An introduction to optimization for machine learning
Rodolphe Le Riche, Dédji Whannou, Espéran Padonou

To cite this version:
Rodolphe Le Riche, Dédji Whannou, Espéran Padonou. An introduction to optimization for machine learning. DEUG. Cotonou, Benin. 2021. hal-03345681v2

HAL Id: hal-03345681
https://hal.archives-ouvertes.fr/hal-03345681v2
Submitted on 18 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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

An introduction to optimization for machine learning

1. Introduction
   - Objectives, acknowledgements
   - Optimization problem formulation
   - Examples of optimization usages
   - Basic mathematical concepts for optimization

2. Steepest descent algorithm

3. Improved gradient based searches
   - Fixed step steepest descent algorithm
   - Line search
   - Search directions for acceleration
   - A word about constraints
   - Towards ML: regularized quadratic function
   - Making it more global: restarts

4. Application to neural network

5. Bibliography
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 way of mathematically modeling decision.

\[
\min_{x \in S} f(x)
\]

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

\( ^1 \) non unique, incomplete when considering human beings or life
Optimization example: design

\[ x = \text{aircraft parameters (here distributed electrical propulsion)} \]
\[ f() = -1 \times \text{performance metric (aggregation of } -1 \times \text{range, cost, take-off length, ...)} \]
At the minimum, the design is “optimal”.

(from [Sgueglia et al., 2018])
Optimization example: model identification

\( x = \text{dike position, geometry, internal pressure} \)

\( f() = \text{distance between measures (from RADARSAT-1 satellite) and model (boundary elements, non trivial computation)} \)

At the minimum, the model best matches measurements and should correspond to the underground phenomenon.

(from [Fukushima et al., 2010])
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.

\[ x = \text{neural network (NN) weights and biases} \]

\[ f() = \text{an error of the NN predictions (a cross-entropy error):} \]

- 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 \( t(e) = 1 \)
- \[ f(x) = -\sum_{i=1}^{N} \left\{ t^i \log(y(e^i; x)) + (1 - t^i) \log(1 - y(e^i; x)) \right\} \]
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 probability that \( t(e) = 0 \), \( 0 \leq y(e; x) \leq 1 \).

The probability of \( t \) knowing \( e \) can be written as \( y(e; x)^t \times (1 - y(e; x))^{1-t} \).

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

\[ e_1 \rightarrow e_2 \rightarrow y(e; x) \rightarrow t(e) \]

- \( x = \) 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) \hspace{1cm} (noisy (observed)) \hspace{1cm} (denoised (optimized))

(from [Ravikumar and Singh, 2017])
Introduction

- Objectives, acknowledgements
- Optimization problem formulation
- Examples of optimization usages
- Basic mathematical concepts for optimization

Steepest descent algorithm

- Fixed step steepest descent algorithm

1. Introduction

2. Steepest descent algorithm

3. Improved gradient based searches

- Line search
- Search directions for acceleration
- A word about constraints
- Towards ML: regularized quadratic function
- Making it more global: restarts

4. Application to neural network

5. Bibliography

Bibliography
Local versus global optimum

\[
\min_{x \in S \subset \mathbb{R}^d} f(x)
\]

R code to generate such a 3D plot given in the project folder, 3Dplots.R
Gradient of a function

Gradient of a function $= \text{direction of steepest ascent} = \text{vector of partial derivatives}$

$$\nabla f(x) = \left( \frac{\partial f}{\partial x_1}(x) \right) \left( \begin{array}{c} \vdots \\ \frac{\partial f}{\partial x_d}(x) \end{array} \right)$$
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,

\[
f(x + he^i) = f(x) + he_i^\top \nabla f(x) + h^2/2e_i^\top \nabla^2 f(x + \rho he^i)e_i, \quad \rho \in ]0, 1[
\]

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

and make \( h \) very small \( \Box \)

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).
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 □
Necessary optimality condition (1)

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

$$x^* \in \arg \min_{x \in S} f(x) \Rightarrow \nabla f(x^*) = 0$$
necessary is not sufficient (works with a max)
\[ \nabla f(x^*) = 0 \] does not make \( x^* \) 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

1. Introduction
   - Objectives, acknowledgements
   - Optimization problem formulation
   - Examples of optimization usages
   - Basic mathematical concepts for optimization

2. Steepest descent algorithm
   - Fixed step steepest descent algorithm
   - Line search

3. Improved gradient based searches
   - Search directions for acceleration
   - A word about constraints
   - Towards ML: regularized quadratic function
   - Making it more global: restarts

4. Application to neural network

5. Bibliography
Optimizers as iterative algorithms

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

- 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.

\[
\text{Require: } f(), \alpha \in ]0, 1], x^1, \epsilon^{\text{step}}, \epsilon^{\text{grad}}, i^{\text{max}}
\]
\[
i \leftarrow 0, f^{\text{best so far}} \leftarrow \text{max\_double}
\]
\[
\text{repeat}
\]
\[
i \leftarrow i + 1
\]
\[
\text{calculate } f(x^i) \text{ and } \nabla f(x^i)
\]
\[
\text{if } f(x^i) < f^{\text{best so far}} \text{ then }
\]
\[
\text{update } x^{\text{best so far}} \text{ and } f^{\text{best so far}} \text{ with current iterate}
\]
\[
\text{end if}
\]
\[
\text{direction: } d^i = -\nabla f(x^i)/\|\nabla f(x^i)\|
\]
\[
\text{step: } x^{i+1} = x^i + \alpha\|\nabla f(x^i)\|d^i
\]
\[
\text{until } i > i^{\text{max}} \text{ or } \|x^i - x^{i-1}\| \leq \epsilon^{\text{step}} \text{ or } \|\nabla f(x^i)\| \leq \epsilon^{\text{grad}}
\]
\[
\text{return } x^{\text{best so far}} \text{ and } f^{\text{best so far}}
\]
(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 visualizations.
- **neural_net.R**: a feedforward network coded from scratch.
- **main_neural_net.R**: main script for learning a neural network with the optimization functions.
The choice of the step size factor $\alpha$ is critical: the steeper the function, the smaller $\alpha$. 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.

$$f(x) = \frac{1}{2}x^\top Hx, \ H \text{ positive definite}$$
Fixed step steepest descent algorithm (3)

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

\[ x^* = (1, 1), \quad f(x^*) = 0 \]
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 \(\alpha\) 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 $i$,

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

In practice, perfectly optimizing for $\alpha^i$ 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 $\delta$ proportion of the progress promised by order 1 Taylor expansion.
Approximate line search (2)

Sufficient decrease condition rewritten with line search notation:

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

\[ f(\alpha) = f(x^i + \alpha d^i) \]
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 $f(x) = \text{Rosenbrock}$ where fixed step size diverged

![Graph of log(1+f) vs nb. fct. eval and x1 vs x2 showing oscillations.]

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

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

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,

\[ -d^{i+1\top} d^i = 0 \quad \text{i.e.} \quad d^{i+1} \text{ and } d^i \text{ perpendicular} \]
An introduction to optimization for machine learning

1. Introduction
   - Objectives, acknowledgements
   - Optimization problem formulation
   - Examples of optimization usages
   - Basic mathematical concepts for optimization

2. Steepest descent algorithm

3. Improved gradient based searches
   - Fixed step steepest descent algorithm
   - Line search
   - Search directions for acceleration
   - A word about constraints
   - Towards ML: regularized quadratic function
   - Making it more global: restarts

4. Application to neural network

5. Bibliography
Gradient with momentum (1)

Recall fixed step gradient descent,

\[ x^{i+1} = x^i + \alpha s^i \]

where \( s^i = -\nabla f(x^i) \)

\( s^i \), the step, corrected by a fixed or optimized (line search) \( \alpha \).

Introduce a momentum (i.e., a memory) in the search step [Polyak, 1964],

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

where\(^3 \beta = 0.9.\) This should contribute to avoid the oscillations occurring at the bottom 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, \( d = 2, \ x^* = (1, 1) \), \( f(x^*) = 0 \), budget=4000, \( x^1 = (4, 4) \)

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]:

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

On the sketch, \( \nabla f \) 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

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

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

An introduction to optimization for machine learning

1 Introduction
- Objectives, acknowledgements
- Optimization problem formulation
- Examples of optimization usages
- Basic mathematical concepts for optimization

2 Steepest descent algorithm
- Fixed step steepest descent algorithm
- Line search

3 Improved gradient based searches
- Search directions for acceleration
- A word about constraints
- Towards ML: regularized quadratic function
- Making it more global: restarts

4 Application to neural network

5 Bibliography
A word about constraints

\[
\begin{cases}
\min_{x \in S} f(x), & S = \mathbb{R}^d \\
\text{such that } g_i(x) \leq 0, & i = 1, m
\end{cases}
\]
Bound constraints

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

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, $S = [LB, UB]$, project the search direction vector onto the active bound.

Exercise: how to code this?

This can even happen for a convex function in a convex $S$, as the drawing shows.
Constraints handling by penalizations (1)

\[
\begin{align*}
\min_{x \in S \in \mathbb{R}^d} f(x) \\
\text{such that } g(x) \leq 0
\end{align*}
\] (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

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

- 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)
Lagrangian: for problems without duality gap\(^4\), e.g., convex problems, there exists Lagrange multipliers \(\lambda^*\) such that

\[
x^* \in \arg \min_{x \in S} L(x; \lambda^*)
\]

where \(L(x; \lambda^*) := f(x) + \lambda^* g(x)\)

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

- **Pros:** duality provides a way to calculate \(\lambda^*\), yields a feasible solution.

- **Cons:** estimating \(\lambda^*\) has a numerical cost. For most problems with local optima there is a duality gap \(\Rightarrow\) rely on augmented Lagrangians\(^4\).

\(^{4}\text{cf. duality, out of scope for this course}\)
Example: \( f(x) = (x - 2)^2, \ g(x) = 4 - x \leq 0, \ x^* = 4, \) convex problem

\( f \) and \( g \) in black, \( L(x; \lambda^* = 4) \) 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, \( x_r^* \to x^* \) but the curvature of \( f_r() \) increases.
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.

Main flaw: gets trapped in local minima.
Towards ML: regularized quadratic function

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

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

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 \(\lambda\). 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 $x^*$ found? Why, if the $x$’s were a NN weights, would it improve the generalization ability of the NN?
Cf. function sphereL1 at the end of file test_functions.

\[
\begin{array}{|c|cccccccccc|}
\hline
\lambda = 0.01 & 0.99 & 1.99 & 2.99 & 3.99 & 4.99 & 5 & 5 & 5 & 5 & 5 \\
\lambda = 1 & 0.5 & 1.5 & 2.5 & 3.5 & 4.5 & 5 & 5 & 5 & 5 & 5 \\
\lambda = 3 & 0 & 0.5 & 1.5 & 2.49 & 3.49 & 4.49 & 5 & 5 & 5 & 5 \\
\lambda = 5 & 0 & 0 & 0.48 & 1.50 & 2.49 & 3.51 & 4.50 & 5 & 5 & 5 \\
\lambda = 10 & 0 & 0 & 0 & 0 & 0.16 & 0.90 & 2.3 & 3.23 & 3.92 & 5 \\
\hline
\end{array}
\]

As \( \lambda \) 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.

\[
\begin{cases}
\min_x f(x) = \|x - c\|^2 \\
tel que \quad g(x) = \|x\|_1 - \tau \leq 0, \quad \tau > 0
\end{cases}
\]
has the following Lagrangian, to be minimized on $x$

$$\min_{x} f(x) + \lambda^* g(x) = \|x - c\|^2 + \lambda^* \|x\|_1 - \lambda^* \tau$$

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

The diagram illustrates a point $x^*$ in a space $\mathbb{R}^n$ with a radius $\|x\|_1 = 2$. The point is constrained within a circle and a diamond shape, indicating constraints in both $l_2$ and $l_1$ norms.
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 $\lambda$ 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)

\[
\text{at a cost } \times \text{ nb}_\text{restarts}:
\]

Require: budget, nb\_restarts

\[
\text{for } i \text{ in 1 to } \text{nb}_\text{restarts} \text{ do}
\]
\[
\begin{align*}
\text{xinit} & \leftarrow \text{runif}(n=d, \text{min}=\text{LB}, \text{max}=\text{UB}) \\
\text{res} & \leftarrow \text{gradient\_descent}(\text{xinit}, \text{budget}=\text{budget}/\text{nb}_\text{restarts})
\end{align*}
\]
update global search results

\[
\text{end for}
\]
Restarted local searches: example

Execution of the restarted_descent file.

```r
fun <- rastrigin, d<-2, budget<-1000, nb_restart<-10:
```

![Graph showing function evaluations and variable values](image-url)
Application to neural network

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

where, for classification,

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

and for regression,

\[
f(x) = \frac{1}{2} \sum_{i=1}^{N} (t^i - y(e^i; x))^2
\]
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 \( x^* \)?
Solution I

Regularization vs. training size: the experiments yield:

<table>
<thead>
<tr>
<th></th>
<th>$\lambda = 0.01$</th>
<th>$\lambda = 0.1$</th>
<th>$\lambda = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntrain=5</td>
<td>0.62266703</td>
<td>0.53166419</td>
<td>0.5217468</td>
</tr>
<tr>
<td>ntrain=10</td>
<td>0.66836290</td>
<td><strong>0.21722179</strong></td>
<td>0.4202079</td>
</tr>
<tr>
<td>ntrain=50</td>
<td><strong>0.05191297</strong></td>
<td>0.06292801</td>
<td>0.4229770</td>
</tr>
</tbody>
</table>

**Table:** Root Mean Square Error on the test set, $n_{test}=100$. In red: smallest error per row (i.e., per training data set size).
When $n_{\text{train}}$ 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 $\lambda$ 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.

$r\text{mse}_{\text{train}}$ increases with $\lambda$ since the emphasis is put on the minimization of $\|x\|_1$ at the expense of data fitting.

When $\lambda$ 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 enthusiasm 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].


