Outline of the lecture

Many machine learning problems can be cast as optimization problems. This lecture introduces optimization. The objective is for you to learn:

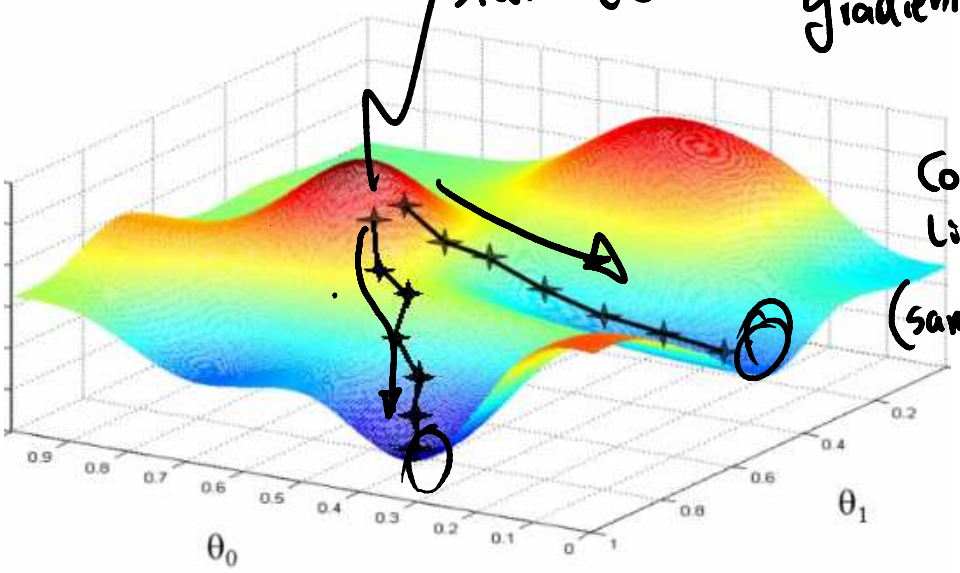
- ❑ The definitions of gradient and Hessian.
- ❑ The gradient descent algorithm.
- ❑ Newton's algorithm.
- ❑ The stochastic gradient descent algorithm for online learning.
- ❑ How to apply all these algorithms to linear regression.

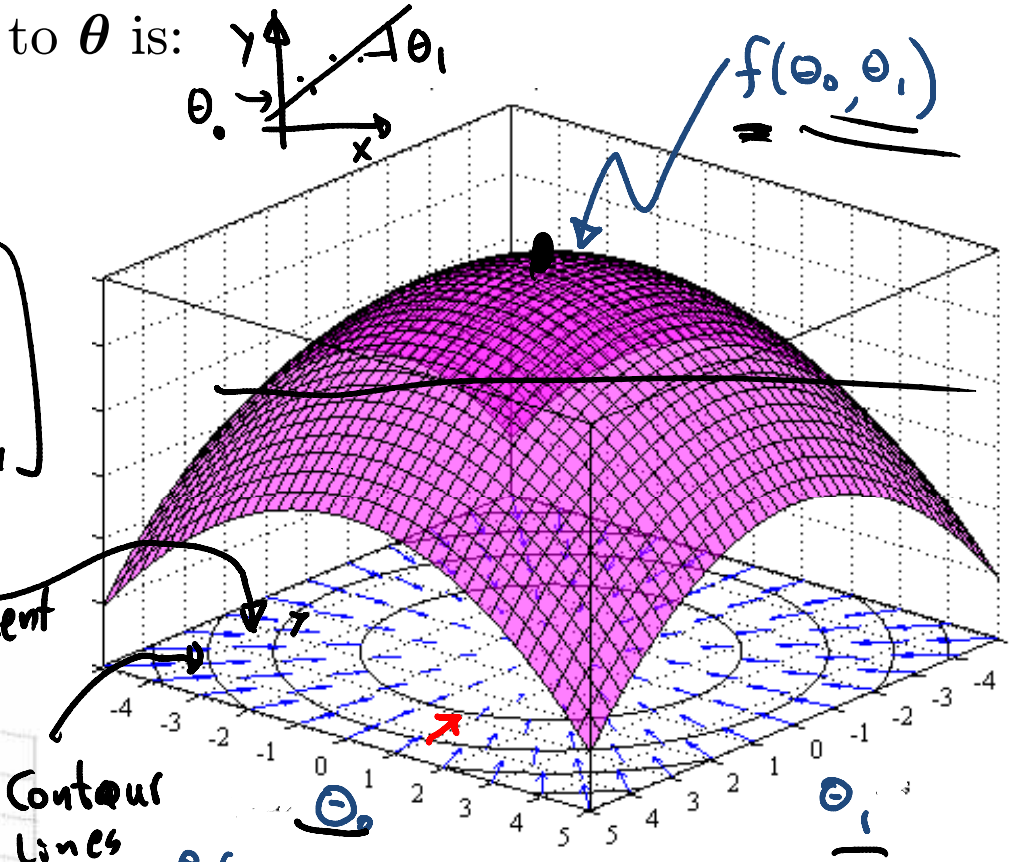
Gradient vector

Let $\boldsymbol{\theta}$ be an d-dimensional vector and $f(\boldsymbol{\theta})$ a scalar-valued function. The gradient vector of $f(\cdot)$ with respect to $\boldsymbol{\theta}$ is:

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$

$\nabla_{\theta} f(\theta_0, \theta_1) = \begin{bmatrix} \partial f/\partial \theta_0 \\ \partial f/\partial \theta_1 \end{bmatrix}$





e.g. $f(\theta_0, \theta_1) = -\sum_{i=1}^{n} (y_i - x_i \theta)^2$

$\theta = (\theta_0, \theta_1) \in \mathbb{R}^2$

Hessian matrix

The **Hessian** matrix of $f(\cdot)$ with respect to $\boldsymbol{\theta}$, written $\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ or simply as $\mathbf{H}$, is the $d \times d$ matrix of partial derivatives,

$$\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1^2} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d^2} \end{bmatrix}$$

In **offline** learning, we have a **batch** of data $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$. We typically optimize cost functions of the form

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}, \mathbf{x}_{1:n}) = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{\theta}, \mathbf{x}_i)$$

The corresponding gradient is

$$g(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{x}_i)$$

For linear regression with training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$, we have have the quadratic cost

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = \frac{1}{n}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i\boldsymbol{\theta})^2$$

$f(\theta, x_i, y_i)$

Gradient vector and Hessian matrix

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = \underbrace{(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})} = \sum_{i=1}^{n}(y_i - \mathbf{x}_i\boldsymbol{\theta})^2$$

$$\nabla f(\theta) = \frac{\partial}{\partial \theta}\left(y^T y - 2y^T X\theta + \theta^T X^T X\theta\right)$$

$$= -2X^T y + 2X^T X\theta \quad = \quad \nabla f(\theta) = -2\sum_{i=1}^{n} x_i^T (y_i - x_i\theta)$$

$$D^2 f(\theta) = 0 + 2X^T X$$

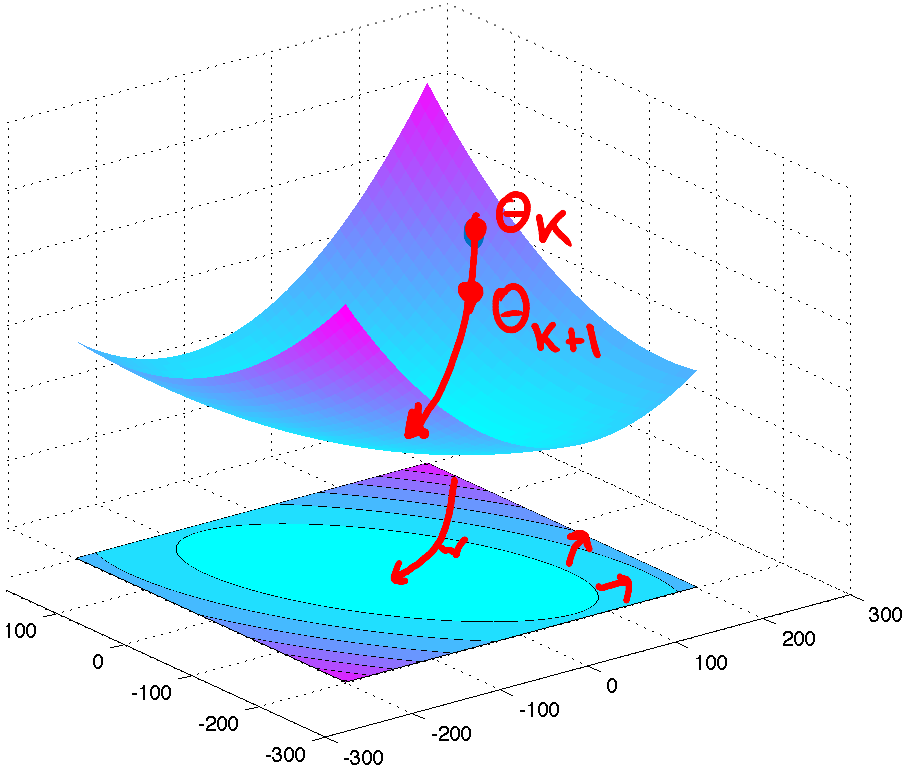
$$= 2X^T X$$

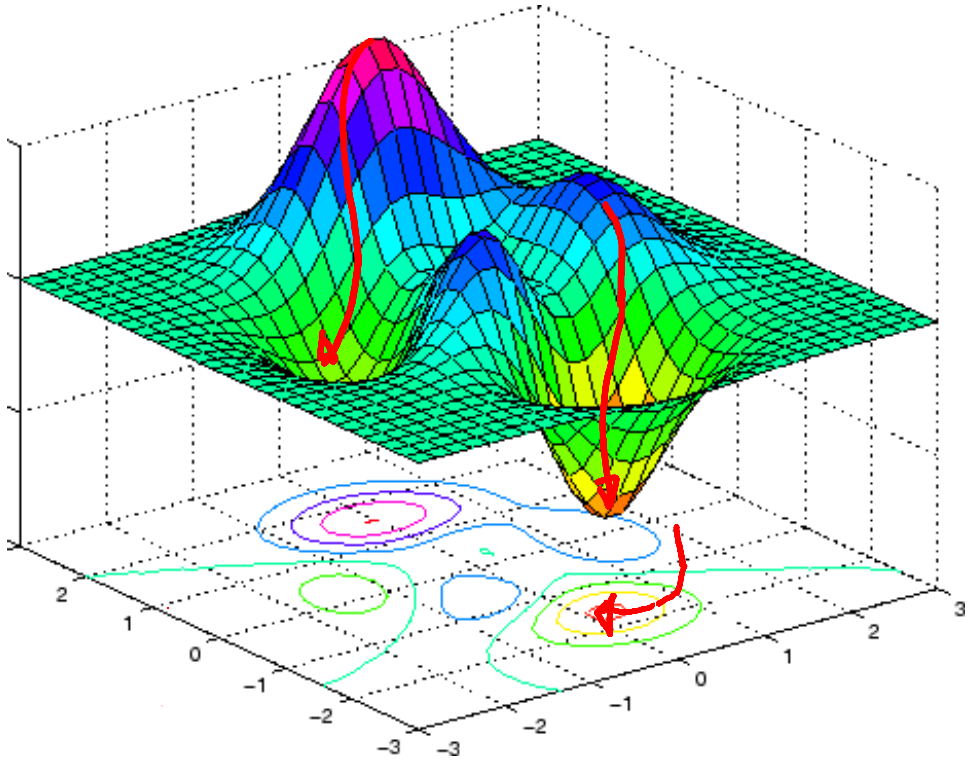
Steepest gradient descent algorithm

One of the simplest optimization algorithms is called **gradient descent** or **steepest descent**. This can be written as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k = \boldsymbol{\theta}_k - \eta_k \nabla f(\boldsymbol{\theta}_k)$$

where k indexes steps of the algorithm, $\mathbf{g}_k = \mathbf{g}(\boldsymbol{\theta}_k)$ is the gradient at step k, and $\eta_k > 0$ is called the **learning rate** or **step size**.





Steepest gradient descent algorithm for least squares

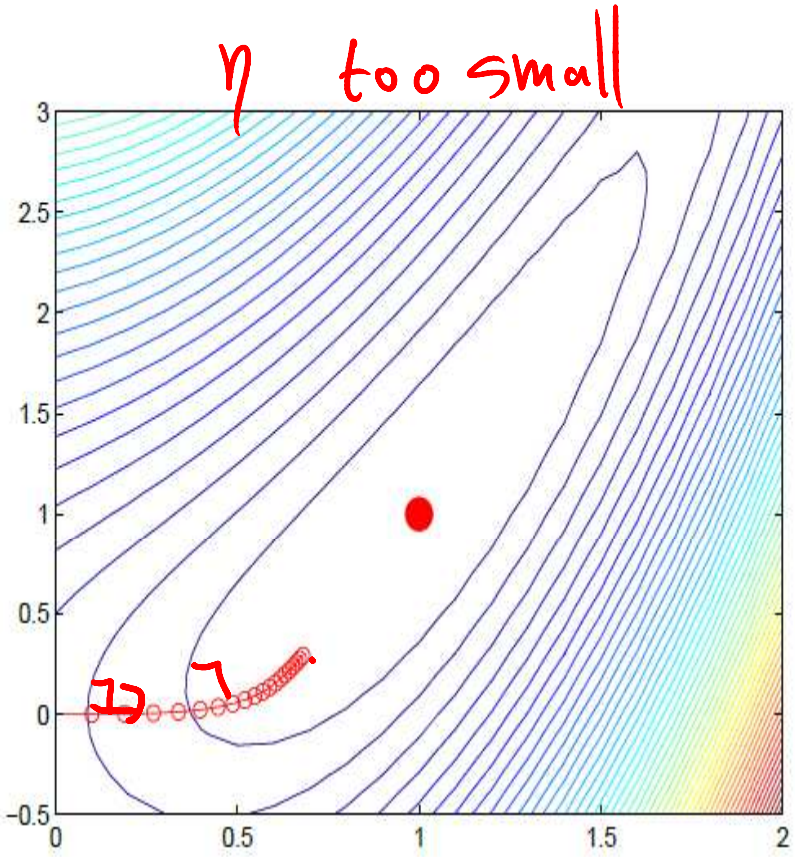
$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i\boldsymbol{\theta})^2$$

$$\nabla f(\theta) = -2X^T y + 2X^T X\theta$$

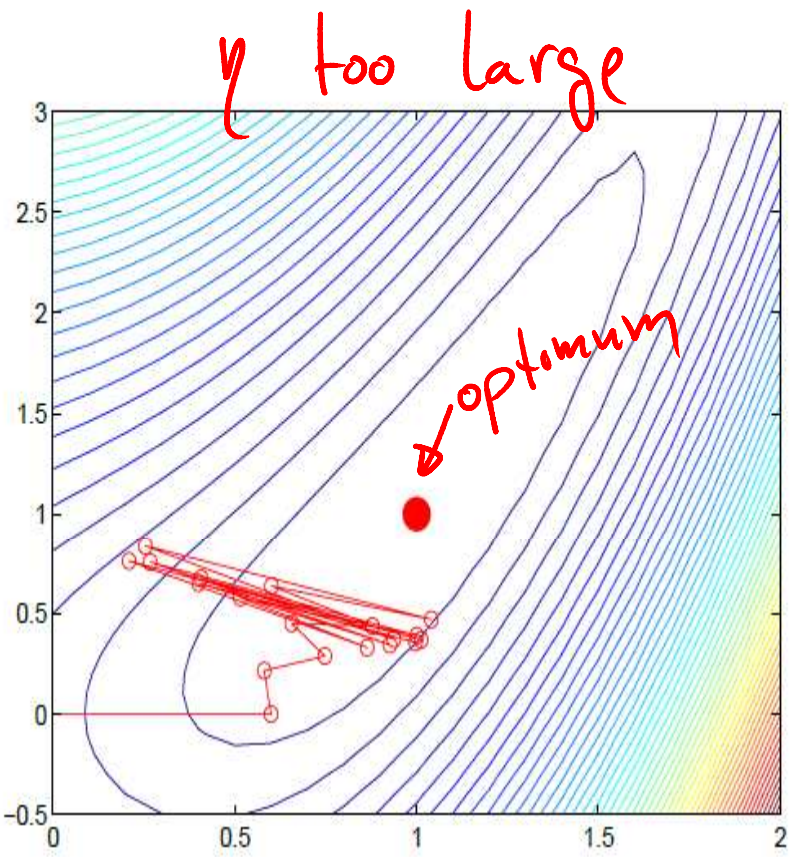
$$\theta_{k+1} = \theta_k - \eta\left[-2X^T y + 2X^T X\theta_k\right]$$

$$\theta_{k+1} = \theta_k - \eta\left[-2\sum_{i=1}^{n} x_i^T\underbrace{(y_i - x_i\theta_k)}_{\text{Fit}}\right]$$

How to choose the step size ?



$\eta = 0.1$



$\eta = 0.6$

$$\theta_{k+1} = \theta_k - \eta \nabla f(\theta_k)$$

Newton's algorithm

The most basic second-order optimization algorithm is **Newton's algorithm**, which consists of updates of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_K^{-1}\mathbf{g}_k$$

This algorithm is derived by making a second-order Taylor series approximation of $f(\boldsymbol{\theta})$ around $\boldsymbol{\theta}_k$:

$$f_{quad}(\boldsymbol{\theta}) = f(\boldsymbol{\theta}_k) + \mathbf{g}_k^T(\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T\mathbf{H}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

old value, $\Delta\theta$, new value

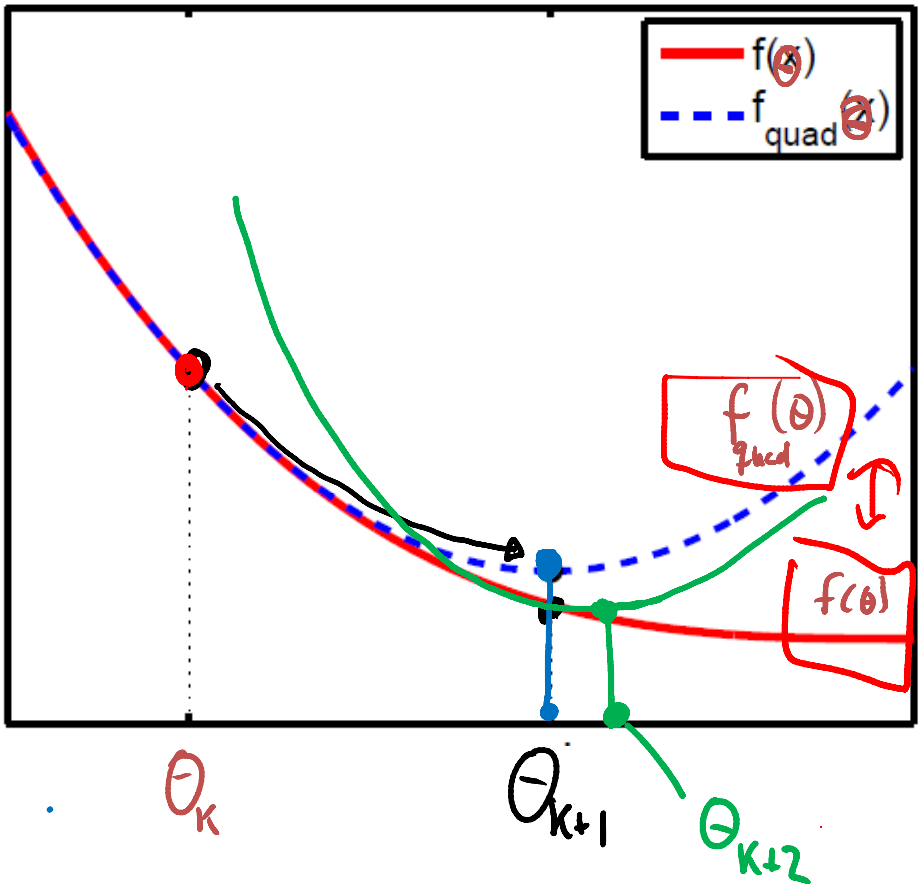
differentiating and equating to zero to solve for $\boldsymbol{\theta}_{k+1}$.

$$\nabla f_{quad}(\theta) = 0 + g_k + H_k(\theta - \theta_k) = 0$$

$$-g_k = H_k(\theta - \theta_k)$$

$$\theta = \theta_k - H_k^{-1} g_k$$

Newton's as bound optimization



Newton's algorithm for linear regression

$$f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) = \sum_{i=1}^{n}(y_i - \mathbf{x}_i\boldsymbol{\theta})^2$$

$g = \nabla f(\theta) = -2X^T y + 2X^T X\theta$

$H = \nabla^2 f(\theta) = 2X^T X$

$\theta_{k+1} = \theta_k - H_k^{-1} g_k$

$= \theta_k - [2X^T X]^{-1}[-2X^T y + 2X^T X\theta_k]$

$= \theta_k + (X^T X)^{-1} X^T y - (X^T X)^{-1}(X^T X)\theta_k$

Advanced: Newton CG algorithm

Rather than computing $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ directly, we can solve the linear system of equations $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$.

One efficient and popular way to do this, especially if $\mathbf{H}$ is sparse, is to use a conjugate gradient method to solve the linear system.

---

1 Initialize $\boldsymbol{\theta}_0$
2 **for** $k = 1, 2, \ldots$ *until convergence* **do**
3 Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$
4 Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$
5 Solve $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$
6 Use line search to find stepsize η_k along $\mathbf{d}_k$
7 $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k\mathbf{d}_k$

Estimating the mean recursively

average = $\hat{\theta}_N = \frac{1}{N} \sum_{i=1}^{N} x_i$ BATCH

$$\hat{\theta}_N = \frac{1}{N} x_N + \frac{1}{N} \frac{N-1}{N-1} \sum_{i=1}^{N-1} x_i$$

$$= \frac{1}{N} x_N + \frac{1}{N-1} \left(\frac{N-1}{N}\right) \sum_{i=1}^{N-1} x_i = \frac{1}{N} x_N + \left(\frac{N-1}{N}\right) \hat{\theta}_{N-1}$$

$$\hat{\theta}_N = \left(1 - \frac{1}{N}\right) \hat{\theta}_{N-1} + \frac{1}{N} x_N$$ ONLINE

Online learning
aka stochastic gradient descent

$x^{(i)} \sim P(x)$

$$J(\theta) = \underbrace{\int J(\theta, x) P(x) dx}_{\text{expected cost}} \approx \underbrace{\frac{1}{N} \sum_{i=1}^{N} J(\theta, x_i)}_{\text{empirical cost}}$$

data

risk

$$\nabla J(\theta) = \int \nabla J(\theta, x) P(x) dx$$

$$\theta_{k+1} = \theta_k - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla J(\theta_k, x_i) \overset{\text{Let } N=1}{=} \theta_k - \eta \nabla J(\theta_k, x_k)$$

$$= \theta_k - \eta \underbrace{\nabla J(\theta_k)}_{\text{true grad}} + \eta \underbrace{\left[\nabla J(\theta_k) - \nabla J(\theta_k, x_k) \right]}_{\text{noise}}$$

Online learning
aka stochastic gradient descent

Batch

$$\theta_{k+1} = \theta_k + \eta \sum_{i=1}^{n} x_i^T (y_i - x_i \theta_k)$$

(n data points)

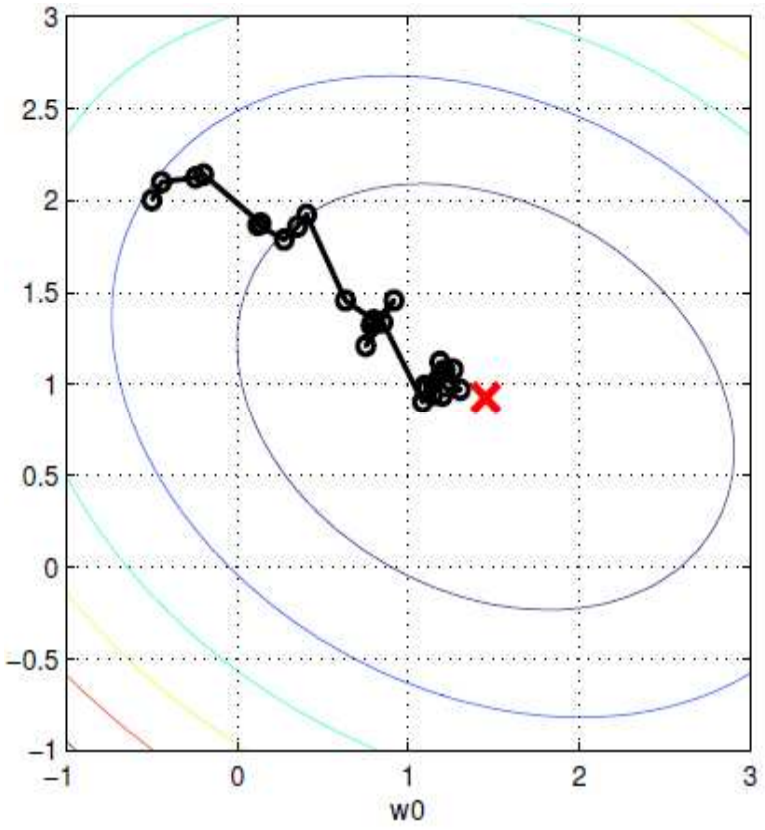
Online

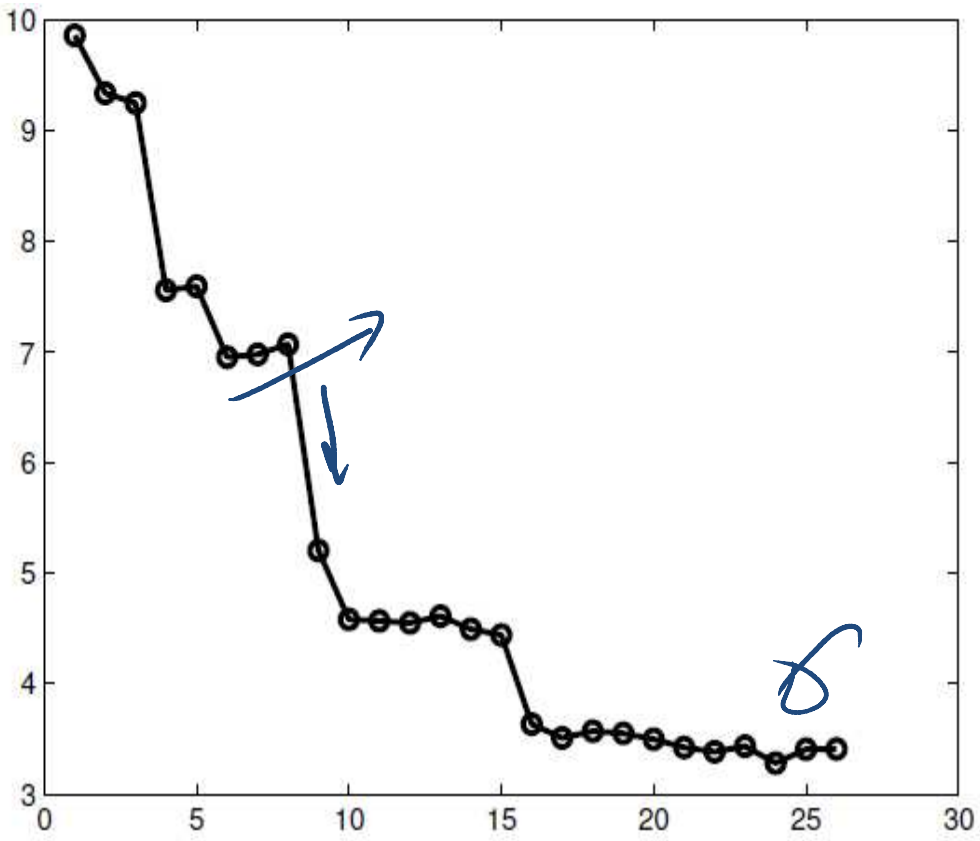
$$\theta_{k+1} = \theta_k + \eta\, x_k^T (y_k - x_k \theta_k)$$

mini-batch

$$\theta_{k+1} = \theta_k + \eta \sum_{j=1}^{20} x_j^T (y_j - x_j \theta_k)$$

The online learning algorithm





Stochastic gradient descent

SGD can also be used for offline learning, by repeatedly cycling through the data; each such pass over the whole dataset is called an **epoch**. This is useful if we have **massive datasets** that will not fit in main memory. In this offline case, it is often better to compute the gradient of a **mini-batch** of B data cases. If $B = 1$, this is standard SGD, and if $B = N$, this is standard steepest descent. Typically $B \sim 100$ is used.

Intuitively, one can get a fairly good estimate of the gradient by looking at just a few examples. Carefully evaluating precise gradients using large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space.

SGD is often less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise". Consequently it is quite popular in the machine learning community for fitting models such as neural networks and deep belief networks with non-convex objectives.