Course Notes for EE227C (Spring 2018): Convex Optimization and Approximation

Instructor: Moritz Hardt

Email: `hardt+ee227c@berkeley.edu`

Graduate Instructor: Max Simchowitz

Email: `msimchow+ee227c@berkeley.edu`

March 1, 2018

14 Lecture 14: Algorithms using Duality

The Lagrangian duality theory from the previous lecture can be used to design improved optimization algorithms which perform the optimization on the dual function. Oftentimes, passing to the dual can simplify computation or enable parallelization.

14.1 Review

Recall the *primal problem*

$$\min_{x\in\Omega} f(x) \tag{1}$$

$$\text{s.t. } Ax = b \tag{2}$$

The corresponding dual problem is obtained by considering the *Lagrangian*

$$L(x,\lambda) = f(x) + \lambda^T(Ax - b)$$

where λ_i are called *Lagrange multipliers*. The *dual function* is defined as

$$g(\lambda) = \inf_{x\in\Omega} L(x,\lambda)$$

and the *dual problem* is

$$\sup_{\lambda\in\mathbb{R}^m} g(\lambda)$$

Definition 14.1 (Concave functions)**.** A function f is concave $\iff$ $-f$ is convex.

Fact 14.2. *The dual function is always concave (even if f and Ω are not convex).*

Proof. For any $x \in \Omega$, $L(x, \lambda)$ is a linear function of λ so $g(\lambda)$ is an infimum over a family of linear functions, hence concave. ■

14.2 Dual gradient ascent

Concavity of the dual function $g(\lambda)$ ensures existence of subgradients, so the subgradient method can be applied to optimize $g(\lambda)$. The *dual gradient ascent* algorithm is as follows:

Start from initial λ_0. For all $t \geqslant 0$:

$$x_t = \arg\inf_{x \in \Omega} L(x, \lambda_t)$$

$$\lambda_{t+1} = \lambda_t + \eta(Ax_t - b)$$

This yields the $O(1/\sqrt{t})$ convergence rate obtained by the subgradient method.

14.3 Augmented Lagrangian method / method of multipliers

Whereas dual gradient ascent updates λ_{t+1} by taking a step in a (sub)gradient direction, a method known as the *dual proximal method* can be motivated by starting with using the proximal operator [PB+14] as an update rule for iteratively optimizing λ:

$$\lambda_{t+1} = \text{prox}_{\eta_t g}(\lambda_t) = \arg\sup_{\lambda} \underbrace{\underbrace{\inf_{x \in \Omega} f(x) + \lambda^T(Ax - b)}_{g(\lambda)} - \underbrace{\frac{1}{2\eta_t}\|\lambda - \lambda_t\|^2}_{\text{proximal term}}}_{h(\lambda)}$$

Notice that this expression includes a proximal term which makes $h(\lambda)$ strongly convex.

However, this update rule is not always directly useful since it requires optimizing $h(\lambda)$ over λ, which may not be available in closed form. Instead, notice that if we can interchange inf and sup (e.g. strong duality, Sion's theorem applied when Ω is compact) then we can rewrite

$$\sup_{\lambda} \inf_{x \in \Omega} f(x) + \lambda^T(Ax - b) - \frac{1}{2\eta_t}\|\lambda - \lambda_t\|^2 = \inf_{x \in \Omega} \sup_{\lambda} f(x) + \lambda^T(Ax - b) - \frac{1}{2\eta_t}\|\lambda_t - \lambda\|^2 \tag{3}$$

$$= \inf_{x \in \Omega} f(x) + \lambda_t^T(Ax - b) + \frac{\eta_t}{2}\|Ax - b\|^2 \tag{4}$$

where the inner sup is optimized in closed-form by $\lambda = \lambda_t + \eta_t(Ax - b)$. To isolate the remaining optimization over x, we make the following definition.

Definition 14.3 (Augmented Lagrangian)**.** The *augmented Lagrangian* is

$$L_\eta(x, \lambda) = f(x) + \lambda_t^T (Ax - b) + \frac{\eta_t}{2}\|Ax - b\|^2$$

The *augmented Lagrangian method* (aka Method of Multipliers) is defined by the following iterations:

$$x_t = \arg \inf_{x \in \Omega} L_{\eta_t}(x, \lambda_t)$$
$$\lambda_{t+1} = \lambda_t + \eta_t (Ax_t - b)$$

While the iterations look similar to dual gradient ascent, there are some noteworthy differences

- The method of multipliers can speed up convergence (e.g. for non-smooth functions), but computing x_t may be more difficult due to the additional $\frac{\eta_t}{2}\|Ax - b\|^2$ term in the augmented Lagrangian
- $L(x, \lambda_t)$ is convex in x, but $L_\eta(x, \lambda_t)$ is strongly convex in λ (if A is full-rank)
- Convergence at a $O(1/t)$ rate. More precisely, for constant step size η, we can show show the method of multipliers satisfies

$$g(\lambda_t) - g^* \geqslant -\frac{\|\lambda^*\|^2}{2\eta t}$$

14.4 Dual decomposition

A major advantage of dual decomposition that it can lead to update rules which are trivially parallelizable.

Suppose we can partition the primal problem into blocks of size $(n_i)_{i=1}^N$, i.e.

$$x^T = ((x^{(1)})^T, \cdots, (x^{(N)})^T) \qquad x_i \in \mathbb{R}^{n_i}, \sum_{i=1}^N n_i = n \tag{5}$$

$$A = [A_1 | \cdots | A_N] \qquad Ax = \sum_{i=1}^N A_i x^{(i)} \tag{6}$$

$$f(x) = \sum_{i=1}^N f_i(x^{(i)}) \tag{7}$$

Then the Lagrangian is also separable in x

$$L(x, \lambda) = \sum_{i=1}^N \left(f_i(x^{(i)}) + \lambda^T A_i x^{(i)} - \frac{1}{N}\lambda^T b \right) = \sum_{i=1}^N L_i(x^{(i)}, \lambda)$$

Each term in the sum consists of one non-interacting partition $(x^{(i)}, A_i, f_i)$, so minimization of each term in the sum can occur in parallel. This leads to the *dual decomposition algorithm*:

- In parallel on worker nodes: $x_t^{(i)} = \arg\inf_{x^{(i)}} L_i(x^{(i)}, \lambda_t)$
- On a master node: $\lambda_{t+1} = \lambda_t + \eta(Ax - b)$

Example 14.4 (Consensus optimization)**.** Consensus optimization is an application that comes up in distributed computing which can be solved using dual decomposition. Given a graph $G = (V, E)$,

$$\min_x \sum_{v \in V} f_v(x_v) : x_v = x_u \ \forall (u, v) \in E$$

This problem is separable over $v \in V$, so dual decomposition applies.

Example 14.5 (Network utility maximization)**.** Suppose we have a network with k links, each with capacity c_i. We are interested in allocating N different flows with fixed routes over these links such that utility is maximized and resource constraints are not exceeded. Let $x_i \in \mathbb{R}^N$ represent the amount of flow i allocated and $U_i : \mathbb{R} \to \mathbb{R}$ a convex utility function which returns the amount of utility obtained from having x_i amount of flow i. The optimization problem is

$$\max_x \sum_{i=1}^N U_i(x_i) : Rx \leqslant c$$

where R is a $k \times N$ matrix whose (k, i)th entry gives the amount of the capacity of link k is consumed by flow i.

To rewrite the primal problem in standard form (i.e. as a minimization), take negatives:

$$\min_x - \sum_i U_i(x^{(i)}) : Rx \leqslant c$$

The dual problem is then

$$\max_{\lambda \geqslant 0} \min_x \sum_i -U_i(x^{(i)}) + \lambda^T(Rx - c)$$

where the $Rx \leqslant c$ primal inequality constraint results in the $\lambda \geqslant 0$ constraint. The second term can be rewritten as $\lambda^T \left(\sum_i R_i x_i - \frac{1}{N} c \right)$, showing that the dual splits over i and hence dual decomposition applies. This leads to a parallel algorithm which each worker node computes

$$\arg\max_{x_i} U_i(x_i) - \lambda^T R_i x_i$$

and the master node computes

$$\lambda_{t+1} = [\lambda_t + \eta(Rx - c)]_+$$

We take the positive part because of the $\lambda \geqslant 0$ constraint.

Aside: In resource allocation problems, the values of the dual variables λ at the optimal point have an economic interpretation as "prices" to the resources. In this example, λ_k should be interpreted as the price per unit of flow over link k.

14.5 ADMM — Alternating direction method of multipliers

While dual decomposition can be used to parallelize dual subgradient ascent, it doesn't work with the augmented Lagrangian. This is because the coupling between the decision variables introduced by the $\|Ax - b\|^2$ term prevents the augmented Lagrangian from being separable over x.

The goal of the alternating direction method of multipliers (ADMM) is to obtain the best of both worlds: we would like both the parallelism offered by the method of multipliers as well as the faster convergence rate of the augmented Lagrangian. We will see that similar to dual decomposition, ADMM partitions the decision variables into two blocks. Also, similar to the method of multiplers, ADMM uses the augmented Lagrangian $L_\eta(x, z, \lambda_t)$.

Consider a problem of the form

$$\min_{x,z} f(x) + g(z) : Ax + Bz \leqslant c$$

In other words, we can split the objective and constraints into two blocks x and z.

The method of multipliers would jointly optimize the augmented Lagrangian on both blocks in one single optimization step:

$$(x_{t+1}, z_{t+1}) = \inf_{x,z} L_\eta(x, z, \lambda_t) \tag{8}$$

$$\lambda_{t+1} = \lambda_t + \eta(Ax_{t+1} + Bz_{t+1} - c) \tag{9}$$

In contrast, ADMM alternates (the "A" in "ADMM") between optimizing the augmented Lagrangian over x and z:

$$x_{t+1} = \inf_{x} L_\eta(x, z_t, \lambda_t) \tag{10}$$

$$z_{t+1} = \inf_{z} L_\eta(x_{t+1}, z, \lambda_t) \tag{11}$$

$$\lambda_{t+1} = \lambda_t + \eta(Ax_{t+1} + Bz_{t+1} - c) \tag{12}$$

Unlike the method of multipliers, this is not parallelizable since x_{t+1} must be computed before z_{t+1}. Also, convergence guarantees are weaker: rather than getting a convergence rate we only get an asymptotic convergence guarantee.

Theorem 14.6. *Assume*

- *f, g have a closed, non-empty, convex epigraph*
- *L_0 has a saddle x^*, z^*, λ^*, i.e.:*

$$\forall x, z, \lambda : L_0(x^*, z^*, \lambda) \leqslant L_0(x^*, z^*, \lambda^*) \leqslant L(x, z, \lambda^*)$$

Then, as $t \to \infty$, ADMM satisfies

$$f(x_t) + g(z_t) \to p^* \tag{13}$$

$$\lambda_t \to \lambda^* \tag{14}$$

Aside: Saddles are useful because inf and the sup can be swapped. To see this, note the saddle condition

$$L(x^*, \lambda) \leqslant L(x^*, \lambda^*) \leqslant L(x, \lambda^*)$$

implies that

$$\begin{aligned}
\inf_x \sup_\lambda L(x, \lambda) &\leqslant \sup_\lambda L(x^*, \lambda) && (15)\\
&\leqslant L(x^*, \lambda^*) && (16)\\
&= \inf_x L(x, \lambda^*) && (17)\\
&\leqslant \sup_\lambda \inf_x L(x, \lambda) && (18)
\end{aligned}$$

References

[PB$^+$14] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.