An introduction to optimization for machine learning

Rodolphe Le Riche[1], Dédji Brian Whannou[2], Kévin Kpakpo Akouété[3], Espéran Padonou[4]

[1] CNRS LIMOS at Mines Saint-Etienne, France
[2] UBS Group AG
[3] Descartes Underwriting
[4] Fondation Vallet

July 2023
Ecole d'Eté en Intelligence Artificielle
fondation Vallet
Cotonou, Bénin

Foreword

This course was given during a summer school on AI in Godomey, Benin, July-Aug. 2023. The school was organized by the Benin Excellence NGO and the Vallet Foundation (cf. `https://www.fondationvallet.org/eeia`).

- The course provides basic concepts for numerical optimization
- for an audience interested in machine learning
- with a background corresponding to 1 year after high school
- through examples coded in python from scratch.
- Limitation: the algorithms are not exactly those used in state-of-the-art deep learning, but the main concepts, related to gradient descent, will be presented.

The code, the slides and the project statement are available at
`https://github.com/ML-for-B-E/Optimisation`

Course outline

An introduction to optimization for machine learning

Bibliographical references for the class

This course is based on

- [Ravikumar and Singh, 2017] : a detailed up-to-date presentation of the main convex optimization algorithms for machine learning (level end of undergraduate, bac +3)
- [Minoux, 2008] : a classic textbook for optimization, written before the ML trend but still useful (level end of undergraduate / bac+3)
- [Bishop, 2006] : a reference book for machine learning with some pages on optimization (level end of undergraduate / bac+3)
- [Schmidt et al., 2007] : L1 regularization techniques (research article)
- [Sun, 2019] : review of optimization methods and good practices for tuning neural nets.

The content of these references will be simplified for this class.

Optimization = a quantitative formulation of decision

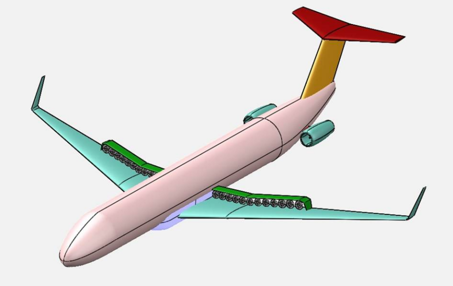
Optimization is a[1] way of mathematically modeling decision.

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

- x vector of decision parameters (variables) : dimensions, investment, tuning of a machine / program, . . .
- $f(x)$: decision cost x
- $\mathcal{S}$: set of possible values for x, search space

[1] non unique, incomplete when considering human beings or life

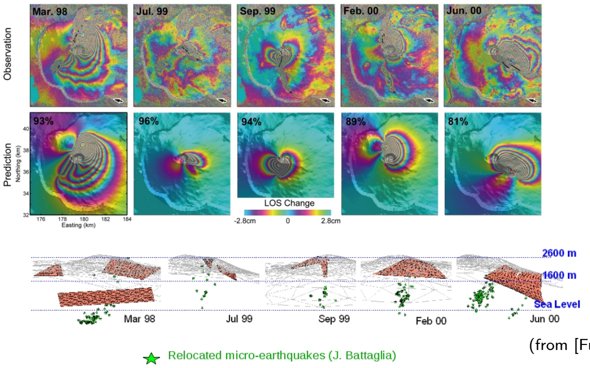
Optimization example: design



(from [Sgueglia et al., 2018])

x = aircraft parameters (here distributed electrical propulsion)
$f() = -1\times$ performance metric (aggregation of $-1\times$ range, cost, take-off length, . . .)
At the minimum, the design is "optimal".

Optimization example: model identification

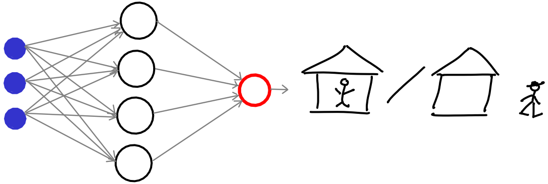


(from [Fukushima et al., 2010])

x = dike position, geometry, internal pressure
$f()$ = distance between measures (from RADARSAT-1 satellite) and model (boundary elements, non trivial computation)
At the minimum, the model best matches measurements and should correspond to the underground phenomenon.

Optimization example: neural net classification

Predict if a person stays at home or goes out based on longitude, latitude and temperature = a 2 classes classification problem.



x = neural network (NN) weights and biases
$f()$ = an error of the NN predictions (a cross-entropy error):

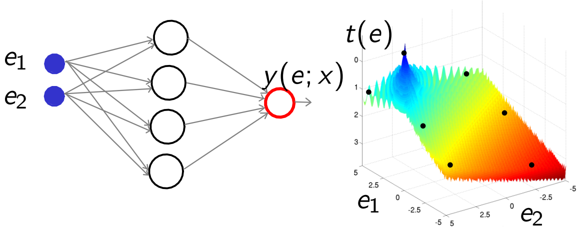
- e entries: e_1 longitude, e_2 latitude, e_3 temperature
- $t = 1$ if person stays, $t = 0$ otherwise
- Observed data set: (e^i, t^i), $i = 1, \ldots, N$
- $y(e; x)$: output of the NN, the probability that $t(e) = 1$
- $f(x) = -\sum_{i=1}^{N} \{t^i \log(y(e^i; x)) + (1 - t^i) \log(1 - y(e^i; x))\}$

- View the relationship between the entry e and the class t as probabilistic (generalizes deterministic functions): $t(e)$ is a Bernoulli variable with a given probability that $t(e) = 1$
- The NN models this probability: $y(e;x)$ is the probability that $t(e) = 1$, $1 - y(e;x)$ is the proba that $t(e) = 0$, $0 \leq y(e;x) \leq 1$.
- The probability of t knowing e can be written $y(e;x)^t \times (1 - y(e;x))^{1-t}$
- The likelihood of the N i.i.d observations is $\prod_{i=1}^N \left[y(e^i;x)^{t^i} \times (1 - y(e^i;x))^{1-t^i} \right]$, to be maximized
- The likelihood is turned into an error, to be minimized, by taking $-\log(\text{likelihood})$,

$$f(x) = -\sum_{i=1}^{N} \{ t^i \log(y(e^i;x)) + (1 - t^i) \log(1 - y(e^i;x)) \}$$

Optimization example: neural net regression

learn a function from a discrete limited set of observations



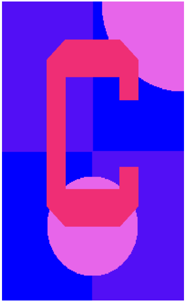
x = neural network (NN) weights and biases
$f()$ = an error of the NN predictions (sum-of-squares error):

- e entries, $t(e)$ target function to learn
- observed data set, "•" : $(e^i, t^i),\ i = 1, \ldots, N$
- $y(e;x)$: output of the NN, the expected value of $t(e)$
- $f(x) = 1/2 \sum_{i=1}^{N} (t^i - y(e^i;x))^2$

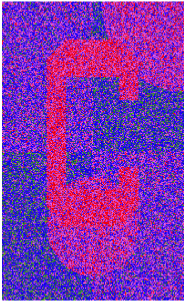
Optimization example: image denoising

$$\min_x f(x) \quad , \quad f(x) = \frac{1}{2} \sum_{i=1}^{N_{\text{pixels}}} (y_i - x_i)^2 + \lambda \sum_{i=1}^{N_{\text{pixels}}} \sum_{j \text{ near } i} |x_i - x_j|$$

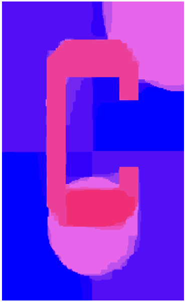
$\lambda \geq 0$ regularization constant



target image



noisy (observed) $= y_i$'s



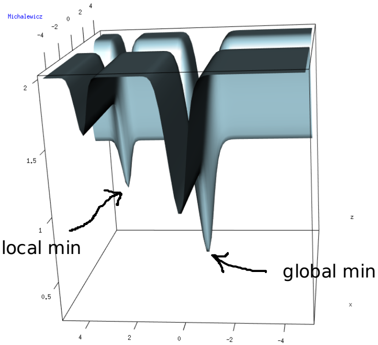
denoised (optimized) $= x^\star$

(from [Ravikumar and Singh, 2017])

Basic mathematical concepts for optimization

Local versus global optimum

$$\min_{x \in \mathcal{S} \subset \mathbb{R}^n} f(x)$$

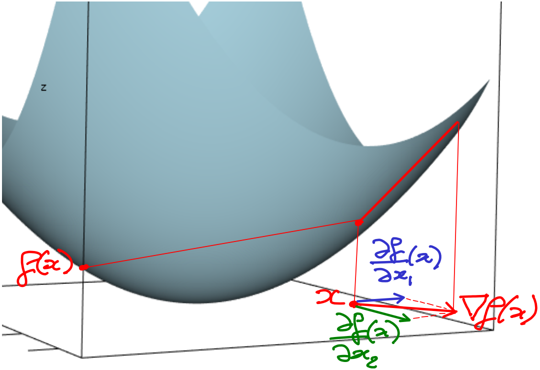


Python code to generate such a 3D plot given in the `Code` folder, `3D_plots.py`

Gradient of a function

Gradient of a function = direction of steepest ascent = vector of partial derivatives

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \dots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix}$$



Hessian of a function

It is the matrix of second derivatives,

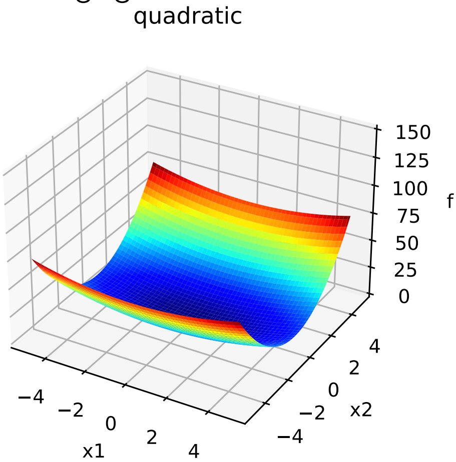
$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \ddots & \cdots \\ \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

$=$ the matrix of curvatures $=$ the gradient of the gradient.

Quadratic function and Hessian I

$$f(x) = \frac{1}{2}x^\top Hx \quad , \quad \nabla^2 f(x) = H$$

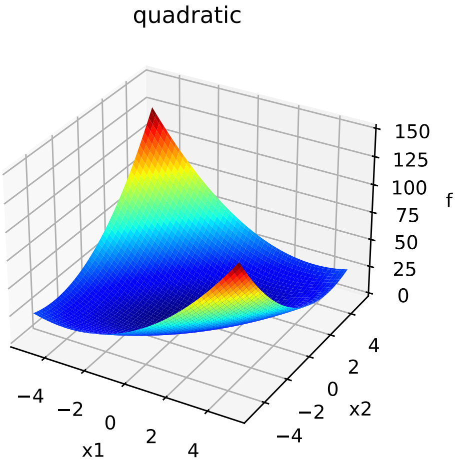
a good approximation to what happens on any function when converging



$$H = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

(guess the eigenvalues and eigenvectors)

Quadratic function and Hessian II



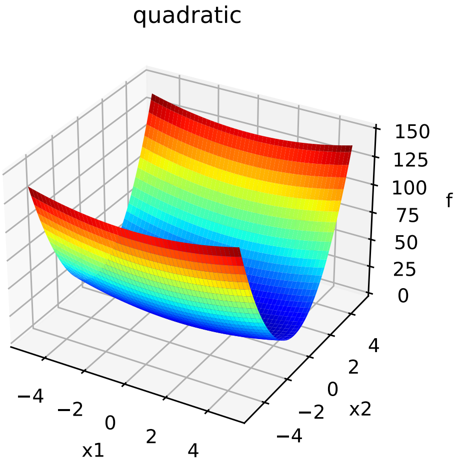
the same rotated by 45°

$$H = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix}$$

$$\text{eig.vect} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$$

$$\text{eig.val} = [1, 5]$$

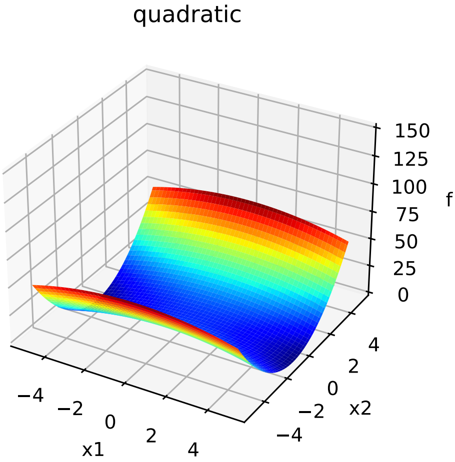
Quadratic function and Hessian III



increased curvature (condition number)

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}$$

Quadratic function and Hessian IV



Non positive definite Hessian

$$H = \begin{bmatrix} -1 & 0 \\ 0 & 5 \end{bmatrix}$$

what is the problem ?

Numerical approximation of the gradient

By forward finite differences

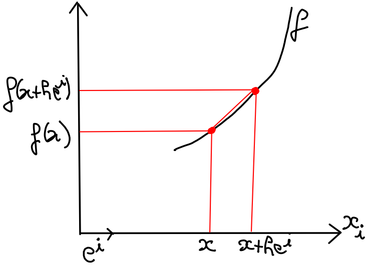
$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x+he^i) - f(x)}{h}$$

Proof: by Taylor,

$f(x + he^i) = f(x) + {he^i}^\top.\nabla f(x) + h^2/2{e^i}^\top \nabla^2 f(x + \rho he^i)e^i$, $\rho \in]0,1[$

$\partial f(x)/\partial x_i = \frac{f(x+he^i)-f(x)}{h} - h/2{e^i}^\top \nabla^2 f(x+\rho he^i)e^i$

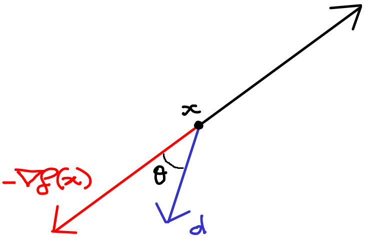
and make h very small $\square$



Other (better but more difficult to implement) schemes: central differences, automatic differentiation (e.g., in TensorFlow or PyTorch), (semi-)analytic differentiation (e.g., backpropagation in NN).

Descent direction

A search direction d which makes an acute angle with $-\nabla f(x)$ is a descent direction, i.e., for a small enough step, f is guaranteed to decrease!

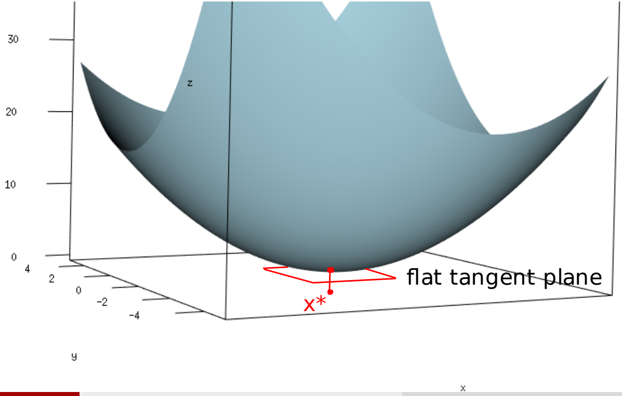


Proof: by Taylor, $\forall \alpha,\ \exists \epsilon \in [0,1]$ such that
$f(x+\alpha d) = f(x) + \alpha d^\top.\nabla f(x) + \frac{\alpha^2}{2} d^\top \nabla^2 f(x+\alpha\epsilon d) d$
$\lim_{\alpha \to 0^+} \frac{f(x+\alpha d)-f(x)}{\alpha} = d^\top.\nabla f(x) = -1 \times \|\nabla f(x)\| \cos(d, -\nabla f(x))$
is negative if the cosine is positive $\square$

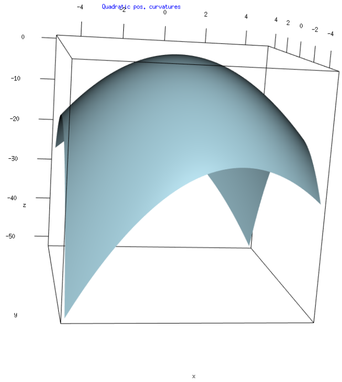
Necessary optimality condition (1)

A necessary condition for a differentiable function to have a minimum at $x^\star$ is that it is flat at this point, i.e., its gradient is null

$$x^\star \in \arg\min_{x \in \mathcal{S}} f(x) \Rightarrow \nabla f(x^\star) = 0$$

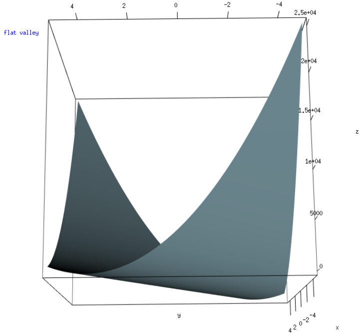


Necessary optimality condition (2)



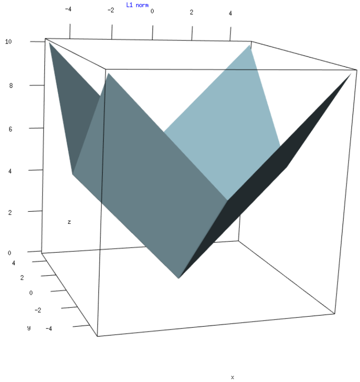
necessary is not sufficient (works with a max)

Necessary optimality condition (3)



$\nabla f(x^\star) = 0$ does not make $x^\star$ unique (flat valley)

Necessary optimality condition (4)



$\nabla f()$ not defined everywhere, example with L1 norm $= \sum_i^n |x_i|$

Course outline

An introduction to optimization for machine learning

Optimizers as iterative algorithms

We look for $\quad x^\star \in \arg\min_{x \in \mathcal{S}} f(x) \quad , \quad \mathcal{S} = \mathbb{R}^n$

- Except for special cases (e.g., convex quadratic problems), the solution is not obtained analytically through the optimality conditions ($\nabla f(x^\star) = 0$ + higher order conditions).
- We typically use iterative algorithms: x^{i+1} depends on previous iterates, $x^1, \ldots, x^i$ and their f's.
- Often calculating $f(x^i)$ takes more computation than the optimization algorithm itself.
- Qualities of an optimizer: robustness, speed of convergence. Have to strike a compromise between them.

Fixed step steepest descent algorithm (1)

Repeat steps along the steepest descent direction, $-\nabla f(x^t)$ [Cauchy, 1847, Curry, 1944].
The size of the steps is proportional to the gradient norm.

Require: $f()$, $\bar{\alpha} \in]0,1]$, x^1, ϵ^{step}, ϵ^{grad}, $i^{\max}$
  $i \leftarrow 0$, $f^{\text{bestSoFar}} \leftarrow$ `max_double`
  repeat
    $i \leftarrow i + 1$
    calculate $f(x^i)$ and $\nabla f(x^i)$
    if $f(x^i) < f^{\text{bestSoFar}}$ **then**
      update $x^{\text{bestSoFar}}$ and $f^{\text{bestSoFar}}$ with current iterate
    end if
    direction: $d^i = -\nabla f(x^i)/\|\nabla f(x^i)\|$
    step: $x^{i+1} = x^i + \bar{\alpha}\|\nabla f(x^i)\| d^i$
  until $i > i^{\max}$ **or** $\|x^i - x^{i-1}\| \leq \epsilon^{\text{step}}$ **or** $\|\nabla f(x^i)\|/\sqrt{n} \leq \epsilon^{\text{grad}}$
  return $x^{\text{bestSoFar}}$ and $f^{\text{bestSoFar}}$

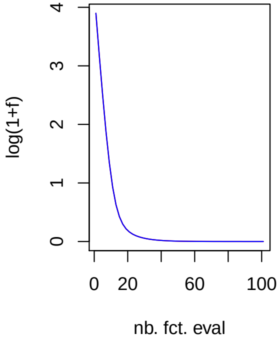
(code organization)

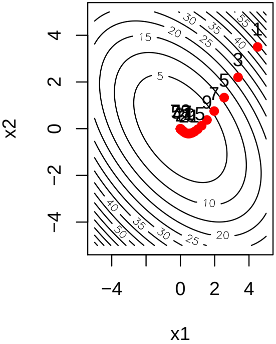
- `main_optim.py`: main script for starting the descent algorithms.
- `gradient_descent.py`: gradient-based descent algorithms; the current gradient fixed-step version, and the ones coming up (other direction, with a line search).
- `random_search.py`: a random search algorithm.
- `test_functions.py`: a collection of test functions.
- `3D_plots.py`: plots a 2 dimensional function in a 3D dynamic plot + contour plot.
- `optim_utilities.py`: additional routines.

Fixed step steepest descent algorithm (2)

- The choice of the step size factor $\bar{\alpha}$ is critical : the steeper the function, the smaller $\bar{\alpha}$. Default value $= 0.1$
- The true code (cf. `gradient_descent.R`) is a bit longer because it is necessary to record the points visited.

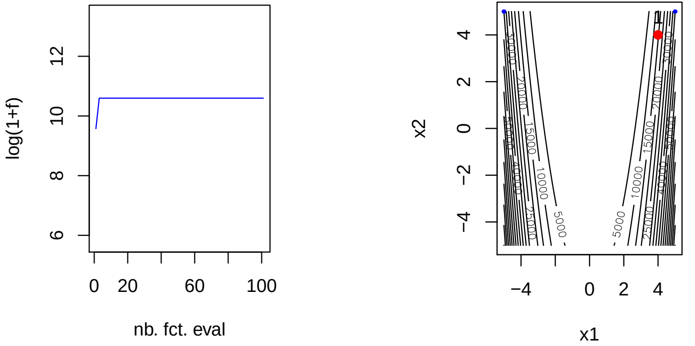
$f(x) = 1/2x^\top Hx$, H positive definite





Fixed step steepest descent algorithm (3)

$\bar{\alpha} = 0.1$ on $f(x) =$ Rosenbrock (banana shaped) function in $d = 2$ dimensions, example of divergence:



$$x^\star = (1,1) \ , \ f(x^\star) = 0$$

Descent with line search

At each iteration, search for the best step size in the descent[2] direction d^i (which for now is $-\nabla f(x^i)/\|\nabla f(x^i)\|$ but it is general). Same algorithm as before, just change the **step** instruction:

Require: ...
  initializations but no α now ...
  repeat
    increment i, calculate $f(x^i)$ and $\nabla f(x^i)$ ...
    direction: $d^i = -\nabla f(x^i)/\|\nabla f(x^i)\|$ or any other **descent** direction
    step: $\alpha^i = \arg\min_{\alpha>0} f(x^i + \alpha d^i)$
$$x^{i+1} = x^i + \alpha^i d^i$$
  until stopping criteria
  return best so far

---

[2] if d^i is not a descent direction, $-d^i$ is. Proof left as exercise.

Approximate line search (1)

Notation: during line search i,

$$
\begin{aligned}
x &= x^i + \alpha d^i \\
f(\alpha) &= f(x^i + \alpha d^i) \\
\frac{df(0)}{d\alpha} &= \sum_{j=1}^{n} \frac{\partial f(x^i)}{\partial x_j} \frac{\partial x_j}{\partial \alpha} = \sum_{j=1}^{n} \frac{\partial f(x^i)}{\partial x_j} d^i_j = \nabla f(x^i)^\top . d^i
\end{aligned}
$$

In practice, perfectly optimizing for α^i is too expensive and not useful $\Rightarrow$ approximate the line search by a sufficient decrease condition:

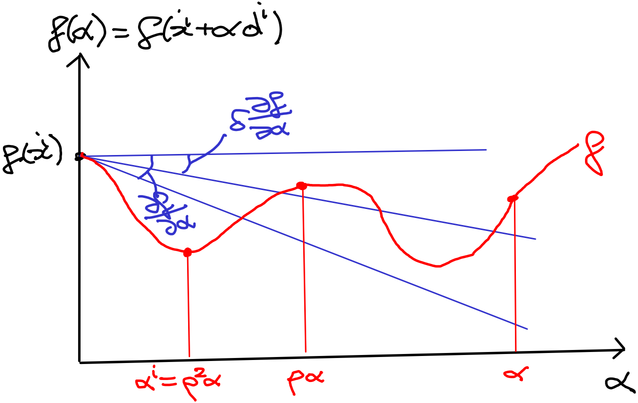
$$\text{find } \alpha^i \text{ such that } f(x^i + \alpha^i d^i) < f(x^i) + \delta \alpha^i \nabla f(x^i)^\top . d^i$$

where $\delta \in [0, 1]$, i.e., achieve a δ proportion of the progress promised by order 1 Taylor expansion.

Approximate line search (2)

Sufficient decrease condition rewritten with line search notation:

$$\text{find } \alpha^i \text{ such that } f(\alpha^i) < f(x^i) + \delta\alpha^i \frac{df(0)}{d\alpha}$$



Approximate line search (3)

At iteration i:

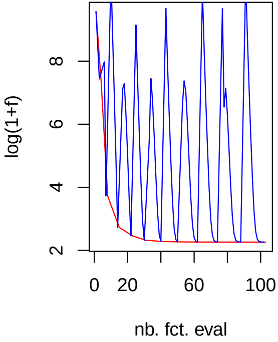
Backtracking line search (Armijo)

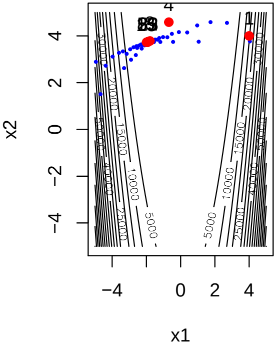
Require: d^i a descent direction, x^i, $\delta \in [0,1]$, $\rho \in]0,1[$, $C > 0$
(defaults: $\delta = 0.1$, $\rho = 0.5$, $C = 1$)
initialize step size: $\alpha = \max(C \times \|\nabla f(x^i)\| \ , \ \sqrt{n}/100)$
while $f(x^i + \alpha d^i) \geq f(x^i) + \delta\alpha\nabla f(x^i)^\top d^i$ **do**
  decrease step size: $\alpha \leftarrow \rho \times \alpha$
end while
return $\alpha^i \leftarrow \alpha$

From now on, use line search, and the number of calls to f is no longer equal to the iteration number since many function calls can be done during a line search within a single iteration.

Approximate line search (4)

Look at what line search does to $f(x) =$ Rosenbrock where fixed step size diverged

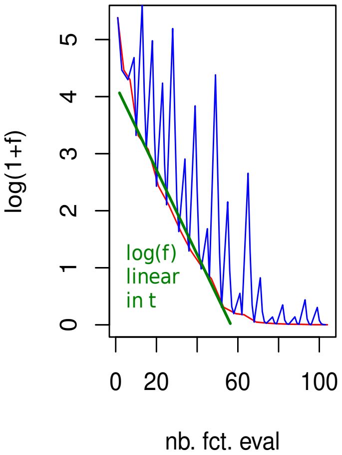




Better, but not perfect: oscillations make progress very slow.

Gradient convergence speed

$f(x) = \frac{1}{2}x^\top Hx$ in $n = 10$ dimensions, $H > 0$, not aligned with the axes, condition number $= 10$.



Empirically (for proofs and more info cf. [Ravikumar and Singh, 2017]): on convex and differentiable functions, gradient search with line search progresses at a speed such that $f(x^t) \propto \xi\gamma^t$ where $\gamma \in [0,1[$. Equivalently, to achieve $f(x^t) < \varepsilon$, $t > \mathcal{O}(\log(1/\varepsilon))$

$\log f(x^t) \propto t\log(\gamma) + \log(\xi) \;\Rightarrow\; \log(\gamma) < 0$ slope of the green curve.

$\xi\gamma^t < \varepsilon \Leftrightarrow t > \frac{\log(\varepsilon) - \log(\xi)}{\log(\gamma)} = \frac{-1}{\log(\gamma)}\log(\xi/\varepsilon)$

$\Rightarrow \; t > \mathcal{O}(\log(1/\varepsilon))$.

Gradient descent oscillations

Perfect line search solves

$$\alpha^i = \arg\min_{\alpha>0} f(\alpha) \quad \text{where} \quad f(\alpha) = f(x^i + \alpha d^i)$$

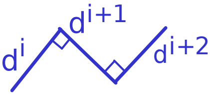
Necessary conditions of optimal step size:

$$\frac{df(\alpha^i)}{d\alpha} = \sum_{j=1}^{n} \frac{\partial f(x^i + \alpha^i d^i)}{\partial x_j} \frac{\partial x_j}{\partial \alpha} = \nabla f(x^{i+1})^\top . d^i = 0$$

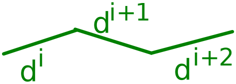
If the direction is the gradient,

$$-{d^{i+1}}^\top . d^i = 0 \quad \text{i.e. } d^{i+1} \text{ and } d^i \text{ perpendicular}$$

gradient
does



less oscillations
seems better



Course outline

An introduction to optimization for machine learning

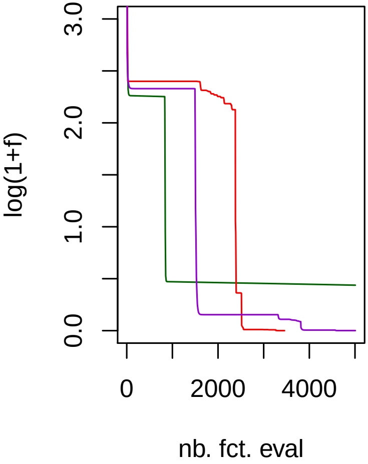
Changing the search direction

Improved gradient searches slightly (but importantly) change the search direction from minus the gradient:

- Momentum : search direction = minus gradient moved a bit towards previous search direction.
- Nesterov [Nesterov, 1983] : search direction = momentum direction with an anticipation about point of the next gradient.
- Adam [Kingma and Ba, 2014] : state-of-the-art in deep learning. Stochastic gradient method with independent adaptation of each variable based on momentum.

Comparison of methods (1)

Rosenbrock, $d = 2$: ability to handle curved ravines



green=gradient, red=momentum, violet=NAG

Course outline

An introduction to optimization for machine learning

A word about constraints

$$\begin{cases} \min_{x \in \mathcal{S}} f(x) \quad , \quad \mathcal{S} = \mathbb{R}^n \\ \text{such that } g_i(x) \leq 0 \quad , \quad i = 1, m \end{cases}$$

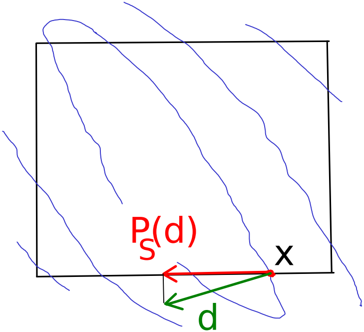
Bound constraints

$\mathcal{S}$ is an hypercube of $\mathbb{R}^n$, $\mathcal{S} = [LB, UB] \subset \mathbb{R}^n$.

It could be described by constraints, $g_{2i-1}(x) := LB_i - x_i \leq 0$, $g_{2i}(x) := x_i - UB_i \leq 0$, $i = 1, \ldots, d$ but these constraints are so simple that they can be directly handled by projection.

If x^i is at a bound and the search direction d^i takes it outside[a] $\mathcal{S} = [LB, UB]$, project the search direction vector onto the active bound.

Exercise: how to code this?



[a]This can even happen for a convex function in a convex $\mathcal{S}$, as the drawing shows.

Constraints handling by penalizations (1)

$$\begin{cases} \min_{x \in \mathcal{S} \in \mathbb{R}^d} f(x) \\ \text{such that } g(x) \leq 0 \end{cases}$$

(vector notation for the constraints)

We give two techniques to aggregate f and the g_i's into a new objective function (to minimize).

External penalty function: penalize points that do not satisfy the constraints

$$f_r(x) = f(x) + r\,[\max(0, g(x))]^2 \quad , \quad r > 0$$

- Pros: simple, $\nabla f_r()$ continuous accross the constraint boundary (if f and g are)
- Cons: Convergence by the infeasible domain (hence external), need to find r large enough to reduce infeasibility, but not too large because of numerical issue (high curvature accross constraint)

Constraints handling by penalizations (2)

Lagrangian: for problems without duality gap[3], e.g., convex problems, there exists Lagrange multipliers $\lambda^\star$ such that

$$x^\star \in \arg\min_{x \in \mathcal{S}} L(x; \lambda^\star)$$
$$\text{where } L(x; \lambda^\star) := f(x) + \lambda^\star g(x)$$

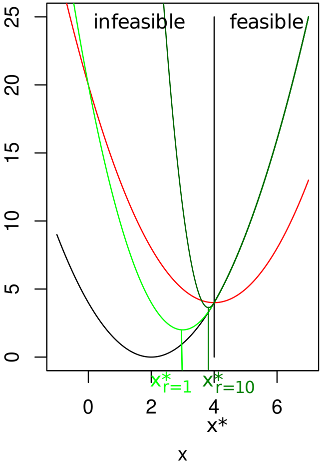
The Lagrangian $L(; \lambda^\star)$ is (when no duality gap) a valid penalty function.

- Pros: duality provides a way to calculate $\lambda^\star$, yields a feasible solution.
- Cons: estimating $\lambda^\star$ has a numerical cost. For most problems with local optima there is a duality gap $\Rightarrow$ rely on augmented Lagrangians[4].

[3] cf. duality, out of scope for this course

Constraints handling by penalizations (3)

Example: $f(x) = (x-2)^2$, $g(x) = 4 - x \leq 0$, $x^* = 4$, convex problem



f and g in black, $L(x;\lambda^\star = 4)$ in red, exterior penalty $f_r()$ with $r = 1$ and 10 in light and dark green, respectively.

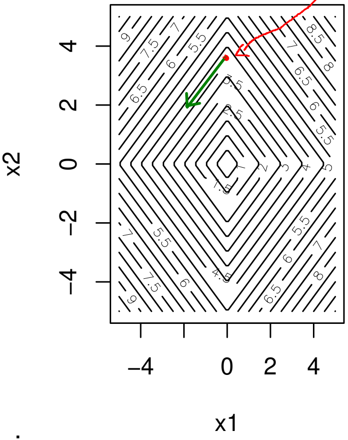
The Lagrangian is a valid penalty here.

As r grows, $x_r^\star \to x^\star$ but the curvature of $f_r()$ increases.

Comments on gradient based descent algorithms

Use on nondifferentiable functions: theoretically may converge at a point which is not a minimum even on convex functions (e.g., if an iterate is at a kink). This rarely happens in practice. Try function $f(x) = \sum_{i=1}^{n} |x_i|$ ("L1norm") with the code.

forward finite difference estimation to the gradient: no progress, stops at



Main flaw: gets trapped in local minima.

Restarted local searches

Simple principle: restart descent searches from initial points chosen at random.
Use randomness to make deterministic descent searches more robust.
A mix between 2 extremes: local vs global, line search vs volume search, specific (to unimodal differentiable functions) vs without assumption, efficient vs very slow.
Simplistic implementation (cf. code provided)

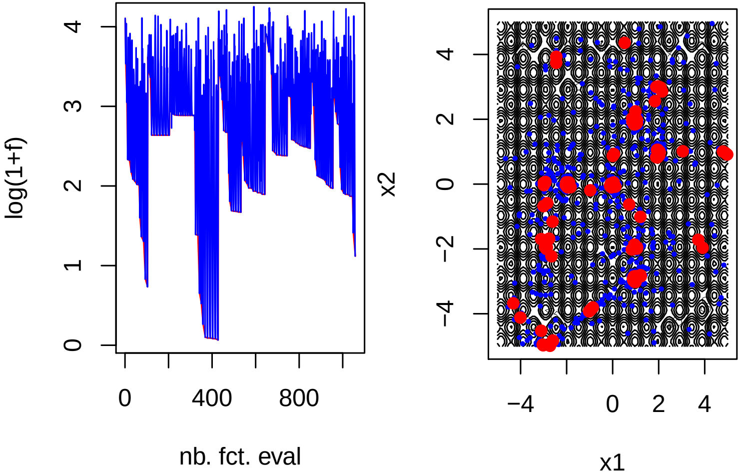
at a cost × `nb_restarts`:

Require: `budget`, `nb_restarts`

```
for i in 1 to nb_restarts do
  xinit <- runif(n=d,min=LB,max=UB)
  res<-gradient_descent(xinit,budget=budget/nb_restarts)
  update global search results
end for
```

Restarted local searches: example

Execution of the `restarted_descent` file.
`fun <-rastrigin, d<-2, budget<-1000, nb_restart<-10`:



Application to neural network

The practical applications are available through the `project` notebook on github, cf. `https://github.com/ML-for-B-E/Optimisation/blob/main/notebook/project.ipynb`

Conclusions

- Numerical optimization is a fundamental technique for quantitative decision making, statistical modeling, machine learning, . . .
- The enthousiasm for machine learning has led to very many optimization algorithms which we did not discuss in this introductory course: see for example [Sun et al., 2019, Sra et al., 2012].
- Also not covered yet emerging: Bayesian optimization for hyper-parameters tuning (regularization constants, number of NN layers, types of neurons, parameters of the gradient based algorithms) [Snoek et al., 2012].

References I

Bishop, C. M. (2006).
Pattern recognition and machine learning.

Cauchy, A. L. (1847).
Méthode générale pour la résolution des systèmes d'équations simultanées.
Comp. Rend. Sci. Paris, 25(1847):536–538.

Curry, H. B. (1944).
The method of steepest descent for non-linear minimization problems.
Quarterly of Applied Mathematics, 2(3):258–261.

Fukushima, Y., Cayol, V., Durand, P., and Massonnet, D. (2010).
Evolution of magma conduits during the 1998–2000 eruptions of piton de la fournaise volcano, réunion island.
Journal of Geophysical Research: Solid Earth, 115(B10).

Kingma, D. P. and Ba, J. (2014).
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980.

Minoux, M. (2008).
Programmation mathématique. Théorie et algorithmes.
Lavoisier.

References II

Nesterov, Y. (1983).
A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$.
In *Doklady an USSR*, volume 269, pages 543–547.

Ravikumar, P. and Singh, A. (2017).
Convex optimization.
http://www.cs.cmu.edu/~pradeepr/convexopt/.

Schmidt, M., Fung, G., and Rosales, R. (2007).
Fast optimization methods for l1 regularization: A comparative study and two new approaches.
In *European Conference on Machine Learning*, pages 286–297. Springer.

Sgueglia, A., Schmollgruber, P., Bartoli, N., Atinault, O., Benard, E., and Morlier, J. (2018).
Exploration and sizing of a large passenger aircraft with distributed ducted electric fans.
In *2018 AIAA Aerospace Sciences Meeting*, page 1745.

Snoek, J., Larochelle, H., and Adams, R. P. (2012).
Practical bayesian optimization of machine learning algorithms.
Advances in neural information processing systems, 25.

References III

Sra, S., Nowozin, S., and Wright, S. J. (2012).
Optimization for machine learning.
Mit Press.

Sun, R. (2019).
Optimization for deep learning: theory and algorithms.
arXiv preprint arXiv:1912.08957.

Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019).
A survey of optimization methods from a machine learning perspective.
IEEE transactions on cybernetics, 50(8):3668–3681.