CS 170: The Multiplicative Weights Update Method — Lecture Notes

Course: CS 170 — Efficient Algorithms and Intractable Problems, Spring 2026
Instructors: Lijie Chen, Umesh V. Vazirani

The Setting

In the previous lecture on zero-sum games, we proved the Minimax Theorem using LP duality: every zero-sum game has a well-defined value V, and both players have optimal mixed strategies achieving this value. But the proof was existential — it told us optimal strategies *exist* (because LPs can be solved), but it didn't give us a simple, intuitive algorithm for *finding* them.

In this lecture, we develop the **Multiplicative Weights Update (MWU)** method, a remarkably simple and powerful algorithmic technique. It arises from a natural question: *how should you combine the advice of multiple experts when you don't know which one to trust?* We will see that this seemingly simple question leads to an algorithm that:

1. Achieves near-optimal performance in online prediction problems, and
2. Provides a constructive, iterative proof of the Minimax Theorem.

Part 1: The Experts Problem

You have n experts, each making a binary prediction every day (say, whether the stock market goes up or down). After each day, the truth is revealed. You want to somehow combine their advice to make as few mistakes as possible.

Let's start with a clean case: suppose one of the n experts is **always correct** — you just don't know which one. Here's a simple strategy: maintain a set S of experts who haven't been wrong yet, and each day predict with the majority vote of S. Whenever an expert makes a mistake, throw them out.

Why does this work? The perfect expert never gets thrown out, so S always contains at least one expert. And every time *you* make a mistake, the majority of S was wrong — meaning at least half the experts in S just got eliminated. So $|S|$ gets halved with each mistake you make. Since you start with n experts and can't go below 1, you make at most $\log_2 n$ mistakes total. With 1000 experts, that's at most 10 mistakes, no matter how many days you play. Not bad!

This is called the **Halving Algorithm**. But of course, in reality there is no perfect expert — everyone makes mistakes sometimes. And once we drop that assumption, the Halving algorithm falls apart: it keeps eliminating experts until nobody is left.

The fix is natural: instead of eliminating wrong experts entirely (multiplying their weight by 0), just **penalize** them — multiply their weight by $1 - \epsilon$ for some small $\epsilon > 0$. Wrong experts gradually lose influence, but they're never fully removed, so they can recover if they start doing well again. This is the idea behind MWU.

Part 2: The Multiplicative Weights Update Algorithm

2.1 The Algorithm

Algorithm (Multiplicative Weights Update):

1. Initialize weights: $w_i^{(1)} = 1$ for all $i = 1, \ldots, n$.
2. For each round $t = 1, \ldots, T$:
   - Compute the probability distribution:

$$p_i^{(t)} = \frac{w_i^{(t)}}{\sum_{j=1}^{n} w_j^{(t)}}$$

   - **Randomly sample** expert i with probability $p_i^{(t)}$ and predict $p^{(t)} = f_i^{(t)}$.
   - After observing $y^{(t)}$, update weights: for every expert i that was wrong (i.e., $f_i^{(t)} \neq y^{(t)}$), set

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

     Correct experts keep their weight unchanged: $w_i^{(t+1)} = w_i^{(t)}$.

The algorithm is beautifully simple: maintain a weight for each expert, sample proportionally, and multiplicatively downweight whoever was wrong. That's it.

2.2 Intuition

Why does this work? There are a few ways to think about it.

Connection to the Halving algorithm. The Halving algorithm is MWU with $\epsilon = 1$: wrong experts get their weight multiplied by $1 - 1 = 0$, i.e., eliminated. Setting $\epsilon < 1$ is a "soft" version of the same idea — we still punish wrong experts, just less aggressively. The parameter ϵ controls the tradeoff: large ϵ means we react strongly to mistakes (good if the best expert is much better than the rest), while small ϵ means we're more forgiving (good if even the best expert makes many mistakes).

Why randomize? You might wonder: why not just predict with the weighted majority vote (the prediction favored by the higher total weight), like the Halving algorithm does? You can — and this is called the **Weighted Majority** algorithm.

But it turns out to lose a factor of 2 in the bound. The issue is that when you predict deterministically, you might be wrong even if only slightly more than half the weight is on the wrong side. Randomizing avoids this: your expected mistake rate in each round is exactly the fraction of weight on wrong experts, not a 0/1 coin flip based on the majority.

Why multiplicative updates? The multiplicative rule $w_i \leftarrow w_i \cdot (1-\epsilon)$ has a natural "exponential forgetting" property: after k mistakes, expert i has weight $(1-\epsilon)^k$, which decays exponentially in the number of mistakes. This means the algorithm automatically concentrates its probability mass on experts with the fewest mistakes — exactly what we want. An additive update (subtract a fixed penalty) wouldn't have this self-correcting property: a bad expert with large initial weight could dominate for too long.

2.3 Analysis

Let M denote the **total number of mistakes** the algorithm makes over all T rounds, and let $m^* = \min_i m_i$ be the total mistakes of the **best expert in hindsight**. Since the algorithm randomizes, M is a random variable, so we analyze $\mathbb{E}[M]$.

The key observation is that the algorithm's expected mistake rate in any single round t equals the total probability mass on wrong experts:

$$\mathbb{E}[\text{mistake in round } t] = \sum_{i : f_i^{(t)} \neq y^{(t)}} p_i^{(t)} = F^{(t)}$$

where $F^{(t)}$ is the **fraction of total weight on wrong experts** in round t. Summing over all rounds, $\mathbb{E}[M] = \sum_{t=1}^{T} F^{(t)}$.

Theorem (MWU Regret Bound). For any setting of the outcomes and any expert i^* that makes m^* mistakes:

$$\mathbb{E}[M] \leq (1+\epsilon)\, m^* + \frac{\ln n}{\epsilon}$$

Proof. We use a **potential function argument** — tracking the total weight $\Phi^{(t)} = \sum_i w_i^{(t)}$.

Potential drop per round. In round t, every wrong expert has its weight multiplied by $(1-\epsilon)$. Thus:

$$\Phi^{(t+1)} = \Phi^{(t)} - \epsilon \cdot \sum_{i \text{ wrong}} w_i^{(t)} = \Phi^{(t)} \left(1 - \epsilon \cdot F^{(t)}\right)$$

After T rounds:

$$\Phi^{(T+1)} = n \cdot \prod_{t=1}^{T} \left(1 - \epsilon \cdot F^{(t)}\right)$$

Lower bound. The weight of the best expert: $\Phi^{(T+1)} \geq w_{i^*}^{(T+1)} = (1-\epsilon)^{m^*}$.

Combining. Taking logarithms:

$$m^* \ln(1-\epsilon) \leq \ln n + \sum_{t=1}^{T} \ln(1 - \epsilon \cdot F^{(t)})$$

Using $\ln(1-x) \leq -x$:

$$-m^*\epsilon \leq \ln n - \epsilon \sum_{t=1}^{T} F^{(t)}$$

Since $\mathbb{E}[M] = \sum_t F^{(t)}$:

$$\epsilon \cdot \mathbb{E}[M] \leq \ln n + m^*\epsilon$$

Dividing by ϵ and using the tighter bound $\ln(1-\epsilon) \geq -\epsilon/(1-\epsilon) \geq -\epsilon(1+\epsilon)$ for small ϵ:

$$\boxed{\mathbb{E}[M] \leq (1+\epsilon)\, m^* + \frac{\ln n}{\epsilon}}$$

□

2.4 Optimal Parameter Choice and No-Regret

Setting $\epsilon = \sqrt{\frac{\ln n}{m^*}}$ gives:

$$\mathbb{E}[M] \leq m^* + O\left(\sqrt{m^* \ln n}\right)$$

If we don't know m^* in advance, we can use the **doubling trick**: guess $m^* = 1, 2, 4, 8, \ldots$, restart when the guess is exceeded, and lose only a constant factor.

Alternatively, setting $\epsilon = \sqrt{\frac{\ln n}{T}}$ (using the time horizon) gives a regret bound of $O(\sqrt{T \ln n})$, which grows **sublinearly** in T — meaning the **average** regret per round vanishes as $T \to \infty$:

$$\boxed{\text{Regret} \leq 2\sqrt{T \ln n}}$$

This is the celebrated **no-regret** guarantee. With $n = 100$ experts over $T = 10{,}000$ rounds, the total regret is at most $2\sqrt{10{,}000 \cdot \ln 100} \approx 430$ — an average of 0.043 extra mistakes per round compared to the best expert.

Part 3: Application — Solving Zero-Sum Games

3.1 The Connection

Here is the punchline that ties this lecture to the previous one. Recall from the zero-sum games lecture:

> A zero-sum game is defined by an $m \times n$ payoff matrix G. The Minimax Theorem says $\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x}^\top G \mathbf{y} = \min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x}^\top G \mathbf{y} = V$.

We proved this using LP duality. Now we give a **constructive, algorithmic proof** using MWU.

3.2 The Setup

Consider the game from **Row's perspective**. Row has m strategies (actions). In each round t, Row picks a mixed strategy $\mathbf{x}^{(t)}$ and Column responds with a best response — the column $j^{(t)}$ that minimizes $\sum_i x_i^{(t)} G_{ij}$.

From Row's point of view, this is exactly the experts problem! Row's m strategies are the "experts," and in round t, the "cost" of strategy i is $-G_{i,j^{(t)}}$ (we negate because Row wants to *maximize* payoff).

3.3 The Algorithm

MWU for Zero-Sum Games:

1. Initialize $w_i^{(1)} = 1$ for $i = 1, \ldots, m$.
2. For $t = 1, \ldots, T$:
   - Row plays the distribution $x_i^{(t)} = w_i^{(t)} / \sum_j w_j^{(t)}$.
   - Column plays a best response: $j^{(t)} = \arg\min_j \sum_i x_i^{(t)} G_{ij}$.
   - Update: $w_i^{(t+1)} = w_i^{(t)} \cdot (1-\epsilon)^{-G_{i,j^{(t)}}}$ (rescaling costs to $[0,1]$).
3. Output: $\bar{\mathbf{x}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}^{(t)}$ (the time-averaged strategy).

3.4 Why It Works

By the MWU regret bound, for every row strategy i:

$$\frac{1}{T} \sum_{t=1}^{T} \langle \mathbf{x}^{(t)}, \mathbf{c}^{(t)} \rangle \leq \frac{1}{T} \sum_{t=1}^{T} c_i^{(t)} + \frac{\epsilon T + \frac{\ln m}{\epsilon}}{T}$$

Translating back to game payoffs: the average payoff of the algorithm's time-averaged strategy $\bar{\mathbf{x}}$ against Column's best responses is at most δ-worse than any pure row strategy, where $\delta = O(\sqrt{\frac{\ln m}{T}})$.

This means $\bar{\mathbf{x}}$ is an **approximate maximin strategy**: it guarantees Row an expected payoff of at least $V - \delta$. Similarly, if we also run MWU for Column, we get an approximate minimax strategy. As $T \to \infty$, $\delta \to 0$, and we recover the exact game value and optimal strategies.

This is a constructive proof of the Minimax Theorem — using only a simple iterative update rule, no LP solver required!

3.5 Worked Example

Consider the Matching Pennies game from the zero-sum games lecture:

$$G = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

We know the game value is $V = 0$ and the optimal strategy is $(1/2, 1/2)$ for both players. Let's run MWU with $\epsilon = 0.3$ and see how the weights evolve.

Round 1. Weights: $w_1 = 1, w_2 = 1$. Row plays $(1/2, 1/2)$. Column's best response: both columns yield payoff 0, say Column picks column 1. Costs: strategy 1 gets payoff $G_{11} = 1$ (good), strategy 2 gets $G_{21} = -1$ (bad). After rescaling to costs in $[0, 1]$: $c_1 = 0, c_2 = 1$.

Update: $w_1 = 1 \cdot (0.7)^0 = 1$, $w_2 = 1 \cdot (0.7)^1 = 0.7$.

Round 2. Weights: $w_1 = 1, w_2 = 0.7$. Row plays $(0.588, 0.412)$. Column best response against $(0.588, 0.412)$: payoff for col 1 is $0.588 - 0.412 = 0.176$, for col 2 is $-0.588 + 0.412 = -0.176$. Column picks column 2 (minimizes Row's payoff).

Costs: $c_1 = 1$ (strategy 1 got payoff -1), $c_2 = 0$ (strategy 2 got payoff 1).

Update: $w_1 = 1 \cdot 0.7 = 0.7$, $w_2 = 0.7 \cdot 1 = 0.7$.

Round 3. Weights: $w_1 = 0.7, w_2 = 0.7$. Row plays $(1/2, 1/2)$ — we're back to the equilibrium!

The algorithm oscillates around the equilibrium and converges in the time-averaged sense to the optimal strategy $(1/2, 1/2)$.

Part 4: Further Reading

- **Sanjeev Arora, Elad Hazan, Satyen Kale — "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications"** (Theory of Computing, 2012). The definitive survey unifying the many incarnations of MWU across computer science.

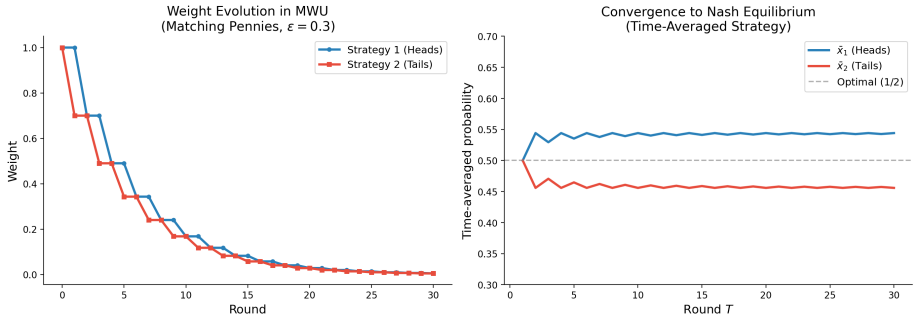

Figure 1: MWU on Matching Pennies: weight evolution (left) and convergence of the time-averaged strategy to the Nash equilibrium (right).

- **Dasgupta, Papadimitriou, Vazirani** — ***Algorithms***, Chapter 7. Background on linear programming and zero-sum games that sets the stage for MWU.
- **Umesh Vazirani** — **CS 270 Lecture Notes** (UC Berkeley). The lecture on the experts problem and weighted majority that inspired these notes.
- **Yoav Freund and Robert Schapire** — **"A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting"** (JCSS, 1997). Introduces the Hedge algorithm and the connection to AdaBoost.
- **Nick Littlestone** — **"Learning Quickly When Irrelevant Attributes Abound"** (Machine Learning, 1988). The Winnow algorithm, an early instance of multiplicative updates for online learning.