DS 320 Algorithms for Data Science **Lecture #21**
Spring 2026 Prof. Kira Goldner

The Minimax Theorem and Online Learning

Notation:

- $m \times n$ payoff matrix $\mathbf{A}$—a_{ij} is the row player's payoff for outcome (i, j) when row player plays strategy i and column player plays strategy j
- mixed row strategy $\mathbf{x}$ (a distribution over rows)
- mixed column strategy $\mathbf{y}$ (a distribution over columns)

Expected payoff of the row player:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} \Pr[\text{outcome } (i,j)]\, a_{ij} = \sum_{i=1}^{m}\sum_{j=1}^{n} \underbrace{\Pr[\text{row } i \text{ chosen}]}_{=x_i} \underbrace{\Pr[\text{column } j \text{ chosen}]}_{=y_j} a_{ij}$$
$$= \mathbf{x}^T \mathbf{A} \mathbf{y}$$

The minimax theorem is the amazing statement that turn order *doesn't matter*.

Theorem 1 (Minimax Theorem)**.** *For every two-player zero-sum game* $\mathbf{A}$*,*

$$\max_{\mathbf{x}} \left(\min_{\mathbf{y}} \mathbf{x}^T \mathbf{A} \mathbf{y} \right) = \min_{\mathbf{y}} \left(\max_{\mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{y} \right). \tag{1}$$

On the left, the row player goes first, choosing a strategy to maximize their payoff and protect against the fact that the column player goes second and adapts to their strategy. The right is the opposite situation. The *value of the game* (value that both sides will equal) is 0 in this case: the first player will play randomly and the second will respond arbitrarily.

From LP Duality to Minimax

This is not the original or only argument, but we will now derive Theorem 1 from LP duality arguments. The first step is to formalize the problem of computing the best strategy for the player forced to go first.

Two issues: (1) the nested min/max, and (2) the quadratic (nonlinear) character of $\mathbf{x}^T \mathbf{A} \mathbf{y}$ in the decision variables $\mathbf{x}$, $\mathbf{y}$.

Observation 2. *The second player never needs to randomize. If the row player goes first and chooses any distribution* $\mathbf{x}$*, the column player can then simply compute the expected payoff (with respect to* $\mathbf{x}$*) of each column and choose the best.*

In math, we have argued that

$$\max_{\mathbf{x}} \left(\min_{\mathbf{y}} \mathbf{x}^T \mathbf{A} \mathbf{y} \right) = \max_{\mathbf{x}} \left(\min_{j=1}^{n} \mathbf{x}^T \mathbf{A} \mathbf{e}_j \right) \tag{2}$$

$$= \max_{\mathbf{x}} \left(\min_{j=1}^{n} \sum_{i=1}^{m} a_{ij} x_i \right) \tag{3}$$

where $\mathbf{e}_j$ is the jth standard basis vector, corresponding to the column player deterministically choosing column j.

We've solved one of our problems by getting rid of $\mathbf{y}$. But there is still the nested max/min.

Specifically, we introduce a decision variable v, intended to be equal to (2), and

$$\max v$$

subject to

$$v - \sum_{i=1}^{m} a_{ij} x_i \leq 0 \quad \text{for all } j = 1, \ldots, n$$

$$\sum_{i=1}^{m} x_i = 1$$

$$x_1, \ldots, x_m \geq 0 \quad \text{and} \quad v \in \mathbb{R}.$$

Note that this is a linear program. Rewriting the constraints in the form

$$v \leq \sum_{i=1}^{m} a_{ij} x_i \quad \text{for all } j = 1, \ldots, n$$

makes it clear that they force v to be at most $\min_{j=1}^{n} \sum_{i=1}^{m} a_{ij} x_i$.

If $(v^*, \mathbf{x}^*)$ is an optimal solution, then $v^* = \min_{j=1}^{n} \sum_{i=1}^{m} a_{ij} x_i$. By feasibility, v^* cannot be larger than $\min_{j=1}^{n} \sum_{i=1}^{m} a_{ij} x_i^*$. If it were strictly less, then we can increase v^* slightly without destroying feasibility, yielding a better feasible solution (contradicting optimality).

Since the linear program explicitly maximizes v over all distributions $\mathbf{x}$, its optimal objective function value is

$$v^* = \max_{\mathbf{x}} \left(\min_{j=1}^{n} \mathbf{x}^T \mathbf{A} \mathbf{e}_j \right) = \max_{\mathbf{x}} \left(\min_{\mathbf{y}} \mathbf{x}^T \mathbf{A} \mathbf{y} \right) \tag{4}$$

Now, we do the same thing for the column player, where the column player moves first:

$$\min w$$

subject to

$$w - \sum_{j=1}^{n} a_{ij} y_j \geq 0 \quad \text{for all } i = 1, \ldots, m$$

$$\sum_{j=1}^{n} y_j = 1$$

$$y_1, \ldots, y_n \geq 0 \quad \text{and} \quad w \in \mathbb{R}.$$

At an optimal solution $(w^*, \mathbf{y}^*)$, $\mathbf{y}^*$ is the optimal strategy for the column player (when going first, assuming optimal play by the row player) and

$$w^* = \min_{\mathbf{y}} \left(\max_{i=1}^{m} \mathbf{e}_i^T \mathbf{A} \mathbf{y} \right) = \min_{\mathbf{y}} \left(\max_{\mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{y} \right) \tag{5}$$

These two linear programs are duals! For example, the one unrestricted variable (v or w) corresponds to the one equality constraint in the other linear program ($\sum_{j=1}^{n} y_j = 1$ or $\sum_{i=1}^{m} x_i = 1$, respectively). The n x variables correspond to the remaining dual constraints, and the m y variables correspond to the remaining primal constraints. Then strong duality implies that $v^* = w^*$; in light of (4) and (5), the minimax theorem follows directly.

Online Learning

Think back to when we learned about caching or job scheduling. We always assumed that we knew everything that was coming in advance and could make decisions about the future. What if we couldn't see the future? This is called an *online* setting, not like the internet, but as if the input is waiting *on line*.

An Online Problem

1. The input arrives "one piece at a time."
2. An algorithm makes an irrevocable decision each time it receives a new piece of the input.

Now, for an *Online Decision-Making Problem*, we should consider the event when you have a bunch of *experts* advising you on the stocks or the weather, and you have to choose one to trust each day. Or, equivalently, a bunch of *actions* you could take. Each day (or time step), you get to see how right or wrong the experts are—they are assigned some *loss* (or *loss*) by an *adversary*. Your goal is to come up with a *strategy* of how to choose experts as time goes on such that, after you choose your strategy for each successive time step, the adversary assigns losses, and you get the best losses (or minimal losses) possible. The adversary knows your (possibly randomized) strategy, but does not see the result of the randomness until after assignment losses.

Online Decision-Making

At each time step $t = 1, 2, \ldots, T$:

a decision-maker picks a probability distribution $\mathbf{p}^t$ over her experts or actions $i = 1, \ldots, N$

an adversary picks a **loss** vector $\ell^t : A \to [0, 1]$

an action i^t is chosen according to the distribution $\mathbf{p}^t$, and the decision-maker receives loss ℓ_i^t

the decision-maker learns ℓ^t, the entire **loss** vector

The input arrives "one piece at a time."

What should we compare to?

Thus far, we've been trying to achieve optimal solutions, or comparing to optimal solutions assuming we know full information about the future and what is optimal. Does that still make sense?

Example 1 (Comparing to the Best Action Sequence)**.** Suppose your set of experts (or actions) is $A = \{1, 2\}$. Each day t, the adversary chooses the loss vector ℓ^t as follows: if the algorithm chooses a distribution $\mathbf{p}^t$ for which the probability on action 1 is at least $\frac{1}{2}$, then ℓ^t is set to the vector $(1, 0)$. Otherwise, the adversary sets ℓ^t equal to $(0, 1)$.

This adversary forces the expected cumulative loss of the algorithm to be at least $T/2$, while ensuring that the loss of the best action sequence in hindsight is 0. Thus, the algorithm's approximation is $0/\frac{T}{2}$—no approximation at all.

Example 1 tells us that we should not be trying to compare to the Best Action Sequence—this is too strong of a goal. Instead, we compare it to the loss incurred by the best *fixed action* in hindsight. In words, we change our benchmark from

$$\sum_{i=1}^{T} \min_{i=1}^{N} \ell_i^t \quad \text{to} \quad \min_{i=1}^{N} \sum_{i=1}^{T} \ell_i^t.$$

Definition 1 (Regret)**.** Fix loss[1] vectors $\ell^1, \ldots, \ell^T$. The regret of the action sequence $a^1, \ldots, a^T$ is

$$\underbrace{\sum_{t=1}^{T} \ell^t(a^t)}_{\text{our algorithm}} - \underbrace{\min_{i=1}^{N} \sum_{t=1}^{T} \ell_i^t}_{\text{best fixed action}} . \tag{6}$$

[1]Note that for the rewards setting, the definition of regret would instead be $\max_{i=1}^{N} \sum_{t=1}^{T} r_i^t - \sum_{t=1}^{T} r^t(a^t)$, still minimizing the difference between the algorithm and the best fixed action, but now the maximum reward for the best fixed action will be larger than the algorithm instead of the minimum loss being smaller.

Specifically, our goal is to minimize *regret*.

Example 2 (Randomization Is Necessary for No Regret)**.** Fix a deterministic online decision-making algorithm. At each time step t, the algorithm commits to a single action a_t. The obvious strategy for the adversary is to set the loss of action a_t to 1, and the loss of every other action to 0. Then, the cumulative loss of the algorithm is T while the cumulative loss of the best action in hindsight is at most T/n and the regret is at least $T(1 - \frac{1}{n})$. Even when there are only 2 actions, for arbitrarily large T, the worst-case regret of the algorithm is at least $\frac{T}{2}$.

For randomized algorithms, the next example limits the rate at which regret can vanish as the time horizon T grows.

Example 3 (Regret Lower Bound)**.** Suppose there are $n = 2$ actions, and that we choose each loss vector ℓ^t independently and equally likely to be $(0, 1)$ or $(1, 0)$. No matter how smart or dumb an online decision-making algorithm is, with respect to this random choice of loss vectors, its expected loss at each time step is exactly $\frac{1}{2}$ and its expected cumulative loss is thus $\frac{T}{2}$. The expected cumulative loss of the best fixed action in hindsight is $T/2-b\sqrt{T}$, where b is some constant independent of T. This follows from the fact that if a fair coin is flipped T times, then the expected number of heads is $\frac{T}{2}$ and the standard deviation is $\frac{1}{2}\sqrt{T}$.

Fix an online decision-making algorithm $\mathcal{A}$. A random choice of loss vectors causes $\mathcal{A}$ to experience expected regret at least $b\sqrt{T}$, where the expectation is over both the random choice of loss vectors and the action realizations. At least one choice of loss vectors induces an adversary that causes $\mathcal{A}$ to have expected regret at least $b\sqrt{T}$, where the expectation is over the action realizations.

A similar argument shows that, with n actions, the expected regret of an online decision-making algorithm cannot grow more slowly than $b\sqrt{T \ln n}$, where $b > 0$ is some constant independent of n and T.