Open in app Get started

Published in Towards Data Science

Vincenzo Taccardi Follow
Mar 30, 2020 · 7 min read · Listen

Save

Unboxing Lasso regularization with proximal gradient method

ISTA (Iterative Soft-Thresholding Algorithm)

I start this story by first telling you that i am not a mathematician and since this topic will need **some** *math (brace yourself!) i want to remark that it won't be totally rigorous.*

This post's purpose is to summarize the whole path that leads to the application of ISTA (Iterative Soft-Thesholding Algorithm) *to solve the problem of lasso regularization for gradient descent optimization .*

Of course i assume that if you are reading at least you have an hint about what we are going to talk about here and i hope my post will be helpful.

Let's first compose the mathematical puzzle that will lead us to understand how to compute lasso regularization with gradient descent even if the cost function is not differentiable, as in the case of Lasso.

Sub-gradient

The first concept to grasp is the definition of a convex function.

A complete differentiable function ***f*** *is said to be convex if:*

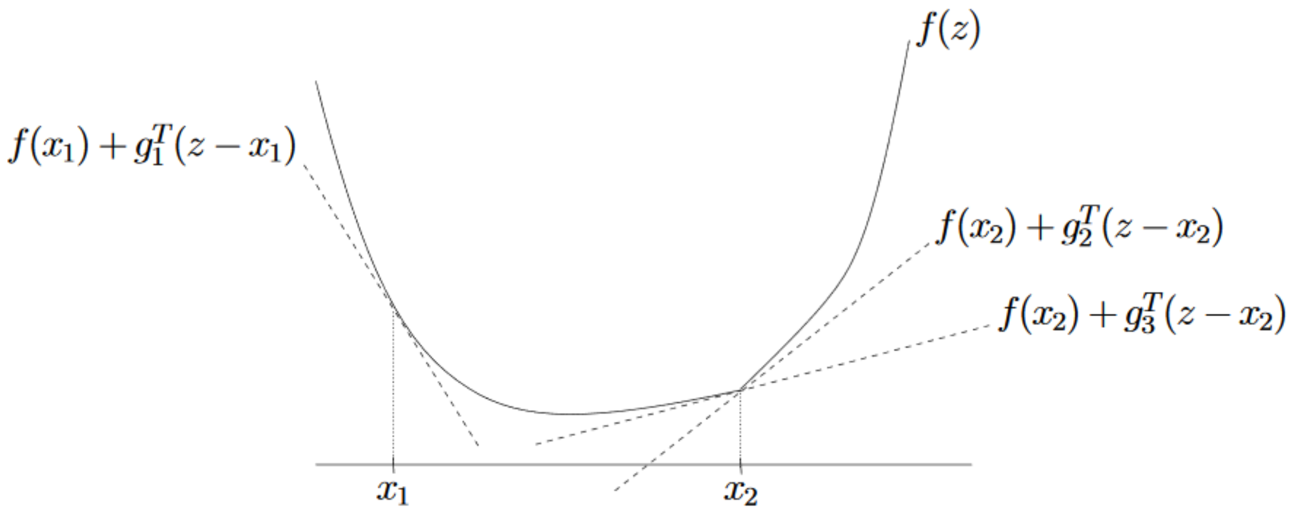
In other words, its local linear approximation (Taylor expansion) always underestimates f. An example of such convex function is our beloved mean square error.

Let's go further, and straight from convex functions comes the sub-gradient, *this tool is necessary when we are dealing with a function that is not differentiable at every point, and since the lasso uses the L1 norm of our variables (absolute value) we know that this function is* ***not differentiable*** *at zero.*

The sub-gradient is a way of generalizing the gradient of a convex function at non-differentiable points. A sub-gradient of a convex function f at a point x is any g such that:

$$f(y) \geq f(x) + g^T(y - x) \qquad \forall y$$

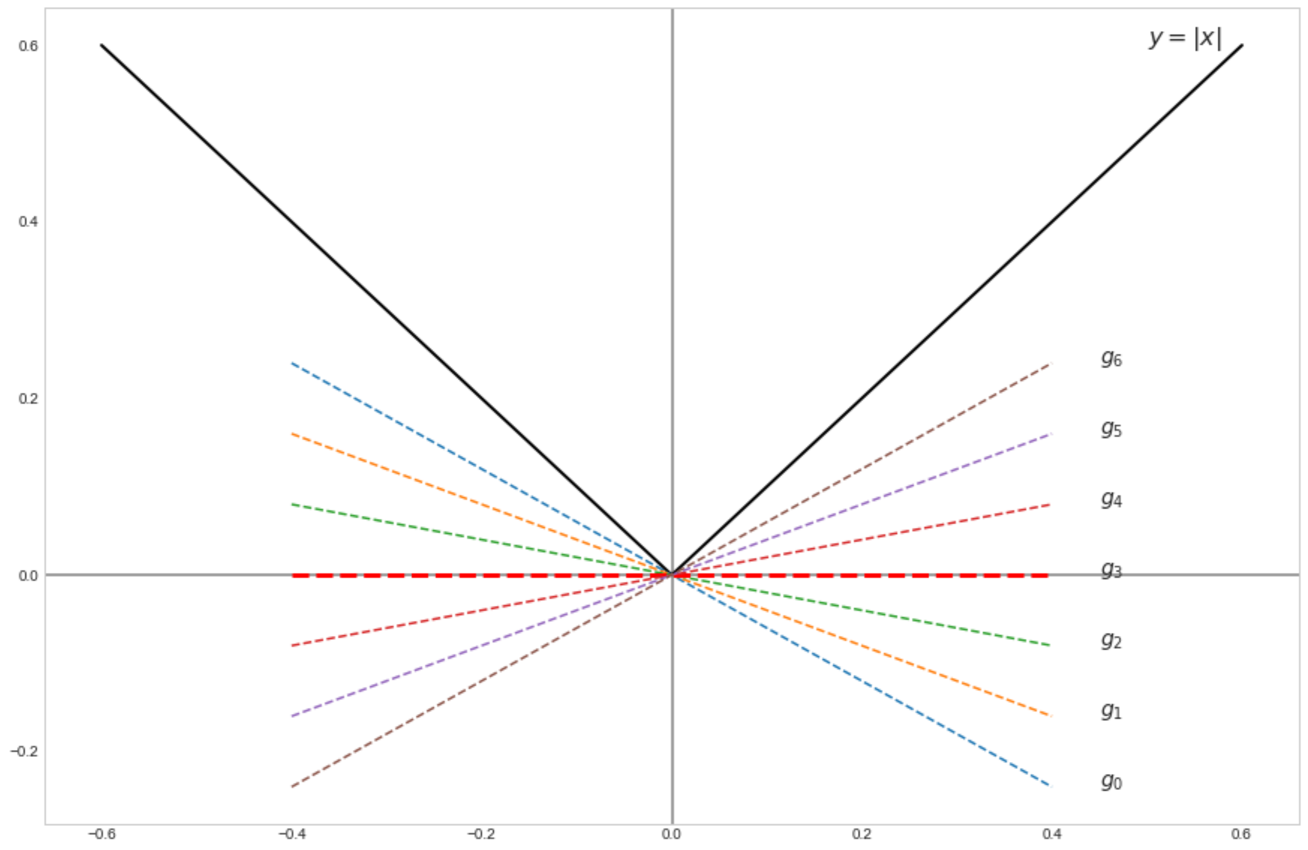
This expression is again a linear approximation of f where instead of gradient we use sub-gradient which is inside a ***set*** *of values that respect the condition above.*


$f(z)$
$f(x_1) + g_1^T(z - x_1)$
$f(x_2) + g_2^T(z - x_2)$
$f(x_2) + g_3^T(z - x_2)$
x_1
x_2

sub-gradients for a convex function f(x)

g1, g2, g3 are sub-gradients of f at x1, x2 .

In the case of f(x)=|x| the subgradients g at x = 0 looks like in below graph and they range among all possible values included in [-1,1] closed set. Any line passing through x = 0 with a slope in this range ***will lower bound the function****.*



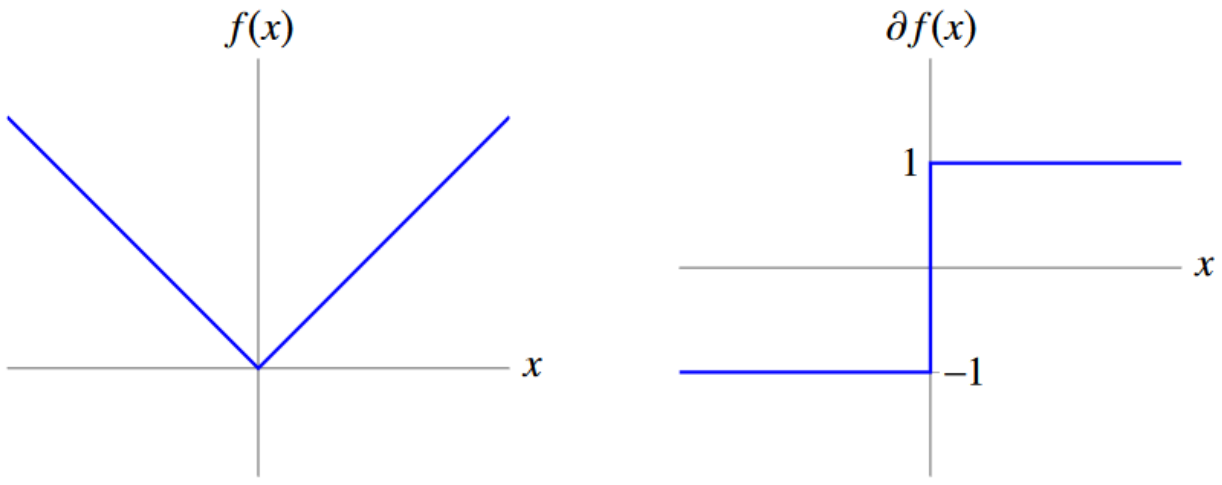
some subgradients for f(x)=|x| at x=0

The set of all subgradients of a convex f is called the subdifferential:

$$\partial f(x) = \{g \in \mathbb{R}^n : g \text{ is a subgradient of } f \text{ at } x\}$$

The subdifferential's definition can be extend to all kinds of convex function, differentiable or not. The main point is that when a function is completely differentiable at each point in

Considering again the L1 norm for a single variable x:



The absolute value function (left), and its subdifferential ∂f(x) as a function of x (right)

$$\partial f(x) = J_1 \times \cdots \times J_n, \qquad J_k = \begin{cases} [-1,1] & x_k = 0 \\ \{1\} & x_k > 0 \\ \{-1\} & x_k < 0 \end{cases}$$

subdifferential of f(x) = |x|; k=1,2,3 in this case

Given the subdifferential, thus the optimality condition for any ***f*** *(differentiable or not) is:*

$$f(x^\star) = \min_x f(x) \iff 0 \in \partial f(x^\star)$$

x is a minimizer if and only if 0 is a subgradient of f at x*. This is called the* ***subgradient optimality condition.***

Why? Easy, g= 0 being a subgradient means that for all y:

$$\partial f(x) = \{\nabla f(x)\}$$

Proximal mapping

Now that we are familiar with the subgradient, other tools we have to understand are: Proximal operator and Soft-thresholding operator.

***Proximal operator** definition:*

$$prox_{\alpha f}(x^k) = \underset{x}{argmin}\, f(x) + \frac{1}{2\alpha}\left\|x - x^k\right\|_2^2$$

*here we are searching the point **x***, which minimize a generic convex function f , but at same time remaining close to a reference point **x**k (square L2 norm). We chose how much close by **alpha** as a relative weight or trade-off parameter between these terms*

***Soft-thresholding operator** definition:*

$$S_\lambda(\hat{x}) = \begin{cases} \hat{x} - \lambda & if\ \hat{x} > \lambda \\ 0 & if\ |\hat{x}| \le \lambda \\ \hat{x} + \lambda & if\ \hat{x} < -\lambda \end{cases}$$

or in a more compact form:

$$S_\lambda(\hat{x}) = sign(\hat{x}) \max(|\hat{x}| - \lambda, 0)$$

I understand you bro/sis. But before your beer you need to go through the harsh part. Keep fighting!

So, let me try to give a sense to all of these and let's start to apply proximal operator.

*Define f as local linear approximation at x*k *for a generic function, we assume convex and differentiable, and let's plug it into the proximal operator:*

$$f(x) = f(x^k) + \nabla^T f(x^k)(x - x^k)$$

$$prox_{\alpha f}(x^k) = \underset{x}{argmin}\ f(x^k) + \nabla^T f(x^k)(x - x^k) + \frac{1}{2\alpha}\left\|x - x^k\right\|_2^2$$

given it, the point x is a local minimizer if:*

$$x^* = prox_{\alpha f}(x^k) \Leftrightarrow$$

$$\nabla\left(f(x^k) + \nabla^T f(x^k)(x - x^k) + \frac{1}{2\alpha}\left\|x - x^k\right\|_2^2\right) = 0$$

$$\alpha\nabla^T f(x^k) + x - x^k = 0$$

$$x^* = x^k - \alpha\nabla^T f(x^k)$$

I bet this reminds you something … Wait, i write it in a more familiar notation:

$$x^{k+1} = x^k - \alpha\nabla^T f(x^k)\ ; \quad k = 1, 2, 3, \ldots$$

Better now? Yes, that is a gradient descent step computed just with the proximal mapping of a linear approximation. Indeed gradient descent and proximal operator are veryyy close friends.

$$h(x) = \lambda \|x\|_1$$

again we can plug it into the prox operator and we get:

$$prox_{\alpha\|\cdot\|_1}(\hat{x}) = \underset{x}{argmin}\ \lambda\|x\|_1 + \frac{1}{2\alpha}\ \|x - \hat{x}\|_2^2$$

From the definition of L1 and L2 norm we have:

$$if \quad x \in \mathbb{R}^n$$

$$\lambda\|x\|_1 + \frac{1}{2\alpha}\ \|x - \hat{x}\|_2^2 = \sum_{i=1}^{n} \lambda|x_i| + \frac{1}{2\alpha}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

these are separable functions hence we can solve as a component-wise scalar problem:

$$prox_{\alpha|\cdot|}(\hat{x}_i) = \underset{x}{argmin}\ \lambda|x_i| + \frac{1}{2\alpha}(x_i - \hat{x}_i)^2$$

$$x_i^* = prox_{\alpha|\cdot|}(\hat{x}_i) \Leftrightarrow 0 \in \partial\left(\lambda|x_i| + \frac{1}{2\alpha}(x_i - \hat{x}_i)^2\right)$$

$$0 \in x_i - \hat{x}_i + \alpha\lambda\,\partial(|x_i|)$$

$$with\ v \in \partial|x_i|$$

$$0 = x_i - \hat{x}_i + \alpha\lambda\ v$$

$$x_i^* = \hat{x}_i - \alpha\lambda\ v$$

$$-1 \qquad if\ x_i^* < 0$$

so we get the final results:

$$x^*_i = \begin{cases} \hat{x}_i - \alpha\lambda & if\ \hat{x}_i > \alpha\lambda \\ 0 & if\ |\hat{x}_i| \le \alpha\lambda \\ \hat{x}_i + \alpha\lambda & if\ \hat{x}_i < -\alpha\lambda \end{cases}$$

or rearranging with the soft-thersholding operator introduced above:

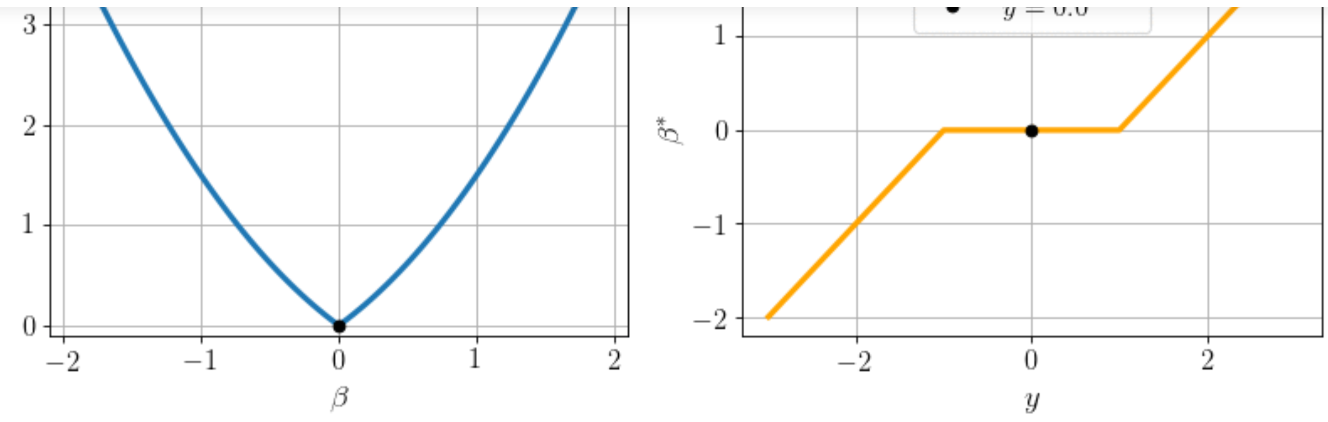
$$x_i^* = S_{\alpha\lambda}(\hat{x}_i) = sign(\hat{x}_i)\max(|\hat{x}_i| - \alpha\lambda, 0)$$

This last result can be a bit tricky, let me explain what's going on. From the absolute value's sub-differential and the optimality condition obtained above we have 3 possible cases:

1. $x_i^* > 0;\ \ x_i^* = \hat{x}_i - \alpha\lambda \cdot 1 = \hat{x}_i - \alpha\lambda > 0 \Leftrightarrow \hat{x}_i > \alpha\lambda$
2. $x_i^* < 0;\ \ x_i^* = \hat{x}_i - \alpha\lambda \cdot (-1) = \hat{x}_i + \alpha\lambda < 0 \Leftrightarrow \hat{x}_i < -\alpha\lambda$
3. $x_i^* = 0;\ \ 0 = \hat{x}_i - \alpha\lambda \cdot [-1,1] \Leftrightarrow \hat{x}_i = [-\alpha\lambda, \alpha\lambda] \Rightarrow |\hat{x}_i| \le \alpha\lambda$

So in the end we can generalize that:

$$prox_{\mu\|\cdot\|_1}(x) = S_\mu(x)$$



image's credits to Pierre Ablin

Here we go, the last round … Adrianaaaaaa!

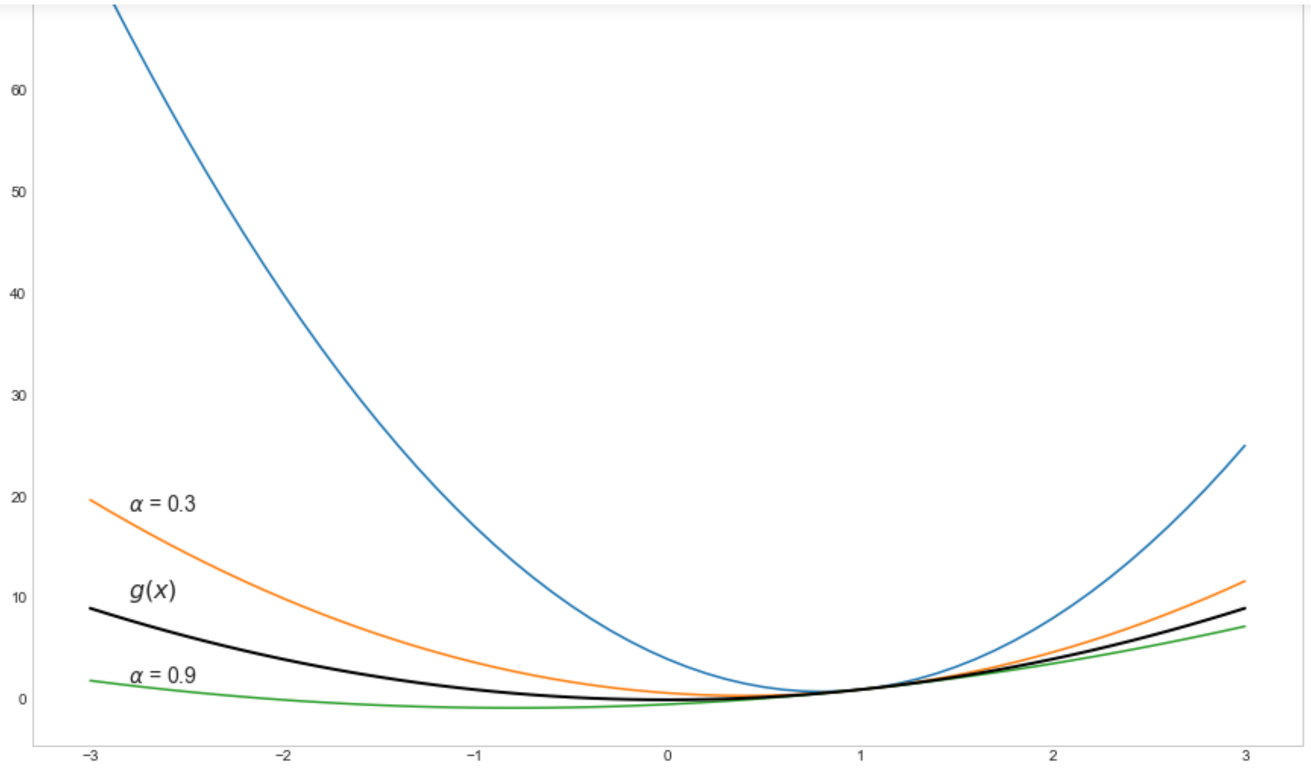
Now consider:

$$f(x) = g(x) + h(x)$$

as a sum of a generic convex differentiable function g (i.e. least square) plus a convex not differentiable function h, in our case the L1 norm, and we want to minimize it. For this purpose we first apply a local quadratic approximation to g and leave h untouched.

$$replace\ \nabla^2\, g(x^k)\ with\ \frac{1}{\alpha}\, I$$

$$g(x) = g(x^k) + \nabla^T g(x^k)(x - x^k) + \frac{1}{2\alpha}\left\|x - x^k\right\|_2^2 \quad ; \quad h(x) = \lambda\|x\|_1$$



local quadratic approximations with different alphas for a generic g(x) at x=1

Note, the local quadratic approximation we obtained above without the Hessian is just the proximal operator for a linear approximation we worked out before.

For our purpose we further need to rearrange g:

$$\begin{aligned} g(x) &= 2\alpha\, g(x^k) + \left(2\alpha\, \nabla^T g(x^k)(x-x^k) + \|x-x^k\|_2^2 + \|\alpha\, \nabla^T g(x^k)\|_2^2\right) - \|\alpha\, \nabla^T g(x^k)\|_2^2 \\ &= 2\alpha\, g(x^k) + \|\alpha\, \nabla^T g(x^k) + (x-x^k)\|_2^2 - \|\alpha\, \nabla^T g(x^k)\|_2^2 \\ &= g(x^k) + \frac{1}{2\alpha}\left\|x - \left(x^k - \alpha\, \nabla^T g(x^k)\right)\right\|_2^2 - \frac{1}{2\alpha}\|\alpha\, \nabla^T g(x^k)\|_2^2 \end{aligned}$$

and after removing the constant terms not depending on x and adding h we can minimize the whole f:

$$with \quad \hat{x} = x^k - \alpha \nabla^T g(x^k)$$

$$x^{k+1} = prox_{\alpha\lambda\|.\|_1}(\hat{x}) = \underset{x}{argmin} \ \lambda\|x\|_1 + \frac{1}{2\alpha}\|x - \hat{x}\|_2^2$$

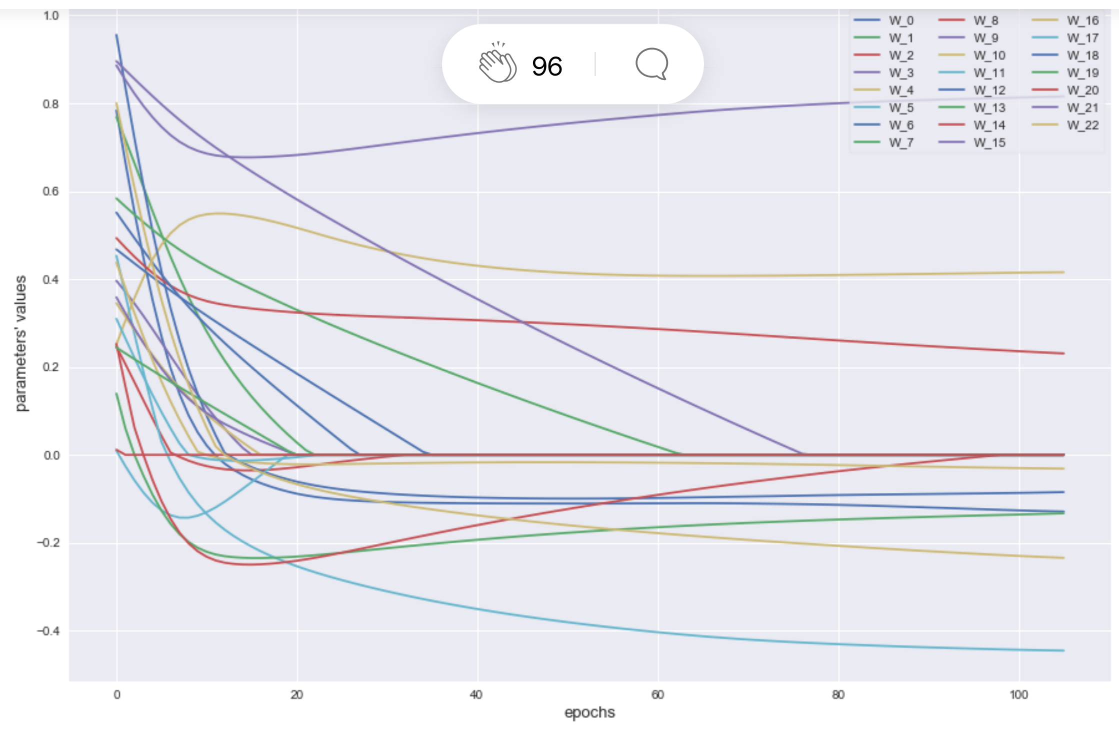
Finally after tons of math we can define **ISTA** (Iterative Shrinckage-Thresholding Algoritm) *as:*

$$x_i^{k+1} = prox_{\alpha\lambda|\cdot|}(\hat{x}_i) = S_{\alpha\lambda}(\hat{x}_i) = sign(\hat{x}_i)\max(|\hat{x}_i| - \alpha\lambda, 0)$$

$$\boldsymbol{x^{k+1}} = \begin{bmatrix} \boldsymbol{S_{\alpha\lambda}(\hat{x}_1)} \\ \boldsymbol{S_{\alpha\lambda}(\hat{x}_2)} \\ \vdots \\ \boldsymbol{S_{\alpha\lambda}(\hat{x}_n)} \end{bmatrix} \quad i = 1, \dots, n \quad ; \quad k = 1, 2, 3, \dots$$

$$\boldsymbol{S_{\alpha\lambda}(\hat{x}_i) = sign\left(\left[x^k - \alpha\nabla^T g(x^k)\right]_i\right) \max\left(\left|\left[x^k - \alpha\nabla^T g(x^k)\right]_i\right| - \alpha\lambda, 0\right)}$$

for each component of our n-variables separable function.



Yes, it looks nasty but is not as bad as it seems. You just calculate gradient descent as usual with the only difference that you apply a further transformation with the soft thresholding operator before passing the value to the next iteration.

Then as for gradient descent you can define a rule to stop iteration and get very close to the function's minimum.

It worths to mention that this results doesn't depends on g that can be as complicate as we
d t its gradient.

Done! From now on, no more math just beer. We deserve it.