Proximal algorithms

The subgradient algorithm is one generalization of gradient descent. It is simple, but the convergence is typically very slow (and it does not even converge in general for a fixed step size).

One way to deal with this is to add a smooth *regularization* term. Specifically, it is easy to show that if $\boldsymbol{x}^\star$ is a minimizer of $f(\boldsymbol{x})$, then it is also the minimizer of

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}}\, f(\boldsymbol{x}) + \delta\|\boldsymbol{x}-\boldsymbol{x}^\star\|_2^2,$$

where $\delta > 0$. While the resulting optimization problem is still nonsmooth, it is now strongly convex, and we know that strongly convex functions are generally much easier to minimize. The "only" challenge is that it requires us to already know the solution $\boldsymbol{x}^\star$, which would seem to limit the practical applicability of this idea.

We can turn this into an actual algorithm by adopting an iterative approach. The **proximal algorithm** or **proximal point method** uses the following iteration:[1]

$$\boldsymbol{x}_{k+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^N}\left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|_2^2\right). \tag{1}$$

As noted above, when f is convex, $f(\boldsymbol{x}) + \delta\|\boldsymbol{x}-\boldsymbol{z}\|_2^2$ is strictly convex for all $\delta > 0$ and $\boldsymbol{z}\in\mathbb{R}^N$, so the mapping from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$ is well-defined. We will sometimes use the "prox operator" to denote this mapping:

$$\text{prox}_{\alpha_k f}(\boldsymbol{z}) = \arg\min_{\boldsymbol{x}\in\mathbb{R}^N}\left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2\right).$$

[1] The notation $\arg\min_{\boldsymbol{x}\in\mathbb{R}^N} f(\boldsymbol{x})$ denotes a function that returns the vector $\boldsymbol{x}\in\mathbb{R}^N$ that minimizes $f(\boldsymbol{x})$. This is in contrast to $\min_{\boldsymbol{x}\in\mathbb{R}^N} f(\boldsymbol{x})$, which refers to the minimum possible value for $f(\boldsymbol{x})$.

At this point, you would be forgiven for having doubts about what we are really doing here. We have taken an optimization problem and turned it into... a sequence of *many* optimization problems. However, these problems can sometimes be far easier to solve that the original problem. One way to think about the additional $\frac{1}{2\alpha_k}\|\boldsymbol{x}-\boldsymbol{x}_k\|$ term is as a *regularizer* that makes each subproblem computationally easier to solve, and whose influence naturally disappears as we approach the solution, even for a fixed "step size" $\alpha_k = \alpha$.

Implicit gradient descent ("backward Euler")

The proximal point method can also be interpreted as a variation on gradient descent. To see this, let us return for a moment to the differential equations for the "gradient flow" of f:

$$\boldsymbol{x}'(t) = -\nabla f(\boldsymbol{x}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0. \tag{2}$$

The equilibrium points for this system are the $\boldsymbol{x}$ such that $\nabla f(\boldsymbol{x}) = \mathbf{0}$, which are precisely the minimizers for $f(\boldsymbol{x})$.

As we first discussed in the context of momentum-based methods, we can interpret gradient descent as a first-order numerical method for tracing the path from $\boldsymbol{x}_0$ to a solution $\boldsymbol{x}^\star$. This comes from discretizing the derivative on the right using a forward finite difference:

$$\frac{\boldsymbol{x}(t+h) - \boldsymbol{x}(t)}{h} \approx -\nabla f(\boldsymbol{x}(t)) \quad \text{for small } h.$$

Thus the gradient descent iterations

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h\nabla f(\boldsymbol{x}_k)$$

approximate the solution at equispaced times spaced h seconds apart – the step size in gradient descent can be interpreted as the time scale

to which we are approximating the derivative. This is known as the *forward Euler method* for discretizing (2).

But now suppose we used a *backward difference* to approximate the derivative:

$$\frac{\boldsymbol{x}(t) - \boldsymbol{x}(t-h)}{h} \approx -\nabla f(\boldsymbol{x}(t)) \quad \text{for small } h.$$

Now the iterates must obey

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - h\nabla f(\boldsymbol{x}_{k+1}).$$

This is an equally valid technique for discretizing (2) known as the *backward Euler method*. However, computing the iterates is not as straightforward – we can't just compute the gradient at the current point, we have to find the next point by finding an $\boldsymbol{x}_{k+1}$ that obeys the equation above.

This is exactly what the proximal operator does. If f is differentiable, then

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\ &\Updownarrow \\ \boldsymbol{0} &= \nabla f(\boldsymbol{x}_{k+1}) + \frac{1}{\alpha_k}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k). \end{aligned} \tag{3}$$

So the proximal point method can be interpreted as a backward Euler discretization for gradient flow.

Note that we assumed the differentiability of f above purely for illustration; we can compute the prox operator whether or not f has a gradient.

Example: Least squares

Suppose we want to solve the standard least-squares problem

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

When $\boldsymbol{A}$ has full column rank, we know that the solution is given by $\widehat{\boldsymbol{x}}_{\text{ls}} = (\boldsymbol{A}^{\text{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\text{T}}\boldsymbol{y}$. However, we also know that when $\boldsymbol{A}^{\text{T}}\boldsymbol{A}$ is not well-conditioned, this inverse can be unstable to compute, and iterative descent methods (gradient descent and conjugate gradients) can take many iterations to converge.

Consider the proximal point iteration (with fixed $\alpha_k = \alpha$) for solving this problem:

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}\in\mathbb{R}^N}{\arg\min}\left(\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \frac{1}{2\alpha}\|\boldsymbol{x}-\boldsymbol{x}_k\|_2^2\right).$$

Here we have the closed form solution

$$\begin{aligned}\boldsymbol{x}_{k+1} &= (\boldsymbol{A}^{\text{T}}\boldsymbol{A}+\delta\mathbf{I})^{-1}(\boldsymbol{A}^{\text{T}}\boldsymbol{y}+\delta\boldsymbol{x}_k), \quad \delta = \frac{1}{\alpha} \\ &= \boldsymbol{x}_k + (\boldsymbol{A}^{\text{T}}\boldsymbol{A}+\delta\mathbf{I})^{-1}\boldsymbol{A}^{\text{T}}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}_k).\end{aligned}$$

Now each step is equivalent to solving a least-squares problem, but this problem can be made well-conditioned by choosing δ (i.e., α) appropriately. The iterations above will converge to $\widehat{\boldsymbol{x}}_{\text{ls}}$ for any value of α; as we decrease α (increase δ), the number of iterations to get within a certain accuracy of $\widehat{\boldsymbol{x}}_{\text{ls}}$ increases, but the least-squares problems involved are all very well conditioned. For α very small, we are back at gradient descent (with step size α).

This is actually a well-known technique in numerical linear algebra called *iterative refinement*.

Proximal gradient algorithms

Recall the core update equation for the proximal point method:

$$\boldsymbol{x}_{k+1} = \text{prox}_{\alpha_k f}(\boldsymbol{x}_k) = \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(f(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right).$$

Suppose that we did not wish to fully solve this problem at each iteration. If f is differentiable, we could approximate this update by replacing $f(\boldsymbol{x})$ with its linear approximation $f(\boldsymbol{x}_k)+\langle \boldsymbol{x}-\boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)\rangle$. This would yield the update

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(f(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)\rangle + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(\frac{\alpha_k}{2}\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla f(\boldsymbol{x}_k)\rangle + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\
&= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(\frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k + \alpha_k \nabla f(\boldsymbol{x}_k)\|_2^2 \right) \\
&= \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k).
\end{aligned}$$

Thus, taking a linear approximation of f, the proximal method simply reduces to standard gradient descent.

Where this starts getting interesting is when we encounter optimization problems where the objective function can be broken into the sum two parts, i.e.,

$$f(\boldsymbol{x}) = g(\boldsymbol{x}) + h(\boldsymbol{x}),$$

where both g and h are convex, but g is smooth (differentiable) and h is a non-smooth function for which there is a fast proximal operator. Such optimization problems quite a bit more often than you might expect.

The **proximal gradient** algorithm is the result of applying the proximal point method to minimize the approximation of f where we take a linear approximation to the smooth component g. Using the same argument as above, this results in the update rule

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(g(\boldsymbol{x}_k) + \langle \boldsymbol{x} - \boldsymbol{x}_k, \nabla g(\boldsymbol{x}_k)\rangle + h(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 \right) \\ &= \arg\min_{\boldsymbol{x}\in\mathbb{R}^N} \left(h(\boldsymbol{x}) + \frac{1}{2\alpha_k}\|\boldsymbol{x} - \boldsymbol{x}_k + \alpha_k \nabla g(\boldsymbol{x}_k)\|_2^2 \right) \\ &= \operatorname{prox}_{\alpha_k h}\left(\boldsymbol{x}_k - \alpha_k \nabla g(\boldsymbol{x}_k)\right).\end{aligned}$$

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 our friend the LASSO:

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

We take

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

and

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

The prox operator for the ℓ_1 norm is:

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

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

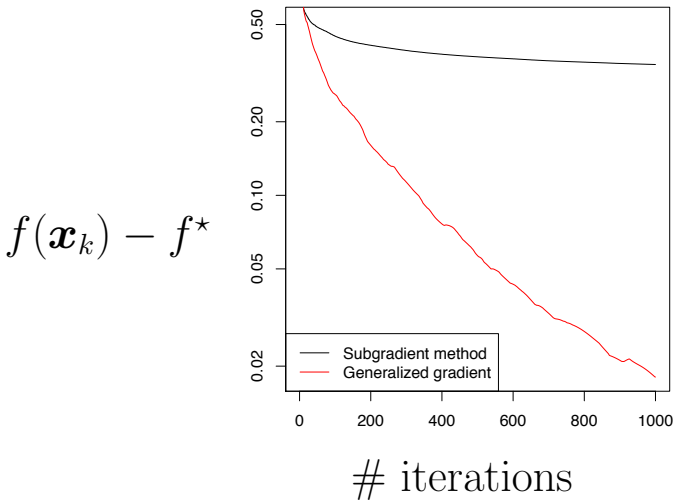
$$[T_{\tau\alpha}(\boldsymbol{z})]_n = \begin{cases} z_n - \tau\alpha, & z_n \geq \tau\alpha, \\ 0, & |z_n| \leq \tau\alpha, \\ z_n + \tau\alpha, & z_n \leq -\tau\alpha. \end{cases}$$

Hence, 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

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

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

Here is a comparison[2] of a typical run for ISTA versus the subgradient method. ISTA absolutely crushes the subgradient method.



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

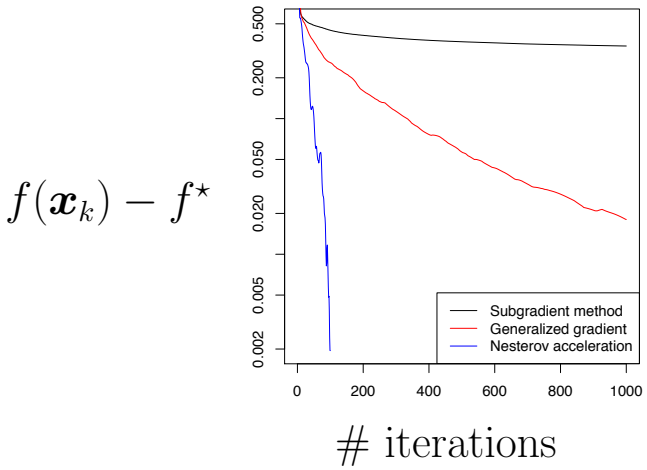
Accelerated proximal gradient

We can accelerate the proximal gradient method in exactly the same way we accelerated gradient descent – in fact, the Nesterov's method for gradient descent is simply a special case as that for the proximal gradient algorithm. The accelerated iteration is

$$\begin{aligned} \boldsymbol{p}_k &= \frac{k-1}{k+2}\left(\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\right) \\ \boldsymbol{x}_{k+1} &= \text{prox}_{\alpha_k h}\left(\boldsymbol{x}_k + \boldsymbol{p}_k - \alpha_k \nabla g(\boldsymbol{x}_k + \boldsymbol{p}_k)\right). \end{aligned}$$

Again, the computations here are in general no more involved than for the non-accelerated version, but the number of iterations can be significantly lower. We will not prove it here (the proof is straightforward, but long), but adding in the momentum term results in convergence rate of $O(1/k^2)$ using a similar argument as before.

The numerical performance can also be dramatically better. Here are typical runs[3] for the LASSO, which compares the standard proximal gradient method (ISTA) to its accelerated version (FISTA):



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

Convergence of the proximal gradient method

The convergence analysis of the proximal gradient method is extremely similar to what we did for gradient descent. In fact, gradient descent is a special case of the proximal gradient method (when $h(\boldsymbol{x}) = 0$), and our analysis will recover the same result. We will assume that g is M-smooth, but we will make no assumptions on h aside from convexity. As before, we will use a fixed "step size", $\alpha_k = 1/M$ for all k. We will $\boldsymbol{x}^\star$ denote any minimizer of f.

The general structure of the argument is as follows:

1. Using the M-smoothness of g as well as the first-order characterization of convexity, we can establish that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k \rangle - \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2 \tag{4}$$

   for all $\boldsymbol{z} \in \mathbb{R}^N$ where $\boldsymbol{d}_k := M(\boldsymbol{x}_k - \boldsymbol{x}_{k+1})$.

2. From (4) we can conclude, by setting $\boldsymbol{z} = \boldsymbol{x}_k$, that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2 \leq f(\boldsymbol{x}_k),$$

   and thus $f(\boldsymbol{x}_k)$ is non-increasing at every step.

3. From (4) we can also conclude, by setting $\boldsymbol{z} = \boldsymbol{x}^\star$, that

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}^\star) + \langle \boldsymbol{x}_k - \boldsymbol{x}^\star, \boldsymbol{d}_k \rangle - \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2.$$

   By exactly the same argument as we have seen in the analysis of both gradient descent and Nesterov's method, we can show that this bound is equivalent to

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^\star) \leq \frac{M}{2}\left(\|\boldsymbol{x}_k - \boldsymbol{x}^\star\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}^\star\|_2^2\right).$$

4. This yields a telescopic sum, and hence by an identical argument to that used in analyzing gradient descent, we arrive at the bound

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^\star) \leq \frac{M}{2k}\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|_2^2.$$

Thus, the proximal gradient algorithm exhibits the same convergence rate as gradient descent: $O(1/k)$. This is remarkable when considering that it holds for *any* h. This result is in fact a kind of "master result" for the convergence rate of many different algorithms:

- gradient descent (take $h(\boldsymbol{x}) = 0$),
- the proximal point method (take $g(\boldsymbol{x}) = 0$),
- the proximal gradient method.

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

Note that the only novelty in the analysis above compared to that of gradient descent is the derivation of (4). To establish this inequality, we first note that

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) &= g(\boldsymbol{x}_{k+1}) + h(\boldsymbol{x}_{k+1}) \\ &\leq g(\boldsymbol{x}_k) - \frac{1}{M}\langle \boldsymbol{d}_k, \nabla g(\boldsymbol{x}_k)\rangle + \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2 + h(\boldsymbol{x}_{k+1}), \end{aligned} \quad (5)$$

where the inequality follows directly from the definition of M-smoothness. We now use two facts to get an upper bound on this expression. First, note that from the first-order characterization of convexity,

$$g(\boldsymbol{z}) \geq g(\boldsymbol{x}_k) + \langle \boldsymbol{z} - \boldsymbol{x}_k, \nabla g(\boldsymbol{x}_k)\rangle. \quad (6)$$

Second, since

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

we know

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

Thus

$$h(\boldsymbol{z}) \geq h(\boldsymbol{x}_{k+1}) + \langle \boldsymbol{z} - \boldsymbol{x}_{k+1}, \boldsymbol{d}_k - \nabla g(\boldsymbol{x}_k)\rangle. \tag{7}$$

We combine (6) and (7) back into (5) to obtain

$$\begin{aligned}f(\boldsymbol{x}_{k+1}) &\leq g(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \nabla g(\boldsymbol{x}_k)\rangle - \frac{1}{M}\langle \boldsymbol{d}_k, \nabla g(\boldsymbol{x}_k)\rangle + \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2 \\ &\quad + h(\boldsymbol{z}) - \left\langle \boldsymbol{z} - \boldsymbol{x}_k + \frac{1}{M}\boldsymbol{d}_k, \boldsymbol{d}_k - \nabla g(\boldsymbol{x}_k)\right\rangle \\ &= f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k\rangle + \frac{1}{M}\left(\frac{M}{2M} - 1\right)\|\boldsymbol{d}_k\|_2^2 \\ &\leq f(\boldsymbol{z}) + \langle \boldsymbol{x}_k - \boldsymbol{z}, \boldsymbol{d}_k\rangle - \frac{1}{2M}\|\boldsymbol{d}_k\|_2^2,\end{aligned}$$

which establishes (4).