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

Foreword

This course was first given during a summer school on AI in Cotonou, Benin, July-Aug. 2021. The school was organized by the Benin Excellence NGO and the Vallet Foundation (cf.

`https://www.fondationdefrance.org/fr/`

`au-benin-une-universite-dete-impulsee-par-la-fondation-vallet`).

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

The code, the slides and the project statement are available at `https://github.com/rleriche/Optimisation_AI_Cotonou2021`

Course outline

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

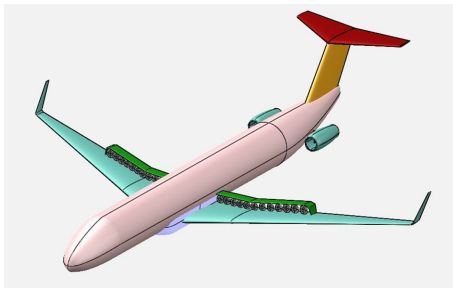Optimization is a[1] way of mathematically modeling decision.

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

- 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.
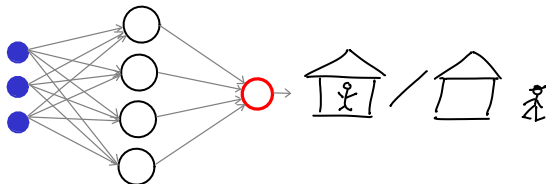
x = neural network (NN) weights and biases

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

(a word on the classification cross-entropy error)

- 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, "$\cdot$" : $(e^i, t^i), \ i = 1, \dots, N$
- $y(e; x)$: output of the NN, the expected value of $t(e)$
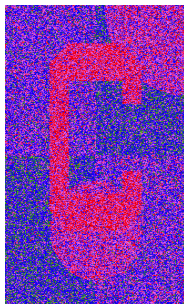- $f(x) = 1/2 \sum_{i=1}^{N} (t^i - y(e^i; x))^2$

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

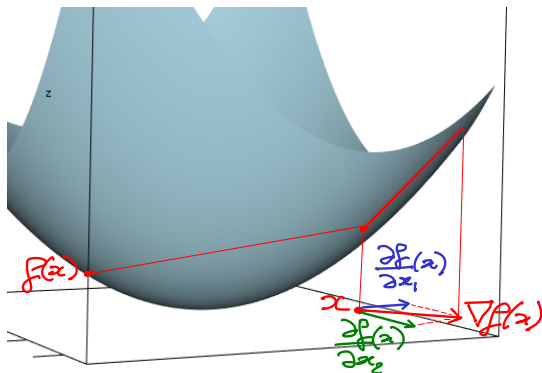$$\min_{x \in \mathcal{S} \subset \mathbb{R}^d} f(x)$$
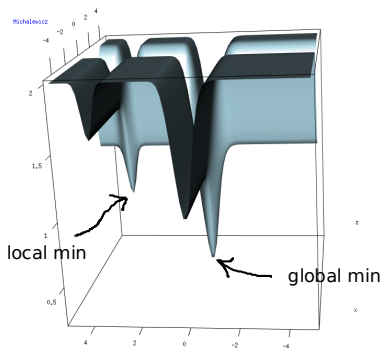
R code to generate such a 3D plot given in the project folder,
3Dplots.R

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

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

Proof: by Taylor,

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

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

and make h very small □

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!
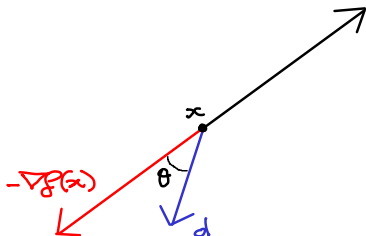
Proof: by Taylor, $\forall \alpha \leq 0, \ \exists \epsilon \in [0, 1]$ such that

$f(x + \alpha d) = f(x) + \alpha d^\top . \nabla f(x) + \frac{\alpha^2}{2} d^\top \nabla^2 f(x + \alpha \epsilon d) d$

$\lim_{\alpha \to 0^+} \frac{f(x + \alpha d) - f(x)}{\alpha} = d^\top . \nabla f(x) = -1 \times \|\nabla f(x)\| \cos(d, -\nabla f(x))$

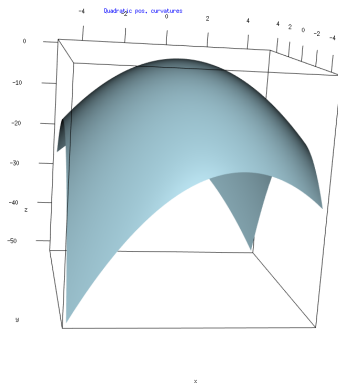is negative if the cosine is positive $\square$

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

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

flat tangent plane

x^*

Necessary optimality condition (2)

Quadratic pos. curvatures
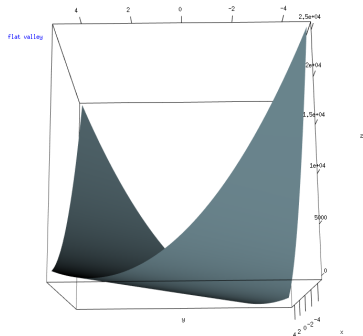
necessary is not sufficient (works with a max)

Necessary optimality condition (3)

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

$\nabla f()$ 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

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, $-\nabla f(x^t)$ [Cauchy et al., 1847].
The size of the steps is proportional to the gradient norm.

Require: $f()$, $\bar{\alpha} \in]0, 1]$, x^1, ϵ^{step}, ϵ^{grad}, $i^{\max}$
  $i \leftarrow 0$, $f^{\text{best so far}} \leftarrow$ `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: $x^{i+1} = x^i + \bar{\alpha}\|\nabla f(x^i)\|d^i$
  until $i > i^{\max}$ **or** $\|x^i - x^{i-1}\| \leq \epsilon^{\text{step}}$ **or** $\|\nabla f(x^i)\| \leq \epsilon^{\text{grad}}$
 **return** $x^{\text{best so far}}$ and $f^{\text{best so far}}$

(code organization)

- `mainOptim.R`: main script for starting the descent algorithms.
- `gradient_descent.R`: gradient-based descent algorithms; the current gradient fixed-step version, and the ones coming up.
- `line_searches.R`: line search routine.
- `test_functions.R`: a collection of test functions.
- `3Dplots.R`: plots a 2 dimensional function in a 3D dynamic plot + contour plot.
- `utilities_optim.R`: additional routines.
- `restarted_descent.R`: program to restart the descent search + associated vizualizations.
- `neural_net.R`: a feedforward network coded from scratch.
- `main_neural_net.R`: main script for learning a neural network with the optimization functions.

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 much longer and filled with instructions for reporting the points visited and doing plots afterwards.

$f(x) = 1/2 x^\top H x$, H positive definite
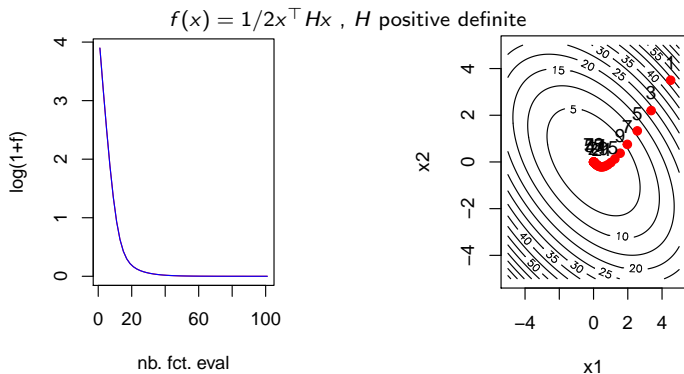
Fixed step steepest descent algorithm (3)

$\bar{\alpha} = 0.1$ on $f(x) =$ Rosenbrock (banana shaped) function in $d = 2$ dimensions, example of divergence:
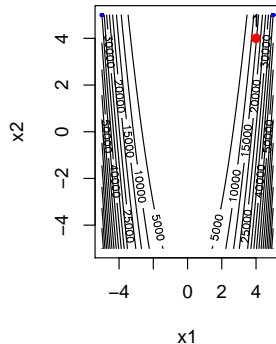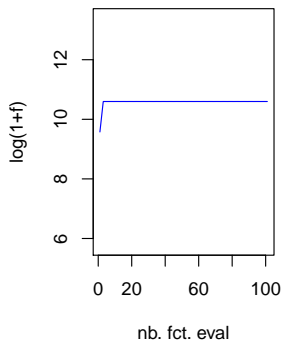
$$x^\star = (1, 1) \ , \ f(x^\star) = 0$$

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,

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

In practice, perfectly optimizing for α^i is too expensive and not useful $\Rightarrow$ approximate the line search by a sufficient decrease condition:
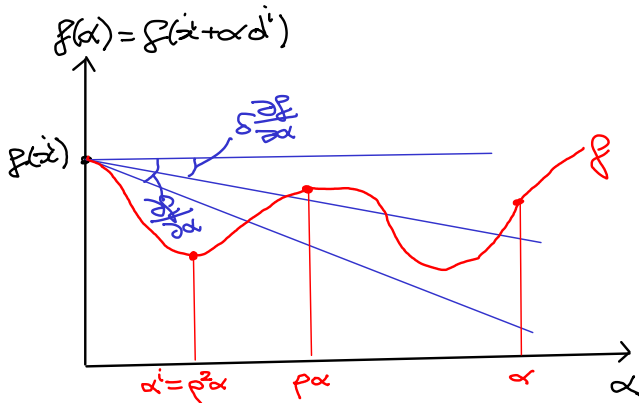
$$
\text{find } \alpha^i \text{ such that } f(x^i + \alpha^i d^i) < f(x^i) + \delta \alpha^i \nabla f(x^i)^\top . d^i
$$

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:

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

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 $\alpha^i \leftarrow \alpha$

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.
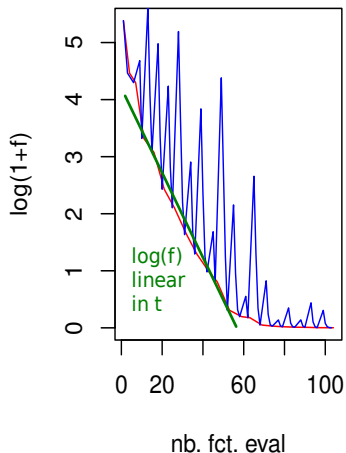
Approximate line search (4)

Look at what line search does to $f(x) =$ Rosenbrock where fixed step size diverged
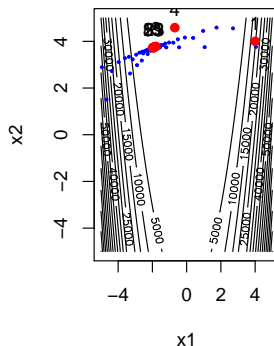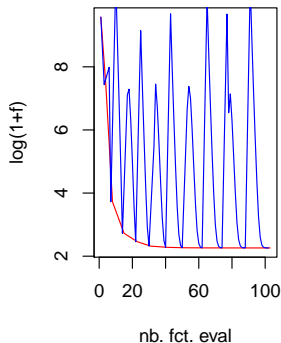
Better, but not perfect: oscillations make progress very slow.

Gradient convergence speed

$f(x) = \frac{1}{2} x^{\top} H x$ in $d = 10$ 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

$$\alpha^i = \arg\min_{\alpha > 0} f(\alpha) \ \text{ where } \ f(\alpha) = f(x^i + \alpha d^i)$$
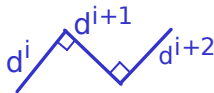
Necessary conditions of optimal step size:

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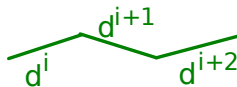
If the direction is the gradient,

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

gradient
does

d^i d^{i+1} d^{i+2}

less oscillations
seems better

d^i d^{i+1} d^{i+2}

Course outline

An introduction to optimization for machine learning

Gradient with momentum (1)

Recall fixed step gradient descent,

$$x^{i+1} = x^i - \bar{\alpha}\nabla f(x^i) \quad \text{i.e.,} \quad s^i = x^{i+1} - x^i = -\bar{\alpha}\nabla f(x^i)$$

s^i, the step, eventually corrected by a line search
$(x^{i+1} = x^i + \alpha^i s^i / \|s^i\|)$. Introduce a momentum (a memory) in the
search step [Polyak, 1964],

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

where[3] $\beta = 0.9$.

This should contribute to avoid the
oscillations occuring at the botton
of valleys.

s^i still a descent direction.

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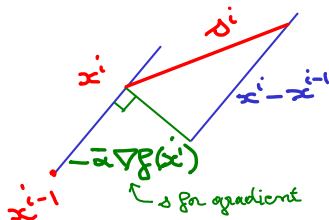
[3]alternatively, for iteration varying momentum, $\beta^i = (i-2)/(i+1)$

Gradient with momentum (2)

Back to Rosenbrock, $d = 2$, $x^\star = (1, 1)$, $f(x^\star) = 0$,
budget=4000, $x^1 = (4, 4)$
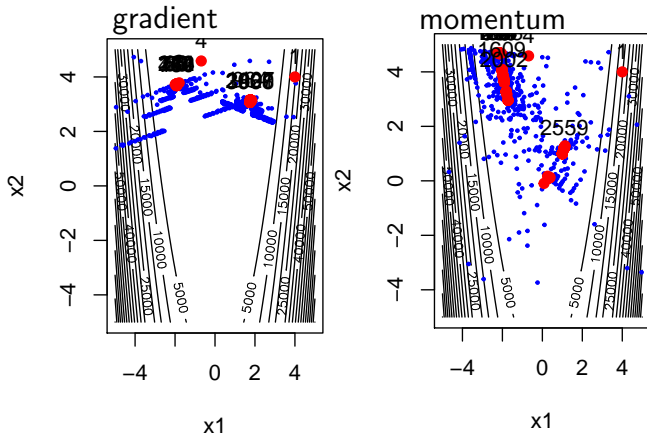
gradient

momentum

The momentum acceleration allows to find the solution.

Nesterov accelerated gradient (NAG)

Same idea as the momentum direction, but anticipate the position of the next point in the gradient calculation [Nesterov, 1983]:

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

On the sketch, ∇f turns upwards and the step is adjusted accordingly.

s^i is no longer necessarily a descent direction.
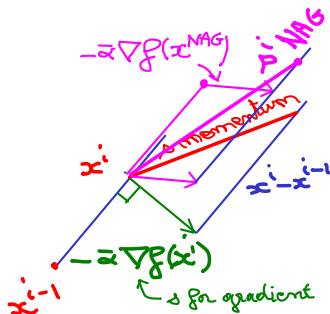
Comparison of methods (1)

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

nb. fct. eval
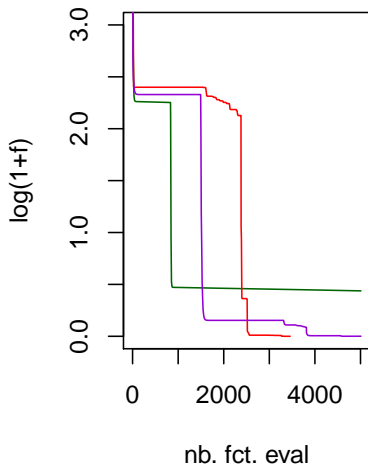
green=gradient, red=momentum, violet=NAG

Comparison of methods (2)

Test convergence speed on quadratic function, $d = 10$, cond. nb. = 100. green=gradient, red=momentum, violet=NAG

nb. fct. eval
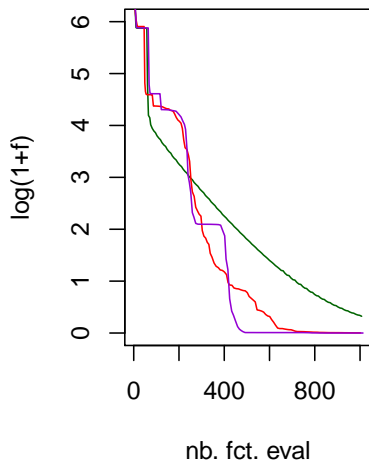
In accordance with theory, convergence speed of NAG is better than momentum which is better than gradient.

Note however the plateaus (or "ripples") typical of NAG and likely magnified by the finite difference approximation to the gradient.

Course outline

A word about constraints

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

Bound constraints

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

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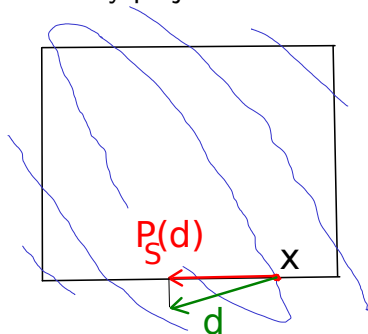
[a]This can even happen for a convex function in a convex $\mathcal{S}$, as the drawing shows.

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

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

- 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[4], 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].

---

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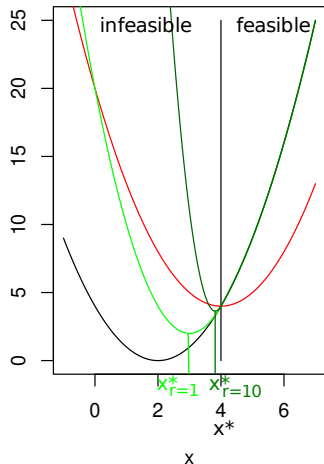
f and g in black, $L(x; \lambda^\star = 4)$ in red, exterior penalty $f_r()$ with $r = 1$ and 10 in light and dark green, respectively.

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 $f(x) = \sum_{i=1}^{d} |x_i|$ ("L1norm") with the code.

forward finite difference estimation to the gradient: no progress, stops at
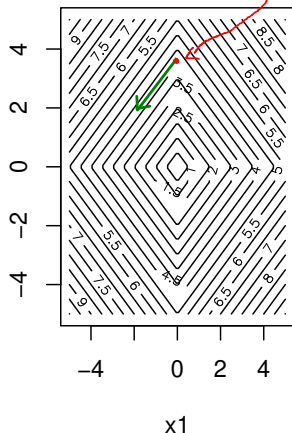
Main flaw: gets trapped in local minima.

Towards ML: regularized quadratic function

Let's consider a function with a relationship to ML:

$$f(x) = \sum_{i=1}^{d}(x_i - c_i)^2 + \lambda \sum_{i=1}^{d}|x_i| \quad , \quad \lambda \geq 0 \qquad (1)$$

First term: sphere function centered at c, $c_i = i$, $i = 1, \ldots, d$. A simplistic model to the mean square error of a NN where c minimizes the training error.

Second term: L1 norm times λ. The x_i's would be the weights of a NN. This term helps in improving the test error. Let's see why.

Exercise

1. Program the regularized sphere function (1).
2. Minimize it with any version of the previous descent algorithm with a line search in dimension $d = 10$ with $\mathcal{S} = [-5, 5]^d$. Try several values of $\lambda \geq 0$. What do you notice on the solution $x^\star$ found? Why, if the x's were a NN weights, would it improve the generalization ability of the NN?

Solution I

1. Cf. function `sphereL1` at the end of file `test_functions`.

2.

$\lambda = 0.01$	0.99	1.99	2.99	3.99	4.99	5	5	5	5	5
$\lambda = 1$	0.5	1.5	2.5	3.5	4.5	5	5	5	5	5
$\lambda = 3$	0	0.5	1.5	2.49	3.49	4.49	5	5	5	5
$\lambda = 5$	0	0	0.48	1.50	2.49	3.51	4.50	5	5	5
$\lambda = 10$	0	0	0	0	0.16	0.90	2.	3.23	3.92	5

As λ increases, more x_i's tend to 0. Because $c_i = i$, the components of lower rank are closer to 0, they are thus the first ones to be set to 0.

This phenomenon can be better understood by interpreting Problem (1) as a constrained optimization problem.

$$\begin{cases} \min_x f(x) = \|x - c\|^2 \\ \text{tel que} \quad g(x) = \|x\|_1 - \tau \leq 0 \quad , \quad \tau > 0 \end{cases}$$

Solution II

has the following Lagrangian, to be minimized on x

$$\min_x f(x) + \lambda^\star g(x) = \|x - c\|^2 + \lambda^\star \|x\|_1 - \lambda^\star \tau$$

The first 2 terms are the regularized function (1), the last term doesn't depend on x. Let's draw the sphere function and the constraint boundary $\|x\|_1$:

Solution III
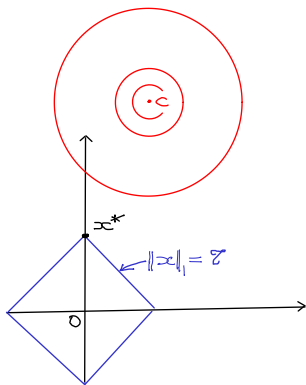
Solution IV

It is seen that the solution tends to be at a vertex of the feasible domain, where some of the x components cancel out. This phenomenon occurs more often when d is large and is less obvious for $d = 2$.

If the x's were a NN weights, setting x_i to 0 is equivalent to cutting the corresponding link, thus reducing the NN capacity, thus making it less prone to overfitting. Finding the best value for λ is something of an art.

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)

$$\text{at a cost} \times \texttt{nb\_restarts}:$$

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`:
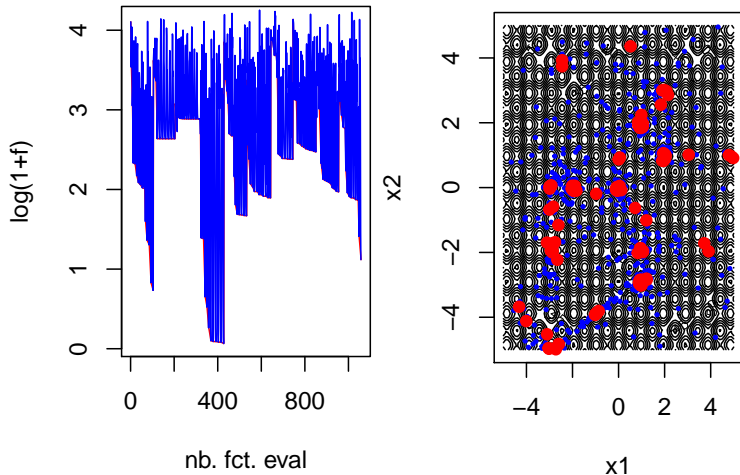
Application to neural network

$$\min_{x \in [LB, UB] \subset \mathbb{R}^d} f_\lambda(x) = f(x) + \lambda \|x\|_1$$

where, for classification,

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

and for regression,

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

Exercise: regularization vs. training data size

Study the relationship between the number of training points and the best choice of the regularization parameter.

To do this, use the file `main_neural_net` from `https://github.com/rleriche/Optimisation_AI_Cotonou2021`. It contains the code for a 2 inputs NN in regression, `ntrain` is the number of training points, `lambda` the regularization parameter. Try all the combinations between `ntrain`=5, 10 et 50, and $\lambda = 0.01,\ 0.1$ et 1.

What do you notice on the test error (`rmsetest`) ? How do you explain it ? How does the training error evolve (`rmsetrain`) ? Same question with the calculated solutions $x^\star$?

Solution I

Regularization vs. training size: the experiments yield:

	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 1$
ntrain=5	0.62266703	0.53166419	0.5217468
ntrain=10	0.66836290	0.21722179	0.4202079
ntrain=50	0.05191297	0.06292801	0.4229770

Table: Root Mean Square Error on the test set, ntest=100. In red: smallest error per row (i.e., per training data set size).

Solution II

When `ntrain` is small, it is best to regularize a lot: $\lambda = 1$ is the best setting with 5 training points. Vice versa, a large training set (50 points) does not need much regularization. There is a trade-off between the NN flexibility (that increases when λ decreases) and the number of training points that constraint the NN: a flexible NN with little constraints may be wrong away from the data points.

`rmsetrain` increases with λ since the emphasis is put on the minimization of $\|x\|_1$ at the expense of data fitting.

When λ is large, some weights of the NN cancel out. The number of null weights grows when the size of the training set diminishes.

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

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.

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.

Polyak, B. T. (1964).
Some methods of speeding up the convergence of iteration methods.
USSR computational mathematics and mathematical physics, 4(5):1–17.

References II

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.

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

References III

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.