# CS261: A Second Course in Algorithms
# Lecture #11: Online Learning and the Multiplicative Weights Algorithm*

Tim Roughgarden†

February 9, 2016

## 1 Online Algorithms

This lecture begins the third module of the course (out of four), which is about *online algorithms*. This term was coined in the 1980s and sounds anachronistic there days — it has nothing to do with the Internet, social networks, etc. It refers to computational problems of the following type:

**An Online Problem**

1. The input arrives "one-piece at a time."
2. An algorithm makes an irrevocable decision each time it receives a new piece of the input.

For example, in job scheduling problems, one often thinks of the jobs as arriving online (i.e., one-by-one), with a new job needing to be scheduled on some machine immediately. Or in a graph problem, perhaps the vertices of a graph show up one by one (with whatever edges are incident to previously arriving vertices). Thus the meaning of "one piece at a time" varies with the problem, but it many scenarios it makes perfect sense. While online algorithms don't get any airtime in an introductory course like CS161, many problems in the real world (computational and otherwise) are inherently online problems.

*©2016, Tim Roughgarden.

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

## 2 Online Decision-Making

### 2.1 The Model

Consider a set $A$ of $n \geq 2$ actions and a time horizon $T \geq 1$. We consider the following setup.

**Online Decision-Making**

At each time step $t = 1, 2, \ldots, T$:

a decision-maker picks a probability distribution $\mathbf{p}^t$ over her actions $A$

an adversary picks a reward vector $\mathbf{r}^t : A \rightarrow [-1, 1]$

an action $a^t$ is chosen according to the distribution $\mathbf{p}^t$, and the decision-maker receives reward $r^t(a^t)$

the decision-maker learns $\mathbf{r}^t$, the entire reward vector

An *online decision-making algorithm* specifies for each $t$ the probability distribution $\mathbf{p}^t$, as a function of the reward vectors $\mathbf{r}^1, \ldots, \mathbf{r}^{t-1}$ and realized actions $a^1, \ldots, a^{t-1}$ of the first $t-1$ time steps. An *adversary* for such an algorithm $\mathcal{A}$ specifies for each $t$ the reward vector $\mathbf{r}^t$, as a function of the probability distributions $\mathbf{p}^1, \ldots, \mathbf{p}^t$ used by $\mathcal{A}$ on the first $t$ days and the realized actions $a^1, \ldots, a^{t-1}$ of the first $t-1$ days.

For example, $A$ could represent different investment strategies, different driving routes between home and work, or different strategies in a zero-sum game.

### 2.2 Definitions and Examples

We seek a "good" online decision-making algorithm. But the setup seems a bit unfair, no? The adversary is allowed to choose each reward function $\mathbf{r}^t$ *after* the decision-maker has committed to her probability distribution $\mathbf{p}^t$. With such asymmetry, what kind of guarantee can we hope for? This section gives three examples that establish limitations on what is possible.[1]

The first example shows that there is no hope of achieving reward close to that of the best action sequence in hindsight. This benchmark $\sum_{t=1}^{T} \max_{a \in A} r^t(a)$ is just too strong.

**Example 2.1 (Comparing to the Best Action Sequence)** Suppose $A = \{1, 2\}$ and fix an arbitrary online decision-making algorithm. Each day $t$, the adversary chooses the reward vector $\mathbf{r}^t$ as follows: if the algorithm chooses a distribution $\mathbf{p}^t$ for which the probability on action 1 is at least $\frac{1}{2}$, then $\mathbf{r}^t$ is set to the vector $(-1, 1)$. Otherwise, the adversary sets $\mathbf{r}^t$ equal

[1]In the first half of the course, we always sought algorithms that are always correct (i.e., optimal). In an online setting, where you have to make decisions without knowing the future, we expect to compromise on an algorithm's guarantee.

to $(1,-1)$. This adversary forces the expected reward of the algorithm to be nonpositive, while ensuring that the reward of the best action sequence in hindsight is $T$.

Example 2.1 motivates the following important definitions. Rather than comparing the expected reward of an algorithm to that of the best action *sequence* in hindsight, we compare it to the reward incurred by the best *fixed action* in hindsight. In words, we change our benchmark from $\sum_{t=1}^{T} \max_{a \in A} r^t(a)$ to $\max_{a \in A} \sum_{t=1}^{T} r^t(a)$.

**Definition 2.2 (Regret)** Fix reward vectors $\mathbf{r}^1, \ldots, \mathbf{r}^T$. The *regret* of the action sequence $a^1, \ldots, a^T$ is

$$\underbrace{\max_{a \in A} \sum_{t=1}^{T} r^t(a)}_{\text{best fixed action}} - \underbrace{\sum_{t=1}^{T} r^t(a^t)}_{\text{our algorithm}} . \tag{1}$$

We'd like an online decision-making algorithm that achieves low regret, as close to 0 as possible (and negative regret would be even better).[2] Notice that the worst-possible regret in $2T$ (since rewards lie in $[-1,1]$). We think of regret $\Omega(T)$ as an epic fail for an algorithm.

What is the justification for the benchmark of the best fixed action in hindsight? First, simple and natural learning algorithms can compete with this benchmark. Second, achieving this is non-trivial: as the following examples make clear, some ingenuity is required. Third, competing with this benchmark is already sufficient to obtain many interesting applications (see end of this lecture and all of next lecture).

One natural online decision-making algorithm is *follow-the-leader* , which at time step $t$ chooses the action $a$ with maximum cumulative reward $\sum_{u=1}^{t-1} r^u(a)$ so far. The next example shows that follow-the-leader, and more generally every deterministic algorithm, can have regret that grows linearly with $T$.

**Example 2.3 (Randomization Is Necessary for No Regret)** Fix a deterministic online decision-making algorithm. At each time step $t$, the algorithm commits to a single action $a^t$. The obvious strategy for the adversary is to set the reward of action $a^t$ to 0, and the reward of every other action to 1. Then, the cumulative reward of the algorithm is 0 while the cumulative reward of the best action in hindsight is at least $T(1-\frac{1}{n})$. Even when there are only 2 actions, for arbitrarily large $T$, the worst-case regret of the algorithm is at least $\frac{T}{2}$.

For randomized algorithms, the next example limits the rate at which regret can vanish as the time horizon $T$ grows.

**Example 2.4 ($\sqrt{(\ln n)/T}$ Regret Lower Bound)** Suppose there are $n = 2$ actions, and that we choose each reward vector $\mathbf{r}^t$ independently and equally likely to be $(1,-1)$ or $(-1,1)$. No matter how smart or dumb an online decision-making algorithm is, with respect to this random choice of reward vectors, its expected reward at each time step is exactly 0 and its

[2]Sometimes this goal is referred to as "combining expert advice" — if we think of each action as an "expert," then we want to do as well as the best expert.

expected cumulative reward is thus also 0. The expected cumulative reward of the best fixed action in hindsight is $b\sqrt{T}$, where $b$ is some constant independent of $T$. This follows from the fact that if a fair coin is flipped $T$ times, then the expected number of heads is $\frac{T}{2}$ and the standard deviation is $\frac{1}{2}\sqrt{T}$.

Fix an online decision-making algorithm $\mathcal{A}$. A random choice of reward vectors causes $\mathcal{A}$ to experience expected regret at least $b\sqrt{T}$, where the expectation is over both the random choice of reward vectors and the action realizations. At least one choice of reward vectors induces an adversary that causes $\mathcal{A}$ to have expected regret at least $b\sqrt{T}$, where the expectation is over the action realizations.

A similar argument shows that, with $n$ actions, the expected regret of an online decision-making algorithm cannot grow more slowly than $b\sqrt{T \ln n}$, where $b > 0$ is some constant independent of $n$ and $T$.

## 3 The Multiplicative Weights Algorithm

We now give a simple and natural algorithm with optimal worst-case expected regret, matching the lower bound in Example 2.4 up to constant factors.

**Theorem 3.1** *There is an online decision-making algorithm that, for every adversary, has expected regret at most* $2\sqrt{T \ln n}$.

An immediately corollary is that the number of time steps needed to drive the expected time-averaged regret down to a small constant is only logarithmic in the number of actions.[3]

**Corollary 3.2** *There is an online decision-making algorithm that, for every adversary and* $\epsilon > 0$, *has expected time-averaged regret at most* $\epsilon$ *after at most* $(4 \ln n)/\epsilon^2$ *time steps.*

In our applications in this and next lecture, we will use the guarantee in the form of Corollary 3.2.

The guarantees of Theorem 3.1 and Corollary 3.2 are achieved by the *multiplicative weights (MW)* algorithm.[4] Its design follows two guiding principles.

**No-Regret Algorithm Design Principles**

1. Past performance of actions should guide which action is chosen at each time step, with the probability of choosing an action increasing in its cumulative reward. (Recall from Example 2.3 that we need a randomized algorithm to have any chance.)

[3]Time-averaged regret just means the regret, divided by $T$.

[4]This and closely related algorithms are sometimes called the multiplicative weight update (MWU) algorithm, Polynomial Weights, Hedge, and Randomized Weighted Majority.

2. The probability of choosing a poorly performing action should decrease at an exponential rate.

The first principle is essential for obtaining regret sublinear in $T$, and the second for optimal regret bounds.

The MW algorithm maintains a weight, intuitively a "credibility," for each action. At each time step the algorithm chooses an action with probability proportional to its current weight. The weight of each action evolves over time according to the action's past performance.

**Multiplicative Weights (MW) Algorithm**

```
initialize $w^1(a) = 1$ for every $a \in A$
for each time step $t = 1, 2, \ldots, T$ do
    use the distribution $\mathbf{p}^t := \mathbf{w}^t / \Gamma^t$ over actions, where
      $\Gamma^t = \sum_{a \in A} w^t(a)$ is the sum of the weights
    given the reward vector $\mathbf{r}^t$, for every action $a \in A$ use the formula
      $w^{t+1}(a) = w^t(a) \cdot (1 + \eta r^t(a))$ to update its weight
```

For example, if all rewards are either -1 or 1, then the weight of each action $a$ either goes up by a $1 + \eta$ factor or down by a $1 - \eta$ factor. The parameter $\eta$ lies between 0 and $\frac{1}{2}$, and is chosen at the end of the proof of Theorem 3.1 as a function of $n$ and $T$. For intuition, note that when $\eta$ is close to 0, the distributions $\mathbf{p}^t$ will hew close to the uniform distribution. Thus small values of $\eta$ encourage exploration. Large values of $\eta$ correspond to algorithms in the spirit of follow-the-leader. Thus large values of $\eta$ encourage exploitation, and $\eta$ is a knob for interpolating between these two extremes. The MW algorithm is obviously simple to implement, since the only requirement is to update the weight of each action at each time step.

## 4 Proof of Theorem 3.1

Fix a sequence $\mathbf{r}^1, \ldots, \mathbf{r}^T$ of reward vectors.[5] The challenge is that the two quantities that we care about, the expected reward of the MW algorithm and the reward of the best fixed action, seem to have nothing to do with each other. The fairly inspired idea is to relate both of these quantities to an intermediate quantity, namely the sum $\Gamma^{T+1} = \sum_{a \in A} w^{T+1}(a)$ of the actions' weights at the conclusion of the MW algorithm. Theorem 3.1 then follows from some simple algebra and approximations.

[5] We're glossing over a subtle point, the difference between "adaptive adversaries" (like those defined in Section 2) and "oblivious adversaries" which specify all reward vectors in advance. Because the behavior of the MW algorithm is independent of the realized actions, it turns out that the worst-case adaptive adversary for the algorithm is in fact oblivious.

The first step, and the step which is special to the MW algorithm, shows that the sum of the weights $\Gamma^t$ evolves together with the expected reward earned by the MW algorithm. In detail, denote the expected reward of the MW algorithm at time step $t$ by $\nu^t$, and write

$$\nu^t = \sum_{a \in A} p^t(a) \cdot r^t(a) = \sum_{a \in A} \frac{w^t(a)}{\Gamma^t} \cdot r^t(a). \tag{2}$$

Thus we want to lower bound the sum of the $\nu^t$'s.

To understand $\Gamma^{t+1}$ as a function of $\Gamma^t$ and the expected reward (2), we derive

$$\begin{aligned} \Gamma^{t+1} &= \sum_{a \in A} w^{t+1}(a) \\ &= \sum_{a \in A} w^t(a) \cdot (1 + \eta r^t(a)) \\ &= \Gamma^t (1 + \eta \nu^t). \end{aligned} \tag{3}$$

For convenience, we'll bound from above this quantity, using the fact that $1 + x \le e^x$ for all real-valued $x$.[6] Then we can write

$$\Gamma^{t+1} \le \Gamma^t \cdot e^{\eta \nu^t}$$

for each $t$ and hence

$$\Gamma^{T+1} \le \underbrace{\Gamma^1}_{=n} \prod_{t=1}^{T} e^{\eta \nu^t} = n \cdot e^{\eta \sum_{t=1}^{T} \nu^t}. \tag{4}$$

This expresses a lower bound on the expected reward of the MW algorithm as a relatively simple function of the intermediate quantity $\Gamma^{T+1}$.

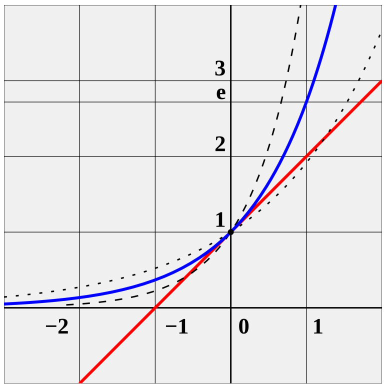

Figure 1: $1 + x \le e^x$ for all real-valued $x$.

[6]See Figure 1 for a proof by picture. A formal proof is easy using convexity, a Taylor expansion, or other methods.

The second step is to show that if there is a good fixed action, then the weight of this action single-handedly shows that the final value $\Gamma^{T+1}$ is pretty big. Combining with the first step, this will imply the the MW algorithm only does poorly if every fixed action is bad.

Formally, let $OPT$ denote the cumulative reward $\sum_{t=1}^{T} r^t(a^*)$ of the best fixed action $a^*$ for the reward vector sequence. Then,

$$\begin{aligned} \Gamma^{T+1} &\geq w^{T+1}(a^*) \\ &= \underbrace{w^1(a^*)}_{=1} \prod_{t=1}^{T} (1 + \eta r^t(a^*)). \end{aligned} \tag{5}$$

$OPT$ is the sum of the $r^t(a^*)$'s, so we'd like to massage the expression above to involve this sum. Products become sums in exponents. So the first idea is to use the same trick as before, replacing $1 + x$ by $e^x$. Unfortunately, we can't have it both ways — before we wanted an upper bound on $1 + x$, whereas now we want a lower bound. But looking at Figure 1, it's clear that the two function are very close to each other for $x$ near 0. This can made precise through the Taylor expansion

$$\ln(1 + x) = x - \tfrac{x^2}{2} + \tfrac{x^3}{3} - \tfrac{x^4}{4} + \cdots .$$

Provided $|x| \leq \frac{1}{2}$, we can obtain a lower bound on $\ln(1 + x)$ by throwing out all terms but the first two, and doubling the second term to compensate. (The magnitudes of the rest of the terms can be bounded above by the geometric series $\frac{x^2}{2}(\frac{1}{2} + \frac{1}{4} + \cdots)$, so the extra $-\frac{x^2}{2}$ term blows them all away.)

Since $\eta \leq \frac{1}{2}$ and $|r^t(a^*)| \leq 1$ for every $t$, we can plug this estimate into (5) to obtain

$$\begin{aligned} \Gamma^{T+1} &\geq \prod_{t=1}^{T} e^{\eta r^t(a^*) - \eta^2 (r^t(a*))^2} \\ &\geq e^{\eta OPT - \eta^2 T}, \end{aligned} \tag{6}$$

where in (6) we're just using the crude estimate $(r^t(a^*))^2 \leq 1$ for all $t$.

Through (4) and (6), we've connected the cumulative expected reward $\sum_{t=1}^{T} \nu^t$ of the MW algorithm with the reward $OPT$ of the best fixed auction through the intermediate quantity $\Gamma^{T+1}$:

$$n \cdot e^{\eta \sum_{t=1}^{T} \nu^t} \geq \Gamma^{T+1} \geq e^{\eta OPT - \eta^2 T}$$

and hence (taking the natural logarithm of both sides and dividing through by $\eta$):

$$\sum_{t=1}^{T} \nu^t \geq OPT - \eta T - \frac{\ln n}{\eta}. \tag{7}$$

Finally, we set the free parameter $\eta$. There are two error terms in (7), the first one corresponding to inaccurate learning (higher for larger learning rates), the second corresponding to overhead before converging (higher for smaller learning rates). To equalize the two terms, we choose $\eta = \sqrt{(\ln n)/T}$. (Or $\eta = \frac{1}{2}$, if this is smaller.) Then, the cumulative expected reward of the MW algorithm is at most $2\sqrt{T \ln n}$ less than the cumulative reward of the best fixed action. This completes the proof of Theorem 3.1.

**Remark 4.1 (Unknown Time Horizons)** The choice of $\eta$ above assumes knowledge of the time horizon $T$. Minor modifications extend the multiplicative weights algorithm and its regret guarantee to the case where $T$ is not known a priori, with the "2" in Theorem 3.1 replaced by a modestly larger constant factor.

## 5 Minimax Revisited

Recall that a two-player zero-sum game can be specified by an $m \times n$ matrix $\mathbf{A}$, where $a_{ij}$ denotes the payoff of the row player and the negative payoff of the column player when row $i$ and column $j$ are chosen. It is easy to see that going first in a zero-sum game can only be worse than going second — in the latter case, a player has the opportunity to adapt to the first player's strategy. Last lecture we derived the minimax theorem from strong LP duality. It states that, provided the players randomize optimally, it makes no difference who goes first.

**Theorem 5.1 (Minimax Theorem)** *For every two-player zero-sum game* $\mathbf{A}$*,*

$$\max_{\mathbf{x}} \left( \min_{\mathbf{y}} \mathbf{x}^\top \mathbf{A} \mathbf{y} \right) = \min_{\mathbf{y}} \left( \max_{\mathbf{x}} \mathbf{x}^\top \mathbf{A} \mathbf{y} \right). \tag{8}$$

We next sketch an argument for deriving Theorem 5.1 directly from the guarantee provided by the multiplicative weights algorithm (Theorem 3.1). Exercise Set #6 asks you to provide the details.

Fix a zero-sum game $\mathbf{A}$ with payoffs in $[-1, 1]$ and a value for a parameter $\epsilon > 0$. Let $n$ denote the number of rows or the number of columns, whichever is larger. Consider the following thought experiment:

- At each time step $t = 1, 2, \ldots, T = \frac{4 \ln n}{\epsilon^2}$:
  - The row and column players each choose a mixed strategy ($\mathbf{p}^t$ and $\mathbf{q}^t$, respectively) using their own copies of the multiplicative weights algorithm (with the action set equal to the rows or columns, as appropriate).
  - The row player feeds the reward vector $\mathbf{r}^t = \mathbf{A}\mathbf{q}^t$ into (its copy of) the multiplicative weights algorithm. (This is just the expected payoff of each row, given that the column player chose the mixed strategy $\mathbf{q}^t$.)
  - Analogously, the column player feeds the reward vector $\mathbf{r}^t = -(\mathbf{p}^t)^T\mathbf{A}$ into the multiplicative weights algorithm.

Let

$$v = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{p}^t)^T\mathbf{A}\mathbf{q}^t$$

denote the time-averaged payoff of the row player. The first claim is that applying Theorem 3.1 (in the form of Corollary 3.2) to the row and column players implies that

$$v \geq \left(\max_{\mathbf{p}} \mathbf{p}^T\mathbf{A}\hat{\mathbf{q}}\right) - \epsilon$$

and

$$v \leq \left(\min_{\mathbf{q}} \hat{\mathbf{p}}^T\mathbf{A}\mathbf{q}\right) + \epsilon,$$

respectively, where $\hat{\mathbf{p}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{p}^t$ and $\hat{\mathbf{q}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{q}^t$ denote the time-averaged row and column strategies.

Given this, a short derivation shows that

$$\max_{\mathbf{p}}\left(\min_{\mathbf{q}} \mathbf{p}^T\mathbf{A}\mathbf{q}\right) \geq \min_{\mathbf{q}}\left(\min_{\mathbf{p}} \mathbf{p}^T\mathbf{A}\mathbf{q}\right) - 2\epsilon.$$

Letting $\epsilon \to 0$ and recalling the easy direction of the minimax theorem ($\max_{\mathbf{p}}\min_{\mathbf{q}}\mathbf{p}^\top\mathbf{A}\mathbf{q} \leq \min_{\mathbf{q}}\max_{\mathbf{p}}\mathbf{p}^\top\mathbf{A}\mathbf{q}$) completes the proof.