THEORETICAL MACHINE LEARNING COS 511

LECTURE #7

MARCH 24, 2015

LECTURER: ELAD HAZAN SCRIBE: KIRAN VODRAHALLI

Contents

1 Last Class and Brief Review

First, a brief note about the final project:

Remark 1.1. Final Project for the Class
Come up with algorithm for some application I will give some suggestions, perhaps a list of papers. The main conference is COLT - papers that appear there, theoretical issues with learning You should give list of ideas on how to continue from that point, if you want can solve your problem.

Or I can also get research questions - it's best if you come up with ideas on your own and try to think how to model it and so on.

An empirical study is fine as well, since no implementation questions so far in the homeworks. Take one of the algorithms that we studied, in some application - portfolio selection, matrix completion, routing, etc. implement a system, see how it works, compare to state of the art. Spend $2-3$ weeks, that's totally sufficient.

The last two weeks we discussed various algorithms for online learning, regret bounding, and so on. We talked about gradient step, and some second order methods - Online Newton Method and Cover's algorithm.

It's sort of a jungle of algorithms, and today we want to answer the question "Is there a way to think of these things in general - how do we minimize regret in general?".

The second thing we want to talk about today is as follows:

Think of minimizing regret for different problems. We talked a bit about shortest path and how to view this as online convex optimization over flow polytope. We'll see a simpler way today of doing that, that will also be more generic. Let us say we want to online learn the best spanning tree of a graph. We have a graph, every iteration there are weights on the edges, we pick a tree, then we have a cost as the sum of the weights on the edges. We want to compete with the best tree in hindsight and minimize regret. This is a hard problem, because it is a discrete set - similar to online shortest path, but there is no polytope here.

In summary, today's topics for lecture will be:

1. Generic treatment of OCO
2. Online learning of Non-convex or Discrete Objects (such as spanning trees)

By the way, if we care about minimizing regret over spanning trees, we have a non-efficient way of doing it. To be more precise, suppose we have $G = (V, E)$. A decision maker picks a spanning tree, and an adversary picks a weight function $W : E \to \{0, 1\}$, and the loss is the weight of the chosen tree (the sum of the weights). We can talk about minimizing regret in this setting. We know how to minimize regret inefficiently: we can treat each tree separately as an expert, and run something like weighted majority. Then $2\text{loss}(\text{tree}_t) - \sum_t \text{loss}(\text{tree}^*) \sim \sqrt{T \lg(N)}$. And then we know that $N \leq |E|^{|V|}$, as a crude upper bound. This algorithm is kind of silly, since there are way too many experts. However, the offline version can be solved efficiently (Kruskal's Algorithm, for instance), and the question is can we solve the online version efficiently?

2 Online learning (OCO)

2.1 Follow the Leader

$K =$ decision set, convex and $\in \mathbb{R}^n$. We iterate: Algorithm picks $x_t \in K$, and the adversary picks a convex loss function f_t. The player suffers $f_t(x_t)$. We want Regret $= \sum_t f_t(x_t) - \min_{x \in K} \sum_t f_t(x) \sim o(T)$.

We've looked at OGD, ONS (though we did not prove anything yet), RWM, Cover – we want to talk about how to view all of these in the generic case. This will be a very intuitive way to view online convex optimization. Here is a suggested algorithm:

Definition 2.1. Follow-the-Leader (FTL), or Fictious Play

$$x_t := \text{argmin}_{x \in K} \{ \sum_{i=1}^{t-1} f_i(x) \} \tag{1}$$

Is this efficient? This can be expressed as offline convex optimization since a sum of convex functions is convex. However, the algorithm does not get good regret.

Claim: Regret(FTL) = $\Omega(T)$. Recall the alternating 2 experts example: Every time the first expert is right, it is wrong the next time. Every time it is wrong, it is right the next time - so we can only do as well as $\frac{T}{2}$. What makes this algorithm bad in general? It switches too frequently. Let us say we force it to not switch too often - can we modify this FTL algorithm to not jump around so much, and be more stable, and get better regret? The answer is yes, this can be done.

2.2 Regularized Follow the Leader

How do we make it not jump too much? Add regularization to the optimization problem, which will force the optimization to be stable. We can stabilize the FTL algorithm by adding regularization. This gives rise to the algorithm RFTL:

Definition 2.2. Regularized Follow-the-Leader (RFTL)

$$x_t := \text{argmin}_{x \in K} \{ \sum_{i=1}^{t-1} f_i(x) + \frac{1}{\eta} R(x) \} \tag{2}$$

$R(x)$ is the noise or stability term, with the hope that this choice results in $|x_t - x_{t+1}|$ being small. We want consecutive decisions to be close to one another.

For the rest of the talk, we will only consider the linear case: f_i's will be linear functions. However this is basically without loss of generality since $f_t(x_t) - f_t(x^*) \leq \nabla f(x_t)(x_t - x^*)$.

So if you like, we can re-write the update step as

$$x_t := \text{argmin}_{x \in K} \{ \sum_{i=1}^{t-1} \nabla_i \cdot (x) + \frac{1}{\eta} R(x) \} \tag{3}$$

noting that $\nabla_i = \nabla f_i(x_i)$. We still did not define η and R. It turns out that under fairly general conditions for R, this algorithm is both efficient and gets good regret.

Theorem 2.3. *For any $R(x) : K \to \mathbb{R}$ such that $R(x)$ is α-strongly-convex and $\eta = \frac{1}{\sqrt{T}}$, then Regret(RFTL) = $\mathcal{O}(\sqrt{T})$.*

But we want to show that the RFTL setup captures many of the algorithms that we have seen. Let us look at some special cases.

Example 2.4. $K = \{x || x| \leq 1\}$, the unit ball, with $R(x) = \frac{1}{2}|x|_2^2$. What does the RFTL algorithm turn out to be? For simplicity, let us first consider what is the result of this optimization for unconstrained $K = \mathbb{R}^n$. We have

$$\begin{aligned} x_t &:= \text{argmin}_{x \in K} \{ \sum_{i=1}^{t-1} \nabla_i \cdot (x) + \frac{1}{\eta} \frac{1}{2} |x|_2^2 \} \\ &= \text{argmin}_{x \in K} \{ \Phi_t(x) \} \end{aligned} \tag{4}$$

Then we solve the optimization problem: $\nabla \Phi_t(x_t) = 0$:

$$\begin{aligned} \sum_{i=1}^{t-1} \nabla_i + \frac{1}{\eta} x_t &= 0 \\ x_t &= -\eta \sum_{i=1}^{t-1} \nabla_i \\ x_{t+1} - x_t &= -\eta \sum_{i=1}^{t} \nabla_i + \eta \sum_{i=1}^{t-1} \nabla_i = -\eta \nabla_t \\ x_{t+1} &= x_t - \eta \nabla_t \end{aligned} \tag{5}$$

so this is equivalent to gradient descent.

Example 2.5. $K = \mathbb{R}^n$, $R(x) = -\sum_i x_i \log(x_i) + \sum_i x_i$, or the negative entropy of x plus a sum. We have this function is convex is you can verify that log is concave.

Then we have that

$$\begin{aligned} \sum_{i=1}^{t-1} \nabla_i + \frac{1}{\eta} \log(x_t) &= 0 \\ x_t &= e^{-\eta \sum_{i=1}^{t-1} \nabla_i} \\ \frac{x_{t+1}(i)}{x_t(i)} &= e^{-\eta \sum_{i=1}^{t} \nabla_i + \eta \sum_{i=1}^{t-1} \nabla_i} = e^{-\eta \nabla_t(i)} \\ x_{t+1} &= x_t e^{-\eta \nabla_t} \end{aligned} \tag{6}$$

which is almost weighted majority. With a bit more complexity, if we set $K = \triangle_n = \{x | \sum_i x_i = 1, x \geq 0\}$. Then

$$x_t(j) = \frac{e^{-\eta \sum_{i=1}^{t-1} \nabla_i(j)}}{\sum_{k=1}^{n} e^{-\eta \sum_{i=1}^{t-1} \nabla_i(k)}} \tag{7}$$

and

$$x_{t+1}(i) = x_t(i) \frac{e^{-\eta l_t(i)}}{\sum_j e^{-\eta l_t(j)}} \tag{8}$$

So it is a distribution. This is exactly weighted majority.

Remark 2.6. This algorithm actually does well for portfolio selection - Schapire and Yoram Singer invented this algorithm, and we get $\mathcal{O}(\sqrt{T \lg(N)})$. This algorithm also has a better dependence on for algorithms on the flow polytope. So it is not necessarily as good as Cover's algorithm and Newton-step, which get $\mathcal{O}(N \lg(T))$ (Newton-step is efficient).

Remark 2.7. On a historical note, this RFTL algorithm was invented in 1957 by Hannan: He talked about randomized regularization, and did not put it in the context of machine learning and online optimization (it was in game theory and statistics, and only rediscovered recently).

Example 2.8. Another interesting application is to matrix completion. Netflix recommends movies to you based on your feedback. The most successful model for it today is as follows: We write the matrix of people as rows, and movies as columns. Then in every cell, put ± 1 based on the feedback of the person, or if you don't know the feedback, leave it 0. This is a very sparse matrix. The state of the art recommendation solution is online convex optimization. Think of your prediction as a matrix. Your convex set K is the set of all convex matrices $K = \{x \in (-1, 1]^{m+n}, |x|_* \leq 10\}$. We assume that the matrix is low-rank because people's preferences may be dependent on each other. The nuclear norm $|x|_*$ is the sum of singular values. We can model $f(x) = |x_{ij} \pm 1|^2$ - if you guessed correctly, it will be 0, otherwise it will be large. We want to minimize this regret. We could use OGD, but this doesn't give the best bounds in theory or in practice. RFTL gives the best bounds for $R(X) = VNE(X) = -\sum_i \sigma_i \lg(\sigma_i)$, where σ_i are the singular values of X. VNE is the Von Neumann entropy (matrix entropy operates on eigenvalues of the matrix). Von Neumann talked about this in the context of quantum theory. So this is another useful regularization function, and there are many others.

Now we would like to prove a general theorem that allows us to get regret bounds for RFTL, since it is more general. We restate the theorem:

Theorem 2.9. *For any $R(x) : K \to \mathbb{R}$ such that $R(x)$ is α-strongly-convex and $\eta = \frac{1}{\sqrt{T}}$, then Regret(RFTL) = $\mathcal{O}(\sqrt{T})$.*

There are two parts to the proof, and generalizes all the proofs we have seen so far. Because it is so general, we need some extra definitions. Here is an overview of the proof.

1. We can bound Regret(RFTL) $\leq \sum_{t=1}^{T} |x_t - x_{t+1}| + \frac{1}{\eta} (R(x_1) - R(x^*))$, where $R(x_1) - R(x^*) \in \mathcal{O}(1)$. This is why it is important that x_t is close to x_{t+1}.

2. We will show that $|x_t - x_{t+1}| \in \mathcal{O}(\eta)$. Then the first term is like stability, and the second term is like noise. We will get something like $\eta T + \frac{1}{\eta}$ when optimized gives $\sim \mathcal{O}(\sqrt{T})$.

Proof. 1. $g_t(x) = \nabla_t \cdot x$; $g_0(x) = \frac{1}{\eta}R(x)$. We let $x_t := \text{argmin}_{x\in K}\{\sum_{i=0}^{t-1} g_i(x)\}$.

Note that we can bound $\sum g_t(x_t) - g_t(x^*)$ since by convexity, $\sum f_t(x_t) - f_t(x^*) \leq \sum g_t(x_t) - g_t(x^*)$.

Lemma 2.10. *Claim* 1
$\forall x \in K$, $\sum_{t=0}^{T} g_t(x_{t+1}) \leq \sum_{t=0}^{T} g_t(x)$

This claim says that if you play the point one iteration in advance, you'll actually have 0 regret. Your total loss will be smaller than any other fixed decision. This is of course unrealistic and assumes knowledge of the next iteration, but it will be useful in the analysis. From this claim, part 1 will follow relatively easily. We have Regret $= \sum_{t=1}^{T} g_t(x_t) - \sum g_t(x^*) \leq \sum g_t(x_t) - g_t(x_{t+1}) + g_0(x_1) - g_0(x^*) = \sum_{t=1}^{T} \nabla_t(x_t - x_{t+1}) + \frac{1}{\eta}[R(x_1) - R(x^*)]$ by the definition of g. We get the additional $g_0(x^*)$ term because of the indices shift. Now we will prove Claim 1.

We do this by induction: Note first that $x_1 := \text{argmin}(g_0(x))$ by definition, so again by definition $g_0(x_1) \leq g_0(x)$ for all $x \in K$. Assume for $t = k$. Prove for $t = k + 1$.

$$\begin{aligned}
\sum_{t=0}^{k+1} g_t(x_{t+1}) &= g_{k+1}(x_{k+2}) + \sum_{t=0}^{k} g_t(x_{t+1}) \\
&\leq g_{k+1}(x_{k+2}) + \sum_{t=1}^{k} g_t(x_{k+2}) \text{which is true by inductive step} \qquad (9) \\
&= \sum_{t=1}^{k+1} g_t(x_{k+1}) \leq \sum_{t=1}^{k+1} g_t(x^*) \forall x^* \in K \text{by definition}
\end{aligned}$$

Therefore we have part 1.

2. We want to show $\nabla_t(x_t - x_{t+1}) = \mathcal{O}(\eta)$. First some facts from linear algebra: we have that $|x|^* = \max_{|y|=1} x \cdot y$. $|x|_2^* = |x|_2$, $|x|_\infty^* = |x|_1$, $|x|_A = \sqrt{x^T A x}$. Also $|x|_{A^*} = |x|_{A^{-1}}$.

   First we have from Taylor approximation $\Phi_t(x_t) = \Phi_t(x_{t+1}) + \nabla\Phi_t(x_{t+1})(x_t - x_{t+1}) + |x_t - x_{t+1}|_\zeta^2$, which looks like $(x_t - x_{t+1})\nabla^2\Phi_t(x_t - x_{t+1})$. Here we use the fact that R has a second derivative. We claim

   $\Phi_t(x_t) \geq \Phi_t(x_{t+1}) + |x_t - x_{t+1}|_\zeta^2$. Recall that this is an optimality condition - x_{t+1} has been chosen optimally, and we have from Karush-Kuhn-Tucker that the gradient term is ≤ 0.

   We can write $|x_t - x_{t+1}|_\zeta^2 \leq \Phi_t(x_t) - \Phi_t(x_{t+1}) = \Phi_{t-1}(x_t) - \Phi_{t-1}(x_{t+i}) + \eta\nabla_t(x_t - x_{t+1}) \leq \eta\nabla_t(x_t - x_{t+1})$ by the optimality of x_t for Φ_{t-1} - the first term must be negative.

   Then by Holder inequality, we have $\leq \eta|\nabla_t|_\zeta^*|x_t - x_{t+1}|_\zeta$. We get $|x_t - x_{t+1}|_\zeta \leq \eta|\nabla_t|_\zeta^* = \mathcal{O}(\eta)$.

   Thus, $\nabla_t(x_t - x_{t+1}) \leq |\nabla_t|_\zeta^* \cdot |x_t - x_{t+1}|_\zeta = \mathcal{O}(\eta)$.

Therefore, Regret $\leq \sum_t \nabla_t(x_t - x_{t+1}) + \frac{1}{\eta}(R(x_1) - R(x_t)) = \mathcal{O}(\eta T_{\frac{1}{\eta}}) = \mathcal{O}(\sqrt{T})$ for $\eta = \frac{1}{\sqrt{T}}$. Note that we don't have specific constants here - it depends on R.

Suppose $R(x) = \frac{1}{2}|x|^2$, so $\nabla^2 = I$. Thus $|\nabla|_{\nabla^2} = |\nabla|_2$, which is just Euclidean norm. You can similarly derive the correct norms for various regularization functions. These are in the notes online (Resource 5).

□

We can derive from this template many tight bounds. Now, we would like to take some particular regularizations to solve the non-convex problem of spanning trees.

3 Online Learning of Non-convex or Discrete Objects

One special case, which is very interesting computationally is randomized regularization.

You can take some random function, effectively noise. It will have the same effect. The whole issue is that online learning works when you have stability (which you get from noise), even though noise hurts you a little bit. With noise, you sometimes gain in terms of computation.

We have $\mathbf{E}[\text{argmin}_x\{\sum_{i=1}^{t-1} \nabla_i \cdot x + \frac{1}{\nabla} n \cdot x\}]$ taken over $n \sim D$. The special case is $f_t(x) = l_t \cdot x$, which is linear loss. We can remove the expectation in this special case because for a linear function, we have $l \cdot \mathbf{E}[x] = \mathbf{E}[l \cdot x]$. If we only care about expected regret, then we can define the following randomized algorithm: Pick some random noise, and then predict. The update is given by

Definition 3.1. Follow the Perturbed Leader (FPL)
$x_t = \text{argmin}_x\{\sum_{i=1}^{t-1} l_i \cdot x + \frac{1}{\eta} n \cdot x\}$, where $n \sim D$.

We will do the general case.

Theorem 3.2. $\boldsymbol{E}[\text{Regret(FPL)}] = \mathcal{O}(\sqrt{T})$

This is an amazing result. For instance, it will allow us to model spanning trees efficiently for this model? We need to define set K and linear loss functions. $K = \{\text{spanning trees on } G(V, E)\}$. Let $x = (10010...) \in R^{|E|}$, where 1 means that edge e is in the tree and 0 means it is not.

Now we can look at loss function $l_t(x) = \sum_{e \in E} W_e \cdot x_e = l_t \cdot x =$ weight of tree x at time t. Remember we have $W : E \to \mathbb{R}$. Note that l_t is a linear function, it is a vector that has all the weights. We want the algorithm to satisfy $\mathbf{E}[\sum_t l_t(x_t) - \sum_t l_t(x^*)] = \mathcal{O}(\sqrt{T})$. We will take $n_i \sim$ uniform$([0, 1])$. FPL says that we want to take $\text{argmin}_{x \in K} L \cdot x$. This is equivalent to minimum spanning tree. Solving this is very efficient, we can do this with Kruskal's and others (Take Tarjan's course for a linear time algorithm).

We can do this approach for any graph problem with linear cost in edges.

The proof by Kalai-Vempola takes the following form:

Proof.

1. $\mathbf{E}[\text{Regret}] \leq \sum_t l_t(x_t - x_{t+1}) + \frac{1}{\eta} n[x_1 - x^*]$. This is exactly what we did before: we proved it by induction.

2. $\mathbf{E}[|x_t - x_{t+1}|] = \mathcal{O}(\eta)$. Now note that $n[x_1 - x^*]$ is $\mathcal{O}(1)$ since we drew from uniform. Now we need to show that $|x_t - x_{t+1}|$ is small. How different are the distributions between l_t and l_{t+1}? We have random noise. $x_t = \text{argmin} L_t \cdot x$, $x_{t+1} = \text{argmin} L_{t+1} \cdot x$. How do we see that the desired quantity is small? Some intuition first: We can imagine the distortion due to the noise as a cube with this $\frac{1}{\eta}$ noise gap $(\sum_{i=1}^{t-1} l(j) \pm [0, 1]\frac{1}{\eta})$. There is another cube around L_{t+1}. When we compute $\mathbf{E}[l_t(x_t - x_{t+1})]$. We compute it over the big cube. The values over the intersections of the two cubes is 0, since the distributions share the same values. Then we have $\mathbf{E}[l_t(x_t - x_{t+1})] = \mathbf{E}[l_t \cdot x_t] - \mathbf{E}[l_t - x_{t+1}]$. These are both integrals, and they share a large portion of the domain. Thus it is 0 over the shared part, and this value is $\leq 1|_{\text{Volume of non-overlap}} \sim \frac{|l_t|}{\frac{1}{\eta}|n|} \sim \eta$, and we have $|x_t - x_{t+1}| \sim \mathcal{O}(\eta)$.

   More rigorously: Let $h_t(n) = \text{argmin}\{(\sum_{i=1}^{t-1} l_i + \frac{1}{\eta} n) \cdot x\}$.

   Then $\mathbf{E}[|x_t - x_{t+1}|] = |\int_{n \sim D} (h_t(n) - h_{t+1}(n))\, dn|$. Now we can do a change of variables. This value is $\leq \int_n |h_t(n) - h_t(n - \eta l_t) dn| = \int_n |h_t(n) dn - h_t(n) d(n - \eta l_t)| \leq |h_t(n)| \int_n |d(n) - d(n - \eta l_t)| dn$. The first term is 1 since we take it over uniform, and the second term is 1 if $n - \eta l_t \in$ cube. So this whole integral is $\leq \rceil \sqcup \dashv$. This is made even more precise in Resource 5 Chapter 5 available online.

□

The point is that this reduces optimization to online learning. We can solve the online version by solving an optimization problem, which works well if we have an efficient optimization oracle, given that we can model the problem appropriately in terms of linear loss functions and so on. It's a very general technique.

4 Summary

Today we learned the following facts:

1. RFTL scheme gives $\mathcal{O}(\sqrt{T})$ regret for OCO.
2. Even for non-convex K, if losses are linear, FPL + efficient optimization oracle (for solving the argmin operation efficiently) to offline problem $\implies$ efficient online algorithm.
3. We have seen so far this fact only partially, but we will see soon in more detail: Efficient online learning (regret minimization) $\implies$ efficient statistical learning according to the model that we started with in the course (hypothesis etc.)

Next time we will focus more on the third fact, and look at what happens when we have only very partial information about the losses. For example, we can model routing with the same scheme, very similar to what we did now. There are efficient algorithms for routing following this scheme. However, we can't necessarily measure the length of specific links in the network, say if you are doing TCP routing. So how do we apply these algorithms with partial feedback. We need to solve an optimization problem which is the minimization: we typically need to know all the losses (for each edge) - we might only get one number. Next time we will show this as well.