Optimization for Machine Learning "Hands On"

Alexandre Gramfort (Inria)
http://alexandre.gramfort.net/

Quentin Bertrand (Inria)
https://qb3.github.io

Disclaimer

- 3 hours is too short for a detailed course on optimization
- We will cover only unconstrained problems
- We will not cover non-smooth problems (e.g., Lasso, SVM)
- The objective is to grasp quickly some theoretical aspects ...
- ... and to code everything to be able to experiment.

References and useful links

- S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004, pp. xiv+716
- J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006

- Lecture notes from Robert Gower, *Master2 Optimization for Data Science*: https://gowerrobert.github.io/
- Cool website for visualization of optimization algorithms: http://fa.bianp.net/teaching/2018/COMP-652/

Is this a cat?

Yes

No

Is this a cat?

Yes

Is this a cat?

Yes

x : Input / Feature

y : Output / Target

Goal: find mapping h that assigns the 'correct' target to each input

$$h : x \in \mathbb{R}^p \longrightarrow y \in \mathbb{R}$$

Empirical Risk Minimization (ERM)

Goal: from examples $(x_1, y_1), \ldots, (x_n, y_n)$ learn a function $h : \mathbb{R}^p \to \mathbb{R}$ such that

$$h(x_{n+1}) \simeq y_{n+1}$$

Ideally, for a given loss function L :

$$h^* \in \arg\min_{h \in \mathcal{H}} \underbrace{\mathbb{E}[L(h(x), y)]}_{\text{Expected risk}}$$

Empirical Risk Minimization (ERM)

Goal: from examples $(x_1, y_1), \ldots, (x_n, y_n)$ learn a function $h : \mathbb{R}^p \to \mathbb{R}$ such that

$$h(x_{n+1}) \simeq y_{n+1}$$

Ideally, for a given loss function L :

$$h^* \in \arg\min_{h \in \mathcal{H}} \underbrace{\mathbb{E}[L(h(x), y)]}_{\text{Expected risk}}$$

In practice:

$$h^* \in \arg\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n} \sum_{i}^{n} L(h(x_i), y_i)}_{\text{Empirical risk}}$$

Empirical Risk Minimization (ERM)

Goal: from examples $(x_1, y_1), \ldots, (x_n, y_n)$ learn a function $h : \mathbb{R}^p \to \mathbb{R}$ such that

$$h(x_{n+1}) \simeq y_{n+1}$$

Ideally, for a given loss function L :

$$h^* \in \arg\min_{h \in \mathcal{H}} \underbrace{\mathbb{E}[L(h(x), y)]}_{\text{Expected risk}}$$

In practice:

$$h^* \in \arg\min_{h \in \mathcal{H}} \underbrace{\frac{1}{n} \sum_{i}^{n} L(h(x_i), y_i)}_{\text{Empirical risk}}$$

Examples of ERM in Practice

Let $X = [x_1, \ldots, x_n]^\top \in \mathbb{R}^{n \times p}$ (design matrix)

(Regularized) Linear Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \underbrace{\frac{1}{2} \|y - Xw\|^2}_{\ell_2 \text{ Loss}} + \frac{\lambda}{2} \|w\|^2$$

(Regularized) Logistic Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \underbrace{\log(1 + \exp(-y_i x_i^\top w))}_{\text{Logistic Loss}} + \frac{\lambda}{2} \|w\|^2$$

Examples of ERM in Practice

Let $X = [x_1, \ldots, x_n]^\top \in \mathbb{R}^{n \times p}$ (design matrix)

(Regularized) Linear Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \underbrace{\frac{1}{2} \|y - Xw\|^2}_{\ell_2 \text{ Loss}} + \frac{\lambda}{2} \|w\|^2$$

(Regularized) Logistic Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \underbrace{\log(1 + \exp(-y_i x_i^\top w))}_{\text{Logistic Loss}} + \frac{\lambda}{2} \|w\|^2$$

- All of these are convex optimization problems!

Examples of ERM in Practice

Let $X = [x_1, \ldots, x_n]^\top \in \mathbb{R}^{n \times p}$ (design matrix)

(Regularized) Linear Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \underbrace{\frac{1}{2} \|y - Xw\|^2}_{\ell_2 \text{ Loss}} + \frac{\lambda}{2} \|w\|^2$$

(Regularized) Logistic Regression:

$$w^* = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \underbrace{\log(1 + \exp(-y_i x_i^\top w))}_{\text{Logistic Loss}} + \frac{\lambda}{2} \|w\|^2$$

- All of these are convex optimization problems!

Gradient

Definition (Gradient)

For $f : \mathbb{R}^p \to \mathbb{R}$ smooth the gradient reads:

$$\nabla f(w) = \left[\frac{\partial f(w)}{\partial w_1}, \ldots, \frac{\partial f(w)}{\partial w_p}\right]^\top \in \mathbb{R}^p$$

Examples

- $f(w) = x^\top w$ where $x \in \mathbb{R}^p$, then $\nabla f(w) = x$
- $f(w) = g(x^\top w)$ where $g : \mathbb{R} \to \mathbb{R}$, then $\nabla f(w) = g'(x^\top w)x$
- $f(w) = w^\top A w$ where $A \in \mathbb{R}^{p \times p}$, $\nabla f(w) = (A + A^\top)w$

Hessian

Definition (Hessian matrix)

For $f : \mathbb{R}^p \to \mathbb{R}$ with smooth gradient the hessian matrix reads:

$$\nabla^2 f(w) = \begin{bmatrix} \frac{\partial^2 f(w)}{\partial w_1^2} & \frac{\partial^2 f(w)}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f(w)}{\partial w_1 \partial w_p} \\ \frac{\partial^2 f(w)}{\partial w_2 \partial w_1} & \frac{\partial^2 f(w)}{\partial w_2^2} & \cdots & \frac{\partial^2 f(w)}{\partial w_2 \partial w_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(w)}{\partial w_p \partial w_1} & \frac{\partial^2 f(w)}{\partial w_p \partial w_2} & \cdots & \frac{\partial^2 f(w)}{\partial w_p^2} \end{bmatrix} = (\partial^2_{i,j} f(w))_{i,j}$$

Examples

- $f(w) = x^\top w$ where $x \in \mathbb{R}^p$, then $\nabla^2 f(w) = 0$
- $f(w) = w^\top A w$ where $A \in \mathbb{R}^{p \times p}$, $\nabla^2 f(w) = A + A^\top$

Warm up!

- You have 3 minutes to compute the gradient ∇f and the Hessian $\nabla^2 f$ of the linear regression function:
$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Warm up!

- You have 3 minutes to compute the gradient ∇f and the Hessian $\nabla^2 f$ of the linear regression function:

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

- Solution:

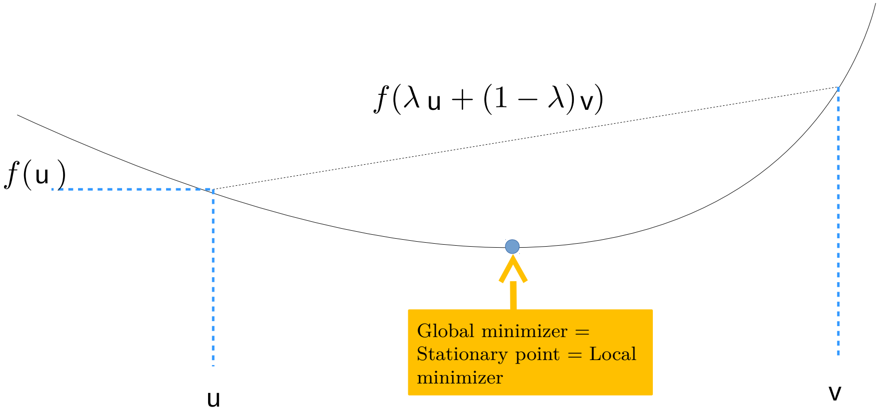
$$\nabla f(w) = X^\top (Xw - y)$$
$$\nabla^2 f(w) = X^\top X$$

Convexity I

Definition (Convex function)

$f : \mathbb{R}^p \to \mathbb{R}$ is convex if and only if, $\forall u, v \in \mathbb{R}^p$, $\forall \lambda \in [0,1]$:

$$f(\lambda u + (1-\lambda)v) \leq \lambda f(u) + (1-\lambda) f(v)$$

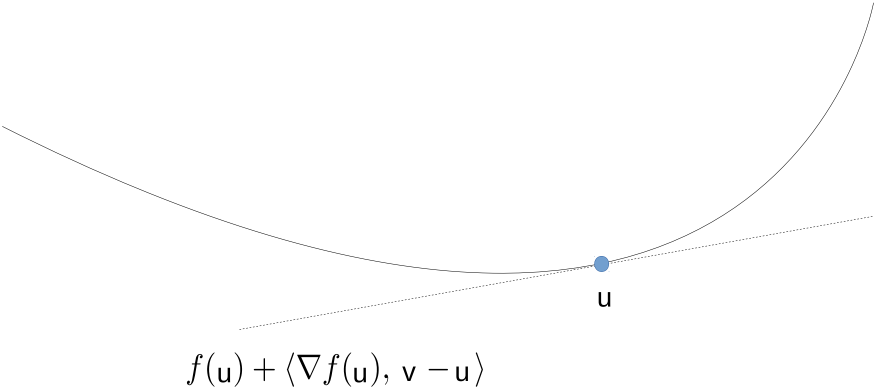


Convexity II

Proposition (Convex differentiable function / has a gradient)

$f : \mathbb{R}^p \to \mathbb{R}$ *is convex if and only if,* $\forall u, v \in \mathbb{R}^p$*:*

$$f(v) \geq f(u) + \langle \nabla f(u), v - u \rangle$$

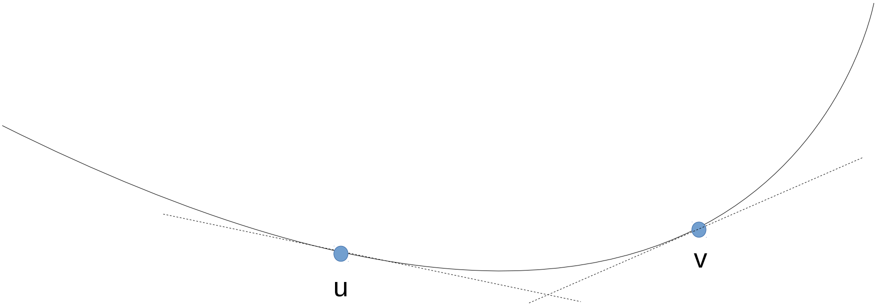


Convexity III

Proposition (Convex twice differentiable function)

$f : \mathbb{R}^p \to \mathbb{R}$ is convex if and only if, $\forall u \in \mathbb{R}^p$:

$$\nabla^2 f(u) \succeq 0$$



$$\mathrm{u} \leq \mathrm{v} \qquad \Rightarrow f'(\mathrm{u}) \leq f'(\mathrm{v})$$

Strong Convexity

Definition (Strongly convex function)

$f : \mathbb{R}^p \to \mathbb{R}$ is μ-strongly convex if and only if, $\forall u, v \in \mathbb{R}^p$:

$$f(v) \geq f(u) + \langle \nabla f(u), v - u \rangle + \frac{\mu}{2}\|v - u\|^2$$

Proposition (Strongly convex twice differentiable function)

$f : \mathbb{R}^p \to \mathbb{R}$ is μ-strongly convex if and only if, $\forall u \in \mathbb{R}^p$:

$$\nabla^2 f(u) \succeq \mu \, \mathrm{Id}_p$$

Strong Convexity

Definition (Strongly convex function)

$f : \mathbb{R}^p \to \mathbb{R}$ is μ-strongly convex if and only if, $\forall u, v \in \mathbb{R}^p$:

$$f(v) \geq f(u) + \langle \nabla f(u), v - u \rangle + \frac{\mu}{2} \|v - u\|^2$$

Proposition (Strongly convex twice differentiable function)

$f : \mathbb{R}^p \to \mathbb{R}$ is μ-strongly convex if and only if, $\forall u \in \mathbb{R}^p$:

$$\nabla^2 f(u) \succeq \mu \, \mathrm{Id}_p$$

Smoothness

Definition (L-Smoothness)

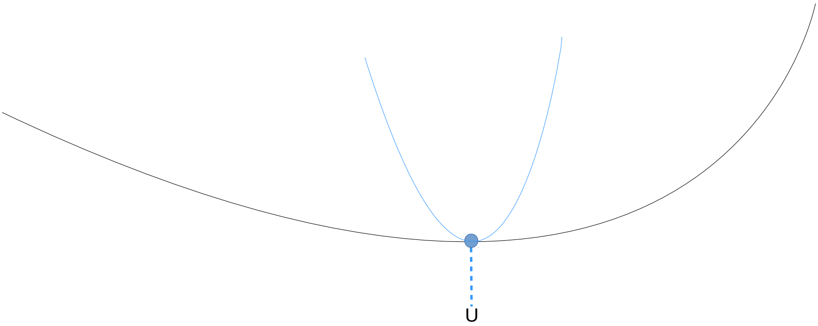
A function $f : \mathbb{R}^p \to \mathbb{R}$ is L-smooth if ∇f is L-Lipschitz:

$$\|\nabla f(u) - \nabla f(v)\| \leq L \|u - v\|, \quad \forall u, v \in \mathbb{R}^p$$

in particular this implies

$$f(v) \leq f(u) + \langle \nabla f(u), v - u \rangle + \frac{L}{2} \|v - u\|^2$$

Remark: In practice one wants L as small as possible



From surrogate minimization to Gradient Descent (GD)

For f L-smooth Taylor expansion gives:

$$f(w) \leq \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|^2}_{\text{Surrogate function } \tilde{f}(w)}$$

Idea: Minimize the surrogate function $\tilde{f}$

$\tilde{f}$ is convex, differentiable, infinite at the infinite (coercive)
$\Rightarrow$ its minimum w^* is achieved where gradient is 0:

$$0 = \nabla \tilde{f}(w^*) = \nabla f(w^k) + L(w^* - w^k)$$

From surrogate minimization to Gradient Descent (GD)

For f L-smooth Taylor expansion gives:

$$f(w) \leq \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|^2}_{\text{Surrogate function } \tilde{f}(w)}$$

Idea: Minimize the surrogate function $\tilde{f}$

$\tilde{f}$ is convex, differentiable, infinite at the infinite (coercive)
$\Rightarrow$ its minimum w^* is achieved where gradient is 0:

$$0 = \nabla \tilde{f}(w^*) = \nabla f(w^k) + L(w^* - w^k)$$

Taking w^{k+1} as the minimizer of $\tilde{f}$ leads to gradient descent:

$$w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$$

From surrogate minimization to Gradient Descent (GD)

For f L-smooth Taylor expansion gives:

$$f(w) \leq \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{L}{2}\|w - w^k\|^2}_{\text{Surrogate function } \tilde{f}(w)}$$

Idea: Minimize the surrogate function $\tilde{f}$

$\tilde{f}$ is convex, differentiable, infinite at the infinite (coercive)
$\Rightarrow$ its minimum w^* is achieved where gradient is 0:

$$0 = \nabla \tilde{f}(w^*) = \nabla f(w^k) + L(w^* - w^k)$$

Taking w^{k+1} as the minimizer of $\tilde{f}$ leads to gradient descent:

$$w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$$

Exercise 1

- Exercise Write the GD algorithm for the linear regression

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: Gradient Descent

---

init : $w^0 = 0_p,\ L$

for $\text{iter} = 1, \ldots,$ **do**

$\quad w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$

Exercise 1

- Solution Write the GD algorithm for the linear regression

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: Gradient Descent

---

init : $w^0 = 0_p,\ L := \|X\|_2^2$

for $\text{iter} = 1, \ldots,$ **do**

$\quad w^{k+1} = w^k - \frac{1}{L} X^\top (Xw^k - y)$

Line-search[1]

What to do when a function is smooth, but you do not know the Lipschitz constant L? Or when L is too conservative?

Gradient descent with variable stepsize:

$$w^{k+1} = w^k - \alpha^k \nabla f(w^k)$$

where the stepsize α^k changes at each iteration and is found by line-search.

[1] J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006, Chap. 3.

Line-search[1]

What to do when a function is smooth, but you do not know the Lipschitz constant L? Or when L is too conservative?

Gradient descent with variable stepsize:

$$w^{k+1} = w^k - \alpha^k \nabla f(w^k)$$

where the stepsize α^k changes at each iteration and is found by line-search.

[1] J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006, Chap. 3.

Hands on 0

→ See notebook: `00-gradient_descent_line_search.ipynb`

Gradient descent: theoretical results

Algorithm: GD

init : $w^0 = 0_p,\ L$

for $\mathrm{iter} = 1, \ldots,$ **do**

$\quad w^{k+1} = w^k - \frac{1}{L} \nabla f(w^k)$

Proposition

If f is convex and L-smooth, then:

$$f(w^k) - f(w^*) \leq \frac{2L \left\| w^0 - w^* \right\|^2}{k}$$

One has "sublinear" convergence.

Gradient descent: theoretical results

Algorithm: GD

init : $w^0 = 0_p,\ L$

for $\text{iter} = 1, \dots,$ **do**

$\quad w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$

Proposition

If f is convex and L-smooth, then:

$$f(w^k) - f(w^*) \leq \frac{2L\left\|w^0 - w^*\right\|^2}{k}$$

One has "sublinear" convergence.

Proposition

If f is μ-strongly convex and L-smooth, then:

$$\|w^k - w^*\| \leq \left(1 - \frac{\mu}{L}\right)^k \|w^0 - w^*\|$$

One has "linear" (a.k.a. "exponential") convergence.

Gradient descent: theoretical results

Algorithm: GD

init : $w^0 = 0_p,\ L$

for $\mathrm{iter} = 1, \ldots,$ **do**

$\quad w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k)$

Proposition

If f is convex and L-smooth, then:

$$f(w^k) - f(w^*) \leq \frac{2L\left\|w^0 - w^*\right\|^2}{k}$$

One has "sublinear" convergence.

Proposition

If f is μ-strongly convex and L-smooth, then:

$$\|w^k - w^*\| \leq \left(1 - \frac{\mu}{L}\right)^k \|w^0 - w^*\|$$

One has "linear" (a.k.a. "exponential") convergence.

Hands on 1

$\rightarrow$ See notebook: `01-logistic_gd.ipynb`

Remark

- In all the hands on the main algorithm is already implemented
- We propose you to add little modifications
- All the solutions are in the solutions folder
- But do not look at them too quickly ...

Newton method: intuition

Taylor expansion to the order 2:

$$f(w) \approx \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{1}{2}(w - w^k)^\top \nabla^2 f(w^k)(w - w^k)}_{\tilde{f}(w)}$$

Idea: Minimize the quadratic approximation $\tilde{f}$:

$$0 = \nabla \tilde{f}(w^{k+1}) = \nabla f(w^k) + \nabla^2 f(w^k)(w^{k+1} - w^k)$$

Newton method: intuition

Taylor expansion to the order 2:

$$f(w) \approx \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{1}{2}(w - w^k)^\top \nabla^2 f(w^k)(w - w^k)}_{\tilde{f}(w)}$$

Idea: Minimize the quadratic approximation $\tilde{f}$:

$$0 = \nabla \tilde{f}(w^{k+1}) = \nabla f(w^k) + \nabla^2 f(w^k)(w^{k+1} - w^k)$$

This leads to the following update for the Newton algorithm:

$$w^{k+1} = w^k - [\nabla^2 f(w^k)]^{-1} \nabla f(w^k)$$

Newton method: intuition

Taylor expansion to the order 2:

$$f(w) \approx \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{1}{2}(w - w^k)^\top \nabla^2 f(w^k)(w - w^k)}_{\tilde{f}(w)}$$

Idea: Minimize the quadratic approximation $\tilde{f}$:

$$0 = \nabla \tilde{f}(w^{k+1}) = \nabla f(w^k) + \nabla^2 f(w^k)(w^{k+1} - w^k)$$

This leads to the following update for the Newton algorithm:

$$w^{k+1} = w^k - [\nabla^2 f(w^k)]^{-1} \nabla f(w^k)$$

What about convergence guarantees?

Newton method: intuition

Taylor expansion to the order 2:

$$f(w) \approx \underbrace{f(w^k) + \langle \nabla f(w^k), w - w^k \rangle + \frac{1}{2}(w - w^k)^\top \nabla^2 f(w^k)(w - w^k)}_{\tilde{f}(w)}$$

Idea: Minimize the quadratic approximation $\tilde{f}$:

$$0 = \nabla \tilde{f}(w^{k+1}) = \nabla f(w^k) + \nabla^2 f(w^k)(w^{k+1} - w^k)$$

This leads to the following update for the Newton algorithm:

$$w^{k+1} = w^k - [\nabla^2 f(w^k)]^{-1} \nabla f(w^k)$$

What about convergence guarantees?

Exercise 2

- Exercise Write the Newton algorithm for the linear regression

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: Newton algorithm

init : $w^0 = 0_p$
for $\text{iter} = 1, \ldots,$ **do**
$\quad w^{k+1} = w^k - [\nabla^2 f(w^k)]^{-1} \nabla f(w^k)$

Exercise 2

- Solution Write the Newton algorithm for the linear regression

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: Newton algorithm

---

init : $w^0 = 0_p$

for iter $= 1, \dots,$ **do**

$\quad w^{k+1} = w^k - [X^\top X]^{-1} X^\top (Xw^k - y)$

Exercise 2

- Solution Write the newton algorithm for the linear regression

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: Newton algorithm

---

init : $w^0 = 0_p$

for $\text{iter} = 1, \ldots,$ **do**

$\quad w^{k+1} = w^k - [X^\top X]^{-1} X^\top (Xw^k - y)$

Question: Is there an interest of applying Newton method on a linear regression problem?

Convergence of Newton method

Theorem (Convergence of Newton method)

Suppose that f is twice differentiable, and the Hessian $\nabla^2 f(w^*)$ is Lipschitz continuous in a neighborhood of a solution w^* such that $\nabla f(w^*) = 0$ and $\nabla^2 f(w^*) \succ 0$.
Then there exists a closed ball $\mathcal{B}$ centered on w^*, such that for every $w^0 \in \mathcal{B}$, the sequence w^k obtained with Newton algorithm stays in $\mathcal{B}$ and converges towards w^*. Furthermore, there is a constant $\gamma > 0$, such that $\|w^{k+1} - w^*\| \leq \gamma \|w^k - w^*\|^2$.
One has "super linear" convergence.

Drawbacks:

- Convergence of Newton is local (see proof in[2]). The method may diverge if the initial point is too far from w^* or if the Hessian is not positive definite
- One has to solve a linear system at each step! $O(p^3)$

---

[2] J. Nocedal and S. J. Wright. *Numerical optimization.* Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006, Thm. 3.5.

Convergence of Newton method

Theorem (Convergence of Newton method)

Suppose that f is twice differentiable, and the Hessian $\nabla^2 f(w^*)$ is Lipschitz continuous in a neighborhood of a solution w^* such that $\nabla f(w^*) = 0$ and $\nabla^2 f(w^*) \succ 0$.
Then there exists a closed ball $\mathcal{B}$ centered on w^*, such that for every $w^0 \in \mathcal{B}$, the sequence w^k obtained with Newton algorithm stays in $\mathcal{B}$ and converges towards w^*. Furthermore, there is a constant $\gamma > 0$, such that $\|w^{k+1} - w^*\| \leq \gamma \|w^k - w^*\|^2$.
One has "super linear" convergence.

Drawbacks:

- Convergence of Newton is local (see proof in[2]). The method may diverge if the initial point is too far from w^* or if the Hessian is not positive definite
- One has to solve a linear system at each step! $O(p^3)$

[2] J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006, Thm. 3.5.

From Newton to quasi-Newton

Idea:

- Construct iterative approximations of (the inverse of) the Hessian
- Combine it with line-search strategies

$$\begin{cases} d^k & = -B^k \nabla f(w^k) \quad \text{find a descent direction} \\ w^{k+1} & = w^k + \alpha^k d^k \quad \text{line-search} \end{cases}$$

- There exist a whole jungle of iterative approximations for the inverse of the Hessian: $B^{k+1} = B^k + \Delta^k$ [3]
- The one you should know about is BFGS strategy and the memory efficient L-BFGS[4]

[3] J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006.

[4] J. Nocedal. "Updating quasi-Newton matrices with limited storage". In: *Mathematics of computation* 35.151 (1980), pp. 773–782.

From Newton to quasi-Newton

Idea:

- Construct iterative approximations of (the inverse of) the Hessian
- Combine it with line-search strategies

$$\begin{cases} d^k & = -B^k \nabla f(w^k) \quad \text{find a descent direction} \\ w^{k+1} & = w^k + \alpha^k d^k \quad \text{line-search} \end{cases}$$

- There exist a whole jungle of iterative approximations for the inverse of the Hessian: $B^{k+1} = B^k + \Delta^k$ [3]
- The one you should know about is BFGS strategy and the memory efficient L-BFGS[4]

[3] J. Nocedal and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006.

[4] J. Nocedal. "Updating quasi-Newton matrices with limited storage". In: *Mathematics of computation* 35.151 (1980), pp. 773–782.

Hands on 2

$\rightarrow$ See notebook: `02-logistic_newton.ipynb`

Remark

- All the solutions are in the solutions folder ...

Optimization for ML: finite sum

Optimization in general:

$$\min_{x \in \mathcal{H}} f(x)$$

Optimization for Machine Learning (linear or logistic regression):

$$\min_{x \in \mathbb{R}^p} f(x) := \sum_{i=1}^{n} f_i(x)$$

Optimization for ML: finite sum

Optimization in general:

$$\min_{x \in \mathcal{H}} f(x)$$

Optimization for Machine Learning (linear or logistic regression):

$$\min_{x \in \mathbb{R}^p} f(x) := \sum_{i=1}^{n} f_i(x)$$

- Can we take advantage of this finite sum structure?
- In particular when the number of samples n is large?

Optimization for ML: finite sum

Optimization in general:

$$\min_{x \in \mathcal{H}} f(x)$$

Optimization for Machine Learning (linear or logistic regression):

$$\min_{x \in \mathbb{R}^p} f(x) := \sum_{i=1}^{n} f_i(x)$$

- Can we take advantage of this finite sum structure?
- In particular when the number of samples n is large?

Stochastic Gradient Descent (SGD)

Question: what is the cost per iteration of one update of GD?

- Full gradient methods can be expensive $O(np)$ per iteration!
- Idea: use a single gradient $\nabla f_i(w)$ instead of full gradient $\sum_i^n \nabla f_i(w)$ (n times faster!)

Algorithm: SGD

init : $w = 0_p$, α

for iter $= 1, \dots,$ **do**

  Sample i in $1, \dots, n$

  `// Single gradient` $\nabla f_i(w)$ `call`

  `//` $O(p)$ `per update`

  $w \leftarrow w - \alpha \nabla f_i(w)$

return w

Algorithm: GD

init : $w = 0_p$, α

for iter $= 1, \dots,$ **do**

  `// Full gradient` $\nabla f(w)$ `call`

  `//` $O(np)$ `per update`

  $w \leftarrow w - \alpha \nabla f(w)$

return w

Stochastic Gradient Descent (SGD)

Question: what is the cost per iteration of one update of GD?

- Full gradient methods can be expensive $O(np)$ per iteration!
- Idea: use a single gradient $\nabla f_i(w)$ instead of full gradient $\sum_i^n \nabla f_i(w)$ (n times faster!)

Algorithm: SGD

init : $w = 0_p$, α

for $\text{iter} = 1, \dots,$ **do**

  Sample i in $1, \dots, n$

  `// Single gradient` $\nabla f_i(w)$ `call`

  `//` $O(p)$ `per update`

  $w \leftarrow w - \alpha \nabla f_i(w)$

return w

Algorithm: GD

init : $w = 0_p$, α

for $\text{iter} = 1, \dots,$ **do**

  `// Full gradient` $\nabla f(w)$ `call`

  `//` $O(np)$ `per update`

  $w \leftarrow w - \alpha \nabla f(w)$

return w

Exercise 3

- Exercise Write the algorithm for the linear regression

$$f : w \mapsto \sum_{i=1}^{n} \underbrace{\frac{1}{2}(y_i - x_i^\top w)^2}_{f_i(w)}$$

Algorithm: SGD

init : $w = 0_p, \alpha$
for iter $= 1, \ldots,$ **do**
  Sample i in $1, \ldots, n$
  // Single gradient $\nabla f_i(w)$ call
  // $O(p)$ per update
  $w \leftarrow w - \alpha \nabla f_i(w)$
return w

Algorithm: GD

init : $w = 0_p, \alpha$
for iter $= 1, \ldots,$ **do**
  // Full gradient $\nabla f(w)$ call
  // $O(np)$ per update
  $w \leftarrow w - \alpha \nabla f(w)$
return w

Exercise 3

- Solution Write the algorithm for the linear regression

$$f : w \mapsto \sum_{i=1}^{n} \underbrace{\frac{1}{2}(y_i - x_i^\top w)^2}_{f_i(w)}$$

Algorithm: SGD

init : $w = 0_p$, α

for iter $= 1, \dots,$ **do**

  Sample i in $1, \dots, n$

  `// Single gradient` $\nabla f_i(w)$ `call`

  `//` $O(p)$ `per update`

  $w \leftarrow w - \alpha(x_i^\top w - y_i)x_i$

return w

Algorithm: GD

init : $w = 0_p$,

  $\alpha = 1/\|X\|_2^2$

for iter $= 1, \dots,$ **do**

  `// Full gradient` $\nabla f(w)$ `call`

  `//` $O(np)$ `per update`

  $w \leftarrow w - \alpha X^\top (Xw - y)$

return w

Exercise 3

- Solution Write the algorithm for the linear regression

$$f : w \mapsto \sum_{i=1}^{n} \underbrace{\frac{1}{2}(y_i - x_i^\top w)^2}_{f_i(w)}$$

Algorithm: SGD

init : $w = 0_p$, α

for $\text{iter} = 1, \ldots,$ **do**

  Sample i in $1, \ldots, n$

  `// Single gradient` $\nabla f_i(w)$ `call`

  `//` $O(p)$ `per update`

  $w \leftarrow w - \alpha(x_i^\top w - y_i)x_i$

return w

Algorithm: GD

init : $w = 0_p$, $\alpha = 1/\|X\|_2^2$

for $\text{iter} = 1, \ldots,$ **do**

  `// Full gradient` $\nabla f(w)$ `call`

  `//` $O(np)$ `per update`

  $w \leftarrow w - \alpha X^\top (Xw - y)$

return w

Question: how to choose α for the SGD?

SGD with constant stepsize α: theory

Proposition

If f is μ-strongly convex and L-smooth, then:

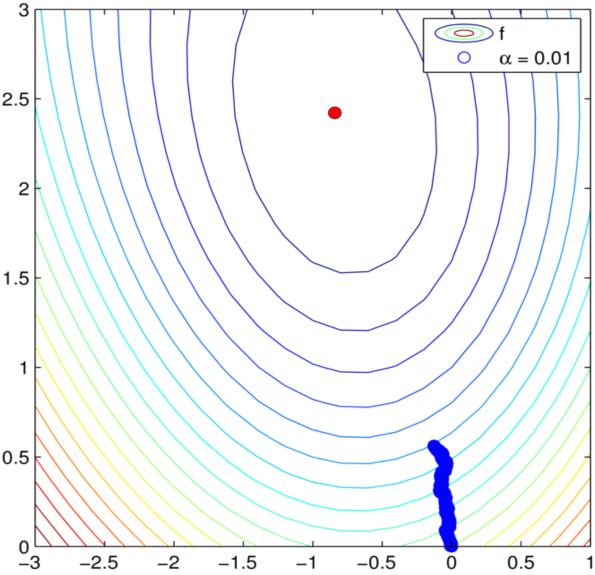
$$\mathbb{E}(\|w^k - w^*\|^2) \leq (1-\alpha\mu)^k \|w^0 - w^*\|^2 + \frac{\alpha}{\mu} C \ .$$

Remark

- Constant C depends on the norm for the f_i gradients (bounded gradient hypothesis)
- SGD with constant step size α does not converge
- In ML if the estimation and the approximation are bigger than the optimization error it's ok![a]

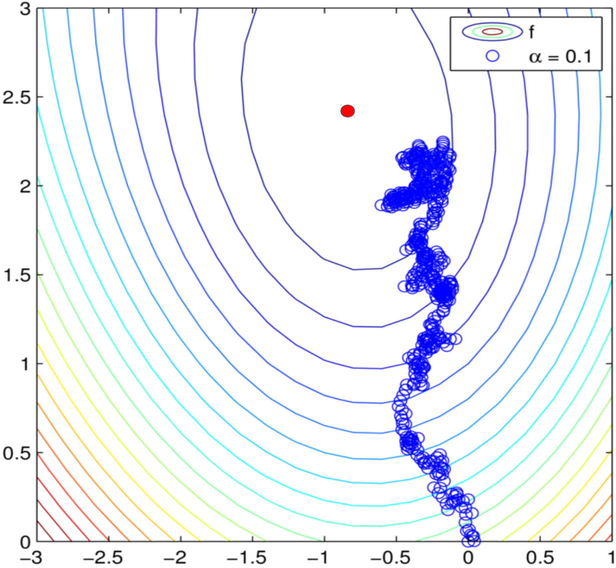
[a] L. Bottou and O. Bousquet. "The tradeoffs of large scale learning". In: *Advances in neural information processing systems*. 2008, pp. 161–168.

Example of SGD



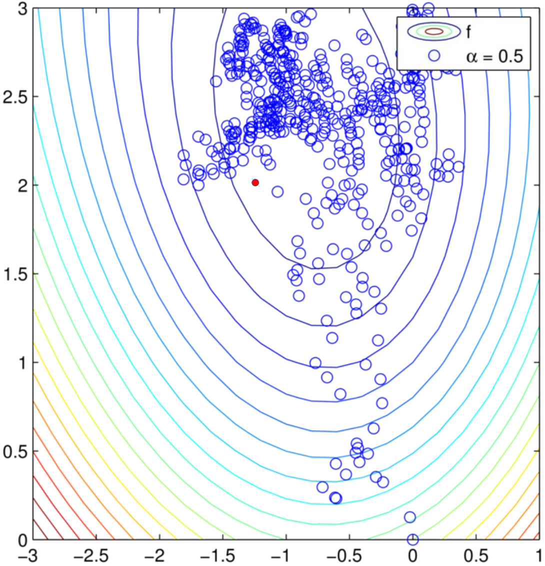
SGD on a 2D problem

Example of SGD



SGD on a 2D problem

Example of SGD



SGD on a 2D problem

Hands on 3

→ See notebook: `03-stochastic_gradient_descent.ipynb`

Remark

- All the solutions are in the solutions folder . . .

(Exact) Coordinate Descent

Goal:

$$\min_{w \in \mathbb{R}^p} f(w_1, \ldots, w_p)$$

Idea: solve smaller and simpler problems (one coordinate at a time)

(Exact) Coordinate Descent

Goal:

$$\min_{w \in \mathbb{R}^p} f(w_1, \ldots, w_p)$$

Idea: solve smaller and simpler problems (one coordinate at a time)

Algorithm: Exact coordinate descent

init : $w = 0_p$
for $\mathrm{iter} = 1, \ldots,$ **do**
  for $j = 1, \ldots, p$ **do**
    `// Update only one coef.`
    $w_j \leftarrow \arg\min_{z \in \mathbb{R}} f(w_1, \ldots, w_{j-1}, z, w_{j+1}, \ldots, w_p)$
return w

(Exact) Coordinate Descent

Goal:

$$\min_{w \in \mathbb{R}^p} f(w_1, \dots, w_p)$$

Idea: solve smaller and simpler problems (one coordinate at a time)

Algorithm: Exact coordinate descent

init : $w = 0_p$
for $\text{iter} = 1, \dots,$ **do**
  for $j = 1, \dots, p$ **do**
    `// Update only one coef.`
    $w_j \leftarrow \arg\min_{z \in \mathbb{R}} f(w_1, \dots, w_{j-1}, z, w_{j+1}, \dots, w_p)$
return w

Remark

- The order of cycle through coordinates is arbitrary, can use any permutation of $1, 2, \dots, p$
- We just have to solve 1D optimization problems but a lot of them...

(Exact) Coordinate Descent

Goal:

$$\min_{w \in \mathbb{R}^p} f(w_1, \ldots, w_p)$$

Idea: solve smaller and simpler problems (one coordinate at a time)

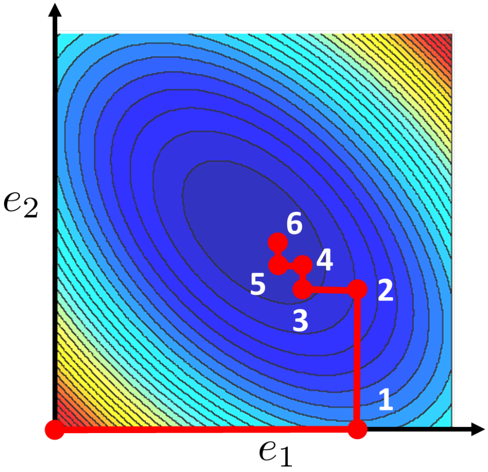
Algorithm: Exact coordinate descent

init : $w = 0_p$
for $\text{iter} = 1, \ldots,$ **do**
  for $j = 1, \ldots, p$ **do**
    `// Update only one coef.`
    $w_j \leftarrow \arg\min_{z \in \mathbb{R}} f(w_1, \ldots, w_{j-1}, z, w_{j+1}, \ldots, w_p)$
return w

Remark

- The order of cycle through coordinates is arbitrary, can use any permutation of $1, 2, \ldots, p$
- We just have to solve 1D optimization problems but a lot of them...

Example of CD



Coordinate descent on a 2D problem

Coordinate Gradient Descent: algorithm

- Exact minimization can be expensive
- Idea: do a local gradient step instead of exact minimization
- Step-sizes are now given by the coordinate-wise functions

Algorithm: CD

init : $w = 0_p,\ L_1,\ \dots,\ L_p$

for $\mathrm{iter} = 1, \dots,$ **do**

  for $j = 1, \dots, p$ **do**

    `// Update only one coef.`

    $w_j \leftarrow w_j - \frac{1}{L_j} \nabla_j f(w)$

return w

Algorithm: GD

init : $w = 0_p,\ L$

for $\mathrm{iter} = 1, \dots,$ **do**

  `// Update all the coef.`

  $w \leftarrow w - \frac{1}{L} \nabla f(w)$

return w

Exercise 4

- Exercise Write the CD algorithm for the linear regression

$$f : w \mapsto \|y - Xw\|^2$$

Algorithm: CD

init : $w = 0_p,\ L_1,\ \ldots,\ L_p$
for iter $= 1, \ldots,$ **do**
  for $j = 1, \ldots, p$ **do**
    `// Update only one coef.`
    $w_j \leftarrow w_j - \frac{1}{L_j} \nabla_j f(w)$
return w

Algorithm: GD

init : $w = 0_p,\ L$
for iter $= 1, \ldots,$ **do**
  `// Full gradient` $\nabla f(w)$ `call`
  $w \leftarrow w - \frac{1}{L} \nabla f(w)$
return w

Exercise 4

- Solution Write the CD algorithm for the linear regression

$$f : w \mapsto \|y - Xw\|^2$$

Algorithm: CD

init : $w = 0_p$, $L_j = \|X_{:j}\|^2$
for iter $= 1, \ldots,$ **do**
  for $j = 1, \ldots, p$ **do**
    `// Update only one coef.`
    $w_j \leftarrow w_j - \frac{X_{:j}^\top (Xw - y)}{L_j}$
return w

Algorithm: GD

init : $w = 0_p$, $\alpha = 1/\|X\|_2^2$
for iter $= 1, \ldots,$ **do**
  `// Full gradient` $\nabla f(w)$ `call`
  $w \leftarrow w - \alpha X^\top (Xw - y)$
return w

Exercise 4

- Solution Write the CD algorithm for the linear regression

$$f : w \mapsto \|y - Xw\|^2$$

Algorithm: CD

```
init  : w = 0_p, L_j = ||X_{:j}||^2
for iter = 1, ..., do
    for j = 1, ..., p do
        // Update only one coef.
        w_j ← w_j − X_{:j}^⊤(Xw − y) / L_j
return w
```

init : $w = 0_p,\ L_j = \|X_{:j}\|^2$
for iter $= 1, \ldots,$ **do**
  for $j = 1, \ldots, p$ **do**
    // Update only one coef.
    $w_j \leftarrow w_j - \frac{X_{:j}^\top (Xw - y)}{L_j}$
return w

Algorithm: GD

init : $w = 0_p,\ \alpha = 1/\|X\|_2^2$
for iter $= 1, \ldots,$ **do**
  // Full gradient $\nabla f(w)$ call
  $w \leftarrow w - \alpha X^\top (Xw - y)$
return w

Question What is the cost of one update of CD?

Convergence speed of CD[6]

Assume f is convex; ∇f is Lipschitz continuous

Proposition (Beck and Tetruashvili (2013))

$$f(w^{k+1}) - f(w^*) \leq 4L_{\max}(1 + n^3 L_{\max}^2 / L_{\min}^2) \frac{R^2(w^0)}{k + 8/n}$$

where $R^2(w^0) = \max_{u,v \in \mathcal{V}} \{\|u - v\| : f(v) \leq f(u) \leq f(w^0)\}$, $L_{\max} = \max_j L_j$ *and* $L_{\min} = \min_j L_j$.

- Worst case complexity rates of CD are bad (n^3 complexity) because it is possible to construct adversarial examples[5]
- Yet CD performs surprisingly well on real-world problems (see hands on)
- Random coordinate selection has a better average complexity

[5] R. Sun and Y. Ye. "Worst-case complexity of cyclic coordinate descent: $O(n^2)$ gap with randomized version". In: *Mathematical Programming* (2019), pp. 1–34.

[6] A. Beck and L. Tetruashvili. "On the convergence of block coordinate type methods". In: *SIAM J. Imaging Sci.* 23.4 (2013), pp. 651–694.

Take a look back before "Hands on 4"

$$f : w \mapsto \frac{1}{2}\|y - Xw\|^2$$

Algorithm: GD

init : $w = 0_p$, $\alpha = 1/\|X\|_2^2$
for $\mathrm{iter} = 1, \ldots,$ **do**
    `// Full gradient` $\nabla f(w)$ `call`
    $w \leftarrow w - \alpha X^\top (Xw - y)$
return w

Algorithm: SGD

init : $w = 0_p$, α
for $\mathrm{iter} = 1, \ldots,$ **do**
    Sample i in $1, \ldots, n$
    `// Single gradient` $\nabla f_i(w)$ `call`
    $w \leftarrow w - \alpha X_{i:}^\top (X_{i:} w - y_i)$
return w

Algorithm: CD

init : $w = 0_p$, $\alpha_j = 1/\|X_{:j}\|^2$
for $\mathrm{iter} = 1, \ldots,$ **do**
    Sample j in $1, \ldots, p$
    `// Single coordinate update`
    $w_j \leftarrow w_j - \alpha_j X_{:j}^\top (Xw - y)$
return w

Hands on 4

→ See notebook: `04-coordinate_descent.ipynb`

Remark

- All the solutions are in the solutions folder . . .

- Beck, A. and L. Tetruashvili. “On the convergence of block coordinate type methods”. In: *SIAM J. Imaging Sci.* 23.4 (2013), pp. 651–694.
- Bottou, L. and O. Bousquet. “The tradeoffs of large scale learning”. In: *Advances in neural information processing systems*. 2008, pp. 161–168.
- Boyd, S. and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004, pp. xiv+716.
- Nocedal, J. “Updating quasi-Newton matrices with limited storage”. In: *Mathematics of computation* 35.151 (1980), pp. 773–782.
- Nocedal, J. and S. J. Wright. *Numerical optimization*. Second. Springer Series in Operations Research and Financial Engineering. New York: Springer, 2006.
- Sun, R. and Y. Ye. “Worst-case complexity of cyclic coordinate descent: $O(n^2)$ gap with randomized version”. In: *Mathematical Programming* (2019), pp. 1–34.