Optimization

Nando de Freitas

UNIVERSITY OF OXFORD

Outline of the lecture

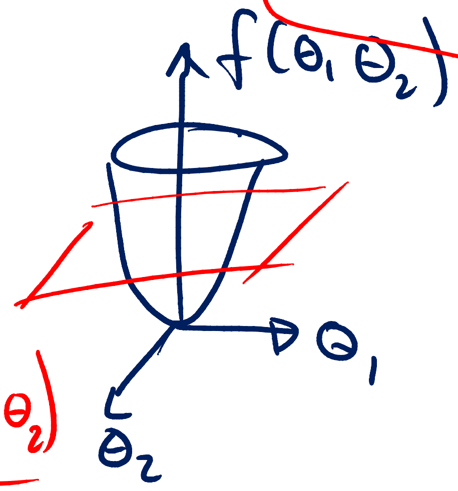
Many machine learning problems can be cast as optimization problems. This lecture introduces optimization. The objective is for you to learn:

- ❑ The definitions of gradient and Hessian.
- ❑ The gradient descent algorithm.
- ❑ Newton's algorithm.
- ❑ Stochastic gradient descent (SGD) for online learning.
- ❑ Popular variants, such as AdaGrad and Asynchronous SGD.
- ❑ Improvements such as momentum and Polyak averaging.
- ❑ How to apply all these algorithms to linear regression.

Calculus background: Partial derivatives and gradient

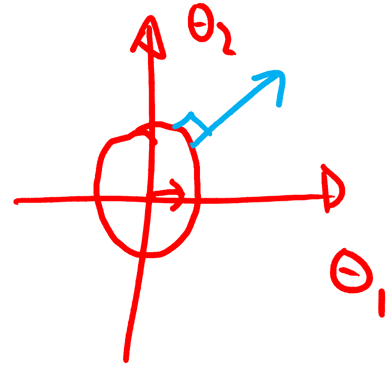
∇

$$f(\theta) = f(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$$



$$\frac{\partial f(\theta_1, \theta_2)}{\partial \theta_1} = \lim_{\Delta\theta_1 \to 0} \frac{f(\theta_1 + \Delta\theta_1, \theta_2) - f(\theta_1, \theta_2)}{\Delta\theta_1}$$

$$\frac{\partial f(\theta)}{\partial \theta_1} = \boxed{2\theta_1} \qquad \frac{\partial f(\theta)}{\partial \theta_2} = 2\theta_2$$



$$\nabla J(\theta) = \begin{bmatrix} 2\theta_1 \\ 2\theta_2 \end{bmatrix}$$

$$(\theta_1 = 1, \theta_2 = 1)$$

$$\nabla J(\theta_1, \theta_2) = (2, 2)$$

Necessary calculus background: Hessian

$$\frac{\partial}{\partial \theta_1}\left(\frac{\partial f(\theta)}{\partial \theta_1}\right) = \frac{\partial^2 f(\theta)}{\partial \theta_1^2} = 2$$

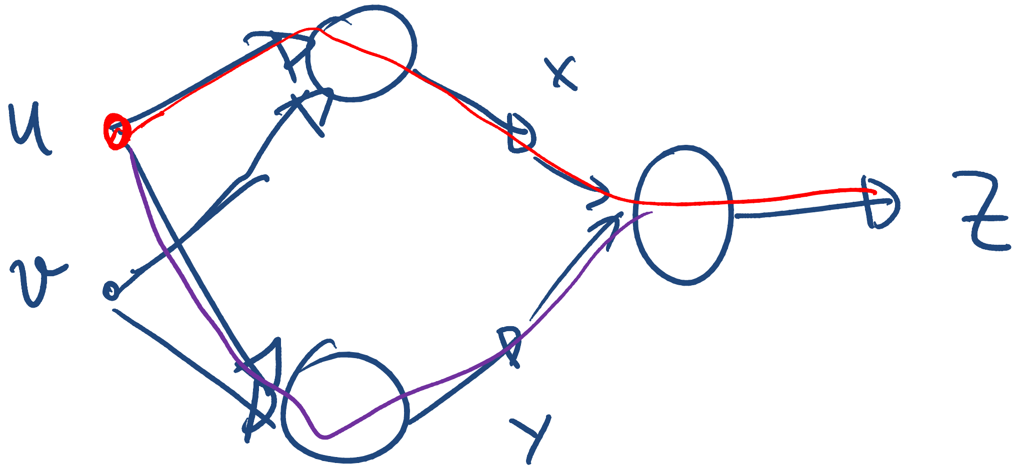
$$\frac{\partial^2 f(\theta)}{\partial \theta_2^2} = 2$$

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\frac{\partial^2 f(\theta)}{\partial \theta_1 \partial \theta_2} = \frac{\partial^2 f(\theta)}{\partial \theta_2 \partial \theta_1} = 0$$

Necessary calculus background: Chain rule

$$z = f(x(u,v), y(u,v))$$



$$\frac{\partial z}{\partial u} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial u}$$

Necessary calculus background: Linear regression

$$y_i = \theta_0 + \theta_1 x_i \qquad i = 1, 2, \ldots, n$$

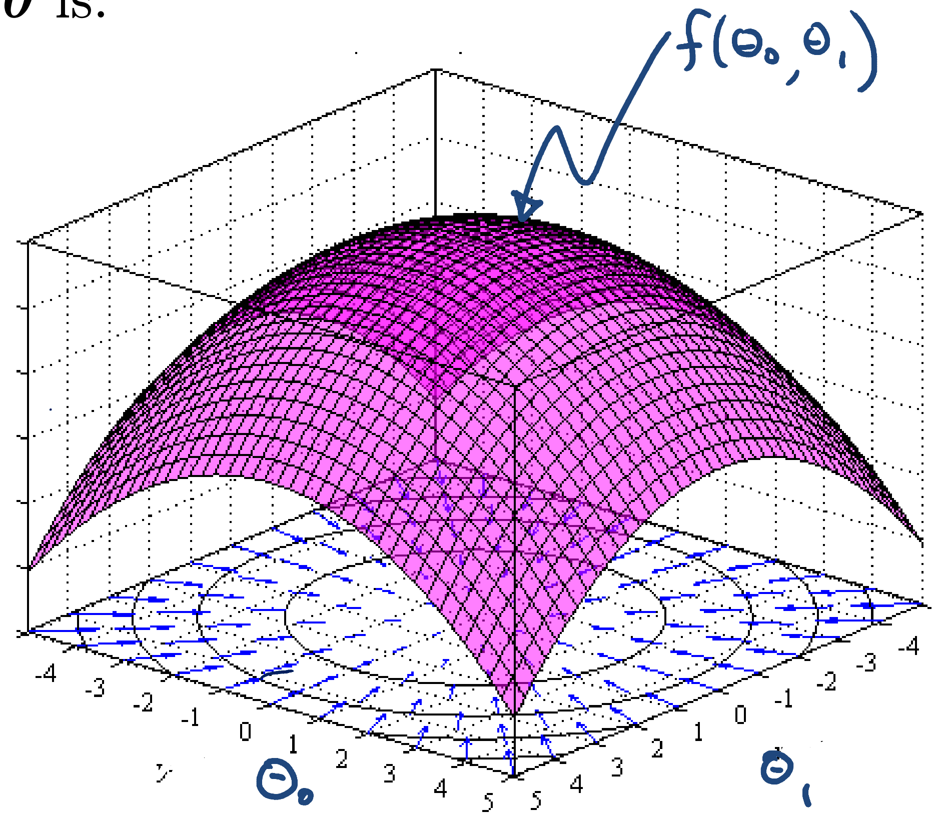
$$\boxed{J(\underline{\theta}) = \sum_{i=1}^{n} \left(y_i - \theta_0 - \theta_1 x_i\right)^2 + \delta^2 \theta_1^2}$$

$$\nabla J = \begin{bmatrix} \sum_{i=1}^{n} 2\left(y_i - \theta_0 - \theta_1 x_i\right)(-1) \\ \sum_{i=1}^{n} 2\left(y_i - \theta_0 - \theta_1 x_i\right)(-x_i) + 2\delta^2 \theta_1 \end{bmatrix}$$

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:

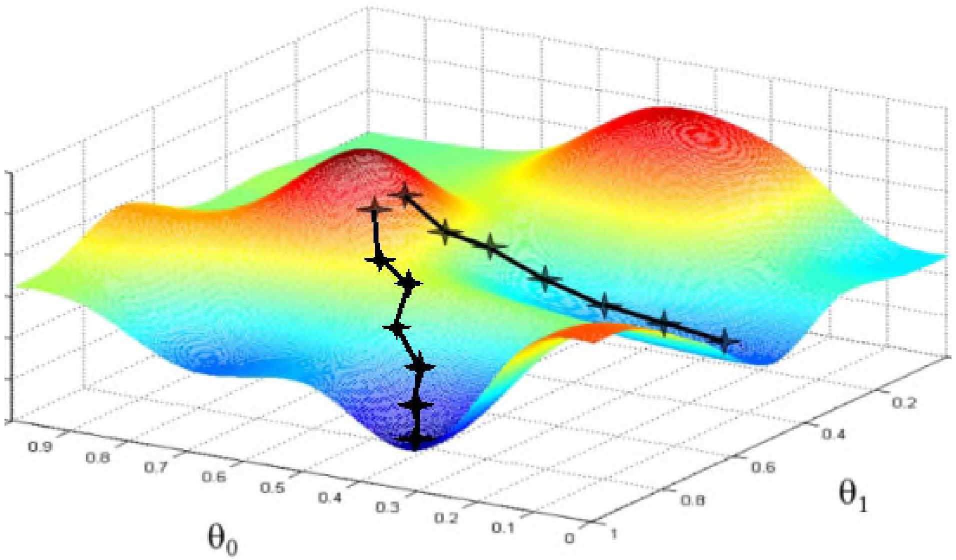
$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$



e.g.

$$f(\theta_0, \theta_1) = -\sum_{i=1}^{n} \underline{(y_i - \underline{x}_i \underline{\theta})^2}$$

$$\underline{\theta} = (\theta_0, \theta_1)$$



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

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

The corresponding gradient is

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

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

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

Gradient vector and Hessian matrix

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

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

$$= -2X^T y + 2X^T X \theta$$

$$\nabla f(\theta) = -2\sum_{i=1}^{n} \underline{x}_i^T (y_i - \underline{x}_i \underline{\theta})$$

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

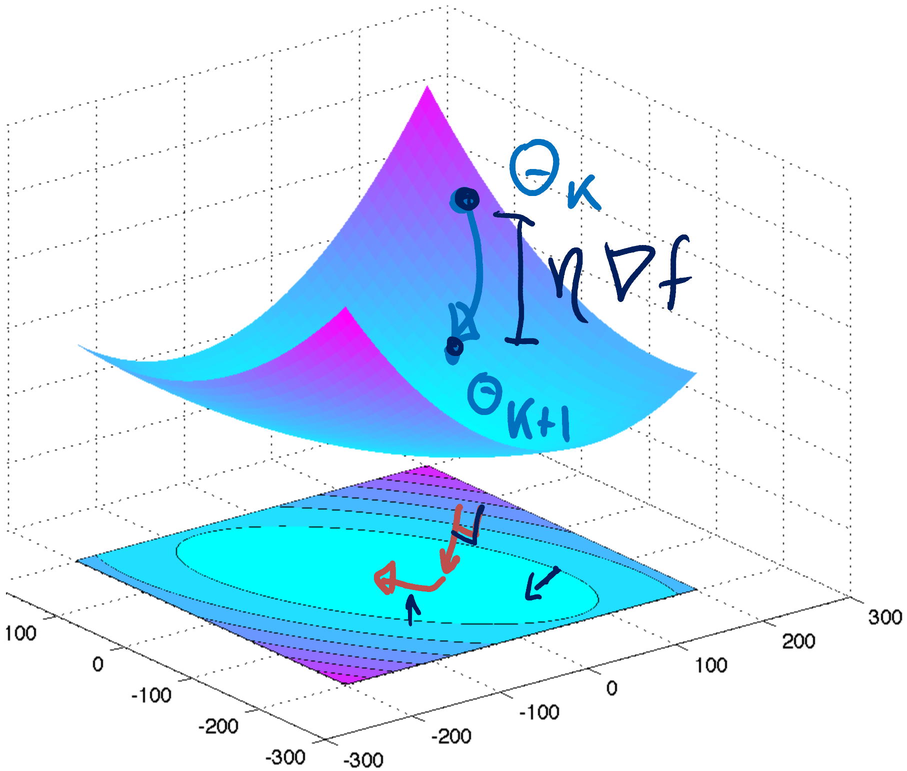
$$= 2\underset{d\times n}{X^T}\underset{n\times d}{X}$$

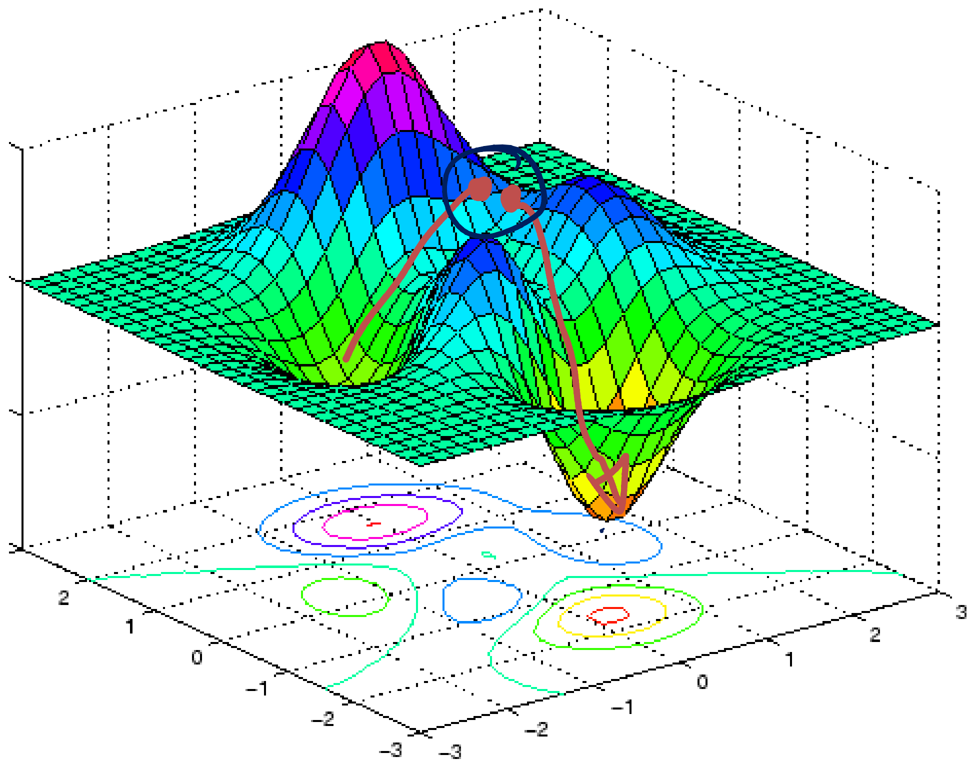
Steepest gradient descent algorithm

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

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

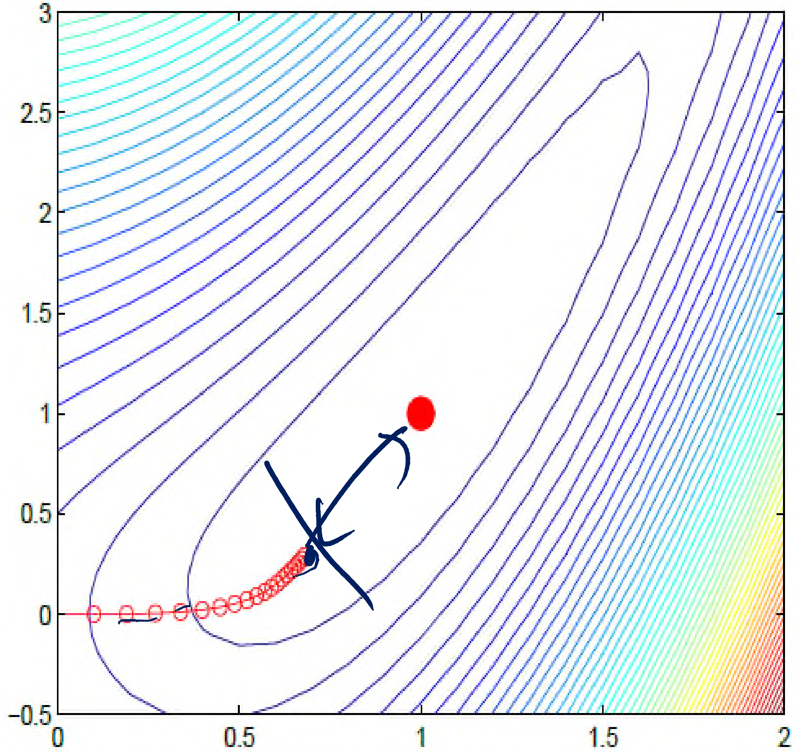
$$\nabla f(\theta) = -2X^T y + 2X^T X \theta$$

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

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

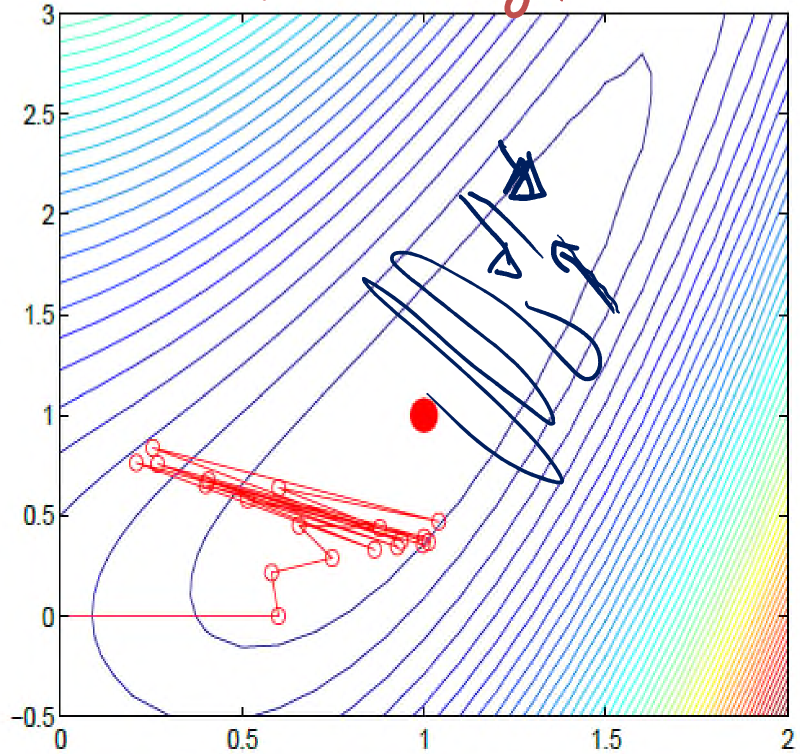
How to choose the step size ?

too small



$\eta = 0.1$

too large



$\eta = 0.6$

Newton's algorithm

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

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

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

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

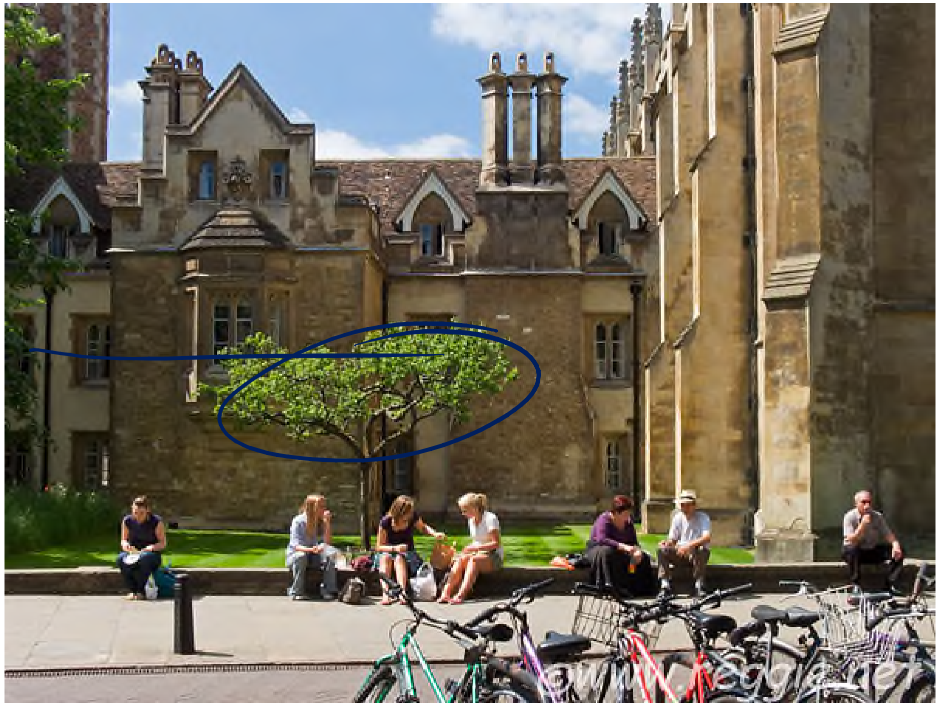
differentiating and equating to zero to solve for $\boldsymbol{\theta}_{k+1}$.

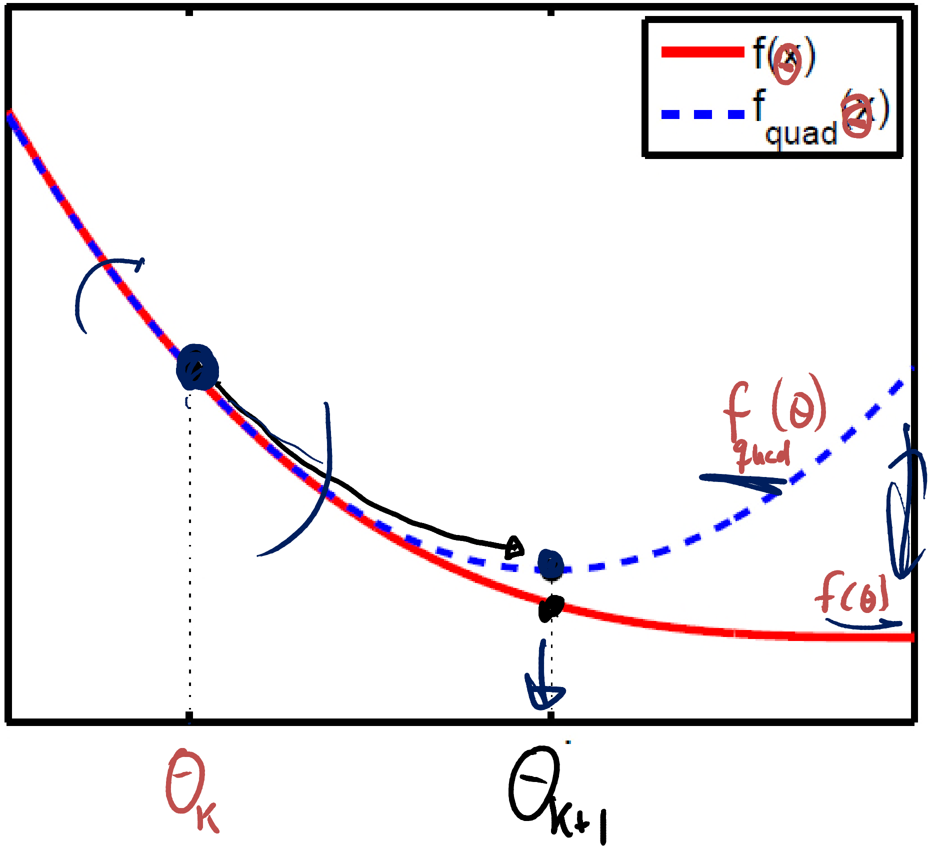
$$\nabla f_{quad}(\theta) = \cancel{0} + g_k + H_k(\theta - \theta_k) = 0$$

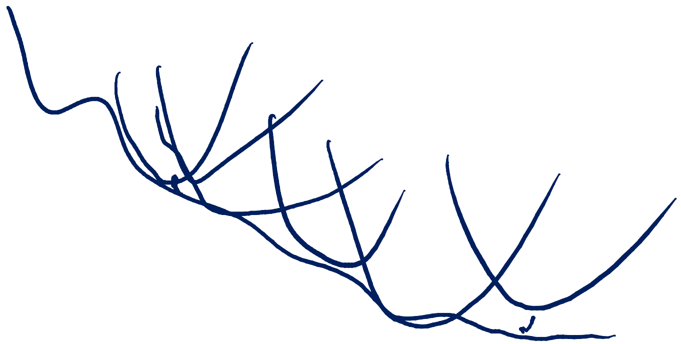
$$-g_k = H_k(\theta - \theta_k)$$

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

Newton as bound optimization





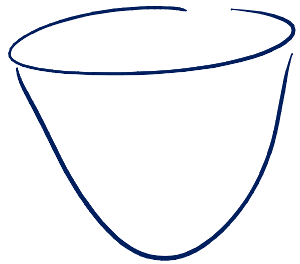


Newton's algorithm for linear regression

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

$g = \nabla f(\theta) = -2X^Ty + 2X^TX\theta$ ✓

$H = \nabla^2 f(\theta) = 2X^TX$ ✓



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

$$= \theta_k - (2X^TX)^{-1}\left[-2X^Ty + 2X^TX\theta_k\right]$$

$$= \cancel{\theta_k} + (X^TX)^{-1}X^Ty - \cancel{(X^TX)^{-1}X^TX\theta_k}$$

$$= (X^TX)^{-1}X^Ty$$

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.

Initialize $\boldsymbol{\theta}_0$

for $k = 1, 2, \ldots$ *until convergence* **do**

- Evaluate $\mathbf{g}_k = \nabla f(\boldsymbol{\theta}_k)$
- Evaluate $\mathbf{H}_k = \nabla^2 f(\boldsymbol{\theta}_k)$
- Solve $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ for $\mathbf{d}_k$
- Use line search to find stepsize η_k along $\mathbf{d}_k$
- $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k\mathbf{d}_k$

SGD

epoch

$$\nabla_\theta f(\theta) = \underbrace{\int \nabla_\theta f(x,\theta)\, p(x)\, dx}_{\mathbb{E}[\nabla f(x,\theta)]} \overset{\text{want}}{=} 0$$

$$\eta \times \mathbb{E}[\nabla f(x,\theta)] = 0 \times \eta = 0$$

$$\Theta + \eta\, \mathbb{E}[\nabla f(x,\theta)] = \Theta$$

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

$$\Theta_{k+1} = \theta_k - \eta\, \mathbb{E}[\nabla f(x,\theta)]$$

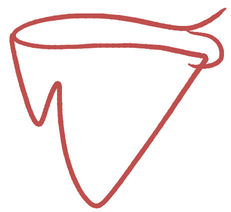
$$\approx \theta_k - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla f(x^{(i)}, \theta_k)$$

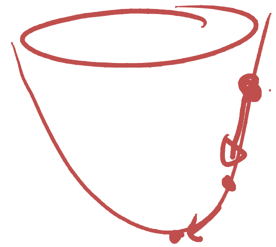
$$\approx \theta_k - \eta \nabla f(x^{(k)}, \theta_k)$$

$\alpha\ 1/k$

$$\theta_{k+1} = \cancel{\theta_k - \eta\, \mathbb{E}[\nabla f]} + \eta_k \underbrace{\left[\mathbb{E}[\nabla f] - \nabla f(x_k, \theta_k)\right]}_{\text{noise}}$$

Online learning with mini-batches





Batch

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

(n data points)

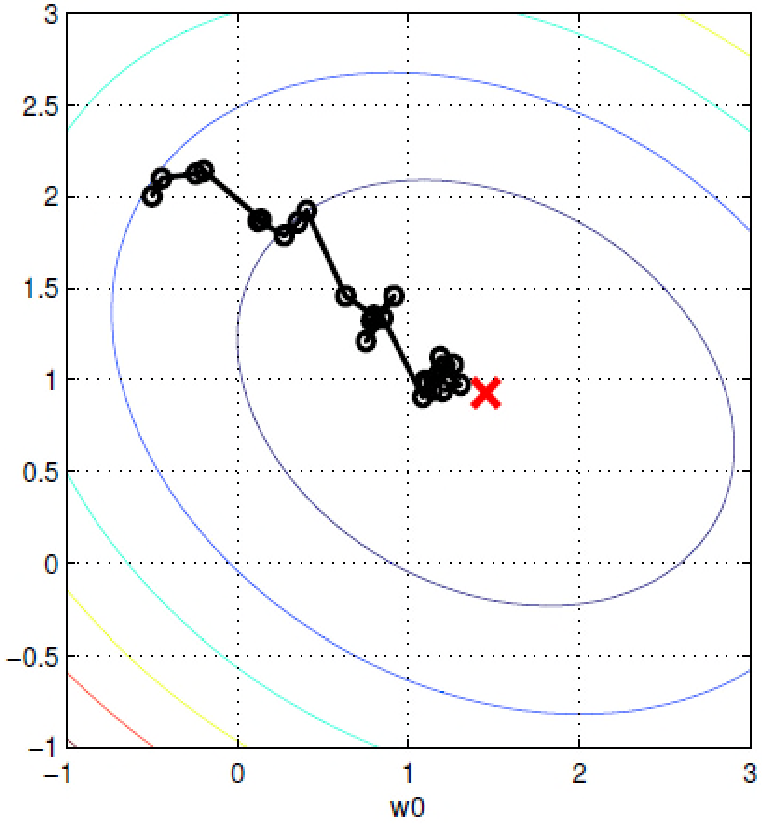
Online

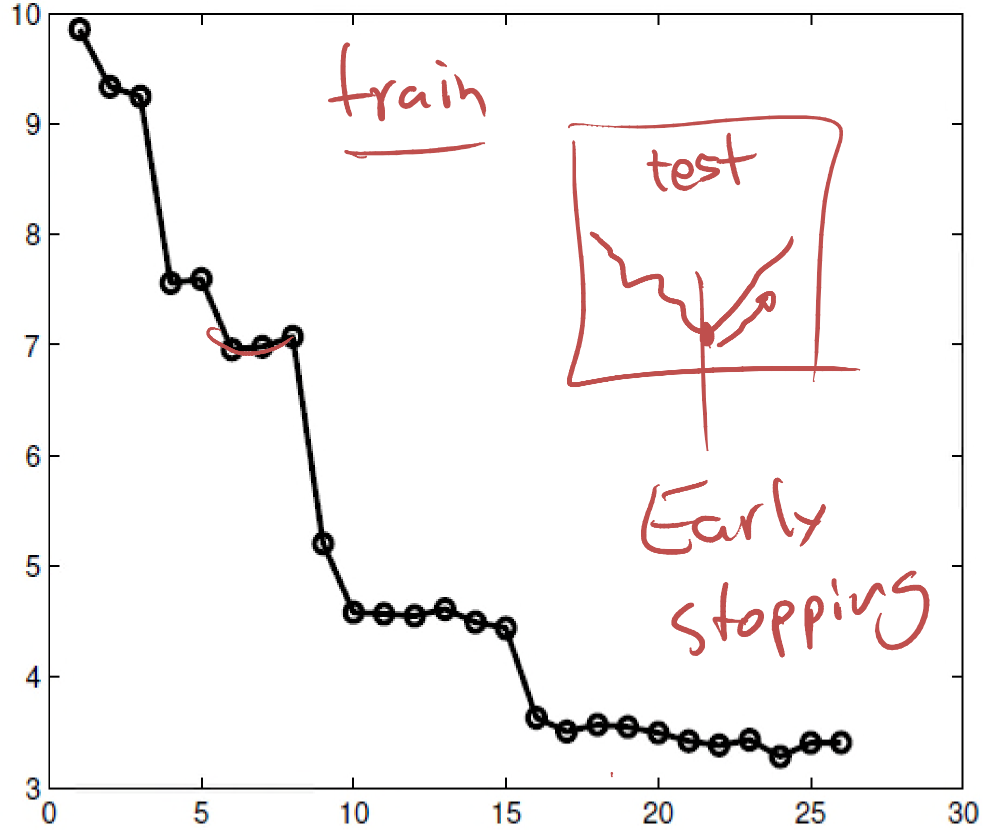
$$\theta_{k+1} = \theta_k + \eta\, x_k^T (y_k - x_k \theta_k)$$

mini-batch

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

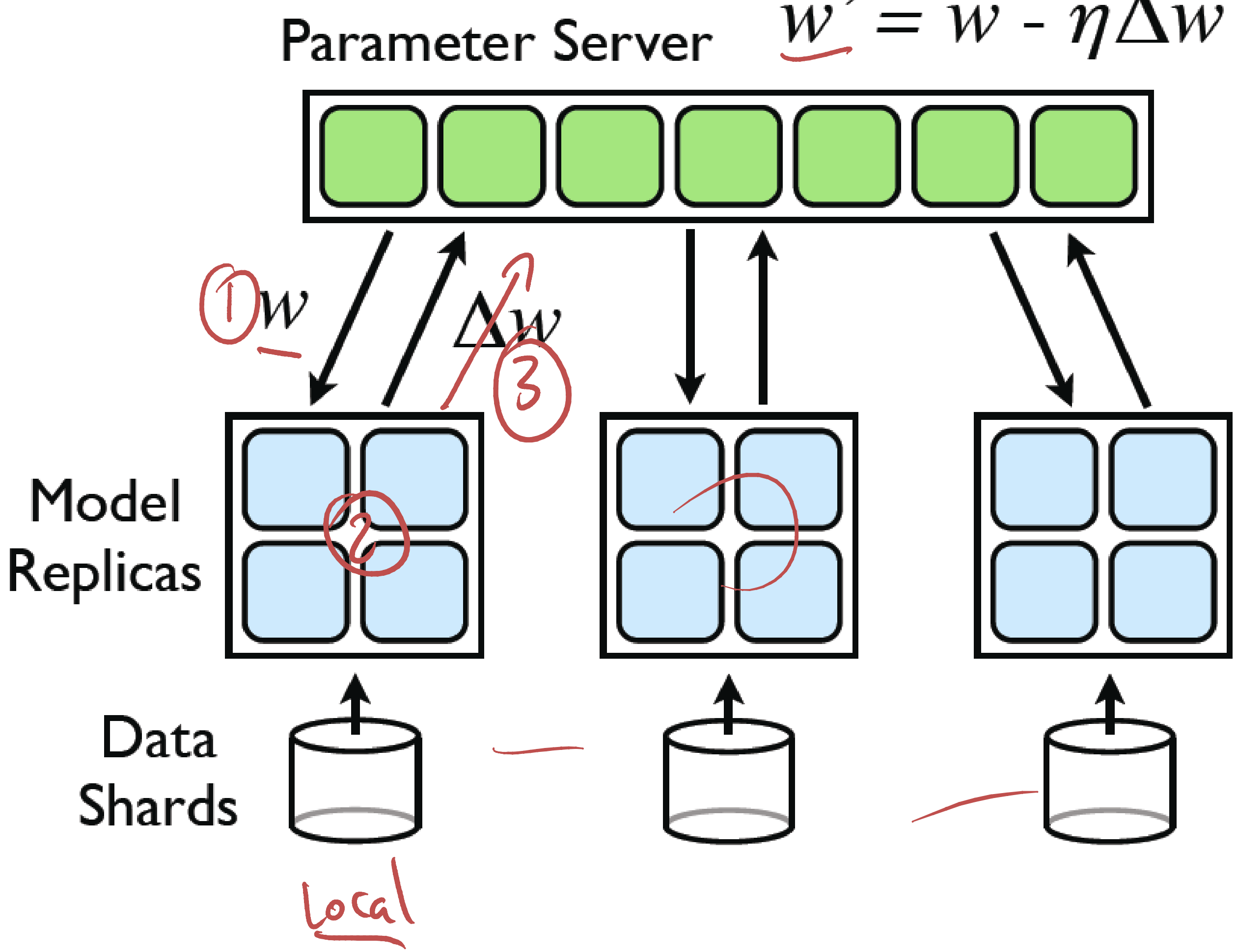
The online learning algorithm

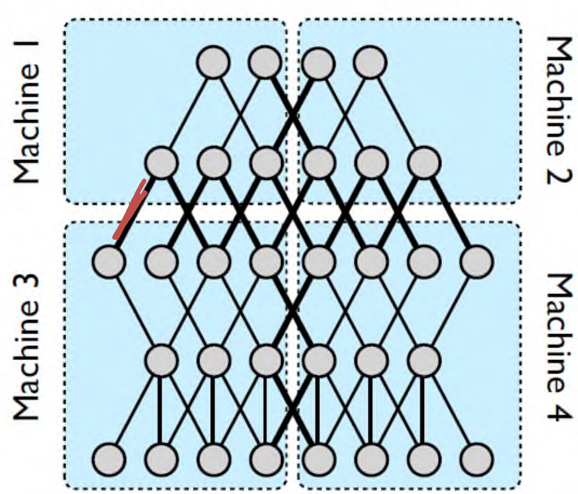




Downpour – Asynchronous SGD

$w' = w - \eta \Delta w$





[Jeff Dean et al.]

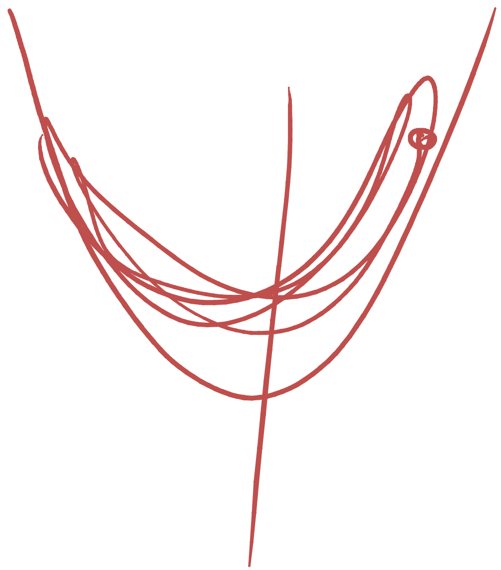
Polyak averaging

Polyak averaging (see papers of Mark Schmidt / Francis Bach)

$$\frac{1}{t+1}\sum_{i}^{t+1} w^{i} \rightarrow$$

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \gamma^{(t)} \nabla L(W^{(t)}, v^{(t)}) \\ \overline{W}^{(t+1)} &= \overline{W}^{(t)} - \frac{1}{t}(\overline{W}^{(t)} - W^{(t)}), \end{aligned}$$

See also predictive variance reduction (Tong Zhang, NIPS 2013)



Momentum

$$\underset{\Delta\theta_{t+1}}{\left[\theta_{t+1} - \theta_t\right]} = \alpha\underbrace{(\theta_t - \theta_{t-1})}_{\Delta\theta_t} + (1-\alpha)\left[-\eta\nabla J(\theta)\right]$$

Adagrad: Put more weight on rare features

y_t	$\phi_{t,1}$	$\phi_{t,2}$	$\phi_{t,3}$
1	1	0	0
-1	.5	0	1
1	-.5	1	0
-1	0	0	0
1	.5	0	0
-1	1	0	0
1	-1	1	0
-1	-.5	0	1

Text data:

The most unsung birthday in American business and technological history this year may be the 50th anniversary of the Xerox 914 photocopier.[a]

[a] *The Atlantic*, July/August 2010.

1. Frequent, irrelevant
2. Infrequent, predictive
3. Infrequent, predictive

gradient

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$$

[Duchi et al.]

Other useful optimization

- BFGS and limited-BFGS (Take e.g. Nick Trefethen's course)
- Nesterov's method (See Nesterov's book)
- Proximal methods (See Bertsekas book)
- Natural gradient (Yoshua Bengio et al – ICLR)
- Hessian-vector updates and automatic differentiation (Bishop's book and Nocedal & Wright)
- Convex optimization / constrained optimization (See Boy's book)

Next lecture

In the next lecture, we apply these ideas to learn a neural network with a single neuron (logistic regression).