NIKA HAGHTALAB AND MICHAEL JORDAN

CS294: DECISIONS, LEARNING AND GAMES

Preliminaries

- **Domain (Instance space):** An arbitrary set $\mathcal{X}$ that includes all possible instances.
- **Labels:** A set $\mathcal{Y}$ that includes all possible labels or predictions for a single instance, such as $\{0, 1\}$, $\{-1, 1\}$, $\{false, true\}$, etc.
- **Labeled instance:** An instance-label pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is called a labeled instance.
- **Hypothesis:** A *hypothesis* (a.k.a classifier or predictor) is a function $h : \mathcal{X} \to \mathcal{Y}$.
- **Hypothesis class:** A concept class $\mathcal{H}$ is a pre-determined set of concepts.
- **Consistency:** We say that a hypothesis $h \in \mathcal{H}$ is *consistent* with a set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, if for all $i \in [m]$, $h(x_i) = y_i$.

Lecture 1

Some Examples

In this section, we give examples of how even simple decisions have consequential implications, have to rely on learning in difficult and complex situations, and contribute to rational behavior that emerges in groups and the society. A common theme in these examples is they involve uncertainty, learning, and decentralized agency.

Example (Which route to use for your daily commute?). Everyday we need to decide on the route P^t we take to work. We want to choose a route that is the shortest (accounting for traffic), but we don't know the traffic ahead of time. In fact, in every day t, the total driving time for route P, denoted by $\text{cost}^t(P)$, may be different from what the traffic looked like in the past, e.g., due to road accidents, first day of school, and changes in the routes other drivers take. So, we want to have a strategy for picking the route P^t that leads to a short commute on average.

The average time spent on commute over T days is $\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^t)$. After T days, we can look back at the historical traffic data and compute the (fixed) route that would have had the best average commute time. The average commute time of the best fixed route in hindsight is $\min_{P*}\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^*)$. Can we have an algorithm for picking P^t that is almost as good as the best route in hindsight, i.e.,

$$\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^t) \leq \min_{P^*}\frac{1}{T}\sum_{t=1}^{T}\text{cost}^t(P^*) + \epsilon,$$

for an $\epsilon \to 0$?

Challenges: When the commute length (traffic) is unpredictable and possibly designed adversarially, we don't know which paths may be good in the future. It's possible that historically well-performing paths don't do well in the next round. So, we need to hedge our bets and *"explore" paths that might have not performed well historically*. We need to balance this off with "exploiting" paths that did perform well, because their average performance will remain good for some time.

Example (Route planners). You could use a route planner, like Google Maps, Apple Maps, or Waze to plan your commute. These planners run a similar "explore-exploit" algorithm in the background not only to hedge their bets but also to gain more information about the traffic as they monitor drivers in action. If this exploration suggest routes that a driver believes to be longer than other alternatives, the driver may benefit from switching his path and not following the instructions. Over the long run, this affects the quality of observations the app may receive. The apps may even lose market share if they use algorithms that explore too carelessly.

Challenges: Can we design explore-exploit algorithms that don't suggest obviously bad routes? Does the competition for market-share between several apps impact their explore-exploit tradoffs and overall learning quality?

Example (Impact of route planning at scale). The commute time and traffic is impacted by the decisions of all other drivers. Often, the more drivers on one road, the longer the commute time. So, cost(P) is formed by the impact of all driver's choices. Assume that 1000s of drivers, which we refer to as one unit of traffic, want to go from A to B. And the congestion cost of an edge, denoted by $c_e(x)$, is a function of the fraction of drivers x that that take edge e.

A social planner, who tells each person which route to take, can plan an optimal routing, i.e., the least time the society spends on commute! In the above example, when half of the drivers take AVB and the other half take AWB, then commute time is just 1.5. This allocation is also individually optimal to each player. That is, no driver can switch to a different route and reduce their own commute time. Such an allocation is called an "equilibrium".

What if the city adds a bridge between V and W, with congestion cost of $c(x) = 0$? While the previously socially optimal solution is still a valid solution with commute time of 1.5, this solution is no longer an equilibrium because players benefit from deviating from their current path and taking the bridge. In fact, the only equilibrium in this case is a solution where all drivers take the bridge and pay the commute cost of 2.

Challenges: Computing socially optimal solutions and equilibria need a lot of information and coordination. Can these be achieved by simple learning algorithms? e.g., what if each diver deploys an online learning algorithm (from Example 1) or simply picks the (instantaneous) shortest route? Can learning dynamics help us arrive at good solutions or are they inherently different from the type of coordination needed for finding these solutions?

Other examples of applications our theoretical framework needs to

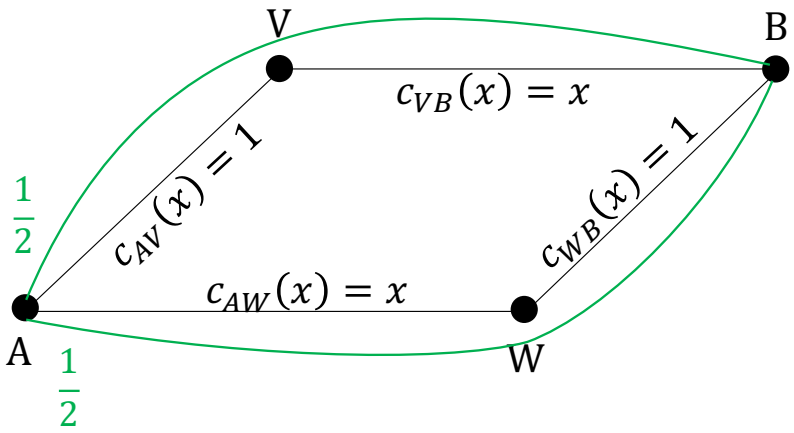


Figure 1: The green allocation is the social optima and an equilibrium. It has a cost of 1.5

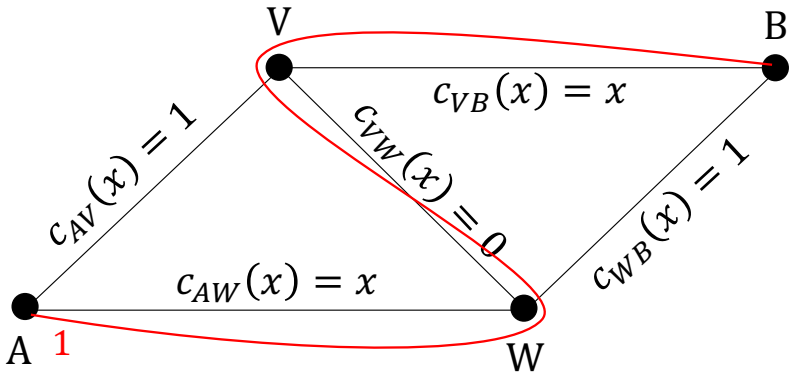


Figure 2: The red allocation is the only equilibrium and costs 2. The allocation in Fig 1 is still a valid solution with cost 1.5.

address include matching drivers and riders in rideshare programs and deciding how to split costs between friends who are sharing a ride.

Online Model

We start with a formal model of learning and learnability in presence of unpredictable, even possibly adversarial, sequences.

Consider a scenario where we are faced with a sequence of incoming instances and we need to make an irrevocable decision about the instance as soon as we see it. To provide robust learning guarantees, we assume that an all-powerful adaptive adversary is choosing the instances and therefore we do not know how well our decisions will perform a priori. An example of this scenario is deciding which route to take to arrive on campus everyday without knowing the traffic en route a priori. At the end of the day, we can look back at the traffic in the city (on all roads) and decide on the route we will be taking tomorrow. What you want is that, over a long period of time, we don't do much worse than if we took the single best route in hindsight.

In another example, consider the setting where everyday we guess whether the stock market index is going to go up or down. You may rely on some recommendation sources: Wall Street Journal, CNN, your coworker, your psychic(!), etc. At the end of the day, you observe whether you were right or wrong. Your goal is not to make many more mistakes compared to your best advisor in hindsight.

This type of decision making will be formalized here and in the next few lectures.

Mistake Bound Model

A simple model for formalizing online decisions is the Mistake Bound model. Given a class of hypotheses $\mathcal{H}$, for $t = 1, 2, 3, ...$

- Receive an $x^t \in \mathcal{X}$.
- Predict $\hat{y}^t \in \{-, +\}$.
- Receive the true label y^t. A mistake is made if $\hat{y}^t \neq y^t$

The goal is to bound the number of mistakes you make, assuming that the sequence of labeled instances (even if produced by an adversary) is consistent with respect to an unknown hypothesis $h^* \in \mathcal{H}$, i.e., for all t, $y^t = h^*(x^t)$. At every round t, the algorithm has access to the sequence of past experience, i.e., $\{(x^1, y^1), (x^2, y^2), \cdots, (x^{t-1}, y^{t-1})\}$.

Definition 1. An algorithm $\mathcal{A}$ learns the hypothesis class $\mathcal{H}$ with mistake bound M, if M is the minimum integer for which the follow-

ing statement holds: For *any* online sequence $(x^1, y^1), (x^2, y^2), ...$ that is consistent with respect to some unknown $h^* \in \mathcal{H}$, $\mathcal{A}$ makes at most M mistakes.

Next, we provide a general-purpose algorithm for learning in the mistake bound model.

One way to look at Mistake Bound model is is that our hypotheses in $\mathcal{H}$ are like "experts". Each expert will make a prediction at every time step t. There is at least one expert who is always right (by consistency) but we don't know who it is. We can observe the history of predictions by every expert to make a decision at time t. Our goal is not to figure out who this perfect expert is, rather, to not make more than M mistakes. In fact, there is a simple and universal algorithm for learning any finite class of hypotheses in the mistake bound model.

Majority/Halving Algorithm Algorithm: At time t, consult all h_i that have yet to make a mistake. Go with their majority vote.

Theorem 1. *The Majority Algorithm has mistake bound* $M \leq \lfloor \log_2 |\mathcal{H}| \rfloor$.

Proof. Note that the mistake bound model assumes that there is some consistent $h^* \in \mathcal{H}$ such that $\forall t, h^*(x^t) = y^t$. Let $S_t = \{h_i : h_i \text{ has not made a mistake so far}\}$ and let $W_t = |S_t|$. If the majority make a mistake, then at least half of S_t were wrong. Thus, $W_{t+1} \leq 0.5 W_t$. Furthermore, we know $W_t \geq 1$ for all t since we assume there exists a "perfect" expert that is always right. This implies that you can halve W_t at most $\lfloor \log_2(|\mathcal{H}|) \rfloor$ times. □

The Majority algorithm relies heavily on the existence of a perfect expert, i.e., a consistent hypothesis. What if there is no consistent $h^* \in \mathcal{H}$? Can we still provide good guarantees on the mistake bound? This is what we will study in the next lecture.

Lecture 2

Learning under Misspecification

In the last lecture, we saw how the Majority Algorithm can have at most $\lfloor \log_2 |\mathcal{H}| \rfloor$ mistakes regardless of the time horizon we run the algorithm for. However, a crucial assumption used in Theorem 1 is that the sequence $\{(x^i, y^i)\}_{i=1}^{\infty}$ is consistent with respect to some unknown $h^* \in \mathcal{H}$. What if there is no such consistent h^*?

We can instead compare our algorithm to the function in $\mathcal{H}$ that performs the best. Let **OPT** denote the number of mistakes made by the best function in $\mathcal{H}$, and denote the function as h^*, i.e.,

$$\textbf{OPT} = \min_{h \in \mathcal{H}} \sum_t \mathbb{I}\{h(x^t) \neq y^t\}, \tag{1}$$

$$h^* = \arg\min_{h \in \mathcal{H}} \sum_t \mathbb{I}\{h(x^t) \neq y^t\}. \tag{2}$$

Weighted Majority Algorithm

Algorithm 1: Weighted Majority (ϵ)

procedure WEIGHTED MAJORITY(ϵ)
  $w_1^1, \ldots, w_n^1 \leftarrow 1$ initial weights ▷ w_i^t is expert i weight @ time t
  for $t = 1, \ldots$ **do**
    $W^t \leftarrow \sum_{i=1}^n w_i^t$
    if $\sum_{i:h_i(x_t)=1} w_i^t \geq W^t/2$ **then**
      Predict $\hat{y}^t \leftarrow 1$
    else
      Predict $\hat{y}^t \leftarrow 0$
    end if ▷ Observe y^t after prediction
    $E^t \leftarrow \{i : h_i(x_t) \neq y^t\}$
    $w_i^{t+1} = w_i^t \left[(1-\epsilon)^{\mathbb{I}\{i \in E^t\}}\right] \forall i$
  end for
end procedure

For convenience, define $n = |\mathcal{H}|$.

Theorem 2. *Weighted Majority(ϵ) algorithm makes at most* $\frac{2}{1-\epsilon}\mathbf{OPT} + \frac{2}{\epsilon}\ln n$. *The described bound is weaker version, in particular, for* $\epsilon = 0.5$, *the bound becomes* $2.4(\mathbf{OPT} + \log_2 n)$.

Proof. We shall derive the bound for $\epsilon = 0.5$. Intuitively, W^t is like the credibility of all the experts combined, which decreases as the algorithm makes mistakes. Let i^* refer to the index of optimal expert, then, we have the bound

$$W^t \geq w_{i^*}^t \geq (\frac{1}{2})^{\mathbf{OPT}}.$$

The above makes use of the fact that the weight $w_{i^*}^t$ of the optimal expert will be at least $(\frac{1}{2})^{\mathbf{OPT}}$ since the algorithm halves the weights of the experts when a mistake is made for $\epsilon = 0.5$.

If the algorithm makes mistake at time t, we have

$$W^{t+1} = \frac{1}{2}\sum_{i \in E^t} w_i^t + \sum_{i \notin E^t} w_i^t \leq \frac{3}{4}W^t. \tag{3}$$

The last inequality follows from $\sum_{i \in E^t} w_i^t \geq W^t/2$. Thus, denoting M^t as number of mistakes made by the algorithm till time t and M as the total number of mistakes by the algorithm, we obtain

$$W^t \leq \left(\frac{3}{4}\right)^{M_t} W^1 \tag{4}$$

$$= \left(\frac{3}{4}\right)^{M_t} n. \tag{5}$$

From (3) and (5), obtain

$$M_t \leq \frac{\mathbf{OPT} + \log_2 n}{\log_2 \frac{4}{3}} \quad \forall t \tag{6}$$

$$\implies M \leq \frac{\mathbf{OPT} + \log_2 n}{\log_2 \frac{4}{3}} \tag{7}$$

□

The weighted majority algorithm has a shortcoming. In case the weights of the functions which predict 0 and 1 are nearly equal, even then the algorithm confidently goes ahead with one of the labels as a prediction. This can be exploited by the adversary. If we consider a randomized algorithm, we can achieve a slightly better mistake bound. Before we dive into this algorithm, let us define some terms to work at a higher level of generality.

Randomized Weighted Majority

There are n experts/options and at time t, each expert i incurs a 'cost' $c_t(i)$, where $c_t : [n] \to [0, 1]$. Let i_t denote the expert chosen by

the algorithm at t, so the overall cost of the algorithm till time T is $\sum_{t=1}^{T} c^t(i^t)$. The optimal cost is given by $\mathbf{OPT} = \min_{i\in[n]} \sum_{t=1}^{T} c^t(i)$, and let i^* denote the optimal expert.

Algorithm 2: Randomized Weighted Majority (ϵ)

procedure RWM(ϵ)
  $w_1^1, \ldots, w_n^1 \leftarrow 1$ initial weights ▷ w_i^t is expert i weight @ time t
  for $t = 1, \ldots$ **do**
    $W^t \leftarrow \sum_{i=1}^{n} w_i^t$
    $p_i^t \leftarrow w_i^t / W^t$
    Sample $i^t \sim p^t$
    Predict i^t and observe $c^t(\cdot)$
    $w_i^{t+1} = w_i^t (1-\epsilon)^{c^t(i)} \quad \forall i$
  end for
end procedure

Theorem 3. *Under RWM(ϵ), for $T > 0$,*

$$\mathbb{E}\left[\sum_{i=1}^{T} c^t(i^t)\right] \leq \frac{1}{1-\epsilon}\mathbb{E}\left[\min_{i\in[n]} \sum_{t=1}^{T} c^t(i)\right] + \frac{\ln n}{\epsilon}$$

Proof.

$$W^{t+1} \geq w_{i^*}^{t+1} = (1-\epsilon)^{\sum_{i=1}^{t} c^t(i^*)}$$

It is important to note that $W^{t+1}, w_{i^*}^{t+1}, i^*, c^t(\cdot)$ are all random variables so the above inequality is technically an almost sure statement. Taking *log* and then expectation over all the randomness,

$$\mathbb{E}[\ln W^{t+1}] \geq \ln(1-\epsilon)\mathbb{E}[\sum_{i=1}^{t} c^t(i^*)]. \tag{8}$$

Next, let us upper bound $\mathbb{E}[\ln W^{t+1}]$.

$$W^{t+1} = \sum_{i\in[n]} (1-\epsilon)^{c^t(i)} w_i^t \tag{9}$$

$$\leq W^t - \epsilon \sum_{i\in[n]} c^t(i) w_i^t, \tag{10}$$

where we used the identity $(1-\epsilon)^c \leq 1 - c\epsilon$ for $c \in [0,1]$ in (9). Let H_t refer to all the history and observations made by the algorithm before loop at time t. Observe that $\sum_{i\in[n]} c^t(i) w_i^t = W^t \sum_{i\in[n]} p_i^t c^t(i) = W^t \mathbb{E}[c^t(i^t)|H_t]$. Again, it is important to note that the above is an almost sure statement involving random variables. Using it in (10) and the fact that $1 - x \leq e^{-x}\ \forall x \geq 0$, get

$$W^{t+1} \leq W^t(1 - \epsilon\mathbb{E}[c^t(i^t)|H_t]) \tag{11}$$

$$\leq W^t \exp(-\epsilon\mathbb{E}[c^t(i^t)|H_t]). \tag{12}$$

Recursing (12), we get $W^{t+1} \leq n \exp(-\epsilon \sum_{k=1}^{t} \mathbb{E}[c^k(i^k)|H_k])$. Taking *log* and then expectation, $\mathbb{E}[\ln W^{t+1}] \leq \ln n - \epsilon \sum_{k=1}^{t} \mathbb{E}\left[\mathbb{E}[c^k(i^k)|H_k]\right]$. By the tower property of conditional expectation, $\mathbb{E}\left[\mathbb{E}[c^k(i^k)|H_k]\right] = \mathbb{E}[c^k(i^k)]$. Combining the above with (8), we get

$$\mathbb{E}\left[\sum_{i=1}^{T} c^t(i^t)\right] \leq -\frac{\ln(1-\epsilon)}{\epsilon}\mathbb{E}[\sum_{i=1}^{t} c^t(i^*)] + \frac{\ln n}{\epsilon} \tag{13}$$

$$\leq \frac{1}{1-\epsilon}\mathbb{E}[\sum_{i=1}^{t} c^t(i^*)] + \frac{\ln n}{\epsilon} \tag{14}$$

□

An immediate corollary of the theorem is as follows.

Corollary 1. *Set* $\epsilon = \min\left\{\frac{1}{2}, \sqrt{\frac{\ln n}{2\mathbf{OPT}}}\right\}$ *and get regret* $O(\sqrt{\mathbf{OPT} \ln n})$.

One problem with the above result is that setting ϵ as described is impractical since we do not know the value of **OPT**, so we can instead upper bound it by T.

Corollary 2. *Set* $\epsilon = \min\left\{\frac{1}{2}, \sqrt{\frac{\ln n}{2T}}\right\}$ *and get regret* $O(\sqrt{T \ln n})$.

What if we do not know T either before starting the algorithm? One smart way of dealing with it is to assume the horizon is some value T_0 and set appropriate ϵ. If during the run of the algorithm, we observe time exceeding T_0, then from that point on, act as if the horizon is $2T_0$ and change ϵ, and so on. It can be shown that for unknown horizon length T, this modified algorithm still gets regret $O(\sqrt{T \ln n})$.

Lecture 3

We start with a few announcements. Homework 0 is out by tonight. It will be graded and there will be peer graders.

Last lecture, we talked about the WeightedMajority, and observed that randomness improved the performance, i.e., RandomizedWeightedMajority. We defined the notion of *regret* as below:

$$\mathsf{Regret}_T(\mathcal{A}, c) = \mathbb{E}\left[\underbrace{\sum_{t=1}^{T} c^t(i^t)}_{\text{alg. output}} - \underbrace{\min_{i\in[n]} \sum_{t=1}^{T} c^t(i)}_{\text{opt}}\right],$$

$$\mathsf{Regret}_T(\mathcal{A}) = \inf_c \mathsf{Regret}_T(\mathcal{A}, c).$$

We proved that

$$\mathsf{Regret}_T(\textsc{RandomizedWeightedMajority}) \leq 2\sqrt{2T\log n}.$$

Today we will discuss the notion of online learnability, and ask what classes of problems are online learnable. We will derive an upper bound and lower bound on online learnability, and describe an algorithm for online learning called SOA.

We noted that the regret of randomized weighted majority does not grow quite as fast as T does. More formally, the average regret

$$\lim_{T\to\infty} \frac{\mathsf{Regret}_T(\textsc{RandomizedWeightedMajority})}{T} = 0.$$

This is what we want for online learnability.

Definition 2 (Online Learning Algorithm)**.** Algorithm $\mathcal{A}$ learns $\mathcal{H}$ in the *online adversarial model* if

$$\lim_{T\to\infty} \frac{\mathsf{Regret}_T(\mathcal{A})}{T} = 0,$$

i.e., there exists a function $T_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for any $\varepsilon \in (0,1)$ and $T \geq T_{\mathcal{H}}(\varepsilon)$,

$$\frac{\mathsf{Regret}_T(\mathcal{A})}{T} \leq \varepsilon.$$

This function $T_{\mathcal{H}}$ is analogous to *sample complexity*, i.e., how many samples we need to achieve a low enough regret. It is ideal to find $T_{\mathcal{H}}$ which gives us a *non-asymptotic* sample complexity. Alternatively one can show that the limit is 0 without finding such a $T_{\mathcal{H}}$ — this is called an *asymptotic* result — but it is not ideal. Also, such $\mathcal{A}$ are called *no-regret learners* for $\mathcal{H}$.

Definition 3 (Online Learnable Hypotheses)**.** Class $\mathcal{H}$ is *online adversarial learnable* if there exists an algorithm $\mathcal{A}$ which learns $\mathcal{H}$ in the online adversarial model.

Example. The RandomizedWeightedMajority regret bound says that all finite hypothesis classes $\mathcal{H}$ of size n are online adversarial learnable, using the RandomizedWeightedMajority algorithm and

$$T_{\mathcal{H}}(\varepsilon) = 8\frac{\log n}{\varepsilon^2}$$

for some absolute constant c_1. Indeed, we have, for $T \geq T_{\mathcal{H}}(\varepsilon)$, that

$$\frac{\mathsf{Regret}_T(\textsc{RandomizedWeightedMajority})}{T} = \frac{2\sqrt{2\log n}}{\sqrt{T}} \leq \frac{2\sqrt{2\log n}}{\sqrt{T_{\mathcal{H}}(\varepsilon)}} \leq \varepsilon.$$

Here is the question we will try to answer soon. Every time we deal with a finite $\mathcal{H}$, we know the answer as above. But the above bounds are increasing logarithmically in n, and it is very natural to study $\mathcal{H}$ which are infinite, i.e., the binary threshold example from earlier. We want some characterization, well-defined for infinite hypothesis classes, that captures online learnability.

The first question is whether the size of $\mathcal{H}$ is the correct measure. The answer turns out to be no. The reasoning is the following. If we are given a set of functions, we can copy and perturb each function by a tiny amount, and get essentially the same set of functions and the same learnability property. More formally, we can get a lot of functions which will disagree on small parts of the domain.

Now that we're thinking of the domain $\mathcal{X}$, let's ask whether the size of the domain is the correct characterization. Is it possible to keep padding $\mathcal{X}$ without changing the difficulty of regret minimization? The answer is that it is of course possible by adding redundant coordinates.

So the problem is about the "richness" of $\mathcal{H}$ on $\mathcal{X}$.

Example (Lack of Online Learnability). When $\mathcal{H}$ and $\mathcal{X}$ are infinite, online learnability may not be possible. The idea is to use our familiar one-dimensional thresholds:

$$\mathcal{H} = \left\{h_a \colon [0,1] \to \{-1,1\} \text{ given by } h_a(x) \doteq 2 \cdot \mathbb{I}(x \geq a) - 1 \;\middle|\; a \in [0,1]\right\}.$$

Suppose we are given $x^1 = \frac{1}{2}$. Our algorithm says that $h(x^1) = +1$, but the adversary hands us $y^1 = -1$, giving us a mistake. The

adversary's choice is then $x^2 = \frac{3}{4}$, and we construct an infinite binary tree in this fashion. See Figure 3.

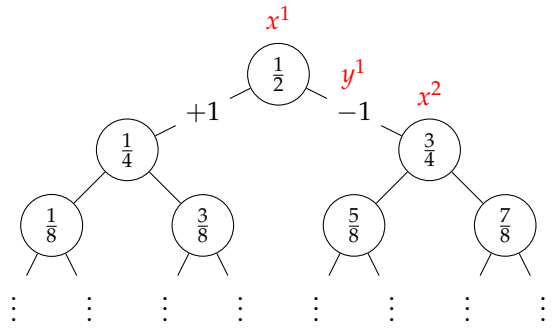


Figure 3: Online learning for thresholds on $[0, 1]$.

More generally, as an adversary, our strategy is very simple; we pick an edge at random at each iteration, or equivalently, we pick a path at random in the infinite binary tree. Since we picked an answer at every node at random, the algorithm can do no better than random guessing, and after T iterations will make roughly $\frac{T}{2}$ mistakes. There will exist some $h \in \mathcal{H}$ which is consistent with the path we chose, and so the optimal $h \in \mathcal{H}$ will achieve loss 0. Thus the regret is $\frac{T}{2}$, which is not sublinear, and so this class is not online learnable.

There is an interesting criticism of this counterexample, where we don't really care about small enough differences between functions in $\mathcal{H}$ in practice — e.g., in our example, we only care about determining the threshold up to a given precision — but formally this model is still not online learnable. Even as we narrow down the threshold to a smaller interval, the adversary can exploit the fact that there is some uncertainty to cause large amounts of regret.

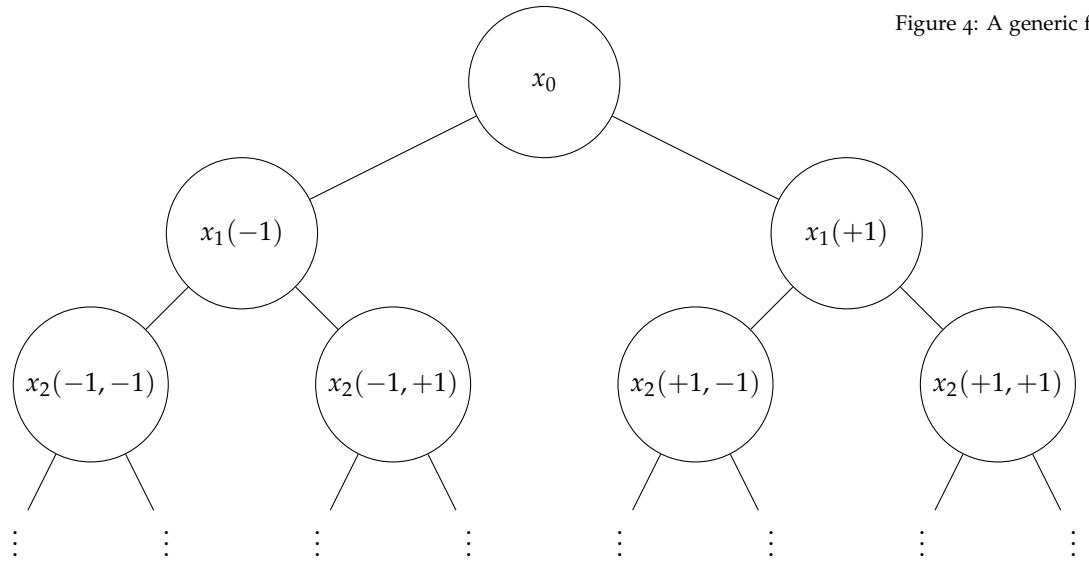
There are two ways to deal with this discrepancy. The first is to make everything stochastic instead of adversarial, in which case smaller regions of uncertainty are assigned smaller probabilities, so the probability that a mistake is made in this region goes to 0. This definition is too naive; producing a satisfying mechanism to weld together stochastic and adversarial behavior has been the topic of recent research.

If this tree for our problem is infinite, then the problem is not learnable. The relevant quantity in determining online learnability is the depth of this tree.

More formally, consider a full binary tree of depth d. The nodes of this tree are $x \in \mathcal{X}$. For every $y = (y_1, \ldots, y_d) \in \{+1, -1\}^d$, define $x_t(y)$ as the node in the tree at depth t if we start at the root and, at time $i \in [t]$, go left if $y_i = -1$ and right if $y_i = +1$. See Figure 4

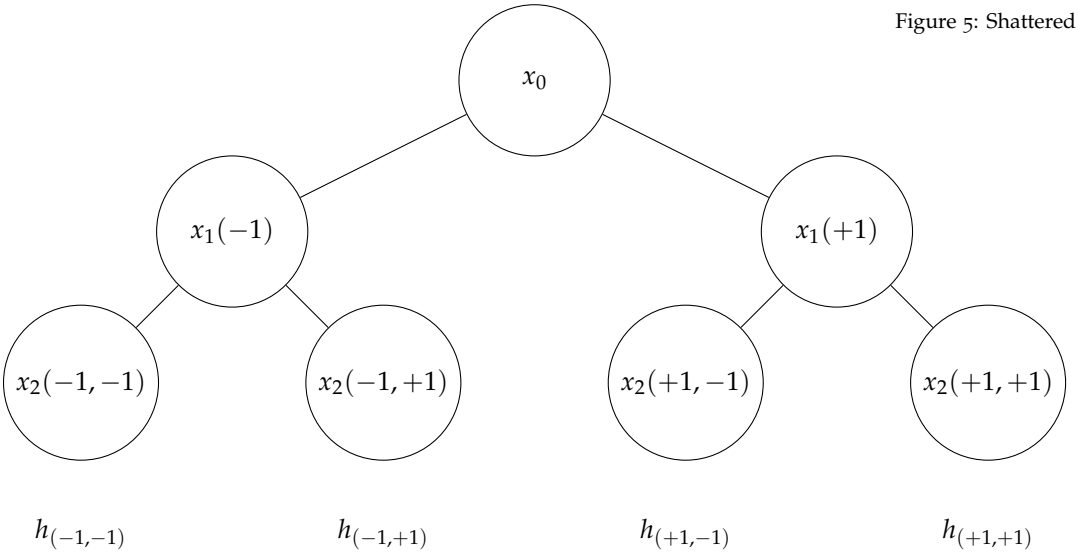
Such a tree is called a *shatter tree* if it has the following property: for all $y = (y_1, \ldots, y_d) \in \{-1, +1\}^d$ there exists $h \in \mathcal{H}$ such that

Figure 4: A generic full binary tree.



$h(x_i(y)) = y_i$ for all $i \in [d]$. See Figure 5.

Figure 5: Shattered tree.



As in the above figure, a shattered tree conceptually assigns a $h \in \mathcal{H}$ to every leaf, which predicts the response to all samples on the path from the root to the leaf correctly. Note that in this model we can send $d \to \infty$.

We formalize a general notion that captures the above intuition .

Definition 4 (Littlestone Dimension [Lit87])**.** The Littlestone dimension of class $\mathcal{H}$, denoted $\mathsf{LD}(\mathcal{H})$, is the maximal depth of any shatter tree for $\mathcal{H}$.

Now the answer to our online learnability question is that a hy-

pothesis class $\mathcal{H}$ is online learnable if and only if $\mathsf{LD}(\mathcal{H})$ is finite.

Many other quantities are formalized by Littlestone dimension; for example, differential privacy and game theoretic dynamics.

Theorem 4 (Regret Lower Bound)**.** *The regret of any algorithm $\mathcal{A}$ is lower bounded by*

$$\mathsf{Regret}_T(\mathcal{A}) \geq \Omega(\sqrt{T \cdot \min\{T, \mathsf{LD}(\mathcal{H})\}}).$$

We will not prove this, but the argument is similar to what we already proved.

Corollary 3. *If $\mathsf{LD}(\mathcal{H}) = \infty$ then $\mathcal{H}$ is not online adversarially learnable.*

Proof. The same game we played above in the linear thresholding case can prove linear regret in infinite-dimensional shatter trees. □

Theorem 5 (Regret Upper Bound [BDR21])**.** *There exists an algorithm $\mathcal{A}$ such that*

$$\mathsf{Regret}_T(\mathcal{A}) \leq O(\sqrt{T \cdot \min\{T, \mathsf{LD}(\mathcal{H})\}}).$$

While this is the product of recent research, the similar upper bound

$$\mathsf{Regret}_T(\mathcal{A}) \leq O(\sqrt{T \log T \cdot \min\{T, \mathsf{LD}(\mathcal{H})\}})$$

was known more than 10 years ago[1].

[1] Shai Ben-David, Dávid Pál, and Shai Shalev-Shwartz. Agnostic online learning. In *Proceedings of the Conference on Learning Theory (COLT)*, 2009

Theorem 6 (Regret Upper Bound via SOA [Lit87])**.** *In the mistake bound model, with the "promise" that there exists $h \in \mathcal{H}$ which is perfect, there is an algorithm called "Standard Optimal Algorithm (SOA)" that makes $\leq \mathsf{LD}(\mathcal{H})$ mistakes.*

Proof. We want to cook up an algorithm which implicitly or explicitly uses the Littlestone dimension. The idea is very similar to the halving algorithm. In the halving algorithm we kept track of the "credibility" of each hypothesis, and at each timestep took the majority of all hypotheses weighted by credibility, reweighting the credibility if a mistake is made.

The SOA will transplant this credibility to the Littlestone dimension world. At each mistake, we will remove the more credible, or complex part of $\mathcal{H}$ from consideration.

This is the halving algorithm, except instead of taking into account the credibility we use the Littlestone dimension.

It suffices to show that if we make a mistake at time t, then

$$\mathsf{LD}(\mathcal{H}_{t+1}) \leq \mathsf{LD}(\mathcal{H}_t) - 1.$$

What this is saying is that at each timestep t, if we make a mistake then we move to the smaller sub-tree corresponding to $\mathcal{H}_{t+1}$. Since

Algorithm 3: Standard Online Algorithm

$\mathcal{H}_1 \doteq \mathcal{H}$.
for times $t \in \{1, 2, \dots\}$ **do**
  Receive x^t
  Define $\mathcal{H}_t^{+1} \doteq \{h \in \mathcal{H}_t \colon h(x^t) = +1\}$
  Define $\mathcal{H}_t^{-1} \doteq \{h \in \mathcal{H}_t \colon h(x^t) = -1\}$
  Predict $\widehat{y}^t \doteq \arg\max_{y \in \{-1,1\}} \mathsf{LD}(\mathcal{H}_t^y)$
  Observe y^t
  Update $\mathcal{H}_{t+1} = \mathcal{H}_t^{y^t}$
end for

we always go to a smaller sub-tree at a mistake, we make at most $\mathsf{LD}(\mathcal{H})$ mistakes.

It remains to prove the above inequality. Suppose for the sake of contradiction that $\mathsf{LD}(\mathcal{H}_{t+1}) = \mathsf{LD}(\mathcal{H}_t)$. Since we chose the $\mathcal{H}_t^{\pm}$ with the highest Littlestone dimension, and we made a mistake, we must have

$$\mathsf{LD}(\mathcal{H}_{t+1}) = \mathsf{LD}(\mathcal{H}_t) = \mathsf{LD}(\mathcal{H}_t^{+1}) = \mathsf{LD}(\mathcal{H}_t^{-1}).$$

But if the trees corresponding to $\mathcal{H}_t^{+1}$ and $\mathcal{H}_t^{-1}$ are shattered and have depth d, the tree corresponding to $\mathcal{H}_t$, with root x^t, must be shattered by one of the functions in $\mathcal{H}_t^{y^t}$, and has depth $d + 1$. Thus we must have

$$\mathsf{LD}(\mathcal{H}_t^{+1}) = \mathsf{LD}(\mathcal{H}_t^{-1}) = d \implies \mathsf{LD}(\mathcal{H}_t) = d + 1.$$

This is a contradiction. □

Lecture 4

Today, we will introduce game theory, drawing a connection to the no-regret learning algorithms discussed in the past few lectures.

Game theory as a field was introduced by John von Neumann in 1928 as a means to reason about how strategic agents interact with each other. Most courses on this topic consider strategic agents to be competitive or non-cooperative; however, there are many cases where we are interested in the behavior of systems with cooperative agents. Later on in the class, we will learn about Shapley values, which measure the incentive for coalitions to cooperate.

A game is defined by a set of K agents, each with (possible different) actions sets A_j. Each agent j selects an action from their action set $a_j \in A_j$. These actions can be selected sequentially or simultaneously. To measure the outcome of the actions chosen, each agent possesses their own loss function $\ell_j(a_j, a_{-j})$, where a_{-j} represents the set of actions chosen by all other agents. Each agent seeks to strategically select their action to minimize their own loss function.

Example (Chicken). Consider two drivers approaching each other at high speed. Each driver has the choice to swerve or drive straight ahead. In the best outcome, a driver would continue straight ahead while causing their opponent to swerve, so that their opponents would be a "chicken". If both drivers choose to swerve, both are equally cowardly. However, if both refuse to swerve, a horrible collision ensues.

We can model the set of possible outcomes and losses the following matrix:

	Swerve	Drive Ahead
Swerve	0, 0	-1, +1
Drive Ahead	+1, -1	-10, -10

For two player games, we use the notation that in cell i, j, we list the losses as $\ell_r(a_i, a_j), \ell_c(a_i, a_j)$, where r represents the row player and c represents the column player.

Now, consider the case where the row player rips off their steering wheel, committing to driving straight ahead. The best response for

the column player would be to swerve. This would be an instance of a sequential game. We will study sequential games such as Stackelberg games later on in the semester.

Example (Hide/Choose). One person (called the hider) has the option to either place one coin in their left hand or two coins in their right hand, and then places both hands behind their back. Another person (called the chooser) then selects left or right, and receives the coins (if any) from that respective hand of the hider.

With the chooser as the row player and the hider as the column player, this game can be represented by the following matrix, where we use utilities instead of losses:

	Left	Right
Left	1, -1	0, 0
Right	0, 0	2, -2

Rather than committing to a fixed action (or strategy), each player may find it beneficial to instead select a distribution over their actions. The best distribution for the hider is to hide it in their left hand with probability $\frac{2}{3}$ and in their right with probability $\frac{1}{3}$; conversely, the choser finds it optimal to select left with probability $\frac{1}{3}$ and right with probability $\frac{2}{3}$.

With these random strategies, both players choose left with probability $\frac{4}{9}$ and both players choose right with probability $\frac{1}{9}$. This means that the expected reward under these (optimal) random strategies is $\frac{2}{3}$, while the expected loss for the hider is $\frac{2}{3}$.

Since the utilities sum to zero in each cell, this is an instance of a zero-sum game. Zero-sum games have a value, which is the expected return (or loss) while playing in a game. In the example above, the value of the game is $\frac{2}{3}$.

Example (Rock Paper Scissors). We can model the popular game of rock paper scissors as a zero-sum game between two players. It induces the following game matrix:

	Rock	Paper	Scissors
Rock	0, 0	-1, 1	1, -1
Paper	1, -1	0, 0	-1, 1
Scissors	-1, 1	1, -1	0, 0

If one player was forced to reveal their strategy before the other player selected theirs, committing to a single strategy (such as always choosing rock) would make the game extremely easy for the other player. The best thing for the first player to do would be to commit to a strategy of selecting each action with probability $\frac{1}{3}$. If so, the best thing for the second player would be to commit to the same strategy. This leads to a value of zero for the game.

Intuitively, it would seem that when the first player is forced to commit to a strategy, the second player would have an advantage. However, in 1928, Von Neumann proved that in two player zero-sum games with finite action spaces, there is no advantage.

Before stating this formally, we first introduce the following procedure:

- The row player first selects a distribution P over their action set
- The column player then selects their own distribution Q that minimizes their loss
- Sample $i \sim P$ and $j \sim Q$
- Calculate the expected loss of the row player by, where L coalesces $\ell_r(i, j)$ into a matrix

$$\mathcal{L}(P, Q) = \sum_i \sum_j P_i Q_j \ell_r(a_i, a_j) = P^\intercal L Q$$

When the row player is forced to commit first, they will try to minimize their expected loss; when the column player responds, they try to maximize the loss of the row player, thereby increasing their utility. We can represent this by:

$$\max_{Q \in \Delta(A_c)} \min_{P \in \Delta(A_r)} \mathcal{L}(P, Q)$$

On the other hand, we can represent the column player committing first as:

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} \mathcal{L}(P, Q)$$

This leads us to our primary result of the lecture.

Theorem 7 (The Minimax Theorem)**.** *Every two player, zero-sum game with finite action spaces satisfies:*

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} \mathcal{L}(P, Q) = \max_{Q \in \Delta(A_c)} \min_{P \in \Delta(A_r)} \mathcal{L}(P, Q)$$

Proof. We will prove this by showing the inequality holds in both directions. Showing $\geq$ is simple, as it follows from first principles. To show the more difficult direction $\leq$, we will leverage the no-regret online learning strategies discussed over the past few lectures.

First, to show $\geq$, consider the relationship below, which we know to be true by the definition of a maximum.

$$\max_{Q \in \Delta(A_c)} \mathcal{L}(P, Q) \geq \mathcal{L}(P, Q) \quad \forall P, Q \tag{15}$$

Then, it must also be true that

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} \mathcal{L}(P, Q) \geq \min_{P \in \Delta(A_r)} \mathcal{L}(P, Q) \quad \forall Q \tag{16}$$

Note that the left hand side is greater than the right hand side for any choice of Q. Then, it must be true that

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} \mathcal{L}(P, Q) \geq \max_{Q \in \Delta(A_c)} \min_{P \in \Delta(A_r)} \mathcal{L}(P, Q) \tag{17}$$

Now, it remains to show $\leq$. Over the years, this has been shown through many different methods, but many rely on the use of duality. Instead, we will model this as a repeated game, where each player is free to choose P_t and Q_t at each iteration, and adjust their strategies according to the outcomes of past iterations.

Formally, consider a repeated game where the row player chooses P_t according to a no-regret algorithm that uses the history of distributions played by the column player $Q_1, \ldots, Q_{t-1}$. At each time t, the column player plays Q_t, a best response to P_t (i.e. minimizes their expected loss). We can define the time-averaged distribution of strategies for the row and column player as $\bar{P} = \frac{1}{T} \sum_{t=1}^{T} P_t$ and $\bar{Q} = \frac{1}{T} \sum_{t=1}^{T} Q_t$ respectively.

Remember that we can express $\mathcal{L}(P, Q)$ as $P^\intercal L Q$, where $[L]_{i,j} = \ell_r(a_i, a_j)$. We will be working with this matrix expression for the remainder of the proof.

Begin by considering the following expression. This is true by the definition of minimum, since $\bar{P}$ is a valid distribution in $\Delta(A_r)$.

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} P^\intercal L Q \leq \max_{Q \in \Delta(A_c)} \bar{P}^\intercal L Q \quad \forall \bar{P} \tag{18}$$

Now, we can substitute $\bar{P}$ with its definition

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} P^\intercal L Q \leq \max_{Q \in \Delta(A_c)} \frac{1}{T} \sum_{t=1}^{T} P_t^\intercal L Q \quad \forall P_t \tag{19}$$

The maximum of a sum is less than or equal to the sum of maximums. This is a consequence of Jensen's inequality.

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} P^\intercal L Q \leq \max_{Q \in \Delta(A_c)} \frac{1}{T} \sum_{t=1}^{T} P_t^\intercal L Q \leq \frac{1}{T} \sum_{t=1}^{T} \max_{Q \in \Delta(A_c)} P_t^\intercal L Q \quad \forall P_t \tag{20}$$

When P_t is selected, the column player selects Q_t as their best response. We can replace the maximum in the summation with this.

$$\min_{P \in \Delta(A_r)} \max_{Q \in \Delta(A_c)} P^\intercal L Q \leq \frac{1}{T} \sum_{t=1}^{T} P_t^\intercal L Q_t \quad \forall Q_t \tag{21}$$

Recall from lecture 2 how we define average regret. We expressed average regret as

$$\frac{\text{Regret}}{T} = \underbrace{\frac{1}{T} \sum_{t=1}^{T} P_t^\intercal L Q_t}_{\text{output of the algorithm}} - \underbrace{\min_{P \in \Delta(A_r)} P^\intercal L \bar{Q}}_{\text{optimal } P \text{ in hindsight}} \quad \forall \bar{Q}, Q_t \tag{22}$$

Rearranging, we find

$$\frac{1}{T}\sum_{t=1}^{T} P_t^\intercal L Q_t = \min_{P\in\Delta(A_r)} P^\intercal L\bar{Q} + \frac{\text{Regret}}{T} \quad \forall \bar{Q}, Q_t \tag{23}$$

Thus, we can replace the right hand side of (21) with this expression

$$\min_{P\in\Delta(A_r)} \max_{Q\in\Delta(A_c)} P^\intercal L Q \leq \min_{P\in\Delta(A_r)} P^\intercal L\bar{Q} + \frac{\text{Regret}}{T} \quad \forall \bar{Q} \tag{24}$$

By the definition of maximum, we can upper bound the right hand side by

$$\min_{P\in\Delta(A_r)} \max_{Q\in\Delta(A_c)} P^\intercal L Q \leq \max_{Q\in\Delta(A_c)} \min_{P\in\Delta(A_r)} P^\intercal L Q + \frac{\text{Regret}}{T} \tag{25}$$

Since we are playing a no-regret algorithm, for any ϵ, we can find a T such that $\frac{\text{Regret}}{T} < \epsilon$. We find that as T approaches infinity, $\frac{\text{Regret}}{T}$ converges to 0. Thus, we are justified in stating

$$\min_{P\in\Delta(A_r)} \max_{Q\in\Delta(A_c)} P^\intercal L Q \leq \max_{Q\in\Delta(A_c)} \min_{P\in\Delta(A_r)} P^\intercal L Q \tag{26}$$

This completes our proof of the Minimax Theorem. □

Lecture 5

Lecture 6

Lecture 7

Lecture 8

Lecture 9

Lecture 10

Lecture 11

Lecture 12

Lecture 13

Lecture 14

Lecture 15

Lecture 16

Lecture 17

Lecture 18

Lecture 19

Lecture 20

Lecture 21

Lecture 22

Lecture 23

Lecture 24

Lecture 25

Lecture 26

Bibliography

[BDPSS09] Shai Ben-David, Dávid Pál, and Shai Shalev-Shwartz. Agnostic online learning. In *Proceedings of the Conference on Learning Theory (COLT)*, 2009.

[BDR21] Adam Block, Yuval Dagan, and Alexander Rakhlin. Majorizing measures, sequential complexities, and online learning. In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pages 587–590. PMLR, 2021.

[Lit87] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, 1987.