A Course in Graduate Algorithms **Lecture 14**

Online No-Regret Learning

Designed by Prof. Bo Waggoner for the University of Colorado-Boulder *Updated: 2023*

We will consider repeatedly making a decision from a fixed set of options. After each decision, we learn our utility for that decision as well as the ones we chose not to take. Our goal is to learn over time to make good decisions in terms of maximizing total utility, or minimizing total loss. The setting is a bit similar in spirit to online algorithms such as online matching, but we will see that the benchmarks are different.

Objectives:

- Know the setting of online learning.
- Understand the definition of regret and average per-step regret.
- Understand the intuition and details of the Multiplicative Weights (MW) algorithm.
- Be able to adapt MW to specific settings, including how to set the parameters.

1 Online Learning: Making Repeated Decisions

Example: picking a route to school. Each day, you need to decide which route you will take to school. There are N different routes, and depending on traffic and other factors, each is sometimes fast or slow. Each day, after choosing a route, you experience a delay. You also can use the internet to find out what would have happened if you took a different route.

Over time, you track the total delay you've experienced. Meanwhile, you can ask: if you had taken different routes, what would your total delay be? You evaluate your route-choosing algorithm based on how well it performs compared to this hypothetical.

Formal model. We will study a decisionmaking process over rounds $t = 1, \dots, T$. The learner has a fixed finite set of possible actions $i = 1, \dots, N$, the same in each round. Each round t:

1. The algorithm A picks an action i_t.
2. Each action i experiences a loss $\ell_i^t \in [0, 1]$.
3. The algorithm experiences the loss of the action it picked, $\ell_A^t = \ell_{i_t}^t$.

The total loss of action i over all rounds is $L_i = \sum_{t=1}^T \ell_i^t$. The total loss of the algorithm is $L_A = \sum_{t=1}^T \ell_A^t$.

Regret. We want to compare the algorithm's total loss L_A to some benchmark. The benchmark we will consider is what the loss would be if we chose some fixed action i in every round. In particular, we compare to the best fixed action, $\text{OPT} = \min_i L_i$.

Definition 1. The **regret** of the algorithm A is $L_A - \text{OPT}$. The **average per-step regret** of the algorithm is $\frac{1}{T}(L_A - \text{OPT})$.

We say an algorithm **guarantees regret** $f(T)$ if, for all possible sequences of losses, the expected regret of the algorithm (over any internal randomness) is bounded by $f(T)$.

Example revisited. In the daily route example, the actions i are the different routes to school. The delay ℓ_i^t of each route is between zero and one hours. Suppose there are $T = 100$ days. The total delay experience by the algorithm over all rounds is L_A hours, for example, 50 hours, while if it had chosen route i every time, it would have been L_i hours. The benchmark is $\text{OPT} = \min_i L_i$, and let us suppose $\text{OPT} = 30$ hours. The **regret** is $L_A - \text{OPT} = 20$ hours.

Notice that the average per-day delay experienced by the algorithm is $\frac{1}{T} L_A = \frac{1}{2}$ hour or 30 minutes per day, while the average per-day delay of the benchmark is $\frac{1}{T}\text{OPT} = \frac{3}{10}$ hours or 18 minutes. Therefore, the **average per-day regret** is $30 - 18 = 12$ minutes. If we had just taken a certain route i every day, instead of following our algorithm, we could have saved on average 12 minutes per day.

We will see that it is possible to achieve a regret guarantee where the average per-day regret converges to **zero** as $T \to \infty$. Such an algorithm is called "no regret".

Exercise 1. Which of the following regret guarantees would qualify an algorithm as "no regret"?

1. $f(T) = T^{2/3}$
2. $f(T) = 0.03T$
3. $f(T) = \sqrt{T}$

Exercise 2. Is it possible for there to be a sequence of losses and an algorithm where the average per-day regret converges to zero, but the algorithm *never* selects the optimal action?

1.1 The importance of randomization

Our goal is to have low regret on all possible sequences of losses. Because of this, it turns out to be important to pick our actions randomly. If we have a deterministic algorithm, then at a given round, its decision i_t is completely determined by the past history. There is a sequence of losses where $\ell_{i_t}^t = 1$ but $\ell_i^t = 0$ for all actions $i \neq i_t$. On sequences like this, the deterministic algorithm will perform very poorly.

To combat this issue, we will turn to randomization. At each round, the algorithm picks a probability distribution. Meanwhile, the losses ℓ_i^t are determined independently. Then the algorithm draws an action from the distribution. It is evaluated by its expected regret.

2 The Multiplicative Weights Algorithm

The Multiplicative Weights (MW) algorithm and its variants are immensely popular and important in theoretical computer science and machine learning. It picks the action from a distribution at each step. The distribution biases toward actions that have had a lower loss so far. Notice that $\sum_{s=1}^{t-1} \ell_i^s$ is the total loss of actio ni prior to round t (the sum is zero if $t = 1$).

Algorithm 1 Multiplicative Weights

```
Choose parameter $\epsilon \in (0, \frac{1}{2})$.
for $t = 1$ to $T$ do
    Let $w_i^t = \exp\left(-\epsilon \sum_{s=1}^{t-1} \ell_i^s\right)$.
    Let $W^t = \sum_{i=1}^{N} w_i^t$.
    Let the probability of each action $i = 1, \ldots, N$ be $w_i^t / W^t$.
    Choose action $i_t$ from this distribution.
end for
```

The "learning rate" ϵ. The larger ϵ, the more sensitive is the algorithm to a change in loss values. Sensitivity is both good and bad. It allows the algorithm to react and learn quickly, e.g. if a particular action has high losses. However, it can also allow the algorithm to be misled. It may react to noise or temporary changes, which can lead to poor performance. We will next see how to set ϵ for optimal theoretical guarantees.

Exercise 3. What distribution over actions does MW use at round $t = 1$?
Hint: assume that an empty summation evaluates to zero.

Exercise 4. Suppose there are two actions, and consider a sequence with $\ell_1^t = 0$ and $\ell_2^t = 1$ for all rounds t. Will MW perform better on this sequence if ϵ is large or if it is small?

2.1 Regret analysis

First, we will show a regret bound for any fixed choice of $\epsilon \in (0, 1)$. Then, we will pick ϵ to get a "no-regret" guarantee.

Theorem 1. *The Multiplicative Weights algorithm with $\epsilon \in (0, 1)$ guarantees*

$$\text{regret} \le \frac{\epsilon T}{2} + \frac{\ln(N)}{\epsilon}.$$

Proof. Let $i^* = \operatorname{argmin}_i L_i$, the optimal action in hindsight. First, let us lower-bound the total weight after the rounds are over:

$$\begin{aligned} W^{T+1} &\ge w_{i^*}^{T+1} \\ &= \exp\left(-\epsilon L_{i^*}\right). \qquad (1) \end{aligned}$$

Second, let us upper-bound it. We use three facts: $\ell_i^t \in [0, 1]$; for any $x \ge 0$, we have $e^{-x} \le 1 - x + \frac{x^2}{2}$; and our friend $1 + x \le e^x$ for all x. For any t,

$$\begin{aligned} W^{t+1} &= \sum_{i=1}^N w_i^{t+1} \\ &= \sum_{i=1}^N w_i^t \exp\left(-\epsilon \ell_i^t\right) \\ &\le \sum_{i=1}^N w_i^t \left(1 - \epsilon \ell_i^t + \frac{\epsilon^2 (\ell_i^t)^2}{2}\right) \\ &\le \sum_{i=1}^N w_i^t \left(1 - \epsilon \ell_i^t + \frac{\epsilon^2}{2}\right) \\ &= W^t - \left(\epsilon \sum_{i=1}^N w_i^t \ell_i^t\right) + W^t \frac{\epsilon^2}{2} \\ &= W^t - \epsilon W^t \, \mathbb{E}\, \ell_A^t + W^t \frac{\epsilon^2}{2} \\ &= W^t \left(1 - \epsilon \, \mathbb{E}\, \ell_A^t + \frac{\epsilon^2}{2}\right) \\ &\le W^t \exp\left(-\epsilon \, \mathbb{E}\, \ell_A^t + \frac{\epsilon^2}{2}\right). \end{aligned}$$

Iterating this, we get

$$\begin{aligned} W^{T+1} &\leq W^1 \prod_{t=1}^{T} \exp\left(-\epsilon\, \mathbb{E}\, \ell_A^t + \frac{\epsilon^2}{2}\right) \\ &= N \exp\left(-\epsilon\, \mathbb{E}\, L_A + \frac{\epsilon^2 T}{2}\right) \\ &= \exp\left(-\epsilon\, \mathbb{E}\, L_A + \frac{\epsilon^2 T}{2} + \ln(N)\right). \end{aligned} \tag{2}$$

Combining Inequalities 1 and 2, we get

$$\begin{aligned} && \exp\left(-\epsilon L_{i^*}\right) &\leq \exp\left(-\epsilon\, \mathbb{E}\, L_A + \frac{\epsilon^2 T}{2} + \ln(N)\right) \\ \Longrightarrow && -\epsilon L_{i^*} &\leq -\epsilon\, \mathbb{E}\, L_A + \frac{\epsilon^2 T}{2} + \ln(N) \\ \Longrightarrow && \mathbb{E}\, L_A - L_{i^*} &\leq \frac{\epsilon T}{2} + \frac{\ln(N)}{\epsilon}. \end{aligned}$$

□

Theorem 2. *The Multiplicative Weights algorithm with* $\epsilon = \sqrt{\frac{2\ln(N)}{T}}$ *guarantees regret at most* $\sqrt{2\ln(N)T}$.

Proof. Follows immediately by plugging this choice of ϵ into Theorem 1. (We can derive this ϵ by minimizing the bound of Theorem 1, e.g. by taking the derivative.) □

In particular, for a fixed number of actions N, this regret guarantee is $O(\sqrt{T})$, which makes the average per-step regret guarantee $O(1/\sqrt{T})$. For large T, this is almost zero – "no regret".

3 Applications and variations

3.1 Variations and names

The Multiplicative Weights algorithm is sometimes also called "Exponential Weights". Another algorithm that is also called Multiplicative Weights, or sometimes called "Polynomial Weights", is the same, except for the update rule:

$$w_i^{t+1} = w_i^t(1 - \epsilon \ell_i^t).$$

Remember the useful inequality $1 - \epsilon \ell_i^t \leq \exp(-\epsilon \ell_i^t)$. Furthermore, when ϵ is close to zero and $\ell_i^t \in [0, 1]$, this approximation is quite close. So the two algorithms are almost the same, and both satisfy the same regret guarantee (up to small constant factors).

3.2 Learning from expert advice

An important online learning setting is the following. There are N experts who give you advice each day $t = 1, \ldots, T$. You pick one of the experts and follow their advice. Each expert i's advice obtains a loss $\ell_i^t \in [0, 1]$. You obtain the loss of the expert you followed.

We can cast this problem as online learning and apply Multiplicative Weights, obtaining an average per-round regret bound of $O(1/\sqrt{T})$. In other words, over time, the time-average performance of our algorithm approaches the time-average performance of the best expert.