Algorithms, Games, and Networks January 17, 2013

Lecture 2

Lecturer: Avrim Blum *Scribe: Aleksandr Kazachkov*

1 Readings for today's lecture

Today's topic is **online learning, regret minimization, and minimax optimality**. The corresponding readings are **Sections 4.1-4.3** in the AGT book.

2 Recap from last time

First we recap from the end of last class. We ended with the proof that Nash equilibria (NE) exist for games with two players and where each player has a finite number of pure strategies (actions). Recall that a Nash equilibrium (in a two-player game) is a *pair* of (possibly randomized) strategies, which is *stable* in the sense that each is a best response to the other in terms of expected payoff. Thus, in a NE, neither player has any incentive to deviate given that the other player does not change his strategy.

Let's go over the proof slightly. We made use of Brouwer's fixed point theorem:

Theorem 1 (Brouwer fixed point theorem) *Every continuous function f from a convex compact subset S of $\mathbb{R}^n$ to S itself has a fixed point: an $x \in S$ such that $f(x) = x$.*

Noting that the set

$$S = \{(p,q) \in \mathbb{R}^n_+ \times \mathbb{R}^n_+ : \sum_{i=1}^n p_i = 1;\ \sum_{i=1}^n q_i = 1\}$$

of pairs of mixed strategies is a convex compact set, we just had to define a continuous function from pairs of strategies back to pairs of strategies such that every fixed point would be a NE.

Our initial attempt, putting f as the best response function

$$f(p,q) = (p', q'),$$

where p' is the best response to q, q' is the best response to p, failed because of discontinuity (and not being well-defined). We succeeded in our second attempt (see the first lecture's notes), by putting a quadratic penalty for changing your strategy, which gave us a well-defined maximum and a continuous function. We then showed the fixed points of the function corresponded to NE, completing the proof via Theorem 1.

We proved NE exist, but we did not discuss how to find them. We can solve for minimax optimal strategies using linear programing, but we cannot do this for NE.

3 High level overview

A Nash equilibrium is a static concept, giving a set of probability distributions such that no player has any incentive to deviate given that the other players keep their strategies fixed. Today we will talk about dynamics. Suppose, as a player, you find yourself repeatedly making decisions from a finite space of actions in a changing and unpredictable environment. We will find there are ways that you can adapt (learn) that will provide you with pretty strong guarantees on performance no matter what the world may do. This then brings up the question: what happens when all players adapt in such ways? Unfortunately, we won't get what we would ideally like: namely, convergence to a Nash equilibrium. However, we will find that in *zero-sum* games, behavior will approach minimax-optimal strategies. In fact, this will yield an alternative proof of the minimax theorem from last time.

4 No regret

Consider the following setting: each morning, you need to pick one of N possible routes to drive to work via a road network, but traffic varies from day to day. Suppose each day, you take a route and record how long it took you that day. In some versions of the problem, you might also be able to get information about how long other routes would have taken that day. So your move each day is to choose a route, and the world's move is to choose the cost vector over all routes, so your cost is then exactly the cost of the route you chose.

Question: Is there a strategy for picking routes so that in the long run, whatever the sequence of traffic patterns has been, you've done nearly as well as the best *fixed* route you could have picked in hindsight? (In expectation, over internal randomness in the algorithm.)

Thus, we have in this way set what we are comparing our algorithm against – a fixed route. This is a standard objective which uses the notion of *external regret*, where we want to do well when measured against an optimal static offline policy. External regret is also known as the problem of *combining expert advice.* The regret of our algorithm is the difference between what our algorithm gets us and what we would have gotten from following any single expert. Other notions of regret exist, such as *internal* or *swap regret*, which gives a different policy against which we measure the performance of our algorithm.

This leads us to explore "no-regret" algorithms for repeated decisions. An algorithm has N options, and the world chooses a cost vector. We can view this as a matrix as shown in

Table 1, where potentially there may be an infinite number of columns:

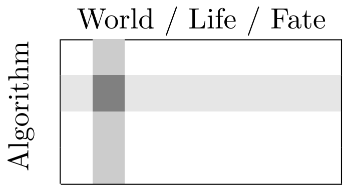

Table 1: Payoff matrix for generic adaptive setting

At each time step, the algorithm picks a row, and the world picks a column. The algorithm pays the cost for the action chosen and it gets the column as feedback (or maybe just its own cost, if we are in the "bandit" model).

We need to assume that some bound on the maximum cost exists in this payoff matrix, since you don't want it to be that some cost leads you to the equivalent of falling off a bridge in the "driving to work" game. By normalizing, we can say all costs are between 0 and 1, which we can do because multiplying all the costs by a constant and adding a constant to all terms does not affect the optimal strategies to a game.

Definition 2 *Let $\mathcal{R}$ denote the set of rows (actions the algorithm has to choose from). The* *average regret* *over T time steps of an algorithm π with cost C_t^π at each time step is*

(average per-day cost of algorithm) - (average per-day cost of the best fixed row in hindsight), or

$$\frac{1}{T}\sum_{t=1}^{T} C_t^\pi - \min_{r\in\mathcal{R}} \frac{1}{T}\sum_{t=1}^{T} C_t^r.$$

We want this to go to zero or better as T gets large – this is called a no-regret algorithm. Whatever the sequence of events, we want to do (at least) as well as the best fixed row in hindsight. Note that we can indeed in general do better via our algorithm than any single row may be capable of achieving.

4.1 Small example

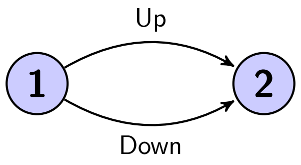

Figure 1: Graph of the small example

Let's look at a small example to get some intuition about the properties of no-regret algorithms. This is our "going to work" game, where we have only two routes to choose from each day, and the world chooses to make exactly one of these routes congested.

$R \backslash C$	Up	Down
Up	1	0
Down	0	1

Table 2: Cost matrix for the small example

We will denote our algorithm by R, since we view it as the row player, and the world will be the column player, C. Figure 1 shows the routes to work we can choose each day: we can either go Up (U) or Down (D). The world then puts a cost of either 1 on U or 1 on D, yielding the costs (to us) shown in Table 2. Essentially, we are looking at the matching pennies game viewed as a road network.

One way of viewing this game is that the world has chosen some sequence $UDUDDU\ldots$, and our goal is to do "well" (in expectation) regardless of the sequence that is chosen. It is clear that to have any hope of succeeding in attaining no regret, any algorithm we use *must* be randomized.

This example highlights why we do not try to compete with the best adaptive strategy – the best *sequence* in hindsight would have zero cost, since it would simply choose the route with no cost each time, whereas the world can force us to incur a cost of T after T time steps. We are only comparing ourselves against the best *fixed* path in hindsight.

In general, Theorem 4.1 in the AGT book shows that if we try to compare our algorithm against an adaptive strategy, then over T time steps, there exists a sequence that gives us a regret of at least $T(1 - 1/N)$, where N is again the number of actions available.

Additionally, we can see from this example that no-regret algorithms can sometimes do much better than playing minimax optimal. (And they will never do much worse. If we are doing nearly as well as any given row, then we are doing nearly as well as any distribution, or weighted average, over rows.) Playing U half the time and D half the time for both R and C is minimax-optimal here, but suppose ninety percent of the days the bottom route is congested while the top one is free. We will have gone the congested route fifty percent of the time, but we would have regret, as a better solution would have placed more weight on U.

These are two different settings thus: if the world is antagonistic, then we could play minimax optimal, but if it is not, we may be able to do better.

4.2 Abridged history

- Hannan'57, Blackwell'56
  - Algorithm with regret $O((N/T)^{1/2})$.
  - Re-phrasing, this bound implies that we need only $T = O(N/\epsilon^2)$ steps to get time-average regret down to ϵ. We will call this quantity T_ϵ.

  - This algorithm gives optimal dependence on T (or ϵ). As a result, game theorists viewed the problem as essentially solved, since the number rows N was regarded as a constant.

- Littlestone-Warmuth'89
  - Imagine a large class C of N prediction rules. We want to perform nearly as well as the best function in our class.
  - The led to the weighted majority algorithm, which we will see in the next section.
  - $\mathbb{E}[Cost] \leq OPT(1+\epsilon) + (\log N)/\epsilon$.
  - This yields a regret of $O((\log N)/T)^{1/2}$, implying $T_\epsilon = O(\log N/\epsilon^2)$:

$$\text{Regret} = \frac{1}{T}\left(\mathbb{E}[Cost] - OPT\right) \leq \frac{1}{T}\left(\epsilon + \frac{\log N}{\epsilon}\right) = \frac{\epsilon}{T}\left(1 + \frac{\log N}{\epsilon^2}\right).$$

  - Also optimal as a function of N.

- Extensions to bandit model (adds extra factor of N).

Let us make a note of why the Hannan and Blackwell algorithm is optimal in T. Returning to the matching pennies road game in Table 2, say the world flips a fair coin each day. Thus, whatever your algorithm, your expected cost in T steps is $T/2$. On the other hand, the minimum of the number of heads and tails we expect to see is not $T/2$. The standard deviation is roughly $\sqrt{T}$ so the expected value of the minimum is closer to $T/2 - \sqrt{T}$. Thus, per day, our gap is $O(T^{1/2})$, meaning the stated bound is optimal when fixing N.

4.3 Combining expert advice: weighted majority algorithm

Now we talk about the problem of combining expert advice. Say we want to predict the stock market. We poll N "experts" for their advice (will the market go up or down). Note, "expert" is simply used to refer to someone offering an opinion. We want to use the advice of these experts to somehow to make our prediction.

Expt 1	Expt 2	Expt 3	dog	truth
down	down	down	up	up
down	up	down	down	down
...	...	...	...	...

Table 3: Experts and their predictions

Is there a strategy that allows as to do nearly as well as the best of these experts in hindsight?

Let us start with a model in which we know we have an expert that makes no mistakes.

Question: Can we find a strategy that combines the strategies of the experts and makes no more than $\lg N$ mistakes?

Solution: We take the pure majority from those who have not made a mistake yet, and whenever an expert makes a mistake, he gets eliminated. Guaranteed, every time you make a mistake, at least half the experts are cut down, so at most $\lg N$ mistakes are made.

But what if no expert is perfect? We can run the same protocol until all experts are eliminated, then restart, but this gives us $\lg N$ mistakes *per each mistake* of the best expert. This seems wasteful – when there is no perfect expert, intuitively, making a mistake should not completely disqualify an expert.

Weighted majority algorithm

1. Start with all experts having weight 1: set $w_i^1 = 1$ for all $i \in [1, N]$.
2. Predict based on weighted majority vote: if at time step t, each expert has a hypothesis $h_i^t \in \pm 1$, then predict $\operatorname{sgn} \sum_{i=1}^n h_i^t w_i^t$.
3. Penalize mistakes by cutting weight in half: if v^t is the true value at time t, then put $Corr = \{i : h_i^t = v^t\}$ and $Incorr = \{i : h_i^t \neq v^t\}$, then update weights by $w_i^{t+1} = w_i^t$ for all $i \in Corr$ and $w_i^{t+1} = w_i^t/2$ for all $i \in Incorr$.

4.4 Analysis of the weighted majority algorithm

The claim is we will do nearly as well as the best expert in hindsight with this algorithm.

Let M be the number of mistakes we have made so far, and let m be the number of mistakes the best expert has made so far. Let W be the total weight (initially, $W = \sum_{i=1}^n w_i^1 = N$).

Claim: after each mistake, the total weight of the system, W drops by at least 25%. To help visualize this, we can draw a pie chart: look at Figure 2.

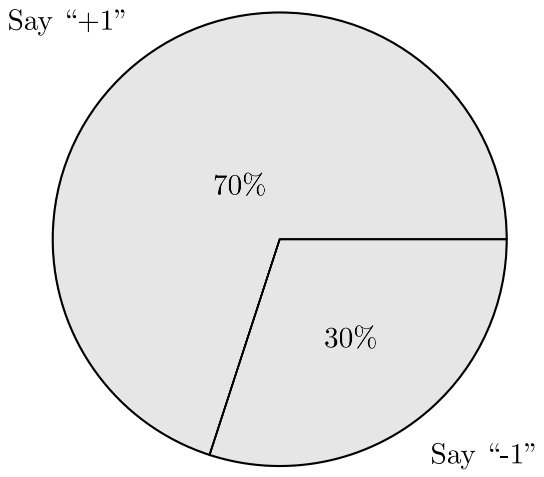

Figure 2: Visualizing the weighted majority algorithm

Whenever we make a mistake, this means more than fifty percent of the weight gave the wrong advice. Thus, at least fifty percent of the weight is penalized by half, so at least a quarter of the current total weight is cut. Of course, we can also get lucky, and it can drop when we don't make a mistake, but note the weight never goes up.

Thus, after M mistakes, W is at most $N(3/4)^M$. On the other hand, the best expert has weight $(1/2)^m$. So since the total weight must be at least the weight of the best expert, since all weights are nonnegative,

$$\begin{aligned}(1/2)^m &\leq N(3/4)^M \\ (4/3)^M &\leq N2^m \\ M &\leq 2.4(m + \lg N) \quad \text{(a constant ratio)}\end{aligned}$$

So if the best expert is perfect, we only do 2.4 times worse than the previous algorithm.

If m is small, so is M. However, if the best expert makes a lot of mistakes, can we do better? If the best expert makes more than 25% of the time, we end up making a mistake half the time in the worst case.

Thus, can we do better? We can, by using randomization in step 2 of the weighted majority algorithm, by viewing the weights as probabilities (e.g., if .7 on up, .3 on down, then pick up with 70% chance, and down with 30% chance). The idea is to smooth out the worst case. Also we can generalize the 1/2 by which we cut weights in step 3 of the algorithm to $1 - \epsilon$.

The intuition is the only way the world can make it so we make a lot of mistakes is to make a lot of the experts be wrong, but then we end up making a lot of progress at each mistake.

Another algorithm for choosing what to pick at each step, instead of using total weight, is to just pick a random expert proportional to its weight, and doing whatever it says. Note that this will lead to the same probability of choosing up or down at each step, since we can view the above pie chart as more granulated, with slices corresponding to the amount of weight contributed by each expert.

4.5 Analysis of randomized weighted majority algorithm

The algorithm we are analyzing is called randomized weighted majority (RWM). We will show, where M is the expected number of mistakes and now n is the number of experts,

$$M \leq \frac{-m\ln(1-\epsilon) + \ln n}{\epsilon} \approx (1 + \epsilon/2)m + \frac{1}{\epsilon}\ln n, \tag{1}$$

where the approximation comes from a Taylor expansion.

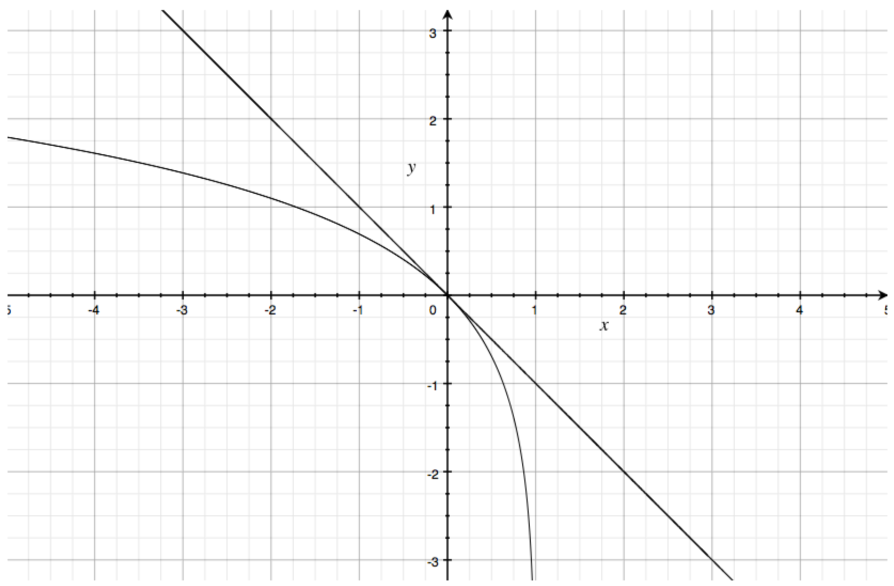

Figure 3: Illustrating that $\ln(1-x) \leq -x$

Plugging in some numbers:

$$\epsilon = 1/2 \Rightarrow M \leq 1.39m + 2\ln n$$
$$\epsilon = 1/4 \Rightarrow M \leq 1.15m + 4\ln n$$
$$\epsilon = 1/8 \Rightarrow M \leq 1.07m + 8\ln n$$

Unlike most worst-case bounds, these numbers are pretty good.

To begin the analysis, say at time t we have a fraction F^t of weight on experts that made a mistake. Thus, F^t is the probability we make a mistake at time t, and we remove an ϵF_t fraction of the total weight (exactly). We can calculate the exact final weight we will see:

$$W^{\text{final}} = n(1-\epsilon F^1)(1-\epsilon F^2)\cdots .$$

Taking logs,

$$\ln W^{\text{final}} = \ln n + \sum_t [\ln(1-\epsilon F^t)] \leq \ln n - \epsilon \sum_t F^t,$$

where we used the fact that $\ln(1-x) \leq -x$ (you can draw a picture to see this, see Figure 3). But now we have a nice interpretation of $\sum_t F_t$ – the sum over all the time steps of the probability we make a mistake at time t. This is the expected total number of mistakes, so

$$\ln W^{\text{final}} \leq \ln n - \epsilon M.$$

On the other hand, if the best expert makes m mistakes, then

$$\ln W^{\text{final}} > \ln((1-\epsilon)^m).$$

Now solving for M in $\ln(n) - \epsilon M > m\ln(1-\epsilon)$, we get the desired bound stated in Equation 1. We can interpret the $\frac{1}{\epsilon}\log n$ term as kind of how quickly we are learning.

Summarizing,

- The expected number of mistakes is $\mathbb{E}[M] \leq (1+\epsilon)m + \epsilon^{-1}\log(n)$.
- If we set $\epsilon = (\log(n)/m)^{1/2}$ to balance the two terms out, we get a bound of

$$\mathbb{E}[M] \leq m + 2(m \cdot \log n)^{1/2}.$$

- Then, since $m \leq T$, this is at most $m + 2(T\log n)^{1/2}$. So our average regret per day is that quantity divided by T.
- We then get average regret $= \frac{1}{T}(\mathbb{E}[M] - m) \leq \frac{2(T\log n)^{1/2}}{T} \to 0$.

5 Applications

What can we use the previous results for? We can now combine multiple algorithms to do nearly as well as the best algorithm in hindsight.

1. We can also consider cases like choosing paths to work, where "experts" are different actions, not different predictions. In this game theoretic version, at each time t, each action has a cost in {0,1}. We can still run the same algorithm: rather than "picking a prediction with probability proportional to its weight," we view it as "picking an expert with probability proportional to its weight." Thus, we choose expert i with probability $p_i = w_i^t/W^t$, where w_i^t is the weight of expert i at time t, and $W^t = \sum_i w_i^t$. As mentioned before, this leads to the same probability of making a decision.

Note that the size of this representation is quite large, such as when we consider each action as a different path in a network.

2. Similarly, if costs are not strictly binary, but in $[0,1]$, if expert i (where again the expert is an action, such as the choice of a route to work, for instance) has cost c_i^t at time step t, one natural generalization of the previous approach is to adjust weights in step 3 by $w_i^{t+1} \leftarrow w_i^t(1 - c_i^t\epsilon)$ (see the example in Table 4).

Our expected cost is then $\sum_i c_i^t w_i^t/W^t$. On the other hand, the amount of weight removed from the system is $\epsilon\sum_i c_i^t w_i^t$. So the fraction of total weight removed is ϵ times our expected cost, and the rest of the proof continues as before. We get a guarantee of

$$\mathbb{E}[Cost] \leq OPT + 2(OPT \cdot \log n)^{1/2}.$$

Since $OPT \leq T$, this is at most $OPT + 2(T\log n)^{1/2}$. Our average regret is then again

$$\text{Average regret} = \frac{1}{T}(\mathbb{E}[Cost] - OPT) \leq \frac{2(T\log n)^{1/2}}{T} \to 0 \text{ as } T \to \infty.$$

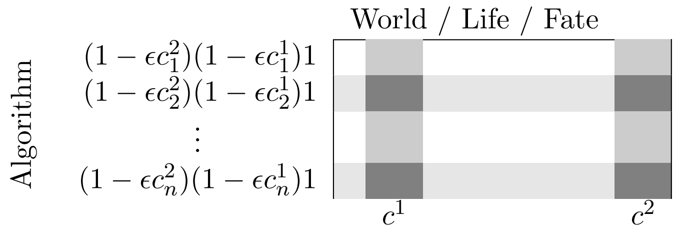
World / Life / Fate
Algorithm
$(1-\epsilon c_1^2)(1-\epsilon c_1^1)1$
$(1-\epsilon c_2^2)(1-\epsilon c_2^1)1$
$\vdots$
$(1-\epsilon c_n^2)(1-\epsilon c_n^1)1$
c^1
c^2

Table 4: Illustration of case when costs are in $[0, 1]$. World first chooses column with costs c^1, then column with costs c^2. The rows are labeled with their updated weights.

"So now we can drive to work, assuming feedback is available." Note that for this algorithm, we do need to know how the other experts are doing (the feedback) to know how to penalize the other experts. If we just find out the cost of our own route to work, the method has to be adjusted.

5.1 Connections to minimax optimality

In the following, let R be the row player and C be the column player. Recall von Neumann's minimax theorem. We will prove it based on the idea that no-regret strategies will do nearly as well or better than minimax-optimal strategies against any sequence the opponent plays. Indeed, we do nearly as well with our no-regret strategy as the best fixed row in hindsight, which implies we do nearly as well as the best distribution over rows in hindsight, which implies we do nearly as well as minimax-optimal.

Theorem 3 (von Neumann, 1928) *All two-player zero-sum games have a unique* **value** *V. The minimax optimal strategy for R guarantees R's expected gain is at least V, while the minimax optimal strategy for C guarantees C's expected loss is at most V.*

Proof: Assume for the sake of contradiction that the theorem is false. This means that some two-player zero-sum game has $V_C > V_R$. If C commits first, then there exists a row that gives R at least V_C, but if R commits first, then C can force R to get only V_R. Rescale so that the payoffs to R are in $[-1, 0]$, and say that $V_C = V_R + \delta$.

Now consider playing the randomized weighted majority algorithm as the row player against the column player, who plays optimally against the row's distribution.

In T steps:

(1) From the bounds we proved before, our algorithm gets at least

$$\text{BRiH} - 2(T \log n)^{1/2},$$

where $OPT = \text{BRiH}$ is the gain of the best row in hindsight.

(2) Also, $\text{BRiH} \geq T \cdot V_C$ (it is best against opponent's empirical distribution).

(3) On the other hand, our algorithm gets at most $T \cdot V_R$ (each time, your opponent knows your randomized strategy).

Combining these together, (2) and (3) have a gap of δT. But this contradicts (1) once $\delta T > 2(T \log n)^{1/2}$, or $T > 4 \log(n)/\delta^2$.

Thus, our assumption was wrong, proving the minimax theorem.

■

This proof also gives a procedure for computing a minimax optimal strategy relatively quickly, if we can simulate C's best response quickly.

5.2 Other applications

We consider two RWMs playing each other. The time-average (meaning, how often over time does the row player play row 1, row 2, etc.) strategies approach minimax optimality.

Let's consider the "Smuggler vs. border guard" game: we are given a graph $G = (V, E)$, source s, sink t. The smuggler chooses a path, and the border guard chooses an edge to watch. If the edge being guarded is on the path, the guard wins; otherwise, the smuggler wins.

For the border guard, the minimax optimal strategy is to find a minimum cut, and pick an edge from the uniform distribution over edges in the cut. The smuggler, on the other hand, needs to find a max flow; after scaling to a unit flow, this gives a probability distribution on paths. That is, if there are some k paths $P_1, \ldots, P_k$ used in the max flow, then after the normalization, each path P_i sends some positive flow f_i from s to t, such that the sum of the flows is 1; thus the f_i values can be seen as probabilities.

The latest fast approximate max-flow algorithms are based on applying RWM to variations on this game. We run RWM for the border guard, where the experts are the edges, and we get the best response by solving a linear system or shortest path problem.