《机器学习基础》

崔志勇

交通科学与工程学院

2024年5月

线性回归

什么是线性回归?

- **模型**
  - 线性方法：$f(\mathbf{x}, \theta) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$

    θ={w,b} 代表模型参数　　x 是输入变量

- **学习准则**
  - 损失函数：平方损失函数　　$\mathcal{L}(y, f(\boldsymbol{x}; \theta)) = \frac{1}{2}\big(y - f(\boldsymbol{x}; \theta)\big)^2$

    y 代表真实值，　f(x; θ) 代表预测值

- **优化算法**
  - 梯度下降

线性回归含义

模型

- 假设$\mathcal{F} = \{f \mid y = f(x)\}$中包含所有可能的函数，$x$、$y$是输入变量
- 如一元线性函数：

$$y = wx + b$$

w：权重　　b：偏差

- 给定一个数据集 $D = \{(x^{(i)}, y^{(i)})\}$ 含有N个样本

$$y = \sum_j w_j x_j + b.$$

线性回归含义

学习准则

- 损失函数: $L(\hat{y}, y)$，其中 $\hat{y} = \mathrm{f}(\mathrm{x}; \theta)$ 是预测值，y 是真实值
  - 如果我们使用平方差:

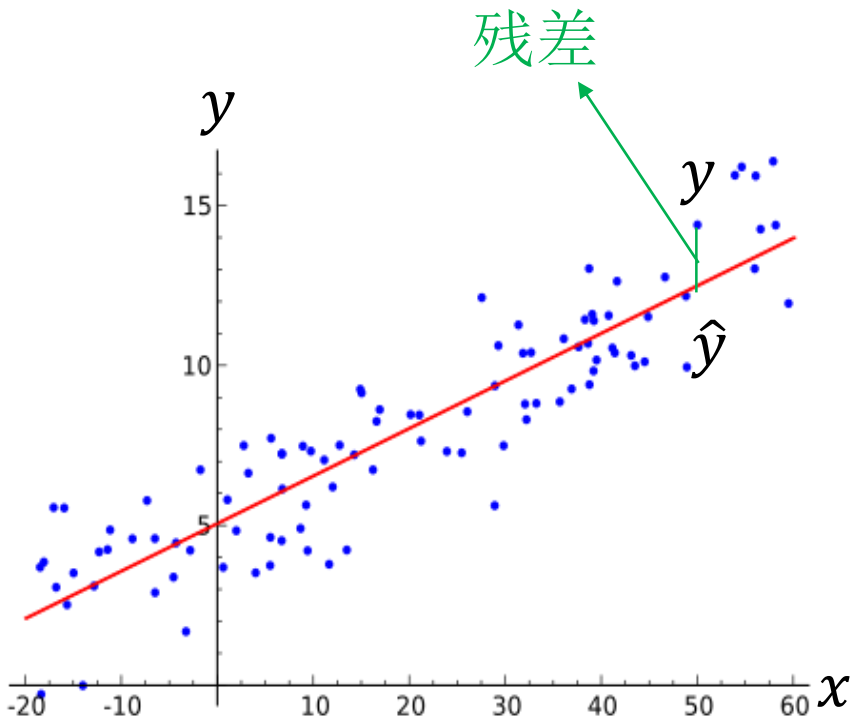
$$\mathcal{L}(\hat{y}, y) = \frac{1}{2} \sum_i (\hat{y}^{(i)} - y^{(i)})^2.$$

- 残差: $\hat{y} - y$

- 代价函数:

$$\mathcal{J}(w, b) = \frac{1}{2} \sum_i (y^{(i)} - wx^{(i)} - b)^2.$$

为什么求和时要在前面加一个 $\frac{1}{2}$?



线性回归含义

优化算法

计算代价函数的偏导数

方法一：线性回归直接求解 (最小二乘法)

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} \left(\sum_{j'=1}^{M} w_{j'} x_{j'}^{(i)} - y^{(i)} \right) = 0.$$

只要N ≥ M,便可求解该线性方程 （ N：数据样本数，M：模型参数量）

接下来计算权重：

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{N}\sum_{j'=1}^{M} \left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) w_{j'} - \frac{1}{N}\sum_{i=1}^{N} x_j^{(i)} y^{(i)} = 0.$$

线性回归含义

优化算法

计算代价函数的偏导数

方法一：线性回归直接求解 (最小二乘法)

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{N} \sum_{j'=1}^{M} \left(\sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \right) w_{j'} - \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} y^{(i)} = 0.$$

$$\sum_{j'=1}^{M} A_{jj'} w_{j'} - c_j = 0 \quad \forall j \in \{1, \ldots, M\},$$

$$A_{jj'} = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \qquad c_j = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} y^{(i)}$$

线性回归含义

优化算法

计算代价函数的偏导数

方法一：线性回归直接求解 (最小二乘法)

向量化表示：

- 输入表示为 $N \times M$ 矩阵 $\boldsymbol{X}$，即 $\boldsymbol{X} \in \mathbb{R}^{\wedge}(N \times M)$
  - $\boldsymbol{X}$的每一行对应一个训练样本
  - 每列对应一个输入维度
- 权重表示为 M 维向量 $\boldsymbol{w}$
- 标签表示为 N 维向量 $\boldsymbol{y}$

使用矩阵向量积计算预测： $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b\mathbf{1}$

向量形式的损失函数： $\mathcal{L} = \frac{1}{2N}\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \frac{1}{2N}\|\mathbf{X}\mathbf{w} + b\mathbf{1} - \mathbf{y}\|^2$

线性回归含义

优化算法

计算代价函数的偏导数

方法一：线性回归直接求解 (最小二乘法)

$$\sum_{j'=1}^{M} A_{jj'} w_{j'} - c_j = 0 \quad \forall j \in \{1, \ldots, M\},$$

$$A_{jj'} = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} x_{j'}^{(i)} \qquad c_j = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)} y^{(i)}$$

向量化表示： $\mathbf{A} = \frac{1}{N} \mathbf{X}^\top \mathbf{X}$ $\mathbf{c} = \frac{1}{N} \mathbf{X}^\top \mathbf{y}$

线性系统 $\boldsymbol{Aw} = \boldsymbol{c}$ 的解由下式给出 $\boldsymbol{w} = \boldsymbol{A}^{-1}\boldsymbol{c}$

最优权重可以直接写成向量化形式： $\mathbf{w} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$

梯度下降法求解线性回归

优化算法

计算代价函数的偏导数

方法二：用梯度下降法求解

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial w_M} \end{pmatrix}$$

这一公式给出了**最陡上升的方向**

更新规则：通过不断迭代,找到函数的最小值点,从而得到最优的模型参数，即梯度下降法。

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{w}} \qquad w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{E}}{\partial w_j}$$

其中α是学习率，通常取0.01,0.001

梯度下降法求解线性回归

优化算法

计算代价函数的偏导数

方法二：用梯度下降法求解

为什么选择梯度下降法进行优化?

1. 梯度下降法**适用于任何模型**，不只是线性回归。

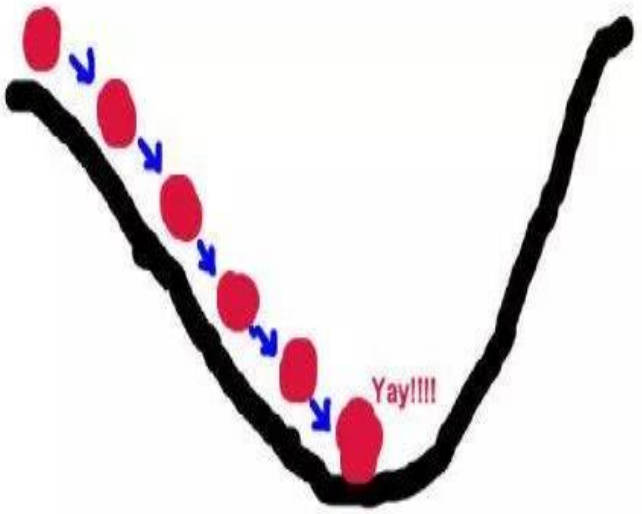
- 此外，它可以**自动完成优化过程**，因此我们可以使用PyTorch和TensorFlow等软件包来计算偏导数，而不需要手动计算。

2. 梯度下降法的**更新过程简单**，相比于直接求解线性方程组，**计算量小**，**收敛速度快**

梯度下降法（Gradient Descent）

- 假设一个人需要从山的某处开始下山，尽快到达山底。在下山之前他需要确认两件事：下 山的方向和下山的距离。因为下山的路有很多，他必须利用一些信息，找到从该处开始最 陡峭的方向下山，这样可以保证他尽快到达山底。此外，这座山最陡峭的方向并不是一成 不变的，每当走过一段规定的距离，他必须停下来，重新利用现有信息找到新的最陡峭的 方向。通过反复进行该过程，最终抵达山底。

- **梯度下降法用于求解无约束最优化问题**：山代表了需要优化的 函数表达式；山的最低点就是该函数的最优值；每次下山的距 离代表后面要解释的学习率；寻找方向利用的信息即为样本数 据；最陡峭的下山方向则与函数表达式**梯度的方向**有关，之所 以要寻找最陡峭的方向，是为了满足最快到达山底的限制条件；某处——代表了我们给优化函数设置的初始值，算法后面正是利用这个初始值进行不断的迭代求出最优解。



线性回归直接求解

优化算法：以 年龄-薪水 为例

（X）年龄	（Y）薪水
20	5,000
30	10,000
25	7,000
40	15,000
…	…

- 回归方程:

$$h_\theta(x) = \theta_1 x$$

- 参数:

$$\theta_1$$ **简化为只有一个参数**

- 代价函数:

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

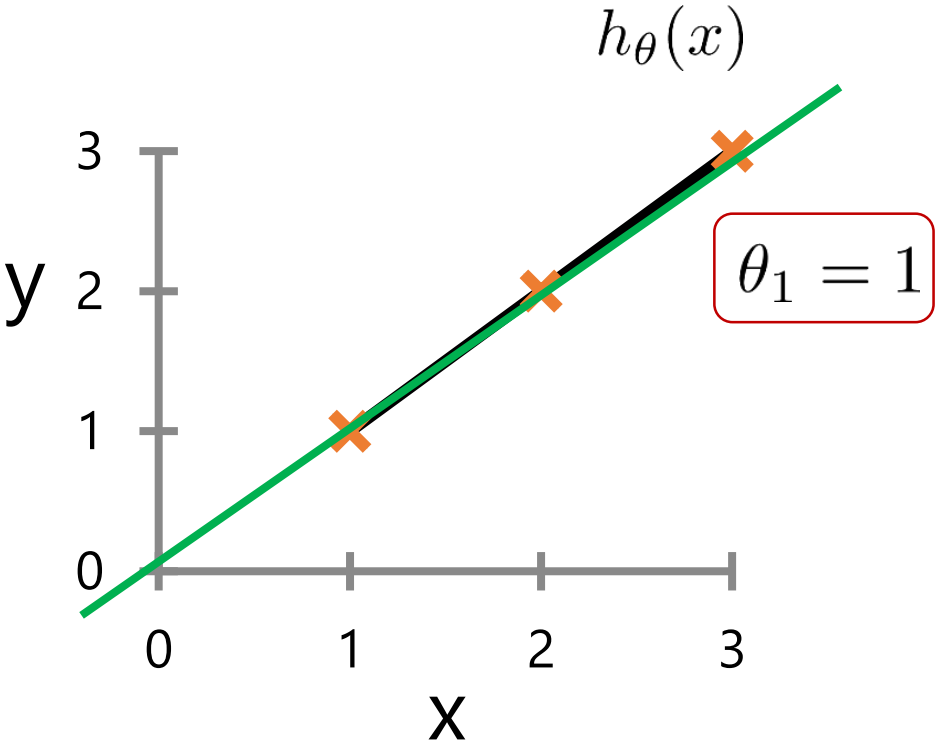
- 目标函数:

$$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$$

梯度下降法求解线性回归

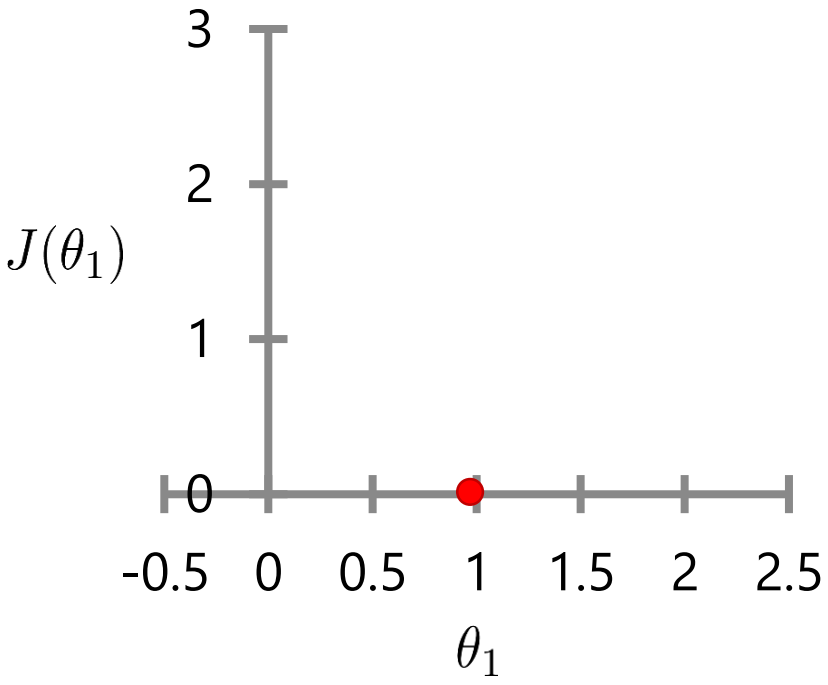
$h_\theta(x)$

(θ_1 不变, 这是x的函数)



$J(\theta_1)$

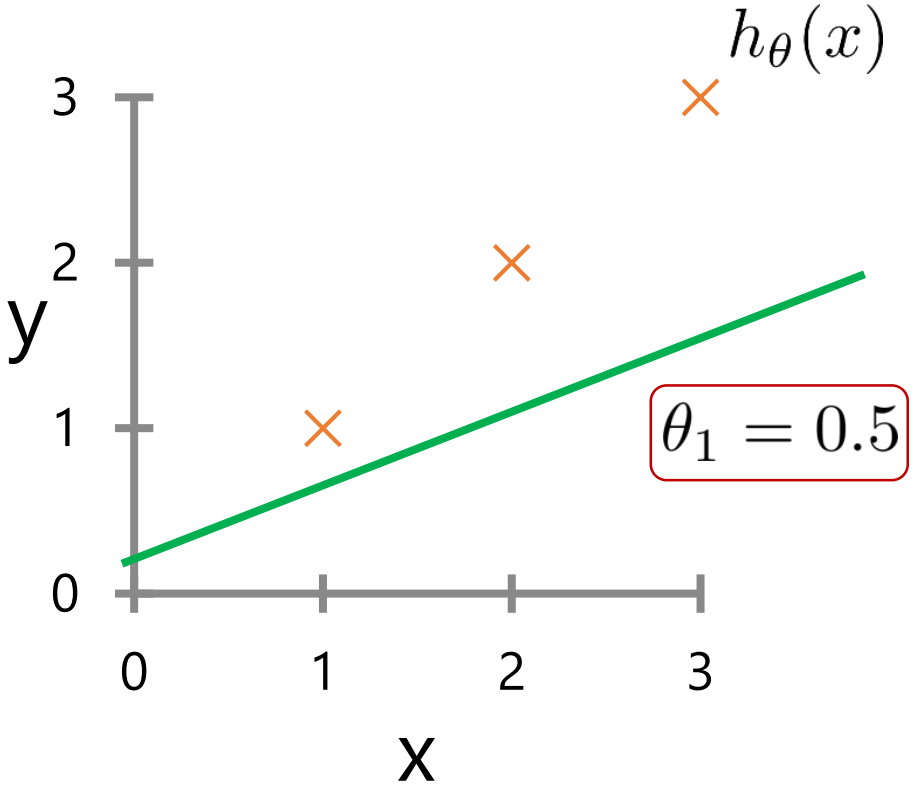
(这是 θ_1 的函数)



梯度下降法求解线性回归

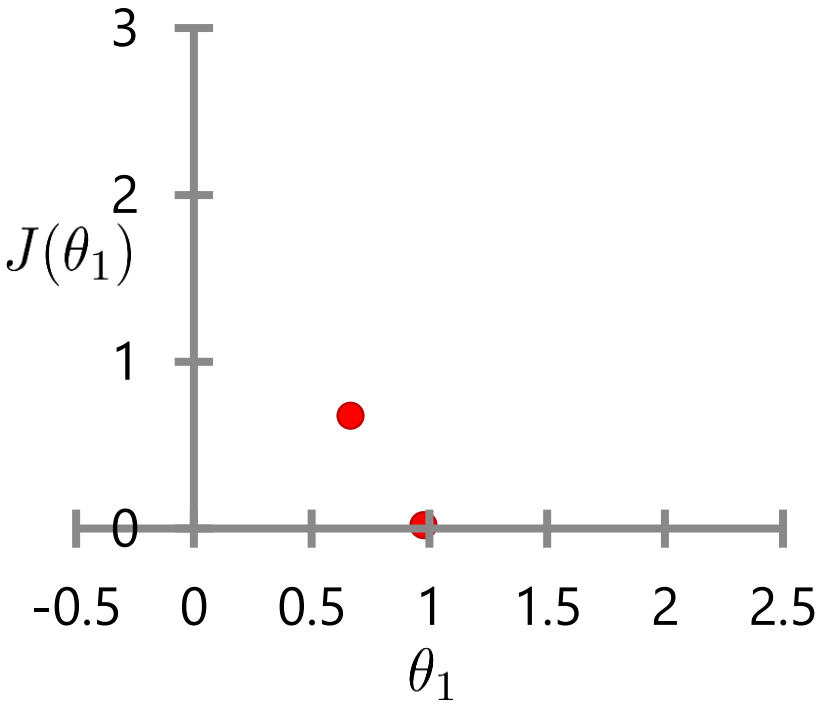
$h_\theta(x)$

(θ_1 不变, 这是x的函数)



$J(\theta_1)$

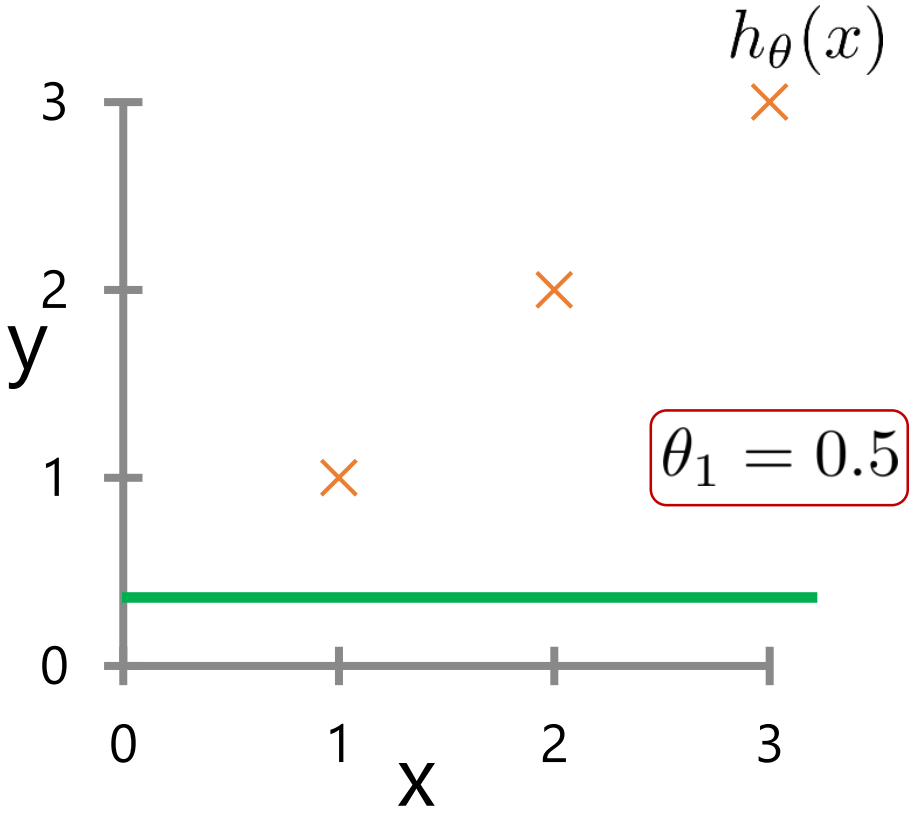
(这是 θ_1 的函数)



梯度下降法求解线性回归

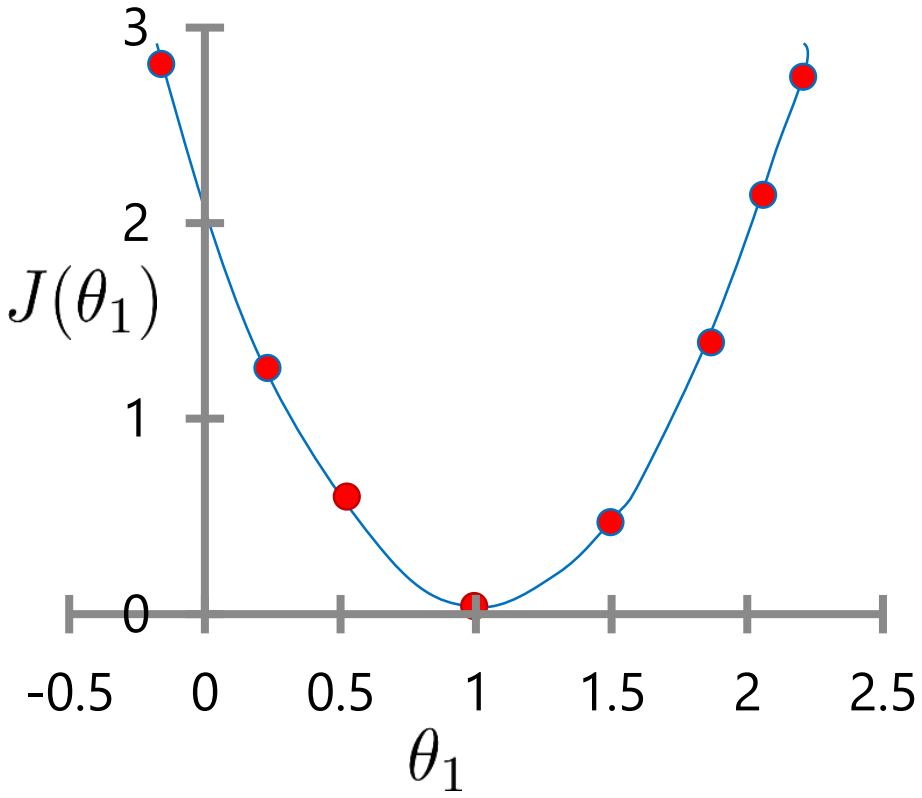
$h_\theta(x)$

(θ_1不变, 这是x的函数)



$J(\theta_1)$

(这是 θ_1 的函数)



梯度下降法求解线性回归

优化算法

- 以房屋交易问题为例，假使我们回归问题的训练集（Training Set）如下表所示：

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

- m 代表训练集中实例的数量
- x 代表特征/输入变量
- y 代表目标变量/输出变量
- (x, y) 代表训练集中的实例
- $(x^{(i)}, y^{(i)})$ 代表第 i 个观察实例
- h 代表学习算法的解决方案或函数也称为假设

在该数据集中，只有一个自变量面积，和一个因变量价格

目的是训练一个线性方程，无限逼近所有数据点，然后利用该方程与给定的某一自变量（本例中为面积），可以预测因变量（本例中为房价）

梯度下降法求解线性回归

- 方程：

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- 参数：

$$\theta_0, \theta_1$$

- 代价函数：

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- 目标函数：

$$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

梯度下降法求解线性回归

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

}

朝梯度下降的方向

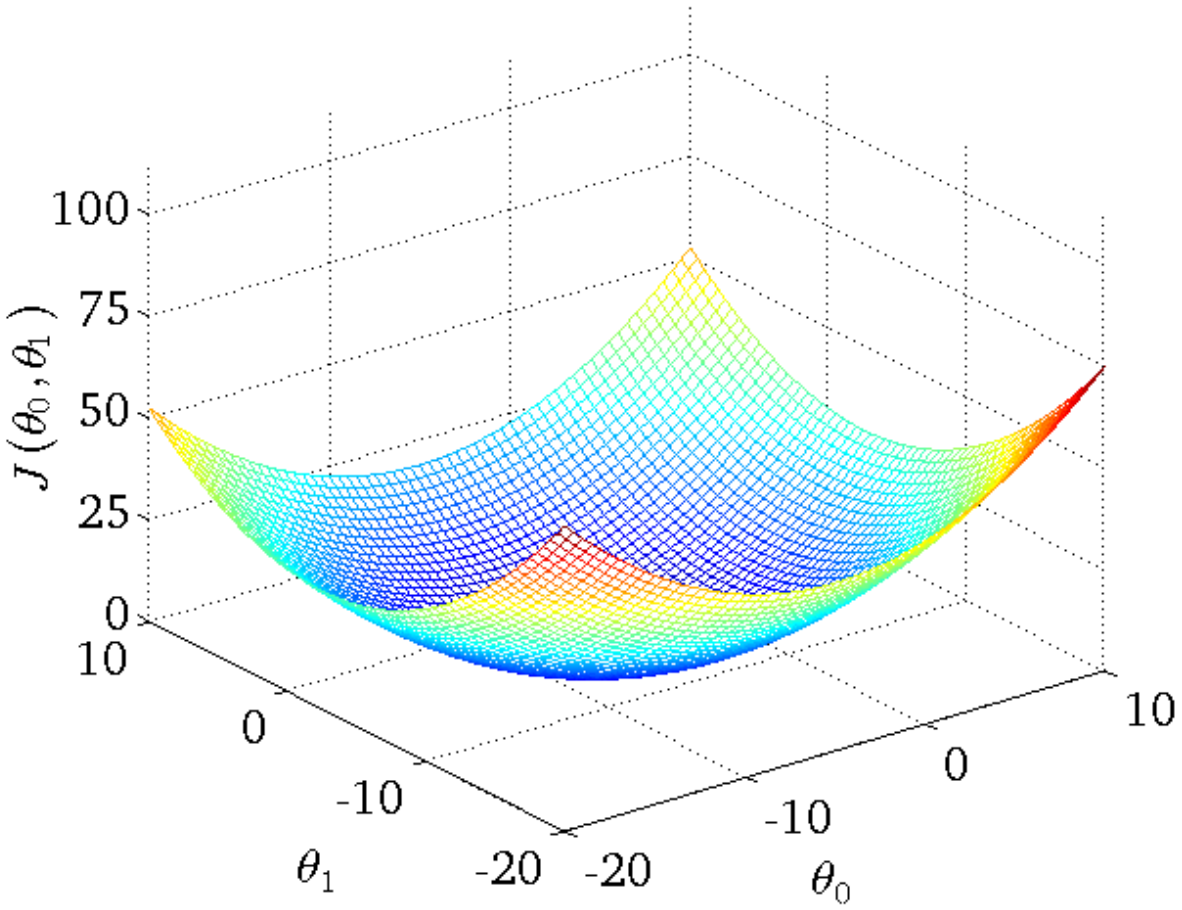
同步更新 θ_0 和 θ_1

学习率:
控制步长

关于 θ_0 和 θ_1 的偏导

梯度下降法求解线性回归

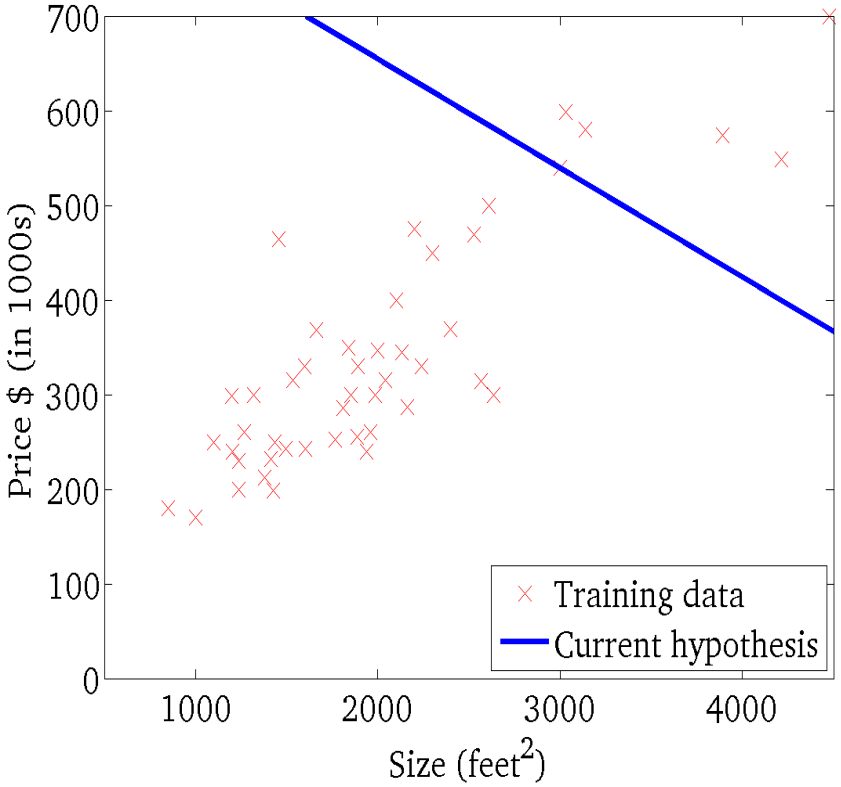
梯度下降法的代价函数是凸函数



梯度下降法优化过程

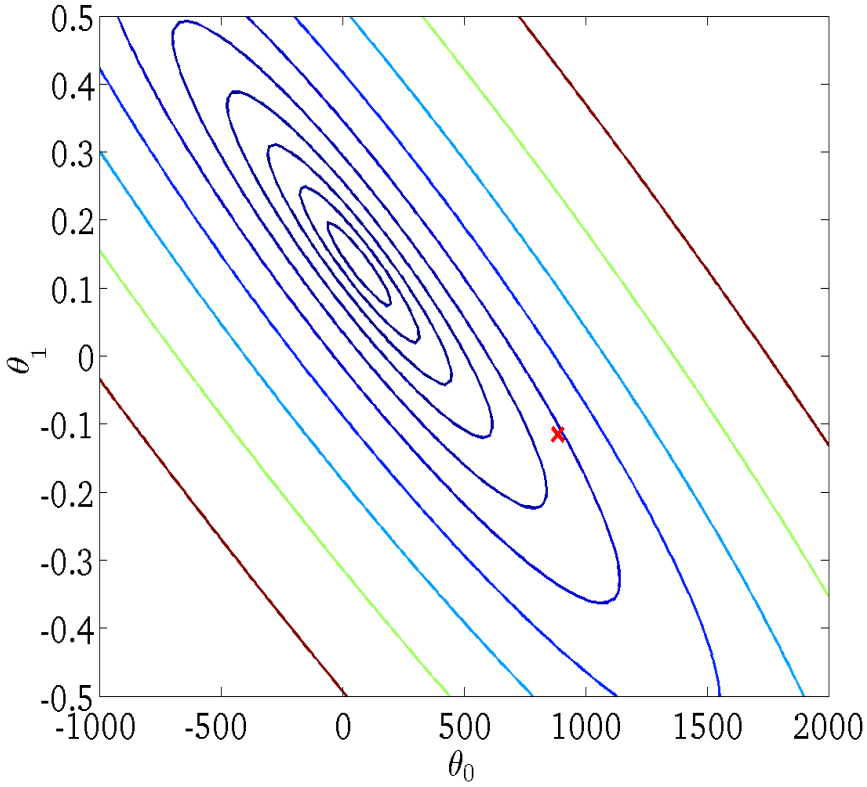
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)



$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)



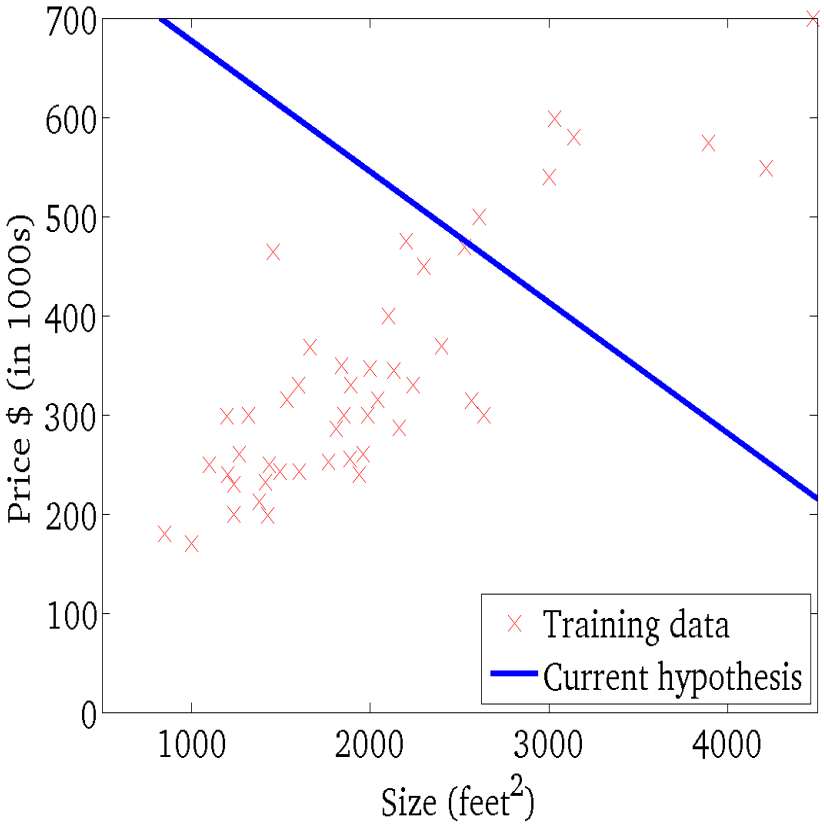
梯度下降法优化过程

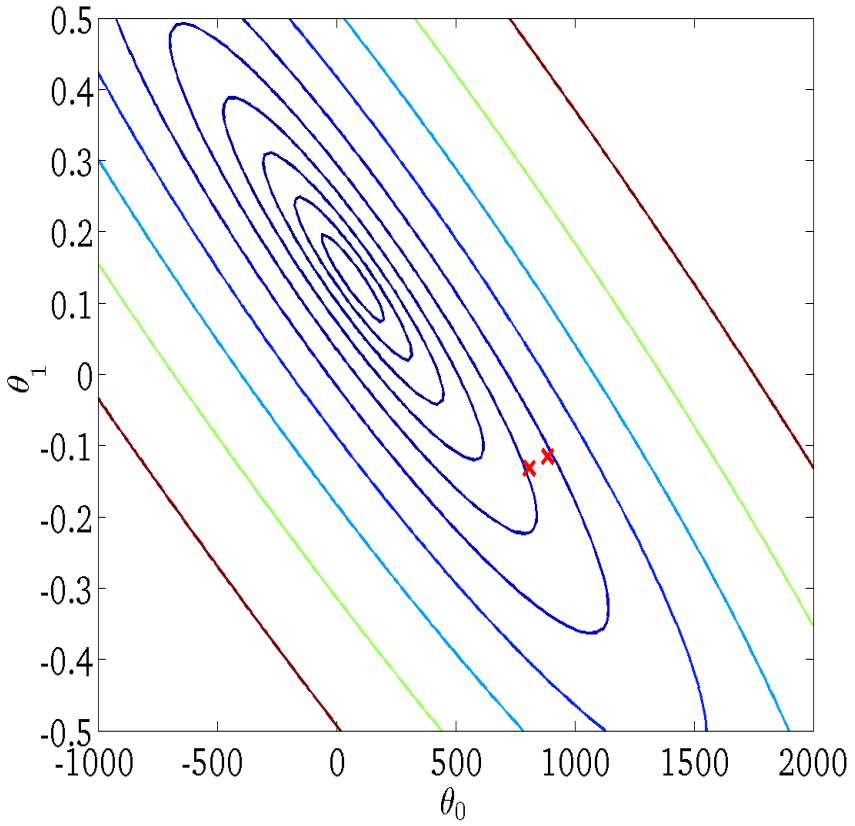
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





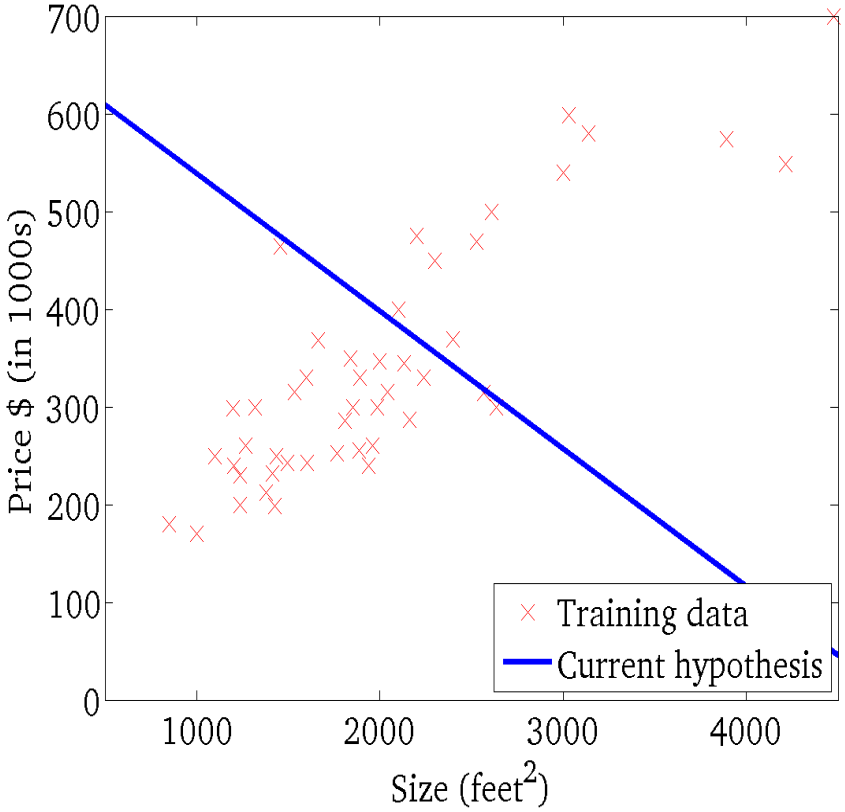
梯度下降法优化过程

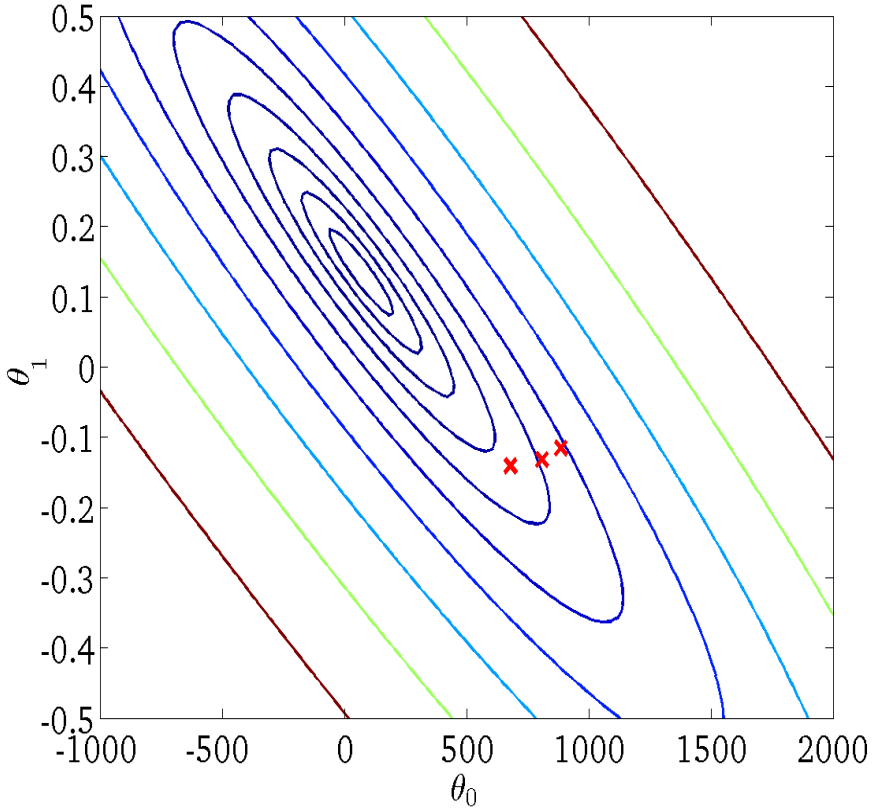
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





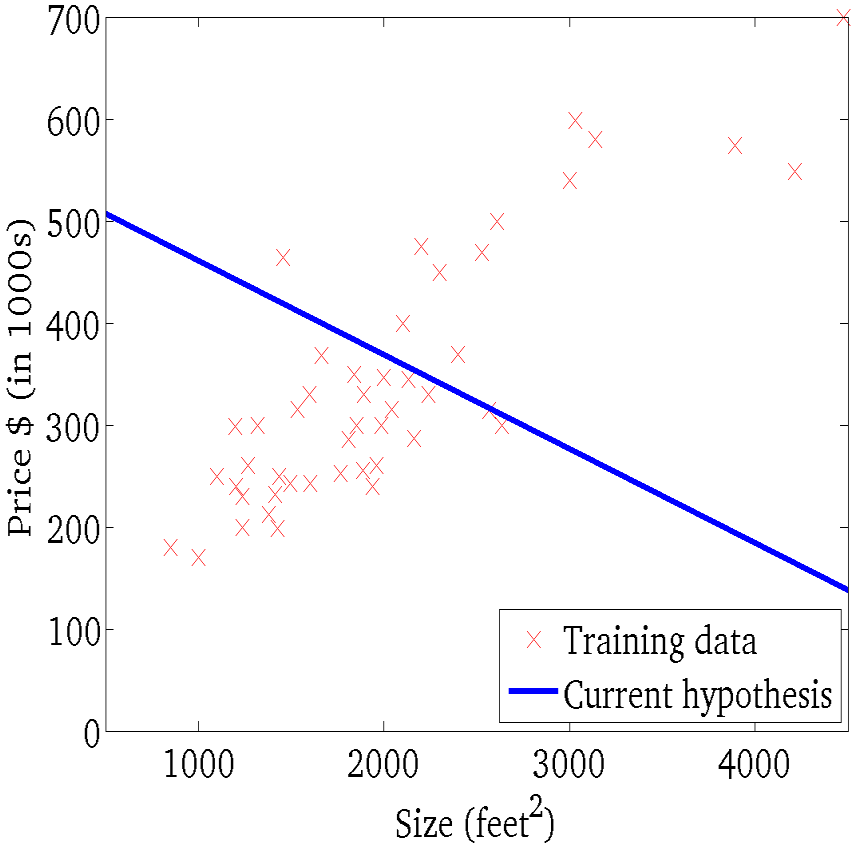
梯度下降法优化过程

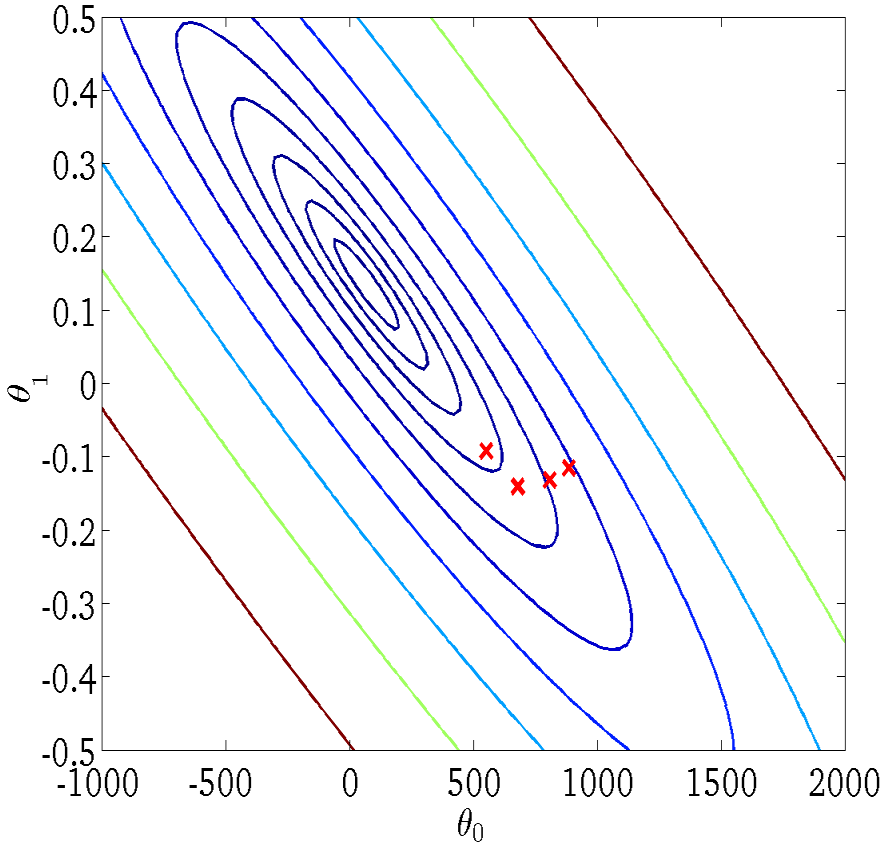
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





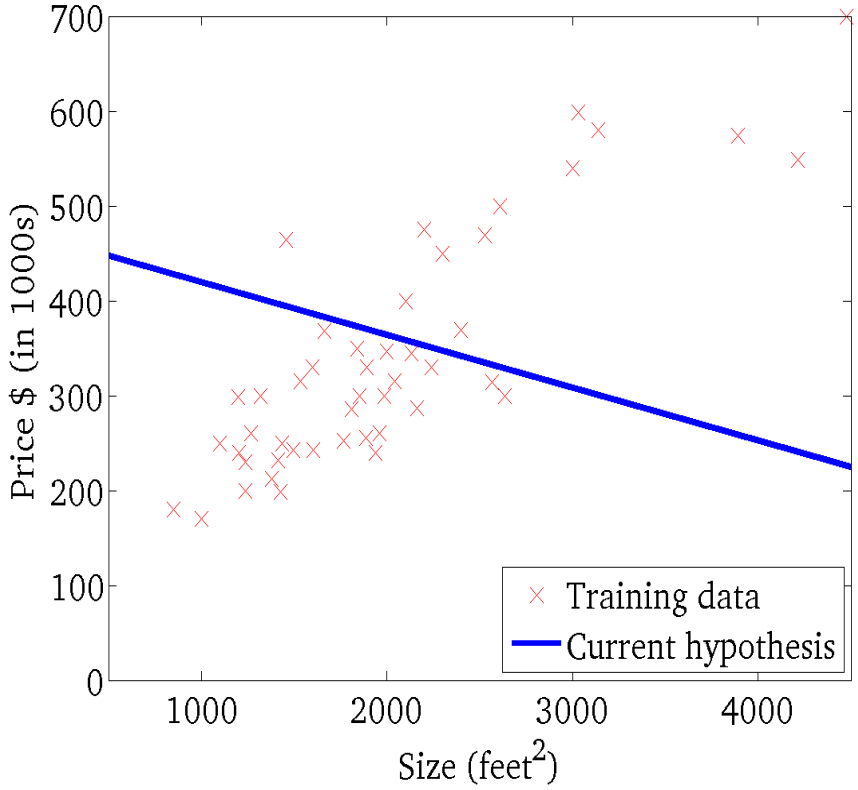
梯度下降法优化过程

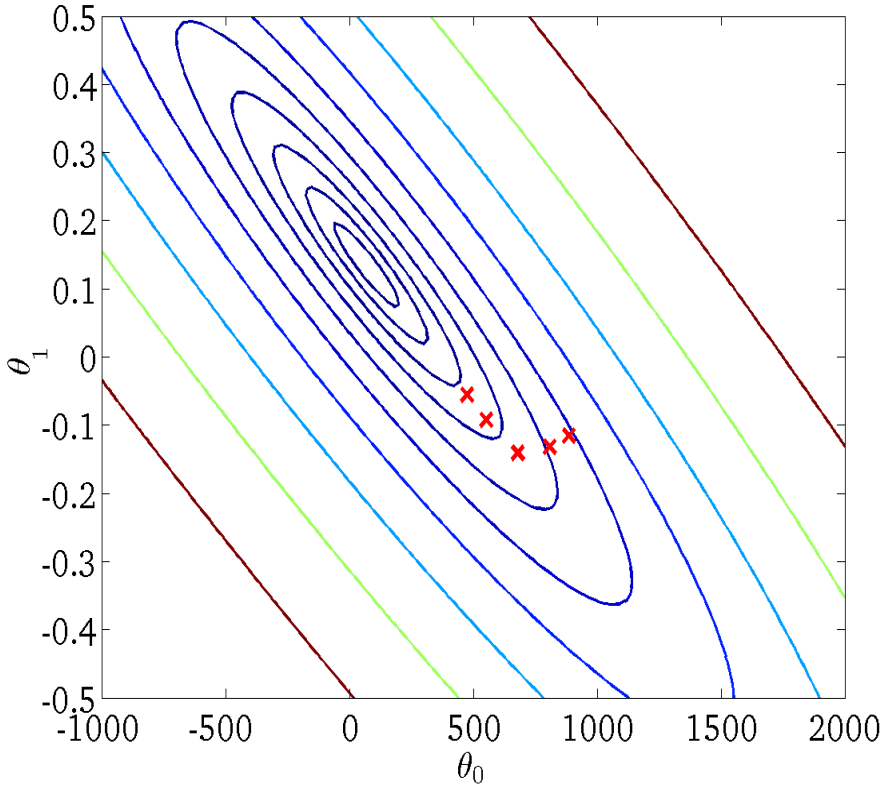
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





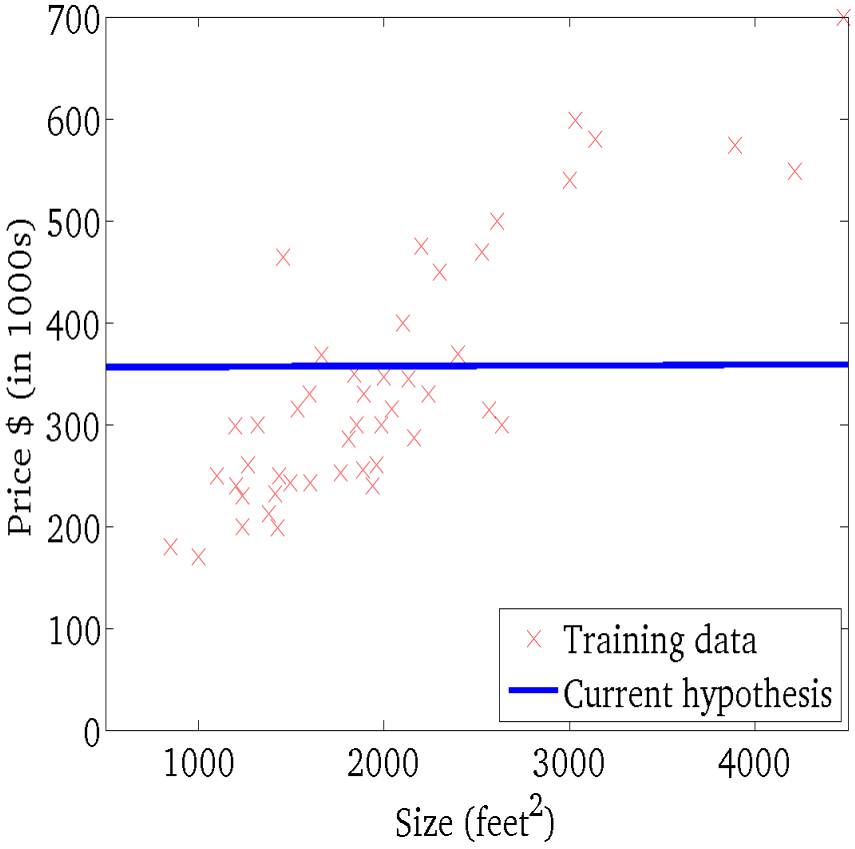
梯度下降法优化过程

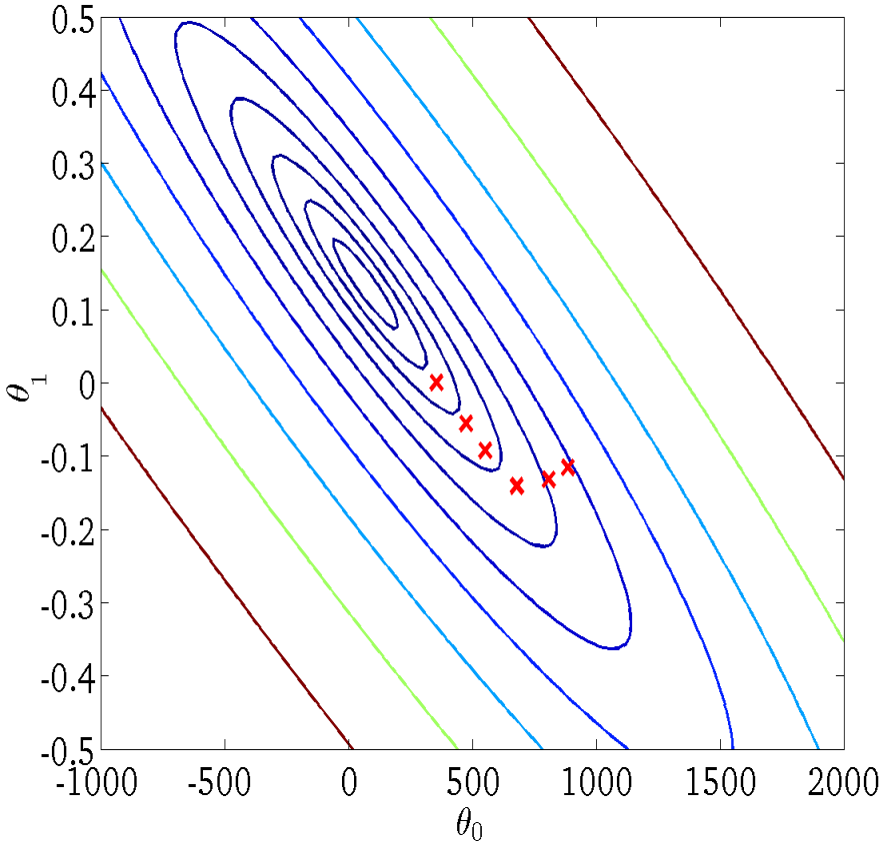
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





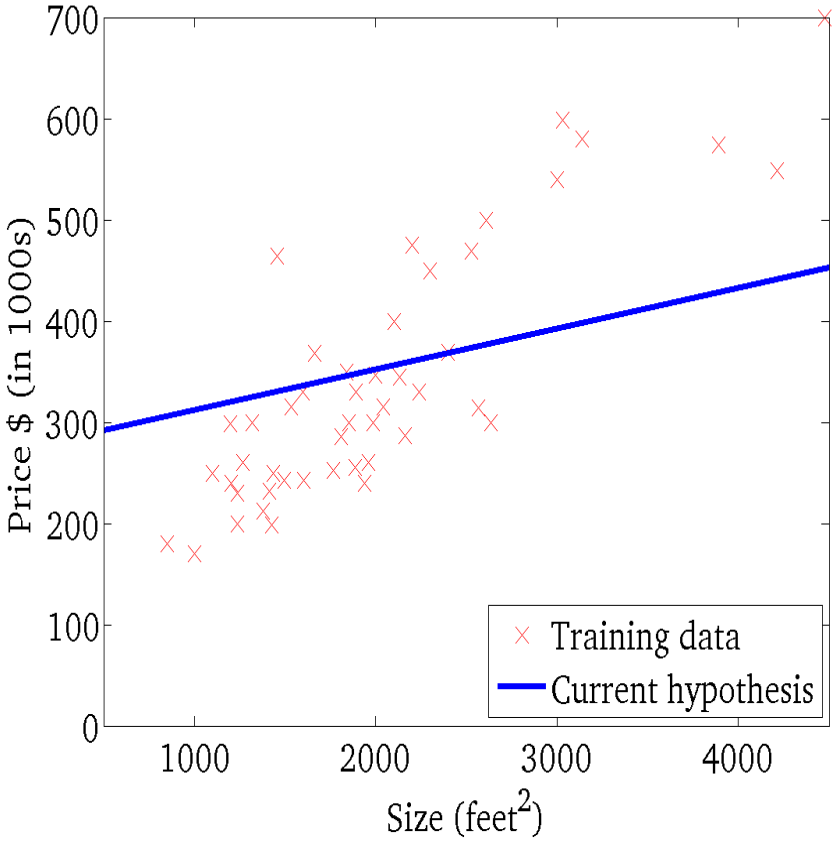
梯度下降法优化过程

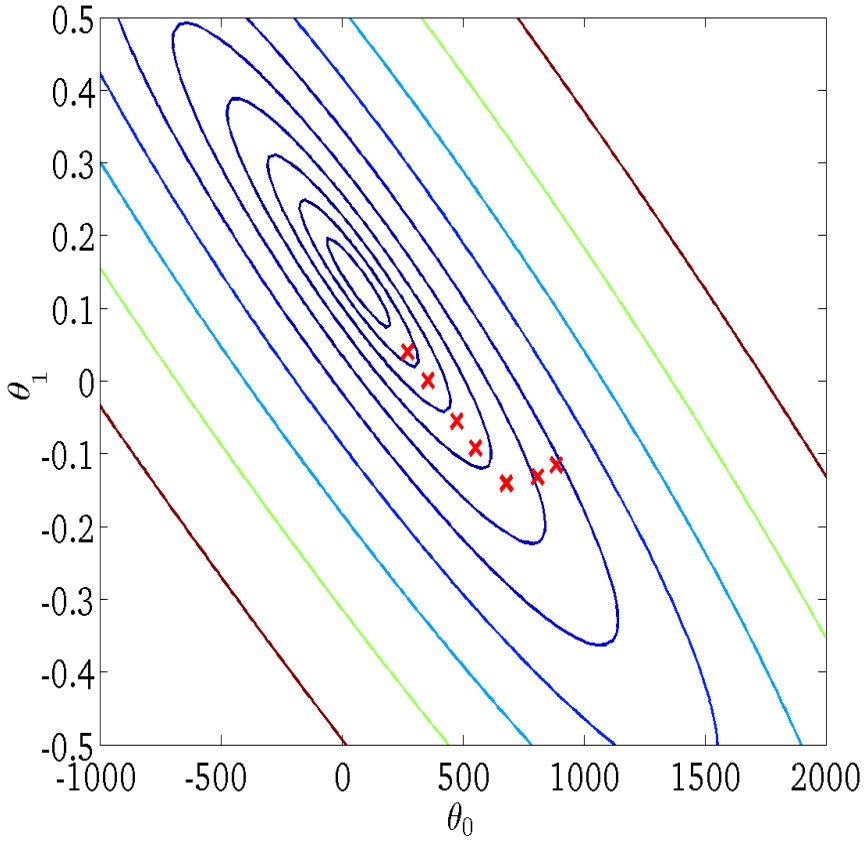
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)





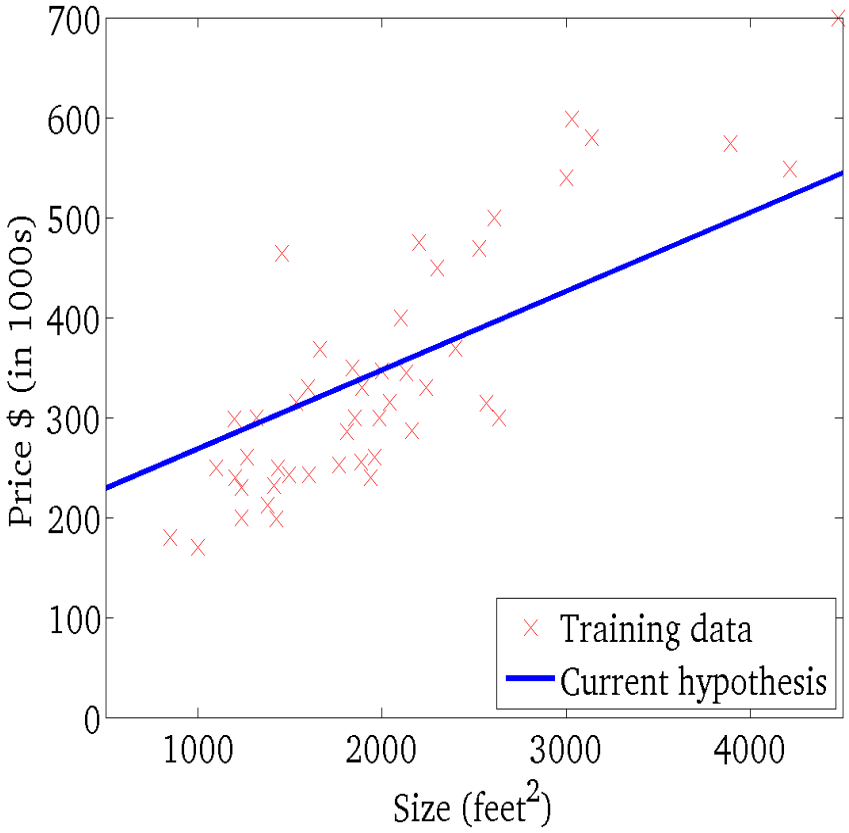
梯度下降法优化过程

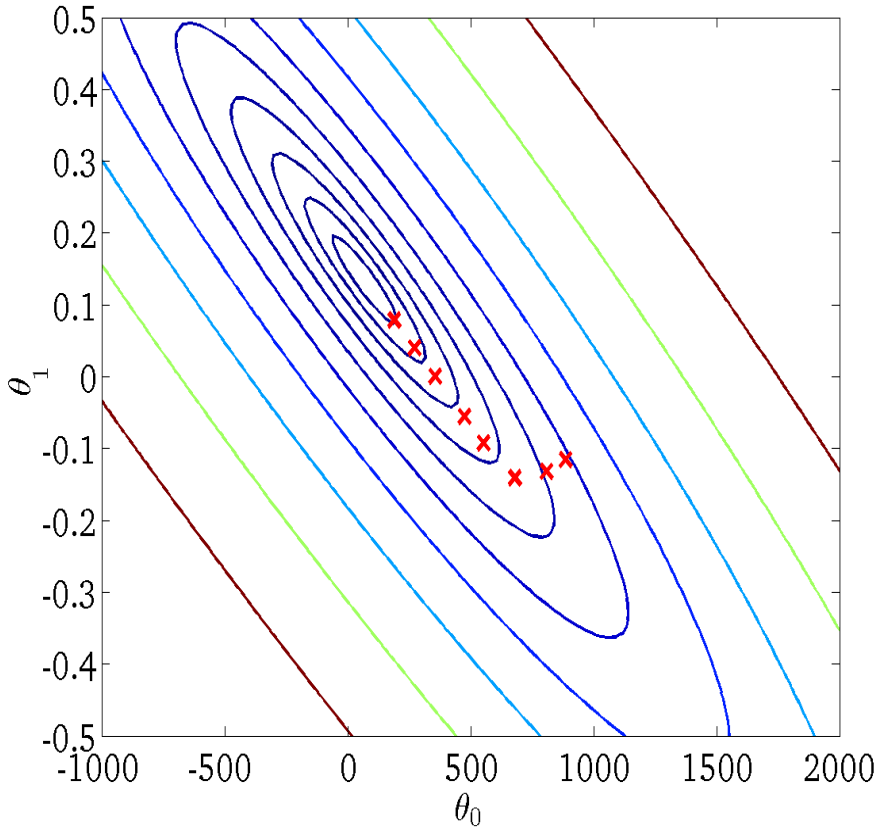
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)

$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)

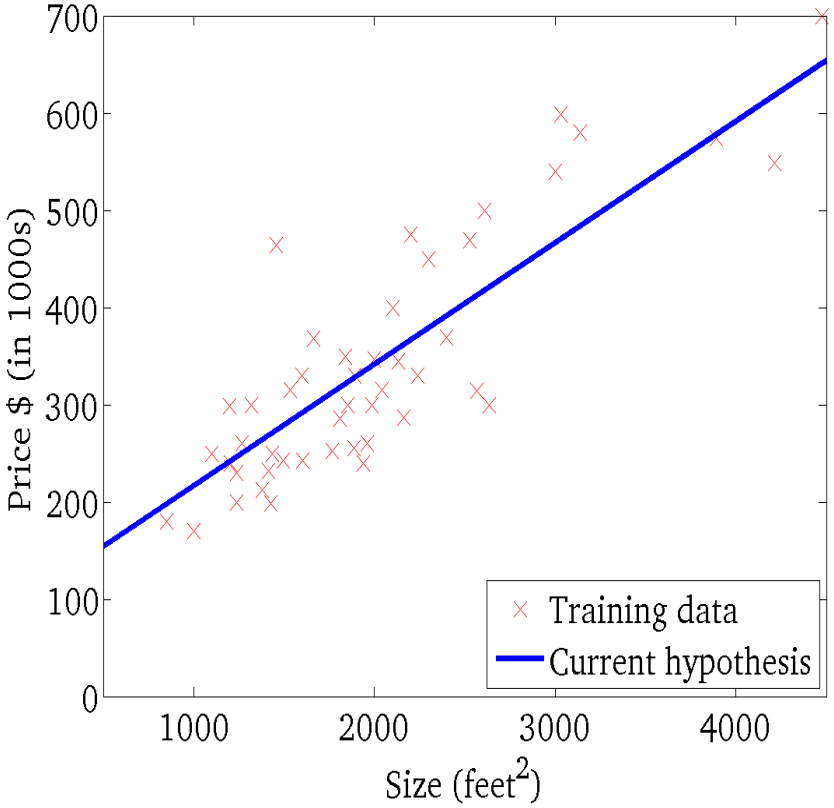




梯度下降法优化过程

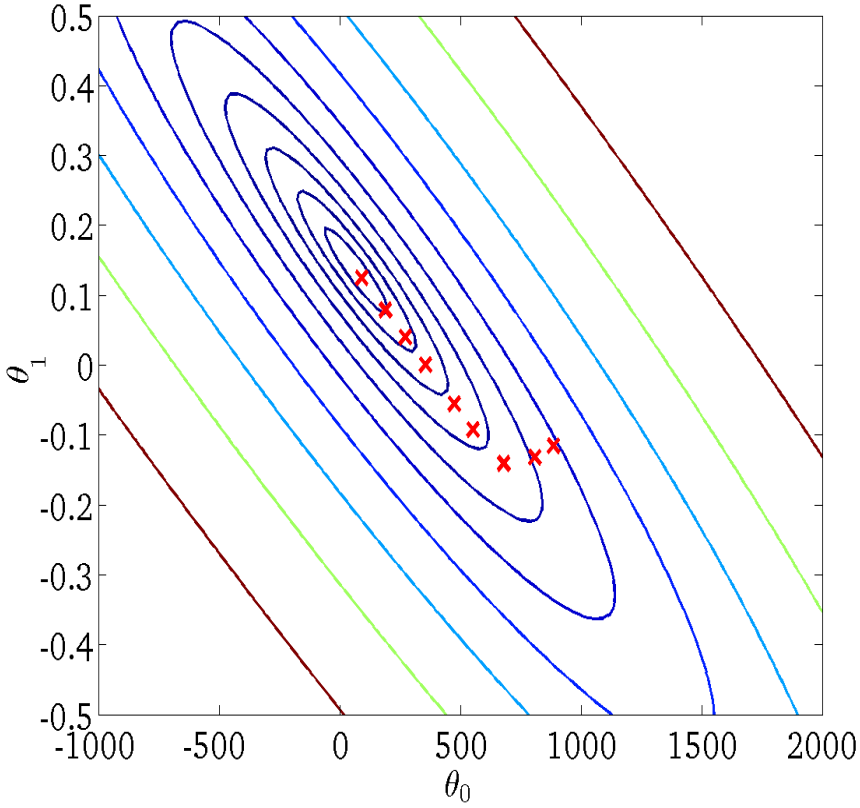
$h_\theta(x)$

(θ_0, θ_1固定,这是x的函数)



$J(\theta_0, \theta_1)$

(参数 θ_0, θ_1 的函数)



梯度下降法求解线性回归

优化算法

计算代价函数的偏导数

方法二：用梯度下降法求解

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{E}}{\partial w_j}$$

$$\Rightarrow \quad w_j \leftarrow w_j - \alpha \frac{1}{N} \sum_{i=1}^{N} x_j \left(\hat{y}^{(i)} - y^{(i)} \right)$$

重复更新多次，直到梯度接近于零

通过设置偏导数为零，我们能够得到准确的解吗？

梯度下降法求解线性回归

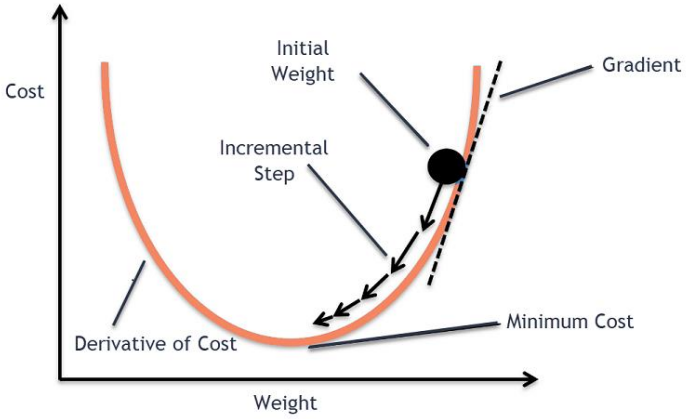
优化算法

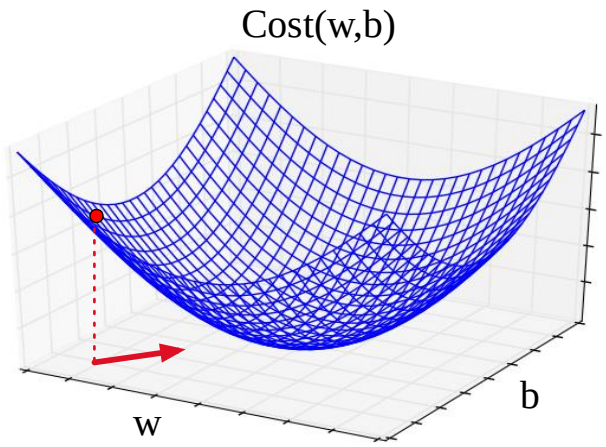
计算代价函数的偏导数

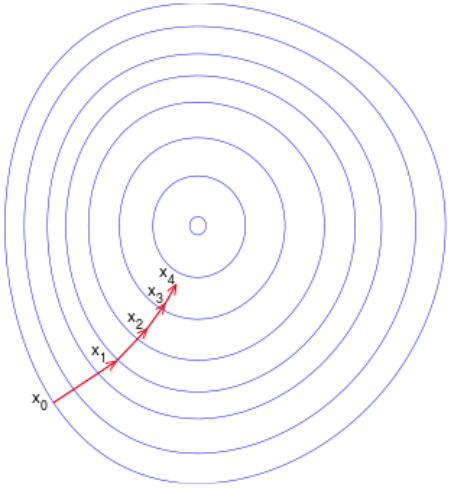
方法二：用梯度下降法求解

通过设置偏导数为零，我们能够得到准确的解吗？

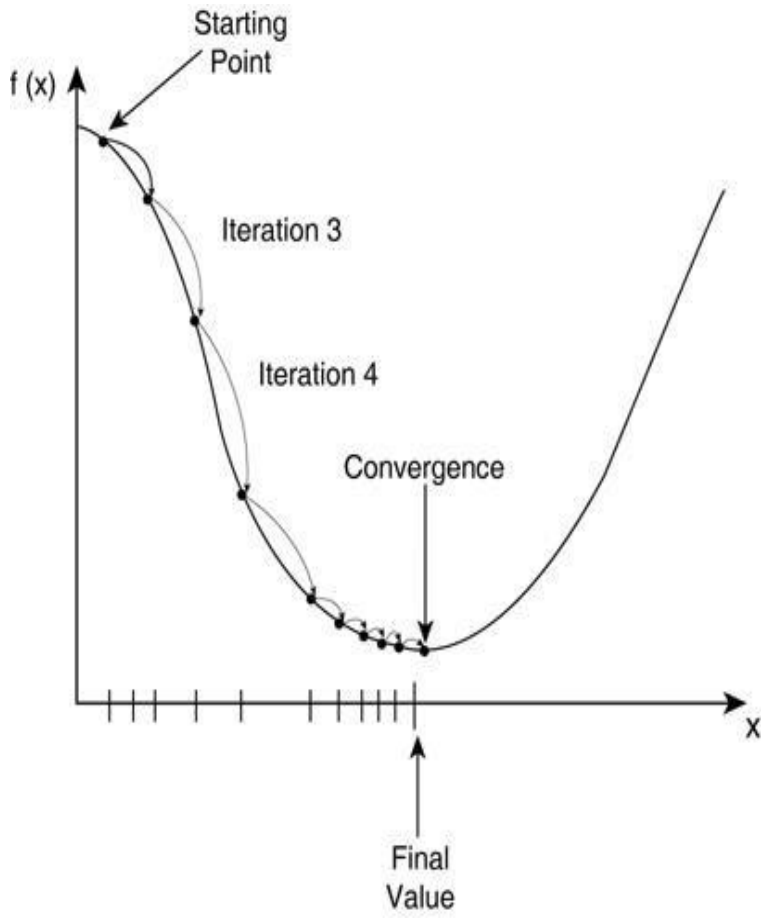
使用梯度下降法，我们永远不能真正达到最优，但可以逐渐接近最优







梯度下降法（Gradient Descent）



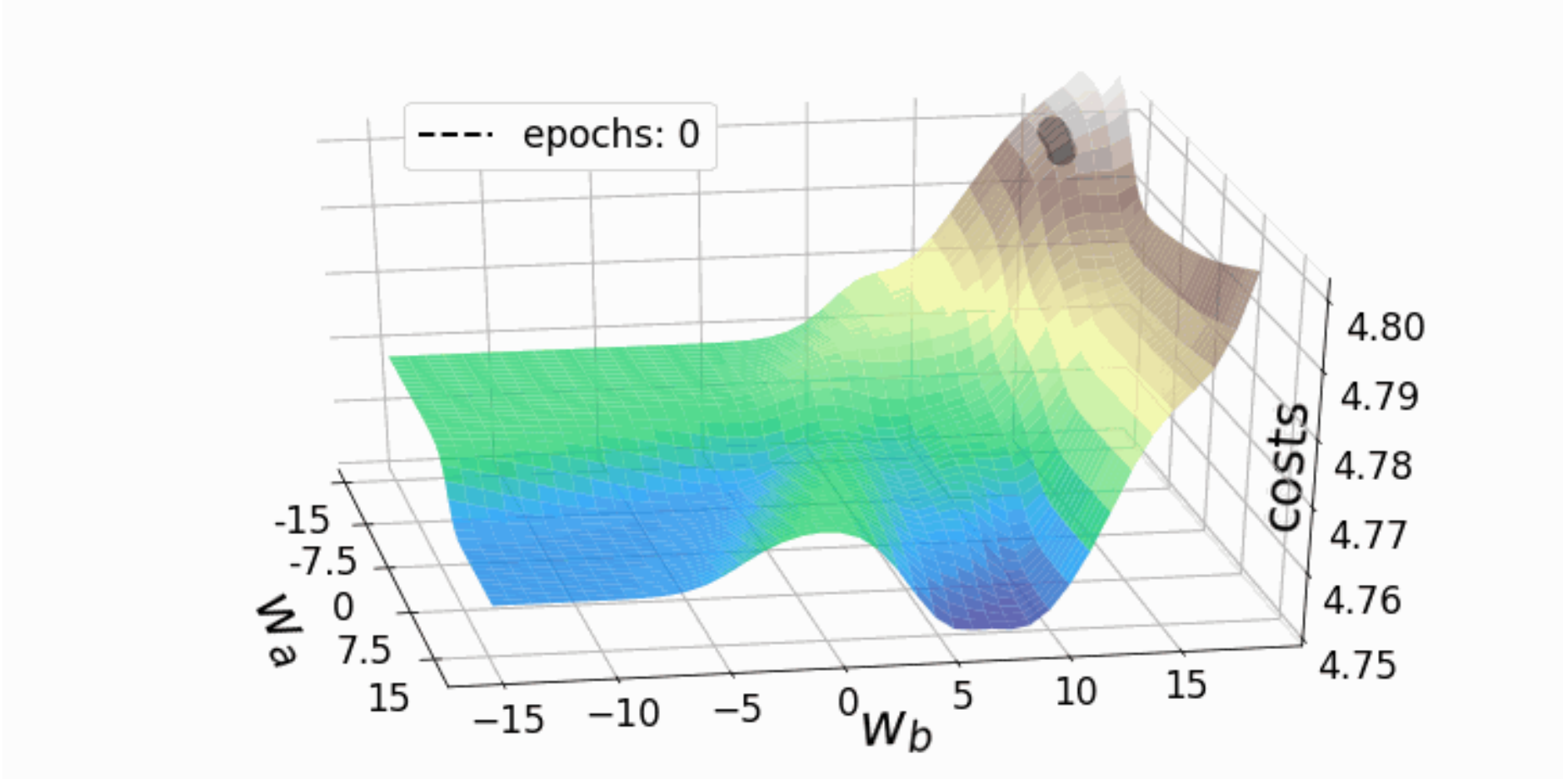
1. 给定待优化连续可微函数 $J(\Theta)$、学习率 α 以及一组初始值 $\Theta_0 = (\theta_{01}, \theta_{02}, \cdots, \theta_{0l},)$
2. 计算待优化函数梯度：$\nabla J(\Theta_0)$
3. 更新迭代公式：$\Theta^{0+1} = \Theta_0 - \alpha \nabla J(\Theta_0)$
4. 计算 Θ^{0+1} 处函数梯度 $\nabla J(\Theta_{0+1})$
5. 计算梯度向量的模来判断算法是否收敛：$\|\nabla J(\Theta)\| \leqslant \varepsilon$
6. 若收敛，算法停止，否则根据迭代公式继续迭代

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t}$$

$$= \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}\left(\theta_t; x^{(i)}, y^{(i)}\right)}{\partial \theta}.$$

- 经过迭代计算风险函数的最小值
- 搜索步长 α 中也叫作学习率（Learning Rate）

梯度下降法（Gradient Descent）



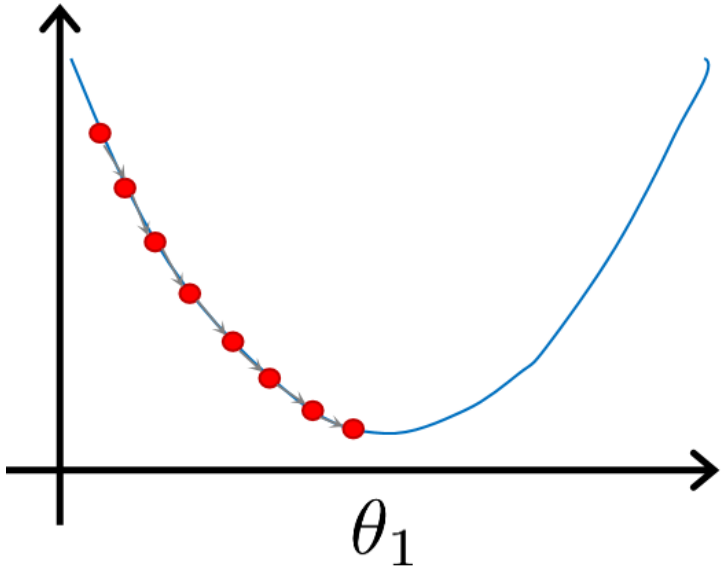
https://towardsdatascience.com/animations-of-gradient-descent-and-loss-landscapes-of-neural-networks-in-python-e757f3584057

梯度下降法求解线性回归

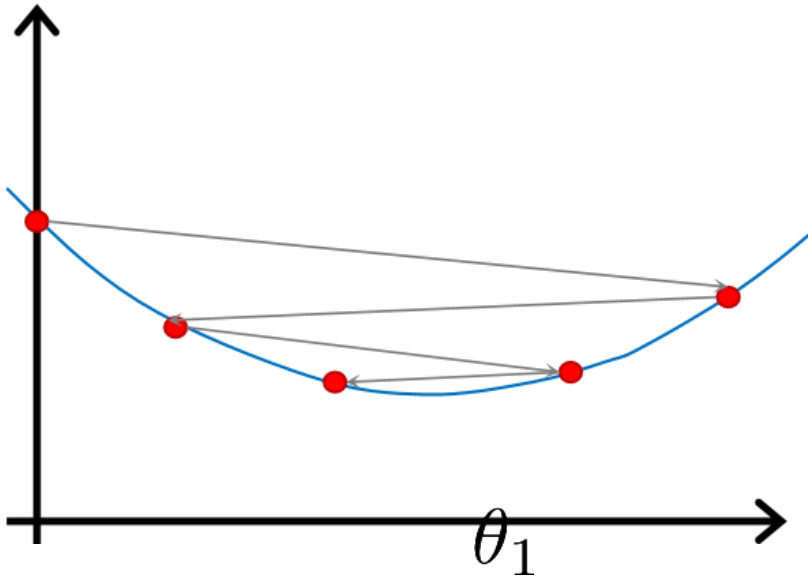
学习率

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

如果学习率过小，收敛速度会减慢。

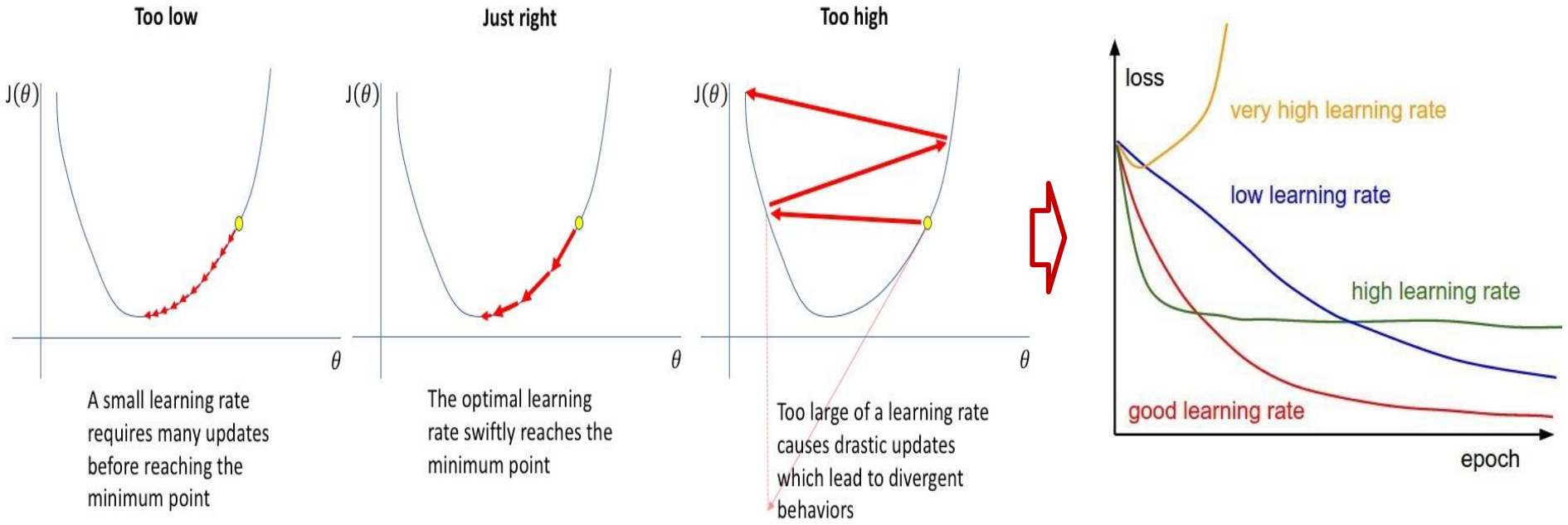


如果学习率过大，会导致模型参数在训练过程中不断震荡，甚至无法收敛。



梯度下降法（Gradient Descent）

- 学习率是十分重要的超参数！



随机梯度下降法（Stochastic Gradient Descent，SGD）

（批量）梯度下降法在每次迭代都需计算每个样本上损失函数的梯度并加和，计算复杂度较大；为了降低迭代的计算复杂度，可以每次迭代只采集一个样本，计算该样本的损失函数的梯度并更新参数，即随机梯度下降法。

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t} \\ &= \theta_t - \alpha \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}\left(\theta_t; x^{(i)}, y^{(i)}\right)}{\partial \theta}.\end{aligned}$$

速度慢！大数据内存不足！

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

方差大！损失函数震荡严重！

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

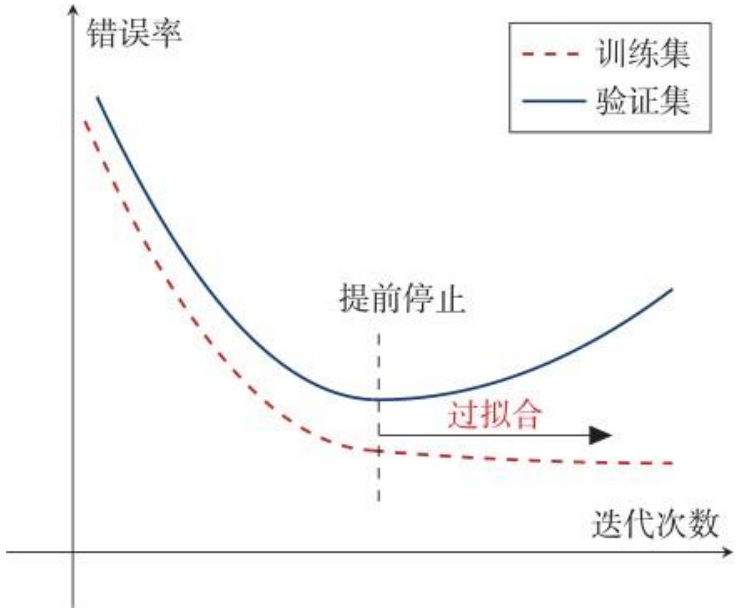
until 模型 $f(\mathbf{x}; \theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: θ

其他优化方法

- **小批量（Mini-Batch）随机梯度下降法**
- 小批量梯度下降法（Mini-Batch Gradient Descent）是批量梯度下降和随机梯度下 降的折中。每次迭代时，随机选取一小部分训练样本来计算梯度并更新参数，这样既 可以兼顾随机梯度下降法的优点，也可以提高训练效率。

- **提前停止法**
- 验证集上错误率不再下降，就停止迭代



多选题 1分

设置

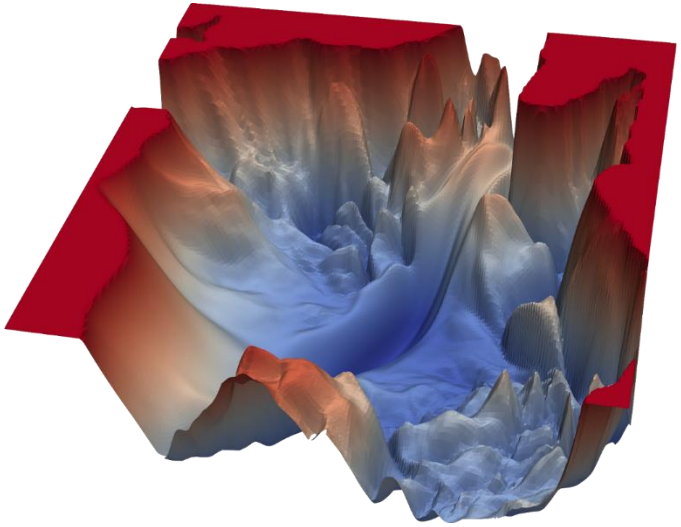
右图是一个一个56层卷积神经网络(VGG-56)的损失函数图景，
下列因素影响模型（神经网络）是否能够收敛到到最优的是 （）

A 学习率

B 优化算法

C 模型初始化参数

D 模型训练轮次



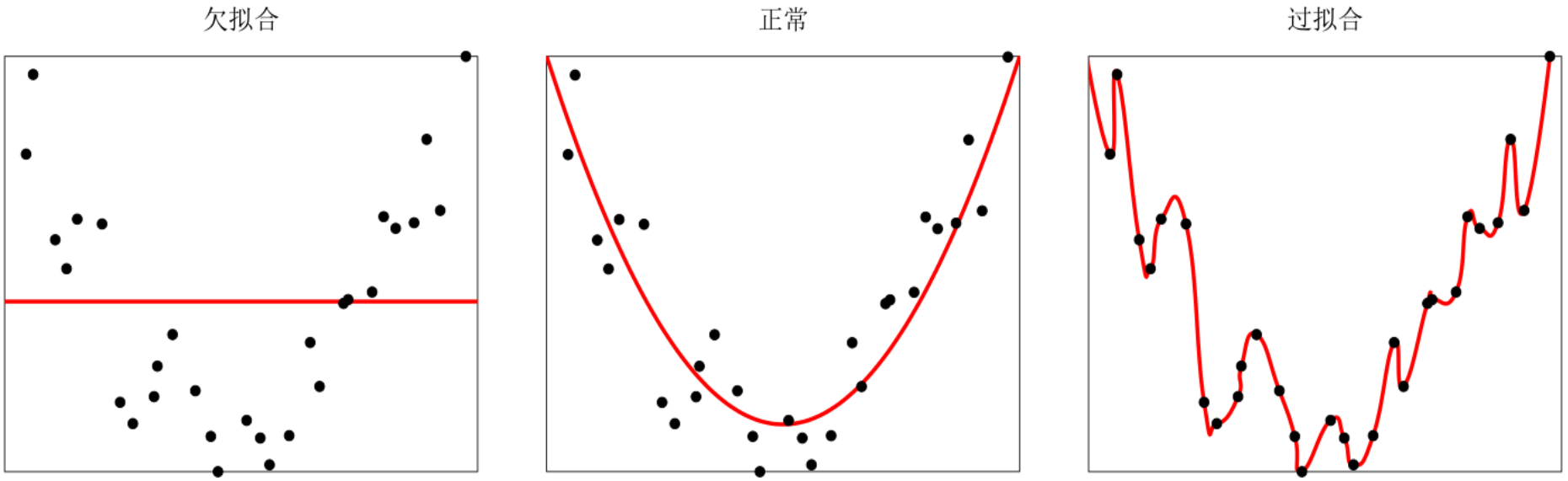
一个56层卷积神经网络(VGG-56)的
损失函数图景

提交

https://arxiv.org/abs/1712.09913

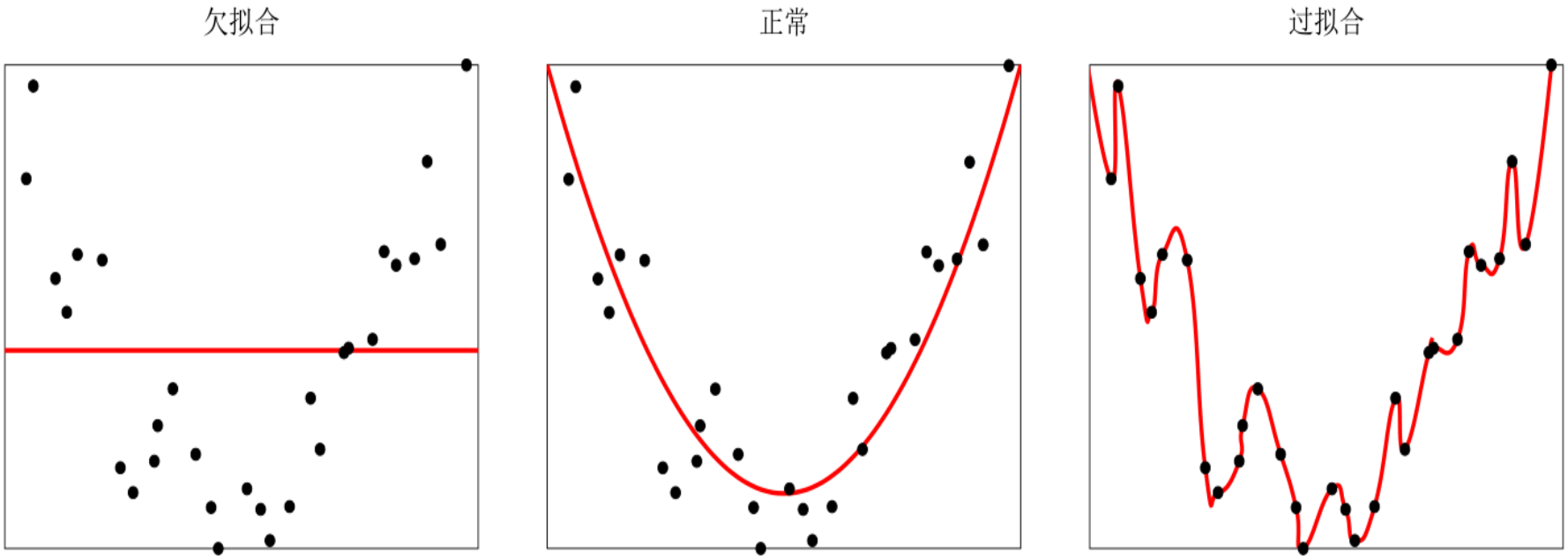
过拟合和欠拟合

- **欠拟合（Underfitting）：**
- 即模型不能很好地拟合训练数据，在训练集上的错误率比较高。
- 欠拟合一般是由于**模型能力不足**造成的，说明其对训练样本的一般性质尚未学好。



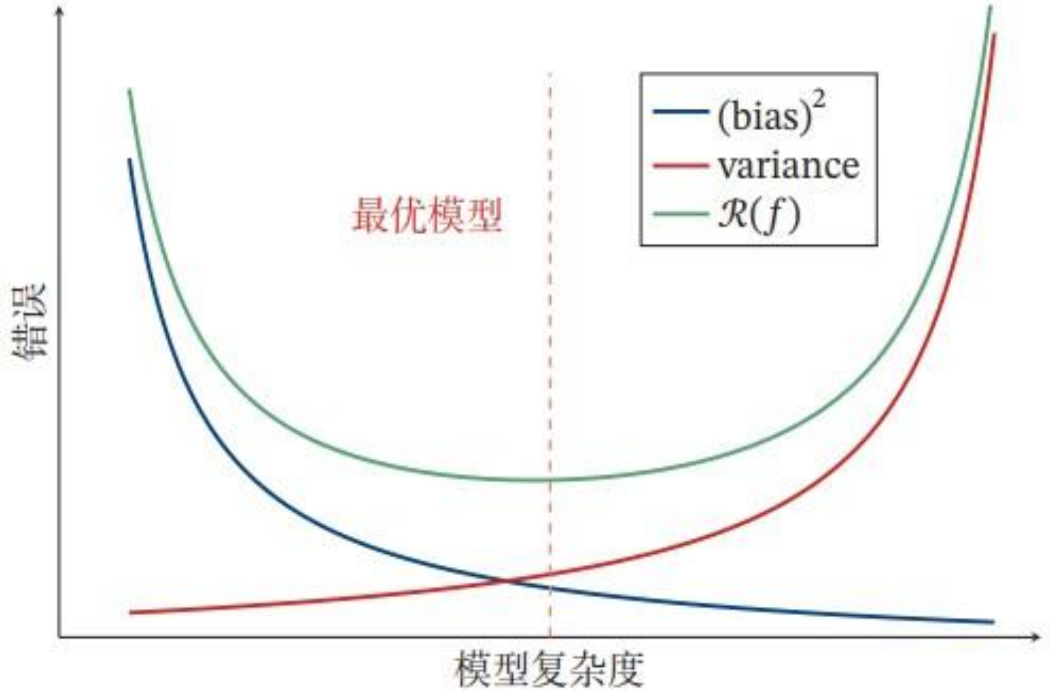
过拟合和欠拟合

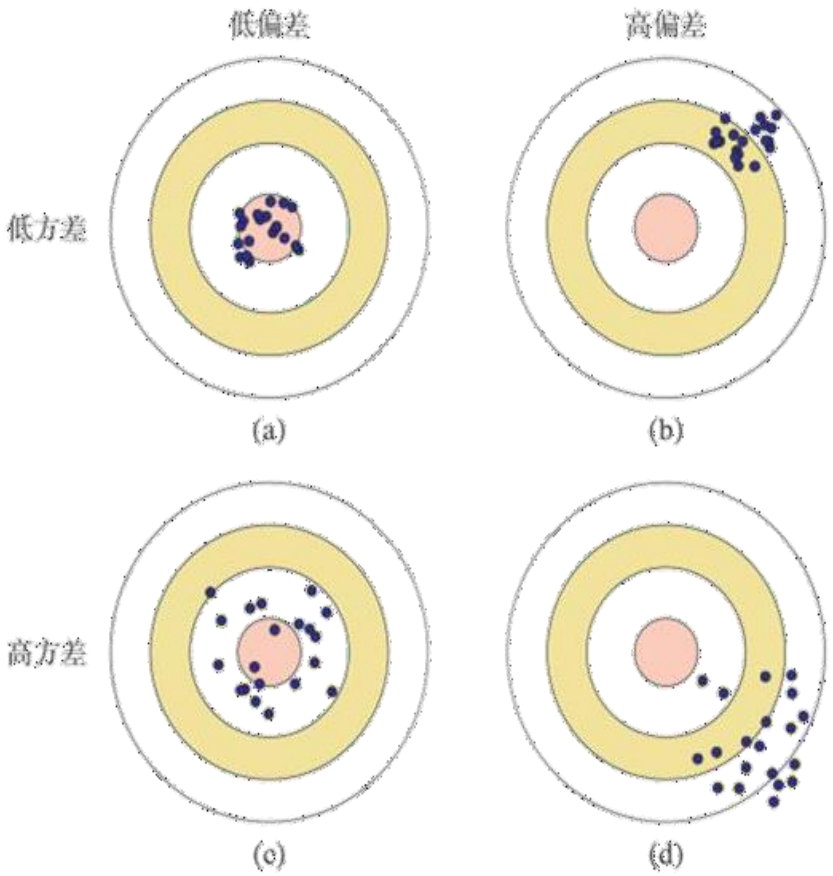
- **过拟合（Overfitting）：**
- 学习器把训练样本学习得“太好”，将训练样本本身的特点当做所有样本的 一般性质，导致泛化性能下降。
- 过拟合问题往往是**由于训练数据少和噪声以及模型能力强**等原因造成的。



偏差-方差分解（Bias-Variance Decomposition）

- 最小化期望错误等价于最小化偏差和方差之和





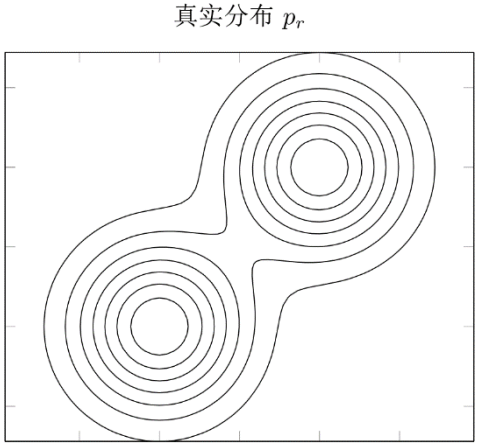
泛化错误

期望风险

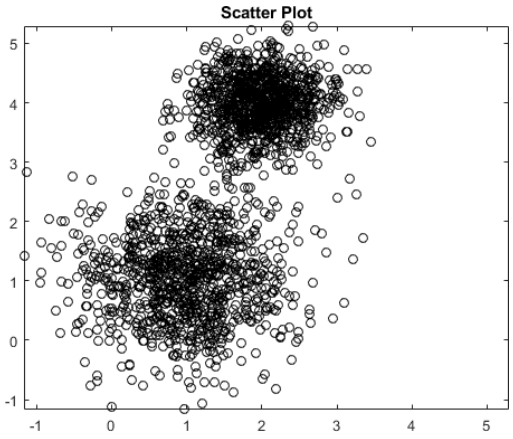
$$\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}), y)],$$

经验风险

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



≠



正则化

限制模型能力，
使其不要过度地
最小化经验风险

泛化错误 $\mathcal{G}_{\mathcal{D}}(f) = \mathcal{R}(f) - \mathcal{R}_{\mathcal{D}}^{emp}(f)$

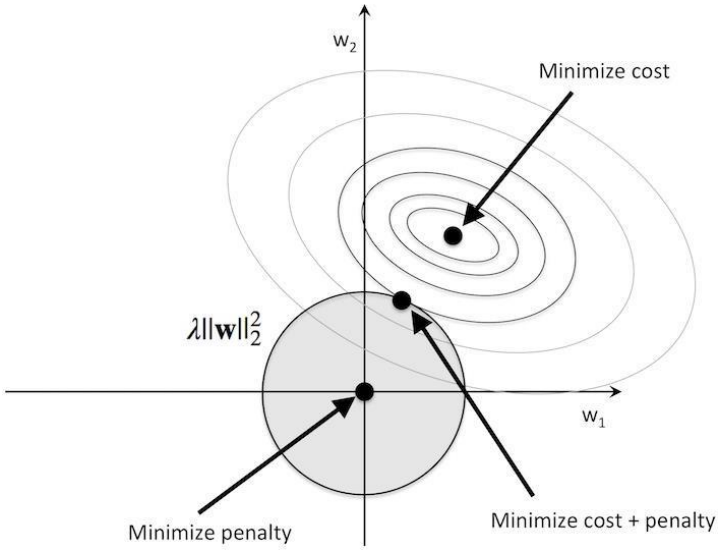
- ✓ 泛化错误可以衡量一个机器学习模型是否可以很好地泛化到未知数据。
- ✓ 泛化错误一般表现为一个模型在训练集和测试集上的错误率。
- ✓ 机器学习的目标是减少泛化错误。

正则化（regularization）

所有损害优化的方法都是正则化

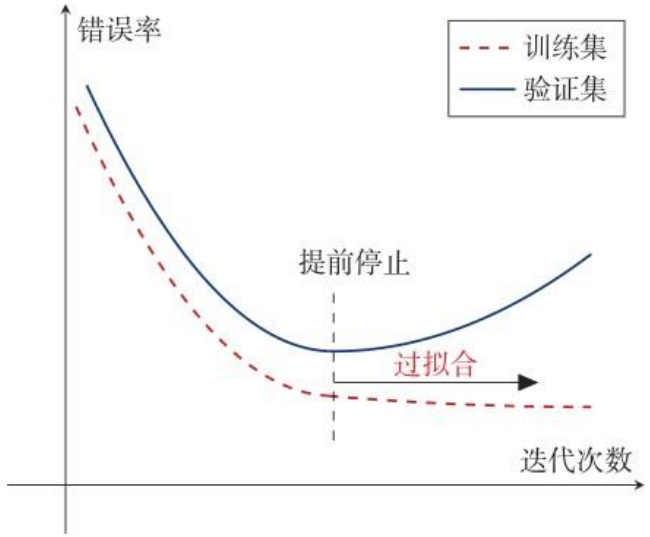
增加优化约束

L1/L2约束、数据增强



干扰优化过程

权重衰减、随机梯度下降、提前停止



正则化（regularization）

- **范数**

矩阵的范数，常用的ℓ_p范数定义

$$\|\boldsymbol{A}\|_p = \left(\sum_{m=1}^{M} \sum_{n=1}^{N} |a_{mn}|^p \right)^{1/p}.$$

ℓ_1范数 ℓ_1范数为向量的各个元素的绝对值之和.

$$\|\boldsymbol{v}\|_1 = \sum_{n=1}^{N} |v_n|.$$

ℓ_2范数 ℓ_2范数为向量的各个元素的平方和再开平方.

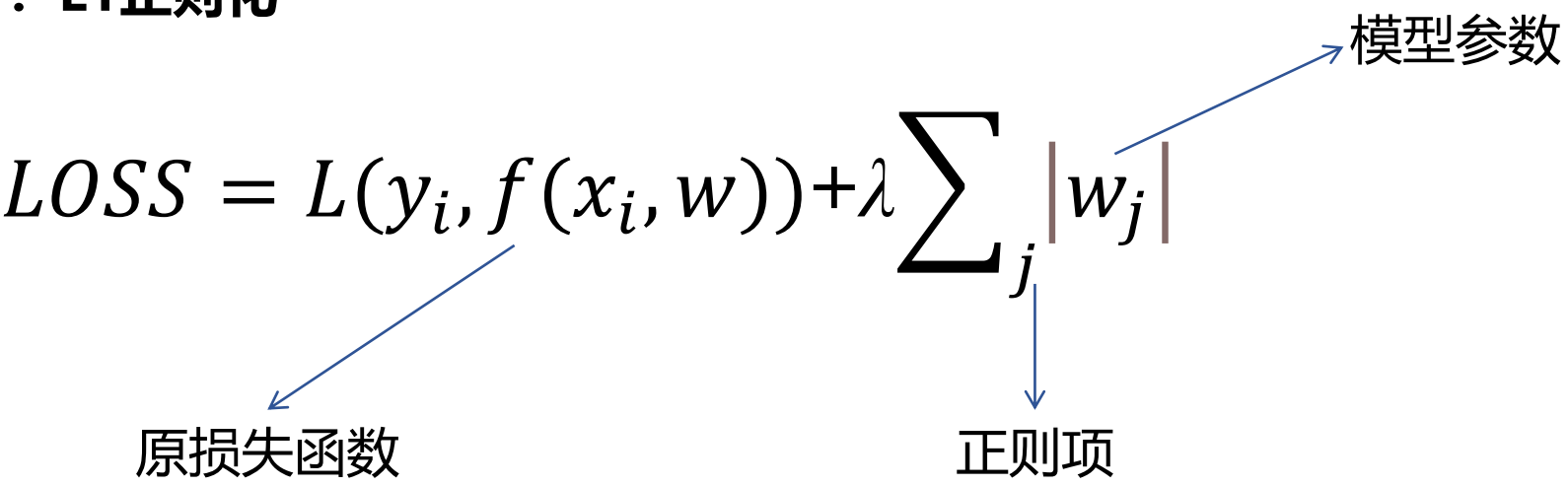
$$\|\boldsymbol{v}\|_2 = \sqrt{\sum_{n=1}^{N} v_n^2} = \sqrt{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}}.$$

ℓ_∞范数 ℓ_∞范数为向量的各个元素的最大绝对值，

$$\|\boldsymbol{v}\|_\infty = \max\{v_1, v_2, \cdots, v_N\}.$$

正则化（regularization）

- **例子：L1正则化**



- 更小的模型参数，代表模型更简单，更不容易出现过拟合
- 模型越复杂，正则项越大，损失则越大
- 有助于选择对预测目标具有重要影响的特征