ECE 8803: Online Decision Making in Machine Learning **Fall 2022**

Lecture 10: September 29

Lecturer: Bhuvesh Kumar

10.1. Recap: Online linear optimization

Last lecture, we bridged prediction and decision-making, and introduced the online linear optimization (OLO) paradigm. In this setting, our decision at time step t is given by the vector $\boldsymbol{w}_t \in \mathcal{B}$ (where $\mathcal{B} \subset \mathbb{R}^d$ is the decision set), and we incur a loss given by $\langle \boldsymbol{w}_t, \boldsymbol{\ell}_t \rangle$, where $\boldsymbol{\ell}_t \in \mathbb{R}^d$ denotes the loss vector at time step t. We then define the algorithm's performance as $H_T := \sum_{t=1}^{T} \langle \boldsymbol{w}_t, \boldsymbol{\ell}_t \rangle$. Our objective is to minimize the regret with respect to the best fixed decision in hindsight, which is defined similarly as in the case of sequence prediction:

$$R_T := H_T - L_T^*, \text{ where} \tag{10.1}$$

$$L_T^* := \min_{\boldsymbol{w} \in \mathcal{B}} \sum_{t=1}^{T} \langle \boldsymbol{w}, \boldsymbol{\ell}_t \rangle. \tag{10.2}$$

We also define as shorthand the notation

$$\ell_t(\boldsymbol{w}) := \langle \boldsymbol{w}, \boldsymbol{\ell}_t \rangle \text{ and}$$

$$L_t(\boldsymbol{w}) := \sum_{s=1}^{t} \ell_s(\boldsymbol{w}).$$

10.2. Recap: Follow the Regularized Leader

In the last lecture we looked at the Follow the Regularized Leader algorithm that tries to balance high-risk elements (FTL) and low-risk elements in our decision making by adding a level of *regularization* to Follow the Leader (FTL). Let's recall the definition of FTRL:

Definition 1 *The Follow-the-Regularized-Leader (FTRL) algorithm chooses*

$$\boldsymbol{w}_t := \arg\min_{\boldsymbol{w} \in \mathcal{B}} \left[L_{t-1}(\boldsymbol{w}) + R(\boldsymbol{w}) \right],$$

where $R(\cdot)$ is a regularization function [1].

Using the regret bound for FTL, we showed that for the following regret bound for FTRL:

1. Note that although we use R as on overloaded notation for regret and regularization, whenever we use R in the context of regret for this lecture, we always subscript with time horizon T, i.e. R_T

Lemma 2 *The regret of FTRL on a loss sequence $\{\ell_t(\boldsymbol{w})\}_{t=1}^T$, for any regularization function $R(\cdot)$ is upper bounded by*

$$R_T \leq \sum_{t=1}^{T} \left(\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_{t+1})\right) + R(\boldsymbol{w}^*) - R(\boldsymbol{w}_1).$$

10.3. FTRL using ℓ_2 regularization

Let consider FTRL with ℓ_2 norm of the iterate as the regularizer, i.e. in round T, the Follow-the-Regularized-Leader (FTRL) algorithm chooses

$$\boldsymbol{w}_t := \arg\min_{\boldsymbol{w}\in\mathcal{B}} \left[L_{t-1}(\boldsymbol{w}) + \frac{1}{2\eta}||\boldsymbol{w}||_2^2 \right],$$

where $\eta > 0$ is a **learning rate** parameter.

Notice that η naturally measures the amount of risk we take: a higher value of η leads to less regularization, and more weight placed on the past observations (therefore, higher risk), while a lower value of η leads to more regularization, and less weight placed on the past observations (therefore, lower risk). Two extreme cases are below:

- If $\eta \to \infty$, FTRL reduces to the highest-risk option, FTL.
- If $\eta \to 0$, FTRL reduces to the lowest-risk option of picking the most stable vector $\boldsymbol{w}_t = \mathbf{0}$.

10.3.1 Regret Bound for FTRL using ℓ_2 regularization

We now prove that FTRL, with a learning rate set to $\eta = \Theta\left(\frac{1}{\sqrt{T}}\right)$, achieves the optimal regret guarantee of $\mathcal{O}(\sqrt{T})$ by effectively trading off stability and ability to exploit meaningful information in data. Recall that we define our decision set to be $\mathcal{B} \subseteq \{\boldsymbol{w} : ||\boldsymbol{w}||_2 \leq D\}$ and assume the loss vectors to be bounded, i.e. $||\boldsymbol{\ell}_t||_2 \leq G$

From Lemma 2, we know that

$$R_T \leq \sum_{t=1}^{T} \left(\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_{t+1})\right) + \frac{1}{2\eta}||\boldsymbol{w}^*||_2^2 - \frac{1}{2\eta}||\boldsymbol{w}_1||_2^2 \tag{10.3}$$

$$\leq \sum_{t=1}^{T} \left(\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_{t+1})\right) + \frac{1}{2\eta}D^2 \qquad (||\boldsymbol{w}||_2 \leq D) \tag{10.4}$$

Let's try to bound $\ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_{t+1}$. Recall that the FTRL update is given by

$$\boldsymbol{w}_t := \arg\min_{\boldsymbol{w}\in\mathcal{B}} F_t(\boldsymbol{w}) \text{ where}$$

$$F_t(\boldsymbol{w}) := \left[L_{t-1}(\boldsymbol{w}) + \frac{1}{2\eta}||\boldsymbol{w}||_2^2 \right].$$

Let us ignore the constraint $\boldsymbol{w} \in \mathcal{B}$ for a moment, and examine what the resulting decision vectors would look like. A calculus exercise gives us

$$\nabla_{\boldsymbol{w}} F_t(\boldsymbol{w}) := \left(\sum_{s=1}^{t-1} \boldsymbol{\ell}_s \right) + \frac{1}{\eta}\boldsymbol{w},$$

Also, note that the objective function $F_t(\boldsymbol{w})$ is clearly convex in $\boldsymbol{w}$, and so the *unconstrained* minimum of $F_t(\boldsymbol{w})$, which we denote by $\boldsymbol{z}_t$, is given by

$$\begin{aligned} \boldsymbol{z}_t &= -\eta \left(\sum_{s=1}^{t-1} \boldsymbol{\ell}_s \right) \\ \implies \boldsymbol{z}_{t+1} &= -\eta \left(\sum_{s=1}^{t} \boldsymbol{\ell}_s \right) \\ &= \boldsymbol{z}_t - \eta \boldsymbol{\ell}_t. \end{aligned}$$

Putting these together, we actually get

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta \boldsymbol{\ell}_t, \tag{10.5}$$

which heavily resembles a step of gradient descent in linear optimization! For this reason, the FTRL algorithm is synonymously described as the *online-gradient-descent*[2]

Of course, in general the unconstrained minimum $\boldsymbol{z}_t$ need not lie within the constrained to decision set $\mathcal{B}$. Thus, to get our actual minimum $\boldsymbol{w}_t$, we simply *project* the unconstrained optimum $\boldsymbol{z}_t$ onto the constrained decision set $\mathcal{B}$, i.e $\boldsymbol{w}_t = argmin_{\boldsymbol{w}} ||\boldsymbol{w} - z_t||^2$. Using convexity of the decision set $\mathcal{B}$, we can show that $\arg\min_{\boldsymbol{w} \in \mathcal{B}} ||\boldsymbol{w} - z_t||^2 \subseteq \arg\min_{\boldsymbol{w} \in \mathcal{B}} F_t(\boldsymbol{w})$[3], i.e. the iterate returned by FTRL in round T.

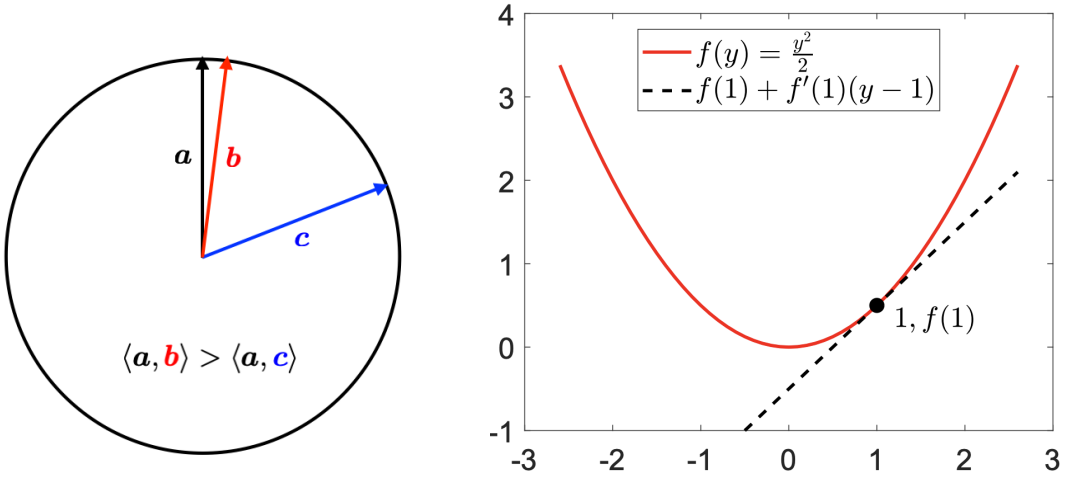
All in all, Equation (10.10) tells us exactly why a small η should lead us to a very small leader change: the decision vector barely moves! To see this concretely, notice that

$$\begin{aligned} \ell_t(\boldsymbol{w}_t) - \ell_t(\boldsymbol{w}_{t+1}) &= \langle \boldsymbol{\ell}_t, \boldsymbol{w}_t - \boldsymbol{w}_{t+1} \rangle \\ &\leq ||\boldsymbol{\ell}_t||_2 ||\boldsymbol{w}_t - \boldsymbol{w}_{t+1}||_2 \\ &\leq ||\boldsymbol{\ell}_t||_2 ||\boldsymbol{z}_t - \boldsymbol{z}_{t+1}||_2 \\ &\leq \eta ||\boldsymbol{\ell}_t||_2^2 \\ &\leq \eta G^2. \end{aligned}$$

Let us unpack these inequalities one by one:

- The first inequality uses the Cauchy-Schwarz inequality on vectors: $\langle \boldsymbol{a}, \boldsymbol{b} \rangle \leq ||\boldsymbol{a}||_2 ||\boldsymbol{b}||_2$ for any two vectors $\boldsymbol{a}, \boldsymbol{b}$. The Cauchy-Schwarz inequality simply says that the inner product between two vectors is maximized when they are exactly aligned; see Figure 10.1a for a pictorial depiction.
- The second inequality uses the fact that projecting into a constrained convex set only *decreases* the distance between two vectors. See Figure **??** for a pictorial depiction of this property and an explanation.
- The third inequality uses the nice form in Equation (10.10), which formally shows us that a smaller learning rate leads to a very small update in the decision due to large regularization.

2. Strictly speaking, FTRL is a *lazy* variant of the online gradient descent algorithm. The more common form of OGD is given by $\boldsymbol{z}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{\ell}_t$, and can also be analyzed via convex analysis techniques.
3. Try proving this statement as an exercise.



(a) Cauchy-Schwarz inequality in 2 dimensions.

(b) First-order approximation to convex function in 1 dimension.

Figure 10.1: Geometric figures that depict the Cauchy-Schwarz inequality in 2 dimensions and the first-order approximation to a convex function in 1 dimension.

- Finally, the fourth inequality uses the upper bound on $||\boldsymbol{\ell}_t||_2$ that we have assumed.

Putting this together with Eq. 10.4, we get

$$R_T \leq T\eta G^2 + \frac{1}{2\eta}D^2 = \mathcal{O}(DG\sqrt{T})$$

where the last inequality is obtained as a consequence of setting $\eta := \frac{D}{G\sqrt{T}}$ and ignoring the extra term ηG^2, (as for this choice of η it will in fact decay with T). This completes our proof.

The ideas from FTRL extend to a remarkably general set of loss functions, well beyond the linear case. In particular, we will show that the same ideas work for any loss function $\ell_t(\boldsymbol{w})$ that is convex in its argument (and well-controlled in some sense). This paradigm is called *online convex optimization* (OCO).

10.4. Convexity Primer

Before getting into the details of OCO, we provide a basic overview of the concepts of convexity. The goal of convex optimization is to minimize a *convex function* over a *convex subset* of the Euclidian space. A set $\mathcal{B}$ is said to be convex if and only if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}$, all the points on the line segment connecting $\boldsymbol{x}$ and $\boldsymbol{y}$ also belong to $\mathcal{B}$, i.e.,

$$\forall \theta \in [0,1], \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in \mathcal{B}.$$

Figure 10.2 shows three examples, where the first one is a convex set, and the other two are

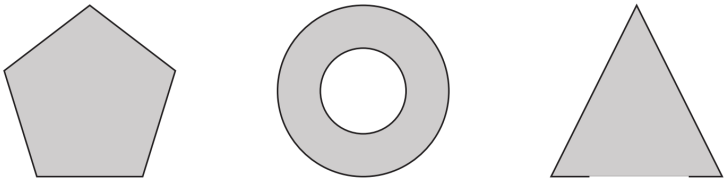


Figure 10.2: Examples of convex and non-convex sets.

non-convex sets, since for each these sets we could always find a line which contains points that are not included in the set.

A function $f : \mathcal{B} \mapsto \mathbb{R}$ is a convex function if and only if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}$,

$$\forall \theta \in [0, 1], f(\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1 - \theta) f(\boldsymbol{y}).$$

This condition is referred to as the *0-th order condition* for convex functions. The following is an example of the 0-th order condition. Initiatively, this condition implies that the line segment connects any two $f(\boldsymbol{x})$ and $f(\boldsymbol{y})$ is always above the function.

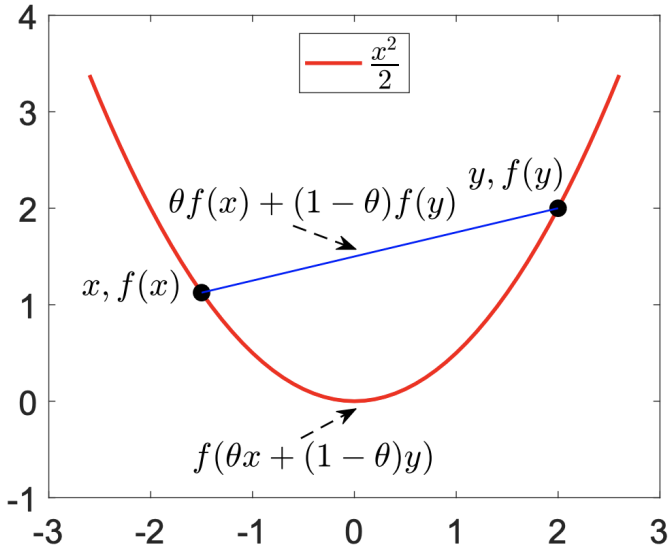


Figure 10.3: Example of the 0-th order condition.

Equivalently, for a differentiable function $f(\boldsymbol{x})$, it is convex if and only if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}$,

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + (\boldsymbol{y} - \boldsymbol{x})^\top \nabla f(\boldsymbol{x}),$$

and it is called the *first-order condition* for convex functions. Figure 10.3 shows an example for the first-order condition. Initiatively, the first-order condition means that, at any point, the function can always be lower bounded by the first-order approximation (the tangent line). Finally, for a twice differentiable function f, it is convex if and only if $\forall \boldsymbol{x} \in \mathcal{B}$,

$$\nabla^2 f(\boldsymbol{x}) \succeq 0,$$

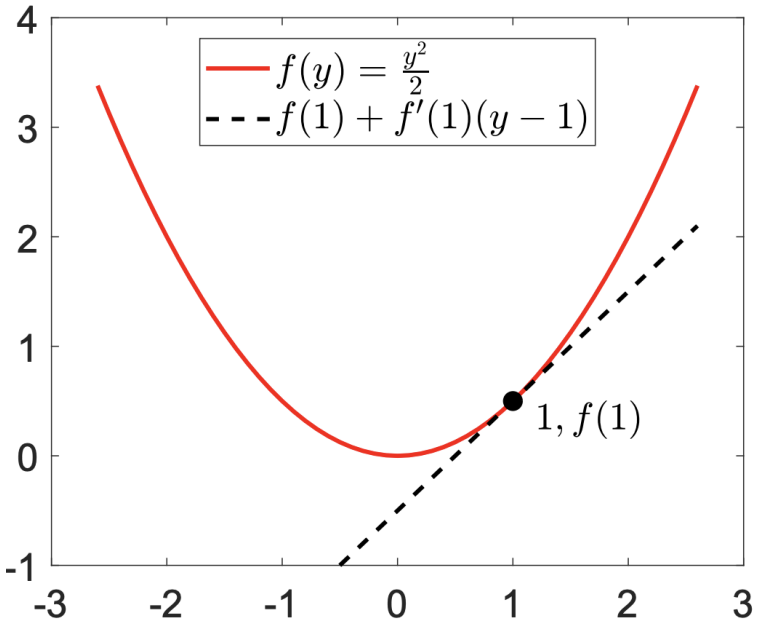


Figure 10.4: Example of the first-order condition.

that is, its Hessian is always positive semi-definite. With the notations defined above, convex optimization can be formally written as

$$\min_{\boldsymbol{x}\in\mathcal{B}} f(\boldsymbol{x}) \tag{10.6}$$

where $f(\cdot)$ is a convex function, and $\mathcal{B}$ is a convex set.

Finally, we introduce some useful function properties other than convexity.

Definition 3 (Lipschitz-continuity) *A function $f : \mathcal{B} \mapsto \mathbb{R}$ is G-Lipschitz continuous if* $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}$,

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq G\|\boldsymbol{x} - \boldsymbol{y}\|_2. \tag{10.7}$$

Note that, when the function is differentiable, (10.7) *is equivalent to* bounded gradients*, i.e.,* $\forall \boldsymbol{x} \in \mathcal{B}$, $\|\nabla f(\boldsymbol{x})\|_2 \leq G$.

10.4.1 Application: Training ML algorithms

The OCO paradigm sees diverse application across engineering problems. One of the most exciting applications of OCO today involves training of machine learning algorithms. In particular, suppose that our decision at time t, $\boldsymbol{w}_t$, denotes the parameters of a machine learning model. In other words, given input $\boldsymbol{x}$, the ML model will give a prediction $\hat{y} = f_{\boldsymbol{w}_t}(\boldsymbol{x})$. Naturally, we want this prediction to be close to the true output y. Accordingly, the loss function at round t is given by its incurred error on a new training example $(\boldsymbol{x}_t, y_t)$, i.e.

$$\ell_t(\boldsymbol{w}) := E(f_{\boldsymbol{w}_t}(\boldsymbol{x}_t), y_t) \tag{10.8}$$

where $E(\hat{y}, y)$ denotes some error function[4] between the prediction and true output. There are several choices of such error functions used in ML. Here are three common examples:

- The squared error function, i.e. $E(\hat{y}, y) = (\hat{y} - y)^2$. This error function is commonly used at both training and test time for regression tasks (although it can also be used for classification tasks).
- The logistic error function, i.e. $E(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$. This is commonly used *at training time* for classification, where the output y is binary but the prediction $\hat{y}$ is real-valued. The intuition is that we want the real-valued prediction not only to match the true output in sign, but also to be large in magnitude to "increase the margin".
- The 0-1 error function, i.e. $E(\hat{y}, y) = \mathbb{I}[\hat{y} \neq y]$. We saw this error function being used in binary sequence prediction. It is commonly used *at test time* to evaluate a classification task (i.e. when the output and prediction are both binary).

Consider the simple example of a linear model, i.e. $f_{\boldsymbol{w}}(\boldsymbol{x}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle$. It turns out that for this example, the loss functions $\ell_t(\boldsymbol{w})$ will be convex in $\boldsymbol{w}$ when we use either the squared error or the logistic error functions on the predictions – because of the linearity of the model and the convexity of the error function on prediction and true output. Thus, for several cases the training procedure in ML would fit into the OCO paradigm. This is a fruitful connection for reasons that we will make clear at the end of the lecture — but we do want to emphasize that the OCO paradigm is significantly more general. This is because we allow for adversarially chosen data, i.e. the training examples $\{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$ can be completely arbitrary, and could be chosen adversarially to our training procedure. This may be a good model for some adverse situations that could arise in ML, such as data poisoning and corruptions in labels.

10.5. From OLO to OCO: FTRL on linearized loss functions

Now, we will show (briefly) that the techniques developed in the previous lecture for OLO extend straightforwardly to OCO. In particular, we introduce the following variant of FTRL on *linearized*[5] versions of the loss functions.

Definition 4 *We modify FTRL to work in the following sequence of steps for convex loss functions. For each $t = 1, \ldots, T$, we perform the following sequence of steps:*

1. *The algorithm selects $\boldsymbol{w}_t$, and incurs loss $\ell_t(\boldsymbol{w}_t)$.*
2. *We define the* first-order *linear approximation to the loss function around $\boldsymbol{w}_t$ as:*

$$\widehat{\ell_t}(\boldsymbol{w}) := \ell_t(\boldsymbol{w}_t) + \langle \nabla \ell_t(\boldsymbol{w}_t), \boldsymbol{w} - \boldsymbol{w}_t \rangle.$$

Recall that Figure 9.3 depicts an example of this first-order approximation in 1 dimension.

4. More commonly called "loss functions" in ML, but we use the terminology error function here to avoid confusion with the loss functions defined for OCO.
5. This trick of linearization is incredibly powerful and is used in several other application domains, most notably control theory.

3. *We select the decision at step $t+1$, denoted by $\boldsymbol{w}_{t+1}$, as FTRL on the linearized loss functions $\{\widehat{\ell}_s(\cdot)\}_{s=1}^t$.*

It turns out that we can show that we continue to have $R_T = \mathcal{O}(DG\sqrt{T})$, where G now represents a *norm* constraint on the gradient of the loss functions, i.e.

$$||\nabla \ell_t(\boldsymbol{w})||_2 \leq G \text{ for all } t \text{ and all } \boldsymbol{w}. \tag{10.9}$$

Equivalently, Equation (10.9) corresponds to the Lipschitz-continuous condition that we defined in Definition 3. This intuitively means that the loss functions need to be *smooth*, i.e. they cannot arbitrarily spike with a change in the decision.

We will now provide a brief proof sketch of this. We will use two important properties of the linearized loss function. It is useful to refer to Figure 10.4 (which is an example in 1-dimension where $w_t = 1$) while verifying these properties for yourself.

- Observe that $\widehat{\ell}_t(\boldsymbol{w}) \leq \ell_t(\boldsymbol{w})$ for any $\boldsymbol{w}$; this is where we use the first-order definition of convexity of the loss functions.
- Also observe that $\widehat{\ell}_t(\boldsymbol{w}_t) = \ell_t(\boldsymbol{w}_t)$; this is simply a consequence of the linearization definition.

We will now use both of these properties to prove our desired regret bound. Note that since FTRL is run on the linearized loss functions, we directly get the regret bound

$$R_{T,\mathsf{lin}} := \sum_{t=1}^T \widehat{\ell}_t(\boldsymbol{w}_t) - \min_{\boldsymbol{w}\in\mathcal{B}} \sum_{t=1}^T \widehat{\ell}_t(\boldsymbol{w}) = \mathcal{O}(DG\sqrt{T}).$$

Here, we have used the fact that the linearized loss vectors $\nabla \ell_t(\boldsymbol{w}_t)$ are assumed to be bounded in norm (i.e. $||\nabla \ell_t(\boldsymbol{w})||_2 \leq G$) as a consequence of the smoothness assumption that we made.

We will now show that $R_T \leq R_{T,\mathsf{lin}}$. Let $\boldsymbol{w}^*$ denote the best decision in hindsight *for the original OCO problem*. Then, we get:

$$\begin{aligned} R_T &= \sum_{t=1}^T \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^T \ell_t(\boldsymbol{w}^*) \\ &\leq \sum_{t=1}^T \ell_t(\boldsymbol{w}_t) - \sum_{t=1}^T \widehat{\ell}_t(\boldsymbol{w}^*) \\ &= \sum_{t=1}^T \widehat{\ell}_t(\boldsymbol{w}_t) - \sum_{t=1}^T \widehat{\ell}_t(\boldsymbol{w}^*) \leq R_{T,\mathsf{lin}}. \end{aligned}$$

Let us unpack these inequalities one by one:

- The first line (equality) simply uses the definition of regret in the original OCO problem.
- The second line (inequality) uses the first property of the linearized loss function that we defined above, i.e. that $\widehat{\ell}_t(\boldsymbol{w}^*) \leq \ell_t(\boldsymbol{w}^*)$ — this directly gives us $-\ell_t(\boldsymbol{w}^*) \leq -\widehat{\ell}_t(\boldsymbol{w}^*)$, which we directly substituted above.

- The third line (equality) uses the second property of the linearized loss function, that $\widehat{\ell}_t(\boldsymbol{w}_t) = \ell_t(\boldsymbol{w}_t)$.

This mini-proof shows us something really remarkable: that the essential ideas that we defined for the special case of the online *linear* optimization problem extend directly to the case of the online *convex* optimization problem! Linearization is indeed a very powerful tool.

10.5.1 The online-gradient descent algorithm

It is instructive to ask what form the FTRL algorithm takes in the OCO setting. We will now see that it explicitly connects to online gradient descent. Recall that we derived the closed-form expression for FTRL on loss vectors $\boldsymbol{\ell}_t$ as:

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta \boldsymbol{\ell}_t,$$

and at every step, $\boldsymbol{w}_t$ is obtained by projecting the unconstrained minimum $\boldsymbol{z}_t$ into the constrained decision set $\mathcal{B}$. In our application of OLO to OCO with linearized losses, we will get

$$\boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta \nabla \ell_t(\boldsymbol{w}_t), \tag{10.10}$$

which is exactly like a step of gradient descent! Thus, FTRL run on linearized losses is synonymously called the *online-gradient-descent* algorithm **(this is the lazy version: HW 3 will have you explore a slightly different version of the online-gradient-descent algorithm).** The only difference between traditional gradient descent and online gradient descent is that traditional gradient descent is run to minimize a *fixed* loss function given by $\ell(\boldsymbol{w})$, while in online gradient descent, the loss function can very completely arbitrarily over time t.

10.6. Additional references

- The proof structure that we have used for OLO is inspired in part by Lecture 12 of the following course: `https://lucatrevisan.github.io/40391/lecture12.pdf`. A much more general version of this statement holds (for arbitrary regularizers and constraints on decisions), and is stated and proved in Theorem 5.2 of Hazan (2016). This is advanced reading (assumes convex analysis background), but worthwhile if you are interested.

- Online convex optimization (OCO) is a significantly more general framework than OLO, as it allows us to consider general convex losses. One of the early applications of OCO was to the portfolio management problem Hazan et al. (2007), that we discussed at a high level last lecture. More recently, the OCO paradigm has been applied to the problem of *model-predictive-control* with unpredictable disturbances in the environment; see Agarwal et al. (2019) for an example of such work.

- Multiplicative weights can actually be written in this FTRL framework! HW 2, Problem 1 explores this formally. Moreover, FTPL with various noise distributions can also be written as various instances of FTRL: see the book chapter `http://dept.stat.lsa.umich.edu/~tewaria/research/abernethy16perturbation.pdf` for more details on this connection.

References

Naman Agarwal, Brian Bullins, Elad Hazan, Sham Kakade, and Karan Singh. Online control with adversarial disturbances. In *International Conference on Machine Learning*, pages 111–119. PMLR, 2019.

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.