Advanced Algorithms 18 December 2018

Lecture 14: Online Algorithms and Competitive Analysis III

Lecturer: Mohsen Ghaffari *Scribe: Davin Choo*

1 Multiplicative Weights Update

In this final lecture, we discuss the Multiplicative Weight Updates (MWU) method. A comprehensive survey on MWU and its applications can be found in [AHK12].

Definition 1 (The learning from experts problem)**.** *Every day, we are to make a binary decision. At the end of the day, a binary output is revealed and we incur a mistake if our decision did not match the output. Suppose we have access to n experts $e_1, \dots, e_n$, each of which makes an recommendation for the binary decision to take per day. How does one make use of the experts to minimize the total number of mistakes on an online binary sequence?*

Toy setting Consider a stock market with only a single stock. Every day, we are decide whether to buy the stock or not. At the end of the day, the stock value will be revealed and we incur a mistake/loss of 1 if we did not buy when the stock value rose, or bought when the stock value fell.

Example — Why it is non-trivial Suppose $n = 3$ and $\sigma = (1, 1, 0, 0, 1)$. In hindsight, we have:

Days	1	1	0	0	1
e_1	1	1	0	0	1
e_2	1	0	0	0	1
e_3	1	1	1	1	0

In hindsight, e_1 is *always* correct so we would have incurred 0 mistakes if we always followed e_1's recommendation. However, we do not know which is expert e_1 (assuming a perfect expert even exists). Furthermore, it is not necessarily true that the best expert always incurs the least number of mistakes on any prefix of the sequence σ. Ignoring e_1, one can check that e_2 outperforms e_3 on the example sequence. However, at the end of day 2, e_3 incurred 0 mistakes while e_2 incurred 1 mistake.

The goal is as follows: If a perfect expert exists, we hope to eventually converge to always following him/her. If not, we hope to *not* do much worse than the best expert on the entire sequence.

1.1 Warm up: Perfect expert exists

Suppose there exists a perfect expert. Do the following on each day:

- Make a decision by taking the majority vote of the remaining experts.
- If we incur a loss, remove the experts that were wrong.

Theorem 2. *We incur at most $\log_2 n$ mistakes on any given sequence.*

Proof. Whenever we incur a mistake, at least half the experts were wrong and were removed. Hence, the total number of experts is *at least* halved whenever a mistake occurred. After at most $\log_2 n$ removals, the only expert left will be the perfect expert and we will be always correct thereafter. □

1.2 A deterministic Multiplicative Weights Update algorithm

Suppose that there may not be a perfect expert. The idea is similar, but we update our trust for each expert instead of completely removing an expert when he/she makes a mistake. Consider the following deterministic algorithm (DMWU):

- Initialize weights $w_i = 1$ for expert e_i, for $i \in \{1, \dots, n\}$.
- On each day:

  - Make a decision by the weighted majority.
  - If we incur a loss, set w_i to $(1-\epsilon) \cdot w_i$ for each wrong expert, for some constant $\epsilon \in (0, \frac{1}{2})$.

Theorem 3. *Suppose the best expert makes m^* mistakes and* DMWU *makes m mistakes. Then,*

$$m \leq 2(1+\epsilon)m^* + \frac{2\ln n}{\epsilon}$$

Proof. Observe that when DMWU makes a mistake, the weighted majority was wrong and their weight decreases by a factor of $(1-\epsilon)$. Suppose that $\sum_{i=1}^{n} w_i = x$ at the start of the day. If we make a mistake, x drops to $\leq \frac{x}{2}(1-\epsilon) + \frac{x}{2} = x(1-\frac{\epsilon}{2})$. That is, the overall weight reduces by at least a factor of $(1-\frac{\epsilon}{2})$. Since the best expert e^* makes m^* mistakes, his/her weight at the end would be $(1-\epsilon)^{m^*}$. By the above observation, the total weight of all experts would be $\leq n(1-\frac{\epsilon}{2})^m$ at the end of the sequence. Then,

$$\begin{array}{rrcll} & (1-\epsilon)^{m^*} & \leq & n(1-\frac{\epsilon}{2})^m & \text{Expert } e^*\text{'s weight is part of the overall weight} \\ \Rightarrow & m^* \ln(1-\epsilon) & \leq & \ln n + m\ln(1-\frac{\epsilon}{2}) & \text{Taking ln on both sides} \\ \Rightarrow & m^*(-\epsilon-\epsilon^2) & \leq & \ln n + m(-\frac{\epsilon}{2}) & \text{Since } -x-x^2 \leq \ln(1-x) \leq -x \text{ for } x \in (0,\frac{1}{2}) \\ \Rightarrow & m & \leq & 2(1+\epsilon)m^* + \frac{2\ln n}{\epsilon} & \text{Rearranging} \end{array}$$

□

Remark 1 In the warm up, $m^* = 0$.

Remark 2 For $x \in (0, \frac{1}{2})$, the inequality $-x - x^2 \leq \ln(1-x) \leq -x$ is due to the Taylor expansion[1] of ln. A more familiar equivalent form would be: $e^{-x-x^2} \leq (1-x) \leq e^{-x}$.

Theorem 4. *No deterministic algorithm $\mathcal{A}$ can do better than 2-competitive.*

Proof. Consider only two experts e_0 and e_1 where e_0 always outputs 0 and e_1 always outputs 1. Any binary sequence σ must contain at least $\frac{|\sigma|}{2}$ zeroes or $\frac{|\sigma|}{2}$ ones. Thus, $m^* \leq \frac{|\sigma|}{2}$. On the other hand, the adversary looks at $\mathcal{A}$ and produces a sequence σ which forces $\mathcal{A}$ to incur a loss every day. Thus, $m = |\sigma| \geq 2m^*$. □

1.3 A randomized Multiplicative Weights Update algorithm

The 2-factor in DMWU is due to the fact that DMWU deterministically takes the (weighted) majority at each step. Let us instead interpret the weights as probabilities. Consider the following randomized algorithm (RMWU):

- Initialize weights $w_i = 1$ for expert e_i, for $i \in \{1, \ldots, n\}$.
- On each day:
  - Pick a random expert with probability proportional to their weight.
    (i.e. Pick e_i with probability $w_i / \sum_{i=1}^{n} w_i$)
  - Follow that expert's recommendation.
  - For each wrong expert, set w_i to $(1-\epsilon) \cdot w_i$, for some constant $\epsilon \in (0, \frac{1}{2})$.

Another way to think about the probabilities is to split all experts into two groups $A = \{$Experts that output 0$\}$ and $B = \{$Experts that output 1$\}$. Then, decide '0' with probability $\frac{w_A}{w_A + w_B}$ and '1' with probability $\frac{w_B}{w_A + w_B}$, where $w_A = \sum_{i \in A} w_i$ and $w_B = \sum_{i \in B} w_i$ are the sum of weights in each set.

Theorem 5. *Suppose the best expert makes m^* mistakes and* RMWU *makes m mistakes. Then,*

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

[1]See https://en.wikipedia.org/wiki/Taylor_series#Natural_logarithm

Proof. Fix an arbitrary day $j \in \{1, \ldots, |\sigma|\}$. Denote $A = \{\text{Experts that output 0 on day } j\}$ and $B = \{\text{Experts that output 1 on day } j\}$, where $w_A = \sum_{i \in A} w_i$ and $w_B = \sum_{i \in B} w_i$ are the sum of weights in each set. Denote F_j be the weighted fraction of wrong experts on day j. If $\sigma_j = 0$, then $F_j = \frac{w_B}{w_A + w_B}$. If $\sigma_j = 1$, then $F_j = \frac{w_A}{w_A + w_B}$. By definition of F_j, RMWU makes a mistake on day j with probability F_j. By linearity of expectation, $\mathbb{E}[m] = \sum_{j=1}^{|\sigma|} F_j$.

Since the best expert e^* makes m^* mistakes, his/her weight at the end would be $(1-\epsilon)^{m^*}$. On each day, RMWU reduces the overall weight by a factor of $(1 - \epsilon \cdot F_j)$ by penalizing wrong experts. Hence, the total weight of all experts would be $n \cdot \Pi_{j=1}^{|\sigma|}(1 - \epsilon \cdot F_j)$ at the end of the sequence. Then,

$$\begin{array}{rrcll}
 & (1-\epsilon)^{m^*} & \leq & n \cdot \Pi_{j=1}^{|\sigma|}(1 - \epsilon \cdot F_j) & \text{Expert } e^*\text{'s weight is part of the overall weight} \\
\Rightarrow & (1-\epsilon)^{m^*} & \leq & n \cdot e^{\sum_{j=1}^{|\sigma|}(-\epsilon \cdot F_j)} & \text{Since } (1-x) \leq e^{-x} \\
\Rightarrow & (1-\epsilon)^{m^*} & \leq & n \cdot e^{-\epsilon \cdot \mathbb{E}[m]} & \text{Since } \mathbb{E}[m] = \sum_{j=1}^{|\sigma|} F_j \\
\Rightarrow & m^* \ln(1-\epsilon) & \leq & \ln n - \epsilon \cdot \mathbb{E}[m] & \text{Taking ln on both sides} \\
\Rightarrow & \mathbb{E}[m] & \leq & -\frac{\ln(1-\epsilon)}{\epsilon} m^* + \frac{\ln n}{\epsilon} & \text{Rearranging} \\
\Rightarrow & \mathbb{E}[m] & \leq & (1+\epsilon) m^* + \frac{\ln n}{\epsilon} & \text{Since } -\ln(1-x) \leq -(-x - x^2) = x + x^2
\end{array}$$

□

1.4 A generalization of the Multiplicative Weights Update algorithm

Consider the following generalized setting:

- Denote the loss of expert i on day t as $l_i^t \in [-\rho, \rho]$, for some constant ρ
- When we incur a loss, update the weights of affected experts from w_i to $(1 - \epsilon \frac{l_i^t}{\rho}) w_i$.

Remark $\frac{l_i^t}{\rho}$ is essentially the normalized loss $\in [-1, 1]$.

Claim 6 (Without proof). *With RMWU, we have* $\mathbb{E}[m] \leq \min_i (\sum_t l_i^t + \epsilon \sum_t |l_i^t| + \frac{\rho \ln n}{\epsilon})$.

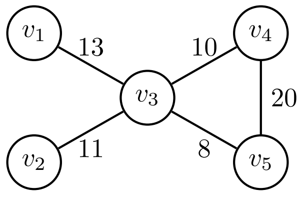
Remark If each expert has a different ρ_i, one can modify the update rule and claim to use ρ_i instead of a uniform ρ accordingly.

1.5 Application: Online routing of virtual circuits

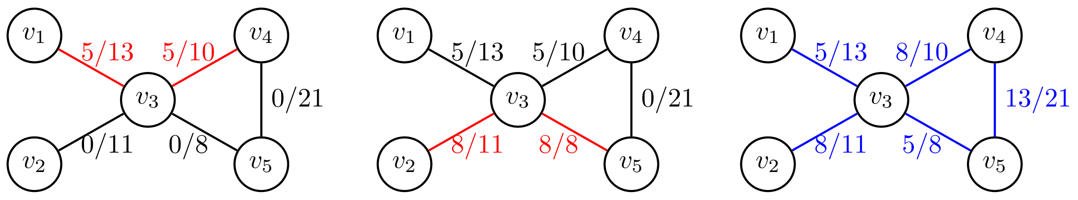
Definition 7 (The online routing of virtual circuits problem). *Consider a graph $G = (V, E)$ where each edge $e \in E$ has a capacity u_e. A request is denoted by a triple $\langle s(i), t(i), d(i) \rangle$, where $s(i) \in V$ is the source, $t(i) \in V$ is the target, and $d(i) > 0$ is the demand for the i^{th} request respectively. Given the i^{th} request, we are to build a connection (single path P_i) from $s(i)$ to $t(i)$ with flow $p(i)$. The objective is to minimize the maximum congestion on all edges as we handle requests in an online manner. To be precise, we wish to minimize* $\max_{e \in E} \frac{\sum_{i=1}^{|\sigma|} \sum_{P_i \ni e} d(i)}{u_e}$ *on the input sequence σ where $P_i \ni e$ is the set of paths that include edge e.*

Remark This is similar to the multi-commodity routing problem in lecture 5. However, in this problem, each commodity flow cannot be split into multiple paths, and the commodities appear in an online fashion.

Example Consider the following graph $G = (V, E)$ with 5 vertices and 5 edges with the edge capacities u_e annotated for each edge $e \in E$. Suppose there are 2 requests: $\sigma = (\langle v_1, v_4, 5 \rangle, \langle v_5, v_2, 8 \rangle)$.



Upon seeing $\sigma(1) = \langle v_1, v_4, 5\rangle$, we (red edges) commit to $P_1 = v_1 - v_3 - v_4$ as it minimizes the congestion to $5/10$. When $\sigma(2) = \langle v_5, v_2, 8\rangle$ appears, we are forced to take $P_2 = v_5 - v_3 - v_2$. This causes the congestion to be $8/8 = 1$. On the other hand, the optimal offline algorithm (blue edges) can attain a congestion of $8/10$ via $P_1 = v_1 - v_3 - v_5 - v_4$ and $P_2 = v_5 - v_4 - v_3 - v_2$.



To facilitate further discussion, we define the following notations:

- $p_e(i) = \frac{d(i)}{u_e}$ is the relative demand i with respect to the capacity of edge e.
- $l_e(j) = \sum_{P_i \ni e, i \leq j} p_e(i)$ as the relative load of edge e *after* request j
- $l_e^*(j)$ as the optimal offline algorithm's load of edge e *after* request j.

In other words, the objective is to minimize $\max_{e\in E} l_e(|\sigma|)$ for a given sequence σ. Denoting Λ as the (unknown) optimal congestion factor, we normalize $\widetilde{p}_e(i) = \frac{p_e(i)}{\Lambda}$, $\widetilde{l}_e(j) = \frac{l_e(j)}{\Lambda}$, and $\widetilde{l}_e^*(j) = \frac{l_e^*(j)}{\Lambda}$. Let a be a constant to be determined. Consider algorithm $\mathcal{A}$ which does the following on request $i+1$:

- Denote the cost of edge e by $c_e = a^{\widetilde{l}_e(i) + \widetilde{p}_e(i+1)} - a^{\widetilde{l}_e(i)}$
- Return the shortest $s(i) - t(i)$ path P_i on G with edge weights c_e

Finding the shortest path via the cost function c_e tries to minimize the load impact of the new $(i+1)^{th}$ request. To analyze $\mathcal{A}$, we consider the following potential function: $\Phi(j) = \sum_{e\in E} a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}_e^*(j))$, for some constant $\gamma \geq 1$. Because of normalization, $\widetilde{l}_e^*(j) \leq 1$, so $(\gamma - \widetilde{l}_e^*(j)) \in \Omega(1)$. Initially, when $j = 0$, $\Phi(0) = \sum_{e\in E} \gamma = m\gamma$.

Lemma 8. *For $\gamma \geq 1$ and $0 \leq x \leq 1$, $(1 + \frac{1}{2\gamma})^x < 1 + \frac{x}{\gamma}$.*

Proof. By Taylor series[2], $(1 + \frac{1}{2\gamma})^x = 1 + \frac{x}{2\gamma} + \mathcal{O}(\frac{x}{2\gamma}) < 1 + \frac{x}{\gamma}$. □

Lemma 9. *For $a = 1 + \frac{1}{2\gamma}$, $\Phi(j+1) - \Phi(j) \leq 0$.*

Proof. Let P_{j+1} be the path that $\mathcal{A}$ found and P_{j+1}^* be the path that the optimal offline algorithm assigned to the $(j+1)^{th}$ request $\langle s(j+1), t(j+1), d(j+1)\rangle$. For any edge e, observe the following:

- If $e \notin P_{j+1}^*$, the load on e due to the optimal offline algorithm remains unchanged. That is, $\widetilde{l}_e^*(j+1) = \widetilde{l}_e^*(j)$. On the other hand, if $e \in P_{j+1}^*$, then $\widetilde{l}_e^*(j+1) = \widetilde{l}_e^*(j) + \widetilde{p}_e(j+1)$.
- Similarly, (i) If $e \notin P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j)$; (ii) If $e \in P_{j+1}$, then $\widetilde{l}_e(j+1) = \widetilde{l}_e(j) + \widetilde{p}_e(j+1)$.
- If e is neither in P_{j+1} nor in P_{j+1}^*, then $a^{\widetilde{l}_e(j+1)}(\gamma - \widetilde{l}_e^*(j+1)) = a^{\widetilde{l}_e(j)}(\gamma - \widetilde{l}_e^*(j))$.
  That is, only edges used by P_{j+1} or P_{j+1}^* affect $\Phi(j+1) - \Phi(j)$.

Using the observations above together with Lemma 8 and the fact that $\mathcal{A}$ computes a shortest path, one

[2]See https://en.wikipedia.org/wiki/Taylor_series#Binomial_series

can show that $\Phi(j+1)-\Phi(j)\leq 0$. In detail,

$$\begin{array}{rll}
 & \Phi(j+1)-\Phi(j) & \\
= & \sum_{e\in E} a^{\widetilde{l}_e(j+1)}(\gamma-\widetilde{l}^*_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma-\widetilde{l}^*_e(j)) & \text{By definition of } \Phi \\
= & \sum_{e\in P_{j+1}\setminus P^*_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma-\widetilde{l}^*_e(j)) & \text{From observations above} \\
 & +\sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}(\gamma-\widetilde{l}^*_e(j)-\widetilde{p}_e(j+1)) - a^{\widetilde{l}_e(j)}(\gamma-\widetilde{l}^*_e(j)) & \\
= & \sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})(\gamma-\widetilde{l}^*_e(j)) - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}\widetilde{p}_e(j+1) & \text{Rewriting} \\
\leq & \sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j+1)}\widetilde{p}_e(j+1) & \widetilde{l}^*_e(j)\geq 0 \\
\leq & \sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1) & \widetilde{l}_e(j+1)\geq \widetilde{l}_e(j) \\
= & \sum_{e\in P_{j+1}} (a^{\widetilde{l}_e(j)+\widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1) & \text{For } e\in P_{j+1},\ \widetilde{l}_e(j)=\widetilde{l}_e(j)+\widetilde{p}_e(j+1) \\
\leq & \sum_{e\in P^*_{j+1}} \Big((a^{\widetilde{l}_e(j)+\widetilde{p}_e(j+1)} - a^{\widetilde{l}_e(j)})\gamma - a^{\widetilde{l}_e(j)}\widetilde{p}_e(j+1)\Big) & \text{Since } P_{j+1} \text{ is the shortest path} \\
= & \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\Big((a^{\widetilde{p}_e(j+1)}-1)\gamma - \widetilde{p}_e(j+1)\Big) & \text{Rewriting} \\
= & \sum_{e\in P^*_{j+1}} a^{\widetilde{l}_e(j)}\Big(((1+\frac{1}{2\gamma})^{\widetilde{p}_e(j+1)}-1)\gamma - \widetilde{p}_e(j+1)\Big) & \text{Since } a=1+\frac{1}{2\gamma} \\
\leq & 0 & \text{Lemma 8 with } 0\leq\widetilde{p}_e(j+1)\leq 1
\end{array}$$

□

Theorem 10. *Let $L=\max_{e\in E}\widetilde{l}_e$. For $a=1+\frac{1}{2\gamma}$, $L\leq\mathcal{O}(\log n)$. That is, $\mathcal{A}$ is $\mathcal{O}(\log n)$-competitive.*

Proof. Since $\Phi(0)=m\gamma$ and $\Phi(j+1)-\Phi(j)\leq 0$, we see that $\Phi(j)\leq m\gamma$, for all $j\in\{1,\dots,|\sigma|\}$. Consider the edge e with the highest congestion. Since $\gamma-\widetilde{l}^*_e(j)\in\Omega(1)$, we see that

$$(1+\frac{1}{2\gamma})^L \leq a^L\cdot(\gamma-\widetilde{l}^*_e(j))\leq\Phi(j)\leq m\gamma\leq n^2\gamma$$

Taking log on both sides and manipulating, we get:

$$L\leq(2\log(n)+\log(\gamma))\cdot\frac{1}{\log(1+\frac{1}{2\gamma})}\leq\mathcal{O}(\log n)$$

□

Handling unknown Λ Since Λ is unknown but is needed for the run of $\mathcal{A}$ (to compute c_e when a request arrives), we use a dynamically estimated $\widetilde{\Lambda}$. Let β be a constant such that $\mathcal{A}$ is β-competitive according to Theorem 10. The following modification to $\mathcal{A}$ is a 4β-competitive: On the first request, we can explicitly compute $\widetilde{\Lambda}=\Lambda$. Whenever the actual congestion exceeds $\widetilde{\Lambda}\beta$, we reset[3] the edge loads to 0, update our estimate to $2\widetilde{\Lambda}$, and start a new phase.

- By the updating procedure, $\widetilde{\Lambda}\leq 2\beta\Lambda$ in all phases.
- Let T be the total number of phases. In any phase $i\leq T$, the congestion at the end of phase i is at most $\frac{2\beta\Lambda}{2^{T-i}}$. Across all phases, we have $\sum_{i=1}^{T}\frac{2\beta\Lambda}{2^{T-i}}\leq 4\beta\Lambda$.

References

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[3]Existing paths are preserved, just that we ignore them in the subsequent computations of c_e.