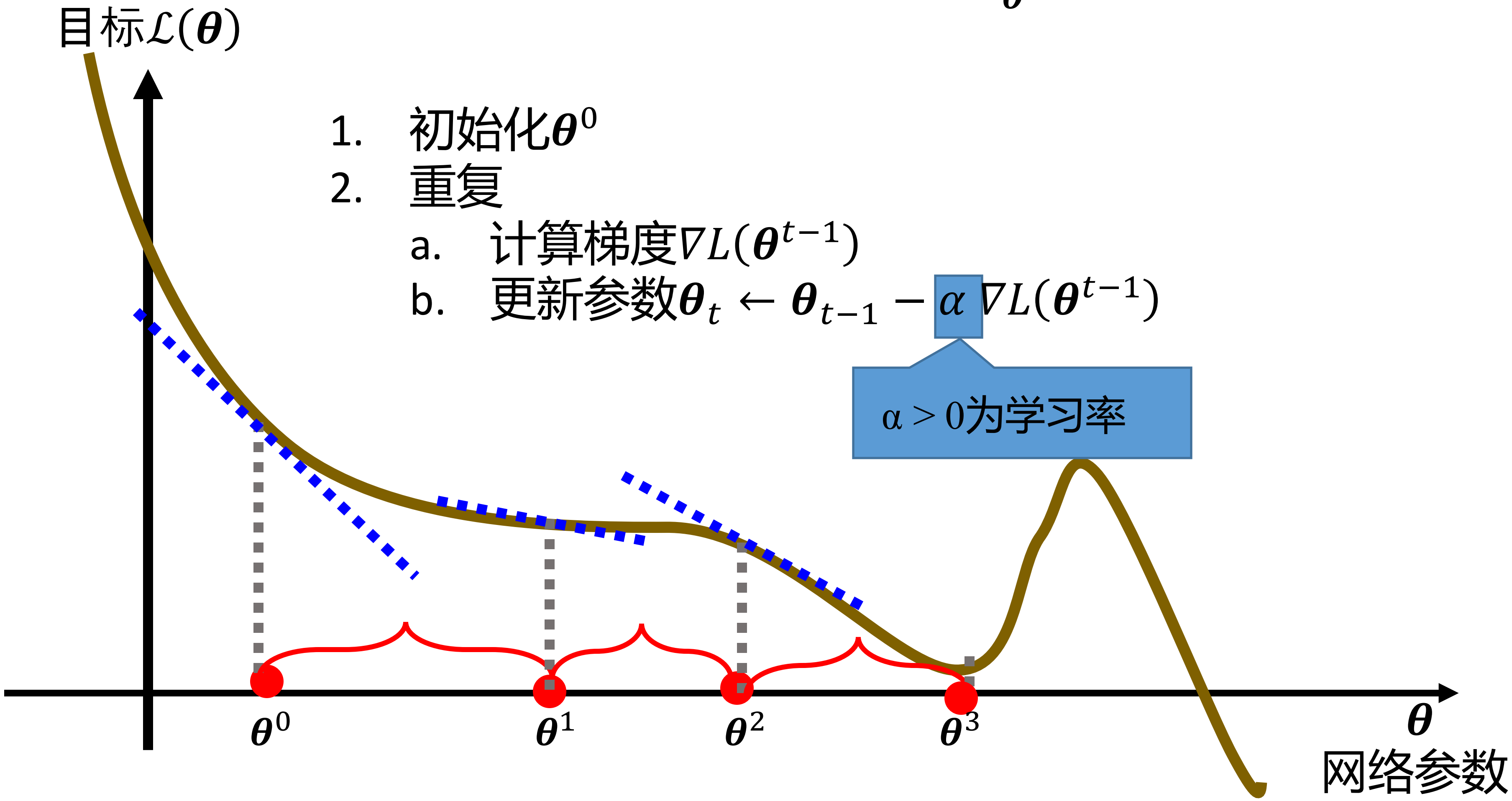
学习率

思考：为什么梯度下降能保证损失是非递增的？有没有什么前提条件？

梯度下降

优化算法解决 $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$

1. 初始化$\boldsymbol{\theta}^0$
2. 重复
   a. 计算梯度$\nabla L(\boldsymbol{\theta}^{t-1})$
   b. 更新参数$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boxed{\alpha}\, \nabla L(\boldsymbol{\theta}^{t-1})$



随机梯度下降 (SGD)

- 由于数据集可能很大，无法全部放入内存计算梯度

- 一般采用小批量随机梯度下降法，每次从数据集中采样一部分样本（称为batch），计算batch上的梯度，并进行参数更新。

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}_{n=1}^{N}$，每次采样B个样本

梯度下降

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

$$\frac{\partial \mathcal{L}_D(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{W}^{(l)}} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial \mathcal{L}\left(\boldsymbol{y}^{(n)}, \hat{\boldsymbol{y}}^{(n)}\right)}{\partial \boldsymbol{W}^{(l)}}$$

小批量随机梯度下降

$$\mathcal{L}_B(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right) + \frac{1}{2}\lambda\|\boldsymbol{W}\|_F^2$$

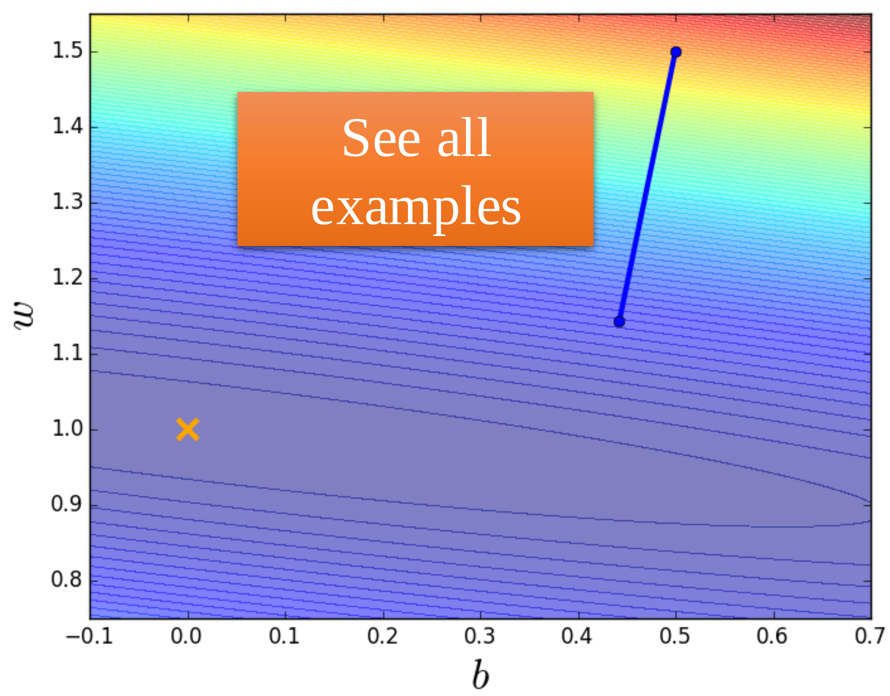
$$\frac{\partial \mathcal{L}_B(\boldsymbol{W}, \boldsymbol{b})}{\partial \boldsymbol{W}^{(l)}} = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial \mathcal{L}\left(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}\right)}{\partial \boldsymbol{W}^{(l)}}$$

思考：数据集规模、模型大小以及GPU内存大小之间的关系

随机梯度下降 vs. 梯度下降

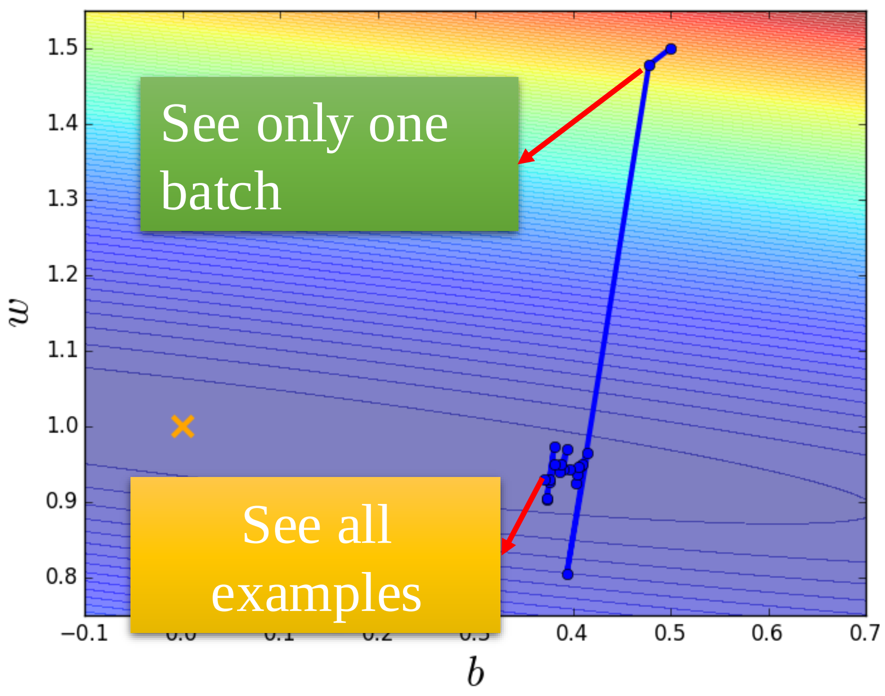
梯度下降

用所有样本对参数更新



随机梯度下降

每个批都对参数更新
更新次数更多



神经网络随机梯度下降的终止条件

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

目录

如何计算梯度?

□神经网络为一个复杂的复合函数

$$y = f^4\left(f^3\left(f^2\left(f^1\left(\boldsymbol{x}, \boldsymbol{\theta}^{(1)}\right), \boldsymbol{\theta}^{(2)}\right), \boldsymbol{\theta}^{(3)}\right), \boldsymbol{\theta}^{(4)}\right)$$

链式求导法则

$$\frac{\partial y}{\partial \boldsymbol{\theta}^{(1)}} = \frac{\partial f^1}{\partial \boldsymbol{\theta}^{(1)}} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \boldsymbol{\theta}^{(2)}} = \frac{\partial f^2}{\partial \boldsymbol{\theta}^{(2)}} \frac{\partial f^3}{\partial f^2} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \boldsymbol{\theta}^{(3)}} = \frac{\partial f^3}{\partial \boldsymbol{\theta}^{(3)}} \frac{\partial y}{\partial f^3}$$

$$\frac{\partial y}{\partial \boldsymbol{\theta}^{(4)}} = \frac{\partial f^4}{\partial \boldsymbol{\theta}^{(4)}}$$

$\frac{\partial f^l}{\partial f^{l-1}}$ 在计算 y 对于每一层的参数的导数时重复出现，可以存储复用。

□反向传播算法

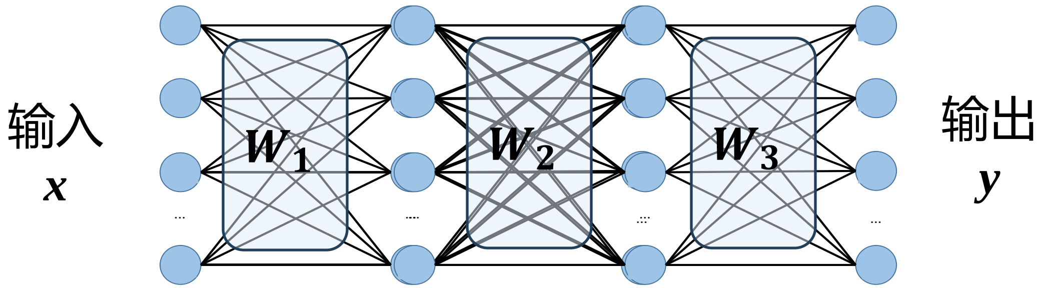
根据前馈网络的特点而设计的高效方法:由最深层开始向浅层依次根据链式法则计算导数值。

□一个更加通用的计算方法

自动微分（Automatic Differentiation，AD）

如何计算梯度?

口神经网络为一个复杂的复合函数



$$y = \sigma\left(\boldsymbol{W}_3\sigma(\boldsymbol{W}_2\sigma(\boldsymbol{W}_1\boldsymbol{x}))\right)$$

(σ为激活函数，不考虑偏置项 b)

我们定义每一层 “激活”：

$$\boldsymbol{z}_1 = \boldsymbol{W}_1\boldsymbol{x};\ \boldsymbol{a}_1 = \sigma(\boldsymbol{z}_1)$$

$$\boldsymbol{z}_2 = \boldsymbol{W}_2\boldsymbol{a}_1;\ \boldsymbol{a}_2 = \sigma(\boldsymbol{z}_2)$$

$$\boldsymbol{z}_3 = \boldsymbol{W}_3\boldsymbol{a}_2;\ \boldsymbol{a}_3 = \sigma(\boldsymbol{z}_3) = y$$

计算梯度 $\frac{\partial y}{\partial \boldsymbol{W}_i}$

链式求导法则

$$\frac{\partial y}{\partial \boldsymbol{W}_3} = \frac{\partial y}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{W}_3}$$

$$\frac{\partial y}{\partial \boldsymbol{W}_2} = \frac{\partial y}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{a}_2}\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{W}_2}$$

$$\frac{\partial y}{\partial \boldsymbol{W}_1} = \frac{\partial y}{\partial \boldsymbol{z}_3}\frac{\partial \boldsymbol{z}_3}{\partial \boldsymbol{a}_2}\frac{\partial \boldsymbol{a}_2}{\partial \boldsymbol{z}_2}\frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{a}_1}\frac{\partial \boldsymbol{a}_1}{\partial \boldsymbol{z}_1}\frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{W}_1}$$

$\frac{\partial y}{\partial z_3}$，$\frac{\partial a_2}{\partial z_2}$ 等在计算 y 对于第二层和第一层的参数的导数时重复出现，可以存储复用。

观察1: 计算梯度过程中需要计算神经网络各层激活值。（前向传播）

观察2: 计算梯度过程中的部分中间结果（如$\frac{\partial a_3}{\partial z_3}, \frac{\partial a_2}{\partial z_2}$）可以复用。（后向传播）

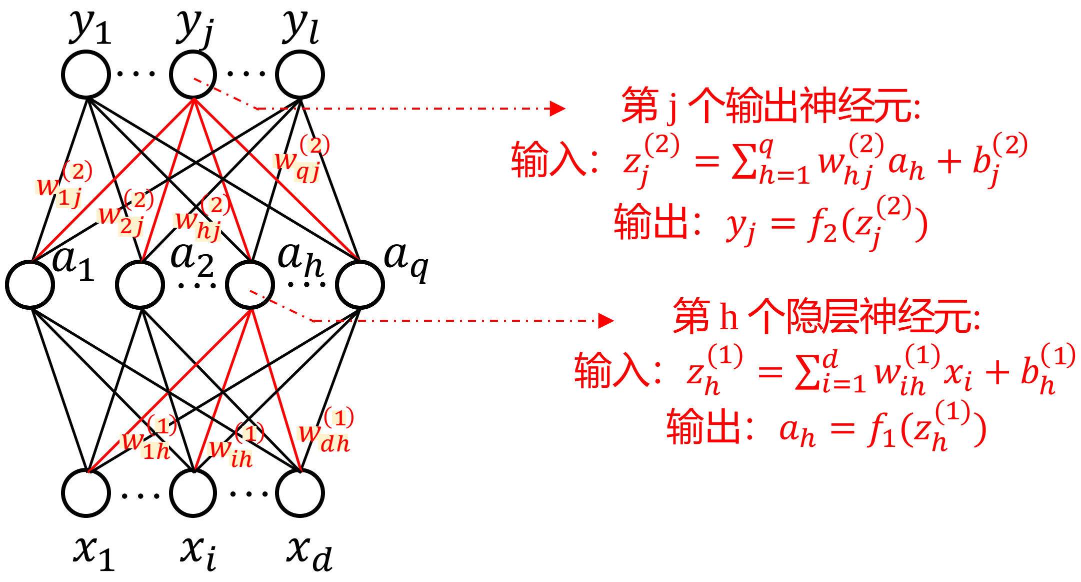
反向传播算法

$f_2(\cdot)$： 输出层激活函数
$w_{hj}^{(2)}$： 隐层与输出层神经元之间的连接权重
$b_j^{(2)}$： 输出层神经元的偏置

$f_2(\cdot)$： 隐层激活函数
$w_{ih}^{(1)}$： 输入层与隐层神经元之间的连接权重
$b_h^{(1)}$： 输出层神经元的偏置

网络中需要 $(d+l+1)q+l$ 个参数需要优化

输出l维实值向量$\boldsymbol{y}$

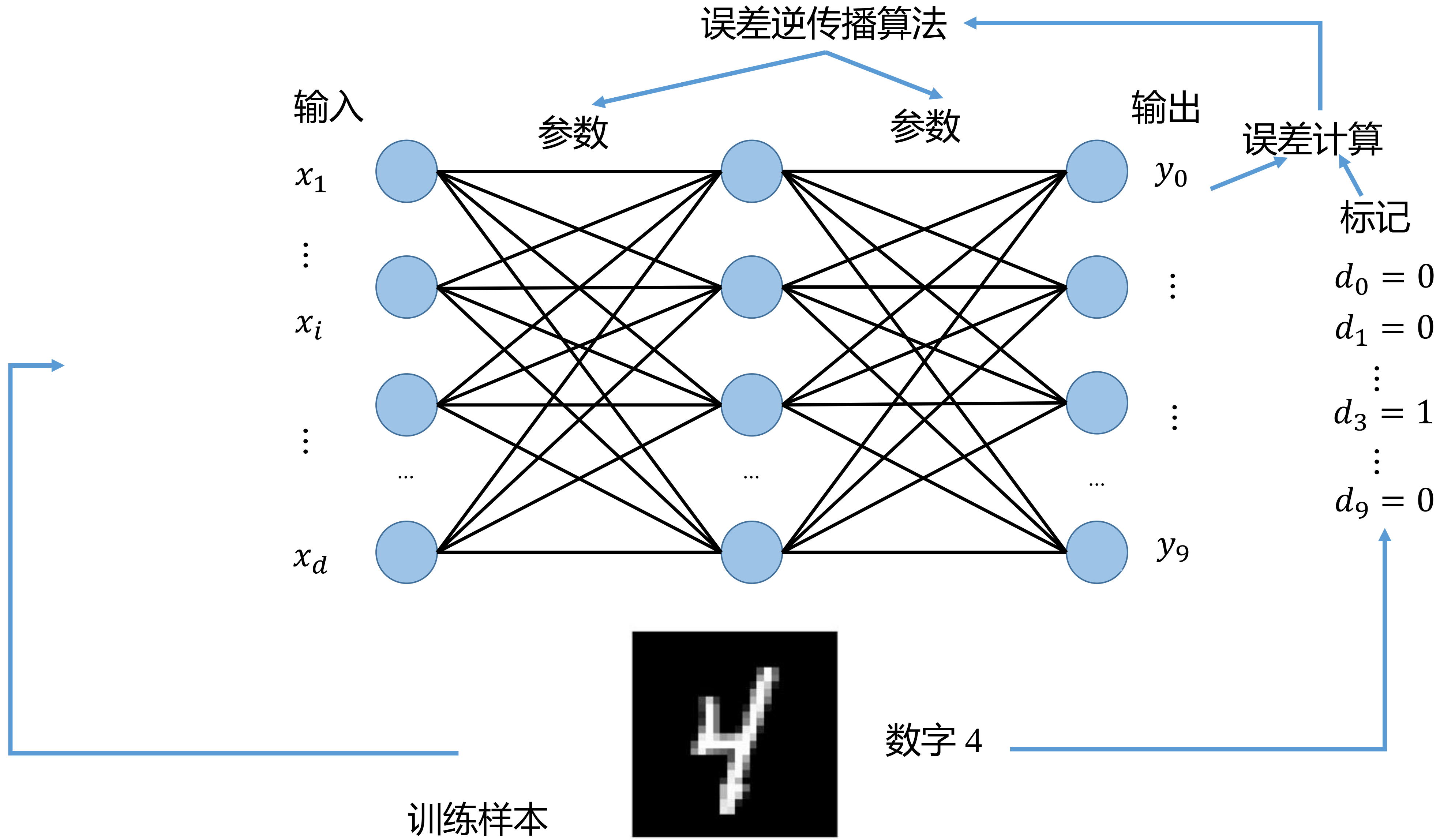


输入示例$\boldsymbol{x}$由d个属性描述

BP是一个迭代学习算法, 在迭代的每一轮中采用广义的感知机学习规则对参数进行更新估计, 任意的参数v的更新估计式为

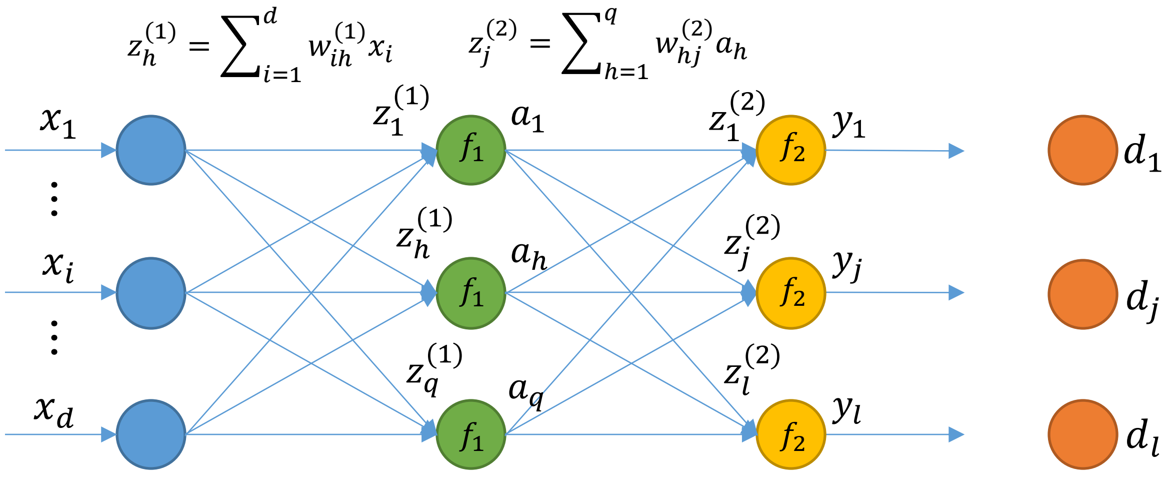
$$v \leftarrow v + \Delta v$$

反向传播算法



误差逆传播算法
输入
参数
参数
输出
误差计算
x_1
x_i
x_d
y_0
y_9
标记
$d_0 = 0$
$d_1 = 0$
$d_3 = 1$
$d_9 = 0$
数字 4
训练样本

反向传播算法—前向



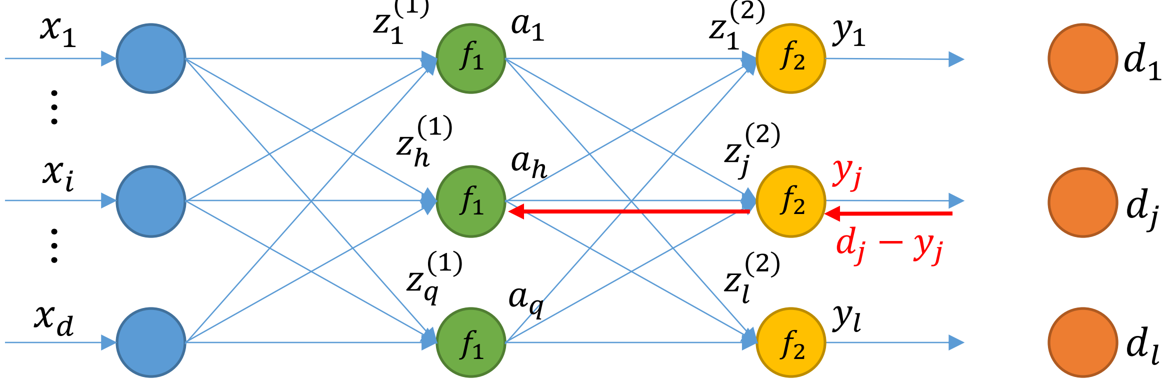
前向预测

$$\boldsymbol{x} \xrightarrow[f_1\left(z_h^{(1)}\right)]{z_h^{(1)} = \sum_{i=1}^{d} w_{ih}^{(1)} x_i} a_h \xrightarrow[f_1\left(z_j^{(2)}\right)]{z_j^{(2)} = \sum_{h=1}^{q} w_{hj}^{(2)} a_h} y_j$$

反向传播算法—后向

$$z_h^{(1)} = \sum_{i=1}^{d} w_{ih}^{(1)} x_i \qquad z_j^{(2)} = \sum_{h=1}^{q} w_{hj}^{(2)} a_h \qquad -\frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial y_j}$$

$$\frac{\partial z_j^{(2)}}{\partial w_{hj}^{(2)}} = a_h$$



$$e_j^{(2)} = (d_j - y_j) f_2'\left(z_j^{(2)}\right)$$

$$e_j^{(2)} = \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial y_j}\frac{\partial y_j}{\partial z_j^{(2)}} = \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial z_j^{(2)}}$$

后向传播

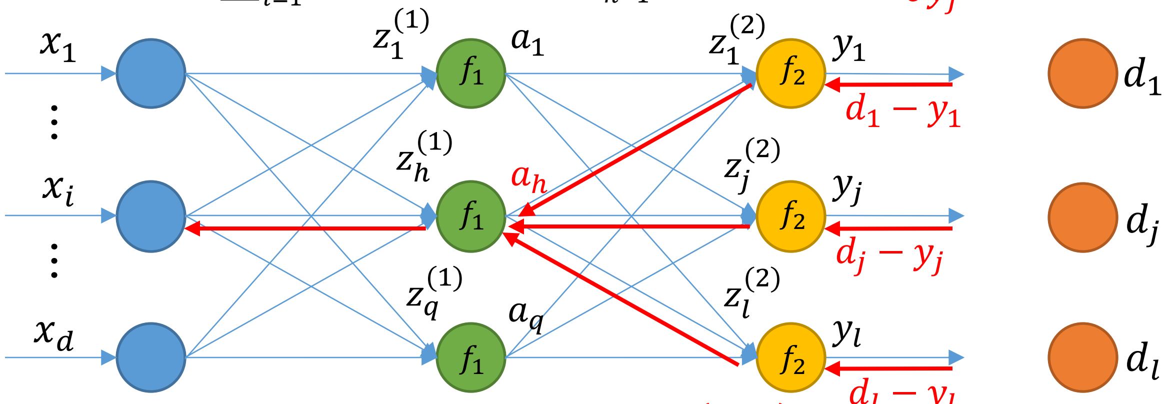
$$\Delta w_{hj}^{(2)} = \eta e_j^{(2)} a_h = \eta \mathrm{Error}_j \mathrm{Output}_h$$

$$w_{hj}^{(2)} = w_{hj}^{(2)} + \Delta w_{hj}^{(2)} \longleftarrow \mathcal{L}(\boldsymbol{d},\boldsymbol{y}) = \frac{1}{2}\sum_{j=1}^{l}(y_j - d_j)^2$$

$$\Delta w_{hj}^{(2)} = -\eta \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial w_{hj}^{(2)}} = -\eta \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial y_j}\frac{\partial y_j}{\partial z_j^{(2)}}\frac{\partial z_j^{(2)}}{\partial w_{hj}^{(2)}} = \eta (d_j - y_j) f_2'\left(z_j^{(2)}\right) a_h = \eta e_j^{(2)} a_h$$

反向传播算法—后向

$$z_h^{(1)} = \sum_{i=1}^{d} w_{ih}^{(1)} x_i \qquad z_j^{(2)} = \sum_{h=1}^{q} w_{hj}^{(2)} a_h \qquad -\frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial y_j}$$



$$e_j^{(2)} = \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial z_j^{(2)}}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial a_h} = \sum_j \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial a_h} = \sum_j e_j^{(2)} \frac{\partial z_j^{(2)}}{\partial a_h} = \sum_j e_j^{(2)} w_{hj}^{(2)}$$

后向传播

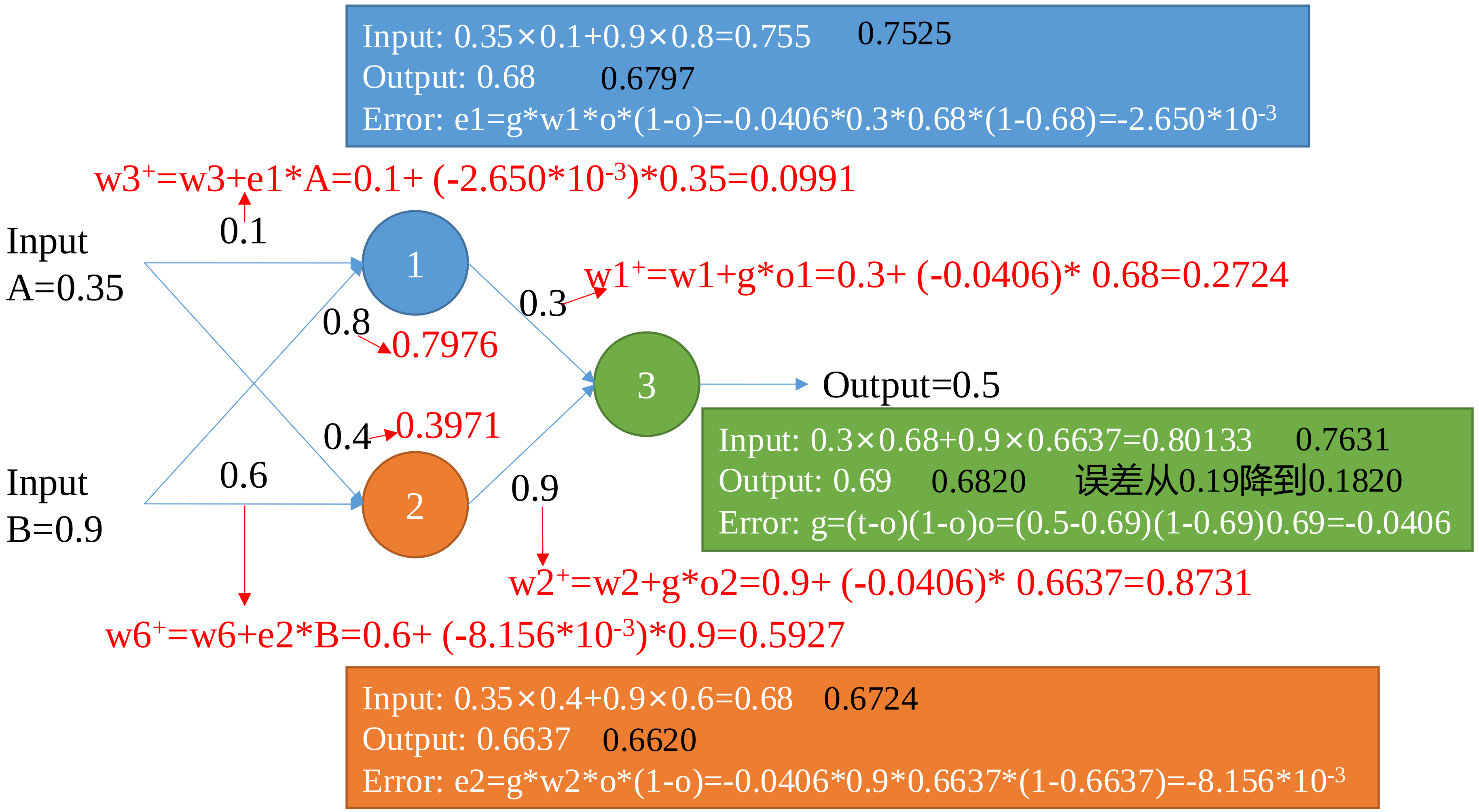
$$w_{ih}^{(1)} = w_{ih}^{(1)} + \Delta w_{ih}^{(1)} \quad \Delta w_{ih}^{(1)} = \eta e_h^{(1)} x_i = \eta \mathrm{Error}_h \mathrm{Output}_i \qquad \mathcal{L}(\boldsymbol{d},\boldsymbol{y}) = \frac{1}{2}\sum_{j=1}^{l} (y_j - d_j)^2$$

$$\Delta w_{ih}^{(1)} = -\eta \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial w_{ih}^{(1)}} = -\eta \frac{\partial \mathcal{L}(\boldsymbol{d},\boldsymbol{y})}{\partial a_h} \frac{\partial a_h}{\partial z_h^{(1)}} \frac{\partial z_h^{(1)}}{\partial w_{ih}^{(1)}} = \eta \sum_j e_j^{(2)} w_{hj}^{(2)} f_1'(z_h^{(1)}) x_i = \eta e_h^{(1)} x_i$$

$$e_h^{(1)} = f_1'(z_h^{(1)}) \sum_j e_j^{(2)} w_{hj}^{(2)}$$

反向传播算法：简单例子

□考虑如下简单网络　假设激活函数为Sigmoid函数



后向传播算法 — 一般情形

前向计算激活过程

输入	第 1 层	第 2 层	第 L-1 层	第 L 层
$\boldsymbol{x}=\boldsymbol{a}^{(0)}$	$\to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)}$	$\to \boldsymbol{z}^{(2)} \to \boldsymbol{a}^{(2)}$	$\to \cdots \to \boldsymbol{a}^{(L-1)}$	$\to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)}$

仿射变换 $\boldsymbol{z}^{(l)}=\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}$

$\boldsymbol{a}^{(l)}=f_l\left(\boldsymbol{z}^{(l)}\right)$ 非线性变换

后向计算梯度过程

链式法则 $$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}}=\left\langle \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \right\rangle$$

$\boldsymbol{W}^{(l)}$的第k行

$$z_k^{(l)}=\langle \boldsymbol{w}_k^{(l)}, \boldsymbol{a}^{(l-1)} \rangle + b_k^{(l)}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}} \triangleq \boldsymbol{e}^{(l)}$$ **误差项**

$$\frac{\partial z_k^{(l)}}{\partial w_{ij}^{(l)}}=\delta(k=i)a_j^{(l-1)}$$

链式法则 $$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}}=\left\langle \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \right\rangle$$

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}}=\mathbf{1}_{m^{(l)}}$$ 单位阵

后向传播算法 — 一般情形

前向计算激活过程

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \boldsymbol{a}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)}$$

后向计算梯度过程

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{W}^{(l+1)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l+1)} \qquad \boldsymbol{a}^{(l)} = f_l\left(\boldsymbol{z}^{(l)}\right) \Rightarrow a_k = f_l\left(z_k^{(l)}\right)$$

误差项分解 $\boldsymbol{e}^{(l)} = \dfrac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}$

$$\boxed{\frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} = \boldsymbol{W}^{(l+1)}}$$

链式求导法则 $= \boxed{\dfrac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l+1)}}} \boxed{\dfrac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}}} \boxed{\dfrac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}}$

$$\boxed{\begin{aligned}\frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} &= \frac{\partial f_l\left(\boldsymbol{z}^{(l)}\right)}{\partial \boldsymbol{z}^{(l)}} \\ &= \operatorname{diag}\left(\nabla f_l\left(\boldsymbol{z}^{(l)}\right)\right)\end{aligned}}$$

$$= \boxed{\boldsymbol{e}^{(l+1)}} \boxed{\boldsymbol{W}^{(l+1)}} \boxed{\operatorname{diag}\left(\nabla f_l\left(\boldsymbol{z}^{(l)}\right)\right)}$$

$$= \nabla f_l\left(\boldsymbol{z}^{(l)}\right) \odot \left(\boldsymbol{e}^{(l+1)} \boldsymbol{W}^{(l+1)}\right)$$

观察：$\boldsymbol{e}^{(l+1)}$可以用来计算$\boldsymbol{e}^{(l)}$

后向传播算法 — 一般情形

回过头来

$$\frac{\partial z_k^{(l)}}{\partial w_{ij}^{(l)}} = \delta(k=i)a_j^{(l-1)}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \left\langle \boldsymbol{e}^{(l)}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \right\rangle = e_i^{(l)} a_j^{(l-1)} \quad \Rightarrow \quad \frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{e}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\top}$$

后向传播算法计算参数梯度的核心公式组

$$\boldsymbol{e}^{(l)} = \nabla f_l\left(\boldsymbol{z}^{(l)}\right) \odot \left(\boldsymbol{e}^{(l+1)} \boldsymbol{W}^{(l+1)}\right)$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{e}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\top}$$

观察：$e^{(l+1)}$可以用来计算$e^{(l)}$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \left\langle \boldsymbol{e}^{(l)}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \right\rangle = \boldsymbol{e}^{(l)}$$

反向传播算法

□前馈神经网络的训练过程可以分为以下步骤

数据准备：从训练集中随机采样 k 个样本，作为一个训练batch

前向计算：从第一层开始计算每一层的状态和激活值，直到最后一层

反向计算：从最后一层开始计算每一层的参数的偏导数

更新参数：使用合适的学习率更新每一层的参数

目录

深度学习的三个步骤

1 定义网络

2 损失函数

3 优化

深度学习就是这么简单

Keras快速入门

https://keras.io/examples/

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy',  optimizer='sgd', metrics=['accuracy'])

model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

loss = model.evaluate(X_test, Y_test, batch_size=32)
```

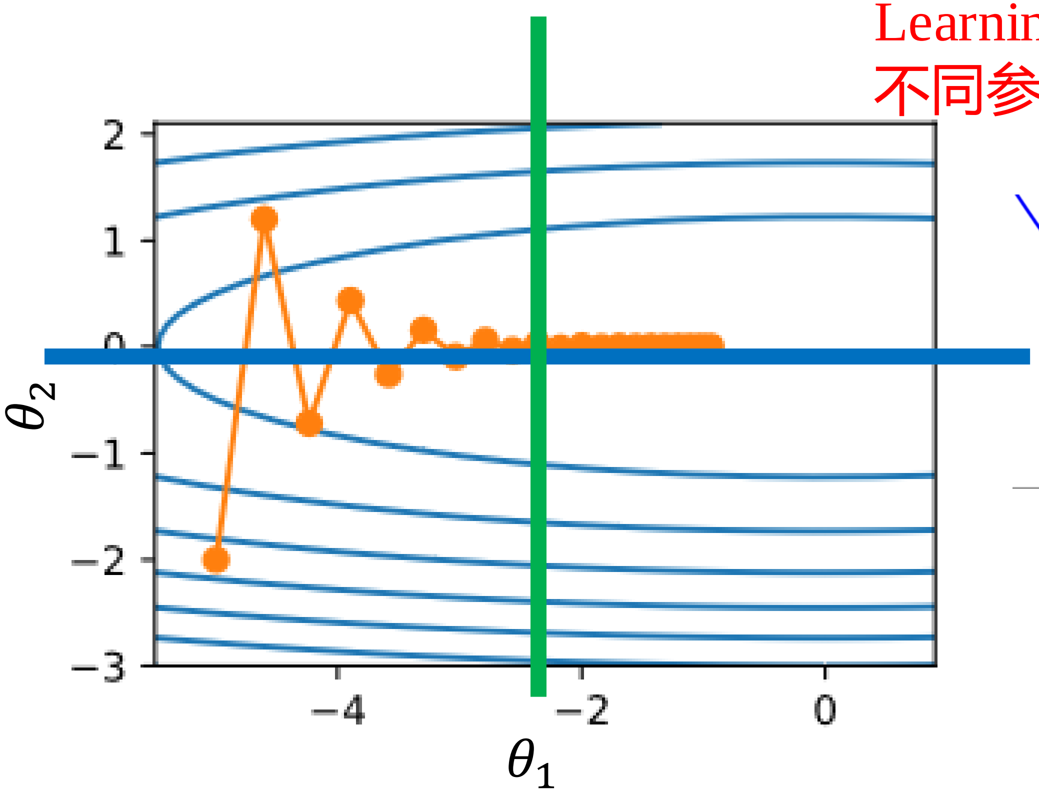
Pytorch快速入门

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

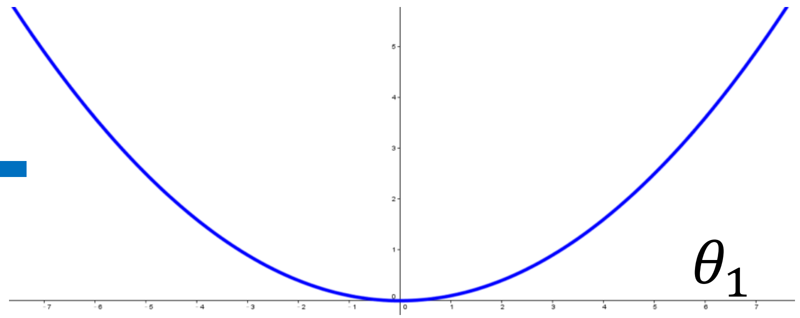
```python
model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

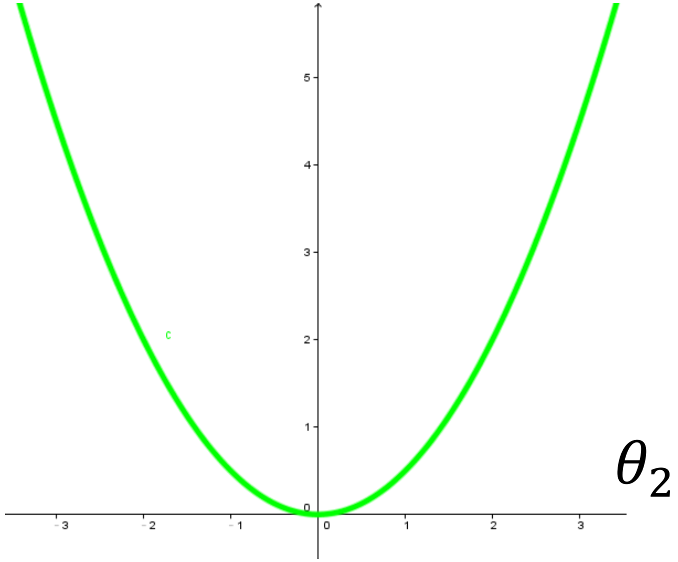
目录

随机梯度下降的问题



Learning rate cannot be one-size-fits-all
不同参数不同学习率

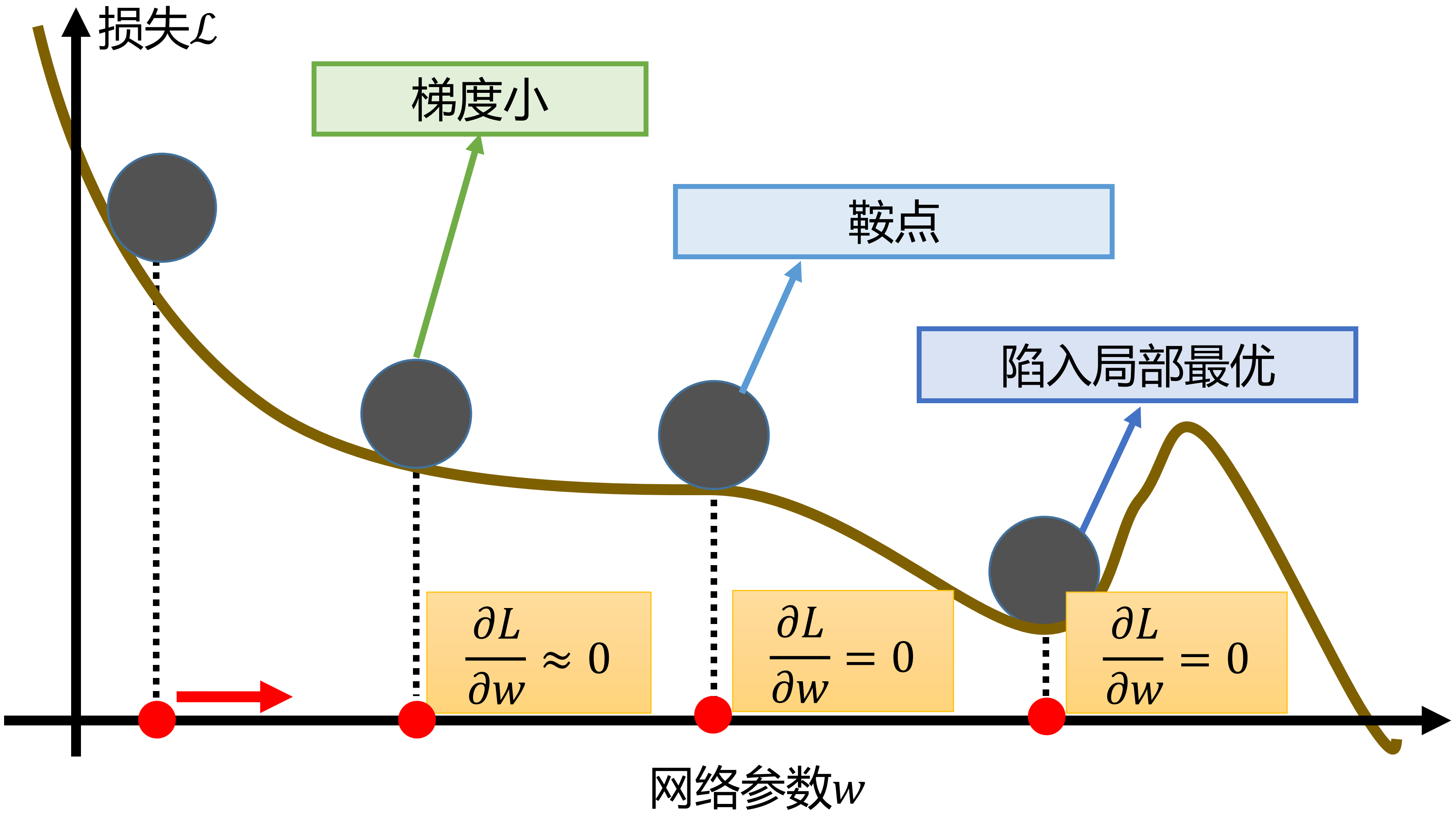




- 同一位置，损失在θ_2 轴方向比在 θ_1 轴方向的斜率的绝对值更大

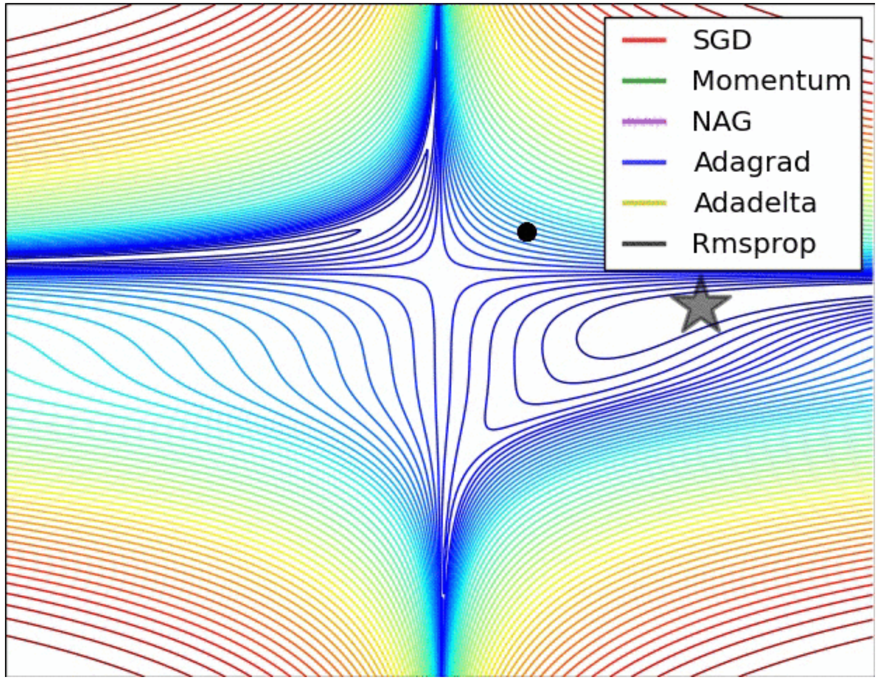
- 给定学习率，梯度下降迭代参数时在竖直方向会比在水平方向移动幅度更大

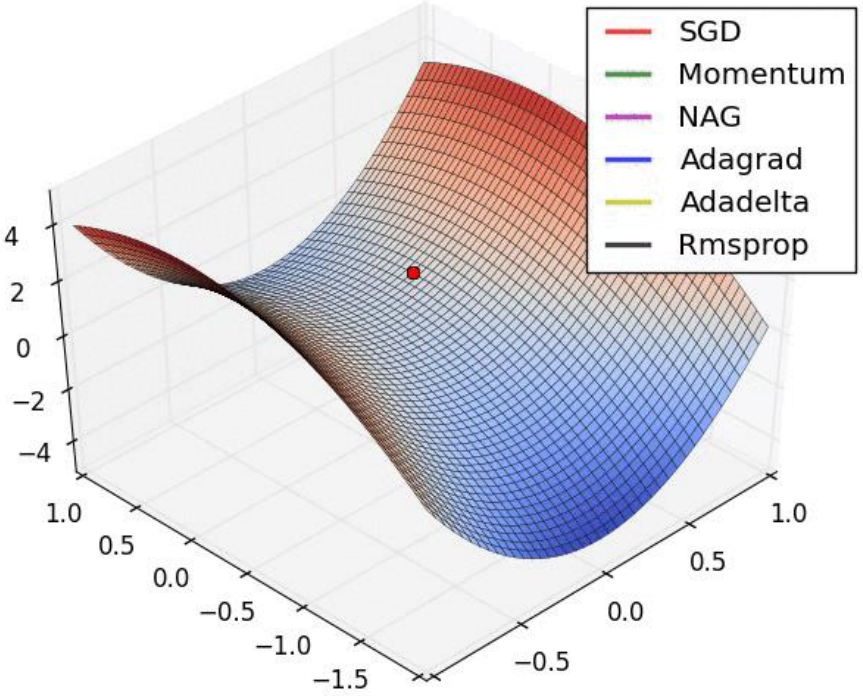
随机梯度下降的问题



损失$\mathcal{L}$
梯度小
鞍点
陷入局部最优
$\frac{\partial L}{\partial w} \approx 0$
$\frac{\partial L}{\partial w} = 0$
$\frac{\partial L}{\partial w} = 0$
网络参数w

优化算法的对比


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

目录

样本随机性的影响

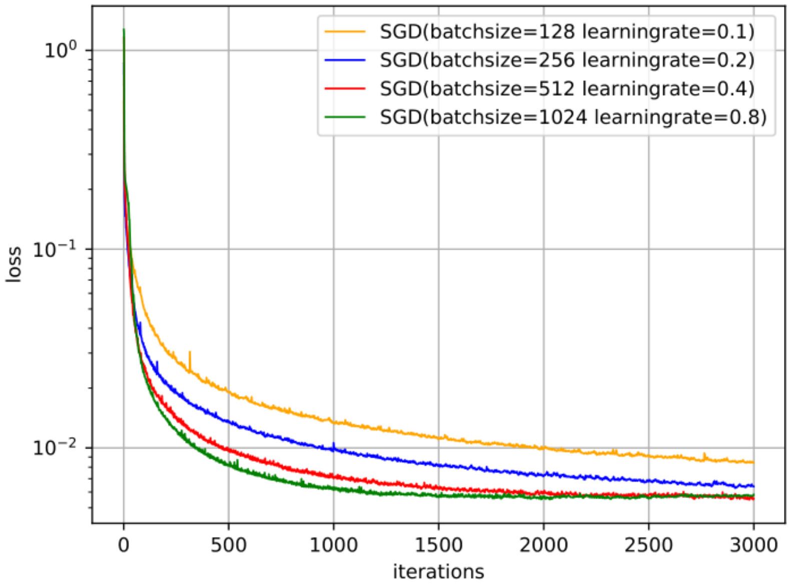
- 在每次迭代时，随机选择B个样本，这里的随机性非常重要
- 但随机性在大规模数据情况下很难满足
- 实践中通常将样本顺序打乱一次，然后按照这个顺序存储起来
- 虽然偏离真实随机采样，但不会有严重的有害影响

```
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

批量大小的影响

口批量大小不影响梯度期望，但会影响梯度方差，一般为2的幂数

- 批量越大，随机梯度的方差越小，引入的噪声也越小，训练也越稳定，因此可以设置较大的学习率
- 批量较小时，需要设置较小的学习率，否则模型会不收敛

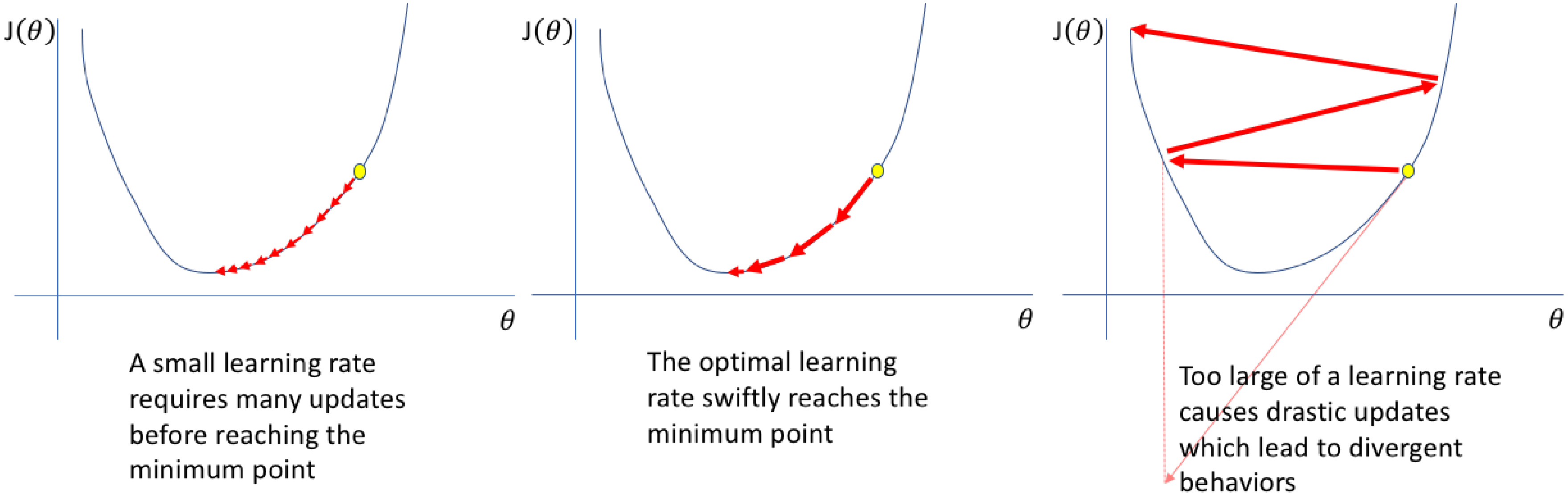


(a) 按 Iteration 的损失变化

小批量梯度下降中，每次选取样本数量对损失下降的影响

学习率的影响

Too low | Just right | Too high

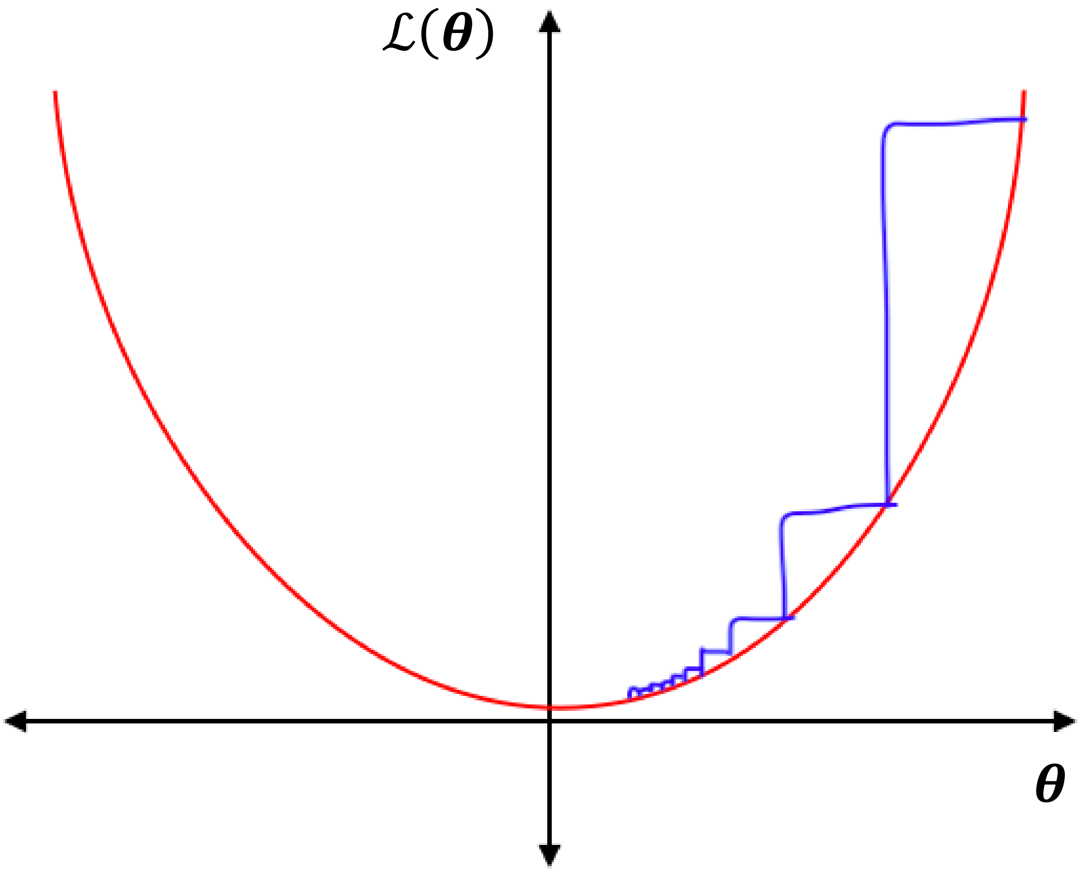


A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

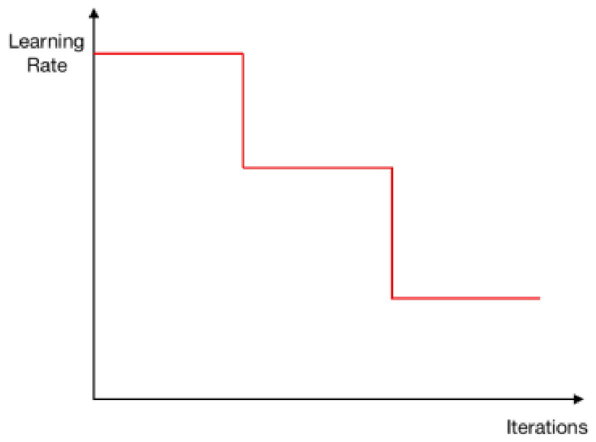
Too large of a learning rate causes drastic updates which lead to divergent behaviors

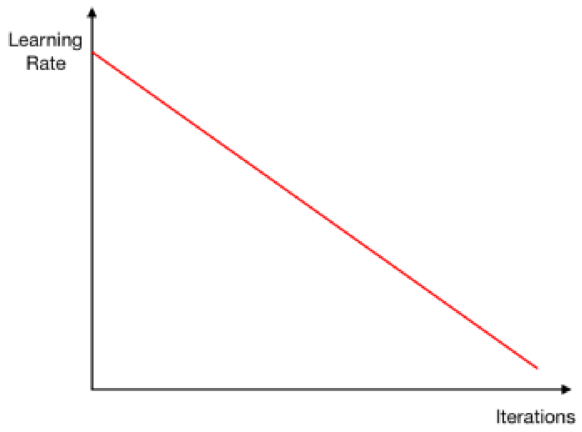
学习率衰减

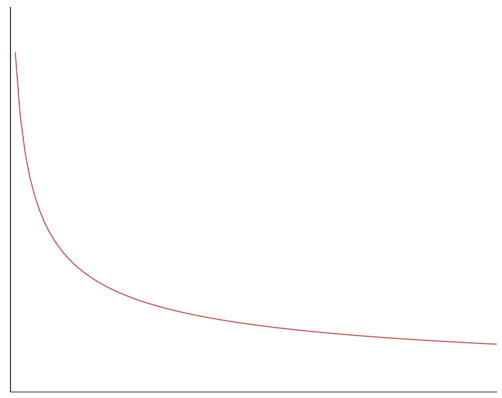


- 学习率一开始要保持大些保证收敛速度
- 在收敛到最优点附近时要小些以避免来回振荡

学习率衰减







梯级衰减（step decay） 线性衰减（Linear Decay） 1/t衰减（1/t decay）

正则化 (Regularization)

口新优化目标：不仅最小化损失，而且要让权重尽可能小

$$\mathcal{L}'(\boldsymbol{\theta}) = \underline{\mathcal{L}(\boldsymbol{\theta})} + \frac{1}{2}\lambda\underline{\|\boldsymbol{\theta}\|_2^2}$$

→ 正则化项目

损失，比如平方损失，
交叉熵损失等等

一般不考虑bias

$\boldsymbol{\theta} = \{w_1, w_2, \dots\}$

L2 正则:

$\|\boldsymbol{\theta}\|_2^2 = (w_1)^2 + (w_2)^2 + \cdots$

L1 正则:

$\|\boldsymbol{\theta}\|_1 = |w_1| + |w_2| + \cdots$

正则化 (Regularization)

L2 正则:

$$\|\boldsymbol{\theta}\|_2^2 = (w_1)^2 + (w_2)^2 + \cdots$$

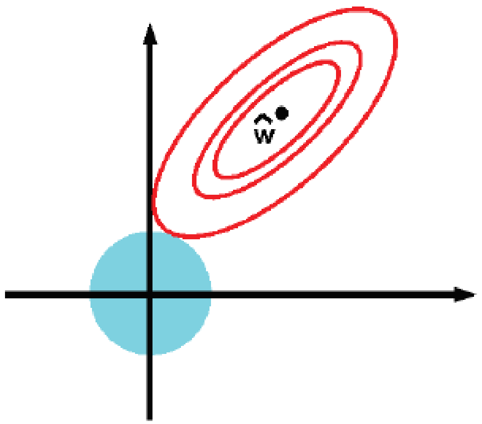
$$\mathcal{L}'(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{1}{2}\lambda\|\boldsymbol{\theta}\|_2^2$$

梯度: $\nabla\mathcal{L}' = \nabla\mathcal{L} + \lambda\,\boldsymbol{\theta}$

梯度下降更新:

$$\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} - \eta\nabla\mathcal{L}' = \boldsymbol{\theta}^{t-1} - \eta\nabla\mathcal{L} - \eta\lambda\,\boldsymbol{\theta}^{t-1}$$

$$\eta < 1, \lambda < 1 \qquad = \underline{(1-\eta\lambda)\boldsymbol{\theta}^{t-1}} - \eta\nabla\mathcal{L}$$

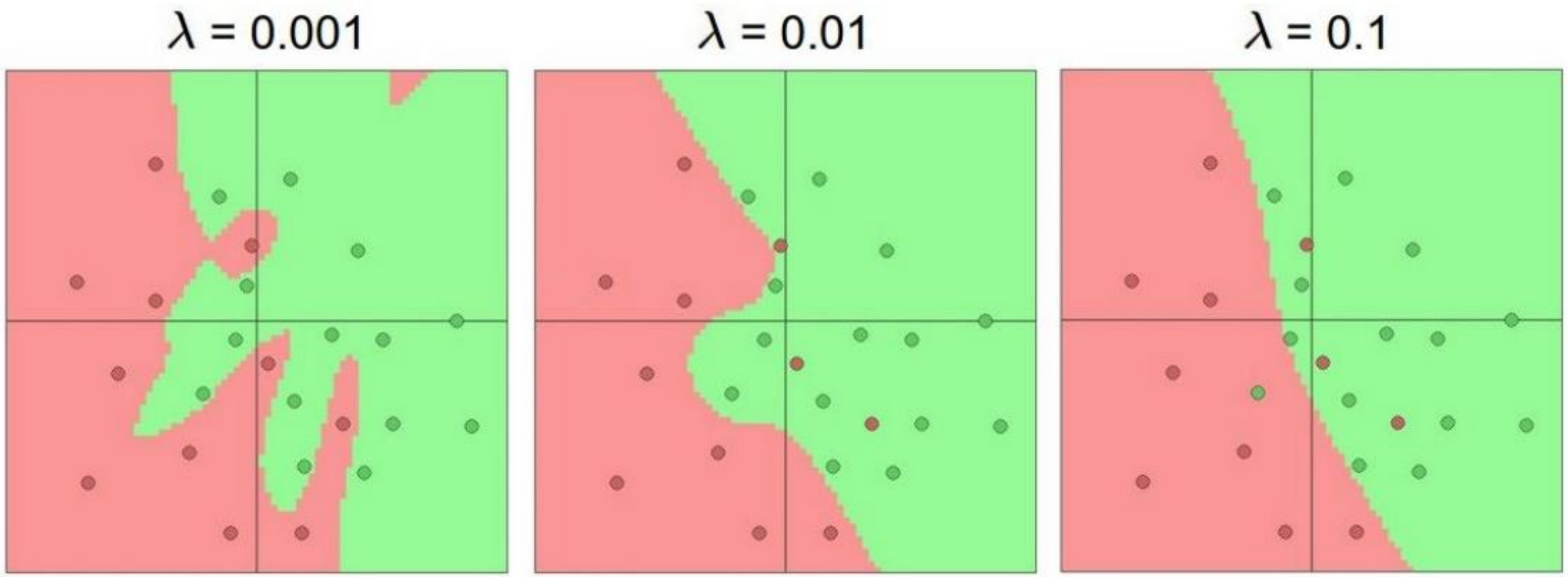


越来越小 ，但由于$-\eta\nabla L$ 项，使得参数不会变为0

L2正则也称权重衰减（weight decay）

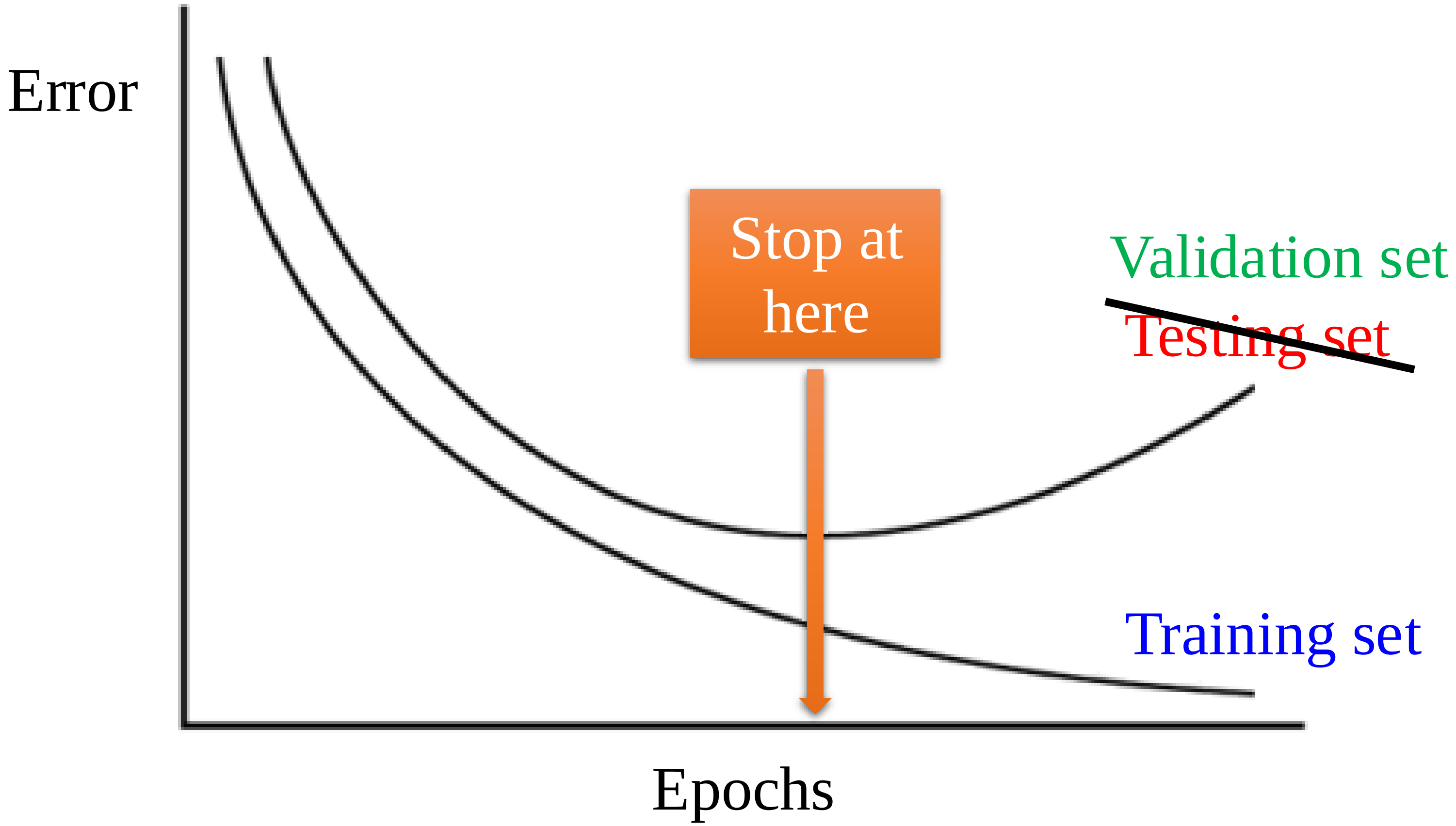
神经网络示例

- 不同正则项权重对于分类性能的影响



http://playground.tensorflow.org/

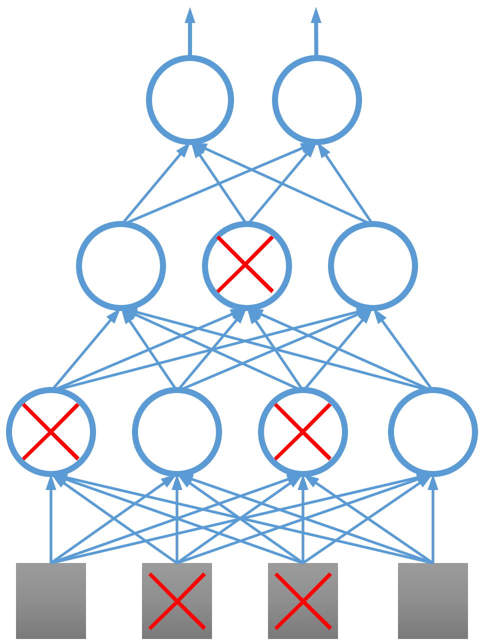
提前终止 (Early Stop)



- 验证集上准确率下降（损失上升）的时候停止训练
- 训练很长时间，保存在验证集上最优的模型

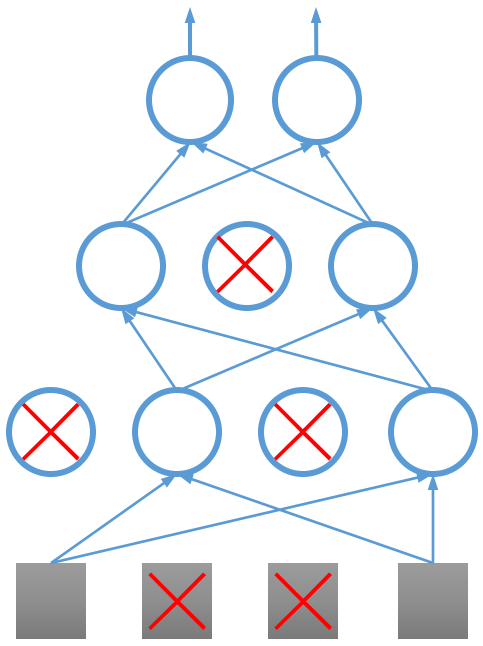
Dropout

- 在训练时，取得一个batch后，以概率p设置一些神经元为0



Dropout

□在训练时，取得一个batch后，以概率p设置一些神经元为0



- 网络结构发生了变化
- 用新网络在batch上计算梯度
- 在新网络上进行参数更新

每次取新batch时，都需要重新随机对神经元置为0

Dropout—训练时的实现

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

输入单元被包括概率为0.8，隐藏单元为0.5

Dropout

- **测试时，没有dropout，即所有的神经元都处于激活状态**

- **缩放激活函数的输出，使得每个神经元测试时输出等于训练时的期望输出**

权重比例推断规则

如果训练时dropout 的概率为 p，那么所有的权重要乘以 $1-p$

Dropout—训练和测试时实现

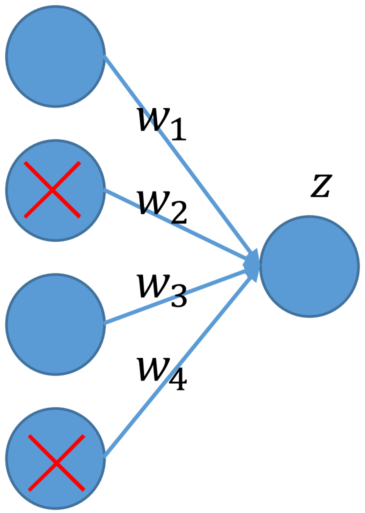
该概率为保留概率

```
p = 0.5 # ng a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Dropout—直观解释

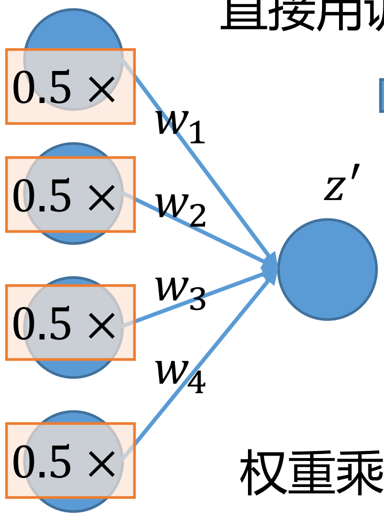
训练时dropout

dropout 概率为0.5



测试时dropout

No dropout



直接用训练后的权重

$\Rightarrow z' \approx 2z$

权重乘以$1-p$

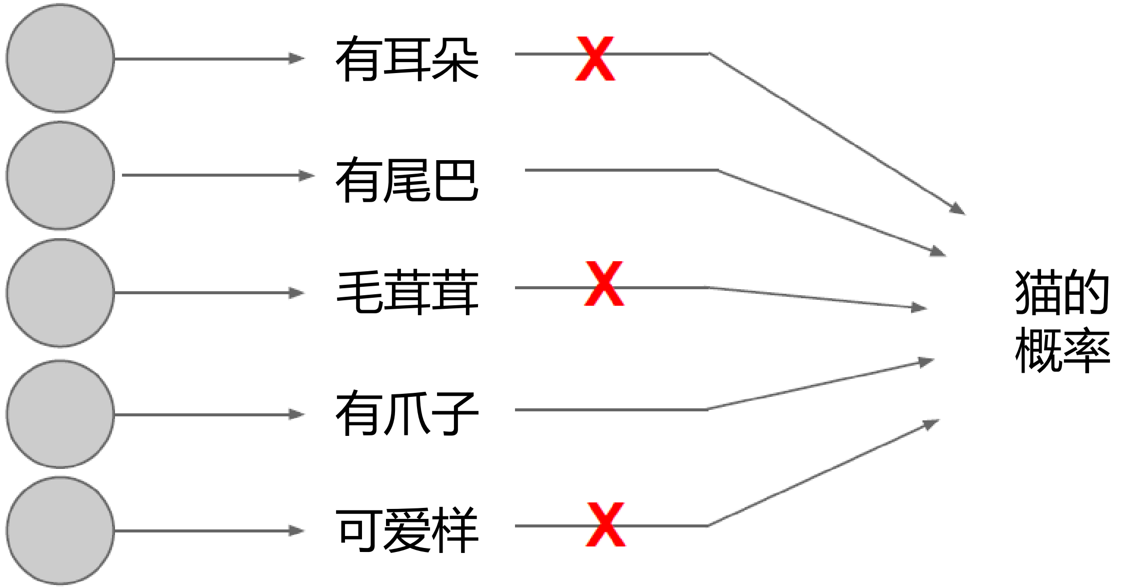
$\Rightarrow z' \approx z$

Dropout动机

1 强制网络有冗余特征表示

2 防止特征的co-adaptation

co-adaptation: feature detectors只有在一些其它特定的feature detectors存在时才能发挥作用的情况



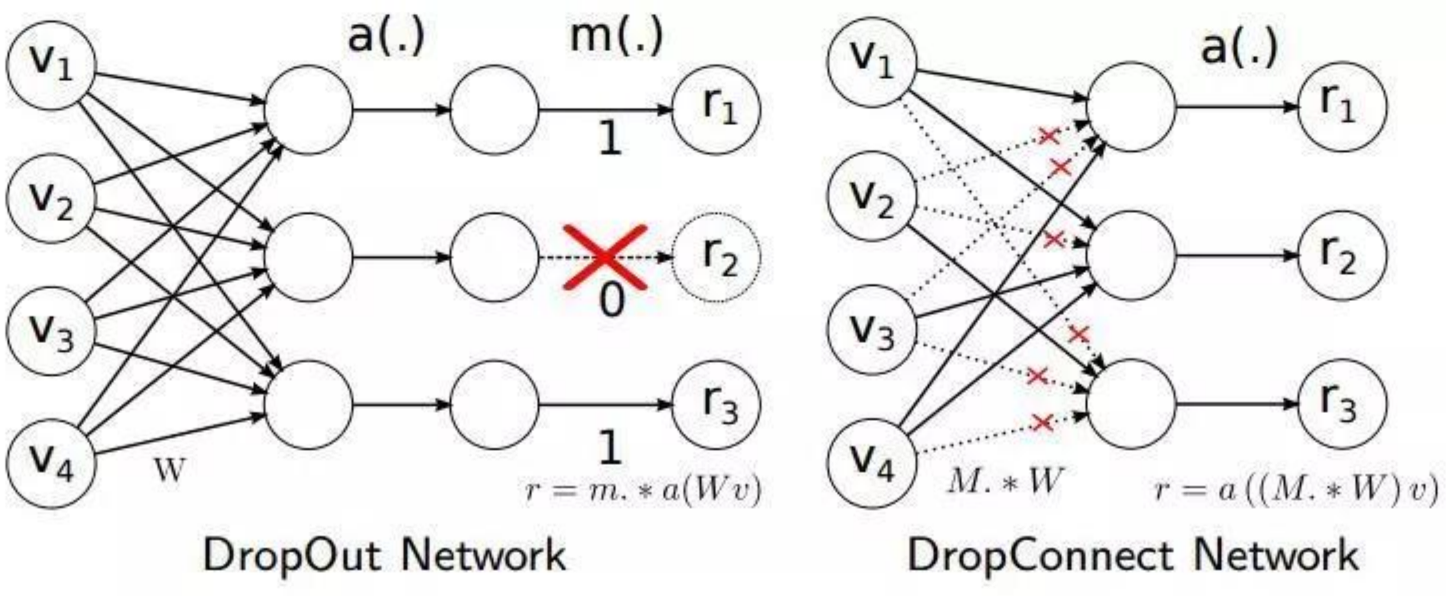
Dropout动机

□人脸识别情形下的动机：

1. 模型学得通过鼻检测脸的隐藏单元h_i，丢失h_i 对应于擦除图像中有鼻子的信息

2. 模型必须学习另一种h_i，要么是鼻子存在的冗余编码，要么是像嘴这样的脸部的另一特征

DropConnect



DropConnect：将节点中的每个与其相连的输入权值以1-p的概率变成0

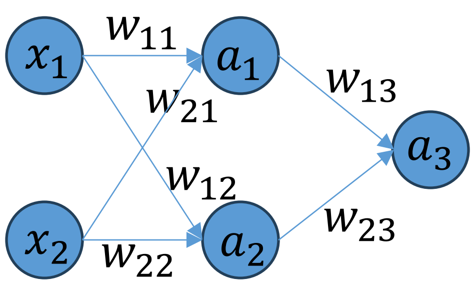
Dropout：随机的将隐层节点的输出变成0

目录

参数初始化

□参数不能全部初始化为0！为什么?

- 对称权重问题：前馈神经网络中同一层权重初始化为相同值。
- 前向传播中得到的每一层中的不同神经元激活值相同。
- 反向传播中得到的同一层的各个权重参数的梯度值相同。
- 使得隐层神经元没有区分性，神经网络无法有效学习数据中的特征。



$$a_1 = f(w_{11}x_1 + w_{21}x_2 + b_1) = f(0)$$
$$a_2 = f(w_{12}x_1 + w_{22}x_2 + b_2) = f(0)$$
$$a_3 = sigmoid(w_{13}a_1 + w_{23}a_2 + b_3) = sigmoid(0)$$

第一次更新：$a_1 = a_2 \Rightarrow \frac{\partial L}{\partial w_{13}} = \frac{\partial L}{\partial w_{23}} \Rightarrow w_{13} - \eta\frac{\partial L}{\partial w_{13}} = w_{23} - \eta\frac{\partial L}{\partial w_{23}}$

第一次更新：$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial w_{22}} = 0$

随后更新：由于对称性，保持 $a_1 = a_2 \Rightarrow$ 神经元输出没有区分性

□初始化方法

- 预训练初始化
- 随机初始化
- 固定值初始化 （例如：偏置 （ Bias ） 通常用 0 来初始化）

随机初始化：基于固定方差的初始化

□Gaussian分布初始化

- 参数从一个固定均值（比如0）和固定方差（比如0.01）的Gaussian分布进行随机初始化。

$$\boldsymbol{W} = \sigma * \mathrm{np.random.randn(fan\_in, fan\_out)}$$

□均匀分布初始化

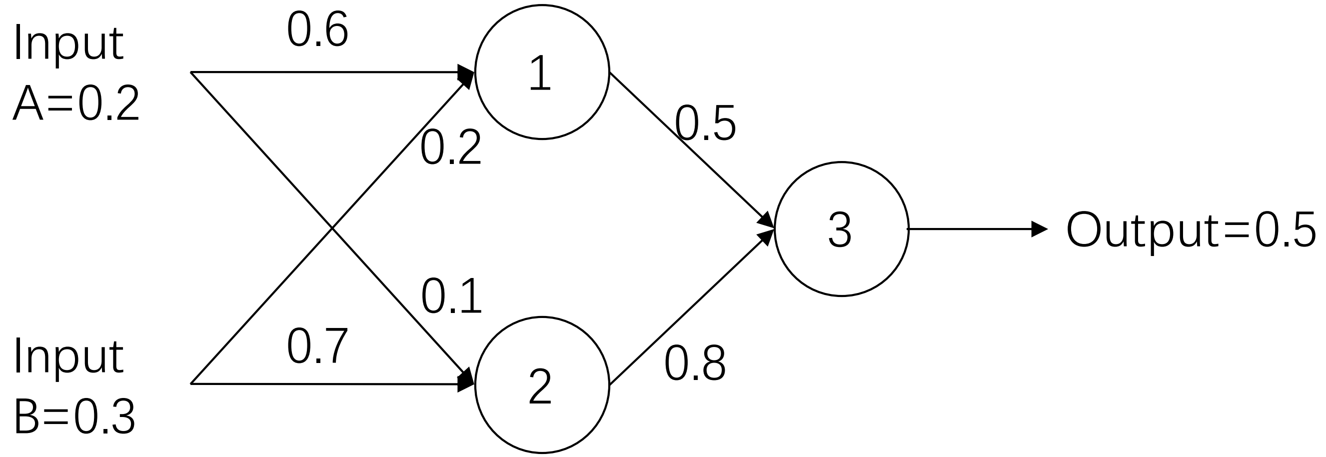
- 参数可以在区间$[-r, r]$内采用均匀分布进行初始化

若方差为σ^2，
那么$r = \sqrt{3\sigma^2}$

$$\boldsymbol{W} = \mathrm{r} * \mathrm{np.random.rand(fan\_in, fan\_out)}$$

作业习题

（1）在如下神经网络，假设激活函数为ReLU，用平方损失$\frac{1}{2}(y-\hat{y})^2$计算误差，请用BP算法更新一次所有参数（学习率为1），给出更新后的参数值，并计算给定输入值x=(0.2,0.3)时初始时和更新后的输出值，检查参数更新是否降低了平方损失值



（2）计算$\sigma(x)=\frac{1}{1+\exp(-x)}$的一阶和二阶导数、$\log\mathrm{softmax}(\boldsymbol{x})_{[i]}=\log\frac{\exp(x_i)}{\sum_{j=1}^{C}\exp(x_j)}$的梯度