AM 221: Advanced Optimization Spring 2016

Prof. Yaron Singer *Lecture 11 — March 3rd*

1 Overview

In this lecture we will introduce the notion of online convex optimization. This is an extremely useful framework which allows us to think about problems where the input appears in an online fashion. In addition, many offline problems can be solved via online convex optimization techniques, as we will see later in the course. We will begin by analyzing algorithms for learning from expert advice, which an important example of the online convex optimization framework. We will then generalize this example and specify the model. Finally, we will introduce and analyze the online gradient descent algorithm.

2 Learning from Experts

Consider the following scenario: for T days, you find yourself having to decide every day between two alternatives A and B. Each day, after you make your decision, one of the alternatives realizes. If on a given day you choose correctly (e.g. if you choose A and A realizes) you are not penalized, but if you choose incorrectly (e.g. choose B when A realizes) you have one mistake counted against you on that given day. To help you decide, there is a set N of n experts who on each day $t = 1, \ldots, T$ recommend either A or B. What is a "good" strategy to implement?

The above problem models typical decision-making scenarios where there are some external sources that can provide useful information (e.g. buying stock), as well as making online predictions. One can imagine a scenario in which input data (say feature vector) comes in an *online* fashion, each expert is some (possibly weak) classifier, and the goal is to make a decision using the help of the n classifiers at our disposal.

2.1 Defining a meaningful benchmark

In order to define what we mean by "good" strategy we need to define a good benchmark that we can compare our strategy against. We will seek algorithms (decision strategies) that are competitive against a *worst-case* adversary. Such an adversary has the power to decide, at each stage, on whether A or B will realize, as well as the experts' advice on every day. A priori, this may seem like too much to ask for, though quite remarkably even under these severe assumptions one can design algorithms with surprisingly good guarantees.[1]

[1] A weaker notion is one in which the realizations of the outcomes everyday are determined by some unknown distribution, but we will not discuss this in this lecture.

One possibility is to seek a strategy whose number of mistakes as a function of T is as small as possible. It turns out that this is not a good benchmark however, since there is a trivial strategy for obtaining an optimal strategy against this benchmark:

- $T/2$ **mistakes is trivial:** By randomly selecting between A and B at every stage, we will make, in expectation, at most $T/2$ mistakes;
- $T/2$ **mistakes is optimal:** It turns out that in the worst case, no randomized algorithm can make fewer than $T/2$ mistakes, in the worst case (see discussion in further reading);

The discussion above implies that counting the total number of mistakes is not a meaningful benchmark. A different approach is to compare ourselves against the single best expert. To do so, let $m_t(i)$ denote the number of mistakes that expert i makes on day t. Our goal would be to minimize *regret* formally defined as the difference between the average number of mistakes we make and the average number of mistakes that the *best* expert makes:

$$\frac{1}{T}\left(\sum_{t=1}^{T} m_t(i_t) - \min_{i \in N} \sum_{t=1}^{T} m_t(i)\right)$$

We will now see a very simple algorithm called the weighted majority (or multiplicative weights update) algorithm which does remarkably well against the best expert.

2.2 The weighted majority algorithm

A simple case. Before presenting the algorithm, let's start with some intuition: suppose there is one expert that makes no mistakes in all T rounds, what would be a good strategy to compete with such an expert? One reasonable approach is to start off by taking a majority vote between all the experts. After the outcome realizes, we can discard all the experts that made a mistake, and continue with this strategy until either exhausting T or the total pool of experts. What is the total number of mistakes we would make in this case? Notice that at every stage that we make a mistake we discard at least half of the experts. This is due to the fact that we took a majority vote, and if we made a mistake it was because at least half of the experts that were not yet discarded at that stage were wrong. Therefore, at the end of this stage we discard all these experts. Thus, since we know there is at least one expert that is never wrong, the total number of mistakes we make cannot exceed $\log n$, where n is the number of experts. If we think of T being very large in comparison to n (we typically think of $T \to \infty$), then the regret of this algorithm converges to zero.

The general case. In general, we do not have a single expert that is always correct, but we can still generalize the algorithm we suggested for the simple case. Notice that we can think of the above algorithm as a *weighted majority algorithm*: the algorithm initially assigns weight of 1 to all experts; at each step the algorithm takes a *weighted majority vote* amongst the experts and uses that to make its decision; after the realization of the outcome, the algorithm penalizes the experts that made a mistake by assigning them a weight of 0 (which is equivalent to discarding them). In general, if we do not have an expert that is always correct, this strategy may discard the best expert at some stage that the expert makes a mistake. A more forgiving approach is to reduce the weight of the experts that make a mistake by a factor of $(1-\epsilon)$ every time they make a mistake, for

some $\epsilon \in (0,1)$. That is, we would assign a weight of $(1-\epsilon)w_t(i)$ to every expert i, where $w_t(i)$ is their current weight at stage t. This is the weighted majority algorithm. We present the algorithm formally below and use $S_t(A), S_t(B)$ to denote the set of experts whose opinion is A and B at stage t, respectively.

Algorithm 1 Weighted Majority Algorithm

```
Initialize: w_1(i) ← 1, ∀i ∈ [n]
for t = 1, ..., T do
    if Σ_{i∈S_t(A)} w_t(i) ≥ Σ_{i∈S_t(B)} w_t(i) then
        choose A, otherwise choose B
    end if
    for i which made a mistake at stage t do
        w_t(i) = (1 − ε)w_t(i)
    end for
end for
```

Analysis of the weighted majority algorithm. The analysis of the above algorithm largely follows from the same principles we applied in the analysis of the simple case.

Theorem. *Let M_T and $M_T(i)$ be the total number of mistakes the algorithm and expert $i \in [n]$ make until step T, respectively. Then, for any $i \in [n]$:*

$$M_T < 2(1+\epsilon)M_T(i) + \frac{\log n}{\epsilon}$$

Proof. We will analyze the algorithm using a *potential function* ϕ_t which simply sums up the total weight of all the experts in a given stage t, i.e. $\phi_t = \sum_{i \in N} w_t(i)$. Notice that by definition $\phi_1 = n$ and that since the weights can only decrease we have that $\phi_t \leq \phi_{t+1}$.

At every stage t that we make a mistake, say we chose A instead B, we have that at least half of the weighted majority selected A. Meaning that $\sum_{i \in S_t(A)} w_t(i) \geq \phi_t/2$. The weight update rule will discount their weights and thus:

$$\phi_{t+1} \leq \sum_{i \in S_t(A)} (1-\epsilon)w_t(i) + \sum_{i \in S_t(B)} w_t(i) \leq \frac{1}{2}\phi_t(i) \cdot (1-\epsilon) + \frac{1}{2}\phi_t(i) = \left(1 - \frac{\epsilon}{2}\right)\phi_t$$

We therefore get that:

$$\phi_T = \phi_1 \left(1 - \frac{\epsilon}{2}\right)^{M_T} = n \cdot \left(1 - \frac{\epsilon}{2}\right)^{M_T} \tag{1}$$

On the other hand we know that:

$$\phi_T = \sum_{i=1}^{n} w_T(i) > w_T(i) = (1-\epsilon)^{M_T(i)} \tag{2}$$

where the last equality is due to the fact that the weight of an expert is discounted by a factor of $(1-\epsilon)$ every time she makes a mistake and at stage $t = 1$ her weight is 1. Putting (1) and (2) together we get that:

$$(1-\epsilon)^{M_T(i)} < n \cdot \left(1 - \frac{\epsilon}{2}\right)^{M_T}$$

Taking logarithms on both sides we get:

$$\log(1-\epsilon)\cdot M_T(i) < \log n + \log\left(1-\frac{\epsilon}{2}\right)\cdot M_T$$

Applying the approximation $-x - x^2 < \log(1-x) < -x$ for all $x \in (0, 1/2)$ (which follows from a Taylor series expansion) we get:

$$-1\cdot(\epsilon+\epsilon^2)\cdot M_T(i) < \log n - \log\left(\frac{\epsilon}{2}\right)\cdot M_T$$

Rearranging, we get the desired result. □

The above result implies that when $T >> n$ the number of mistakes the deterministic algorithm above is approximately twice the number of mistakes the best expert makes. Is this good enough? It turns out that no deterministic algorithm can do substantially better.

Theorem. *Assume the number of mistakes the best expert makes is $L \leq T/2$. Then, in the worst case, no deterministic algorithm can make fewer than $2L$ mistakes.*

Proof. Assume that there are only two experts. The first expert always advises A, while the second expert always advises B. For any given decision the algorithm makes, the adversary chooses the realizations to be the exact opposite (the adversary can do that since the algorithm is deterministic). Therefore, the total number of mistakes the algorithm makes is T. But notice that the best expert here makes no more than $L = T/2$ mistakes since on every day one of them cannot be wrong. □

2.3 Randomized weighted majority

The lower bound stated above holds for *deterministic* algorithms. Can randomized algorithms do better? The randomized weighted majority algorithm is a minor variant of the weighted majority algorithm, where intend of taking a weighted majority vote we select each expert with probability proportional to her weight: an expert i is selected with probability $p_t(i) = w_t(i)/\sum_{j\in[n]} w_t(j)$.

Algorithm 2 Randomized Weighted Majority Algorithm

```
Initialize: w_1(i) ← 1, ∀i ∈ [n]
for t = 1, ..., T do
    Select advice of expert i with probability p_t(i) = w_t(i) / Σ_{j∈[n]} w_t(j)
    for i which made a mistake at stage t do
        w_t(i) = (1 − ε) w_t(i)
    end for
end for
```

Theorem. *Let $m_T, m_T(i)$ denote the total number of mistakes made by the randomized weighted majority algorithm and an expert $i \in [n]$, respectively. Then, for any $i \in [n]$:*

$$\mathbb{E}[m_T] < (1+\epsilon)m_T(i) + \frac{\log n}{\epsilon}.$$

Proof. The proof of the above theorem is similar to that of the analysis of the (deterministic) weighted majority algorithm. The difference lies in the bound on the potential function ϕ_t. Let M_t be the random variable which equals 1 if the algorithm makes a mistake at stage t and 0 otherwise. Similarly, let $M_t(i)$ be the indicator which equals one if expert i makes a mistake at stage t and 0 otherwise. Notice that since $p_t(i) = w_t(i)/\sum_{j\in[n]} w_t(j)$ this implies that $w_t(i) = p_t(i) \cdot \sum_{j\in[n]} w_t(j) = p_t(i) \cdot \phi_t(i)$. We therefore have:

$$\phi_{t+1} = \sum_{i\in[n]} w_t(i)\left(1 - \epsilon \cdot m_t(i)\right) = \phi_t \left(1 - \epsilon \cdot \sum_{i\in[n]} p_t(i) M_t(i)\right) = \phi_t(1 - \epsilon \cdot \mathbb{E}[M_t]) \leq \phi_t \cdot e^{-\epsilon\,\mathbb{E}[M_t]}$$

where the last inequality is due to the fact that $1 + x \leq e^x$. We therefore get:

$$\phi_{t+1} \leq \phi_t \cdot e^{-\epsilon\,\mathbb{E}[m_t]}$$

The rest of the proof is now almost identical to the proof in the deterministic case.

$$(1-\epsilon)^{m_T(i)} = w_T(i) \leq \phi_T \leq n \cdot e^{-\epsilon\,\mathbb{E}[m_T]}$$

Again, taking logarithms on both sides on the inequality and applying the bounds on $\log(1-x)$ as before we get our result. □

3 Online Convex Optimization

The problem of prediction from experts advice falls under a category of a broad and heavily used framework known as *online convex optimization*. The framework is quite general, and can be used to model problems from a broad range of areas.

The online convex optimization model. The online convex optimization model can be defined through the following key elements:

- At every stage $t = 1, \ldots, T$, a decision maker makes a decision $\mathbf{x}_t \in S$, where $S \subseteq \mathbb{R}^n$ is some convex set.
- At every stage $t = 1, \ldots, T$, after the decision maker made her decision, the adversary reveals the *cost function* $f_t : S \to \mathbb{R}$ of the decision at that stage. The cost function is modeled as a convex function, and can change at every stage t.[2]
- The goal of the decision maker is to minimize regret. That is, at every stage the decision maker aims to choose a strategy $\mathbf{x}_t$ s.t. in aggregate over all the stages, the difference between the total cost of the strategies chosen and the single best strategy is minimized. That is, the goal is to minimize:

$$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in S} \sum_{t=1}^{T} f_t(\mathbf{x})$$

[2] In cases where the cost function does not have a closed form, one may assume that after the decision has been made, the adversary enables oracle access to the function. That is, the algorithm can query the oracle to obtain the value of the function and its gradients (if they exist) at any point.

Interestingly, many interesting problems can be modeled as online convex optimization. A few examples include:

- **Learning from expert advice:** In this case the set S is the simplex over the experts: $S = \{\mathbf{x} : \sum_{i\in[n]} x_i = 1\}$. At every stage, the cost function is $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[g_t(\mathbf{x})]$ where $g_t = (g_t(1), \ldots, g_t(n))$ and $g_t(i) = 1$ if expert i, and $\mathcal{D}$ is the distribution chosen by the algorithm.
- **Online spam detection:** In online spam detection, we observe emails $\mathbf{w}_1, \ldots, \mathbf{w}_T$ and need to classify them as `spam` / `not spam` at every period $t \in [T]$. We can model each email $\mathbf{w}_t$ as a bag of words over dictionary of size d: there are d words in our language and each email is a vector over $\{0,1\}^n$ s.t. each index j receives a value of 1 if the word $j \in [d]$ is in the email and 0 otherwise. At every step t, our goal is to create a classifier which is a vector $\mathbf{x}_t \in \mathbb{R}^d$ s.t. $\mathbf{x}_t^\intercal \mathbf{w}_t \geq 0$ if the email is not spam and $\mathbf{x}_t^\intercal \mathbf{w}_t < 0$ if it is spam. At every stage, after classifying the email we observe whether our classifier $\mathbf{x}_t$ made a mistake. One natural cost function is to assign cost 1 on each iteration we make a mistake. Another natural notion is to have a cost function that measures the square loss: $(\hat{y} - y)^2$ where $\hat{y}$ is the prediction made with our classifier at time step t (1 if not spam, -1 if spam) and y is the true label.
- **Online recommendation systems:** Recommendation systems are often modeled matrix completion problems. We assume we have some sparse $0, 1$ matrix, where the rows are the people and the columns are media items. For example, for a service like Netflix, the entry M_{ij} takes value 1 if person i enjoyed (say, rated with at least 4 stars) movie j. In the online settings, at each iteration, a matrix $M_t \subseteq \{0,1\}^{n\times m}$ is revealed, and the adversary chooses a user/movie pair together with the real preference. Our goal is to use existing matrix an make an educated guess that minimizes the square loss.

4 Online Gradient Descent

After defining he notion of online convex optimization we are interested in developing algorithms that can solve such problems. The *online gradient descent* is a modest modification of the gradient descent algorithm we previously defined, where the step sizes η_t are chosen appropriately, and at each step, the next point is being projected back to the feasible set S if the update rule of the gradient descent algorithm happens to land at a point outside the domain S. In the formal description below we use $\prod_S \mathbf{y}^{(t+1)}$ to denote the projection of the point $\mathbf{y}_{(t+1)}$ back to S, i.e. $\prod_S \mathbf{y}_{(t+1)}$ is the point in S whose distance to $\mathbf{y}_{(t+1)}$ is minimal.

Algorithm 3 Online Gradient Descent

1: Initialize guess $\mathbf{x}_1 \in S$
2: **for** $t = 1, \ldots, T$ **do**
3: Choose $\mathbf{x}_t$ and observe $f_t(\mathbf{x}_t)$.
4: $\mathbf{y}_{t+1} \leftarrow \mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t)$
5: $\mathbf{y}_{t+1} \leftarrow \prod_S \mathbf{y}_{t+1}$
6: **end for**

Theorem. *Assume that at every stage, the step size is $\eta_t = D/G\sqrt{t}$ where D is the diameter of the domain, i.e. $\|\mathbf{x} - \mathbf{y}\| \leq D$, $\forall \mathbf{x}, \mathbf{y} \in S$, and G is a bound on the gradient, $\|\nabla f(\mathbf{x})\| \leq G, \forall \mathbf{x} \in S$.*

Then:

$$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}} \sum_{t=1}^{T} f_t(\mathbf{x}) \leq \frac{3}{2} GD\sqrt{T}$$

Proof. Let $\mathbf{x}^\star = \text{argmin}_{\mathbf{x}\in S} \sum_{t=1}^{T} f_t(\mathbf{x})$. By convexity we have that:

$$f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star) \leq \nabla f_t(\mathbf{x}_t)^\intercal (\mathbf{x}_t - \mathbf{x}^\star)$$

From the update rule and that fact that the projection to the set is the smallest distance to $\mathbf{y}_{t+1}$ we have that:

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 = \|\prod_{S}(\mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t)) - \mathbf{x}^\star\|^2 \leq \|\mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t) - \mathbf{x}^\star\|^2$$

From the law of the parallelogram and the bound on $\|\nabla f(\mathbf{x})\| \leq G$ we get that:

$$\begin{aligned}
\|\mathbf{x}_t - \eta_t \nabla f(\mathbf{x}_t) - \mathbf{x}^\star\|^2 &= \|\mathbf{x}_t - \mathbf{x}^\star\|^2 + \eta_t^2 \|\nabla f(\mathbf{x}_t)\|^2 - 2\eta_t \nabla f(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}^\star) \\
&\leq \|\mathbf{x}_t - \mathbf{x}^\star\|^2 + \eta_t^2 G^2 - 2\eta_t \nabla f(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}^\star)
\end{aligned}$$

Putting the above together we get that:

$$\|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2 \leq \|\mathbf{x}_t - \mathbf{x}^\star\|^2 + \eta_t^2 G^2 - 2\eta_t \nabla f(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}^\star)$$

Rearranging we get:

$$2 \cdot \nabla f_t(\mathbf{x}_t)^\intercal (\mathbf{x}_t - \mathbf{x}^\star) \leq \frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} + \eta_t G^2$$

Summing over all the stages and applying the above inequalities we get:

$$\begin{aligned}
2\sum_{t=1}^{T} (f_t(\mathbf{x}_t) - f_t(\mathbf{x}^\star)) &\leq 2\sum_{t=1}^{T} \nabla f_t(\mathbf{x}_t)^\intercal (\mathbf{x}_t - \mathbf{x}^\star) \\
&\leq \sum_{t=1}^{T} \left(\frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} + \eta_t G^2 \right) \\
&= \sum_{t=1}^{T} \left(\frac{\|\mathbf{x}_t - \mathbf{x}^\star\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^\star\|^2}{\eta_t} \right) + G^2 \sum_{t=1}^{T} \eta_t \\
&\leq \sum_{t=1}^{T} \left(\|\mathbf{x}_t - \mathbf{x}^\star\|^2 \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} \right) \right) + G^2 \sum_{t=1}^{T} \eta_t \\
&\leq D^2 \sum_{t=1}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} \right) + G^2 \sum_{t=1}^{T} \eta_t \\
&\leq D^2 \frac{1}{\eta_T} + G^2 \sum_{t=1}^{T} \eta_t \\
&\leq 3DG\sqrt{T}
\end{aligned}$$

where in the last inequality we used the fact that $\eta_t = D/G\sqrt{t}$ and $\sum_{i=1}^{T} 1/\sqrt{t} \leq 2\sqrt{T}$. □

5 Further Reading

This lecture is based on Chapters 1 and 3 from a fantastic short book called *Online Convex Optimization* by Elad Hazan which is available online. A broader (also fantastic) text book on online learning is called *Prediction, Learning, and Games* by Cesa-Bianchi and Lugosi is available online as well.