Convex Optimization 2: Algorithms (CS5660)

Instructor: Saketh

January 5, 2024

Contents

Chapter 1

Introduction

We began by introducing the framework/model for studying algorithms for solving convex programs. This is analogous to that in classical CS algorithms. Let us denote by $C_L^{k,p}$, the set of all convex[1] functions over $\mathbb{R}^n$ that[2] are k^{th} order continuously differentiable[3] and whose p^{th} derivative is L-Lipschitz[4]. Below are the key components of this model:

Class of Programs: This is a specific set of (interesting) mathematical programs. In this course we will focus on sub-classes of Convex Programs[5] (CPs) like: Unconstrained CPs with objective in $C_L^{k,p}$ etc.

Class of Algorithms: This is a set of numerical procedures[6] (algorithms) that have access to exactly the same amount of information about the programs they intend to solve. Initially, we will focus on so-called First Order Black Box methods: methods that access an unconstrained program (to be solved) only via an oracle that provides gradient[7] of objective at x, given x.

Success: An algorithm is said to solve a program successfully if it is guaranteed to return a $\hat{x}$ in the feasibility set such that $f(\hat{x}) - f(x^*) \leq \epsilon$ for any given $\epsilon > 0$. This condition at a specific $\epsilon > 0$ is known as the ϵ-optimality condition. Here, x^* is any optimal solution of the program (to be solved).

[1] Refer Lecture 17 in `https://www.overleaf.com/project/5c2d7deeeb7e57626953789f`.

[2] We always consider extended functions where $f(x) \equiv \infty \; \forall \; x \notin dom(f)$.

[3] Continuously differentiable in the (relative) interior of their domain.

[4] We say f is L-Lipschitz iff $|f(x) - f(y)| \leq L||x - y|| \; \forall x, y \in dom(f)$

[5] Refer Lecture 22 in `https://www.overleaf.com/project/5c2d7deeeb7e57626953789f`.

[6] Typically iterative in nature, though not necessarily.

[7] Like Nesterov Nesterov [2014] (pg 7), we may also assume the oracle returns the function values too.

Analytical Complexity: This is the number of calls to the oracle made by an algorithm for achieving ϵ-optimality for a program, expressed as a function of ϵ. Needless to say, an algorithm is better if its analytical complexity is lower. The inverse function, i.e., ϵ written as a function of number of calls is called the convergence rate.

Optimal Algorithm: Given a class of programs and a class of algorithms, an algorithm in the given class (of algorithms) that is guaranteed to succeed on all programs in the given class (of programs) and has the worst-case analytical complexity (over the class of programs) $\leq$ to that with any other algorithm in the class is said to be an optimal algorithm[8].

Interested students, please refer section 1.1 in Nesterov [2014] for more details.

[8]Note that still it may happen that on the specific program you need to solve, an optimal algorithm may converge slower than a suboptimal one!

Chapter 2

(Sub)-Gradient Descent

We begin the study with a particular algorithm called gradient descent. Motivated by the very definition of derivative, we defined (vanilla) gradient descent as:

$$x^{(k+1)} \equiv x^{(k)} - \eta \nabla f(x^{(k)}), \forall\ k = 0, 1, 2, \dots \tag{2.1}$$

We also noted it's dual interpretation, which is not only insightful, but motivates various other algorithms:

$$x^{(k+1)} = \arg\min_{x \in \mathbb{R}^n} \frac{1}{2\eta} \left\| x - x^{(k)} \right\|^2 + f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\left(x - x^{(k)}\right),\ \forall\ k = 0, 1, 2, \dots \tag{2.2}$$

In simple words, gradient descent approximates function by its first order approximation and then minimizes this proxy in a regularized fashion.

Note that, even if the function is not differentiable, a sub-gradient exists, which is the same as gradient in the differentiable case. We hence use the same notation for gradient as well as sub-gradient. In either case we refer to the algorithm as gradient descent, while we may sometimes qualify it as sub-gradient descent.

2.1 Unconstrained Non-smooth Convex Programs

Here, we assume that the objective is convex and is L-Lipschitz continuous. We presented two different yet related derivations:

1. theorem 3.2 in Bubeck [2015] (with $x_{t+1} = y_{t+1}$ in (3.3)).
2. theorem 3.2.2 in Nesterov [2014] (with $Q = \mathbb{R}^n$).

Interestingly, the derivations show that sub-gradient descent achieves *optimal*[1] convergence rate of $O\left(1/\sqrt{k}\right)$.

2.2 Unconstrained Smooth Convex Programs

Here, we assume that the objective is convex and is smooth, i.e., the gradient is L-Lipschitz. The interplay between these two structures (convexity and smoothness) leads to important inequalites as detailed in theorem 2.1.5 in Nesterov [2014]. The most important is (2.1.7), which provides a positive lower bound for the first order approximation.

We then presented theorem 2.1.14 in Nesterov [2014], which shows that gradient descent converges at rate $O(1/k)$, which is faster than that in the non-smooth case, nevertheless[2] (perhaps) sub-optimal.

2.3 Unconstrained Smooth Strongly Convex Programs

Here, we additionally assume that the objective is μ-strongly convex. This leads to the tighter bounds in 2.1.24 in Nesterov [2014]. We then presented theorem 2.1.15, which shows that the convergence is linear with rate $\frac{Q-1}{Q+1}$, where $Q \equiv \frac{L}{\mu}$, is known as the condition number. However, again this is (perhaps) sub-optimal[3]. Please refer theorem 3.10 Bubeck [2015] for a simplified version of this theorem.

[1] Theorem 3.2.1 in Nesterov [2014] shows that the lower bound is $O\left(1/\sqrt{k}\right)$.

[2] Theorem 2.1.7 Nesterov [2014] shows that $O(1/k^2)$ is a lower bound.

[3] Theorem 2.1.13 Nesterov [2014] shows lower bound of $\left(\frac{\sqrt{Q}-1}{\sqrt{Q}+1}\right)^2$

Chapter 3

Nesterov's Optimal Methods (Unconstrained)

We followed the derivation in section 3 in `https://arxiv.org/pdf/1407.1537.pdf`[1]. This gave us an insight that acceleration (i.e., $O(1/k^2)$ convergence rate) can be achieved by carefully mixing the strict decrease steps (like gradient descent majorization-minimization in the smooth case) with decrease on average case steps (like sub-gradient descent in non-smooth case). We concluded by intuitively describing the ODE based understanding of momentum methods: see `https://arxiv.org/pdf/1503.01243.pdf`. The momentum based methods are formally defined and analyzed in the original work of Nesterov in section 2.2 in Nesterov [2014]. However, this derivation is beyond the scope of this course.

`https://distill.pub/2017/momentum/` is a very nice and easy to read article on momentum methods. The last para (onwards and downwards) summarizes various explanations for this algo.

[1]Please read this section by replacing all occurences of the term "Mirror descent" by "sub-gradient descent" and $w(y) = \frac{1}{2}\|y\|^2$. This makes "mirror descent" same as sub-gradient descent.

Chapter 4

Projected Gradient Descent

Here we consider constrained minimization of the form $\min_{x \in \mathcal{F}} f(x)$. While it is not clear how to extend the primal definition of gradient descent (2.1), the dual definition (2.2) naturally generalizes to the following:

(4.1)

$$x^{(k+1)} \equiv \arg\min_{x \in \mathcal{F}} \frac{1}{2\eta} \left\| x - x^{(k)} \right\|^2 + f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\left(x - x^{(k)}\right), \ \forall \ k = 0, 1, 2, \ldots$$

In simple words, the objective function's first order approximation is minimized within the feasibility set, in a regularized fashion. Interestingly, (4.1) is same as the following (primal definition):

$$x^{(k+1)} \equiv \Pi_{\mathcal{F}}\left(x^{(k)} - \eta \nabla f(x^{(k)})\right), \forall \ k = 0, 1, 2, \ldots, \tag{4.2}$$

where $\Pi_{\mathcal{F}}(x)$ is the projection of x onto the set $\mathcal{F}$, defined by $\Pi_{\mathcal{F}}(x) \equiv \arg\min_{y \in \mathcal{F}} \|x - y\|_2^2$. The method defined by (4.2), equivalently, (4.1), is popularly known as Projected (Sub)Gradient Descent. The most interesting inequality that is satisfied by projection is formalized in Lemma 3.1 in Bubeck [2015]. This inequality turns out to be of critical importance in the convergence analysis below.

Given that the feasibility set $\mathcal{F}$ is simple enough that the projection onto it can be computed efficiently (i.e., comparable to that of gradient oracle), one can still compare algorithms using convergence rate. Ofcourse, now we additionally assume that all the algorithms have access to an appropriate projection oracle (for efficiently projecting onto the feasibility set). We then gave examples of feasibility sets where projection can be done in polynomial time:

1. n-dimensional 1-norm ball and simplex can be projected in $O(n \log(n))$ time using sorting [1].

[1] See Fig. 3 in https://link.springer.com/content/pdf/10.1023/A:1013637720281.pdf.

2. n-dimensional 2-norm ball projection can be done by simply normalizing the vector in $O(n)$ time.

3. n-dimensional ∞-norm ball projection can be done by soft-thresholding in $O(n)$ time i.e, $\Pi_{\Delta_\infty}(x) = \max(\min(x, 1), -1)$, where Δ_∞ is the infinity norm unit ball.

4. n size matrix nuclear norm ball and spectrahedron can be projected using EVD/SVD in $O(n^3)$ time[2].

The following analysis interestingly shows that the projected gradient descent is as fast as gradient descent in the unconstrained case!

4.1 Constrained, Non-smooth Convex Programs

We simply followed theorem 3.2 and proof in Bubeck [2015].

4.2 Constrained, Smooth Convex Programs

The key ingredient of the analysis in this case is the gradient mapping and the inequality satisfied by it, which is formalized in Lemma 3.6 in Bubeck [2015]. We followed the analysis in Theorem 3.7 in Bubeck [2015].

4.3 Constrained, Smooth and Strongly Convex Programs

We simply followed theorem 3.10 in Bubeck [2015].

Also see `https://stanford.edu/~jduchi/projects/DuchiShSiCh08.pdf` for an improved algorithm.

[2]See for e.g., `https://ee227c.github.io/code/lecture5.html`.

Chapter 5

Frank-Wolfe aka. Conditional Gradient Decent

We turned our attention to cases where there are no known algorithms for projection onto the given feasibility or the existing ones are too inefficient[1]. In such cases, one cannot use projected gradient descent. Conditional gradient or Frank-Wolfe (FW) algorithm, detailed below, is one alternative that may help.

The key idea in Conditional gradient descent is to decouple the regularization and linearization in (4.1) in projected gradient descent. More specifically, in Conditional gradient descent, minimization of unregularized linearization is performed leading to an intermediate iterate:

$$y^{(k+1)} \equiv \arg\min_{x \in \mathcal{F}} f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\left(x - x^{(k)}\right), \ \forall\ k = 0, 1, 2, \ldots \tag{5.1}$$

and this is regularized separately to obtain the actual iterate:

$$x^{(k+1)} = (1 - \eta)\, x^{(k)} + \eta y^{(k+1)}. \tag{5.2}$$

One key advantage with this decoupling is that (5.1) can be solved by computing the support function[2] of the feasibility set rather than projection onto it. Tables 1,2 in Jaggi [2013] provide a non-exhautive list of feasibility sets, whose support functions (and their (sub)gradients) can be evaluated in polynomial time.

[1] For example, consider the case where the gradient oracle computation takes linear time, wheres the feasibility set is a spectrahedron, whose projection time is more than linear.

[2] More specifically, if g is the support function of the feasibility set, then $y^{(k+1)} = \nabla g\left(-\nabla f\left(x^{(k)}\right)\right)$. ∇g as well as g can be computed by solving the optimization problem involved in the definition of the support function. Refer theorem 20.0.4 in `https://www.iith.ac.in/~saketha/teaching/cs5580notes.pdf`.

More importantly, the table shows that FW can be successfully applied when feasibility set is an l_p ball for $p \in (1, \infty), p \neq 2$; whereas there are no known polynomial time algorithms for projection onto the same (hence projected gradient descent cannot be applied). We hence-forth, in this section, assume that a linear minimization oracle (LMO) that efficiently solves (5.1) is available.

As long as the gradient oracle and LMO require comparable computational effort, again it is enough to analyze the convergence rate of FW. We simply followed theorem 3.8 in Bubeck [2015] for the convergence rate. This rate matches that of projected gradient in the smooth case.

In the special case when the feasibility set is a polytope, FW exposes its most interesting feature: the iterates are always sparse combinations of the extreme points of this polytope. Infact, the k^{th} iterate will be a combination of atmost $k+1$ extreme points (assuming the initial point is an extreme point). In other words, FW convergence rate can also be described in terms of sparsity: An FW iterate that is expressible using k extreme points is atleast $O(1/k)$-accurate. Lemmas 3,4 in Jaggi [2013] show that this sparsity-accuracy tradeoff is optimal[3] for FW! We then followed the case-study in pages 274-275 in Bubeck [2015] that showcases the advantage of this sparsity trade-off.

[3]Even Nesterov's accelerated/momentum methods may be sub-optimal in terms of this trade-off.

Chapter 6

Mirror Descent

Continuing our pursuit from the previous section, we seek an algorithm in the non-smooth case[1] when projection onto the feasibility set is costly. Mirror descent, detailed below, is one alternative that might work.

The key idea in Mirror descent is to choose the regularizer based on the feasibility set such that the iterate computation cost becomes comparable (or negligible when compared) to that of computing the gradient. For example, if the feasibility set is a simplex, then instead of using Euclidean distance to regularize, we can employ the KL-divergence[2] leading to:

(6.1)

$$x^{(k+1)} = \arg\min_{x \in \Delta_n} \frac{1}{\eta} KL\left(x \| x^{(k)}\right) + f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\left(x - x^{(k)}\right), \ \forall \ k = 0, 1, 2, \ldots$$

Interestingly, the above admits a closed-form solution: $x_i^{(k+1)} = \frac{x_i^{(k)} e^{-\eta \frac{\partial f\left(x^{(k)}\right)}{\partial x_i}}}{\sum_{j=1}^{n} x_j^{(k)} e^{-\eta \frac{\partial f\left(x^{(k)}\right)}{\partial x_j}}}$, which can be computed in linear time (efficiently). This is also known as exponentiated gradient update formula. Mirror descent generalizes this idea (of exponentiated gradient descent) to generic Bregman-divergence based regularizers.

Let ϕ be any σ-strongly-convex function defined over the feasibility set. Then, Bregman divergence is defined as: $D_\phi(x, y) \equiv \phi(x) - \phi(y) - \nabla\phi(y)^\top (x - y)$. Since $D_\phi(x, y) \geq \frac{1}{2}\sigma^2 \|x - y\|^2$, it can be easily shown that $D_\phi(x, y) \geq 0 \ \forall x, y \in dom(\phi)$ and $D_\phi(x, y) = 0 \iff x = y$. More interestingly, the correspondingly defined Bregman projection: $\Pi_{\mathcal{F}}^{\phi}(x) \equiv \arg\min_{y \in \mathcal{F}} D_\phi(y, x)$, satisfies some kind of

[1] Recall that FW works nicely in the smooth case.

[2] Recall that KL-divergence is more natural way of measuring differences between distributions than Euclidean.

triangle inequality as formalized in Lemma 4.1 in Bubeck [2015]. Mirror descent generalizes exponentiated gradient descent to generic Bregman-divergence based regularizers:
(6.2)

$$x^{(k+1)} = \arg\min_{x \in \mathcal{F}} \frac{1}{\eta} D_\phi \left(x, x^{(k)} \right) + f \left(x^{(k)} \right) + \nabla f \left(x^{(k)} \right) \left(x - x^{(k)} \right), \ \forall \ k = 0, 1, 2, \ldots$$

There are some known standard set-ups where (6.2) can be solved efficiently. These are summarized in section 4.3 in Bubeck [2015].

After giving a primal equivalent of this using conjugate (plays role of inverse mirror map), we then followed the derivation in section 4.1, equations (4.4-4.5) in Bubeck [2015] to show that the primal equivalent of mirror descent is given by equations (4.2-4.3) in Bubeck [2015]. This equivalence showcases the other important motivation for mirror descent, which is solving programs in non-Euclidean normed spaces. We also repeated the converge analysis in theorem 4.2 in Bubeck [2015][3].

[3]There seems to be a typo in the statement of this theorem. It has to be read that L is Lipschitz under $\|\cdot\|_*$ (dual norm) and NOT under $\|\cdot\|$ (original norm of the space)

Chapter 7

Proximal Gradient Descent

Here we consider programs that are Tikhonov regularized versions of those in the previous three sections. More specifically, we consider the following kind of programs:

$$\min_{x \in \mathbb{R}^n} f(x) + g(x), \tag{7.1}$$

where f, g are convex; however f is smooth and g is non-smooth. Classical analysis shows that gradient descent can only guarantee $O(1/\sqrt{k})$ convergence rate. However, proximal gradient is a slight modification that guarantees $O(1/k)$, which can be further (Nesterov) accelerated.

The key idea is to linearize only the smooth component (f) and keep g as it is in the per-step problem. More specifically:

$$x^{(k+1)} \equiv \arg\min_{x \in \mathbb{R}^n} \frac{1}{2\eta}\|x - x^{(k)}\|^2 + f\left(x^{(k)}\right) + \nabla f\left(x^{(k)}\right)\left(x - x^{(k)}\right) + g(x),\ \forall\ k = 0, 1, 2, \ldots, \tag{7.2}$$

which is equivalent to $x^{(k+1)} = \text{Prox}_{\eta g}\left(x^{(k)} - \eta \nabla f\left(x^{(k)}\right)\right)$. Here, Prox[1] is a generalization for notion of projection/conjugate, and is defined as:

$$\text{Prox}_g(x) \equiv \arg\min_{y \in \mathbb{R}^n} g(y) + \frac{1}{2}\|x - y\|^2. \tag{7.3}$$

The above algorithm is called as Proximal gradient[2]. We then looked at some special cases where prox can be efficiently computed. These appear in `https://people.eecs.berkeley.edu/~elghaoui/Teaching/EE227A/lecture18.pdf`. Finally, we followed the converge as summarized in the same pdf. ISTA/FISTA are popular for solving the lasso problem in ML.

[1] Prox is also a special of well-studied notion of c-conjugate.

[2] OR ISTA as in section 5.1 in Bubeck [2015], which can be further accelerated leading to FISTA

Chapter 8

Stochastic Gradient Descent

We mainly followed sections 6,6.1 in Bubeck [2015]. Interestingly, this shows that though gradient is not computed exactly, as long as it is unbiased, descent remains optimal for non-smooth problems (in expected sense). However, Tsybakov [2003] showed that unbiased estimates may adversely affect convergence for smooth problems. In general, SGD rate remains $O(1/\sqrt{k})$ even for smooth programs. This can be corrected by employing a variance reduced estimate as detailed in the next lecture.

Chapter 9

Stochastic Variance Reduced Gradient

We followed section 6.3 inBubeck [2015].

Chapter 10

Co-ordinate descent Methods

Bibliography

Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3–4):231–357, 2015.

Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–427–I–435. JMLR.org, 2013.

Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014. ISBN 1461346916.

Alexandre B. Tsybakov. Optimal rates of aggregation. In *16th Annual Conference on Computational Learning Theory*, pages 303–313, 2003.