Optimization Algorithms **Winter 2023**

Lecture 4: Dimension-free convex optimization

Lecturer: Constantine Caramanis *Scribes: Chanho Jung*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Disclaimer: *These notes are based on the freely available online lectures by Constantine Caramanis.*

4.1 Oracle Lower Bounds

A natural question now that we've derived this is are these optimal. so is it possible for example that our analysis was loose or is it impossible that there is a better algorithm out there. this is by itself a subtle question to ask because there's no such thing as an optimal algorithm for a specific optimization problem. every specific optimization algorithm problem has an answer so the algorithm that just outputs that answer is unbeatable for that specific function but of course is not going to be very good for other functions. so this is how we're gonna make this concept a little bit more precise and this will allow us to reason about optimality of certain algorithms.

4.1.1 Review: Rates of Convergence - are the optimal?

Subgradient method : $x_{t+1} = x_t - \eta g_t$, where $g_t \in \partial f(x_t)$

Gradient method : $x_{t+1} = x_t - \eta \nabla f(x_t)$

Subgradient method for Lipschitz convex functions : Lipschitz means that subgradients are uniformly bounded so $\exists G$, $||g||_2 \leq G$, $\forall g \in \partial f(x)$, as long as f is convex and satisfies this property then we saw that the sub gradient method guarantees that after T iterations,

$$Error : \mathcal{O}\left(\frac{1}{\sqrt{T}}\right) \leftrightarrow \mathcal{O}\left(\frac{1}{\epsilon^2}\right)$$

Gradient descent for smooth convex functions :

$$Error : \mathcal{O}\left(\frac{1}{T}\right) \leftrightarrow \mathcal{O}\left(\frac{1}{\epsilon}\right)$$

Gradient descent for β-smooth and α-strongly convex functions :

$$Error : \mathcal{O}\left(\left(\frac{\kappa - 1}{\kappa + 1}\right)^T\right) \leftrightarrow \mathcal{O}\left(\kappa \log\left(\frac{1}{\epsilon}\right)\right), \kappa = \frac{\beta}{\alpha}$$

4.1.2 Oracle model of computation

What do we know about our functions? :

$$min : f(x), st : x \in \mathfrak{X} = \mathbb{R}^d$$

the way to make precise this idea of complexity and optimality in terms of how of convergence rate is to say what do we know about our function what do we use for every single algorithm that we've seen so far we found that all we need is to be able to evaluate the function and also evaluate its gradient or to get an arbitary element of the sub differential. in other words think about this as though we don't actually know our function f at all. all we have access to is something that I'll call an Oracle.

$$x \longrightarrow \boxed{\text{First order Oracle}} \longrightarrow f(x), \nabla f(x) \text{ or } g_x {\in} \partial f(x)$$

Let's consider only algorithms that can be cast as having access to our function only through this Oracle. Suppose the only access we have to our function f is through the Oracle. then we can frame the question as follows.

How many calls to the Oracle do we need in the worst case over a class of functions, in order to guarantee error ϵ?

we're making no assumptions about what you do once you get back the gradient or the subgradient of f. you could be doing some incredibly complicated procedure possibly exponential time. This question doesn't care about that it's just saying how many times you need to evaluate this Oracle.

4.1.3 Oracle based lower bounds

For the class of Lipchitz convex functions : the subgradient method is unimprovable, so it is optimal.

this is the class for which subgradient method guarantees $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$, there is no algorithm which can guarantee error better than $\mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$ for all functions in this class(In the worst case)

Note: *That doesn't mean if I have a particular Lipschitz and convex function I can't find a better algorithm than subgradient descent. It means that I can't find an algorithm that uses fewer than T square root evaluations.If you don't have any other information about your function then this is the best.*

For the case of smooth functions : can be imporved

$$\text{the lower bound is } \mathcal{O}\left(\frac{1}{T^2}\right) \leftrightarrow \mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$$

For β-smooth and α-strongly convex functions : can be imporved

$$\text{the lower bound is } \mathcal{O}\left(\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^T\right) \leftrightarrow \mathcal{O}\left(\kappa \log\left(\frac{1}{\epsilon}\right)\right), \kappa = \frac{\beta}{\alpha} {\geq} 1 \Rightarrow \sqrt{\kappa} {<} \kappa$$

4.1.4 Discussion about the oracle model

Is the assumption reasonable? What is the takeaway? : If we develop any algorithm that only uses evaluations of $f(x)$ and $\nabla f(x)$, then we are bound by the Oracle lower bound

this is the only thing we know about our function.

$$x \longrightarrow \boxed{\text{First order Oracle}} \longrightarrow f(x), \nabla f(x) \text{ or } g_x {\in} \partial f(x)$$

it' very rare that this is in fact the model that we have. In that sense, how is this helpful to us well it says the following thing one way to think about this Oracle model and what the takeaway is or what it word guarantees is if we develop any algorithm that only uses evaluations of $f(x)$ and $\nabla f(x)$.then we can imagine that we have only access to the Oracle in other words even if you know f if what you're the algorithm that you develop only uses evaluations of f and $\nabla f(x)$. then even if you didn't know f and you only knew you only had access to the Oracle you would be running exactly the same algorithm and therefore that means that you're bound by whatever lower bounds we have from the Oracle model.

4.2 Accelerated Gradient Descent

we're now going to talk about an improved algorithm that uses gradients and point evaluations of function in the case where our function has smoothness.

Recall

f is β-smooth, GD: $\mathcal{O}\left(\frac{1}{T}\right)$,LB: $\mathcal{O}\left(\frac{1}{T^2}\right)$

f is β-smooth and α-strongly convex, GD: $\mathcal{O}\left(\left(\frac{\kappa-1}{\kappa+1}\right)^T\right)$, LB: $\mathcal{O}\left(\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^T\right)$, $\kappa = \frac{\beta}{\alpha}$

so, we want to see if we can do better

4.2.1 Accelerated Gradient Descent: momentum

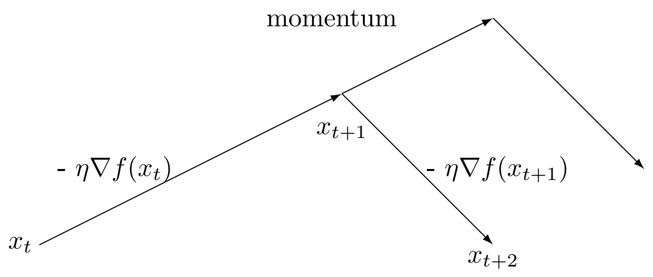
The key idea here is a concept called momentum. so let's recall what the basic gradient descent algorithm

Gradient method : $x_{t+1} = x_t$- $\eta\nabla f(x_t)$

the idea of momentum is to include a term here that capture some aspect of where I was before the direction that I was moving in.so the name is suggestive.

Idea of momentum : x_t - x_{t-1}

x_t - x_{t-1} gives me the direction in which I was moving in before.



4.2.2 Acceleration: Smooth and strongly convex functions

the algorithm sometimes called momentum sometimes called accelerated gradient descent is a following.

$$min: f(x) \text{ , } f \text{ is } \beta\text{-smooth and } \alpha\text{-strongly convex}$$

$$\text{Gradient descent: } x_{t+1} = x_t\text{- } \eta\nabla f(x_t) \text{ , } \eta=\frac{1}{\beta}$$

Accelerated Gradient descent(momentum): initialize: $x_1 = y_1 = x_{init}$, after t iterations

$$y_{t+1} = x_t \text{ - } \eta \nabla f(x_t)$$

$$x_{t+1} = \left(1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) y_{t+1} \text{ - } \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) y_t$$

x_{t+1} is convex combination between y_{t+1} and y_t. x_{t+1} depends not only on x_t but also x_{t-1}. because y_{t+1} is a function of x_t and therefore y_t is a function of x_{t-1}. So x_{t+1} is a function of the past two iterates. that's exactly where the momentum comes in.

Theorem: $f(y_t)$ - $f(x^*) \leq \frac{\alpha+\beta}{2}||x_1 - x^*||^2 \cdot exp(-\frac{(t+1)}{\sqrt{\kappa}})$

we have improved from $\kappa \rightarrow \sqrt{\kappa}$.

4.2.3 Acceleration vs Gradient descent: an example

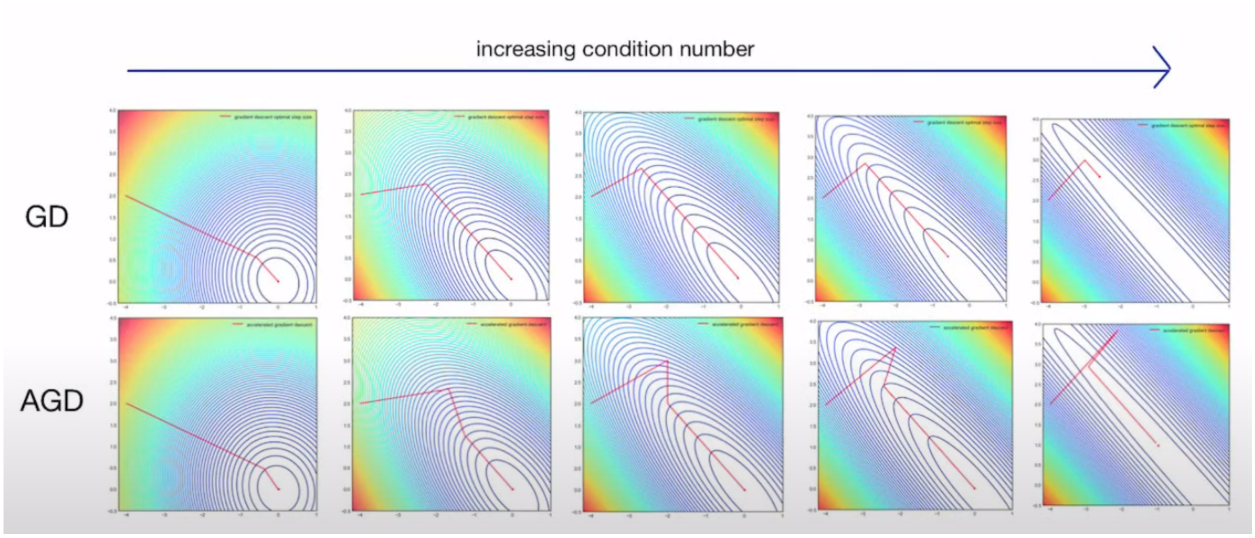


Figure 4.1: Trajectories of GD and AGD for different values of $\kappa = \frac{\beta}{\alpha}$

β as we've seen is equal to the largest eigenvalue and α is equal to the smallest eigenvalue. this is why where this terminology comes in of the condition number as a standart definition in this case of the condition number of a matrix.

Let's look at what we have starting from small condition number. small condition number means that the largest eigenvalue and smallest eigenvalue are pretty close together. contours are very close to being spherical which is what we would expect if the condition number is close to one.

gradient descent is doing what is locally best ignoring everything else it's just looking very locally and doing the moving in the optimal direction infinitesimally optimal direction. as a condition number increase here the trajectories of GD and AGD begin to look significantly different.*we're starting to see the impact of momentum.* the third column or it's even more pronounced in the fourth and fifth column you can see that in the same number of iterations and they both run for their optimal step size which is $\frac{1}{\beta}$, AGD is outperforming GD.

we see two things. first of all AGD is significantly outperforming GD but also we start seeing increasingly pronounced overshoot which is something else that we would expect from having this momentum terms.

Conclusion GD is always moving to a closer and closer contour.GD is in fact a descent algorithm it strictly improves at every single iteration untill it gets to the optimal solution but this is again not the case for AGD. There are situations where AGD takes a step and it actually will increase the function value nevertheless it is faster and it achieves the lower bounds so in that sense it's optimal.

4.2.4 Acceleration: smooth functions

We will show we match $LB : \mathcal{O}\left(\frac{1}{T^2}\right)$ compared to convergence rate :$\mathcal{O}\left(\frac{1}{T}\right)$

$$f \text{ is convex and } \beta\text{-smooth}$$

$$\text{Gradient step: } x^+ = x \text{ - } \eta\nabla f(x_t) \text{ , } \eta{=}\tfrac{1}{\beta}$$

$$\text{Momentum term: } d_t = \gamma_t(x_t - x_{t-1}) \text{ , } \gamma_t \text{ is a parameter we choose}$$

$$\text{GD algorithm: } x_{t+1} = (x_t)^+$$

$$\text{Nesterov Acceleration: } (x_t + d_t)^+ = x_t + d_t \text{ - } \eta\nabla f(x_t + d_t)$$

Proof of convergence rate

Thinking back to our proof of $\mathcal{O}\left(\frac{1}{T}\right)$ convergence rate for a gradient descent there were two things that we needed repeatedly namely upper and lower bounds on f.

f is convex : $f(x) - f(y) \leq \langle\ \nabla f(x), x - y\rangle$. this is a linear lower bound on $f(y)$. And because of smoothness I also have a quadratic upper bound on $f(y)$. if we phrase if we write y as the update.

$f(y) \leq f(x) + \langle\ \nabla f(x), x - y\rangle + \frac{\beta}{2}||y - x||^2$.

f is β-smooth : $f(x^+) - f(x) \leq -\frac{1}{2\beta}||\nabla f(x)||^2$, $\eta = \frac{1}{\beta}$. we've mentioned over and over again this is a statement that gradient descent is in fact a descent algorithm and this was the basis the starting point for the proof.

Notation: $\delta_t = f(x_t) - f(x^*) \geq 0$, $g_t = -\frac{1}{\beta}\nabla f(x_t + d_t)$: $x_{t+1} = x_t + d_t + g_t$

there is two quantities that I want to bound.

(a) $\delta_{t+1} - \delta_t \leq -\frac{\beta}{2}\left(||g_t||^2 + 2g_t^T d_t\right)$

(b) $\delta_{t+1} \leq -\frac{\beta}{2}\left(||g_t||^2 + 2g_t^T\left(x_t + d_t - x^*\right)\right)$

$$\begin{aligned}
\delta_{t+1} - \delta_t &\leq -\frac{\beta}{2}\left(||g_t||^2 + 2g_t^T d_t\right) \\
&= -\frac{\beta}{2}\left(\frac{1}{\beta^2}||\nabla f(x_t + d_t)||^2 - \frac{2}{\beta}\nabla f(x_t + d_t)^T d_t\right) \\
&= -\frac{2}{\beta}||\nabla f(x_t + d_t)||^2 + \nabla f(x_t + d_t)^T d_t \\
&\geq f\left(x_t + d_t - \eta\nabla f(x_t + d_t)\right) - f(x_t + d_t) + f(x_t + d_t) - f(x_t) \leftarrow \text{This is LHS of (a)}
\end{aligned}$$

$$(\lambda_t - 1)\cdot(a) + 1\cdot(b) \le -\frac{\beta}{2}\left[\frac{1}{\lambda_t}\left(||x_t+\lambda_t d_t - x^* + \lambda_t g_t||^2 - ||x_t + \lambda_t d_t - x^*||^2\right)\right]$$

we would like for the RHS to be telescoping.

$$\text{we want, } x_t + \lambda_t d_t - x^* + \lambda_t g_t = x_{t+1} + \lambda_{t+1} d_{t+1} - x^*$$

$$\text{recall : } d_{t+1} = \gamma_{t+1}(d_t + g_t)$$

$$\begin{aligned} x_t + \lambda_t d_t - x^* + \lambda_t g_t &= x_{t+1} + \lambda_{t+1} d_{t+1} - x^* \\ &= x_t + d_t + g_t + \lambda_{t+1}\gamma_{t+1}(d_t+g_t) - x^* \end{aligned}$$

if $\lambda_t(d_t+g_t) = (\lambda_{t+1}\gamma_{t+1}+1)(d_t+g_t)$, then we can get what we want.

γ_t is a parameter that I'm able to choose. so I'm going to choose γ_{t+1},γ_t to make this telescope.

$$\frac{2}{\beta}u_{t+1} \coloneqq ||x_t + \lambda_t d_t - x^* + \lambda_t g_t||^2,$$

$$\frac{2}{\beta}u_t \coloneqq ||x_t + \lambda_t d_t - x^*||^2$$

multiplying by λ_t and remeber λ_t is also a free parameter. γ_t is a function of λ_t that still allow me to pick,

$$\lambda_t^2\cdot(\delta_{t+1}) - (\lambda_t^2 - \lambda_t)\delta_t \le u_t - u_{t+1}$$

we're going to pick λ_t to make this telescope. that is It holds $\lambda_t^2 - \lambda_t = \lambda_{t-1}^2$: $\lambda_t \approx t$ will satisfy this, specifically: $\lambda_t = \frac{-1\pm\sqrt{1+4\lambda_{t-1}^2}}{2}$

sum both sides for T iterations:

$$\lambda_T^2\delta_{T+1} - \lambda_0\delta_1 \le u_0 - u_T \le u_0 \Rightarrow \delta_{T+1} \le \frac{constant}{\lambda_T^2}$$

$$\Rightarrow f(x_t) - f(x^*) \le \frac{constant}{T^2}$$

Summary :

1. Lower bounds for smooth,convex optimization are matched by GD+Monentum method.
2. Choose the coefficients carefully.

References

[1] Amir Beck. *First-Order Methods in Optimization.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[2] Sébastien Bubeck. Convex optimization: Algorithms and complexity. 2014.

[3] Constantine Caramanis. Optimization algorithms, 2020.

[4] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning.* Cambridge University Press, 2020.