Deterministic and Stochastic Optimization

Krishna Balasubramanian

April 20, 2020

1 Basics of Convexity

Convex sets and convex functions form the basis of optimization which is a very important aspect of computational statistics. It is important because it leads to **computationally efficient** ways of solving a wide class of optimization problems and because it may be used to derive powerful theoretical results.

Definition 1 (Convex Sets)**.** A set $\boldsymbol{\Theta} \subset \mathbb{R}^d$ is called a **convex set** if every line segment between two points in $\boldsymbol{\Theta}$ is in $\boldsymbol{\Theta}$ i.e. $\forall\, \mathbf{a}, \mathbf{b} \in \boldsymbol{\Theta}$, $\forall c \in [0,1]$, $c\,\mathbf{a} + (1-c)\,\mathbf{b} \in \boldsymbol{\Theta}$. The **convex hull** of any set $\boldsymbol{\Theta}$ (that is **not** necessarily convex to start with) is the set of all convex combinations of points in $\boldsymbol{\Theta}$. That is, $\text{conv}(\boldsymbol{\Theta}) = \{\sum c_i \mathbf{a}_i : \mathbf{a}_1, \ldots, \mathbf{a}_k \in \boldsymbol{\Theta},\, c_i \geq 0, \forall i,\ \text{and}\ \sum c_i = 1\}$. A set $\boldsymbol{\Theta}$ is a cone if for every $\mathbf{a} \in \boldsymbol{\Theta}$ and $c \geq 0$, $c\mathbf{a} \in \boldsymbol{\Theta}$. If it is also convex it is called a convex cone. An example of a convex cone that is relevant to us is the set of symmetric positive semidefinite matrices.

Listed below are some operations that preserve convexity of sets

- Intersection: An arbitrary intersection of convex sets is a convex set. [Union of convex sets are not convex in general]
- Scaling: If $\boldsymbol{\Theta}$ is convex, $c\,\boldsymbol{\Theta} = \{c\,\mathbf{a} : \mathbf{a} \in \boldsymbol{\Theta}\}$ is convex.
- Translation: If $\boldsymbol{\Theta}$ is convex, $\boldsymbol{\Theta} + \mathbf{b} = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in \boldsymbol{\Theta}\}$ is convex.

Definition 2 (Convex functions)**.** Let $\boldsymbol{\Theta} \subset \mathbb{R}^d$ be a convex set. A function $f : \boldsymbol{\Theta} \to \mathbb{R}$ is convex if

$$\forall c \in [0,1], \qquad \forall \mathbf{a}, \mathbf{b} \in \boldsymbol{\Theta}, \qquad f(c\,\mathbf{a} + (1-c)\,\mathbf{b}) \leq c\,f(\mathbf{a}) + (1-c)\,f(\mathbf{b}).$$

If we have strict inequality above we say that f is strictly convex. If $-f$ is convex, we say that f is concave.

Note: We also have an equivalent definition of convexity, which is easier to check often times in practice. A function $f(\boldsymbol{\theta})$ is convex if and only if the Hessian of the function $\nabla^2 f(\boldsymbol{\theta})$ is positive semidefinite for all points $\boldsymbol{\theta}$.

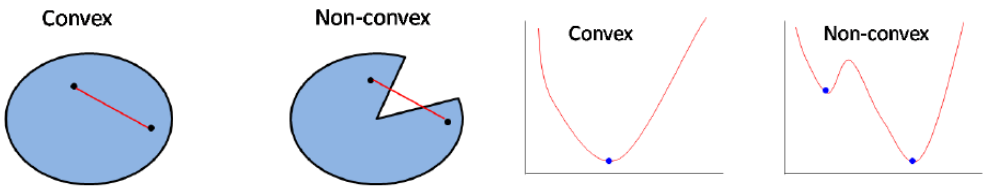

Figure 1: Left: Convex set. Right: Convex function

Definition 3 (Smooth and Strongly-Convex functions)**.** If f is twice differentiable on a convex domain $\boldsymbol{\Theta}$, then it is said to be μ-strongly-convex and L-smooth if $\nabla^2 f(\boldsymbol{\theta})$ is positive definite matrix and $0 < \mu = \lambda_{\min}\left(\nabla^2 f(\boldsymbol{\theta})\right) \leq \lambda_{\max}\left(\nabla^2 f(\boldsymbol{\theta})\right) = L < \infty$ for all points $\boldsymbol{\theta} \in \boldsymbol{\Theta}$.

Strong convexity is a condition representing how curved the function is. Note that strong-convexity says the Hessian matrix is positive definite (as opposed to positive semidefinite for general convex functions). Furthermore, one can show that if $\lambda_{\max}\left(\nabla^2 f(\boldsymbol{\theta})\right) = \mathrm{L}$, then $\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\|_2 \leq L\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2$. Hence smoothness here refers to how fast the gradient can change between nearby points.

For some examples of convex and strongly convex functions, take a look here.

2 Gradient Descent for Unconstrained Optimization

We now study the most basic algorithm for optimization. The algorithm is called as **gradient descent** as it utilizes the gradient of the function to minimize the function. To fix notations, consider minimizing

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} f(\boldsymbol{\theta}) \tag{1}$$

In the context of optimization, the function $f(\boldsymbol{\theta})$ is called as the objective function and the set $\boldsymbol{\Theta}$ is called as the constraint set. Below, we assume that $\boldsymbol{\Theta} = \mathbb{R}^d$, in which case the problem is called as **unconstrained optimization problem**.

A natural idea to find the optimal value of the function $f(\boldsymbol{\theta})$ above to start from a random point and iteratively update the point so that the function value is decreased in each iteration. That is, we start with $\boldsymbol{\theta}^{(0)}$ and produce a sequence $\{\boldsymbol{\theta}^{(t)}\}$ such that $f(\boldsymbol{\theta}^{(t+1)}) < f(\boldsymbol{\theta}^{(t)})$. Let us fix our updates to be of the form

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \eta_t \mathbf{d}^{(t)}$$

where η_t is a positive scalar called as **stepsize**. Here, $\mathbf{d}$ is called as the **descent direction**

Algorithm 1 Gradient Descent Algorithm

Input: Initial vector $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^d$
do
$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)})$
while $\|\nabla f(\boldsymbol{\theta}^{(t)})\| \geq \tau$
return $\boldsymbol{\theta}^{(t)}$

at a point $\boldsymbol{\theta}_1$ if the directional derivative is negative. That is,

$$f'(\boldsymbol{\theta}_1; \mathbf{d}) \stackrel{\text{def}}{=} \lim_{\tau \to 0} \frac{f(\boldsymbol{\theta}_1 + \tau \mathbf{d}) - f(\boldsymbol{\theta}_1)}{\tau} = \nabla f(\boldsymbol{\theta}_1)^\top \mathbf{d} < 0.$$

So now, among all such directions $\mathbf{d}$, we can find the direction that is the direction of **steepest descent** or most descent. That is, we formulate that problem as below. Among all vectors $\mathbf{d}$ for which, $\nabla f(\boldsymbol{\theta}_1)^\top \mathbf{d} < 0$, find the vector that maximizes $\nabla f(\boldsymbol{\theta}_1)^\top \mathbf{d}$. Now, without the negativity constraint, the direction that would maximize the inner product $\nabla f(\boldsymbol{\theta}_1)^\top \mathbf{d}$ is $\mathbf{d} = \nabla f(\boldsymbol{\theta}_1)^\top$. Hence with the negative constraint, the negative of the gradient $\nabla f(\boldsymbol{\theta}_1)^\top$ does the required job. This argument leads to the following update steps for the **gradient descent algorithm**:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)}) \tag{2}$$

The algorithm is formally provided in Algorithm 1. The algorithm has a automatic termination criterion which is based on the fact that gradient at the optimal point must be zero (or close to zero). The τ in the algorithm a tolerance level and is chosen to be as close to zero as we require (this different from the ϵ as they are measuring two different things).

2.1 Quadratic Minimization

To understand the performance of gradient descent algorithm, let us study minimizing convex quadratic problems of the form

$$f(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})^\top \mathbf{A} (\boldsymbol{\theta} - \bar{\boldsymbol{\theta}}) \tag{3}$$

Note that the gradient of $\nabla f(\boldsymbol{\theta}) = \mathbf{A}(\boldsymbol{\theta} - \bar{\boldsymbol{\theta}})$ and the Hessian is $\nabla^2 f(\boldsymbol{\theta}) = \mathbf{A}$. Furthermore, we see easily the minimum value of the function $f(\boldsymbol{\theta})$ is $\bar{\boldsymbol{\theta}}$, that is,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} f(\boldsymbol{\theta}) = \bar{\boldsymbol{\theta}}.$$

So if the matrix $\mathbf{A}$ is positive definite, with bounded eigenvalues, i.e, $0 < \lambda_{\min}(\mathbf{A}) \leq \lambda_{\max}(\mathbf{A}) < \infty$, then the function $f(\boldsymbol{\theta})$ is smooth and strongly-convex. Hence, quadratic functions serve as a nice testbed for studying the performance of gradient descent on

μ-strongly-convex and L-smooth convex optimization problems. Now lets calculate the **rate of convergence** of the gradient descent algorithm for finding $\boldsymbol{\theta}^*$.

Theorem 2.1. *Let $\boldsymbol{\theta}^{(0)}$ be the initial point and let us run the iterations defined in Equation 2 for minimizing the function in Equation 3. Furthermore, assume the matrix $\mathbf{A}$ is positive definite, with bounded eigenvalues, i.e, $0 < \lambda_{\min}(\mathbf{A}) \leq \lambda_{\max}(\mathbf{A}) < \infty$. If*

$$\eta_t = \eta = \frac{2}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})},$$

then

$$\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \leq \left(\frac{\lambda_{\max}(\mathbf{A}) - \lambda_{\min}(\mathbf{A})}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})}\right)^t \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2.$$

A word about proof: The proof of analyzing iterative algorithms, typically involves two main steps. **Step 1**: Find the relation between successive iterations in terms of some norm, i.e., relation between $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2$ and $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2$, for any t. **Step 2**: Apply recursion to get relation between $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2$ and $\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2$.

Proof. From the gradient descent iterations, with constant step-size $\eta_t = \eta$, we have

$$\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^* = \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* - \eta \nabla f(\boldsymbol{\theta}^{(t)}) = (\mathbf{I} - \eta \mathbf{A})(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)$$

Fact 2.1.1. For a vector $\mathbf{b}$ and matrix $\mathbf{A}$, we have $\|\mathbf{A}\mathbf{b}\|_2 \leq \|\mathbf{A}\|\|\mathbf{b}\|_2$, where $\|\mathbf{A}\|$ is the operator norm of the matrix.

Hence, we have by Fact 2.1.1,

$$\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \leq \|\mathbf{I} - \eta \mathbf{A}\|\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2.$$

Now note that the eigenvalues of the matrix $\mathbf{I} - \eta \mathbf{A}$ are of the form

$$(1 - \eta\lambda_{\max}(\mathbf{A})), \ldots, (1 - \eta\lambda_{\min}(\mathbf{A})).$$

Hence, $\|\mathbf{I} - \eta \mathbf{A}\|$, which is the maximum eigenvalue of $\mathbf{I} - \eta \mathbf{A}$ is given by

$$\|\mathbf{I} - \eta \mathbf{A}\| = \max\{|(1 - \eta\lambda_{\max}(\mathbf{A}))|, |(1 - \eta\lambda_{\min}(\mathbf{A}))|\}$$

and depends on η. Now the choice

$$\eta_t = \eta = \frac{2}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})}$$

minimizes the norm $\|\mathbf{I} - \eta \mathbf{A}\|$ which becomes,

$$\|\mathbf{I} - \eta \mathbf{A}\| = \left(\frac{\lambda_{\max}(\mathbf{A}) - \lambda_{\min}(\mathbf{A})}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})}\right)$$

To recap, we have shown that

$$\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \leq \left(\frac{\lambda_{\max}(\mathbf{A}) - \lambda_{\min}(\mathbf{A})}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})}\right) \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2$$

Applying this bound, recursively yields the desired result. □

2.1.1 Things to note

1. Note that we can write

$$\left(\frac{\lambda_{\max}(\mathbf{A}) - \lambda_{\min}(\mathbf{A})}{\lambda_{\max}(\mathbf{A}) + \lambda_{\min}(\mathbf{A})}\right) = \left(\frac{\frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} - 1}{\frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})} + 1}\right) = \left(\frac{\kappa - 1}{\kappa + 1}\right)$$

   where $\kappa = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}$ is called as the **condition number** of the problem and plays a key role in determining the rate of convergence. Indeed, as $0 < \left(\frac{\kappa-1}{\kappa+1}\right) < 1$ the higher the value of κ the slower the convergence is.

2. If the point $\boldsymbol{\theta}^{(t)}$ satisfies the condition $\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \leq \epsilon$, we say that the point $\boldsymbol{\theta}^{(t)}$ is a ϵ-optimal solution for the problem. The number of iterations t required to obtain an ϵ-optimal solution is called as the iteration complexity of the problem. In our case, based on Theorem 2.1, to get an ϵ-optimal solution, we want

$$\left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2 = \epsilon.$$

   But note that we have $0 < \left(\frac{\kappa-1}{\kappa+1}\right) < 1$ which implies that

$$t \approx \left(\frac{\log\left(\frac{\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2}{\epsilon}\right)}{\log\left(\frac{\kappa-1}{\kappa+1}\right)}\right) = \mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$$

3. The rate of convergence of gradient descent for minimizing any μ-strongly-convex and L-smooth function will resemble the rates in the Theorem 2.1, with μ replacing $\lambda_{\min}$ and L replacing $\lambda_{\max}$.

4. The choice of η selected requires knowledge of eigenvalues of $\mathbf{A}$ which might take some additional computation to calculate. An alternative is to choose the step-size as

$$\eta_t = \arg\min_{\eta \geq 0} f(\boldsymbol{\theta}^{(t)} - \eta \nabla f(\boldsymbol{\theta}^{(t)})).$$

   This called as **gradient descent with line search** in the literature.

5. One can ask what happens when $\mu = 0$. In this case, we are minimizing quadratic functions that are smooth but not strongly convex. One can show that gradient descent will still work in this case, but the rate of convergence will be slower. Specifically, we will have $t = \mathcal{O}(1/\epsilon)$ as opposed to $t = \mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$ for strongly-convex problems.

6. In the above mentioned rates, the dimensionality d never appears explicitly. In this sense, the results are "dimension-free". But in practice the constants μ and L will invariably depend on the dimensionality! So, when we say rate of convergences in convex optimization in dimension-free one should be careful about the hidden dimensionality in μ and L. Indeed, you will see when you implement your algorithm that the rates are slower for high-dimensional problems.

3 Gradient Descent for Constrained Optimization

We now consider the case in Equation 1, when the set $\boldsymbol{\Theta}$ is a **convex** subset of $\mathbb{R}^d$ and the objective function $f(\boldsymbol{\theta})$ is convex. There are two algorithms for doing constrained optimization that we will look at. The choice of the algorithm is determined by the geometry of the constrained set $\boldsymbol{\Theta}$.

3.1 Projected Gradient Descent

This algorithm is preferred when the constraint set $\boldsymbol{\Theta}$ is such that it is easy to project onto to the set. Specifically, the Euclidean projection onto the set $\boldsymbol{\Theta}$ of a vector $\mathbf{a}$ is denoted as $P_{\boldsymbol{\Theta}}(\mathbf{a})$ and is defined as follows:

$$P_{\boldsymbol{\Theta}}(\mathbf{a}) = \arg\min_{\mathbf{c} \in \boldsymbol{\Theta}} \|\mathbf{a} - \mathbf{c}\|_2.$$

It finds the vector in the set $\boldsymbol{\Theta}$ which is closest to the vector $\mathbf{a}$. Depending on the geometry of the set $\boldsymbol{\Theta}$ computing the $P_{\boldsymbol{\Theta}}(\mathbf{a})$ might be easier or harder:

1. One of the easiest cases is when the set $\boldsymbol{\Theta} = \{\boldsymbol{\theta} \in \mathbb{R}^d : \|\boldsymbol{\theta}\|_2 \leq 1\}$. This situation arises when we are dealing with Ridge regression. In this case, if $\mathbf{a} \in \boldsymbol{\Theta}$, then we just leave it as such and we already have $\|\mathbf{a}\|_2 \leq 1$. If $\mathbf{a} \notin \boldsymbol{\Theta}$, then it means $\|\mathbf{a}\|_2 > 1$. So the closest vector in $\boldsymbol{\Theta}$ to $\mathbf{a}$ is $\mathbf{a}/\|\mathbf{a}\|_2$. Putting both cases together, we have

$$P_{\boldsymbol{\Theta}}(\mathbf{a}) = \frac{\mathbf{a}}{\max\{1, \|\mathbf{a}\|_2\}}$$

2. Next, consider the case of $\boldsymbol{\Theta} = \{\boldsymbol{\theta} \in \mathbb{R}^d : \|\boldsymbol{\theta}\|_1 \leq s\}$, which arises when dealing with LASSO. Now, things becomes a bit complicated but still it is possible to project efficiently on to this set – If you are curious about this case, you can refer to [DSSSC08].

Coming back to constrained optimization, a simple modification of gradient descent will work for constrained optimization. Specifically, in each iteration of the gradient descent, the

Algorithm 2 Projected Gradient Descent Algorithm

Input: Initial vector $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^d$
do
$\quad \boldsymbol{\theta}^{(t+1)} = P_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)})\right)$
while $\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2 \geq \tau$
return $\boldsymbol{\theta}^{(t)}$

vector $\boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)})$ may or may not lie inside the set $\boldsymbol{\Theta}$. So, we modify the iteration by projection operation as follows:

$$\boldsymbol{\theta}^{(t+1)} = P_{\boldsymbol{\Theta}}\left(\boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)})\right) \tag{4}$$

The algorithm is formally provided in Algorithm 2. There is a subtle technicality with the termination criterion in Algorithm 2. For the termination criterion to work, the minimizer **must** be inside the set $\boldsymbol{\Theta}$ and not on the boundary of the set $\boldsymbol{\Theta}$. We will not go too much into this subtle issue.

Theorem 3.1. *Let $\boldsymbol{\theta}^{(0)}$ be the initial point and let us run the iterations defined in Equation 4 for minimizing the function in Equation 1. Furthermore, assume the function is μ-strongly convex and L-smooth. If*

$$\eta_t = \eta = \frac{2}{\mu + L},$$

then

$$\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2.$$

where $\kappa = L/\mu$ is the condition number.

Proof Sketch. The proof of the above theorem is very similar to that of Theorem 2.1 except for one additional fact (Fact 3.1.1) about the Projection operation $P_{\boldsymbol{\Theta}}(\cdot)$:

Fact 3.1.1. If $\boldsymbol{\Theta}$ is a convex set, then for two vectors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, the projection operation is **non-expansive**, i.e.,

$$\|P_{\boldsymbol{\Theta}}(\boldsymbol{\theta}_1) - P_{\boldsymbol{\Theta}}(\boldsymbol{\theta}_2)\|_2 \leq \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2$$

Now note that in Theorem 2.1, we implicitly proved that

$$\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 = \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* - \eta \nabla f(\boldsymbol{\theta}^{(t)})\|_2 \leq \frac{\kappa - 1}{\kappa + 1} \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2$$

Now, consider the projected gradient iterations

$$\begin{aligned}\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 &= \|P_{\boldsymbol{\Theta}}(\boldsymbol{\theta}^{(t)} - \eta\nabla f(\boldsymbol{\theta}^{(t)})) - P_{\boldsymbol{\Theta}}(\boldsymbol{\theta}^*)\|_2 \\ &\leq \|\boldsymbol{\theta}^{(t)} - \eta\nabla f(\boldsymbol{\theta}^{(t)}) - \boldsymbol{\theta}^*\|_2 \\ &\leq \frac{\kappa - 1}{\kappa + 1}\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2\end{aligned}$$

Here, the first inequality follows by Fact 3.1.1. Now, applying the above inequality recursively, we get the result of the theorem. □

Remark 3.1.1. Similar to gradient descent, the above projected gradient algorithm takes $t = \mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$ for get an ϵ-accurate solution for problem 1 when f is strongly-convex and smooth (based on the above theorem). For the case of convex and smooth problems, it takes $t = \mathcal{O}(\frac{1}{\epsilon})$.

3.2 Frank-Wolfe or Conditional Gradient Descent

Recall that the projected gradient descent algorithm is preferred when projection to the set $\boldsymbol{\Theta}$ could be computed efficiently. In some cases, projection might be a costly operation to compute, but one can easily minimze a linear function over the set $\boldsymbol{\Theta}$. Recall that we are producing a sequence of iterates as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \eta_t \mathbf{d}^{(t)}$$

where $\mathbf{d}^{(t)}$ is descent direction, such that $f(\boldsymbol{\theta}^{(t+1)}) \leq f(\boldsymbol{\theta}^{(t)})$. For constrained optimization problems, we additionally require

$$\boldsymbol{\theta}^{(t)} + \eta_t \mathbf{d}^{(t)} \in \boldsymbol{\Theta}.$$

So the main question is: **Can we guarantee that the iterates stay in the set Θ while decreasing the objective value in each iteration ?**

It turns out one can do the following constrained linear optimization step to get a descent direction:

$$\mathbf{d}^{(t)} = \arg\min_{\mathbf{c} \in \boldsymbol{\Theta}} \ \mathbf{c}^\top \nabla f(\boldsymbol{\theta}^{(t)}) \tag{5}$$

This is a linear optimization over a convex set, which might be simpler to compute in several cases. The motivation for the above step is the same as gradient descent: We are finding the direction of steepest descent but now along with the constraint enforced by the optimization problem given to us! This leads to the Frank-Wolfe or Conditional Gradient algorithm outlined in Algorithm 3. One caveat: Similar to the projected gradient algorithm, the termination criterion given in Algorithm 3 works only when the optimal point $\boldsymbol{\theta}^*$ is inside the constraint set and not on the boundary.

Algorithm 3 Conditional Gradient or Frank-Wolfe

Input: Initial vector $\boldsymbol{\theta}^{(0)} \in \boldsymbol{\Theta}$
do
Solve $\mathbf{d}^{(t)} = \arg\min_{\mathbf{c} \in \boldsymbol{\Theta}} \ \mathbf{c}^\top \nabla f(\boldsymbol{\theta}^{(t)})$
Update: $\boldsymbol{\theta}^{(t+1)} = (1 - \eta_t)\boldsymbol{\theta}^{(t)} + \eta_t \mathbf{d}^{(t)}$
while $\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2 \geq \tau$
return $\boldsymbol{\theta}^{(t)}$

For the Frank-Wolfe algorithm, we will also derive rate of convergence under the case of smooth but convex function. The rates will indicate that we require $t = \mathcal{O}(\frac{1}{\epsilon})$ iterations to achieve an ϵ-optimal solution. But analyzing Frank-Wolfe algorithm under strongly convex setting to obtain the faster rates similar to projected gradient descent and unconstrained gradient descent is more subtle. It turns out that only for some special constraint sets $\boldsymbol{\Theta}$, we will get the improved rates. Below, we state without proof a result for the performance of Frank-Wolfe update steps for convex smooth constrained optimization problems. If you are curious about the proof, refer to [Jag13].

Theorem 3.2. *Let f be a convex function that is L-smooth. Set*

$$\eta_t = \frac{2}{t+2},$$

then one has

$$f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \frac{2Ld_{\boldsymbol{\Theta}}}{t+2}$$

where $d_{\boldsymbol{\Theta}}$ is the squared diameter of the set $\boldsymbol{\Theta}$ defined as $d_{\boldsymbol{\Theta}} = \max_{\mathbf{r},\mathbf{s} \in \boldsymbol{\Theta}} \|\mathbf{r} - \mathbf{s}\|_2^2$

Remark 3.2.1. We immediately see that to get $f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \epsilon$, we require $t = \mathcal{O}(\frac{1}{\epsilon})$ as long as the diameter of the set $\boldsymbol{\Theta}$ is finite. This matches the complexity of projected gradient descent and unconstrained gradient descent (for convex smooth problems). One main difference is the here the error is measured in terms of function values. But in the previous cases, we measured in terms of optimal vectors itself. But this distinction is minor for the purpose of this course.

3.3 Power Method as Frank-Wolfe

Although our focus here is on convex optimization, we briefly deviate to point out a connection between the power method for computing the eigenvector of a positive semidefinite matrix and the Frank-Wolfe algorithm. Specifically, recall that finding the top-eigenvector of a positive semidefinite matrix $\mathbf{A}$ could be formulated as the following problem:

$$\arg\max_{\{\mathbf{c}:\|\mathbf{c}\|_2 \leq 1\}} \ \mathbf{c}^\top \mathbf{A}\mathbf{c}$$

Note: Eigenvectors, by definition, have to be unit vectors. So, ideally we should do the maximization over the set $\{\mathbf{c} : \|\mathbf{c}\|_2 = 1\}$. But because we are dealing with a maximization problem, we can take the convex hull due to which we get the set $\{\mathbf{c} : \|\mathbf{c}\|_2 \leq 1\}$. This is possible because if the norm of the vector is not equal to one, it is not going to maximize the quadratic form. This still does not make the above problem a *convex optimization* problem we are maximizing a convex function over convex set! (Only when we minimize a convex function over a convex set, we get a convex optimization problem).
The above problem, though, is equivalent to the following problem:

$$\arg\min_{\{\mathbf{c}:\|\mathbf{c}\|_2\leq 1\}} \quad -\mathbf{c}^\top \mathbf{A}\mathbf{c}$$

In this form, we are minimizing a concave function over a convex set. So still this is not a convex optimization problem! But one can still go ahead and apply Frank-Wolfe update step for this problem! Doing so, we get

$$\begin{aligned}\mathbf{d}^{(t)} &= \arg\min_{\{\mathbf{c}:\|\mathbf{c}\|_2\leq 1\}} \mathbf{c}^\top \nabla f(\boldsymbol{\theta}^{(t)}) \\ &= -\frac{\nabla f(\boldsymbol{\theta}^{(t)})}{\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2}\end{aligned}$$

Our objective function is $-\mathbf{c}^\top \mathbf{A}\mathbf{c}$ and hence the gradient of this function at $\boldsymbol{\theta}^{(t)}$ is $-\mathbf{A}\boldsymbol{\theta}^{(t)}$. Substituting this in the above equation, we get

$$\mathbf{d}^{(t)} = \frac{\mathbf{A}\boldsymbol{\theta}^{(t)}}{\|\mathbf{A}\boldsymbol{\theta}^{(t)}\|_2}$$

This implies that the Frank-Wolfe update step is given by

$$\boldsymbol{\theta}^{(t+1)} = (1-\eta_t)\boldsymbol{\theta}^{(t)} + \eta_t \frac{\mathbf{A}\boldsymbol{\theta}^{(t)}}{\|\mathbf{A}\boldsymbol{\theta}^{(t)}\|_2}$$

When we set $\eta_t = 1$, which is what we actually get when we do the **line search method** for setting η, we see that the Power method is equivalent to Frank-Wolfe method!

4 Acceleration

We now study faster methods for convex optimization which are collectively called as accelerated methods for convex optimization. The style of the algorithms are similar to the gradient descent algorithm, but with crucial modifications.

4.1 Strongly Convex Setting

Coming back to unconstrained optimization for μ-strongly convex and L-smooth functions, recall that we have the following result for gradient descent:

$$\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2.$$

where $\kappa = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}$. But gradient descent algorithm focuses on improving per-cost iteration rather than overall iteration complexity. Furthermore, the iterates of gradient descent may be zig-zag and may experience abrupt changes. One can ask if the above bound could be improved. It turns out that we can improve the dependence on κ to $\sqrt{\kappa}$ using the so-called Heavy-ball method:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla f(\boldsymbol{\theta}^{(t)}) + \gamma_t \left(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\right) \tag{6}$$

The term $\left(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^{(t-1)}\right)$ is called as the momentum term as it helps the iterates not go zig-zag (especially when γ_t is chosen appropriately). One can write the iterates of the heavy-ball methods through the following matrix form.

$$\begin{bmatrix} \boldsymbol{\theta}^{(t+1)} \\ \boldsymbol{\theta}^{(t)} \end{bmatrix} = \begin{bmatrix} (1+\gamma_t)\mathbf{I} & -\gamma_t \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}^{(t)} \\ \boldsymbol{\theta}^{(t-1)} \end{bmatrix} - \begin{bmatrix} \eta_t \nabla f(\boldsymbol{\theta}^{(t)}) \\ \mathbf{0} \end{bmatrix}$$

which could be equivalently written as

$$\begin{bmatrix} \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^* \\ \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \end{bmatrix} = \begin{bmatrix} (1+\gamma_t)\mathbf{I} & -\gamma_t \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \\ \boldsymbol{\theta}^{(t-1)} - \boldsymbol{\theta}^* \end{bmatrix} - \begin{bmatrix} \eta_t \nabla f(\boldsymbol{\theta}^{(t)}) \\ \mathbf{0} \end{bmatrix}$$
$$\begin{bmatrix} (1+\gamma_t)\mathbf{I} - \eta_t \int_0^1 \nabla^2 f(\boldsymbol{\theta}_\tau)\, d\tau & -\gamma_t \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \\ \boldsymbol{\theta}^{(t-1)} - \boldsymbol{\theta}^* \end{bmatrix}$$

where $\boldsymbol{\theta}_\tau = \boldsymbol{\theta}^{(t)} + \tau(\boldsymbol{\theta}^* - \boldsymbol{\theta}^{(t)})$ and the last equality follows from the fundamental theorem of calculus. The matrix

$$\mathbf{S}_t = \begin{bmatrix} (1+\gamma_t)\mathbf{I} - \eta_t \int_0^1 \nabla^2 f(\boldsymbol{\theta}_\tau)\, d\tau & -\gamma_t \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{bmatrix}$$

is called as the system matrix and controls the performance of the heavy-ball method. For the above heavy-ball method, we have the following result.

Theorem 4.1. *Let f be a μ-strongly convex L-smooth function. Let*

$$\eta_t = \eta = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \qquad \theta_t = \max\{|1 - \sqrt{\eta_t L}|, |1 - \sqrt{\eta_t \mu}|\}^2$$

Then, we have

$$\left\| \begin{bmatrix} \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^* \\ \boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^* \end{bmatrix} \right\|_2 \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^t \left\| \begin{bmatrix} \boldsymbol{\theta}^{(1)} - \boldsymbol{\theta}^* \\ \boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^* \end{bmatrix} \right\|_2$$

Rough Proof Sketch. The proof follows by calculating the eigenvalues of the matrix $\mathbf{S}_t$ and picking η_t and γ_t. Specifically, we can show that with the choices of η_t and γ_t above,

$$\|\mathbf{S}_t\| \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)$$

Then, we use fact 2.1.1 and recursion to complete the proof. □

4.2 Convex Setting

We saw in the last subsection that for the strongly-convex case, we have improvements on the conditional number only. One can ask what happends for the convex case (without strong convexity). It turns out that we can get an interesting improvement using an interesting algorithm called as Nesterov's accelerated gradient algorithm (**note: no descent**), proposed in 1983. In his paper, Nesterov outlined an algorithm and a result showing the improvement. No intuition was provided till literally a couple of years ago, when people **started to partially understand** how the algorithm actually worked! So, until recently, it remained as one of the mysteries of convex optimization (and continues to remain so, to some extent). The iterates works as follows:

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} - \eta_t \nabla f(\boldsymbol{\beta}^{(t)}) \\ \boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\theta}^{(t+1)} + \frac{t}{t+3} \left(\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)} \right) \end{aligned}$$

It turns out that this is not a descent algorithm as it does not satisfy $f(\boldsymbol{\theta}^{(t+1)}) \leq f(\boldsymbol{\theta}^{(t)})$. Furthermore, the number 3 in the second iteration seems crucial for convergence for some reasons! For this algorithm, Nesterov showed the following result.

Theorem 4.2. *Let f be a convex L-smooth function. Let*

$$\eta_t = \eta = \frac{1}{L}.$$

Then

$$f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \frac{2L \|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2^2}{(t+1)^2}$$

We immediately see that to get $f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \epsilon$, we require $t = \mathcal{O}(\frac{1}{\sqrt{\epsilon}})$ as opposed to $t = \mathcal{O}(\frac{1}{\epsilon})$ for gradient descent algorithm. Still this method does not match the iteration

complexity of strongly-convex functions! The proof the above algorithm is purely analytical with no intuition. Nesterov, when asked, mentioned he got it by trail and error!

5 Non-smooth Problems

In the previous section, we considered smooth-problems, where we characterized smoothness through the rate of change of gradient vector. For this to be well-defined, we needed the function to be differentiable everywhere. But in several problems, the objective function is not differentiable. For example,

- Robust least-squares: $\mathbf{b}^* = \arg\min_{\mathbf{b}\in\mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|_1$
- Lasso: $\mathbf{b}^* = \arg\min_{\mathbf{b}\in\mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2 + \lambda\|\mathbf{b}\|_1$

The above two problems are non-smooth due the presence of $\|\cdot\|_1$ norm.

5.1 Sub-gradient Methods

Sub-gradient methods are techniques that generalize the gradient descent type methods. Recall the definition of the convex function in Definition 2. It is called as a zeroth-order definition of convexity as we only use function evaluations. We also saw an equivalent way to define convex function is by using Hessian matrix (when it is positive semidefinite) which is a second-order condition for convexity. We now look at a first-order condition of convexity:

Fact 5.0.1. A differentiable function $f(\boldsymbol{\theta})$ is convex if and only if

$$f(\boldsymbol{\theta}_2) \geq f(\boldsymbol{\theta}_1) + \nabla f(\boldsymbol{\theta}_1)^\top (\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1)$$

Based on this condition, we define what is called as the **sub-gradient** below.

Definition 4. We say that a vector $\mathbf{g}(\boldsymbol{\theta}_1) \in \mathbb{R}^d$ is the sub-gradient at the point $\boldsymbol{\theta}_1$ if

$$f(\boldsymbol{\theta}_2) \geq f(\boldsymbol{\theta}_1) + \mathbf{g}(\boldsymbol{\theta}_1)^\top (\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1) \qquad \forall\ \boldsymbol{\theta}_2$$

Note that **any** vector that satisfies the above condition is called as the sub-gradient. Hence, in many cases, we have a **set** of sub-gradients!

Example 5.0.1. Let us look at an example when $d = 1$. Take $f(\theta) = |\theta|$. Obviously at $\theta = 0$, the function is not differentiable. Hence the smooth-gradient assumption made in the previous section does not hold when dealing with this function. We now calculate the sub-gradient of the function at all points $\theta \in \mathbb{R}$. For any $\theta < 0$ the sub-gradient consists of the set $g(\theta) = \{-1\}$. Similarly for any $\theta > 0$, the sub-gradient consists of the set

$g(\theta) = \{+1\}$. At $\theta = 0$, the sub-gradient is defined by the following inequality: $|\theta_2| \geq g(0)\theta_2$ for all $\theta_2 \in \mathbb{R}$. Hence, we have the set of sub-gradients to be $g(0) = [-1, 1]$. So, we have

$$g(\theta) = \begin{cases} \{-1\} & \text{if } \theta < 1 \\ [-1, 1] & \text{if } \theta = 0 \\ \{+1\} & \text{if } \theta > 1 \end{cases}$$

Exercise 5.0.1. Calculate the sub-gradient of the function $f(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$.

With this definition of sub-gradient, one can just run the gradient descent algorithm (or the projected gradient descent algorithm). One still cannot analyze the performance of the algorithm for the set of all non-smooth function (what does it mean for a function in general to be non-smooth!) We now define a notion of smoothness called as Lipschitz smoothness, i.e, the function satisfies

$$|f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}_2)| \leq L_f \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2, \qquad \forall \boldsymbol{\theta}_1, \boldsymbol{\theta}_2.$$

Theorem 5.1. *Consider optimizing $f(\boldsymbol{\theta})$ with the gradient descent algorithm.*

1. *When f is μ-strongly convex and L_f-Lipschitz continuous, if $\eta_t = \eta = \frac{2}{\mu(t+1)}$, we have*

$$f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \frac{2L_f^2 \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2}{\mu(t+1)}$$

2. *When f is convex and L_f-Lipschitz continuous, if η_t is picked satisfying the condition of the Polyak step-size, we have*

$$f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \frac{L_f \|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2}{\sqrt{t+1}}$$

   where Polyak step-size refers to

$$\eta_t = \frac{f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*)}{\|g(\boldsymbol{\theta}^{(t)})\|_2^2}$$

   and depends on the unknown $f(\boldsymbol{\theta}^*)$.

Note that based on the above theorem, we have the following iteration complexity.

1. For strongly convex and Lipschitz continuous convex functions, to get $f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \epsilon$, we require $t = \mathcal{O}\left(\frac{1}{\epsilon}\right)$ sub-gradient iterations

2. For convex and Lipschitz continuous convex functions, to get $f(\boldsymbol{\theta}^{(t)}) - f(\boldsymbol{\theta}^*) \leq \epsilon$, we require $t = \mathcal{O}\left(\frac{1}{\epsilon^2}\right)$ sub-gradient iterations.

Algorithm	Assumption	Iteration Complexity
Gradient descent	μ-SC and L-smooth	$\mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$
Gradient descent	C and L-smooth	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$
Projected gradient descent	μ-SC and L-smooth	$\mathcal{O}\left(\log\left(\frac{1}{\epsilon}\right)\right)$
Projected gradient descent	C and L-smooth	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$
Conditional gradient descent	μ-SC and L-smooth	Not completely understood
Conditional gradient descent	C and L-smooth	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$
Heavy-ball Method	μ-SC and L-smooth	Improvements in κ
Nesterov's accelerated gradient	C and L-smooth	$\mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$
Sub-gradient descent	μ-SC and L_f-Lipschitz	$\mathcal{O}\left(\frac{1}{\epsilon}\right)$
Sub-gradient descent	C and L_f-Lipschitz	$\mathcal{O}\left(\frac{1}{\epsilon^2}\right)$

Table 1: Iteration Complexity of Convex Optimization. SC stands for strongly-convex. C stands for (just) convex. L-Smooth stands for gradient smoothness. L_f-smooth stands for Lipschitz smooth functions. Projected and Conditional Gradient Descent methods are used for constrained problems. Iteration complexity refers to how many steps do we need to run the algorithm to get ϵ-close to the global minimum (in terms of the vector itself or function values).

6 Stochastic Optimization

In stochastic optimization, the function $f(\boldsymbol{\theta})$ is an expectation of a stochastic/random function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta}) = \mathsf{E}_{\boldsymbol{\xi}}\left[F(\boldsymbol{\theta};\boldsymbol{\xi})\right]. \tag{7}$$

Here the $\boldsymbol{\xi} \in \mathbb{R}^d$ is a random vector. One way to solve the above problem is the so-called Sample-Average Approximation, where one gets n i.i.d. samples $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$ to approximate the expectation with the sample-average:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f_n(\boldsymbol{\theta}; \{\boldsymbol{\xi}_i\}_{i=1}^n) = \frac{1}{n}\sum_{i=1}^n F(\boldsymbol{\theta};\boldsymbol{\xi}_i). \tag{8}$$

A standard example of the above framework is the population maximum likelihood estimator when $F(\boldsymbol{\theta};\boldsymbol{\xi}) = -\log p_{\boldsymbol{\theta}}(\boldsymbol{\xi})$. Here, $p_{\boldsymbol{\theta}}(\cdot)$ refers to the density of $\boldsymbol{\xi}$. In this case, the above problem is the population maximum likelihood estimator:

$$\boldsymbol{\theta}_{mle} = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \; -\mathsf{E}_{\boldsymbol{\xi}}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{\xi})\right]. \tag{9}$$

The sample maximum likelihood estimator is given by

$$\hat{\boldsymbol{\theta}}_{mle} = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f_n(\boldsymbol{\theta}; \{\boldsymbol{\xi}_i\}_{i=1}^n) = -\frac{1}{n}\sum_{i=1}^n \log p_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i) \tag{10}$$

Under regularity condition on the function $F(\boldsymbol{\theta};\boldsymbol{\xi})$, one can show that as $n \to \infty$, we have $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*\|_2 \to 0$ almost surely. Furthermore, one can show that the estimator or optimal value $\hat{\boldsymbol{\theta}}$ is asymptotically normal centered around $\boldsymbol{\theta}^*$.

But note that in SAA or sample MLE, we still need to provide an algorithm to solve the problem. We can think of $f_n(\boldsymbol{\theta}; \{\boldsymbol{\xi}_i\}_{i=1}^n)$ as a deterministic function (by conditioning on the randomness in $\boldsymbol{\xi}$) and apply the algorithms that we had from previous section. But often times, the assumption required by the algorithms from previous section are not satisfied. Furthermore, the consistency and asymptotic normality properties mentioned previously are not established for the algorithm but only for the solution $\hat{\boldsymbol{\theta}}$ (or $\hat{\boldsymbol{\theta}}_{mle}$).

6.1 Stochastic Gradient Descent

Stochastic gradient method is a technique for directly optimizing the stochastic optimization problem in 7. To understand how it works, we look at the gradient descent iteration of the

problem 7 (assuming integration and differentiation could be interghanced):

$$\begin{aligned}\boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - \eta_t \nabla_t f(\boldsymbol{\theta}^{(t)}) \\ &= \boldsymbol{\theta}^{(t)} - \eta_t \nabla_{\boldsymbol{\theta}} \mathsf{E}_{\boldsymbol{\xi}} \left[F(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}) \right] \\ &= \boldsymbol{\theta}^{(t)} - \eta_t \mathsf{E}_{\boldsymbol{\xi}} \left[\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}) \right]\end{aligned}$$

Of course to apply this method, we need to compute this expectation: $\mathsf{E}_{\boldsymbol{\xi}} \left[\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}) \right]$. Indeed, we have done nothing but to transfer the computation of the expectation from the function to the gradient. The idea in stochastic gradient descent method, is to approximate this expectation of the gradient with **just one sample** in each iteration. That is,

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \mathbf{g}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\xi}^{(t)}) \tag{11}$$

Here, $\mathbf{g}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\xi}^{(t)}) = \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)})$ is called as the stochastic gradient as it is a random vector. Furthermore, note that $\boldsymbol{\theta}^{(t+1)}$ is also a random vector as the gradients are stochastic. Notice that now we have switched the index for $\boldsymbol{\xi}$ from $\boldsymbol{\xi}_i$ to $\boldsymbol{\xi}^{(t)}$ to denote the fact that in each iteration, we use only one sample to approximate the gradient. Surprisingly, approximating the gradient in each step with just one sample works in practice. One can also use multiple samples $\boldsymbol{\xi}_i$ in each iteration to compute the stochastic gradient by averaging, i.e.,

$$\mathbf{g}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\xi}^{(t)}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}_i).$$

This version of stochastic gradient is called as mini-batch stochastic gradient descent. To analyze the performance of the stochastic gradient descent algorithm, we assume the following.

1. The function $F(\boldsymbol{\theta}, \boldsymbol{\xi})$ is μ-strongly convex and L-smooth in $\boldsymbol{\theta}$.
2. The stochastic gradient is unbiased: $\mathsf{E}_{\boldsymbol{\xi}}[\mathbf{g}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\xi}^{(t)})] = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}^{(t)})$ given $\boldsymbol{\xi}^{(t)}, \ldots, \boldsymbol{\xi}^{(t-1)}$.
3. The variance of the gradient vector is bounded: $\mathsf{E}_{\boldsymbol{\xi}}[\|\mathbf{g}(\boldsymbol{\theta}^{(t)}, \boldsymbol{\xi}^{(t)})\|_2^2] \leq \sigma_{\mathbf{g}}^2 + M_g \mathsf{E} \left[\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2^2 \right]$ for all $\boldsymbol{\theta}^{(t)}$.

Theorem 6.1. *Suppose we run the stochastic gradient descent algorithm (as in Equation 11) for solving the optimization problem in Equation 7. If*

$$\eta_t = \frac{c}{t+1}$$

for some $c > 0$, then we have

$$E \left[\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 \right] \leq \frac{c_0}{t+1}$$

where c_o is a numerical constant.

Proof.

$$\begin{aligned}\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2 &= \|\boldsymbol{\theta}^{(t)} - \eta_t \mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)}) - \boldsymbol{\theta}^*\|_2^2 \\ &= \|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2 + \eta_t^2 \|\mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)})\|_2^2 - 2\eta_t (\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)})\end{aligned}$$

The expectation of the gradient term could be calculated as

$$\begin{aligned}\mathsf{E}\left[\|\mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)})\|_2^2\right] &\leq \sigma_{\mathbf{g}}^2 + M_g \mathsf{E}\left[\|\nabla f(\boldsymbol{\theta}^{(t)})\|_2^2\right] \\ &\leq \sigma_{\mathbf{g}}^2 + M_g L^2 \mathsf{E}\left[\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\right] \end{aligned} \tag{12}$$

where in the last inequality, we used two facts: if F is smooth (in gradients) then f is also smooth and the gradient at $\boldsymbol{\theta}^*$ is zero vector.

The expectation of the inner product term is calculated as follows, by using law of total expectation and the fact that $\boldsymbol{\theta}^{(t)}$ is independent of $\boldsymbol{\xi}^{(t)}$.

$$\begin{aligned}\mathsf{E}\left[(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)})\right] &= \mathsf{E}\left[\mathsf{E}\left[(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)}) | \boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(t-1)}\right]\right] \\ &= \mathsf{E}\left[(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \mathsf{E}\left[\mathbf{g}(\boldsymbol{\theta}^{(t)}; \boldsymbol{\xi}^{(t)}) | \boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(t-1)}\right]\right] \\ &= \mathsf{E}\left[(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \nabla f(\boldsymbol{\theta}^{(t)})\right]\end{aligned}$$

Fact 6.1.1. If the function is μ-strongly convex, then $\left(\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\right)^\top (\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) \geq \mu \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2^2$. Prove it.

By Fact 6.1.1 and the fact that the gradient at $\boldsymbol{\theta}^*$ is zero vector, we have

$$\mathsf{E}\left[(\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*)^\top \nabla f(\boldsymbol{\theta}^{(t)})\right] \geq \mu \mathsf{E}\left[\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\right] \tag{13}$$

By Equations 12 and 13, we have

$$\mathsf{E}\left[\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2\right] \leq \left(1 - 2\mu\eta_t + \eta_t^2 M_g L^2\right) \mathsf{E}\left[\|\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\right] + \eta_t^2 \sigma_{\mathbf{g}}^2$$

Note that we have followed the first step in our general strategy: relating the subsequent steps of the iterative algorithm. Now one could to solve (although because of the presence of the constant $\eta_t^2 \sigma_{\mathbf{g}}^2$, it might be not as easy) the above recursion equation to obtain the desired result

$$\mathsf{E}\left[\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\right] \leq \frac{c_0}{t+1}$$

where c_o is a numerical constant. □

Note that in the last step, $\sigma_{\mathbf{g}}^2$ and M_g are constants bounding the variance of the stochastic gradient. If $\sigma_{\mathbf{g}}^2 = M_g = 0$ (which means we are in the deterministic optimization setup), we then have the much simpler recursion we had for the deterministic gradient descent algorithm! The algorithm requires $t = \mathcal{O}(\frac{1}{\epsilon})$ iteration to get an ϵ-optimal solution. In

comparison, for the deterministic case, we only required $t = \mathcal{O}(\log\left(\frac{1}{\epsilon}\right))$ iterations.

Now, let us look at when is it advantageous to use the Stochastic Gradient Descent algorithm. In order to do so, we consider problems of the form in Equation 8 and solve them using: (i) gradient descent (by treating the problem as a deterministic one) and (ii) stochastic gradient descent (where we sample uniformly randomly one term for computing the gradient in each iteration). Using approach (i), we only require $\mathcal{O}(\log\left(\frac{1}{\epsilon}\right))$ to get ϵ-optimal solution. But in each iteration, we require n gradient computation. Hence the overall complexity is $n \cdot \mathcal{O}(\log\left(\frac{1}{\epsilon}\right))$. Using approach (ii), we require $\mathcal{O}(\frac{1}{\epsilon})$ iterations, but in each iteration, we require only 1 gradient computation. Hence the overall complexity is $\mathcal{O}(\frac{1}{\epsilon})$. Hence, when $\mathcal{O}(\frac{1}{\epsilon}) < n \cdot \mathcal{O}(\log\left(\frac{1}{\epsilon}\right))$, i.e., extremely large n and moderately accurate solution (which is the situation in big-data statistics), we prefer SGD.

Finally, in the context of MLE, the above theorem shows that the algorithm is consistent in expectation, i.e., $\mathsf{E}\left[\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2\right] \to 0$ as $t \to 0$. One can also show that if we average the iterates of the above algorithm, i.e., take

$$\bar{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_{t=0}^{t} \boldsymbol{\theta}^{(t)}$$

then $\bar{\boldsymbol{\theta}}^{(t)}$ is consistent almost surely and asymptotically (optimally) normal [PJ92] and quantify the rates of the convergence to normality as well [ABE19], if you are curious. Hence, you now have an algorithmic procedure that is consistent and asymptotically normal for parameter estimation, instead of the abstract minimizer of the MLE that is consistent and asymptotical normal, as discussed in STA 231A/B or STA 200A/B/C.

6.1.1 More intuition on SGD

Let $d = 1$ and $\boldsymbol{\xi} \sim N(\mu, 1)$ be a Gaussian random variable. Given n i.i.d. copies $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$, we know from basis statistics courses that the sample mean estimator (of the true mean μ) is given by

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\xi}_i$$

Furthermore, it is known that the population mean is the minimizer of the following objective function:

$$\mu = \arg\min_{\theta \in \mathbb{R}} \; f(\theta) = \arg\min_{\theta \in \mathbb{R}} \; \mathsf{E}\; (\theta - \boldsymbol{\xi})^2 \tag{14}$$

The SAA estimator is then given by

$$\tilde{\mu} = \arg\min_{\theta \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} (\theta - \boldsymbol{\xi}_i)^2 \tag{15}$$

The solution $\tilde{\mu}$ is indeed equal to the sample mean $\hat{\mu}$, i.e., $\tilde{\mu} = \hat{\mu}$. Now, let us use the SGD algorithm to find the solution of Equation 15 directly. The stochastic gradient based is given by $2\,(\theta - \boldsymbol{\xi})$. Hence the SGD iterations are given by

$$\begin{aligned} \theta^{(t+1)} &= \theta^{(t)} - 2 \cdot \eta_t (\theta^{(t)} - \boldsymbol{\xi}^{(t)}) \\ &= \theta^{(t)}(1 - 2 \cdot \eta_t) + 2 \cdot \eta_t \boldsymbol{\xi}^{(t)} \end{aligned}$$

Now, let us assume $\eta_t = 1/2t$. Then, we have

$$\begin{aligned} \theta^{(t+1)} &= \frac{(t-1)}{t}\theta^{(t)} + \frac{1}{t}\boldsymbol{\xi}^{(t)} \\ \implies t \cdot \theta^{(t+1)} &= (t-1) \cdot \theta^{(t)} + \boldsymbol{\xi}^{(t)} \end{aligned}$$

Hence, we get the following sequences:

$$\begin{aligned} t = 1: \quad & \theta^{(2)} && = \boldsymbol{\xi}^{(1)} \\ t = 2: \quad & 2\theta^{(3)} && = \theta^{(2)} + \boldsymbol{\xi}^{(2)} = \boldsymbol{\xi}^{(1)} + \boldsymbol{\xi}^{(2)} \\ t = 3: \quad & 3\theta^{(4)} && = 2\theta^{(3)} + \boldsymbol{\xi}^{(3)} = \boldsymbol{\xi}^{(1)} + \boldsymbol{\xi}^{(2)} + \boldsymbol{\xi}^{(3)} \\ t = 4: \quad & 4\theta^{(5)} && = 3\theta^{(4)} + \boldsymbol{\xi}^{(4)} = \boldsymbol{\xi}^{(1)} + \boldsymbol{\xi}^{(2)} + \boldsymbol{\xi}^{(3)} + \boldsymbol{\xi}^{(4)} \end{aligned}$$

From the above expressions, we see that we are basically estimating the sample mean via the iterative stochastic gradient descent algorithm. The main difference is that, we are doing it *on the fly*, i.e., we don't need to store all the n (or (t)) samples all the time. We just use the sample once and throw it away. Of course, this is the natural procedure one would come up with if you are asked to estimate the sample mean *on the fly*!

7 Deterministic Optimization Problems with Random Gradients

Previously we studied deterministic optimization problems, where the gradient vector was a deterministic vector. We also studied stochastic optimization problems, where the gradient vector was a random vector because computing expected gradient might be costly or impossible. Now, we consider a class of algorithms for deterministic optimization where computing the gradient is impossible or prohibitive and hence we use random estimators of the true gradient.

7.1 Zeroth-Order Optimization

So far we covered algorithms for deterministic and stochastic optimization using gradient information. Such methods are collectively called as first-order algorithms as they are using gradient information in one way or the other. In some situations, it might be difficult (or even impossible) to calculate the gradient of the objective function $f(\boldsymbol{\theta})$. But it would be possible to evaluate the function at any point $\boldsymbol{\theta}_1$ and obtain $f(\boldsymbol{\theta}_1)$. Minimizing a function $f(\boldsymbol{\theta})$ when we are able to evaluate the function at different point is called as zeroth-order optimization.

Specifically, let us stick to deterministic optimization problems of the form

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} f(\boldsymbol{\theta}) \tag{16}$$

when we can only evaluate it point-wise. If we know the function $f(\boldsymbol{\theta})$, we can run the gradient descent iterations. Without this information, one way to proceed is to **estimate** the gradient information from just point-wise evaluations and plug-in those estimates of the gradients in the gradient descent iterations. The fundamental problem boils down to estimating the gradient from just point-wise evaluations of the functions. We leverage Stein's identity to accomplish the task.

Fact 7.0.1 (Stein's Identity)**.** A random vector $\mathbf{u} \in \mathbb{R}^d$ is standard Gaussian if and only if $\mathsf{E}_{\mathbf{u}}[\mathbf{u}\, h(\mathbf{u})] = \mathsf{E}[\nabla_{\mathbf{u}}\, h(\mathbf{u})]$ is true for all functions h with well-defined gradients.

Based on Fact 7.0.1, we have the following estimator for the gradient of the function $f(\boldsymbol{\theta})$ at a given point $\boldsymbol{\theta}_1$:

$$\mathbf{g}(\boldsymbol{\theta}_1) = \frac{f(\boldsymbol{\theta}_1 + \nu\mathbf{u}) - f(\boldsymbol{\theta}_1)}{\nu}\, \mathbf{u}$$

Note that $\mathbf{g}(\boldsymbol{\theta}_1)$ is a random estimator of the deterministic gradient vector $\nabla f(\boldsymbol{\theta}_1)$. Define a closely related function $f_\nu(\boldsymbol{\theta}) = \mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta} + \nu\ \mathbf{u})]$. It turns out $\mathbf{g}(\boldsymbol{\theta}_1)$ is a **biased** estimator of the gradient vector $\nabla f(\boldsymbol{\theta}_1)$, but is an unbiased estimator of the gradient vector $\nabla f_\nu(\boldsymbol{\theta}_1) = \nabla_{\mathbf{u}}\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta} + \nu\ \mathbf{u})] = \mathsf{E}_{\mathbf{u}}[\nabla_{\mathbf{u}} f(\boldsymbol{\theta} + \nu\ \mathbf{u})]$ (assuming integration and differentiation could be interchanged). To see that, note that we have

$$\begin{aligned}
\mathsf{E}_{\mathbf{u}}[\mathbf{g}(\boldsymbol{\theta}_1)] &= \frac{\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1 + \nu\mathbf{u})\mathbf{u}]}{\nu} - \frac{\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1)\mathbf{u}]}{\nu} \\
&= \frac{\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1 + \nu\mathbf{u})\mathbf{u}]}{\nu} \\
&= \frac{\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1 + \nu\mathbf{u})\mathbf{u}]}{\nu}
\end{aligned}$$

Now, we need to compute $\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1 + \lambda\mathbf{u})\mathbf{u}]$. To do so, we apply Fact 7.0.1 with $h(\mathbf{u}) =$

$f(\boldsymbol{\theta}_1 + \nu\ \mathbf{u})$. Doing so, we get

$$\mathsf{E}_{\mathbf{u}}[f(\boldsymbol{\theta}_1 + \nu\mathbf{u})\mathbf{u}] = \mathsf{E}_{\mathbf{u}}[\nabla f(\boldsymbol{\theta}_1 + \nu\mathbf{u})] \cdot \nu$$

From this, we see that

$$\mathsf{E}_{\mathbf{u}}[\mathbf{g}(\boldsymbol{\theta}_1)] = \mathsf{E}_{\mathbf{u}}[\nabla f(\boldsymbol{\theta}_1 + \nu\mathbf{u})].$$

Note that, as $\nu \to 0$, the estimator becomes unbiased! So, a practical zeroth-order algorithm for solving any deterministic optimization problem of the form in Equation 16 is of the following form:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \mathbf{g}(\boldsymbol{\theta}^{(t)}) \tag{17}$$

where

$$\mathbf{g}(\boldsymbol{\theta}^{(t)}) = \frac{f(\boldsymbol{\theta}^{(t)} + \nu\mathbf{u}) - f(\boldsymbol{\theta}^{(t)})}{\nu}\ \mathbf{u}$$

Two main differences from the standard gradient descent algorithms are as follows: (1) We need to set two tuning parameters η_t and ν for this algorithm and (2) It turns out that the iteration complexity to achieve an ϵ-accurate solution using the zeroth-order gradient descent algorithm for μ-strongly convex and L-smooth function depends linearly on the problem dimensionality d (where as the standard gradient descent algorithms are dimension-free). Finally, although we discussed this idea of zeroth-order optimization in the context of deterministic optimization problems, it could also be used for stochastic optimization problems.

7.2 Randomized Coordinate Descent Algorithm

The idea behind coordinate descent algorithms is to pick a coordinate randomly and update the gradient only along that direction while running the gradient descent algorithm. For a function $f(\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}$, denote by

$$g_i(\boldsymbol{\theta}) = \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \qquad \forall i \in \{1, \ldots, d\}$$

Here, we denote by $\boldsymbol{\theta}_i$, the i-th coordinate of the vector $\boldsymbol{\theta} \in \mathbb{R}^d$. Note that $g_i(\boldsymbol{\theta})$ is not a vector, but a scalar. With this notation, the coordinate descent algorithm runs as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t g_{i_t}(\boldsymbol{\theta}^{(t)}) \cdot \mathbf{e}_{i_t} \tag{18}$$

where $i_t \in \{1, \ldots, d\}$ is the randomly picked coordinate in the t-th step of the algorithm and vectors $\mathbf{e}_j \in \mathbb{R}^d$, for $j = 1, \ldots, d$ represent the standard basis for $\mathbb{R}^d$.

To under the performance of this algorithm, we still assume the function is μ-strongly convex. But we make the following change to the function in terms of the L-smoothness assumption. Recall that we had $\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\|_2 \leq L\|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_2$ before. This is a *global*

lipschitz continuity assumption. We now introduce a coordinate wise Lipschitz continuity assumption, for $\lambda \in \mathbb{R}$ and for all $\boldsymbol{\theta} \in \mathbb{R}^d$:

$$\|\nabla f(\boldsymbol{\theta} + \lambda \mathbf{e}_j) - \nabla f(\boldsymbol{\theta})\|_2 \leq L_i|\lambda| \qquad \forall\ j \in \{1, \ldots, d\}$$

Note that this measures how smooth the gradients are along each coordinate wise. Define

$$L_{\max} = \max_j\ L_j$$

One can show that, under some (mild, technical) assumptions, L and $L_{\max}$ are related as follows:

$$1 \leq \frac{L}{L_{\max}} \leq d$$

This is what we meant when we mentioned that the Lipschitz constant might "hide" the dimensionality fact in point 6 in page 6. Specifically, we see that if the function f is coordinate wise smooth (in the above sense), then in the worst case, we can have $L \approx dL_{\max}$.

Similar to the gradient descent algorithm, for μ-strongly convex and coordinate wise smooth function, we have the following result for randomized coordinate descent.

Theorem 7.1. *Let $\boldsymbol{\theta}^{(0)}$ be the initial point and let us run the iterations defined in Equation 17 for minimizing the function in Equation 16. Assume that $f(\boldsymbol{\theta})$ is μ-strongly convex and coordinate wise smooth. If*

$$\eta_t = \eta = \frac{1}{L_{\max}},$$

then

$$E[f(\boldsymbol{\theta}^{(t)})] - f(\boldsymbol{\theta}^*) \leq \left(1 - \frac{c_0}{dL_{\max}}\right)^t f(\boldsymbol{\theta}^{(0)}) - f(\boldsymbol{\theta}^*)$$

for some constant c_0.

8 Newton's method

Newton's method is an optimization technique that uses information about the Hessian of the function being optimized. Hence, it a second-order method (as opposed to gradient descent methods that are first-order methods). The iterates of the Newton's method are motivated by a method for finding the roots of functions (In this scenario, the method is also called as Newton-Raphson Method). For a function $h() : \mathbb{R} \to \mathbb{R}$, consider finding the roots, i.e., θ^* such that,

$$h(\theta^*) = 0.$$

Let $\theta^{(0)} \in \mathbb{R}$ be a point close to θ^* such that $\theta^* = \theta^{(0)} + r$. Then, by Taylor's theorem, we have

$$0 = h(\theta^*) = h(\theta^{(0)} + r) \approx h(\theta^{(0)}) + r\nabla h(\theta^{(0)})$$

so unless $\nabla h(\theta^{(0)})$ is close to 0, we have

$$r \approx -\frac{h(\theta^{(0)})}{\nabla h(\theta^{(0)})}$$

Hence, we have

$$\theta^* = \theta^{(0)} + r \approx \theta^{(0)} - \frac{h(\theta^{(0)})}{\nabla h(\theta^{(0)})}$$

The right hand side of the above equation could be thought of as a new guess about the point θ^* that we are trying to find. Hence, the iterates

$$\theta^{(t+1)} = \theta^{(t)} - \frac{h(\theta^{(t)})}{\nabla h(\theta^{(t)})},$$

produce a sequence $\theta^{(t+1)}$ that are increasingly close to θ^*. The same argument could be generalized to multivariate settings with gradient being replaced by Jacobians.

What does this have to do with optimization problem of the form in Equation 16 ? Recall that at the global solution $\boldsymbol{\theta}^*$ of the problem in Equation 16, the gradient vector has to be zero (under some mild technical assumptions to avoid technical issues) i.e,

$$\nabla f(\boldsymbol{\theta}^*) = 0.$$

Considering again the case of $d = 1$ for intuition, we have $\nabla f(\theta^*) = h(\theta^*) = 0$ which gives raise to the updates of the form

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\nabla f(\theta^{(t)})}{\nabla^2 f(\theta^{(t)})}$$

In the multivariate setting, motivated by the above intuition, the Newton's method for optimization proceeds as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left[\nabla^2 f(\boldsymbol{\theta}^{(t)})\right]^{-1} \nabla f(\boldsymbol{\theta}^{(t)}) \tag{19}$$

Let us again go to $d = 1$ to gain more intuition. Consider finding the root of the following function

$$\nabla f(\theta) = h(\theta) = \frac{\theta}{\sqrt{1+\theta^2}}$$

Clearly $\theta^* = 0$ and furthermore,

$$\nabla^2 f(\theta) = \nabla h(\theta) = \frac{1}{(1+\theta^2)^{3/2}}$$

The Newton's method proceeds as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{h(\theta^{(t)})}{\nabla h(\theta^{(t)})} = \theta^{(t)} - \frac{\theta^{(t)}}{\sqrt{1+[\theta^{(t)}]^2}} \cdot (1+\theta^2)^{3/2} = -[\theta^{(t)}]^3$$

So, we see that when the initial point $\theta^{(0)}$ is such that $|\theta^{(0)}| < 1$, then the iterates converges extremely fast! When $\theta^{(0)} = \pm 1$, then the iterates just oscillates. When $|\theta^{(0)}| > 1$, the method diverges!

In the context of optimization, $h(\theta)$ corresponds to the gradient and $\nabla h(\theta)$ corresponds to the Hessian. The Hessian is positive for all θ which means that the function we started with is a convex function! So, even for optimizing convex functions, initialization matters **a lot** for Newton's method! If the initialization is right, the method is **extremely fast**. But this happens only after the iterates a particular region close to the global solution $\boldsymbol{\theta}^*$! This phenomenon is summarized by the following theorem which quantifies the rate of convergence for Newton's method for μ-strongly convex and L-smooth functions.

Theorem 8.1 (Rough Statement)**.** *Assume the function $f(\boldsymbol{\theta})$ is μ-strongly convex and L-smooth. The iterates of the Newton's method in Equation 19 satisfies, for some numerical constant $0 < C < 1$*

$$\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \leq C\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2$$

for all $t > t_N$ where $t_N > 0$.

Recall that with gradient descent, we had $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \leq C\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2$ or equivalently $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2^2 \leq C\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2^2$. In Newton's method we have the L_2 on one-side and the squared L_2 norm on the other side! In terms of recursions, we can then show that (approximately)

$$\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|_2 \leq C^{2^t}\|\boldsymbol{\theta}^{(t_N)} - \boldsymbol{\theta}^*\|_2$$

for all $t > t_N$. In comparison, for gradient descent, we had $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|_2 \leq C^t\|\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}^*\|_2$ for all $t > 0$. What this is means is the following: As long as the iterates *somehow* enters a region around the global solution $\boldsymbol{\theta}^*$ (the time taken for this to happen is denoted by t_N), then to get ϵ close to the global solution $\boldsymbol{\theta}^*$, we only need

$$t \approx \mathcal{O}\left(\log\log\left(\frac{1}{\epsilon}\right)\right)$$

iterations. The overall iteration complexity (to get ϵ-close to $\boldsymbol{\theta}^*$) of the Newton's method is

given by

$$t_N + \mathcal{O}\left(\log\log\left(\frac{1}{\epsilon}\right)\right).$$

This phenomenon is called as locally fast convergence. But keep in mind that t_N might be infinity as well in the worst-case! There are two ways to avoid this issue.

- **Damped Newton's method:** The iterates of the Damped Newton's method is given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \left[\nabla^2 f(\boldsymbol{\theta}^{(t)})\right]^{-1} \nabla f(\boldsymbol{\theta}^{(t)}) \tag{20}$$

  for some tuning-parameter $\eta_t > 0$.

- **Trust-Region method:** The iterates of the Trust-Region methods are given by

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\mathbf{v} \in \mathcal{V}^{(t)}} \left\{ (\mathbf{v} - \boldsymbol{\theta}^{(t)})^\top \nabla f(\boldsymbol{\theta}^{(t)}) + \frac{1}{2}(\mathbf{v} - \boldsymbol{\theta}^{(t)})^\top \nabla^2 f(\boldsymbol{\theta}^{(t)})(\mathbf{v} - \boldsymbol{\theta}^{(t)}) \right\}$$

  where $\mathcal{V}^{(t)}$ is called as the Trust-region of the iterate (t) and is given by

$$\mathcal{V}^{(t)} = \{\boldsymbol{\theta} \in \mathbb{R}^d : \|t - \boldsymbol{\theta}^{(t)}\| \leq \gamma\},$$

  for some $\gamma > 0$. When $\gamma = \infty$, then $\mathcal{V}^{(t)} = \mathbb{R}^d$ and we recover the original Newton's method.

- **Cubic Regularized Newton's method:** The iterates of the Damped Newton's method is given by

$$\boldsymbol{\theta}^{(t+1)} = \arg\min_{\mathbf{v} \in \mathbb{R}^d} \left\{ (\mathbf{v} - \boldsymbol{\theta}^{(t)})^\top \nabla f(\boldsymbol{\theta}^{(t)}) + \frac{1}{2}(\mathbf{v} - \boldsymbol{\theta}^{(t)})^\top \nabla^2 f(\boldsymbol{\theta}^{(t)})(\mathbf{v} - \boldsymbol{\theta}^{(t)}) + \frac{M}{6}\|\mathbf{v} - \boldsymbol{\theta}^{(t)}\|_2^3 \right\} \tag{21}$$

  for some $M > 0$. This is by far the most understood and preferred method in practice (along with the trust-region method). One can show that for this method, starting at any initial point, we will converge the region of fast convergence, very quickly. That is, the t_N for this method is typically is a small number. But note that, similar to the projected gradient type methods, each update itself is given by another optimization problem which might be difficult to solve in some cases. It turns out that one can efficiently solve the above optimization problems using specialized techniques.

8.1 Approximately the Hessian

Recall that each of the iterates of the Newton's method involves computing (and in some cases inverting) the Hessian matrix which is a $d \times d$ matrix. This method might hence be

computationally prohibitive in practice. Hence the question of approximately computing the Hessian matrix becomes important (especially given the fast convergence of the Newton's method). An array of techniques called as quasi-Newton methods have be proposed by various people to approximately compute the Hessian matrix. In the extreme setting, we have the gradient descent method which approximates the hessian matrix as $\nabla^2 f(\boldsymbol{\theta}) = \mathbf{I}$. Similarly, one can just compute only the diagonal entries of the Hessian which would be a diagonal approximation to the Hessian matrix. More sophisticated techniques like LBFGS technique also exists, if you are curious.

8.1.1 Computing Hessian from function values

Similar to the gradient setting, one can compute the Hessian matrix itself from function values by using the following version of Stein's identity.

Fact 8.1.1 (Stein's Identity)**.** For random vector $\mathbf{u} \in \mathbb{R}^d$ that is standard Gaussian, we have

$$\mathsf{E}_{\mathbf{u}}[(\mathbf{u}\mathbf{u}^\top - \mathbf{I})\ h(\mathbf{u})] = \mathsf{E}\left[\nabla^2_{\mathbf{u}}\ h(\mathbf{u})\right]$$

Based on Fact 8.1.1, we have

$$\mathbf{H}(\boldsymbol{\theta}_1) = \frac{f(\boldsymbol{\theta}_1 + \nu\mathbf{u}) + f(\boldsymbol{\theta}_1 + \nu\mathbf{u}) - 2f(\boldsymbol{\theta}_1)}{\nu^2}\ (\mathbf{u}\mathbf{u}^\top - \mathbf{I})$$

This provides a way of computing the Hessian using only function values. Furthermore, note that when using cubic-regularized Newton's method, we require the product of the Hessian matrix and a vector. The operation of Hessian-vector product with the above version of Hessian could be performed extremely fast, i.e., linear in d, as it just involves inner-products.

References

[ABE19] Andreas Anastasiou, Krishnakumar Balasubramanian, and Murat A Erdogdu, *Normal approximation for stochastic gradient descent via non-asymptotic rates of martingale clt*, arXiv preprint arXiv:1904.02130 (2019).

[DSSSC08] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra, *Efficient projections onto the l 1-ball for learning in high dimensions*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 272–279.

[Jag13] Martin Jaggi, *Revisiting frank-wolfe: Projection-free sparse convex optimization*, International Conference on Machine Learning, 2013, pp. 427–435.

[PJ92] Boris T Polyak and Anatoli B Juditsky, *Acceleration of stochastic approximation by averaging*, SIAM Journal on Control and Optimization **30** (1992), no. 4, 838–855.