Proximal Gradient Algorithms

Proximal algorithms are particularly useful when the functional we are minimizing can be broken into two parts, one of which is smooth, and the other for which there is a fast proximal operator.

We are interested in minimizing unconstrained programs of the form

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x}),$$

where

1. $g(\boldsymbol{x})$ is convex and differentiable, and
2. $h(\boldsymbol{x})$ is convex with an inexpensive prox operator (examples to come).

The **proximal gradient** algorithm minimizes f iteratively, with each iteration consisting of

1. taking a gradient step on g, then
2. taking a proximal step on h.

We can combine these steps into one expression:

$$\boldsymbol{x}^{(k+1)} = \text{prox}_{t_k h}\left(\boldsymbol{x}^{(k)} - t_k \nabla g(\boldsymbol{x}^{(k)})\right).$$

This is also called *forward-backward splitting*, with the "forward" referring to the gradient step, and the "backward" to the proximal step. (The prox step is still making progress, just like the gradient step; the forward and backward refer to the interpretations of gradient descent and the proximal algorithm as forward and backward Euler discretizations, respectively.)

Example: the LASSO

Recall the LASSO, for finding sparse solutions to systems of linear equations:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2+\tau\|\boldsymbol{x}\|_1$$

We take

$$g(x)=\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2, \quad \text{so} \quad \nabla g(\boldsymbol{x})=\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}),$$

and

$$h(\boldsymbol{x})=\tau\|vx\|_1.$$

The prox operator for the ℓ_1 norm is something familiar:

$$\begin{aligned}\text{prox}_{th}(\boldsymbol{z}) &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(\tau\|\boldsymbol{x}\|_1+\frac{1}{2t}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right)\\ &= T_{\tau t}(\boldsymbol{x}),\end{aligned}$$

where $T_{\tau t}$ is the soft-thresholding operator

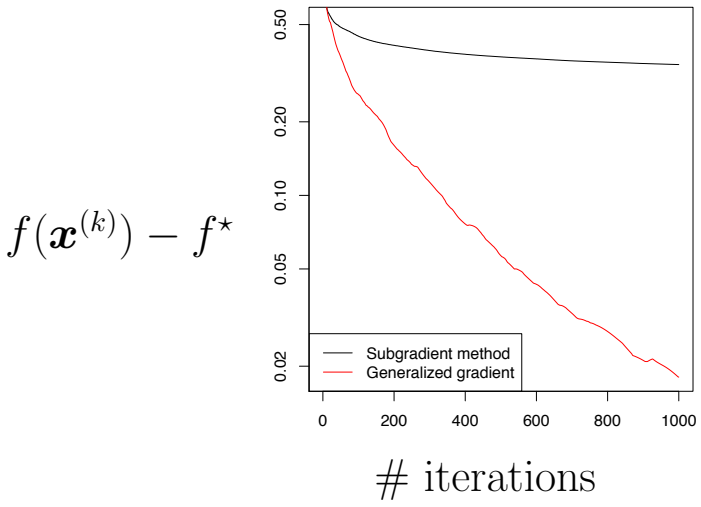
$$\left(T_\lambda(\boldsymbol{x})\right)[i]=\begin{cases}x[i]-\lambda, & x[i]\geq\lambda,\\ 0, & |x[i]|\leq\lambda,\\ x[i]+\lambda, & x[i]\leq-\lambda,\end{cases}$$

with $\lambda=\tau t$. So the gradient step requires an application of $\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}$, and the proximal step simply requires a soft-thresholding operation. The iteration looks like

$$\boldsymbol{x}^{(k+1)}=T_{\tau t_k}\left(\boldsymbol{x}^{(k)}+t_k\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}^{(k)})\right).$$

This is also called the *iterative soft thresholding algorithm*, or ISTA.

Here is a comparison[1] of a typical run for ISTA versus the subgradient method.



As we can see, ISTA absolutely crushes the subgradient method.

Example: Projected Gradient Descent for Constrained Optimization

Consider the constrained optimization program

$$\underset{\boldsymbol{x}\in\mathcal{C}}{\text{minimize}} \;\; g(\boldsymbol{x}), \tag{1}$$

where $\mathcal{C}$ is a convex set and $\boldsymbol{g}$ is a differentiable convex function. This program can be re-written as the unconstrained program

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; g(\boldsymbol{x}) + h(\boldsymbol{x})$$

where $h(\boldsymbol{x})$ is the characteristic function for $\mathcal{C}$:

$$h(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in \mathcal{C}, \\ +\infty, & \boldsymbol{x} \notin \mathcal{C}. \end{cases}$$

[1] This is taken from the lecture notes of Geoff Gordon and Ryan Tibshirani; "generalized gradient" in the legend means ISTA.

As we have seen before, $h(\boldsymbol{x})$ is convex. Its prox function is

$$\begin{aligned}\text{prox}_{tf}(\boldsymbol{z}) &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(h(\boldsymbol{x}) + \frac{1}{2t}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right) \\ &= \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\,\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 \\ &= P_{\mathcal{C}}(\boldsymbol{z}).\end{aligned}$$

That is, for any $t > 0$, $\text{prox}_{tf}(\boldsymbol{z})$ is the closest point in $\mathcal{C}$ to $\boldsymbol{z}$ — it a projection onto the set $\mathcal{C}$.

The iteration for solving (1) is then

$$\boldsymbol{x}^{(k+1)} = P_{\mathcal{C}}\left(\boldsymbol{x}^{(k)} - t_k\nabla g(\boldsymbol{x}^{(k)})\right).$$

That is, we take a gradient step on g, then re-project onto the constraint set. Notice that for $k \geq 1$, the current iterate $\boldsymbol{x}^{(k)}$ will always be feasible.

This is called **projected gradient descent**. It is a very simple (but effective) method for solving constrained optimization problems when we have an efficient method for projecting onto the constraint set $\mathcal{C}$.

Example: Least-squares with positivity constraints

Suppose we want to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\ \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \quad \text{subject to} \quad \boldsymbol{x} \geq \boldsymbol{0}.$$

The projector onto the set of all positive vectors is simple, we simply just set all of the negative entries to zero. The iteration is

$$\boldsymbol{x}^{(k+1)} = \left(\boldsymbol{x}^{(k)} - t_k\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}^{(k)})\right)_+,$$

where

$$(\boldsymbol{z})_+[i] = \begin{cases} z[i], & z[i] \geq 0, \\ 0, & z[i] < 0. \end{cases}$$

Example: Nuclear norm minimization (matrix LASSO)

A common problem in machine learning and signal processing is to recover a low-rank matrix from indirect linear measurements. Just as we use the ℓ_1 norm as a proxy for the sparsity of a vector, we can use the *nuclear norm* (sum of singular values) as a proxy for the rank of a matrix.

FINISH THIS ...

Convergence analysis

The convergence analysis is very similar to what we did for the proximal point method. We assume that the gradient of g is Lipschitz, meaning that for any $\boldsymbol{x} \in \mathbb{R}^N$,

$$g(\boldsymbol{z}) \leq g(\boldsymbol{x}) + \nabla g(\boldsymbol{x})^{\mathrm{T}}(\boldsymbol{z}-\boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{z}-\boldsymbol{x}\|_2^2, \quad \text{for all } \boldsymbol{z} \in \mathbb{R}^N. \tag{2}$$

We will use a fixed "step size", $t = t_k$ for all k, and we will take $t \leq 1/L$. We will also assume that f has a minimizer $\boldsymbol{x}^\star$, and we will use the notation $f^\star = f(\boldsymbol{x}^\star)$.

Let's again look at what happens in the iteration moving from $\boldsymbol{x}^{(k)}$ to $\boldsymbol{x}^{(k+1)}$. As before, let $\boldsymbol{d}$ be the direction we move between these two iterations:

$$\boldsymbol{d} = \frac{1}{t}\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k+1)}\right),$$

so $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - t\boldsymbol{d}$.

Now

$$\begin{aligned} f(\boldsymbol{x}^{(k+1)}) &= g(\boldsymbol{x}^{(k+1)}) + h(\boldsymbol{x}^{(k+1)}) \\ &\leq g(\boldsymbol{x}^{(k)}) - t\nabla g(\boldsymbol{x}^{(k)})^{\mathrm{T}}\boldsymbol{d} + \frac{Lt^2}{2}\|\boldsymbol{d}\|_2^2, \end{aligned} \tag{3}$$

where we have applied the Lipschitz bound (2). We now use two facts to get an upper bound on this expression for arbitrary points $\boldsymbol{z}$. First, since $\nabla g(\boldsymbol{x}^{(k)})$ is a subgradient of g at $\boldsymbol{x}^{(k)}$,

$$g(\boldsymbol{z}) \geq g(\boldsymbol{x}^{(k)}) + \nabla g(\boldsymbol{x}^{(k)})^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k)}). \tag{4}$$

Second, since

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \mathrm{prox}_{th}(\boldsymbol{x}^{(k)} - t\nabla g(\boldsymbol{x}^{(k)})) \\ &= \arg\min_{\boldsymbol{x}} \left(h(\boldsymbol{z}) + \frac{1}{2t}\|\boldsymbol{x} - \boldsymbol{x}^{(k)} + t\nabla g(\boldsymbol{x}^{(k)})\|_2^2 \right), \end{aligned}$$

we know

$$\mathbf{0} \in \partial h(\boldsymbol{x}^{(k+1)}) - \boldsymbol{d} + \nabla g(\boldsymbol{x}^{(k)}) \quad \Rightarrow \quad \boldsymbol{d} - \nabla g(\boldsymbol{x}^{(k)}) \in \partial h(\boldsymbol{x}^{(k+1)}).$$

Thus

$$h(\boldsymbol{z}) \geq h(\boldsymbol{x}^{(k+1)}) + (\boldsymbol{d} - \nabla g(\boldsymbol{x}^{(k)})^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k+1)}). \tag{5}$$

We combine (4) and (5) back into (3) to get

$$\begin{aligned} f(\boldsymbol{x}^{(k+1)}) &\leq g(\boldsymbol{z}) - \nabla g(\boldsymbol{x}^{(k)})^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k)}) - t\nabla g(\boldsymbol{x}^{(k)})^{\mathrm{T}}\boldsymbol{d} + \frac{Lt^2}{2}\|\boldsymbol{d}\|_2^2 \\ &\quad + h(\boldsymbol{z}) - (\boldsymbol{d} - \nabla g(\boldsymbol{x}^{(k)}))^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k)} + t\boldsymbol{d}) \\ &= f(\boldsymbol{z}) - \boldsymbol{d}^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k)}) + t\left(\frac{Lt}{2} - 1\right)\|\boldsymbol{d}\|_2^2 \\ &\leq f(\boldsymbol{z}) - \boldsymbol{d}^{\mathrm{T}}(\boldsymbol{z} - \boldsymbol{x}^{(k)}) - \frac{t}{2}\|\boldsymbol{d}\|_2^2, \end{aligned}$$

where we have used the bound on the step size $t \leq 1/L$ in the last step. Note that the expression above holds for all $\boldsymbol{z} \in \mathbb{R}^N$.

Evaluating the expression for the upper bound above at $\boldsymbol{z} = \boldsymbol{x}^{(k)}$ to see that the functional value is non-increasing at every step:

$$f(\boldsymbol{x}^{(k+1)}) \leq f(\boldsymbol{x}^{(k)}) - \frac{t}{2}\|\boldsymbol{d}\|_2^2 \leq f(\boldsymbol{x}^{(k)}).$$

We can also evaluate the upper bound at $\boldsymbol{z} = \boldsymbol{x}^\star$,

$$f(\boldsymbol{x}^{(k+1)}) - f^\star \leq -\boldsymbol{d}^{\mathrm{T}}(\boldsymbol{x}^\star - \boldsymbol{x}^{(k)}) - \frac{t}{2}\|\boldsymbol{d}\|_2^2,$$

and then complete the square as was done on page 8 of the previous set of notes to yield

$$f(\boldsymbol{x}^{(k+1)}) - f^\star \leq \frac{1}{2t}\left(\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^\star\|_2^2\right),$$

which again results in the convergence bound

$$\begin{aligned} f(\boldsymbol{x}^{(k)}) - f^\star &\leq \frac{1}{k}\sum_{i=1}^{k} f(\boldsymbol{x}^{(k+1)}) - f^\star \\ &\leq \frac{\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2^2}{2kt} \end{aligned}$$

So the proximal gradient algorithm exhibits a convergence rate of $O(1/k)$.

The above is kind of a "master result" for the convergence rate of three different algorithms that are special cases of proximal gradient:

- gradient descent (take $h(\boldsymbol{x}) = 0$),
- conditional gradient descent (take $h(\boldsymbol{x})$ as the characteristic function for a set $\mathcal{C}$),
- the proximal point method (take $g(\boldsymbol{x}) = 0$).

The work above gives a unified analysis for all three of these, showing that they are all $O(1/k)$.

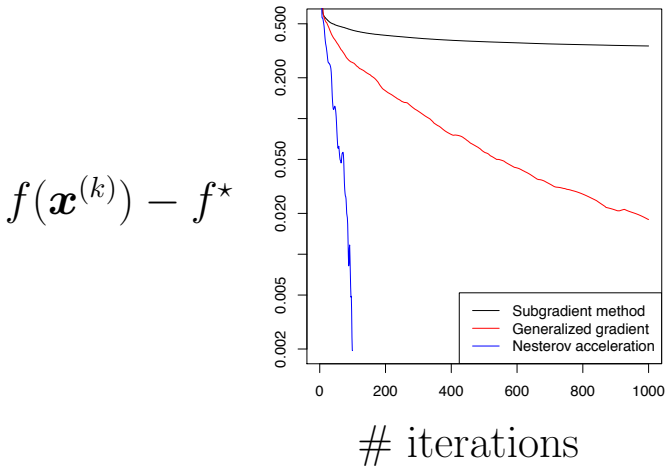
Accelerated proximal gradient

We can accelerate the proximal gradient method in exactly the same way we accelerated the proximal point method — in fact, the ppm acceleration is simply a special case as that for the proximal gradient algorithm. The accelerated iteration is

$$\begin{aligned} \boldsymbol{w}^{(k)} &= \boldsymbol{x}^{(k)} + \frac{k-1}{k+2}\left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\right) \\ \boldsymbol{x}^{(k+1)} &= \mathrm{prox}_{t_k h}\left(\boldsymbol{w}^{(k)} - t_k \nabla g(\boldsymbol{w}^{(k)})\right). \end{aligned}$$

Again, the actual computations here are no more involved than for the non-accelerated version, but the gains in efficiency (in terms of number of iterations to convergence) can be significant. We will not prove it here (see [BT09] for a complete analysis), but adding in the momentum term above results in convergence rate of $O(1/k^2)$.

The numerical performance can also be dramatically better. Here are typical runs[2] for the LASSO, which compares the standard proximal gradient method (ISTA) to its accelerated version (FISTA):



References

[BT09] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202, 2009.

[2] Again, this example comes from Gordon and Tibshirani; as before "generalized gradient" means ISTA, and "Nesterov acceleration" means FISTA.