Khoury College of Computer Sciences — Fall 2023
Northeastern University — 3 November 2023
CS 7870: Seminar in Theoretical Computer Science — Scribe: Matthew Casey

Lecture 17 Outline:

- Introduction to online convex optimization
- Weighted majority algorithm
- Randomized weighted majority algorithm

This lecture introduces the online convex optimization paradigm. We begin with a motivating example that leads us to the definition of the regret framework, a way of analyzing the performance of an online convex optimization algorithm. Then we introduce the learning with expert advice problem. The main focus of the lecture is studying and analysing the weighted majority and randomized weighted majority algorithms for this problem. We finish by proving that the randomized weighted majority algorithm achieves a regret of $O(\sqrt{T \log n})$. The lecture is partially based on [H$^+$16, Chapter 1].

1 Online Computation

Online computation is the study of computing over an online sequence of inputs. This means that there is a notion of time, and inputs come to the algorithm as time passes, and the algorithm must deal with them. Here we assume that the sequence of inputs is finite. Within online computation we may consider two different paradigms: online algorithms and competitive analsysis, and online convex optimization (OCO).

In the competitive analysis framework (which we will study in subsequent lectures), we compare the performance of the online algorithm against the best offline algorithm. This framework has been succesful in certain applications, but sometimes can prove to be too difficult and no good algorithms can be found. The main objective with this paradigm is to study how uncertainty and lack of information affect the quality of the solution.

In the online convex optimization framework we fix some underlying convex set of feasible solutions K. In addition, at each time step t there is some objective function f_t that is unknown to us. Because the objective function is unknown, we cannot simply use a convex optimization solver at each time step t to solve the problem. At each time step t, the flow of the algorithm is as follows:

1. Algorithm selects $x_t \in K$
2. Adversary reveals f_t
3. Algorithm takes on cost $f_t(x_t)$

2 The Regret Framework

In the remainder of the lecture we will focus on online convex optimization. We begin with an example OCO problem that will help us motivate the regret framework, a method of analyzing the performance of OCO algorithms. Consider the following problem, called the portfolio/stock selection problem. Each day we must apportion a fixed amount of wealth to stocks. We then wait a day or two and calculate the return on our investment (equivalent to selling the stocks). We then repeat the process, buying new stocks and waiting. Formally we formulate the problem as follows:

- $K = \{x_1, ...x_n | \sum x_i = 1, x_i \geq 0\}$
- n is the set of investments
- x_i is our allotment of money to stock i, and our total budget is 1
- f_t is the loss on the investment over the time interval $[t, t+1]$

We now ask the question, how can we gauge our performance? We cannot use the competitive analysis framework because we can find a lower bound where we do arbitrarily bad. We construct this lower bound by considering the case where we have only two stocks to choose from, A and B. We then set f_t such that the algorithm makes the wrong choice each time. Suppose that the algorithm picks the sequences (A, B, B, A, B, A, B). Then we set the loss function f_t such that the algorithms loss is $(1, 1, 1, 1, 1, 1, 1)$. Then we consider the optimal solution (B, A, A, B, A, B, A) with corresponding loss $(0, 0, 0, 0, 0, 0, 0)$. So we see that we can set the loss function so that the algorithm always makes the incorrect choice, and thus is arbitrarily bad compared to the optimal solution.

To combat this issue we introduce the regret framework. *Regret* is a quantity which measures the loss of the algorithm compared to the loss acquired by choosing a fixed solution $x \in K$ at every time step.

$$Regret = \sup_{x \in K} \{\sum_t f_t(x_t) - \sum_t f_t(x)\} \tag{1}$$

Now consider the example where again the algorithm picks the sequence (A, B, B, A, B, A, B), but we change the loss function to be $((0, 1), (0, 1), (1, 0), (0, 1), (1, 0), (1, 0), (0, 1))$ where the tuples represent $(f_t(A), f_t(B))$ for each time step. Then we find that the algorithm has loss $0+1+0+0+0+1+1 = 3$, choosing A at each time step has loss $0+0+1+0+1+1+0 = 3$, and choosing B at each time step has loss $1+1+0+1+0+0+1 = 4$. So the algorithm would have a regret of 0 in this example.

3 Learning with Expert Advice

We now define the *learning with expert advice* problem. This problem is similar to the above examples, in that there are two items A and B, from which we have to choose at each time step. We also have that f_t is a function from $\{A, B\}$ to $\{0, 1\}$. Finally, we have a set of n experts $1, ..., n$ that each give a suggestion for which item to take at each time step. We must design an algorithm that chooses an item at each time step by synthesizing the advice from the experts. We evaluate the performance of our algorithm using the regret framework, by comparing the loss of our algorithm to the loss of each of the experts. We start by giving a lower bound on the performance of any deterministic algorithm for this problem.

Theorem 1. Let L be the number of mistakes made by the best expert. Then no deterministic algorithm can make fewer than $2L$ mistakes in the worst case.

Proof. Consider the case where we have two experts A and B where expert A always advises you to choose A, and expert B always advises you to choose B. We set the loss function such that the loss of the item chosen by the algorithm is 1, and the loss of the other item is 0. We then note that the # of mistakes made by alg = (# mistakes of A) + (# mistakes of B). Thus the best expert has at most half the number of mistakes as the algorithm. So the number of mistakes made by the algorithm is at least $2L$. □

If we define T to be the number of time steps, then by extension of the above, we see that the regret of any deterministic algorithm is at least $\frac{T}{2}$ in the case where f_t is set such that the algorithm is always wrong.

Our goal then is to find an algorithm that makes as close to $2L$ mistakes as possible in the worst case. One might first consider the algorithm that at each time step chooses the item that the majority of experts agree on. However, this algorithm turns out to perform very poorly, as we can construct an instance where we have one expert that chooses correctly each time, and $n - 1$ experts that choose incorrectly each time. Then one of the experts would have 0 mistakes while the algorithm would have T mistakes. This problem drives the creation of the weighted majority algorithm.

Algorithm 1: Weighted Majority

```
1 foreach expert $i \in 1, ..., n$ do $w_1(i) \leftarrow 1$
2 for $t = 1$ to $T$ do
3     $W_A = \sum_{\substack{\text{expert } i \text{ s.t.} \\ i \text{ predicts } A}} w_t(i)$
4     $W_B = \sum_{\substack{\text{expert } i \text{ s.t.} \\ i \text{ predicts } B}} w_t(i)$
5     if $W_A \geq W_B$ then
6         Predict $A$
7     else
8         Predict $B$
9     foreach expert $i \in 1, ..., n$ do set $w_{t+1}(i)$
```

So the algorithm works by always selecting the item that has the most "weight" behind it, where the weight of each item (W_A, W_B) is calculated as the sum of the weight of the experts that are suggesting that item. But, it remains for us to define line 9 by giving an update function for the weights of the experts. Our first thought might be to give more weight to experts that answer correctly by using the weight function to count the number of correct answers.

$$w_{t+1}(i) = \begin{cases} w_t(i) + 1 & \text{if } i \text{ was right} \\ w_t(i) & \text{if } i \text{ was wrong} \end{cases}$$

But, this weight update function will not yield a good algorithm. Consider the case where we have $n-1$ experts that are only correct at time steps $1, n-1, 2(n-1), ...$ and the other expert is always correct. Then the $n-1$ experts will always have a greater total weight than the perfect expert, but they make an enormous amounts of errors. So we prefer a weight function that decays the weight exponentially as the expert makes errors. We use equation 2 as the weight update function in line 9. Note that ε is some parameter that we will set ahead of time. We will discuss later how to choose ε.

$$w_{t+1}(i) = \begin{cases} w_t(i) & \text{if } i \text{ was right} \\ (1-\varepsilon)w_t(i) & \text{if } i \text{ was wrong} \end{cases} \tag{2}$$

In addition, we present the randomized majority algorithm, which behaves very similarly to the majority algorithm, but instead of choosing by majority, it selects some expert with a probability and chooses according to that expert.

Algorithm 2: Randomized Weighted Majority

```
1 foreach expert $i \in 1, ..., n$ do $w_1(i) \leftarrow 1$
2 for $t = 1$ to $T$ do
3     Select one of the experts at random, choosing each expert $i$ with probability $\frac{w_t(i)}{\sum_j w_t(j)}$
4     Choose from $A, B$ according to the recommendation of the chosen expert
5     foreach expert $i \in 1, ..., n$ do set $w_{t+1}(i)$
```

Theorem 2. Suppose M_t is the number of mistakes made by an algorithm in steps $1, ..., t$, and that $M_t(i)$ is the number of mistakes made by expert i.

1. For weighted majority: $\forall i, M_T \leq (2+\varepsilon)M_T(i) + \frac{2\log n}{\varepsilon}$
2. For randomized WM: $\forall i, \mathbf{E}[M_T] \leq (1+\varepsilon)M_T(i) + \frac{\log n}{\varepsilon}$

Proof of 1. We use a potential function for this proof defined as $\phi_t = \sum_i w_t(i)$. Next note that $w_t(i) = (1-\varepsilon)^{M_t(i)}$. If the algorithm was correct then $\phi_{t+1} \leq \phi_t$. If not, then $\phi_{t+1} \leq \frac{1}{2}\phi_t + \frac{1}{2}(1-\varepsilon)\phi_t = \phi_t(1-\frac{\varepsilon}{2})$. Thus, $\phi_t \leq n(1-\frac{\varepsilon}{2})^{M_t}$. In addition we have the trivial bound $\phi_t \geq (1-\varepsilon)^{M_t(i)}$. Then we use the inequality $-x - x^2 \leq \ln(1-x) \leq -x$ to finish.

$$\begin{aligned} & & (1-\varepsilon)^{M_T(i)} &\leq n\left(1-\frac{\varepsilon}{2}\right)^{M_T} \\ \Longrightarrow & & M_T(i)\cdot\ln(1-\varepsilon) &\leq \ln n + M_T\cdot\ln\left(1-\frac{\varepsilon}{2}\right) \\ \Longrightarrow & & M_T(i)\cdot(-\varepsilon-\varepsilon^2) &\leq \ln n - \frac{M_T\varepsilon}{2} \\ \Longrightarrow & & M_T &\leq 2(1+\varepsilon)M_T(i) + \frac{2\ln n}{\varepsilon} \end{aligned}$$

□

Proof of 2. We use the same ϕ_t as in the above proof. Let $m_t = 1$ if the algorithm makes a mistake at time t. Lets $m_t(i) = 1$ if expert i makes a mistake at time t. Then we have $\mathbf{E}[m_t] = \sum_i \frac{w_t(i)m_t(i)}{\phi_t}$, this is just the probability of choosing an expert multiplied by whether they were wrong or not. So we have

$$\begin{aligned}\phi_{t+1} &= \sum_i w_t(i)(1 - \varepsilon m_t(i)) \\ &= \phi_t - \varepsilon \sum_i w_t(i)m_t(i) \\ &= \phi_t - \varepsilon\phi_t \mathbf{E}[m_t] \\ &= \phi_t(1 - \varepsilon\mathbf{E}[m_t]) \\ &\leq \phi_t e^{-\varepsilon\mathbf{E}[m_t]}\end{aligned}$$

So we have that $\phi_T \leq ne^{-\varepsilon\mathbf{E}[M_T]}$. In addition we again have the trivial bound $\phi_T \geq w_T(i) = (1-\varepsilon)^{M_T(i)}$. Combining these we get

$$\begin{aligned} & & (1-\varepsilon)^{M_T(i)} &\leq ne^{-\varepsilon\mathbf{E}[M_T]} \\ \implies & & M_T(i)\ln(1-\varepsilon) &\leq \ln n - \varepsilon\mathbf{E}[M_T] \\ \implies & & M_T(i)(-\varepsilon - \varepsilon^2) &\leq \ln n - \varepsilon\mathbf{E}[M_T] \\ \implies & & \mathbf{E}[M_T] &\leq M_T(i)(1+\varepsilon) + \frac{\ln n}{\varepsilon}\end{aligned}$$

□

It remains for us to analyze how to set ε such that we minimize the regret of the algorithm. We will consider only the randomized algorithm, but the other can be done similarly. Note that we can rewrite the final inequality as $\mathbf{E}[M_T] \leq M_T(i) + \varepsilon M_T(i) + \frac{\ln n}{\varepsilon}$. We see that the final two terms vary inversely to each other with ε. One way to set ε is the make these terms equal. We also use the bound that $M_T(i) \leq T$ to replace $M_T(i)$ with T.

$$\begin{aligned}\varepsilon T &= \frac{\log n}{\varepsilon} \\ \implies \varepsilon &= \sqrt{\frac{\log n}{T}}\end{aligned}$$

Finally we conclude that the regret is bounded by $\varepsilon T + \frac{\log n}{\varepsilon} = 2\sqrt{T\log n} = O(\sqrt{T\log n})$.

References

[H+16] Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.