Introduction to online optimization

Yuanzhi Li

Assistant Professor, Carnegie Mellon University

Today

Last lectures

- In the entire course, we have been focusing on how to do optimization in a stationary environment.
- Stationary: we are give a fixed function f, and want to optimize it.

This lecture

- We are going to ask the following fundamental question: What if the function f is actually changing over time?
- Changing over time: the function f is changing as we make a move (for example when we update the current point using gradient descent).
- What can we say in this case?

This lecture

- This is called online optimization. There are two major categories: Online learning and Reinforcement learning. Today we are going to focus on online learning. The next lecture will be about reinforcement learning.
- We will begin by giving the definition, and then we will see some applications.

Online optimization

- The online optimization is the following iterative game:
- At every iteration t, the environment (or God) picks a function $f_t : \mathcal{D} \to \mathbb{R}$.
- The player chooses a point $x_t \in \mathcal{D}$, without knowing f_t.
- The environment then tells the player some information about f_t at x_t: $f_t(x_t)$ and/or $\nabla f_t(x_t)$ and/or $\nabla^2 f_t(x_t)$ etc.

Online optimization

- Online optimization is a model of making decisions in a non-stationary environment.
- Typical example:
- Playing computer games, the environment changes as the player takes a move. (Reinforcement Learning)
- Pushing news to the users: The news are changing rapidly, probably much faster than the time required to observe a user's feed back. (Online Learning)

Online optimization

- Intuitive difference between online learning and reinforcement learning:
- Typically, in reinforcement learning, the environment is changing with the player's action. We want the action to reinforce the environmental changes.
- In online learning, typically, the environment is changing independent of player's action. (Warning: This is not precise, but it is the easiest way to understand online learning). We want the action to adapt to the environment.

Online optimization

- Main question for online optimization: Can the player minimize the sequence of functions $\{f_t\}_{t=1}^{T}$?
- Meaning that we want the accumulated loss:

$$\sum_{t \in [T]} f_t(x_t)$$

- to be as small as possible.

Online optimization

- Key difficult of online optimization:
- The player has to pick a point x_t before seeing the function f_t.
- In the pushing news example, we have to push the news to the user and then collect the feedbacks (such as whether the user clicks it or not).

Online optimization

- On the other hand, stationary optimization can also be viewed as a special case of online optimization, where $f_t = f$.
- The gradient descent algorithm: Each time the player picks the point x_t, and then see $\nabla f(x_t)$, and then update

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

Online learning

- Today we are going to learn the setting called online learning, where we want to minimize

$$\sum_{t \in [T]} f_t(x_t)$$

- So that it can be as small as

$$\min_{x \in \mathcal{D}} \sum_{t \in [T]} f_t(x)$$

- In other words, although f_t is changing, we only want to do as good as using the best fixed point x.
- This is because in online learning, intuitively, the function f_t's are "independent of" the player's choice x_t. (Warning: not precise!)
- In reinforcement learning, the goal is different.

Online learning

- We define the regret of the player as:

$$R := \frac{1}{T} \sum_{t \in [T]} f_t(x_t) - \min_{x \in \mathcal{D}} \frac{1}{T} \sum_{t \in [T]} f_t(x)$$

- We want $R \to 0$ (or $R < 0$) as $T \to +\infty$.
- Question: Is it doable?

Online learning: Applications

- Recall the regret:

$$R := \frac{1}{T}\sum_{t\in[T]} f_t(x_t) - \min_{x\in\mathcal{D}} \frac{1}{T}\sum_{t\in[T]} f_t(x)$$

- Special case 1: When $f_t = f$ is a convex function, then this is doable using gradient descent.
- Special case 2: Stochastic gradient descent: When f_t is a randomly sampled function such that for every x,

$$\mathbb{E}[f_t(x)] = f(x)$$

- For example, in ERM problem, $f(W) = \frac{1}{N}\sum_{i\in[N]} \ell(h(W, x_i), y_i)$ and each $f_t(W) = \ell(h(W, x_j), y_j)$ for j uniformly sampled from $[N]$.
- This is doable using stochastic gradient descent.

Online learning: Application

- Other applications mainly include making decisions online, for example online portfolio selection (stock market), selling ads, buying lottery tickets etc.
- Here, at each time we are going to make a decision x_t, and later we will observe its pay-off. But at that time, the environment (f_t) may already changed.

Online learning: Algorithm

- How to minimize the regret:

$$R := \frac{1}{T} \sum_{t \in [T]} f_t(x_t) - \min_{x \in \mathcal{D}} \frac{1}{T} \sum_{t \in [T]} f_t(x)$$

- When $f_t = f$ is a convex function, we know how to do it using gradient descent.
- Key theory for today: When f_t's are convex functions, gradient descent (or mirror descent in general) also works!
- Without assuming anything (such as f_t's are related or changing slowly etc).

Online learning: Algorithm

- Gradient descent for online learning:
- At each iteration t, update:

$$x_{t+1} = x_t - \eta \nabla f_t(x_t)$$

- Theorem: when each f_t is convex and $L - Lipschitz$, let $x^* = \operatorname{argmin} \sum_{t \in [T]} f_t(x)$. Using learning rate $\eta = \frac{\|x - x_0\|}{L\sqrt{T}}$, the regret is bounded by:

$$\frac{1}{T} \sum_{t \in [T]} f_t(x_t) - \frac{1}{T} \sum_{t \in [T]} f_t(x^*) \leq \frac{2\|x^* - x_0\|_2 L}{\sqrt{T}}$$

Online learning: Algorithm

- The theorem says that for sufficiently large T,

$$\frac{1}{T}\sum_{t\in[T]} f_t(x_t) - \min_x \frac{1}{T}\sum_{t\in[T]} f_t(x) \propto \frac{1}{\sqrt{T}}$$

- Why this is true? There might be no connection between each f_t.
- Case 1: All f_t are sort of random, then there is no x such that $\sum_{t\in[T]} f_t(x)$ is small. So our baseline is pretty bad.
- Case 2: Most of the f_t has a common minimizer x^*, then since f_t is convex, $-\nabla f_t(x_t)$ is pointing to x^*: The gradient information is useful!

Online learning: Algorithm

- We also have the projected gradient descent for constraint online learning:
- At each iteration t, update:

$$x_{t+1} = \Pi_{\mathcal{D}}(x_t - \eta \nabla f_t(x_t))$$

Online learning: Proof

- Let us directly show the regret bound for the projected gradient descent algorithm:
- Recall we defined the gradient mapping:

$$g(x_t) := \frac{1}{\eta}(x_t - \Pi_{\mathcal{D}}(x_t - \eta \nabla f_t(x_t)))$$

- Recall when f_t is L-Lipschitzness, $\|g(x_t)\|_2 \leq \|\nabla f(x_t)\|_2 \leq L$.
- Recall the three-term Mirror Descent Lemma for projected gradient descent is given by: for every $x \in \mathcal{D}$

$$f_t(x_t) \leq f_t(x) + \frac{1}{2\eta}\left(\|x - x_t\|_2^2 - \|x - x_{t+1}\|_2^2 + 2\eta^2 L^2\right)$$

Online learning: Proof

- Now we have: for every t,

$$f_t(x_t) \le f_t(x) + \frac{1}{2\eta}\left(\|x - x_t\|_2^2 - \|x - x_{t+1}\|_2^2 + 2\eta^2 L^2\right)$$

- Sum them up from $t = 1, 2$ up to T, we have:

$$\sum_{t\in[T]} f_t(x_t) \le \sum_{t\in[T]} f_t(x) + \sum_{t\in[T]} \frac{1}{2\eta}\left(\|x - x_t\|_2^2 - \|x - x_{t+1}\|_2^2 + 2\eta^2 L^2\right)$$

- Which is:

$$\frac{1}{T}\sum_{t\in[T]} f_t(x_t) \le \frac{1}{T}\sum_{t\in[T]} f_t(x) + \frac{1}{2\eta T}\|x - x_0\|_2^2 + \eta L^2$$

- Picking $\eta = \frac{\|x-x_0\|}{L\sqrt{T}}$ we complete the proof.

Online learning: Proof

- The spirit of Mirror Descent: When $f_t(x_t)$ is much larger than $f_t(x)$, then x_t is moving closer to x.
- Thus, $\nabla f_t(x_t)$ is "useful" to find the optimal point x^*, without any particular assumption on f_t (except convexity).

Online learning application: Non-convex optimization

- The spirit of this proof is extremely important, it can be extend to analyzing the gradient descent update for non-convex ERM problem:
- Suppose we want to minimize $f(W) = \frac{1}{N}\sum_{i\in[N]} \ell(h(W, x_i), y_i)$ for a convex loss ℓ and a non-convex model $h(W, x)$ (such as a neural network).
- Suppose we update it using gradient descent:

$$W_{t+1} = W_t - \eta \nabla f(W_t)$$

- Each time, we can define a convex function f_t:

$$f_t(W) = \frac{1}{N}\sum_{i\in[N]} \ell(h(W_t, x_i) + \langle \nabla_W h(W_t, x_i), W - W_t\rangle, y_i)$$

Online learning application: Non-convex optimization

- Each time, we can define a convex function f_t:

$$f_t(W) = \frac{1}{N} \sum_{i \in [N]} \ell(h(W_t, x_i) + \langle \nabla_W h(W_t, x_i), W - W_t \rangle, y_i)$$

- Key observation:

$$f(W_t) = f_t(W_t), \quad \nabla_W f(W_t) = \nabla_W f_t(W_t)$$

- Therefore, we are doing convex online learning using gradient descent:

$$W_{t+1} = W_t - \eta \nabla f_t(W_t)$$

Online learning: Multi-arm bandit

- One famous application is the so called the multi-arm bandit (MAB) problem, we consider the so called "full observation" case:
- There are m machines, each time t, each machine i has a reward $\ell_t(i)$, and we want to "pull" a machine $i_t \in [m]$ (before knowing $\ell_t(i)$) and collect the reward. After that, we see all the rewards at time t.
- We want to maximize

$$\frac{1}{T}\sum_{t\in[T]}\ell_t(i_t) - \max_{i\in[m]}\frac{1}{T}\sum_{t\in[T]}\ell_t(i)$$

- Algorithm: Maintain a distribution p_t over $[m]$, each step
- (1). Sample $i_t \sim p_t$.
- (2). Update: (for $\ell_t = (\ell_t(i))_{i\in[m]}$):

$$p_{t+1} = \Pi_{distributions}(p_t + \eta\nabla\ell_t)$$

- Intuition: We can define $f_t(p) = -\langle \ell_t, p\rangle$.

Online learning: Multi-arm bandit

- Actually, the most efficient algorithm here is to use projected mirror descent (using KL divergence).
- This is "the best distance" to measure difference between distributions.

Online learning: Other algorithms

- We saw the gradient descent algorithm, actually, essentially all the optimization algorithm we have learnt work in the online learning setting.
- These all works: Mirror Descent/Adagrad/Proximal Gradient Descent/Newton's method/Ellipsoid.

Online newton's method

- We also take a special look at the Online Newton's Method, or the so called online BFGS optimization algorithm:
- At each iteration, update using a quasi-newton step:

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

- Where the "inverse Hessian" matrix B_t is given by: for $s_{t+1} := \eta_t B_t \nabla f_t(x_t)$ and

$$y_t := \nabla f_t(x_t) - \nabla f_{t-1}(x_{t-1})$$

- Update for $\rho_t = \langle s_t, y_t \rangle^{-1}$:

$$B_t = (I - \rho_t s_t y_t^\top) B_{t-1} (I - \rho_t y_t s_t^\top) + \rho_t s_t s_t^\top$$

- The intuition of this update is we want to update B_t such that $B_t s_t \approx -y_t$.

Online learning, other extensions

- There are several extensions to the online learning setting.
- One of the extension is that at each time t, we can only get the function value $f_t(x_t)$ as the feedback, instead of $\nabla f_t(x_t)$.
- This can be quite natural in some applications, where the gradient is hard to obtain.

Online learning, other extensions

- Recall: In standard optimization, we can estimate $\nabla f(x)$ using function value: for every v,

$$\langle \nabla f(x), v \rangle = \lim_{t \to 0} \frac{f(x + tv) - f(x)}{t}$$

- We just need d many such v to recover the full gradient.
- However, in online learning, after we query a point x and observe $f_t(x)$, the function has changed to f_{t+1}.
- We might not be able to query d points for the same function f_t.

Online learning, other extensions

- At each time t, we can only get the function value $f_t(x_t)$ as the feedback, instead of $\nabla f_t(x_t)$.
- Key technique: Using stokes formula: for every $r \geq 0$, for v being a randomly sampled unit vector in $\mathbb{R}^d$ ($x \in \mathbb{R}^d$),

$$\mathbb{E}_v\left[\frac{d}{r} f(x + rv)v\right] = \nabla \tilde{f}(x)$$

- Where $\tilde{f}(x) = \int_{\|v\|_2 \leq 1} f(x + rv)dv$
- Thus, $\frac{d}{r} f(x + rv)v$ is a stochastic gradient for function $\tilde{f}(x)$.
- Thus, we can query $x_t + rv$ and update

$$x_{t+1} = x_t - \frac{d}{r} f_t(x_t + rv)v$$

Online learning, other extensions

- Another major extension assumes that f_t is not changing very fast, in the sense that

$$V = \sum_{t=2}^{T} \|f_t - f_{t-1}\|$$

- is bounded by a not so large value.
- The regret can scale with $\frac{\sqrt{V}}{T}$ in some cases.

Online learning, other extensions

- Another major extension want to complete with the best sequence of "slow moving" x_t^*, in the sense that the baseline now becomes:

$$\min_{\{x_t^*\}_{t\in[T]}} \sum_{t\in[T]} f_t(x_t^*)$$

- Such that $\sum_{t=2}^{T} \|x_t^* - x_{t-1}^*\|$ is bounded by a not so large value.
- In this setting, sometimes we can achieve competitive ratio poly(d), in the sense that

$$\left(\sum_{t\in[T]} f_t(x_t) + \sum_{t=2}^{T} \|x_t - x_{t-1}\|\right)$$

$$\leq \text{poly}(d) \times \min_{\{x_t^*\}_{t\in[T]}} \left(\sum_{t\in[T]} f_t(x_t^*) + \sum_{t=2}^{T} \|x_t^* - x_{t-1}^*\|\right)$$