An introduction to optimization for machine learning

Rodolphe Le Riche[1], Dédji Brian Whannou[2], Espéran Padonou[3]

[1] CNRS LIMOS at Mines Saint-Etienne, France
[2] KPMG France
[3] Fondation Vallet

July 2021
Ecole d'Eté en Intelligence Artificielle
fondation Vallet
Cotonou, Bénin

Course outline

An introduction to optimization for machine learning

Objectives of this 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 R/python from scratch.
- Limitation: the algorithms are not exactly those used in state-of-the-art deep learning, but the main concepts will be presented.

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 for tuning neural nets, gradient backpropagation (research article)

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

Optimization = a quantitative formulation of decision

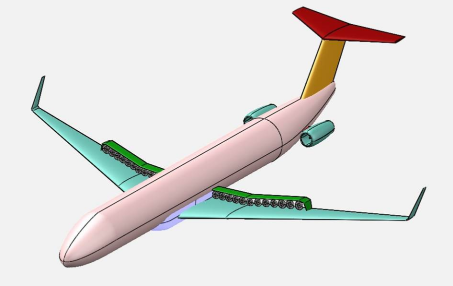
Optimization is a[1] way of mathematically modeling decision.

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

- 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

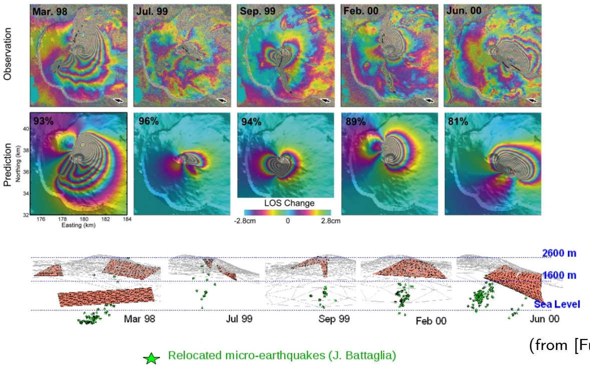
Optimization example: design



(from [Sgueglia et al., 2018])

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

Optimization example: model identification

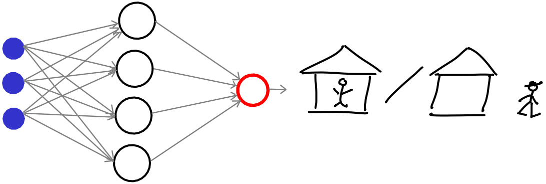


(from [Fukushima et al., 2010])

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):

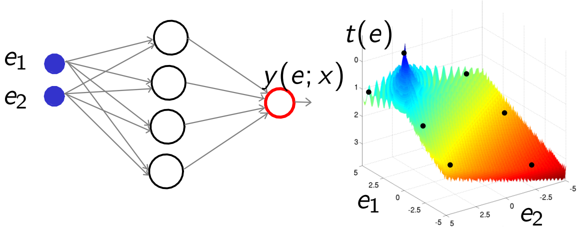
- e entries: e_1 longitude, e_2 latitude, e_3 temperature
- $t = 1$ if person stays, $t = 0$ otherwise
- Observed data set: (e^i, t^i), $i = 1, \ldots, N$
- $y(e; x)$: output of the NN, the probability that $t(e) = 1$
- $f(x) = -\sum_{i=1}^{N} \{t^i \log(y(e^i; x)) + (1 - t^i) \log(1 - y(e^i; x))\}$

- 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 $y(e;x)^t + (1 - y(e;x))^{1-t}$
- The likelihood of the N i.i.d observations is $\prod_{i=1}^N \left[y(e^i;x)^{t^i} + (1 - y(e^i;x))^{1-t^i} \right]$, 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



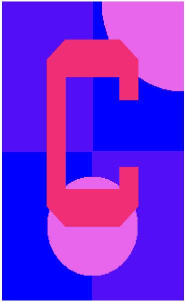
x = neural network (NN) weights and biases
$f()$ = an error of the NN predictions (sum-of-squares error):

- 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 $t(e)$
- $f(x) = 1/2 \sum_{i=1}^{N} (t^i - y(e^i;x))^2$

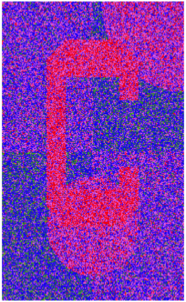
Optimization example: image denoising

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

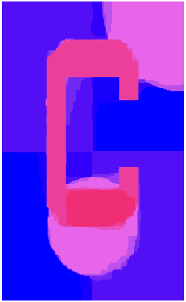
$\lambda \geq 0$ regularization constant



target image



noisy (observed) $= y_i$'s

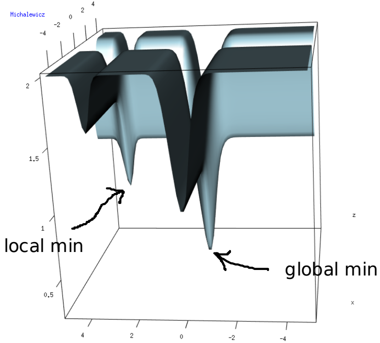


denoised (optimized) $= x^\star$

(from [Ravikumar and Singh, 2017])

Local versus global optimum

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

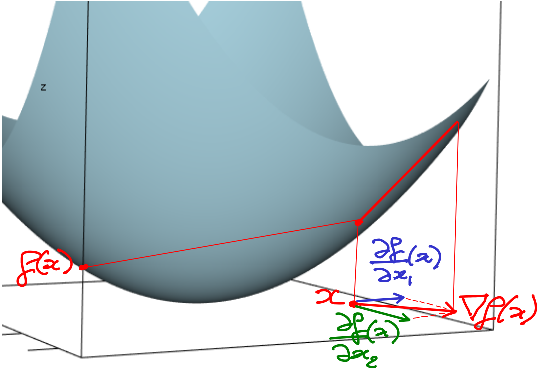


R code to generate the plot given in the project folder

Gradient of a function

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

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



Numerical approximation of the gradient

By forward finite differences

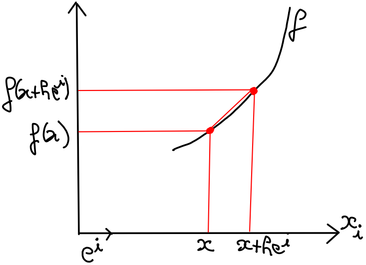
$$\frac{\partial f}{\partial x_i} f(x) \approx \frac{f(x + he^i) - f(x)}{h}$$

Proof: by Taylor,

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

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

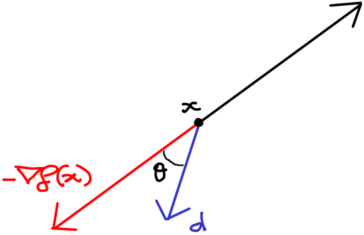
and make h very small $\square$



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!

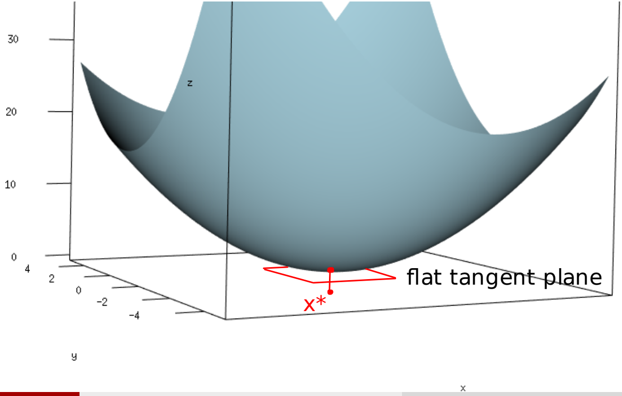


Proof: by Taylor, $\forall \alpha \leq 0$, $\exists \epsilon \in [0,1]$ such that
$f(x+\alpha d) = f(x) + \alpha d^\top . \nabla f(x) + \frac{\alpha^2}{2} d^\top \nabla^2 f(x + \alpha\epsilon d) d$
$\lim_{\alpha \to 0^+} \frac{f(x+\alpha d) - f(x)}{\alpha} = d^\top . \nabla f(x) = -1 \times \|\nabla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

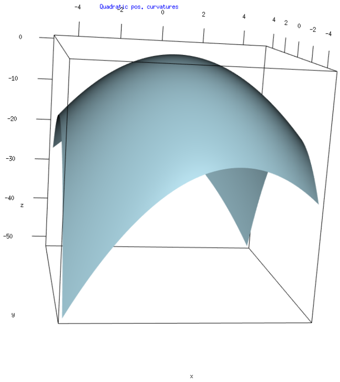
Necessary optimality condition (1)

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

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

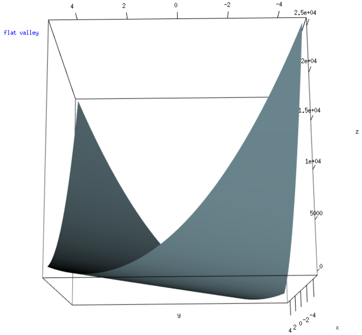


Necessary optimality condition (2)



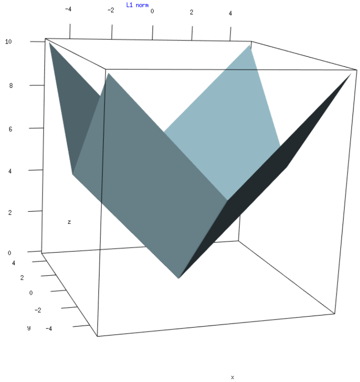
necessary is not sufficient (works with a max)

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 $= \sum_i^d |x_i|$

Course outline

An introduction to optimization for machine learning

Optimizers as iterative algorithms

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

- 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. Often have to strike a compromise between them.

Fixed step steepest descent algorithm (1)

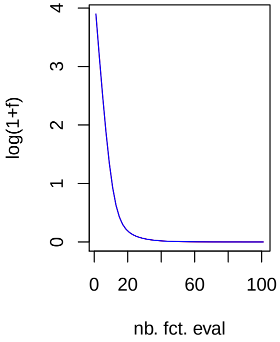
Repeat steps along the steepest descent direction, $-\nabla f(x^t)$.
The size of the steps is proportional to the gradient norm.

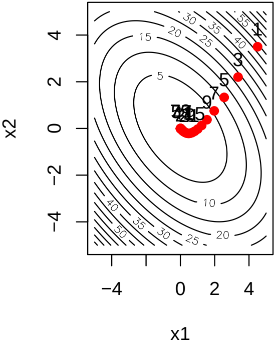
Require: $f()$, $\alpha \in]0,1]$, x^1, ϵ^{step}, ϵ^{grad}, $i^{\max}$
  $i \leftarrow 0$, $f^{\text{best so far}} \leftarrow$ `max_double`
  repeat
    $i \leftarrow i+1$
    calculate $f(x^i)$ and $\nabla f(x^i)$
    if $f(x^i) < f^{\text{best so far}}$ **then**
      update $x^{\text{best so far}}$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

Fixed step steepest descent algorithm (2)

- The choice of the step size factor α is critical : the steeper the function, the smaller α. Default value $= 0.1$
- The true code (cf. project) is much longer and filled with instructions for reporting the points visited and doing plots afterwards.

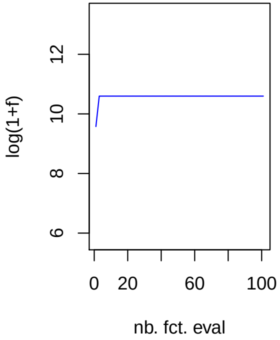
$f(x) = 1/2x^\top Hx$, H positive definite

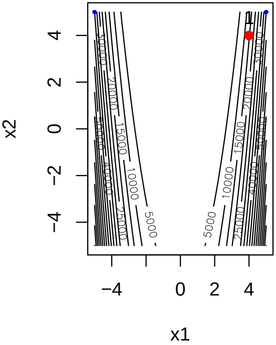




Fixed step steepest descent algorithm (3)

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





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,

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

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

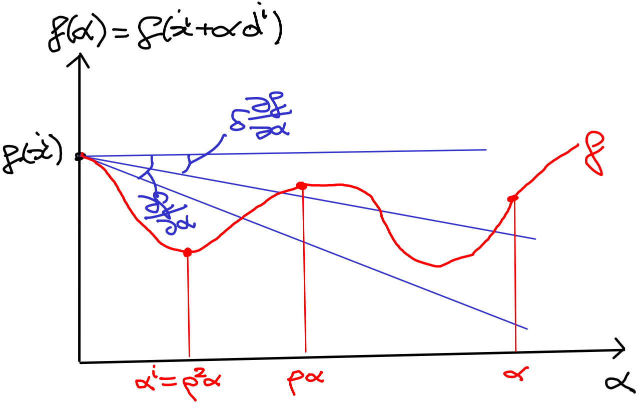
$$
\text{find } \alpha^i \text{ such that } f(x^i + \alpha^i d^i) < f(x^i) + \delta \alpha^i \nabla f(x^i)^\top . d^i
$$

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:

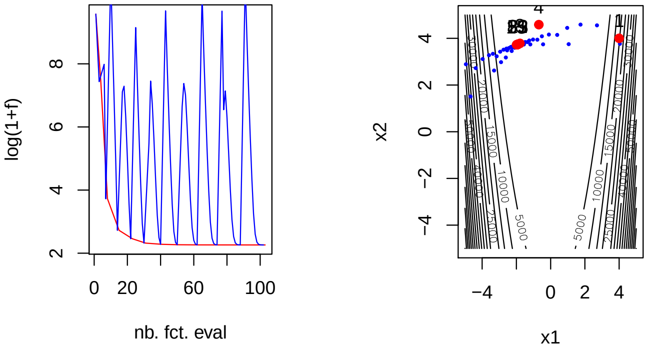
Backtracking line search (Armijo)

Require: d 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: $\alpha = \max(C \times \|\nabla f(x^i)\| \ , \ \sqrt{d}/100)$
while $f(x^i + \alpha d^i) \geq f(x) + \delta \alpha \nabla f(x^i)^\top . d$ **do**
decrease step size: $\alpha \leftarrow \rho \times \alpha$
end while
return $\alpha^i \leftarrow \alpha$

Note: 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)

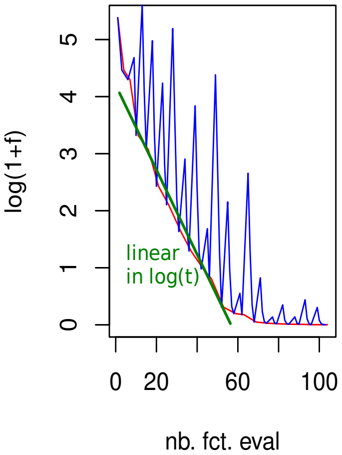
Look at what line search does to $f(x) =$ Rosenbrock where fixed step size diverged



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

Gradient convergence speed

$f(x) = \frac{1}{2}x^\top Hx$ in $d = 10$ dimensions, $H \geq 0$, 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

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

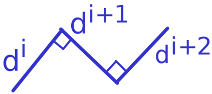
Necessary conditions of optimal step size:

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

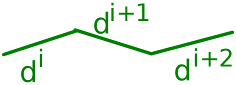
If the direction is the gradient,

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

gradient does



less oscillations seems better



Course outline

An introduction to optimization for machine learning

Gradient with momentum

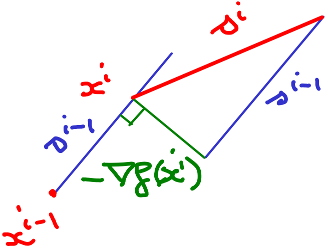
Recall fixed step gradient descent,

$$x^{i+1} = x^i + \alpha s^i \quad \text{where} \quad s^i = -\nabla f(x^i)$$

s^i, the step, corrected by a fixed or optimized (line search) α.
Introduce a momentum (i.e., a memory) in the search step,

$$s^i = -\nabla f(x^i) + \beta s^{i-1}$$

where[3] $\beta = 0.9$.
This should contribute to avoid the oscillations occuring at the botton of valleys:



[3] alternatively, for iteration varying momentum, $\beta^i = (i-2)/(i+1)$

Nesterov accelerated gradient (NAG)

(work in progress)

Course outline

An introduction to optimization for machine learning

Bound constraints

(work in progress)

Constraints handling by penalization

(work in progress)

Comments on gradient based descent algorithms

(work in progress)flaws: no convergence on nondifferentiable functions, gets trapped in local minima

Restarted local searches

(work in progress)(make a simple flow chart)

Conclusions

- L'optimisation numérique est une technique fondamentale associée à la décision optimale et à la modélisation statistique (machine learning).
- Avec l'enthousiasme autour du machine learning, de nombreux algorithmes ont été conçus que nous n'avons pas couverts ici: l'optimisation bayésienne (Bayesian optimization) pour le réglage des hyper-paramètres (paramètres de régularisation, nombre de couches du réseau de neurone, type de neurones, paramètres de l'algorithme d'optimisation des poids).

References I

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

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).

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

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.

References II

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.

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