# Lecture 16/17

## 1 Overview

In these lectures, we discussed the Multiplicative Weights algorithm in the experts framework and the application of it to Zero-Sum Two Person games, Linear Programs, and finally to the path routing problems that we previously discussed.

The lecture is pieced together from the Arora, Hazan, Kale notes linked on the webpage. Please refer to that for details. We sketch the ideas here.

## 2 Experts and Multiplicative Weights

The experts problem consists of $n$ experts who each make a prediction every day. Everyday, you must also make a preductions after seeing all of the expert's predictionss. At the end of the day the true result is revealed and ou lose if you predicted wrong. We would like to lose as little as if we had always listened to the single best expert over all the days. One view of the experts is that they are forecasting something, perhaps the weather. Moreover, each day we do not know a priori what the accuracy of each expert is.

A version of the multiplicative weights algorithm is to choose an expert $i$ with probability proportional to a weight $w(i)$ ($p_i = w(i). \sum_i w(i)$). The weights, $w(\cdot)$, are initialized to 1 for each expert, and on any day that it suffers a loss we reduce its weight by a factor of $(1 - \epsilon)$.

*Question 1: Show that the expected loss of the algorithm over $T$ days is at most $(1 + \epsilon)m_i^T + O(\ln n/\epsilon T)$, where $m_i^T$ is the total loss incurred by expert $i$. (That is, the algorithm does almost as well as any expert.)*

Arora, Hazan, and Kale show that choosing the highest weight expert instead gives a loss that is bounded by $2(1 + \epsilon)m_i^T + O(\log n/\epsilon T)$. In class, we showed the algorithm of choosing according to the distribution yields above result. (That is, we presented the answer to Question 1 in class. Feel free to reproduce it as your answer.)

For intuition, I like the description in http://www.cs.cmu.edu/ãvrim/ML09/lect0114.pdf, pages 3 to 5 (or in fact the whole lecture is great.)

## 3 Two Person Game Framework

A zero sum two person game can be viewed as a matrix where given a pair of strategies $s_1, s_2$ person one pays person two $A(s_1, s_2)$. Sometimes $A$ is called the payoff matrix. The numbers could be positive or negative.

An equilibrium position are distributions over strategies for players 1 and 2, where neither can benefit by changing their distribution. The Famous Minimax Theorem for Zero Sum Two person games of Von Neuman shows that there is such an equilibrium point for zero sum two person games and that its value is equal to the best response of player 1 to any strategy for player 2 and the best response of player 2 to any strategy of player 1. This could be shown using linear programming duality. But the notes in Arora, Hazan, and Kale show it using the multiplicative weights algorithm.

The way to set it up is to view all the strategies of player 1 as experts, and each day player 2 provides the best (or at least a good) response to the current strategy (as defined by the multiplicative weights algorithm). (We assume the payoffs are 0/1.) After $T$ days, the final distribution over player 1 strategies gets close the best response against the average over those days player 2 strategy. Moreover, the player 2 strategy get the best response every day against the player 1 strategy. Thus, we have given a pair of strategies where each player's best response's are close (and converging as $T$ gets large). This is the notion of equilibrium. This, moreover, gives an algorithm to find a point that is close to equilibrium.

## 4 Linear Programs

One can view the feasibility of linear programs as a two person game where the primal wishes to find a point where every inequality is satisfied, and the dual tries to find an inequality that is not satisfied.

That is, the primal needs to find a solution vector where for any positive weighted sum of the inequalities the weighted inequality is satisfied.

Thus, we can set this up in the expert's framework where the inequalities are the "experts", the expert loses when the inequality is satisfied, and the primal "plays" strategies that satisfy the current weighted sum the inequalities (where the weights are given by the multiplicative weights algorithm.)

If the primal cannot find a point for a single weighted sum of inequalities, then the primal is infeasible. If the primal succeeds for $T$ steps every inequality is almost satisfied.

## 5 Generalizations

The algorithms above dealt with $[0, 1]$ losses.

An algorithm for gains can be done as well, where the gain of the algorithm is at least

$$(1 - \epsilon)m_i^t - O(\log N/\epsilon).$$

In fact, we can generalized the situation to $[-\rho, \rho]$ losses with some changes to the bounds. This can be done in a variety of ways, and lead to slightly different error bounds. Again, see AHK for a fuller description.

## 6 Path Routing

Here we wish to find a solution to the path routing problem from the previous lecture. That is, given $G = (V, E)$, and $(s_1, t_1), \ldots, (s_k, t_k)$, find paths connecting each pair that minimize the congestion; i.e., the maximum number of paths that use any edge.

The experts try to predict which edges are congested (and gain when they do.) That is, the gain on the edge is the congestion. In each step, we provide the best response to the current weighting under the expert's algorithm; i.e., route all the paths along shortest paths. Note that this solution minimizes the average congestion and thus minimizes the gain of the current weighted sum of experts.

Output the average of these solutions after $T$ rounds. By noting that the maximum congestion in one step is $k$, and by choosing $\epsilon = \delta/2$ we can conclude that for some $T = O(k \log n/\delta^2)$ rounds that the average solution must get $(1+\delta)$ close to the optimal solution. Intuitively, we get the $O(\log n/\delta^2)$ if we had a $[0, 1]$ gains. The factor of $k$ is due to having to average out the one time expense of hitting a path $k$ times.

Each round takes time $O(km \log n)$, thus we have an $O(k^2 m \log^2 N)$ algorithm.

In fact, by routing a single path in each round, introducing experts which benefit from not having routed enough of one path, and mixing gains and losses and one can improve the running time to $O(km)$ times polylogarithmic factors. That is, the time to route paths that minimize congestion becomes comparable to the time to simply find any set of $k$ paths among the pairs!

## 7 Other Applications.

In class, I neglected to mention a couple of important applications. One is something called "Boosting". This is a method broadly used in machine learning to combine algorithms. Another area is in complexity. There they show that if functions are a little difficult to compute, that there must be a set of inputs where they are very difficult to compute. Both of these directions are discussed by Arora, Hazan, and Kale.