10-725: Convex Optimization **Spring 2023**

Lecture 6: February 2nd

Lecturer: Siva Balakrishnan

Today our focus will be first on constrained optimization, and then on a generalization of constrained optimization problems which we will refer to as proximal optimization problems.

6.1 Projected Gradient Descent

Suppose our goal is to now minimize a convex function f, over a convex set C, i.e. we want to solve:

$$\min_{x \in C} f(x).$$

A natural first attempt might be to ignore the constraint and run gradient descent (or the subgradient method), but of course this won't solve our constrained problem in general. Since our iterates could leave the set C we could end up at an infeasible solution.

The natural second attempt might be to simply take steps of GD or the subgradient method, but then project iterates back to the set C whenever they leave the set C. If the objective is differentiable this is the projected GD algorithm, i.e. for a step-size η, we iterate:

$$\begin{aligned} y^{t+1} &= x^t - \eta \nabla f(x^t), \\ x^{t+1} &= P_C(y^{t+1}). \end{aligned}$$

Here the projection step finds the point in C closest to y^{t+1}, i.e. it solves the optimization problem:

$$x^{t+1} = P_C(y^{t+1}) := \arg\min_{x \in C} \frac{1}{2}\|x - y^{t+1}\|_2^2.$$

Assuming this is a sensible method (it's not clear that it is) projected GD reduces solving one optimization problem to solving a sequence of optimization problems. Perhaps the immediate question is for what types of sets can we efficiently project onto. You'll explore some interesting examples in your HW.

Here is a very simple one: suppose we're solving an optimization problem but would like to constrain all the optimization variables to be positive. For instance, the problem of non-negative least squares is to solve:

$$\min_{x \geq 0} \frac{1}{2}\|Ax - b\|_2^2.$$

Here in order to solve the projection problem we need to solve the following problem: given a vector y^{t+1}, find x^{t+1} which has all positive entries which is closest to y^{t+1}. This has a simple closed form:

$$x^{t+1} = \max\{0, y^{t+1}\},$$

where the max operation is applied elementwise. This is a simple example where projected GD is straightforward to implement, but it's still unclear if the algorithm inherits any of the properties of GD (in the unconstrained case).

6.2 Analyzing the Projected Subgradient Method

We'll do an example analyzing the projected subgradient method, and you'll work out a couple of other cases (analyzing projected GD) on your HW.

Theorem 6.1 *Suppose that f is convex and G-Lipschitz, and define x^{best} to be the best iterate seen so far and choose step-size η_t in each round, then we have the guarantee:*

$$f(x^{best}) - f(x^*) \leq \frac{\|x^0 - x^*\|_2^2 + G^2 \sum_{t=0}^{k-1} \eta_t^2}{2\sum_{t=0}^{k-1} \eta_t}.$$

This is exactly the same guarantee that we had before, and more generally projected methods often inherit properties of their unconstrained counterparts. The major caveat is that each iteration of the projected algorithm needs to solve a projection step.

Proof: The proof will almost exactly mirror our proof for the subgradient method. Lets first observe that,

$$\begin{aligned}\|y^{t+1} - x^*\|_2^2 &= \|x^t - \eta_t g_{x^t} - x^*\|_2^2 \\ &= \|x^t - x^*\|_2^2 + \eta_t^2 \|g_{x^t}\|_2^2 - 2\eta_t g_{x^t}^T (x^t - x^*) \\ &\leq \|x^t - x^*\|_2^2 + \eta_t^2 G^2 + 2\eta_t (f(x^*) - f(x^t)),\end{aligned}$$

where the last step uses convexity. Now, we can re-arrange, and sum from $t = 0, \ldots, k$ to obtain that,

$$\sum_{t=0}^{k} 2\eta_t (f(x^t) - f(x^*)) \leq \sum_{t=0}^{k} \eta_t^2 G^2 + \sum_{t=0}^{k} \left(\|x^t - x^*\|_2^2 - \|y^{t+1} - x^*\|_2^2 \right).$$

Now we arrive at our main problem, which is that the second term on the right no longer telescopes.

The key observation is something we proved earlier, the projection onto a convex set is a contraction, i.e. recall we showed that if we have two point x, y then it is always the case that,

$$\|P_C(x) - P_C(y)\|_2 \leq \|x - y\|_2.$$

Now, observe that $x^* \in C$ (so projecting it onto C gives us the same point) so we have the immediate consequence:

$$\|P_C(y^{t+1}) - x^*\|_2 \leq \|y^{t+1} - x^*\|_2.$$

Returning to our bound we then see that,

$$\begin{aligned}\sum_{t=0}^{k} 2\eta_t(f(x^t) - f(x^*)) &\leq \sum_{t=0}^{k} \eta_t^2 G^2 + \sum_{t=0}^{k} \left(\|x^t - x^*\|_2^2 - \|x^{t+1} - x^*\|_2^2\right) \\ &\leq \sum_{t=0}^{k} \eta_t^2 G^2 + \|x^0 - x^*\|_2^2,\end{aligned}$$

and replacing $f(x^t)$ with $f(x^{\text{best}})$ and re-arranging gives us our main result. ■

In essence, the projection operation at each step only pushes us closer to the optimal point x^* which can only help the algorithm.

6.2.1 Observations

There are lots of interesting things about the projected gradient descent method (some of them should provoke you to think about the proximal method we'll talk about next). Suppose I was interested in solving the constrained form of the LASSO, i.e.

$$\min_{\|x\|_1 \leq L} \frac{1}{2}\|Ax - b\|_2^2,$$

then provided I can project efficiently onto the L1 ball (you can), then we might expect (this is true) that the projected gradient descent will behave like gradient descent on the least squares objective. When $A^T A$ is well-conditioned this means that projected GD will have a (fast) linear rate of convergence.

As you might know (and as we'll see more formally later on) minimizing this objective is very similar to solving the penalized form of the LASSO, i.e.

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1,$$

where λ and L have some correspondence (i.e. roughly, large values of $L > 0$ correspond to small values of $\lambda > 0$).

However, this latter problem is an example of an (unconstrained) non-smooth optimization problem. Our best bet (so far) would be to use the subgradient method (which we know has a very slow $1/\sqrt{k}$ rate of convergence at least in the worst-case).

It's a bit mysterious why folding the non-smoothness into a constraint helps us, and this motivates us to ask if there is an analogous way of dealing with "nice" non-smooth objective functions. It is worth noting that the projected GD algorithm is no longer a simple first-order algorithm (i.e. it operates in a very different "oracle" model, since it is not just using function values and gradients) and so is not subject to the rates of convergence lower bounds we discussed before.

6.3 Proximal Gradient Descent

Motivated by the above discussion, the particular problem class we'll focus on now will be: we want to minimize an unconstrained function f which can be written as the sum of a "nice", convex, function g and a potentially non-smooth convex function h, i.e.

$$\min_{x \in \mathbb{R}^d} g(x) + h(x).$$

The prox. GD algorithm alternates the following steps:

1. We compute $y^{t+1} = x^t - \eta_t \nabla g(x^t)$.
2. We then compute our next iterate by solving:

$$x^{t+1} = \arg\min_{x \in \mathbb{R}^d} \left[h(x) + \frac{1}{2\eta_t} \|x - y^{t+1}\|_2^2 \right].$$

6.3.1 The Proximal Operator

For a convex function f the proximal operator is defined to be:

$$\mathrm{prox}_f(v) = \arg\min_x \left(f(x) + \frac{1}{2}\|x - v\|_2^2 \right).$$

One can interpret this as some type of generalized projection operation, i.e. if you take a point v outside the domain of f the prox operator will return a point which is in the domain close to the original point (where the function f is small). If you are inside the domain then prox operator will move you towards the minimum of f, and if you're at the minimizer of f you will stay there.

More directly, if $f(x) = \mathbb{I}_C(x)$ is a convex indicator function, then $\text{prox}_f(v)$ simply projects v onto C.

With this notation, one can write the prox GD algorithm as:

1. We compute $y^{t+1} = x^t - \eta_t \nabla g(x^t)$.
2. We then compute our next iterate by solving:

$$x^{t+1} = \text{prox}_{\eta_t h}(y^{t+1}).$$

6.3.2 Step Sizes

As with any of the first-order algorithms we have studied so far an important practical choice will be that of the step-size. We'll as usual see some theoretical choices in the future. In practice, often backtracking is used. Usually it is used to ensure sufficient decrease in the smooth part of the objective. Describing the precise backtracking line search will be a bit easier once we've introduced the so-called gradient map so we'll return to it later.

6.3.3 An Example – ISTA

There are many nice examples of proximal algorithms that are popular in practice. Part of the revival of interest in prox GD came from its application to solving the LASSO – in this case the algorithm goes by the name Iterative Shrinkage Thresholding Algorithm (ISTA).

If we want to use the prox GD algorithm to solve the LASSO:

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1,$$

we would:

1. Compute $y^{t+1} = x^t - \eta_t A^T(Ax^t - b)$.
2. Then we would need to solve the prox problem:

$$x^{t+1} = \arg\min_{x \in \mathbb{R}^d} \frac{1}{2\eta_t}\|x - y^{t+1}\|_2^2 + \lambda \|x\|_1.$$

   Fortunately, we seen the solution to this problem in closed form. We set

$$x^{t+1} = S_{\lambda \eta_t}(y^{t+1}),$$

   where S_λ is the soft-thresholding operation we have seen before.

This algorithm is very natural – we simply take a gradient step (ignoring the non-smooth part) and then soft-threshold the iterates (to account for the non-smooth part). It should however be pretty mysterious as to why this algorithm works (is the LASSO solution even a fixed point?).

6.3.4 Motivating Prox GD

There are many different ways to motivate the proximal gradient descent algorithm:

1. **As a generalization of projected GD:** In projected GD, we wanted to solve the constrained problem:

$$\min_{x \in C} f(x),$$

where f was a differentiable function which could be written as:

$$\min_{x \in \mathbb{R}^d} f(x) + \mathbb{I}_C(x),$$

which is exactly of the form "nice function" + non-smooth function. Minimizing this with the prox GD algorithm will simply recover the projected GD algorithm (because the prox operation is simply a projection). So one can view prox GD as a natural extension of projected GD to other types of smooth, non-smooth sums.

2. **The Local Approximation Viewpoint:** Another viewpoint on the proximal GD algorithm is that in order to minimize a sum $f = g + h$ one can locally approximate just the smooth part and carry out the minimization, i.e. we can attempt to minimize:

$$x^{t+1} = \arg\min_{x \in \mathbb{R}^d} \left[g(x^t) + \nabla g(x^t)^T (x - x^t) + \frac{1}{2\eta_t} \|x - x^t\|_2^2 + h(x) \right].$$

In the above expression we have approximated the smooth part (by a linear approximation) and left the non-smooth part untouched.

Now, it is easy to see that this is equivalent to writing:

$$x^{t+1} = \arg\min_{x \in \mathbb{R}^d} \left[\frac{1}{2\eta_t} \|x - (x^t - \eta_t \nabla g(x^t))\|_2^2 + h(x) \right].$$

This is precisely the prox GD algorithm.

3. **Connection to Expectation-Maximization (EM)/Majorization-Minimization (MM) algorithms:** If we have a β-smooth function g, and we choose $\eta_t \leq 1/\beta$ then we know that,

$$g(x) \leq g(x^t) + \nabla g(x^t)^T (x - x^t) + \frac{1}{2\eta_t} \|x - x^t\|_2^2.$$

> As a consequence, our prox GD algorithm in this case is simply finding a "nice" convex upper bound on our function, which is tight at $x = x^t$, and minimizing this upper bound. Algorithms that do this are called MM algorithms, and have lots of nice properties. The EM algorithm is a special case.

Ignore this comment: Another viewpoint that we might return to at some point (that you can look up now if you're curious) is that one can view the prox GD algorithm as a different discretization of gradient flow (gradient flow is the limit of GD when the step-size goes to zero). Discretizing gradient flow in the "usual", explicit way gives the GD algorithm, but discretizing it "backwards" gives the prox GD algorithm.

6.3.5 Some Special Cases

In the prox algorithm:

1. When $h = 0$ this is just regular GD.
2. When $h = \mathbb{I}_C$ this is just projected GD (which we have some understanding of).
3. When $g = 0$ this is a different algorithm (that pre-dates prox GD) known as proximal minimization, where to minimize some function f, we regularize it a bit and minimize it iteratively, i.e. we solve

   $$x^{t+1} = \text{prox}_{\eta f}(x^t).$$

   This iteration will converge to a minimizer of f under weak conditions (and is sometimes easier to solve than the original problem because of the regularization).

Now that we've derived the algorithm and understood some simple special cases, it remains to really make sense of it.