DS 320 Algorithms for Data Science **Lecture #21**
Spring 2022 Prof. Kira Goldner

Zero-Sum Games and the Minimax Theorem

Consider the game *Rock-Paper-Scissors*, where as usual, paper covers rock, scissors cuts paper, and rock breaks scissors (that is: the former beats the latter in the comparison). In a face-off, the winner earns +1 and the loser earns -1. If two of the same type face each other, then there is a tie, and both earn 0.

The matrix below shows the game of Rock-Paper Scissors depicted as a *zero-sum-game*. Suppose that brothers Ron and Charlie Weasley are facing off. Each brother must choose a strategy. In the language of the *payoff matrix* below, Ron is the *row player*, and he must choose a row to play as his strategy. Similarly, Charlie is the *column player* and he just choose which column to play. If Ron chooses row i and Charlie chooses column j, then the payoff to Ron will be a_{ij}, and the payoff to Charlie will be $-a_{ij}$, hence the term "zero-sum." Thus, the row and column players prefer bigger and smaller numbers, respectively.

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

Order of Turns

- Typically, RPS is played by both players simultaneously choosing their strategies.
- But what if I made you go first? That's obviously unfair—whatever you do, I can respond with the winning move.
- Now what if I only forced you to commit to a *probability distribution* over rock, paper, and scissors? (Then I respond choosing a strategy, and *then* nature flips coins on your behalf.)

  You can protect yourself by randomizing uniformly among the three options—then, no matter what I do, I'm equally likely to win, lose, or tie.

The *minimax theorem* states that, in general games of "pure competition," a player moving first can always protect herself by randomizing appropriately.

The Minimax Theorem

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 and the Multiplicative Weights Algorithm

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 *reward* (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 rewards, and you get the best rewards (or minimal losses) possible. The adversary knows your (possibly randomized) strategy, but does not see the result of the randomness until after assignment rewards.

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* A

an adversary picks a reward vector $\mathbf{r}^t : A \to [-1, 1]$

an action at is chosen according to the distribution $\mathbf{p}^t$, and the decision-maker receives reward $r^t(a^t)$

the decision-maker learns $\mathbf{r}^t$, the entire reward 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 reward vector $\mathbf{r}^t$ 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 $\mathbf{r}^t$ is set to the vector $(-1, 1)$. Otherwise, the adversary sets $\mathbf{r}^t$ equal to $(1, -1)$.

This adversary forces the expected reward of the algorithm to be nonpositive, while ensuring that the reward of the best action sequence in hindsight is T. Thus, the algorithm's approximation is x/T where $x \leq 0$—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 reward incurred by the best *fixed action* in hindsight. In words, we change our benchmark from

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

Definition 1 (Regret)**.** Fix reward vectors $\mathbf{r}^1, \ldots, \mathbf{r}^T$. The regret of the action sequence $a^1, \ldots, a^T$ is

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

We'd like an online decision-making algorithm that achieves low regret, as close to 0 as possible (and negative regret would be even better). Notice that the worst-possible regret in $2T$ (since rewards lie in $[-1, 1]$). We think of regret $\Omega(T)$ as an epic fail for an algorithm. What is the justification for the benchmark of the best fixed action in hindsight? First, simple and natural learning algorithms can compete with this benchmark. Second, achieving this is non-trivial: as the following examples make clear, some ingenuity is required. Third, competing with this benchmark is already sufficient to obtain many interesting applications (see end of this lecture and all of next lecture).

The Multiplicative Weights Algorithm

No-Regret Algorithm Design Principles

1. Past performance of actions should guide which action is chosen at each time step, with the probability of choosing an action increasing in its cumulative reward. (Recall that we need a randomized algorithm to have any chance.)

2. The probability of choosing a poorly performing action should decrease at an exponential rate.

The first principle is essential for obtaining regret sublinear in T, and the second for optimal regret bounds.

The MW algorithm maintains a weight, intuitively a "credibility," for each action. At each time step the algorithm chooses an action with probability proportional to its current weight. The weight of each action evolves over time according to the action's past performance.