CS5275 Lecture 6: Multiplicative Weight Update Algorithms

Jonathan Scarlett

February 22, 2025

Acknowledgment. The first version of these notes was prepared by Chen Jiangwei and Wang Jingtan for a CS6235 assignment.

Useful references:

- Blog post by Jeremy Kun[1]
- A survery on MWU and its applications[2]
- USyd lecture notes: `https://ccanonne.github.io/files/compx270-chap12.pdf`
- For more advanced material, the book "Prediction, Learning, and Games"

Categorization of material:

- Everything not marked as "Optional" is "Core", except the doubling trick which is "Extra".

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

In problems such as optimization and prediction, one encounters techniques like gradient descent for *iteratively improving* some candidate solution until it hopefully ends up with a "nearly correct" or "nearly optimal" solution. In particular, gradient descent is based on *additive updates*, which are particularly natural for optimizing real-valued vectors.

Although seemingly less natural at first, similar ideas based *multiplicative updates* can be extremely powerful.[3] This idea has been discovered and re-discovered several times in a diverse variety of research areas, such as machine learning, theoretical computer science, statistics, game theory, and optimization. It is now a major fundamental building block for both theory and practice in these areas; we will discuss some specific applications near the end of the lecture.

[1] `https://jeremykun.com/2017/02/27/the-reasonable-effectiveness-of-the-multiplicative-weights-update-algorithm/`
[2] `https://theoryofcomputing.org/articles/v008a006/`
[3] In fact, multiplicative updates and gradient descent aren't as distinct as they may initially seem – a technique known as *mirror descent* unifies and generalizes them both (but this is beyond our scope).

1.1 Problem Setting

In this lecture, we will focus on the problem of *prediction with expert advise*, described as follows:

- There are n **experts**: $i \in \{1, 2, ..., n\}$
- There are T **iterations**: $t \in \{1, 2, ..., T\}$
- In each iteration:
  - Each expert provides some "advice" (to be formalized later). For example, in decision-making problems the advice might be "take the following action: [...]" or in prediction problems the advice might be "predict the following value: [...]".
  - The player decides on a piece of advice to follow.
  - Based on which advice was followed, the player receives a **payoff/reward** (for maximization problems) or a **cost/loss** (for minimization problems). In addition, after making the decision, the player receives knowledge of what the rewards or costs *would have been* for all of the experts (including those not selected).
- Goal: We want to *maximize the summation of rewards* or *minimize a summation of losses* over all iterations.
  - Note that there is no fundamental difference between maximization and minimization – we can just define loss as the negative reward ($\ell = -r$), or if we want to keep both in the range $[0, 1]$ (see below) then $\ell = 1 - r$.
  - **Important:** In this topic, we place *no statistical assumptions at all* on the sequences of rewards/losses. Surprisingly, these sequences can be completely arbitrary and we'll still be able to make strong theoretical statements about learning a good strategy.

As a specific example, let us consider the application of stock market predictions. We aim to predict whether the price of one specific stock will increase or decrease each day.

- We have n *experts*, in T *days*.
- The price of the stock each day either increases or decreases.
- Each day, each expert provides advice: Stock increase or stock decrease (for the specific single stock we are considering)
- The player makes a prediction after seeing all the experts' advice.
- The correct answer is revealed after the current day's prediction is made.
- **Goal:** Minimize the number of wrong predictions (maximize the number of correct predictions.)

An *informal description* of the Multiplicative Weights Update (MWU) algorithm is as follows:

- Initialize n **experts** with identical **weights** w_i for each expert i.
- Update experts' weights multiplicatively and iteratively based on how good their advice is:

  - For each expert i in day t, there is a **loss** $m_i^{(t)}$ based on the expert's advice. (e.g., in the above example, we could set $m_i^{(t)} = 0$ for a correct prediction, and $m_i^{(t)} = 1$ for an incorrect one)
  - Reduce the weights of any experts that performed badly (i.e., made bad advice).

- In each iteration, we decide which experts' advice to follow based on the weights.
- **Goal: Achieve a loss comparable to the best expert's performance.**

Regarding the goal, one may be tempted to aim higher – if there are (say) two experts such that one is correct at certain times and the other is correct at certain different times, couldn't we try to combine the "best of both" and be even better than either one individually? While that may sometimes be possible, it is too much to ask in general. For example, if one expert always predicts "this stock's price will increase" and the other expert always predicts "this stock's price will decrease", then trivially one of them will always be correct, but since neither of them is doing anything "intelligent", we can't expect to achieve any reasonable notion of the "best of both". The notion of competing with the best single expert turns out to provide the right balance between being powerful/desirable but also realistic/attainable.

1.2 Warm-up: When a Perfect Expert Exists

We start with a warm-up problem that is limited in scope but relatively easy to solve:

- Suppose that the predictions are binary, i.e., each expert's advice is a predicted value of 0 or 1.
- Assume that there exists a perfect expert, i.e., one that never makes mistakes.

In the setup, we consider the following algorithm:

- Let $S_t \subseteq \{1, \ldots, n\}$ be the set of experts that don't make a mistake in the first $t-1$ iterations.
- In each iteration t, the algorithm makes a binary prediction, either 0 or 1, by taking a *majority vote over the experts in* S_t. This means that we can view experts not in S_t (i.e., those that have made a mistake) as having been *eliminated*.

Claim: The preceding algorithm makes at most $\log n$ mistakes.
(Note: In this lecture, log *means* $\log_2$*, and* ln *means* $\log_e$*.)*
Proof:

- Initially: $|S_1| = n$
- If a mistake is made in iteration t, then $|S_{t+1}| \leq \frac{|S_t|}{2}$; this is because we took a majority vote over S_t, so a mistake means at least half of them were wrong (so they get removed).
- If m mistakes are made in the first T iterations: $|S_{T+1}| \leq \frac{|S_1|}{2^m} = \frac{n}{2^m}$
- Now we use the assumption that there is a perfect expert. By construction, it will never get eliminated, and hence:

$$\begin{aligned} & \frac{n}{2^m} \geq |S_{T+1}| \geq 1 \\ \implies & n \geq 2^m \\ \implies & \log n \geq m. \end{aligned}$$

1.3 Weighted Majority Algorithm

We now present an algorithm that drops the assumption of a perfect expert existing. This is done by using a weight function to capture each expert's past performance, giving the Weighted Majority Algorithm. We again focus on binary predictions (rather than general rewards/losses), and this time we explicitly introduce the notation for such a setup:

- There is an unknown sequence of correct decisions, $d_*^{(1)}, \ldots, d_*^{(T)}$, each in $\{0, 1\}$;
- At time t, each expert i recommends a decision $d_i^{(t)} \in \{0, 1\}$;
- The correct experts in that round are the ones for which $d_i^{(t)} = d_*^{(t)}$.

Algorithm 1 Weighted Majority Algorithm

1: Fix $0 < \eta \leq \frac{1}{2}$.
2: Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
3: **for** $1 \leq t \leq T$ **do**
4: We make a binary prediction $d^{(t)}$, either 0 or 1, by taking a weighted majority vote: if $\sum_{i:d_i^{(t)}=0} w_i^{(t)} \geq \sum_{i:d_i^{(t)}=1} w_i^{(t)}$ then $d^{(t)} \leftarrow 0$; else $d^{(t)} \leftarrow 1$.
5: Update the weights:
- For every expert that predicted wrongly, set $w_i^{(t+1)} = (1 - \eta) w_i^{(t)}$;
- For every expert that predicted correctly, set $w_i^{(t+1)} = w_i^{(t)}$.

We consider the algorithm described in Algorithm 1. Observe that Line 5 is taking a *weighted majority vote*, and Line 7 is down-weighing the experts that made a wrong prediction. The following theorem gives a guarantee on the number of mistakes made.

Theorem 1.1. *Consider the Weighted Majority Algorithm. Let M_i denote the total number of mistakes made by expert i, and let M denote the total number of mistakes made by the algorithm. Then, for any expert i,*

$$M \leq 2(1 + \eta) M_i + \frac{2 \ln n}{\eta}.$$

To interpret this result, it is useful to think of η as a small constant (e.g, 0.1). If we choose i to index the best expert, then the first term captures having roughly **twice** as many mistakes as the best expert. The second term is only $O(\log n)$ and is thus fairly small unless the number of experts is very large.

Proof. Firstly, we define a *potential function:*

$$\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)} \quad (\text{and hence } \Phi^{(1)} = n, \text{ since } w_i^{(1)} = 1 \ \forall i).$$

The idea is to derive both the upper bound and lower bound of the potential function. The bounds will be expressed in terms of M and M_i. Then comparing the two bounds will give the desired inequality involving M and M_i.

To derive an upper bound on $\Phi^{(T+1)}$, consider the case when a mistake is made in iteration t, meaning at least half of the weight is on experts that give wrong advice. Since those experts are the ones down-weighed by $1 - \eta$, it follows that

$$\Phi^{(t+1)} \leq \Phi^{(t)} \Big(\frac{1}{2} + \frac{1}{2}(1 - \eta) \Big) = \Phi^{(t)} \Big(1 - \frac{1}{2} \eta \Big).$$

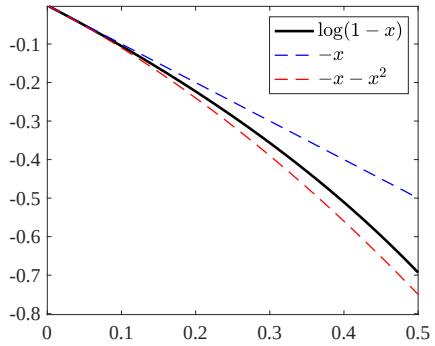

Figure 1: Illustration of bound son $\log(1-x)$.

Stated differently, the first equality holds because half (or more) of the weight decreases by $1-\eta$, and the other half (or less) remains unchanged.

Applying the above inequality inductively, we get

$$\Phi^{(T+1)} \leq \Phi^{(1)}\left(1-\frac{\eta}{2}\right)^M = n \cdot \left(1-\frac{\eta}{2}\right)^M.$$

In addition, we get a trivial lower bound on $\Phi^{(T+1)}$ by just keeping a single term:

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = (1-\eta)^{M_i} w_i^{(1)} = (1-\eta)^{M_i}.$$

Combining the upper and lower bounds, it follows that

$$(1-\eta)^{M_i} \leq n \cdot \left(1-\frac{\eta}{2}\right)^M, \tag{1}$$

or taking the log,

$$M_i \cdot \ln(1-\eta) \leq \ln n + M \cdot \ln\left(1-\frac{\eta}{2}\right).$$

The following inequalities are now useful:

$$-\eta - \eta^2 \leq \ln(1-\eta) \text{ when } 0 < \eta \leq \frac{1}{2} \tag{2}$$

$$\ln\left(1-\frac{\eta}{2}\right) \leq -\frac{\eta}{2} \tag{3}$$

These inequalities can be verified using calculus,[4] but it is easier to see by simply using a picture; see Figure 1. Substituting these inequalities into (1.3), we obtain

$$\begin{aligned} &-M_i \cdot (\eta+\eta^2) \leq M_i \cdot \ln(1-\eta) \leq \ln n + M \cdot \ln\left(1-\frac{\eta}{2}\right) \leq \ln n - M \cdot \frac{\eta}{2} \\ &\implies -M_i \cdot (\eta+\eta^2) \leq \ln n - M \cdot \frac{\eta}{2} \\ &\implies M \leq 2(1+\eta)M_i + \frac{2\ln n}{\eta}. \end{aligned}$$

□

[4]For example, for the first inequality: Let $f(\eta) = \ln(1-\eta)+\eta+\eta^2$, then $f'(\eta) = \frac{1}{\eta-1} + 2\eta + 1 > 0$ when $0 < \eta \leq \frac{1}{2}$. Hence, $f(\eta) \geq f(0) = 0$ when $0 < \eta \leq \frac{1}{2}$. Then, $\ln(1-\eta) \geq -\eta-\eta^2$ when $0 < \eta \leq \frac{1}{2}$.

1.4 Randomized Weighted Majority

The factor of 2 in the above guarantee would ideally be reduced towards 1, but in fact 2 is the best constant possible for deterministic algorithms. For instance, consider a case where there are two experts. Expert 1 always chooses prediction 0 and Expert 2 always chooses prediction 1. If the sequence of correct answers is a "worst-case" one in the sense of making our algorithm perform as poorly as possible, we will make a mistake in all iterations $t \in \{1, \ldots, T\}$. Then, the total number of mistakes made by the algorithm $M = T = M_1 + M_2$, and thus $M \geq 2\min(M_1, M_2)$.

To remove the 2-factor, we may consider randomized algorithms. Intuitively, in examples like the one above, since a fixed strategy might always be wrong, we would actually be better off making completely random decisions, in which case the chance of making mistakes is reduced to 1/2 rather than 1. Accordingly, the randomized weighted majority is similar to the one above, except that it decides each round's prediction using randomization.

Algorithm 2 Randomized Weighted Majority Algorithm

1: Fix $0 < \eta \leq \frac{1}{2}$.
2: Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
3: **for** $1 \leq t \leq T$ **do**
4: We make a binary prediction $d^{(t)}$ randomly, either 0 or 1:
$\Pr[d^{(t)} = 0] = \frac{\sum_{i:d_i^{(t)}=0} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$; $\Pr[d^{(t)} = 1] = \frac{\sum_{i:d_i^{(t)}=1} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$
5: Update the weights:
- For every expert that predicted wrongly, set $w_i^{(t+1)} = (1-\eta) w_i^{(t)}$;
- For every expert that predicted correctly, set $w_i^{(t+1)} = w_i^{(t)}$.

See Algorithm 2, particularly Line 5, which is a form of *randomized weighted majority*. As an example, if the weight of experts guessing 0 is the same as the weight of experts guessing 1, then the algorithm is "completely uncertain" so it just makes a 50/50 guess for its prediction (rather than committing to one of the other). In the multiplicative update rule, we down-weigh the experts whose advice was wrong, so that we pay less attention to them in future rounds.

Theorem 1.2. *Consider the Randomized Weighted Majority Algorithm. If we denote M_i as the total number of mistakes made by expert i, and denote M as the total number of mistakes made by the algorithm, for any expert i,*

$$\mathbb{E}[M] \leq (1+\eta) M_i + \frac{\ln n}{\eta}.$$

This guarantee is similar to the previous one, but with $1+\eta$ instead of $2+\eta$, and with an average $\mathbb{E}[\cdot]$ on the left-hand side since the algorithm is now randomized. Thus, by similar reasoning to before, the randomized weighted majority algorithms can achieve approximately the same number of mistakes as the best expert.

Proof. The *potential function* is defined the same as before: $\Phi^{(t)} = \sum_{i=1}^n w_i^{(t)}$ (and hence $\Phi^{(1)} = n$). Similarly to before, we have the lower bound

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = (1-\eta)^{M_i} w_i^{(1)} = (1-\eta)^{M_i}. \tag{4}$$

The main difference in the proof is in getting the upper bound on $\Phi^{(T+1)}$.

Let $d_*^{(t)}$ be the correct label at time t, and let $f^{(t)} = \frac{\sum_{i:d_i^{(t)} \neq d_*^{(t)}} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$ be the weighted fraction of experts making a mistake in iteration t. Then, the expected number of mistakes made by the algorithm is given by $\mathbb{E}[M] = \sum_{t=1}^T f^{(t)}$ (by Line 6 in the algorithm).

To derive an upper bound of $\Phi^{(T+1)}$, we first express $\Phi^{(t+1)}$ in terms of $\Phi^{(t)}$:

$$\Phi^{(t+1)} = (1 - f^{(t)})\Phi^{(t)} + f^{(t)}(1-\eta)\Phi^{(t)} = \Phi^{(t)}(1 - \eta f^{(t)}) \leq \Phi^{(t)} \cdot e^{-\eta f^{(t)}},$$

where the last step follows from (3). Then, we conduct simple induction to get the upper bound of $\Phi^{(T+1)}$:

$$\Phi^{(T+1)} \leq \Phi^{(1)} e^{-\eta \sum_{t=1}^T f^{(t)}} = n \cdot e^{-\eta \mathbb{E}[M]}$$

Combining this with (4), we have

$$\begin{aligned} (1-\eta)^{M_i} &\leq \Phi^{(T+1)} \leq n \cdot e^{-\eta \mathbb{E}[M]} \\ \implies M_i \ln(1-\eta) &\leq \ln n - \eta \mathbb{E}[M]. \end{aligned}$$

Then, by inequality (2), we have

$$\begin{aligned} -M_i(\eta + \eta^2) \leq M_i \ln(1-\eta) &\leq \ln n - \eta \mathbb{E}[M] \\ \implies \mathbb{E}[M] &\leq (1+\eta) M_i + \frac{\ln n}{\eta}. \end{aligned}$$

□

2 Multiplicative Weights Update (MWU) Algorithm

So far, we have only considered the case of binary prediction, and considered algorithms that multiply the weights by $1-\eta$ whenever a wrong prediction is made. More generally, we might be interested in real-valued prediction, or in settings where choosing an expert incurs some real-valued "loss". (Letting the loss be in $\{0, 1\}$ recovers the earlier setting of "no mistake vs. mistake".)

To handle such scenarios, we consider a general setup in which some real number $m_i^{(t)}$ represents the loss (or "negative reward") when taking the advice of an expert i in iteration t. Handling arbitrarily large losses turns out to be infeasible, so we impose the boundedness assumption $m_i^{(t)} \in [-1, 1]$.

- Note: If the losses are known to be between $[-\rho, \rho]$ for some $\rho > 0$, they can be divided by ρ (i.e., normalized) to become between $[-1, 1]$ (see the tutorial for details), so this assumption is not as restrictive as it may seem.

In more detail, in iteration $t \in \{1, 2, \ldots, T\}$, the following happens:

- The algorithm selects a probability distribution $\boldsymbol{p}^{(t)}$ over the experts:
  $\boldsymbol{p}^{(t)} = (p_1^{(t)}, \ldots, p_n^{(t)})$, where $p_i^{(t)} = \Pr[\text{expert i's advice is followed at time } t]$. Since this is a probability distribution, we have $\sum_{i=1}^n p_i^{(t)} = 1$
- After $\boldsymbol{p}^{(t)}$ is chosen, the losses $m_i^{(t)}$ are revealed.

- The expected loss for the current iteration t is: $\mathbb{E}_{i\sim \boldsymbol{p}^{(t)}}[m_i^{(t)}] = m^{(t)} \cdot \boldsymbol{p}^{(t)} = m_1^{(t)} p_1^{(t)} + m_2^{(t)} p_2^{(t)} + \ldots + m_n^{(t)} p_n^{(t)}$, where here and subsequently, $\mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i$ denotes the regular inner proudct (dot product) between two vectors.

As before, we take each $\{m_i^{(t)}\}_{t=1}^T$ to be a completely arbitrary sequence (in particular, these are *not* necessarily produced by a probabilistic process). In fact, we can even handle "adversarial" settings where, for each t, the losses $\{m_i^{(t)}\}_{i=1}^n$ are generated based on the algorithm's choices up to that time, in a manner that may make the problem as challenging as possible.

The MWU algorithm is described in Algorithm 3.

Algorithm 3 Multiplicative Weights Algorithm

1: Fix $0 < \eta \leq \frac{1}{2}$.
2: Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
3: **for** $1 \leq t \leq T$ **do**
4: Let $\Phi^{(t)} = \sum_{i=1}^n w_i^{(t)}$.
5: Select the distribution $\boldsymbol{p}^{(t)} = \left(\frac{w_1^{(t)}}{\Phi^{(t)}}, \frac{w_2^{(t)}}{\Phi^{(t)}}, \ldots, \frac{w_n^{(t)}}{\Phi^{(t)}}\right)$.
6: Update the weight for every expert: $w_i^{(t+1)} = (1 - \eta m_i^{(t)}) w_i^{(t)}$.

Our objective remains to show that the sum of expected losses (over $t = 1, \ldots, T$) is not too much more than the loss of the best expert. Formally, we have the following.

Theorem 2.1. *With $\sum_{t=1}^T m_i^{(t)}$ being the total loss if we always follow expert i and $\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}$ as the expected loss of the algorithm, we have for any expert i that the MWU algorithm satisfies*

$$\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta}.$$

Before giving the proof, we discuss the terms and give a simpler weakened version. The left-hand side represents the average loss incurred by the algorithm, where at time step t we average over the chosen distribution $\boldsymbol{p}^{(t)}$. On the right-hand side, if we choose i to be the best expert, then the first term is simply that expert's total loss. The second term comes from online decision making incurring slight suboptimality; for example, if all $m_i^{(t)}$ are positive then it just amounts to having $(1 + \eta)$ times the smallest loss, which is only a minor increase if η is small. The final term again gives a mild logarithmic dependence on n, though due to its presence we need to avoid η being *too* small.

To get a weakened (but simpler) bound, we can use $\sum_{t=1}^T |m_i^{(t)}| \leq T$, so that the increase over the best expert's loss (known as the *regret*) is at most $T\eta + \frac{\ln n}{\eta}$. A simple differentiation exercise shows that this is minimized when $\eta = \sqrt{\frac{\ln n}{T}}$, giving a regret of at most $2\sqrt{T \ln n}$, i.e.,

$$\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + 2\sqrt{T \ln n}.$$

Note that $\sqrt{T}$ could be significant, but it is *sub-linear*, whereas the leading term $\sum_{t=1}^T m_i^{(t)}$ often grows linearly. If $\sum_{t=1}^T m_i^{(t)}$ only grows slowly due to most/all of the $m_i^{(t)}$ terms being near zero, then the upper bound in the theorem will tend to be significantly better than the weakened bound.

We now proceed with the proof of the theorem.

Proof. Similar to the previous proof, the goal is to calculate the upper and lower bounds of the potential function (again defined via $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$), expressed using the relevant loss terms. By comparing these limits, we can establish the required inequality that includes $\boldsymbol{m}^{(t)}$ and $m_i^{(t)}$.

Upper bound on Φ: We start with an upper bound on the potential function:

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_{i=1}^{n} w_i^{(t+1)} \\
&= \sum_{i=1}^{n} (1 - \eta m_i^{(t)}) w_i^{(t)} \qquad \text{(by the update rule)} \\
&= \sum_{i=1}^{n} (1 - \eta m_i^{(t)}) p_i^{(t)} \Phi^{(t)} \qquad \text{(by the choice of } p_i^{(t)} \text{ in the algorithm)} \\
&= \Phi^{(t)} - \eta \Phi^{(t)} \sum_{i=1}^{n} m_i^{(t)} p_i^{(t)} \qquad \text{(since the } p_i^{(t)} \text{ sum to one)} \\
&= \Phi^{(t)} (1 - \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}) \\
&\leq \Phi^{(t)} e^{-\eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}}. \qquad \text{(since } 1 - a \leq e^{-a}\text{)}
\end{aligned}$$

Applying this inequality recursively, we obtain

$$\Phi^{(T+1)} \leq \Phi^{(1)} \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}} = n \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}}. \tag{5}$$

Lower bound on Φ: Since $\Phi^{(T+1)}$ is a sum of non-negative weights, we can easily get a lower bound by taking any single weight (say indexed by i):

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = \prod_{t=1}^{T} (1 - \eta m_i^{(t)}).$$

Next, we utilize the following inequalities, which can fairly easily derived from the convex nature of exponential functions (but we will skip the proofs):

$$(1 - \eta)^x \leq 1 - \eta x \text{ if } x \in [0, 1]$$

$$(1 + \eta)^{-x} \leq 1 - \eta x \text{ if } x \in [-1, 0].$$

Since $m_i^{(t)} \in [-1, 1]$, for every expert i, these inequalities allow us to weaken the lower bound on $\Phi^{(T+1)}$ as follows:

$$\Phi^{(T+1)} \geq (1 - \eta)^{\sum_{i\,:\,m_i^{(t)} \geq 0} m_i^{(t)}} \cdot (1 + \eta)^{-\sum_{i\,:\,m_i^{(t)} < 0} m_i^{(t)}}. \tag{6}$$

(Note: This equation and the rest of the analysis become simpler if we assume that the losses are in $[0, 1]$ *instead of* $[-1, 1]$*, but we will handle the latter (more general) case.)*

Combining the bounds: By (5) and (6):

$$n \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}} \geq (1 - \eta)^{\sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)}} \cdot (1 + \eta)^{-\sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)}}.$$

Taking the logarithm on both sides, we get

$$\ln n - \sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \geq \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln(1-\eta) - \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta). \tag{7}$$

Then, negating and scaling by $1/\eta$ gives:

$$\begin{aligned}
\sum_{t=1}^{T} \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} &\leq \frac{\ln n}{\eta} - \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln(1-\eta) + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta) \\
&= \frac{\ln n}{\eta} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln \frac{1}{(1-\eta)} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta) \\
&\leq \frac{\ln n}{\eta} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} (\eta + \eta^2) + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} (\eta - \eta^2) \\
&= \frac{\ln n}{\eta} + \frac{1}{\eta} \left(\eta \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} + \eta \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} + \eta^2 \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} - \eta^2 \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \right) \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} - \eta \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}|,
\end{aligned}$$

where the second inequality is due to:

$$\ln \frac{1}{1-\eta} \leq \eta + \eta^2 \text{ when } \eta \in [0, 1/2]$$
$$\ln(1+\eta) \geq \eta - \eta^2 \text{ when } \eta \in [0, 1/2].$$

(The first of these was shown in Figure 1 (after multiplying -1 on both sides), and the second is analogous.) □

We conclude with some discussion:

- There are two hyperparameters in the MWU algorithm, namely, η and T. The number of iterations T may be dictated by the application, or we may be free to choose it "large enough" to get some desired guarantee (e.g., guarantee a loss at most $1+\epsilon$ higher than that of the best expert for some small ϵ). The parameter η is analogous to the "step size" in other optimization algorithms; it could be chosen according to the weakened bound (as we discussed after the theorem statement), could be set to a fixed value like 0.1, or could be tuned based on data (if available).
- After Theorem 2.1, we derived the choice $\eta = \sqrt{\frac{\ln n}{T}}$, but it should be noted that such a choice relies on knowing T. However, the value of T is not always known in advance. One way to circumvent this issue is to use the *doubling trick* at the price of a small constant factor in the regret bound. The idea is to divide T into intervals of known lengths, then we can set η separately within each interval. It can then be shown that summing up the losses at each interval is not much worse than the optimal one. More details can be found in Sec. 8.2 of the book "Foundations of Machine Learning".

- In the assumption of losses being in $[-1, 1]$, the use of the number 1 is simply for convenience. It is straightforward to extend to $[-\rho, \rho]$ for arbitrary $\rho > 0$, since the learner can always choose to divide the rewards by ρ. The resulting guarantee on the *original* reward changes slightly due to this scaling (see the tutorial for details) but remains fundamnetally similar to the case of $[-1, 1]$-valued losses.

2.1 (**Optional**) The Hedge Algorithm

We briefly mention that there is another algorithm called the Hedge algorithm that closely resembles the MWU algorithm that we studied above. It differs in the following aspects:

- Range of η: The Hedge algorithm is less restrictive by only requiring $0 < \eta \leq 1$.
- Weight update scheme: The Hedge algorithm uses an slightly different exponential update.

See Algorithm 4 for the details.

Algorithm 4 The Hedge Algorithm

Fix $0 < \eta \leq 1$.
Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \dots, n\}$.
for $1 \leq t \leq T$ **do**
    Let $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$.
    Select the distribution $\boldsymbol{p}^{(t)} = \left(\frac{w_1^{(t)}}{\Phi^{(t)}}, \frac{w_2^{(t)}}{\Phi^{(t)}}, \dots, \frac{w_n^{(t)}}{\Phi^{(t)}} \right)$.
    Update the weight for every expert: $w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\eta m_i^{(t)})$.

Some notes:

- The main difference is multiplying by $\exp(-\eta m_i^{(t)})$ instead of $1 - \eta m_i^{(t)}$. These two quantities are closely related via $1 - x \leq e^{-x}$, with the two quantities being increasingly similar as x decreases. In view of this inequality, the resulting guarantee for Hedge follows fairly similarly to MWU.
- In slight more detail: (i) The analysis is still based on upper and lower bounding $\Phi^{(T+1)}$; (ii) The lower bound on $\Phi^{(T+1)}$ is simpler due to the exponential update rule; (iii) The upper bound on $\Phi^{(T+1)}$ is a bit more complicated and requires this inequality $e^{-\eta x} \leq 1 - \eta x + \eta^2 x^2$ for $|\eta x| \leq 1$.

Skipping the remaining details, the resulting theorem is stated as follows without proof.

Theorem 2.1. *Assuming that all losses satisfy $m_i^{(t)} \in [-1, 1]$ and $\eta \in [0, 1]$, the Hedge algorithm guarantees that after T rounds, for any expert $i \in \{1, \dots, n\}$,*

$$\sum_{t=1}^{T} \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} \left(\boldsymbol{m}^{(t)} \right)^2 \cdot \boldsymbol{p}^{(t)} + \frac{\ln n}{\eta}$$

where $\left(\boldsymbol{m}^{(t)} \right)^2$ is the coordinate-wise square of $\boldsymbol{m}^{(t)}$.

Observe that the right-hand side above depends on $\boldsymbol{p}^{(t)}$, while the guarantee of MWU only depends on the loss of the best expert in hindsight, which gives stronger bounds in some applications. For example, in approximation of many NP-hard problems, we explicitly require the use of MWU to obtain better bounds than the Hedge algorithm.

On the other hand, we still have $\sum_{t=1}^{T} \left(\boldsymbol{m}^{(t)} \right)^2 \cdot \boldsymbol{p}^{(t)} \leq T$, and if we weaken the second term in this way, we end up with exactly the same weakened bound as MWU, with a regret of $2\sqrt{T \ln n}$.

3 (**Optional**) Beyond Bounded Losses

We have focused on losses $m_i^{(t)}$ that lie in $[-, 1]$, but are otherwise arbitrary. This means that we are very flexible in being able to handle many different sequences of losses (other than the boundedness restriction).

On the other hand, there are certain specific loss functions where we can get significantly stronger guarantees than the above theorems, or where we can allow for unbounded loss values under which the above theorems are not even applicable. Two such losses are as follows:

- **Logarithmic loss:** Suppose a sequence $x_1, \ldots, x_T$ is given sequentially to the learner, who at time t (having seen $x_1, \ldots, x_{t-1}$), is required to choose[5] some $\theta_t \in \Theta$, where each θ is a parameter (or vector of parameters) associated with a probability density function $f_\theta(x)$. For example, $\theta = (\mu, \sigma^2)$ could represent the mean/variance parameters of a Gaussian distribution.

  We are interested in finding the density function that "best models" $x_1, \ldots, x_T$. For doing so, a natural choice is the *logarithmic loss* (since it's related to maximum-likelihood estimation; see below):

  $$\ell_\theta(x) = \log \frac{1}{f_\theta(x)}.$$

  This leads to the following notion of "regret" (total loss incurred compared to the smallest possible):

  $$R_T = \sum_{t=1}^{T} \log \frac{1}{f_{\theta_t}(x_t)} - \min_{\theta \in \Theta} \sum_{t=1}^{t} \log \frac{1}{f_\theta(x_t)}.$$

  Note that $\sum_{t=1}^{t} \ell_\theta(x_t) = -\log\big(\prod_{t=1}^{T} f_\theta(x_t)\big)$, so what we are essentially doing is seeking θ that maximizes the likelihood of the data, thus the connection to maximum-likelihood estimation.

  Like in the bounded losses setting, it turns out to be beneficial to let the algorithm randomize its decision and return a *distribution w_t over θ values*, leading to the following generalization:

  $$R_T = \sum_{t=1}^{T} \log \frac{1}{\mathbb{E}_{\theta \sim w_t}[f_\theta(x_t)]} - \min_{\theta \in \Theta} \sum_{t=1}^{t} \log \frac{1}{f_\theta(x_t)}.$$

- **Squared loss:** Suppose that instead of just an x-sequence, a sequence of (x_t, y_t) pairs is given, and at time t the learner is required to predict y_t given both x_t and all the previous pairs. This is a *regression* problem if y is real-valued, and a *classification problem* is y is binary-valued. In the former case, a natural loss is the square loss:

  $$\ell(y, \hat{y}) = (y - \hat{y})^2.$$

  This loss is also well-suited to MWU algorithms, albeit not quite to the extent of the logarithmic loss. We will not consider it further here.

In the following, we focus on the logarithmic loss, or log loss for short. The MWU algorithm maintains a set of weights over all the θ values, taking the form

$$w_t(\theta) \propto \exp\bigg(-\eta \sum_{t=1}^{t-1} \ell_\theta(x_i)\bigg)\pi(\theta),$$

[5] If Θ is a finite set then we can interpret each $\theta \in \Theta$ as an "expert", but here we could also have a continuous set.

where $\eta > 0$ is a parameter (typically 1 when log loss is used) and $\pi(\theta)$ is a "prior" (often taken to be uniform if Θ is a finite or compact set). The symbol $\propto$ means the right-hand side should be scaled so that the weights sum (or integrate) to one.

We observe that the above formula closely resembles what we used in the case of bounded losses, and we can similarly interpret it as a multiplicative update: After observing x_t, we update

$$w_{t+1}(\theta) \propto w_t(\theta) e^{-\eta \ell_\theta(x_i)},$$

noting that the multiplication becomes summation inside the exponential.

In the case that Θ is a finite set of size n and $\pi(\cdot)$ is uniform, it can be shown that $R_T \leq \log n$, which is significantly stronger than what we got for bounded losses – the right-hand side is no longer $O(\sqrt{T})$, but instead has no dependence on T at all! (Though the dependence on n remains similar.) Results for continuous sets Θ are also well-known. For more on this and also the square loss, see (e.g.) the tutorial slides `https://maths.anu.edu.au/sites/prod.maths.sca-lws06.anu.edu.au/files/Nikita%20SummerSchoolFinal.pdf`.

4 (**Optional**) Applications of MWU

In this section, we survey several applications and uses of multiplicative weights update algorithms, some of which are far from obvious. Even more applications can be found in Section 3 of the survey `https://theoryofcomputing.org/articles/v008a006/`.

4.1 Machine Learning Algorithms

One of the most fundamental machine learning problems is *binary classification*, where we are given a training set $\{(x_t, y_t)\}_{t=1}^n$ with $x \in \mathbb{R}^d$ and $y_t \in \{-1, 1\}$ (binary labels) and we want to devise a "good" predictor $\hat{y} = f(x)$. A natural starting point is to consider *linear predictors*, taking the form $f_\theta(x) = \langle \theta, x \rangle$, and the task becomes finding a good choice of θ.

One of the earliest algorithms for this task is the *Perceptron algorithm*, which iterates through the training set and updates $\theta \leftarrow \theta + x_t y_t$ whenever a mistake is made. This is an *additive update*. A lesser-known algorithm called the **Winnow algorithm** instead performs *multiplicative updates*, and comes with theoretical guarantees that resemble those of the Perceptron algorithm, but can also be much stronger in some cases. See Section 3.1 of `https://theoryofcomputing.org/articles/v008a006/`.

Another famous algorithm based on MWU is **AdaBoost**, which roughly operates as follows:

- Maintain a collection of weights (summing to one) over the training points;
- In each iteration, choose a classifier (typically from a "simple" class) that minimizes the weighted training error;
- Using the *multiplicative weights method*, up-weigh the points that are mis-classified by that classifier (so that we give them more attention later and "correct" for these mistakes), and down-weigh the ones that are classified correctly.
- Continue until some stopping criterion, and let the final classifier be a suitably-weighted combination of those constructed along the way (i.e., a form of weighted majority vote).

AdaBoost has a number of interesting theoretical guarantees and is also widely adopted in practice. See https://www.comp.nus.edu.sg/~scarlett/CS5339_notes/ for some lecture notes on this topic, or https://jeremykun.com/2015/05/18/boosting-census/ for an introductory blog post.

4.2 Solving Zero-Sum Games

Consider a setting in which there are two agents that choose values $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, n\}$, from which a value $A(i, j) \in [0, 1]$ is formed that the first player wants to minimize and the second player wants to maximize. More generally, the players could randomize their choice, producing distributions $\mathbf{p}$ and $\mathbf{q}$ and an average value of $A(\mathbf{p}, \mathbf{q}) = \mathbb{E}_{i \sim \mathbf{p}, j \sim \mathbf{q}}[A(i, j)]$.

Once one player's (randomized) strategy is fixed, it is easy to show that the other's best strategy is deterministic. Using this observation along with the minimax theorem for convex/concave functions (note that the quantity $\mathbb{E}_{i \sim \mathbf{p}, j \sim \mathbf{q}}[A(i, j)]$ is linear in $\mathbf{p}$ and linear in $\mathbf{q}$), we find that

$$\min_{\mathbf{p}} \max_{j} A(\mathbf{p}, j) = \max_{\mathbf{q}} \min_{i} A(i, \mathbf{q}) =: A^*.$$

The MWU algorithm can be used to not only prove this result, but also efficiently find a pair $(\mathbf{p}, j)$ or $(\mathbf{q}, i)$ whose min-max or max-min value is arbitrarily close to A^*.

The idea is to let the MWU losses be $m_i^{(t)} = A(i, j^{(t)})$, where $j^{(t)}$ is the best deterministic choice for player 2 given player 1's mixed strategy $\mathbf{p}^{(t)}$ formed after the previous round. The intuition is that the total loss incurred by $\mathbf{p}^{(1)}, \dots, \mathbf{p}^{(T)}$ cannot be much higher than the loss of any fixed choice of i (by the preceding theorems), and thus not much higher than the average loss over $i \sim \mathbf{p}$ for any fixed distribution $\mathbf{p}$. When we choose $\mathbf{p} = \mathbf{p}^*$ with $\mathbf{p}^*$ solving the min-max problem $\min_{\mathbf{p}} \max_j A(\mathbf{p}, j)$, it follows that the loss approaches A^*. See Section 3.2 of https://theoryofcomputing.org/articles/v008a006/ for the details.

4.3 Solving Linear Programs

MWU has been applied to solving covering/packing linear programs where the coefficients are non-negative. The idea is to reduce optimizing a linear program to feasibility testing problems then perform binary search on the feasibility variant. The MWU method comes in in solving the feasibility problem approximately. Each constraint in the LP corresponds to an expert. The loss incurred by an expert is defined to be how much the constraint is violated. The intuition is that by putting more weights on the constraints that are not satisfied, we will eventually violate fewer constraints and find a feasible solution as number of iterations increases. More details can be found in Sec. 3.3 of https://theoryofcomputing.org/articles/v008a006/, or the later part of the blog post https://www.jeremykun.com/2017/02/27/the-reasonable-effectiveness-of-the-multiplicative-weights-update-algorithm/.

4.4 Bandit Algorithms

In this lecture, we assumed that after each decision is made, the loss associated with *all experts* are revealed to the learner. That is, after each round, we know how good or bad each expert's decision would have been had we chosen it. A more challenging setting is that of *bandit feedback*, in which we only get to observe the loss associated with the single expert that we chose. This may be more natural in applications, e.g., imagine each "expert" corresponds to a choice of advertisement to display to a user with the hope that they click on

it – we get to observe whether they did so, but we can only guess as to whether they would have clicked on some other one.

Despite the considerably reduced amount of information available the bandit setting, we can still use the MWU idea. The idea is to replace each loss $m_i^{(t)}$ by a surrogate that is *correct on average*, which can be done by setting unobserved losses to 0 and dividing observed losses by the probability assigned to the expert. This gives rise to the *EXP3 algorithm*, which incurs regret at most $2\sqrt{nT \log n}$ after T rounds with n experts. This means that the regret has a similar dependence on T as the full information setting, but a stronger dependence on n. See Chapter 11 of `https://tor-lattimore.com/downloads/book/book.pdf` for the details, or `https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/` for an introductory blog post.

4.5 Others

Some other applications are briefly mentioned as follows:

- Online convex optimization (see the tutorial for a question on this)
- Noisy binary search (see "Noisy binary search and its applications")
- Learning statistical graphical models (see "Learning graphical models using multiplicative weights")
- Approximating integer programs (see "Randomized rounding without solving the linear program")
- Semidefinite programming (see "Fast algorithms for approximate semidefinite programming using the multiplicative weights update method")
- Differential privacy (see Sec. 11.2 of "The algorithmic foundations of differential privacy")
- Constrained consensus optimization (see "Consensus multiplicative weights update: Learning to learn using projector-based game signatures")