Online Convex Optimization

by Elad Hazan

Carlos Gonzalez

University of Oxford, Department of Economics

February, 2023

Introduction

- Today we will cover some basic notions of Online Convex Optimization (OCO)
- OCO is our workhorse is many relevant problems
  - Expert's Advice
  - Supporting Vector Machines (email classification problems)
  - Portfolio Selection
  - Graph Theory
  - Recommendation Systems
- Why OCO instead of conventional Convex Optimization? Optimization as a process

Framework and Notation

- In each iteration t player/learner makes an n-dimensional choice $x_t \in \mathcal{K}$, and observes a loss $f_t(x_t)$
- Where $\mathcal{K}$ is a closed convex decision set, and $f \in \mathcal{F} : \mathcal{K} \to \mathbb{R}$ a G-Lipschitz convex "cost function"
- Interestingly enough, f_t may change adversarily!
- So, certainly, for this to make sense we need (i) bounded losses in every t and (ii) a finite/structured decision set

Framework and Notation

- We are interested in developing algorithms $\mathcal{A} : ? \to \mathcal{K}$ where $?$ is usually the history of loss functions $f_1, \dots, f_{t-1}$ evaluated at the decision choices $x_1, \dots, x_{t-1}$, respectively
- And then, bound the regrets of those algorithms

$$\text{Regret}_{\mathcal{A}} = \sup_{f_{1,\dots,T} \in \mathcal{F}} \left\{ \sum_t f_t(x_t^{\mathcal{A}}) - \sum_t f_t(x^*) \right\} \tag{1}$$

- where x^* is defined as $\arg\min_{x \in \mathcal{K}} \sum_t f_t(x)$
- Literature is also interested in computational time and memory, but we will talk very briefly about those dimensions here

Some final remarks

- Whenever we deal with adversarial environments, some degree of randomization will always be needed in $\mathcal{A}$. Otherwise, the rival could always set a high cost on our deterministic strategy
- We are usually interested in the notion of constrained optimization as our "dreamed" $x_t \notin \mathcal{K}$, where $\mathcal{K}$ is D-bounded
- It is then useful to remember the notion of projections as

$$\Pi_{\mathcal{K}}(y) = \arg\min_{x \in \mathcal{K}} ||x - y|| \quad (2)$$

Gradient Descent

- We now have all the ingredients to talk about OCO
- But first, let's start by presenting two straightforward algorithms in **Offline** Convex Optimization, namely **Gradient Descent** and **Constrained Gradient Descent**
- This gentle introduction will allow us to compare familiar optimization algorithms with more advances OCO techniques
- In Offline CO, the object of interest is usually the optimization error $f(x_t) - f(x^*)$, not the regret

Gradient Descent

Algorithm 1 Gradient Descent

Input T, x_1, and step-sizes $\{\eta_t\}$
for $t = 1, \ldots, T$
$x_{t+1} = x_t - \eta_t \nabla_t$, where $\nabla_t = \nabla_t f(x_t)$
end for
return $\bar{x} = \arg\min_{x_t} \{f(x_t)\}$

Algorithm 2 Constrained Gradient Descent

Input T, $x_1 \in \mathcal{K}$, $\{\eta_t\}$
for $t = 1, \ldots, T$
$y_{t+1} = x_t - \eta_t \nabla_t$
$x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})$
end for
return $\bar{x} = x_{T+1}$

Gradient Descent

- Convergence rate of the optimization error of these algorithms depends on the properties of f
- We won't get too bugged with the details, but for γ-well-conditioned functions convergence rates are $\mathcal{O}(e^{-\gamma T})$
- A function is $\gamma = \frac{\alpha}{\beta}$ well-conditioned if it is α-strongly convex and β-smooth
- Proofs of convergence rates usually rely on "reductions":
  - Derive results for well-conditioned functions
  - Pick general convex functions, change them such that they become well-conditioned
  - Apply our theoretical guarantees
- Tighter bounds can be derived when tailored algorithms are designed from scratch for each type of function. However, the notion of reduction is of great interest to us

Support Vector Machine

- We present now a foundational example in offline optim: SVM
- Later, we describe alg which perform much faster under OCO
- Problem: We receive emails a which can be coded as multi-dimensional arrays of 1s and 0s and n of them have been b humanly labeled as spam
- Idea: To find a vector x which minimizes classification errors

$$\min_{x \in \mathbb{R}^d} \sum_{i \in n} \mathbb{1}(\text{sign}(x^\top a_i) \neq b_i) \tag{3}$$

- Unfortunately, to find such x is very difficult, so instead people have been using a "hinge" loss-function

$$\min_{x \in \mathbb{R}^d} \lambda \frac{1}{n} \sum_{i \in n} l_{a_i, b_i}(x) + \frac{1}{2} ||x||^2 \tag{4}$$

- where $l_{a,b}(x) = \max\{0, 1 - bx^\top a\}$

Supporting Vector Machine

- We can now solve the problem above using standard gradient descent for a strongly convex but non-smooth function such that

Algorithm 3 SVM via Gradient Descent

Input T, examples $\{a, b\}$, $x_1 = 0$, $\{\eta_t\}$
for $t = 1, \ldots, T$
  $\nabla_t = \lambda \frac{1}{n} \sum_i \nabla l_{a_i,b_i}(x_t) + x_t$, where
  $\nabla l_{a_i,b_i}(x_t) = -b_i a_i + b_i a_i \mathbb{1}(b_i x^\top a_i > 1)$
  $x_{t+1} = x_t - \eta_t \nabla_t$ using $\eta_t = \frac{2}{t+1}$
end for
return $\bar{x}_T = \frac{1}{T} \sum_t \frac{2t}{T+1} x_t$

Supporting Vector Machine

- Now we can simply restore on reduction intuitions above or in tailored algorithms to get a regret of $\mathcal{O}(\frac{1}{T})$
- However, this algorithm presents a problem of computational efficiency, as we need to compute n gradients in each iteration... We will come back to this later

Online Gradient Descent

- Time to come back to OCO. Now we are focused on minimizing regret (not optimization error)
- We can however connect our notion of regret with that of optimization error when $f_t = f$, using

$$f(\bar{x}_T) - f(x^*) \leq \frac{1}{T}\sum_t (f(x_t) - f(x^*)) = \frac{\text{Regret}_T}{T} \tag{5}$$

- Consider the following Online Analog of (Constrained) Gradient Descent

Online Gradient Descent

Algorithm 4 (Constrained) Online Gradient Descent

Input T, $x_1 \in \mathcal{K}$, $\{\eta_t\}$
for $t = 1, \ldots, T$
Select x_t and observe cost $f_t(x_t)$

$$y_{t+1} = x_t - \eta_t \nabla f_t(x_t)$$

$$x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})$$

end for
return $\bar{x} = x_{T+1}$

Online Gradient Descent

- You may wonder whether sublinear regret is even possible given that f_t can change in each iteration
- It turns out that OGD can achieve a regret $\leq \frac{3}{2} GD\sqrt{T}$, using stepsizes $\eta_t = \frac{D}{G\sqrt{t}}$
- OK, so ODG is not too bad, but can we do better? In other words, what is the lowest regret that any algorithm may achieve?
- **Theorem.** Any algorithm for OCO incurs regret of $\mathcal{O}(DG\sqrt{T})$ in the worst case. This is true even if the cost functions are generated from a fixed stationary distribution.

Sketch of a Proof for Lower Bound on OCO

- For simplicity assume that $\mathcal{K} = \{x \in \mathbb{R}^n : ||x||_\infty \leq 1\}$
- Consider now 2^n linear cost functions $f_v(x) = v^\top x$, where $v \in \{\pm 1\}^n$, so essentially for any x we pick, each of its elements can be weighted randomly
- Observe

$$D \leq \sqrt{\sum_i 2^2} = 2\sqrt{n}, G \leq \sqrt{\sum_i (\pm 1)^2} = \sqrt{n} \tag{6}$$

- If v is chosen at random with uniform probability $\mathbb{E}[f_t(x_t)] = \mathbb{E}[v_t^\top x_t] = 0$, by independence; and

$$\mathbb{E}[\min_{x\in\mathcal{K}} \sum_t f_t(x^*)] = \mathbb{E}[\min_{x\in\mathcal{K}} \sum_t \sum_i v_t(i) x_i] = n\mathbb{E}[- \mid \sum_t v_t(1) \mid] = \Omega(-n\sqrt{T}) \quad \square \tag{7}$$

Online Gradient Descent

- So... this is it? Is this at good as it gets in OCO?
- We can actually derive smaller upper and lower bounds for interesting classes of functions
- For instance, for strongly convex functions $\text{Regret}_T \leq \frac{2^2}{2\alpha}(1 + \log T)$ by setting $\eta_t = \frac{1}{\alpha t}$
- Unfortunately, smoothness does not buy us any improvements in OCO
- Interesting results can also be derived for exp-concave functions

Stochastic Gradient Descent

- We can also use our online techniques in offline problems. A good example is Stochastic Gradient Descent
- In this case, we also want to $\min_{x \in \mathcal{K}} f(x)$, but, additionally assume that we are given access to a noisy gradient oracle $\boldsymbol{O}(x) = \tilde{\nabla}_x : \mathbb{E}[\tilde{\nabla}_x] = \nabla f(x), \mathbb{E}[||\tilde{\nabla}_x||^2] \leq G^2$

Algorithm 5 (Constrained) Stochastic Gradient Descent

Input $\boldsymbol{O}(x)$, T, $x_1 \in \mathcal{K}$, $\{\eta_t\}$
for $t = 1, \dots, T$
Let $\tilde{\nabla}_t = \boldsymbol{O}(x_t)$
$y_{t+1} = x_t - \eta_t \tilde{\nabla}_t$
$x_{t+1} = \Pi_{\mathcal{K}}(y_{t+1})$
end for
return $\bar{x} = \frac{1}{T} \sum_t x_t$

SGD for SVM

- But by now it should be clear that if we define $f_t(x) = \tilde{\nabla}_t x$, we can recover our previous bounds of $\mathcal{O}(\frac{GD}{\sqrt{T}})$
- We can use similar intuitions in our traditional SVM setting but rather than considering $\nabla_t = \lambda \frac{1}{n} \sum_i \nabla l_{a_i,b_i}(x_t) + x_t$ we may simply use $\tilde{\nabla}_t = \lambda \nabla l_{a_t,b_t}(x_t) + x_t$
- For the appropriate η_t we can recover the same convergence rates using OCO than in the standard offline optimization
- However this algorithm is significantly quicker as it just computes one gradient per iteration (which is a noisy unbiased estimate of the true gradient)

Bandit Convex Optimization

- BCO is very similar to OCO. Minimize regret for a sequence of unknown f_t
- BUT it will not be realistic anymore to have an oracle $\tilde{\nabla}_t$
- Only feedback available is $f_t(x_t)$, so no chance of getting ∇_t neither
- We first study a special (but very general) case of BOC called Multi-Armed Bandit (MAB) problems
- Key element, each iteration t the learner selects an arm i_t from a pool of n arms
- Similar in spirit to traditional expert problems where $i_t = \Delta^n_{i_t}$ and $f_t = \sum_i l_t(i)x(i)$

Exploration vs Exploitation

- Almost immediately a trade-off emerges in this kind of problems
- We can either explore different arms to learn their "true" loss
- Exploit the arm with the highest estimated loss at some iteration t
- In fact, this simple intuition allows us to derive a first "naive" MAB alogirthm

Simple MAB Algorithm

Algorithm 6 Simple MAB Algorithm

Input T, OCO Algorithm $\mathcal{A}$, δ
for $t = 1, \ldots, T$
  Let b_t be a $Bern(\delta)$
  if $b_t = 1$ **then**
    Choose i_t uniformly at random
    Set $\hat{l}_t(i) = \mathbb{1}(i = i_t)\frac{n}{\delta} l_t(i)$, $\hat{f}_t(x) = \hat{l}_t^\top x$
    Update $x_{t+1} = \mathcal{A}(\hat{f}_1, \ldots, \hat{f}_t)$
  else
    choose $i_t \sim x_t$ and **update** $\hat{f}_t = 0$, $\hat{l}_t = 0$, $x_{t+1} = x_t$
  end if
end for

Simple MAB Algorithm

- Intuition: δ % of times we play i_t randomly and we obtain better approximations of the actual loss functions, so we can later apply an algorithm $\mathcal{A}$ on more precise estimates
- And $(1-\delta)$ % of times we play the "best" x we can based on the history of losses $f_1, \ldots, f_{t-1}$ using our algorithm $\mathcal{A}$ a
- **Theorem**

$$\mathbb{E}[\sum_t l_t(i_t) - \sum_t l_t(i^*)] \leq \mathcal{O}(T^{\frac{2}{3}} n^{\frac{2}{3}}) \tag{8}$$

- But we can certainly do better. For instance, we may simultaneously explore and exploit

Exp3 Algorithm

Algorithm 7 Exp3 Algorithm

Input T, $x_1 = (1/n)$, $\varepsilon > 0$
for $t = 1, \ldots, T$
  Choose $i_t \sim x_t$
  Let $\hat{l}_t(i) = \mathbb{1}(i = i_t)\frac{1}{x_t(i_t)} l_t(i)$
  Update $y_{t+1}(i) = x_t(i)e^{\varepsilon \hat{l}_t(i)}$, $x_{t+1} = \frac{y_{t+1}}{||y_{t+1}||_1}$
end for

Exp3 Algorithm

- Intuition: Every period we update the probability of choosing arm i_t based on the observed loss $\hat{l}_t(i)$
- This algorithm turns out to be near optimal with regret of $\mathcal{O}(\sqrt{Tn \log n})$

General BCO

- We now step back from MAB and dive into general BCO
- In particular, we learn how to reduce BCO problems into familiar OCO frameworks
- Intuition: Generate a rv g_t using observables in the BCO which are unbiased estimators of cost function gradients ($\mathbb{E}[g_t] \approx \nabla_t f_t(x_t)$)
- Then, apply OCO algorithms which rely **only** on gradients
- The type of algorithms which can still get sublinear regret using $\mathcal{A}(g_1, \ldots, g_{t-1})$ are called **first-order OCO**
- An algorithm is FO-OCO if the family of loss-functions is closed under addition and if $\hat{f}_t(x) = \nabla f_t(x_t)^\top x \implies \mathcal{A}(f_1, \ldots, f_{t-1}) = \mathcal{A}(\hat{f}_1, \ldots, \hat{f}_{t-1})$
- In a nutshell, we are generating our own approx oracle ∇_t

Reduction to Bandit Feedback

Algorithm 8 Reduction to Bandit Feedback

Input T, $\mathcal{K}$, FO-OCO $\mathcal{A}$
Let $x_1 = \mathcal{A}(\emptyset)$
for $t = 1, \ldots, T$
  Generate distribution $\mathcal{D}_t$, **sample** and **play** $y_t \sim \mathcal{D}_t$ with $\mathbb{E}[y_t] = x_t$
  Observe $f_t(y_t)$, use it to generate g_t st $\mathbb{E}[g_t] = \nabla f_t(x_t)$
  Set $x_{t+1} = \mathcal{A}(g_1, \ldots, g_t)$
end for

Reduction to Bandit Feedback

- Under very mild conditions this reduction ensures the same regret bounds as $\mathcal{A}$ up to the magnitude of g_t
- Now that we know that our intuitions work we are in position to describe how such g_t can be obtained and which is the form of $\mathcal{D}_t$
- Easy example for a one dimensional case
- To compute the derivative $f'(x)$ we need at least two points $f(x+\delta)$ and $f(x-\delta)$. i.e.

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x-\delta)}{2\delta} \tag{9}$$

Reduction to Bandit Feedback

- But in bandit frameworks we just have access to a single observation. Solution?

$$g(x) = \begin{cases} f(x+\delta) & \text{with probability } \frac{1}{2} \\ -f(x-\delta) & \text{with probability } \frac{1}{2} \end{cases} \tag{10}$$

- So, for small δ $\mathbb{E}[g(x)] \approx \nabla f(x)$
- The multidimensional case is a bit more involved as it relies on sampling from a unit ball
- In this general case we can build $g(x) = \frac{n}{\delta} f(x + \delta u)u$ where u is a vector uniformly drawn from the n-dimensional sphere $\mathbb{S}$
- Our g rv can also be made less noisy by drawing from an ellipsoid rather than a sphere

OGD without a Gradient

- The canonical BCO to FO-OCO reduction is given by the FKM Algorithm which has Regret $\leq \mathcal{O}(T^{\frac{3}{4}})$

Algorithm 9 FKM

Input T, $\mathcal{K} \supset 0$, δ, η
Define $\mathcal{K}_\delta = \{x | \frac{1}{1-\delta} x \in \mathcal{K}\}$ and **set** $x_1 = 0$
for $t = 1, \ldots, T$
  draw $u_t \in \mathbb{S}_1$ uniformly at random and **select** $y_t = x_t + \delta u_t$
  Observe $f_t(y_t)$ and define $g_t = \frac{n}{\delta} f_t(y_t) u_t$
  Update $x_{t+1} = \Pi_{\mathcal{K}_\delta}[x_t - \eta g_t]$
end for

Bandit Linear Optimization

- Finally we explore a special (but relevant) case of BCO, called Bandit Linear Optimization (BLO), where cost functions are linear
- $\implies g$ are not biased anymore
- However OGD like methods still pose some problems
  - Lack of efficiency around the boundary of the decision set
  - Large magnitude of the gradient estimates (compared to the distance from the boundary)
- Fortunately, linear functions allow us to solve all these issues by using self-concordant barriers (SCB)

Bandit Linear Optimization

- SCB are a rather advanced technique in convex optimization, so we won't get into much detail
- Intuitively, SCB are a type of barrier which can generate easy to compute Dikin's Ellipsoids in the convex polytope. SCB are available to linear functions
- Optimal Algorithms like **SCRIBLE** based on SCB can obtain near-optimal regret in BLO contexts

Thanks!