- Back to some examples of optimization problems in Machine Learning …

Supervised Learning

- Classification
  - Is there a cat on the picture?

Yes / No

Supervised Learning

- Classification
  - Is there a cat on the picture?

Yes

- Classification
  - Is there a cat on the picture?

Yes

- Classification
  - Is there a cat on the picture?

No

Supervised Learning

- Labelled data / training sets

x^1  x^2  x^3 

Input or features

$y^1 = 1$ $y^2 = 1$ $y^3 = -1$

Output labels Target

Given a set of examples $\{(x^1, y^1), \ldots, (x^n, y^n)\}$ with x^i the features and y^i labels/targets

Supervised Learning

Given a set of examples $\{(x^1, y^1), \ldots, (x^n, y^n)\}$ with x^i the features and y^i labels/targets

Find a mapping $h : x \in X \to y \in \mathbb{R}$ that will assign the "correct" target to each input

Learning algorithm

New image (not in the training set)

$h\Big($  $\Big) = -1$

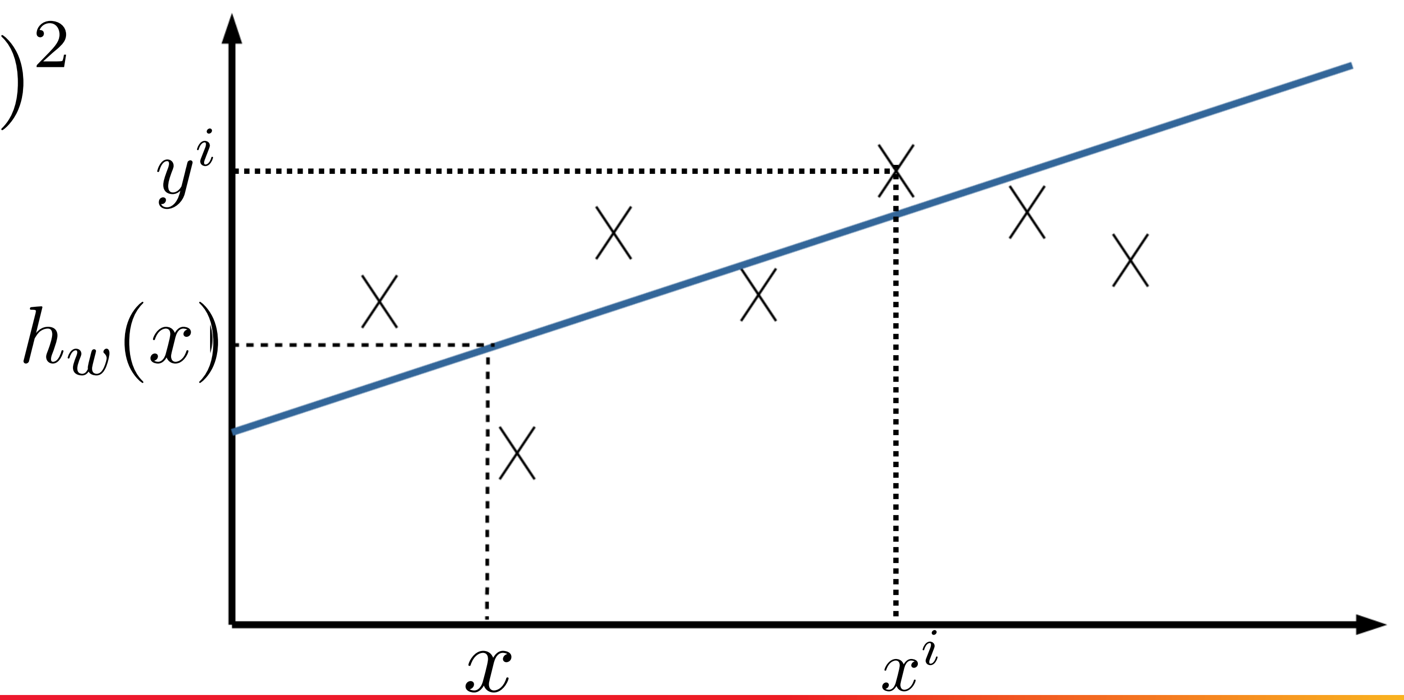
Example 1: Linear Regression

Hypothesis: linear model

$$h_w(x) = w_0 + w_1 x_1 + \ldots + w_{d-1} x_{d-1} \overset{x_0 = 1}{=} \langle w, x \rangle$$

Find $h_w(x)$ via solving the minimization problem

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (h_w(x^i) - y^i)^2$$



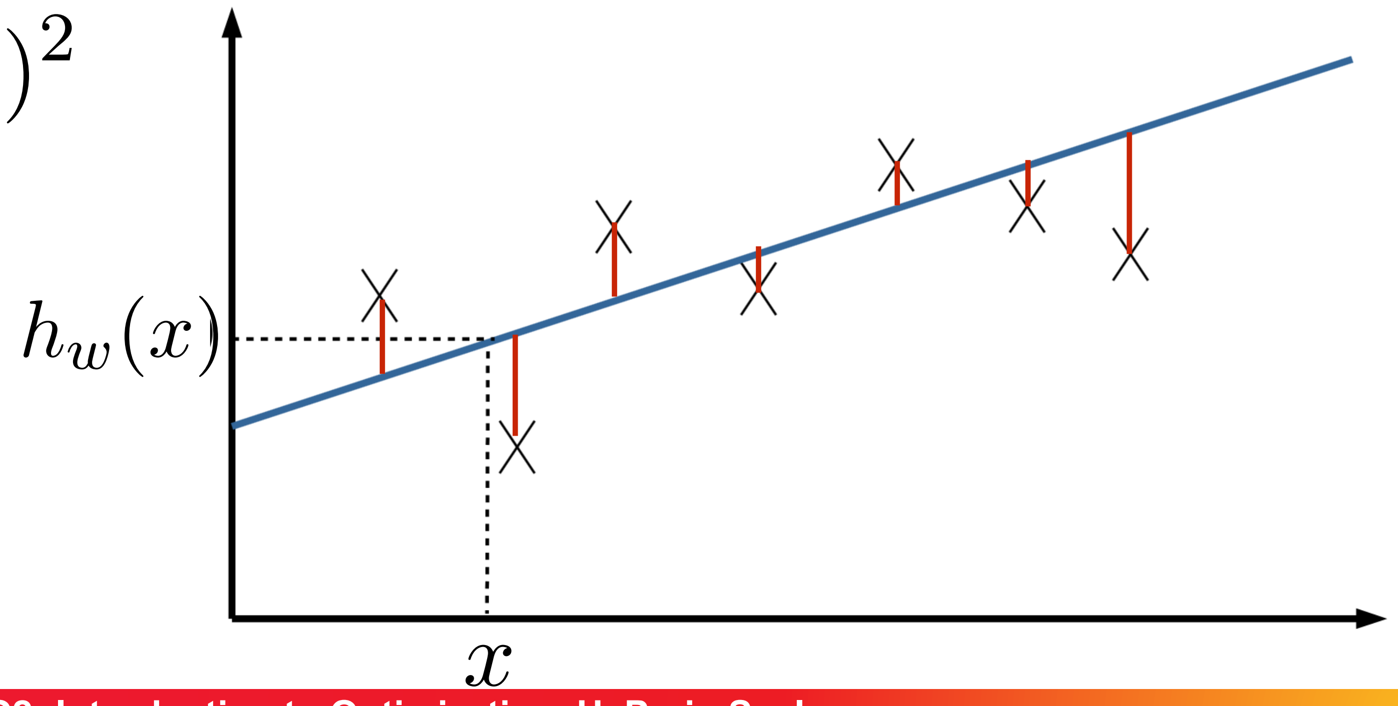
Example 1: Linear Regression

Hypothesis: linear model

$$h_w(x) = w_0 + w_1 x_1 + \ldots + w_{d-1} x_{d-1} \overset{x_0 = 1}{=} \langle w, x \rangle$$

Find $h_w(x)$ via solving the minimization problem

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (h_w(x^i) - y^i)^2$$



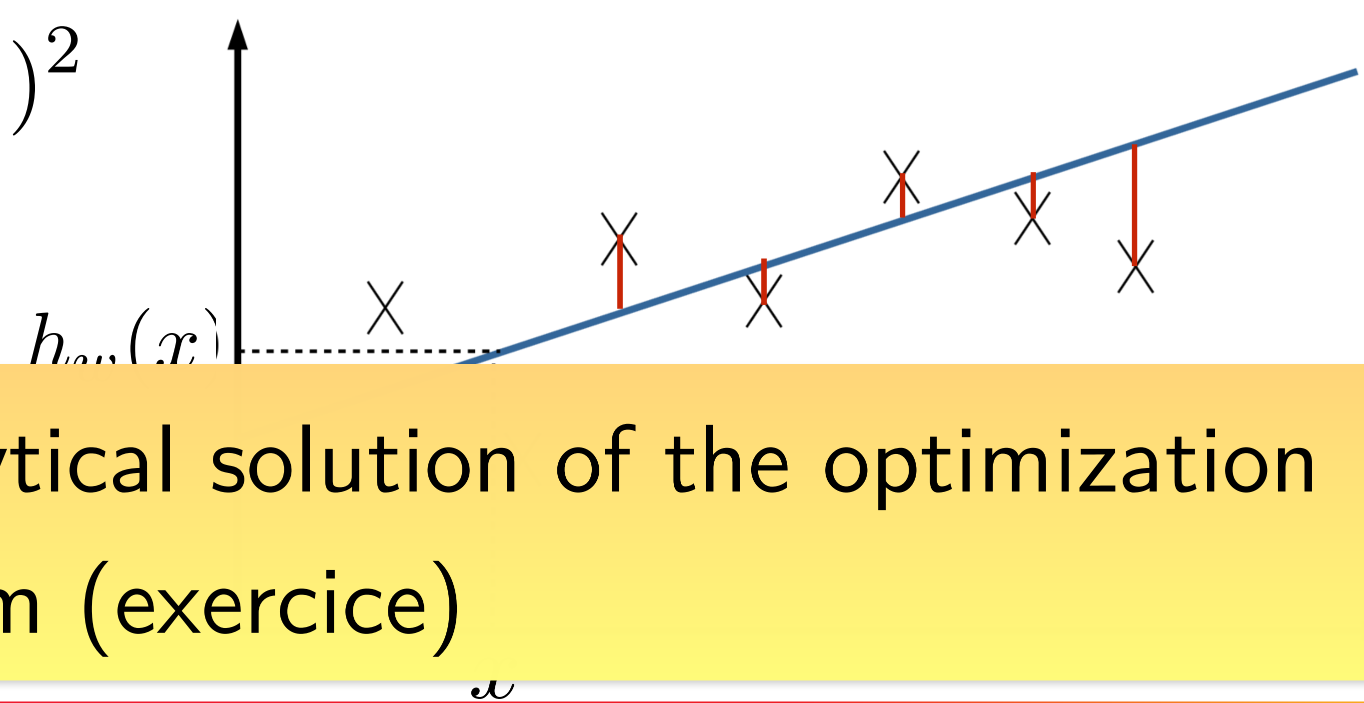
Example 1: Linear Regression

Hypothesis: linear model

$$h_w(x) = w_0 + w_1 x_1 + \ldots + w_{d-1} x_{d-1} \overset{x_0 = 1}{=} \langle w, x \rangle$$

Find $h_w(x)$ via solving the minimization problem

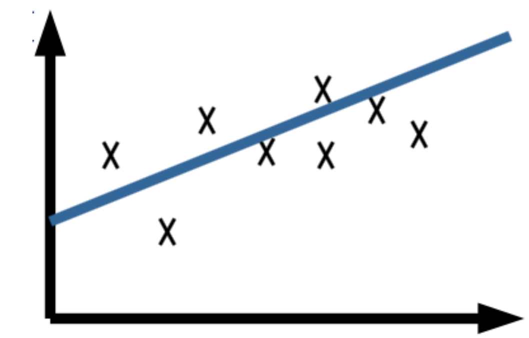
$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (h_w(x^i) - y^i)^2$$



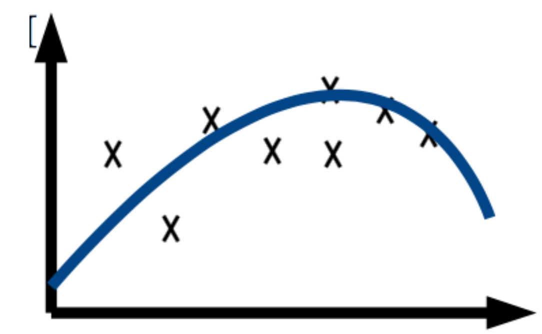
In this case we find an analytical solution of the optimization problem (exercice)

Generalization: Parametrization of the Hypothesis

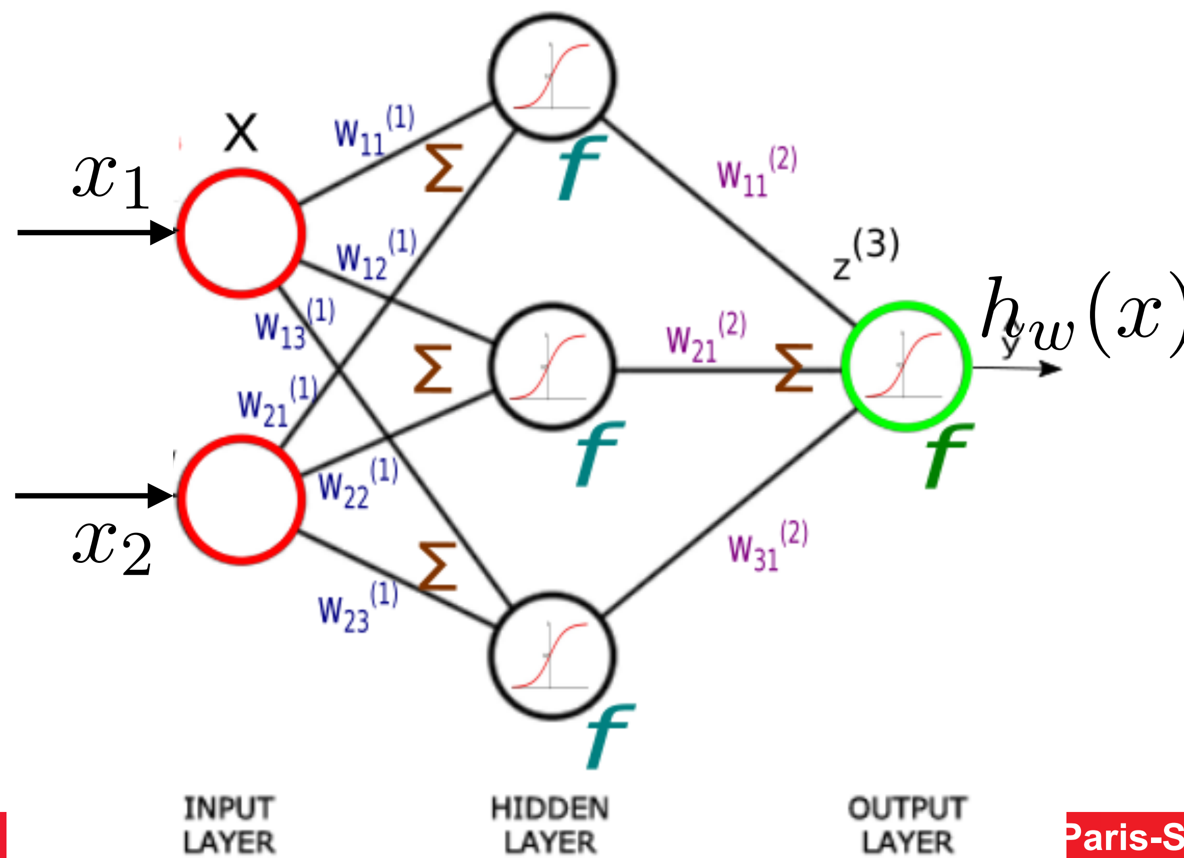
Linear: $$h_w(x) = \langle w, x \rangle = \sum_{i=0}^{d-1} w_i x_i$$



Polynomial: $$h_w(x) = \sum_{i,j=0}^{d-1} w_{i,j} x_i x_j$$



Neural network:



Generalization: Different Loss Functions

Start from the linear regression problem:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (h_w(x^i) - y^i)^2$$

Let $y_h := h_w(x)$

Loss function: $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_+$

$$(y_h, y) \to l(y_h, y)$$

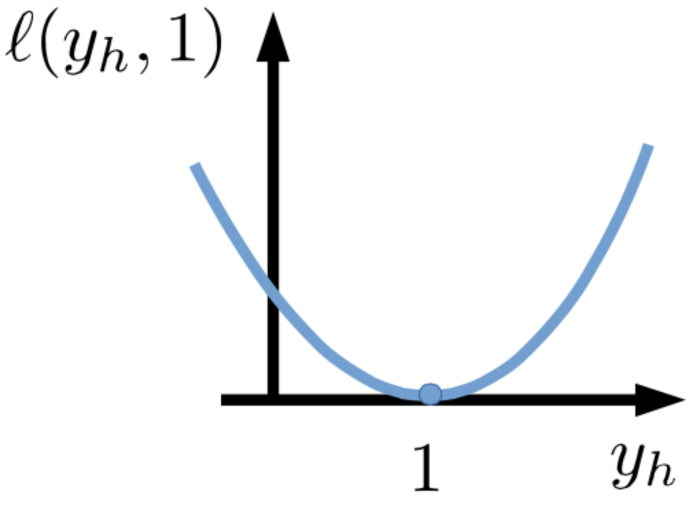
For linear regression
$l(y_h, y) = (y_h - y)^2$

Training (optimization) problem:

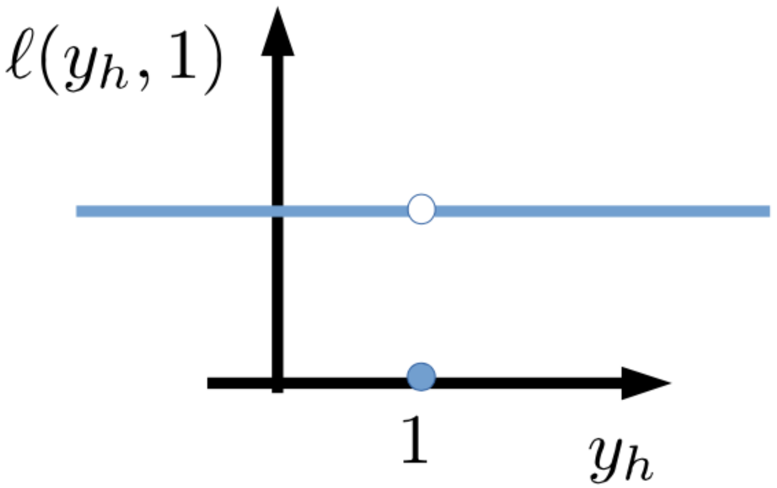
$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} l(h_w(x^i), y^i)$$

Generalization: Different loss functions

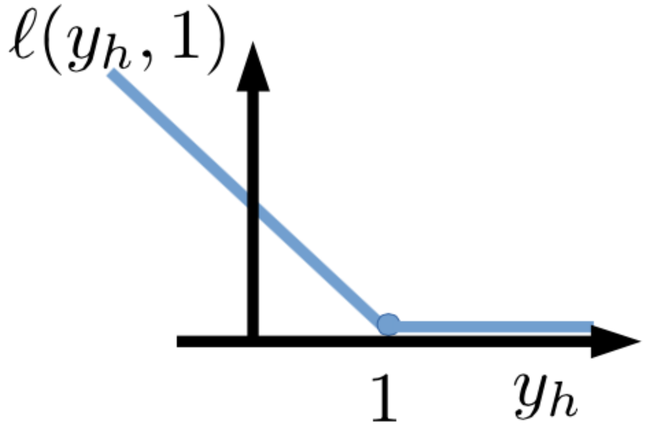
Quadratic loss: $l(y_h, y) = (y_h - y)^2$



Binary loss: $l(y_h, y) = \begin{cases} 0 \text{ if } y_h = y \\ 1 \text{ if } y_h \neq y \end{cases}$



Hinge loss: $l(y_h, y) = \max\{0, 1 - y_h y\}$



Exercice - Linear Regression

- Show that we can formulate the problem of linear regression as minimizing the following function:

$$f(w) := \|Xw - y\|^2 \qquad X \in \mathbb{R}^{n \times d}$$

- Show that f is convex
- Deduce that w_{opt} is solution of $\nabla f(w_{\text{opt}}) = 0$
- Show that

$$w_{\text{opt}} = (X^\top X)^{-1} X^\top y$$

if $X^\top X$ invertible

Numerical Optimization

Very often it is not possible to solve analytically the equation $\nabla f(x) = 0$ and we have to resort to an iterative algorithm (or numerical optimization algorithm) that will generate a sequence of points $\{x_k : k \geq 0\}$ that should converge to $\mathrm{argmin}_x f(x)$

Optimization algorithm:

input $f, \nabla f, (\nabla^2 f)$

initialize $k = 0, x_0$ [other state variables]

while not happy do

update x_k

$k = k + 1$

end-do

return x_k, k

$f(x_{k+1}) \leq f(x_k)$ (typically)

Goal:

$\lim_{k \to \infty} f(x_k) = \min_x f(x)$

$\lim_{k \to \infty} \|x_k - x^*\| = 0$

Algorithm Classes

Depending on the information the algorithm is using to create a new point (or iterate) we distinguish

Zero-order's algorithms: only use f (no gradients, ...). Those methods are also called derivative-free optimization algorithms. Used when gradient or Hessian are difficult to compute, or when the functions are not differentiable.

First-order algorithms: use f and ∇f. Standard algorithms when f is differentiable, convex.

Second-order algorithms: use $f, \nabla f$ and $\nabla^2 f$. When we can have an "easy" access to the Hessian matrix.

Descent Algorithm

descent direction

Illustration idea from “Alexander & Michael Bronstein” Numerical Optimization slides

Descent Algorithm

Generic algorithm:

choose an initial point x_0, $k = 0$

while not happy

choose a descent direction d_k

line-search: choose a step-size σ_k

$x_{k+1} = x_k + \sigma_k d_k$

$k = k + 1$

Line search: 1-d minimization along the descent direction

$\sigma \to f(x_k + \sigma d_k)$

Descent direction: direction such that for σ small enough

$$f(x_k + \sigma d_k) < f(x_k)$$

Stopping criteria

When are we "happy", i.e. when do we stop the algorithm?

- when gradient norm becomes small

$$\|\nabla f(x_k)\| \leq \epsilon$$

- when step-size becomes small

$$\|x_{k+1} - x_k\| \leq \epsilon$$

- when progress in f becomes small

$$\frac{|f(x_{k+1}) - f(x_k)|}{|f(x_k)|} \leq \epsilon$$

Newton's Algorithm

Take as descent direction the Newton step:

$$d_k = -[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$$

The Newton's direction minimizes the best locally quadratic approximation of f. Indeed, by Taylor's expansion we can approximate f locally in x by

$$\begin{aligned} g(h) &= f(x) + \nabla f(x)^\top h + \tfrac{1}{2} h^\top \nabla^2 f(x) h \\ &\approx f(x+h) \end{aligned}$$

Minimizing g with respect to h yields:

$$h = -[\nabla^2 f(x)]^{-1} \nabla f(x)$$

Quasi-Newton's Methods

In quasi-Newton's methods, the Newton direction is approximated by using solely first order information (gradient)

Key idea: successive iterates x_k, x_{k+1} and gradients $\nabla f(x_k)$ yield second order information

$$q_k \approx \nabla^2 f(x_{k+1}) p_k$$

$$p_k = x_{k+1} - x_k, \; q_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm

BFGS algorithm:

B_k approximation of Hessian matrix

$$d_k = -B_k^{-1}\nabla f(x_k)$$
$$x_{k+1} = x_k + \sigma_k d_k \text{ (find } \sigma_k \text{ via line-search)}$$
$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$
$$B_{k+1} = B_k + \frac{y_k y_k^\top}{y_k^\top \sigma_k d_k} - \frac{B_k d_k d_k^\top B_k}{d_k^\top B_k d_k}$$

efficient update to compute the inverse of B_k

Considered as the state-of-the-art quasi-Newton's algorithm.
Implemented in all (good) optimization toolboxes

Gradient Descent - Simple Theoretical Analysis

Theorem[Linear convergence of gradient descent] Assume $f : \mathbb{R}^d \to \mathbb{R}$ is twice continuously differentiable, convex and for all x, $\mu I_d \preccurlyeq \nabla^2 f(x) \preccurlyeq L I_d$ with $\mu > 0$. Let x^* be the unique global minimum of f. The gradient descent algorithm with fixed step-size $\sigma_t = \frac{1}{L}$ satisfies

$$\|x_{k+1} - x^*\|^2 \leq \left(1 - \frac{\mu}{L}\right) \|x_k - x^*\|^2 \ .$$

That is the algorithm converges geometrically (also called linearly):

$$\|x_k - x^*\|^2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x_0 - x^*\|^2$$

algorithm slower and slower with increasing condition number

In comparison, convergence of Newton's method is quadratic:

$$\|x_{k+1} - x^*\| \leq c \|x_k - x^*\|^2 \text{ with } c < 1$$

$$\|x_{k+1} - x^*\|^2 \leq c^2 \left(\|x_k - x^*\|^2\right)^2 \text{ with } c < 1$$

Remarks:

$A \preccurlyeq B$ means $x^T A x \leq x^T B x$ for all x

For f twice continuous differentiable $\mu I_d \preccurlyeq \nabla^2 f(x)$ is called μ-strong convexity

a strongly convex function does not need to be twice continuously differentiable (it is assumed for the sake of simplicity)

$\mu I_d \preccurlyeq \nabla^2 f(x) \preccurlyeq L I_d$ is equivalent to the eigenvalues of the Hessian of f are in between mu and L

Stochastic Gradient - Motivation

We now come back to our training optimization problem

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \underbrace{l(h_w(x^i), y^i)}_{f_i(w)}$$

the f_i can include a regularization term

Gradient descent update:

$$w_{k+1} = w_k - \sigma_k \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_k)$$

Problem: each iteration requires to compute a gradient $\nabla f_i(w)$ for each data point. We don't want to do that when n is large (quite typical).

Stochastic Gradient

The gradient of $f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$ is approximated by the gradient of a single data function $f_i(w)$ at each iteration

$$\nabla f(w) \approx \nabla f_i(w) \text{ for } j \text{ chosen at random}$$

Stochastic gradient descent update:

$$\text{sample } j \in \{1, \ldots, n\}$$
$$w_{k+1} = w_k - \sigma_k \nabla f_i(w_k)$$