E0 206: Theorist's Toolkit December 24, 2020

Lecture 24-25: Multiplicative Weights Update Method

Instructor: Anand Louis & Arindam Khan *Scribes: Prateek Kumar Sinha & Prajval Koul*

1 Introduction

The multiplicative weights update method is an algorithmic technique most commonly used for decision making and prediction, and also widely deployed in game theory and algorithm design. The simplest use case is the problem of prediction from expert advice, in which a decision maker needs to iteratively decide on an expert whose advice to follow. The method assigns initial weights to the experts (usually identical initial weights), and updates these weights multiplicatively and iteratively according to the feedback of how well an expert performed: reducing it in case of poor performance, and increasing it otherwise.

2 Basic Framework when there is a Perfect expert

Suppose X wants to predict the outcome of games, and has n experts for advice. For each game, each expert gives their opinion on who will win the game. X has to make a prediction based on the experts advice. Suppose there exists an expert who predicts the outcome of each game correctly. If we can somehow find out this expert then we can just follow his/ her advice for the rest of the games. But, how do we find that expert is the question. Let us consider Algorithm 1 for this task.

Algorithm 1: Algorithm for Game result prediction in presence of a Perfect Expert

Initialize: $S^{(0)} = [n]$
for *Game* t **do**
  Take majority opinion of Experts in $S^{(t-1)}$.
  Delete from $S^{(t-1)}$ all the experts who made an incorrect prediction in this game and call the the set $S^{(t)}$.
end

Theorem 1. *Number of mistakes made by X when following Algorithm 1 is at most* $\log \lceil n-1 \rceil$

Proof. If X makes a mistake in round t, then it means that at least half the number of experts predicted incorrectly in that round as we are taking the majority opinion. So, it can be stated that

$$|S^{(t)}| \leq |S^{(t-1)}|/2$$

Thus, every time we make a mistake, the number of experts in the pool becomes half. This can go on for maximum $\log \lceil n-1 \rceil$ time, which proves the theorem. □

3 Strategy in the absence of a Perfect Expert

If every expert in the pool makes some mistake then the above strategy will not work as it assumes that there is at-least one perfect expert. In this setting we assume that that there is no perfect expert and the best expert is the one who makes the least number of mistakes among all the experts. We can think of the following strategies in this case:

- Choose a uniform random expert and follow their advice.
- Take the majority opinion of the experts

- Observe for a few games, then pick the best expert and follow their advise henceforth.

It can be easily seen that the first two strategies cannot work if there are only a few good experts among the pool of experts. The third strategy can not work if some expert predicted correctly in the first few games, and makes very few correct predictions thereafter. So, the idea for this kind of problem is to consider the opinion of each expert weighted by their past performance. Let us consider Algorithm 2 for this setting.

Algorithm 2: Multiplicative Weight Update

Initialize: $w_i^{(0)} = 0$ for each expert i
for *Game t* **do**
  Predict based on the weighted majority of the experts predictions, where expert i gets weight $w_i^{(t-1)}/(\sum_j w_j^{(t-1)})$.
  Update weights: If expert i predicted outcome correctly, then set $w_i^{(t)} = w_i^{(t-1)}$, else set $w_i^{(t)} = (1-\varepsilon)w_i^{(t-1)}$.
end

Theorem 2. *At the end of T rounds, let $M_i^{(T)}$ be the number of mistakes made by expert i and $M^{(T)}$ be the number of mistakes made by Algorithm 2, then for a fixed $\varepsilon \in (0, 1/2]$*

$$M^{(T)} \leq 2(1+\varepsilon)M_i^{(T)} + (2\log n)/\varepsilon,\ \forall i \in [n]$$

Proof. We define a potential function

$$\Phi^{(t)} = \textstyle\sum_i w_i^{(t)}$$

If the algorithm made a mistake in round t, then the weighted majority of experts made a mistake in round t. Therefore

$$\begin{aligned}
\Phi^{(t)} &= \sum_i w_i^{(t)} \\
&\leq (1-\varepsilon).1/2(\sum w_i^{(t-1)}) + 1/2(\sum_i w_i^{(t-1)}) \\
&= (1-\varepsilon/2)(\sum_i w_i^{(t-1)}) \\
&= (1-\varepsilon/2)\Phi^{(t-1)}
\end{aligned}$$

Therefore

$$\Phi^{(T)} \leq (1-\varepsilon/2)^{M^{(T)}},\ \Phi^{(0)} = n(1-\varepsilon/2)^{M^{(t)}}$$

For any i, we have

$$\Phi^{(T)} \geq w_i^{(T)} = (1-\varepsilon)^{M_i^{(T)}}$$

Therefore

$$(1-\varepsilon)^{M_i^{(T)}} \leq n(1-\varepsilon/2)^{M^{(T)}}$$

Therefore

$$M^{(T)} \leq \frac{\log n}{\log 1/(1-\varepsilon/2)} + \frac{\log 1/(1-\varepsilon)}{\log 1/(1-\varepsilon/2)} M_i^{(T)}$$

Using the inequalities

$$\varepsilon/2 \leq \log \frac{1}{1-\varepsilon/2}, \text{ and}$$
$$\log \frac{1}{1-\varepsilon} \leq \varepsilon + \varepsilon^2$$

for small enough ε, we get

$$\begin{aligned} M^{(T)} &\leq \frac{\log n}{\log 1/(1-\varepsilon/2)} + \frac{\log 1/(1-\varepsilon)}{\log 1/(1-\varepsilon/2)} M_i^{(T)} \\ &\leq \frac{\log n}{\varepsilon/2} + \frac{\varepsilon+\varepsilon^2}{\varepsilon/2} M_i^{(T)} \\ &= (2/\varepsilon)\log n + 2(1+\varepsilon) M_i^{(T)} \end{aligned}$$

□

4 Saving a factor of 2

The Multiplicative weight algorithm above can be modified to save factor of 2 in the number of mistakes made by the algorithm. Let us consider the algorithm 3.

Algorithm 3: Multiplicative Weight Update: Saving factor of 2

Initialize: $w_i^{(0)} = 1$ for each expert i
for *Game* t **do**
  Sample an expert i with probability $p_i^{(t)} = \frac{w_i^{(t-1)}}{\sum_j w_j^{(t-1)}}$ and follow their advice.
  Let $m_i^{(t)} = 1$ if the expert i made a mistake in round t, and 0 otherwise. Set $w_i^{(t)} = (1-\varepsilon m_i^{(t)}) w_i^{(t-1)}$ for each i.
end

It can be observed in Algorithm 3 that

$$Pr[\text{Algorithm 3 makes mistake in round } t] = \sum_i p_i^{(t)} m_i^{(t)} = p^{(t)}.m^{(j)}$$
$$E[\text{Mistakes made by Algorithm 3 in } t \text{ rounds}] = \sum_{j\in[T]} p^{(j)}.m^{(j)}$$

Theorem 3. *For a fixed $\varepsilon \in (0, 1/2]$ in Algorithm 3*

$$\sum_{t\in[T]} p^{(t)}.m^{(t)} \leq (1+\varepsilon) \sum_{t\in[T]} m_i^{(t)} + \frac{\log n}{\varepsilon}, \forall i \in [n]$$

Proof. We use the same potential function as used in the last proof

$$\begin{aligned} \Phi^{(t)} &= \sum_i w_i^{(t)} \\ &= \sum_i w_i^{(t-1)}(1-\varepsilon m_i^{(t)}) \\ &= \left(\sum_i w_i^{(t-1)}\right) \sum_i \frac{w_i^{(t-1)}}{\sum_i w_i^{(t-1)}} (1-\varepsilon m_i^{(t)}) \\ &= \Phi^{(t-1)}(1-\varepsilon p^{(t)}.m^{(t)}) \\ &= \Phi^{(t-1)} e^{-\varepsilon p^{(t)}.m^{(t)}} \end{aligned}$$

Therefore,

$$\begin{aligned}\Phi^{(T)} &\leq \Phi^{(0)} e^{-\varepsilon \sum_{t\in T} p^{(t)}.m^{(t)}} \\ &= n e^{-\varepsilon \sum_{t\in T} p^{(t)}.m^{(t)}}\end{aligned}$$

For any expert i, we can write

$$\begin{aligned}\Phi^{(T)} &\geq w_i^{(T)} \\ &= (1-\varepsilon)^{\sum_{t\in[T]} m_i^{(t)}}\end{aligned}$$

Therefore,

$$\begin{aligned}(1-\varepsilon)^{\sum_{t\in[T]} m_i^{(t)}} &\leq n e^{-\varepsilon \sum_{t\in T} p^{(t)}.m^{(t)}} \\ \implies \sum_{t\in[T]} p^{(t)}.m^{(t)} &\leq (1+\varepsilon) \sum_{t\in[T]} m_i^{(t)} + \frac{\log n}{\varepsilon}\end{aligned}$$

which completes the proof. □

5 A General Framework

In the previous sections, we considered the outcome of the game as one of the two participants winning it and $m_i^{(t)} \in \{0,1\}$. More generally, we can think P to be a set of possible outcomes and $m_i^{(t)} \in [-1,1]^n$. In this setting , let's consider the Algorithm 4

Algorithm 4: Multiplicative Weight Update: A General Framework

Initialize: $w_i^{(0)} = 1$ for each expert i
for *Game* t **do**
  Sample an expert i with probability $p_i^{(t)} = \frac{w_i^{(t-1)}}{\sum_j w_j^{(t-1)}}$ and follow their advice.
  Observe $m_i^{(t)}$ and set, $w_i^{(t)} = (1-\varepsilon m_i^{(t)}) w_i^{(t-1)}$ for each i.
end

Theorem 4. *For a fixed $\varepsilon \in (0, 1/2]$, for any expert i in Algorithm 4*

$$\sum_{t\in[T]} p^{(t)}.m^{(t)} \leq \sum_{t\in[T]} m_i^{(t)} + \varepsilon \sum_{t\in[T]} |m_i^{(t)}| + \frac{\log n}{\varepsilon}$$

Proof. We use the same potential function as used in the previous proofs

$$\begin{aligned}\Phi^{(t)} &= \sum_i w_i^{(t)} \\ &= \sum_i w_i^{(t-1)} (1-\varepsilon m_i^{(t)}) \\ &= \left(\sum_i w_i^{(t-1)}\right) \sum_i \frac{w_i^{(t-1)}}{\sum_i w_i^{(t-1)}} (1-\varepsilon m_i^{(t)}) \\ &= \Phi^{(t-1)} (1-\varepsilon p^{(t)}.m^{(t)}) \\ &\leq \Phi^{(t-1)} e^{-\varepsilon p^{(t)}.m^{(t)}}\end{aligned}$$

Therefore,

$$\begin{aligned}\Phi^{(T)} &\leq \Phi^{(0)} e^{-\varepsilon p^{(t)}.m^{(t)}} \\ &= n e^{-\varepsilon p^{(t)}.m^{(t)}}\end{aligned}$$

Now, we lower bound the potential

$$\begin{aligned}\Phi^{(T)} &\geq w_i^{(T)} \\ &= \prod_{t\in[T]} (1-\varepsilon m_i^{(t)}) \\ &\geq \prod_{t\in[T]} e^{-\varepsilon m_i^{(t)} - (\varepsilon m_i^{(t)})^2} \\ &\geq e^{-\varepsilon \sum_{t\in[T]} m_i^{(t)} - \varepsilon^2 \sum_{t\in[T]} |m_i^{(t)}|}\end{aligned}$$

Therefore, using the upper and lower bounds of the potential, we get

$$\begin{aligned}& e^{-\varepsilon \sum_{t\in[T]} m_i^{(t)} - \varepsilon^2 \sum_{t\in[T]} |m_i^{(t)}|} \leq n e^{-\varepsilon p^{(t)}.m^{(t)}} \\ \implies & \sum_{t\in[T]} p^{(t)}.m^{(t)} \leq \sum_{t\in[T]} m_i^{(t)} + \varepsilon \sum_{t\in[T]} |m_i^{(t)}| + \frac{\log n}{\varepsilon}\end{aligned}$$

which finishes the proof. □

We can generalize this framework further by taking value of $m^{(t)} \in [-\rho, \rho]$ and then modify update as $w_i^{(t)} = (1 - \varepsilon \frac{m_i^{(t)}}{\rho}) w_i^{(t-1)}$ for each i. Under these changes, we state the following theorem.

Theorem 5. *For a fixed $\varepsilon \in (0, 1/2]$, for any expert i when $m^{(t)} \in [-\rho, \rho]$*

$$\sum_{t\in[T]} p^{(t)}.m^{(t)} \leq \sum_{t\in[T]} m_i^{(t)} + \varepsilon \sum_{t\in[T]} |m_i^{(t)}| + \rho \frac{\log n}{\varepsilon}$$

Equivalently, dividing throughout by T we can also write

$$\begin{aligned}\frac{1}{T}\left(\sum_{t\in[T]} p^{(t)}.m^{(t)}\right) - \frac{1}{T}\left(\sum_{t\in[T]} m_i^{(t)}\right) &\leq \frac{1}{T}\left(\sum_{t\in[T]} |m_i^{(t)}|\right) + \rho \frac{\log n}{\varepsilon T} \\ &\leq \varepsilon\rho + \rho \frac{\log n}{\varepsilon T}\end{aligned}$$

For $T \geq (\log n)/\varepsilon^2$ and $\varepsilon = min\{1/2, \delta/2\rho\}$

$$\frac{1}{T}\left(\sum_{t\in[T]} p^{(t)}.m^{(t)}\right) - \frac{1}{T}\left(\sum_{t\in[T]} m_i^{(t)}\right) \leq 2\varepsilon\rho \leq \delta$$

6 Regret Minimization

While dealing with scenarios involving expert advice, a metric of interest is *regret*, which is defined as the difference between the performance of the expert(s) chosen, and the performance of the best expert.

Assuming that the algorithm runs for T rounds, the regret at the t^{th} round is defined as

$$regret \ := \ \sum_{t\in[T]} p^{(t)}.m^{(t)} - \min_{i\in[n]} m_i^{(t)}$$

where, the first quantity is the expected penalty incurred by the algorithm in T rounds, and the second term is the optimal penalty over all n experts.

Assuming $m^{(t)} \in [-1,1]^n \ \forall t$, then regret can be bounded, using the previous results.

$$regret \leq \varepsilon \sum_{t\in[T]} |m_i^{(t)}| + \frac{\log n}{\varepsilon} \leq \varepsilon T + \frac{\log n}{\varepsilon}$$

If we know T beforehand, then we can choose ε to achieve desired bounds. For instance, choosing $\varepsilon = \sqrt{\frac{\log n}{T}}$ gives $regret \leq 2\sqrt{T \log n}$.

7 Zero Sum Games

One of the applications of multiplicative weights update paradigm is *Zero Sum Games*. Zero sum games are a variant of simultaneous games with payoffs summing up to 0. This implies that the amount gained by one player is the amount lost by the other. There are two players, called Row player (R) and Column player (C). The action sets for both the players are finite. If R chooses i and C chooses j, then R has to pay an amount $M(i,j)$ to C. Assume that the amounts are normalized, i.e. $M(i,j) \in [0,1] \ \forall i,j$. R tries to minimize the value $M(i,j)$ (the amount it has to pay), and C tries to maximize the same.

The players can either use pure strategies, or mixed strategies. A pure strategy is a strategy wherein a player chooses a fixed action based on his analyses. A mixed strategy is a strategy wherein a player randomizes his choice of sction for different instances. In other words, the player with a mixed strategy plays an action that comes from a certian probability distribution over his action set.The payoff in case of mixed strategy is the expected payoff over the payoff set corresponding to the actions, which is given by the following relation.

$$M(P,Q) := \mathbb{E}_{i\sim P, j\sim Q} M(i,j) = P^T M Q$$

where P is the mixed strategy of the row player, Q is the fixed strategy of the column player, and M is the payoff matrix.

Does knowing the opponent's strategy help in any way to enhance payoff? This is a natural question that arises at this point. One would think that having knowledge about opponent's strategy seems to be a hepful thing. But, it turns out that having such knowledge doesn't help enhance payoff, if the player is already playing an optimal strategy. This is a celebrated result due to von Neumann, also known as the minimax theorem.

$$\lambda^* \ := \ \min_P \max_j M(P,j) = \max_Q \min_i M(i,Q)$$

where, λ^* is the optimal payoff of the row player. λ^* is sometimes also refered to as the *value* of the game.

8 Approximating the value of the game

In this section, we will try to approximate the *value* of a Zero sum game using multiplicative weights update paradigm. If we are somehow able to produce the vector of losses or penalties for the experts, we can iteratively minimize the regret. To achieve this, consider the following setting.

Pure strategies of the row player correspond to experts, and pure strategies of the column player correspond to events. At round t, let $p^{(t)}$ correspond to the probability distribution over the experts. Given this, we have to come up with the loss vector $(M^{(t)})$.

Let $j^{(t)} = \text{argmax}_j\, M(p^{(t)}, j)$ be the optimal strategy for the column player when the row player plays the miexed strategy $p^{(t)}$. The penalty for expert i is defined as $M(i, j^{(t)})$. Using this, we can now update the weights and get new probability distributions in successive iterations. This is the corresponding multiplicative weight update algorithm.

For $T = \Theta(\frac{\log n}{\delta^2})$ iterations, we have,

$$\frac{1}{T}\sum_{t\in[T]}\sum_i p_i^{(t)} M(i, j^{(t)}) \leq \delta + \min_i(\frac{1}{T}\sum_{t\in[T]} M(i, j^{(t)}))$$

where, LHS corresponds to the average expected penalty over the first t rounds.

We will use this relation to approximate the value of the game. Let P^* be the optimal mixed strategy for R. Consider the RHS of the above inequation.

$$\min_i(\frac{1}{T}\sum_{t\in[T]} M(i, j^{(t)})) = \min_t e_i^T M(\frac{1}{T}\sum_{t\in[T]} e_{j^{(t)}}) \leq P^* M(\frac{1}{T}\sum_{t\in[T]} e_{j^{(t)}}) \leq \lambda^*$$

The first equality is valid because $M(i, j^{(t)}) = e_i^T M e_{j^{(t)}}$. The first inequality holds because P^* is a probability distribution and hence a convex combination over the rows (actions of the row player). The LHS of the first inequation is the minimum value for a row. The convex combination of a set can always be bounded from below by its smallest element, and hence the inequality. The second inequality holds because λ^* is the optimal payoff for the row player (and the value of the game), and hence weakly dominates every other payoff.

Let $\hat{P} := (\frac{\sum_{t\in[T]} p^{(t)})}{T})$ denote the average value of $p^{(t)}$ over all rounds, and let $\hat{j} := \text{argmax}_j\, M(\hat{P}, j)$ denote the optimal strategy of the column player when the row player plays $\hat{P}$.

Now, consider the following sequence of results.

$$\lambda^* := \min_P \max_j M(P, j) \leq \max_j M(\hat{P}, j) = \frac{1}{T}(\sum_{t\in[T]} p^{(t)})^T M e_{\hat{j}} = \frac{1}{T}\sum_{t\in[T]} (p^{(t)})^T M e_{\hat{j}}$$

$$\leq \frac{1}{T}\sum_{t\in[T]} (p^{(t)})^T M e_{j^{(t)}} \leq \delta + \min_t(\frac{1}{T}\sum_{t\in[T]} e_i^T M e_{j^{(t)}}) = \delta + \min_i(\frac{1}{T}\sum_{t\in[T]} M(i, j^{(t)})) \leq \delta + \lambda^*$$

The first equality comes from the definition of value of a game. $\min_P \max_j M(P, j) \leq \max_j M(\hat{P}, j)$ holds because LHS promises the minimum payoff possible in the worst case, and the RHS returns a payoff when the row player plays a strategy averaged over the probaility distribution. The following equality holds because the max (for the row player's strategy $\hat{P}$) is obtained when the column player plays $j = \hat{j}$. The equality following this one is trivial. Now, $\frac{1}{T}\sum_{t\in[T]} (p^{(t)})^T M e_{\hat{j}} \leq \frac{1}{T}\sum_{t\in[T]} (p^{(t)})^T M e_{j^{(t)}}$ holds because $j^{(t)}$ is the optimal strategy for column player (for all possibile action profiles of row player). The term on the RHS is the average expected penalty in the first t rounds. It can be bounded by $\delta + \min_t(\frac{1}{T}\sum_{t\in[T]} e_i^T M e_{j^{(t)}})$,

which is further equal to $\delta + \min_i(\frac{1}{T}\sum_{t\in[T]} M(i, j^{(t)}))$. And this value can be upper bounded by $\delta + \lambda^*$, by the previous result.

Therefore, $\hat{P}$ is an approximately optimal strategy for R, because the payoff (by playing the average strategy) lies between λ^* and $\delta + \lambda^*$, which is a close approximation of λ^*.

It should be noted that there are several other ways to find the value of a game. The above method was used to illustrate the application of multiplicative weights update paradigm.

9 Linear Programming

Another application of multiplicative weights update paradigm is Linear Programming. Given a matrix $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, does the following have a feasible solution:

$$Ax \geq b \; and \; x \geq 0$$

This is a specific type of linear program (LP) and we will consider it for simplicity.

Given an error parameter $\delta \in (0, 1/2)$, our goal is to compute an $x \geq 0$ such that $A_i x - b_i \geq -\delta \; \forall i$ (where, A_i is the i^{th} row of matrix A), which is essentially an approximate solution of the LP, or to find a certificate that there doesn't exist a feasible solution for it.

To do this, we again use multiplicative weights update paradigm. Consider this oracle: Given $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$, does there exist an $x \in \mathbb{R}^n$ such that $c^T x \geq d$, and $x \geq 0$? Our algorithm uses this oracle as a black box. It is easy to see that this oracle is trivial to design. *No* is returned only when $c < 0$ and $d > 0$. This will be from where the certificates of infeasibility will be generated.

Now, say the LP has m constraints. We will have m experts, one for each constraint. An event corresponds to a vector $x \geq 0$. Penalty for each expert i is equal to $A_i x - b_i$. We assume that penalty $\in [-\rho, \rho]$, where ρ is some number (its significance will come up later). Recall that in this paradigm, we only need to generate the loss vector for each round. Everything else follows from it.

In round t, expert(s) generates inequalities $\sum_i p_i^{(t)} A_i x \geq \sum_i p_i^{(t)} b_i$ by taking linear combinations of the constraints. This is given to the oracle.

If the oracle returns *infeasible* for the convex combination fed into it, the LP also turns out to be infeasible. This is because if the LP has a feasible solution, then the same solution must also satisfy the convex combination (which was fed into the oracle). By contrapositive, if the convex combination doesn't have a feasible solution, the LP doesn't have a feasible solution as well. This can be used to generate the certificate of infeasibility.

If the oracle returns a point satisfying the constraint fed into it as input, it need not necessarily satisfy the original LP, not even approximately for that matter. Say the oracle returns a point $x^{(t)}$, define $m_i^{(t)} := A_i x^{(t)} - b_i$. Update weights accordingly and repeat.

The main idea here is in the definition of $m_i^{(t)}$. If $A_i x^{(t)} < b_i$ for a particular constraint, i.e. it was not satisfied, then its weight will be increased in the next round. By doing so, the oracle will try to satisfy this constraint "a bit more" in the following rounds, by giving it more weightage (the weight). Similarly, if $A_i x^{(t)} > b_i$ for some constraint, i.e. it was satisfied, its weight is decreased in the next round.

Now, we run this algorithm for $T = O(\frac{\rho^2 \log n}{\delta^2})$ rounds. Assuming that no infeasibility is detected for T

rounds, then, for each i, we can say that:

$$0 \leq \frac{1}{T} \sum_{t \in [T]} (\sum_{i} p^{(t)} (A_i x^{(t)} - b_i)) \leq \delta + \frac{1}{T} \sum_{t \in [T]} (A_i x^{(t)} - b_i)$$

The second term is the average expected penalty of the algorithm, and is non negative. The third term is the average penalty for expert i. Equivalently, for each i, taking the first and third term, and taking the summation inside the term, we get:

$$-\delta \leq A_i (\frac{\sum_{t \in [T]} x^{(t)}}{T}) - b_i$$

i.e. b_i subtracted from A_i times the average value of $x^{(t)}$, is not less than $-\delta$. Therefore, $\frac{(\sum_{t \in [T]} x^{(t)})}{T}$ approximately satisfies all constraints.

Clearly T depends on ρ, which further depends on the problem instance at hand. This ρ can be used to fine tune the running time of the algorithm, and hence plays a crucial role

References

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012.

[2] Jonathan Kelner and Dimiter Ostrev. An algorithmist's toolkit: Lecture 24. `https://ocw.mit.edu/courses/mathematics/18-409-topics-in-theoretical-computer-science-an-algorithmists-toolkit-fall-200lecture-notes/MIT18_409F09_scribe24.pdf`.

[3] Jonathan Kelner and Nikolaos Trichakis. An algorithmist's toolkit: Lecture 25. `https://ocw.mit.edu/courses/mathematics/18-409-topics-in-theoretical-computer-science-an-algorithmists-toolkit-fall-200lecture-notes/MIT18_409F09_scribe25.pdf`.

[4] Anupam Gupta and Shiva Kaul. Lecture 16: The multiplicative weights algorithm. `https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture16.pdf`.

[5] Anupam Gupta and Tim Wilson. Lecture 17: Solving lps/ sdps using multiplicative weights. `https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/notes/lecture17.pdf`.

[6] Sanjeev Arora. Lecture 8: Decision making under total uncertainity: the multiplicative weight algorithm. `https://www.cs.princeton.edu/courses/archive/fall16/cos521/Lectures/lec8.pdf`.