Game Theory

Lecture 15

Learning Minmax Strategies in Zero-Sum Games

Introduction

- In this lecture, we will give a natural learning algorithm that players can use to play a game.
- To introduce it, we abstract away the game, and the other players, and start by asking how a player should make predictions in a sequential setting.
- As a simple example to keep in mind, consider the following toy model of predicting the stock market:
  - ➢ Every day the market goes up or down, and you must predict what it will do before it happens (so that you can either buy or short shares).
  - ➢ You don't have any information about what the market will do, and it may behave arbitrarily, so you can't hope to do well in an absolute sense.
  - ➢ However, every day, before you make your prediction, you get to hear the advice of a bunch of ***experts***, who make their own predictions.
  - ➢ These “experts" may or may not know what they are talking about, and you start off knowing nothing about them.
  - ➢ Nevertheless, you want to come up with a rule to aggregate their advice so that you end up doing (almost) ***as well as the best expert (whomever he might turn out to be) in hindsight***.
  - ➢ Sounds tough!

Introduction

- The algorithm we introduce is for repeated play of a matrix game with the guarantee that against any opponent, the player will perform ***nearly as well as the best fixed action in hindsight*** (also called the problem of ***combining expert advice*** or ***minimizing external regret***).
  - In a zero-sum game, such algorithms are ***guaranteed to approach or exceed the minimax value of the game***, and even provide a simple proof of the minimax theorem.

Warm-up: simple halving algorithm

Lets start with an even easier case:

- There are N experts who will make predictions in T rounds.
- At each round t, each expert i makes a prediction $p_i^t \in \{U, D\}$ (up or down).
- We (the algorithm) aggregate these predictions somehow, to make our own prediction $p_A^t \in \{U, D\}$. Then we learn the true outcome $o^t \in \{U, D\}$. If we predicted incorrectly (i.e. $p_A^t \neq o^t$), then we *made a mistake*.
- To make things easy, we will assume at first that there is one *perfect* expert who never makes a mistake (but we don't know who he is).

Can we find a strategy that is guaranteed to make at most $\log(N)$ mistakes?

We can, using the simple halving algorithm!

Warm-up: simple halving algorithm

Algorithm 1 The Halving Algorithm

Let $S^1 \leftarrow \{1, \dots, N\}$ be the set of all experts.
for $t = 1$ to T **do**
  Let $S_U^t = \{i \in S : p_i^t = U\}$ be the set of experts in S^t who predict up, and $S_D^t = S^t \setminus S_U^t$ be the set who predict down.
  Predict with the majority vote: If $|S_U^t| > |S_D^t|$, predict $p_A^t = U$, else predict $p_A^t = D$.
  Eliminate all experts that made a mistake: If $o^t = U$, then let $S^{t+1} = S_U^t$, else let $S^{t+1} = S_D^t$
end for

- Its not hard to see that the halving algorithm makes at most log N mistakes under the assumption that one expert is perfect:

Theorem 1 *If there is at least one perfect expert, the halving algorithm makes at most log N mistakes.*

Proof Since the algorithm predicts with the majority vote, every time it makes a mistake at some round *t*, at least half of the remaining experts have made a mistake and are eliminated, and hence:

$|S^{t+1}| \leq |S^t|/2$. On the other hand, the perfect expert is never eliminated, and hence $|S^t| \geq 1$ for all t. Since $|S^1| = N$, this means there can be at most $\log N$ mistakes. ■

Not bad – $\log N$ is pretty small even if N is large (e.g. if $N = 1024$, $\log N = 10$, if $N = 1,048,576$, $\log N = 20$), and doesn't grow with T, so even with a huge number of experts, the average number of mistakes made by this algorithm is tiny.

Warm-up: the iterated halving algorithm

- What if no expert is perfect? Suppose the best expert makes OPT mistakes. Can we find a way to make not too many more than OPT mistakes?
- The first approach you might try is the iterated halving algorithm:

Algorithm 2 The Iterated Halving Algorithm

Let $S^1 \leftarrow \{1, \dots, N\}$ be the set of all experts.
for $t = 1$ to T **do**
  If $|S^t| = 0$ **Reset**: Set $S^t \leftarrow \{1, \dots, N\}$.
  Let $S_U^t = \{i \in S : p_i^t = U\}$ be the set of experts in S^t who predict up, and $S_D^t = S^t \setminus S_U^t$ be the set who predict down.
  Predict with the majority vote: If $|S_U^t| > |S_D^t|$, predict $p_A^t = U$, else predict $p_A^t = D$.
  Eliminate all experts that made a mistake: If $o^T = U$, then let $S^{t+1} = S_U^t$, else let $S^{t+1} = S_D^t$
end for

Theorem 2 *The iterated halving algorithm makes at most* $\log(N)(OPT + 1)$ *mistakes.*

Proof As before, whenever the algorithm makes a mistake, we eliminate half of the experts, and so the algorithm can make at most $\log N$ mistakes between any two resets. But if we reset, it is because since the last reset, *every* expert has made a mistake: in particular, between any two resets, the *best* expert has made at least 1 mistake. This gives the claimed bound. ■

Warm-up: the weighted majority algorithm

- We should be able to do better though. The above algorithm is wasteful in that every time we reset, we forget what we have learned!

- The weighted majority algorithm can be viewed as a softer version of the halving algorithm: rather than eliminating experts who make mistakes, we just down-weight them:

Algorithm 3 The Weighted Majority Algorithm

Set weights $w_i^1 \leftarrow 1$ for all experts i.
for $t = 1$ to T **do**
  Let $W_U^t = \sum_{i:p_i^t=U} w_i$ be the weight of experts who predict up, and $W_D^t = \sum_{i:p_i^t=D} w_i$ be the weight of those who predict down.
  Predict with the weighted majority vote: If $W_U^t > W_D^t$, predict $p_A^t = U$, else predict $p_A^t = D$.
  Down-weight experts who made mistakes: For all i such that $p_i^t \neq o^t$, set $w_i^{t+1} \leftarrow w_i^t/2$
end for

Theorem 3 *The weighted majority algorithm makes at most* $2.4\,(OPT + \log(N))$ *mistakes.*

Note that $\log(N)$ is a fixed constant, so the ratio of mistakes the algorithm makes compared to OPT is just 2.4 in the limit – not great, but not bad.

Warm-up: the weighted majority algorithm

Proof Let M be the total number of mistakes that the algorithm makes, and let $W^t = \sum_i w_i^t$ be the total weight at step t. Note that on any round t in which the algorithm makes a mistake, at least half of the total weight (corresponding to experts who made mistakes) is cut in half, and so $W^{t+1} \leq (3/4)W^t$. Hence, we know that if the algorithm makes M mistakes, we have $W^T \leq N \cdot (3/4)^M$. Let i^* be the best expert.

We also know that $w_{i^*}^T = (1/2)^{\mathrm{OPT}}$, and so in particular, $W^T > (1/2)^{\mathrm{OPT}}$.

Combining these two observations we know:

$$\left(\frac{1}{2}\right)^{\mathrm{OPT}} \leq W^T \leq N \left(\frac{3}{4}\right)^M$$

$$\left(\frac{4}{3}\right)^M \leq N \cdot 2^{\mathrm{OPT}}$$

$$M \leq 2.4(\mathrm{OPT} + \log(N))$$

as claimed. ■

Towards a better algorithm

We've been doing well; lets get greedy. What do we want in an algorithm? We might want:

1. It to make only 1 times as many mistakes as the best expert in the limit, rather than 2.4 times...
2. It to be able to handle N distinct actions (a separate action for each expert), not just two (up and down)...
3. It to be able to handle experts having arbitrary costs in $[0, 1]$ at each round, not just binary costs (right vs. wrong)

Formally, we want an algorithm that works in the following framework:

In round $t = 1, \ldots, T$, the following happens:

- The player picks a probability distribution $p^t = (p_1^t, \ldots, p_N^t)$ over his strategies.
- The adversary picks a cost vector $\ell^t = (\ell_1^t, \ldots, \ell_N^t)$, where $\ell_i^t \in [0, 1]$ for all i.
- A strategy a^t is chosen according to the probability distribution p^t. The player incurs this strategy's cost and gets to know the entire cost vector.

What is the right benchmark?

The *best action sequence in hindsight* achieves a cost of $\sum_{t=1}^{T} \min_{i \in [N]} \ell_i^t$.

However, getting close to this number is generally hopeless as the following example shows.

Example *Suppose $N = 2$ and consider an adversary that chooses $\ell^t = (1, 0)$ if $p_1^t \geq 1/2$ and $\ell^t = (0, 1)$ otherwise. Then the expected cost of the player is at least $T/2$, while the best action sequence in hindsight has cost* 0.

Instead, we will swap the sum and the minimum, and compare to $L_{\min}^T = \min_{i \in [N]} \sum_{t=1}^{T} \ell_i^t$. That is, instead of comparing to the best action sequence in hindsight, we compare to the *best fixed action in hindsight*.

The expected cost of some algorithm $\mathcal{A}$ that uses probability distributions $p^1, \ldots, p^T$ against cost vectors $\ell^1, \ldots, \ell_T$ is given as $L_{\mathcal{A}}^T = \sum_{t=1}^{T} \sum_{i=1}^{N} p_i^t \ell_i^t$.

The difference of this cost and the cost of the best single strategy in hindsight is called **external regret**.

Definition *The* external regret *of algorithm $\mathcal{A}$ is defined as $R_{\mathcal{A}}^T = L_{\mathcal{A}}^T - L_{min}^T$.*

Definition *An algorithm is called* no-external-regret algorithm *if for any adversary and all T we have $R_{\mathcal{A}}^T = o(T)$.*

Polynomial Weights Algorithm

The polynomial weights algorithm can be viewed as a further smoothed version of the weighted majority algorithm, and has a parameter ϵ which controls how quickly it down-weights experts. Notably, it is *randomized*: rather than making deterministic decisions, it randomly chooses an expert to follow with probability proportional to their weight.

Algorithm 4 The Polynomial Weights Algorithm (PW)

Set weights $w_i^1 \leftarrow 1$ for all experts i.
for $t = 1$ to T **do**
  Let $W^t = \sum_{i=1}^{N} w_i^t$.
  Choose expert i with probability w_i^t / W^t.
  For each i, set $w_i^{t+1} \leftarrow w_i^t \cdot (1 - \epsilon \ell_i^t)$.
end for

Theorem 4 *For any sequence of losses, and any expert k:*

$$\frac{1}{T}\mathrm{E}[L_{PW}^T] \leq \frac{1}{T}L_k^T + \epsilon + \frac{\ln(N)}{\epsilon \cdot T}$$

In particular, setting $\epsilon = \sqrt{\frac{\ln(N)}{T}}$ *we get:*

$$\frac{1}{T}\mathrm{E}[L_{PW}^T] \leq \frac{1}{T}\min_k L_k^T + 2\sqrt{\frac{\ln(N)}{T}}$$

- In other words, the average loss of the algorithm quickly approaches the average loss of the best expert exactly, at a rate of $1/\sqrt{T}$.
- Note that this works against an arbitrary sequence of losses, which might be chosen adaptively by an adversary. This is pretty incredible. In particular, it means **we can use the polynomial weights algorithm to play a game**!
  - ➢ We simply let each of the “experts” correspond to an action in the game, and let the losses of the experts correspond to our costs in the game, given what the other players did.
    - ▪ The guarantee is that no matter what they do (even if they are trying explicitly to cause us high loss), we are guaranteed to obtain payoff nearly as high as that of the best action in hindsight!
    - ▪ In fact, to obtain this guarantee, we don't even need to know the payoff structure of our opponents; i.e., the polynomial weights algorithm is an **uncoupled dynamics** {to run the algorithm, all we need are the realized costs of each action, given what our opponents ended up doing.}

Proof Let F^t denote the expected loss of the polynomial weights algorithm at time t. By linearity of expectation, we have $\mathrm{E}[L_{PW}^T] = \sum_{t=1}^T F^t$. We also know that:

$$\mathbb{E}\left[\sum_{t=1}^{T} L_{PW}^t\right] = \sum_{t=1}^{T} \mathbb{E}[L_{PW}^t]$$

$$F^t = \frac{\sum_{i=1}^N w_i^t \ell_i^t}{W^t}$$

How does W^t change between rounds? We know that $W^1 = N$, and looking at the algorithm we see:

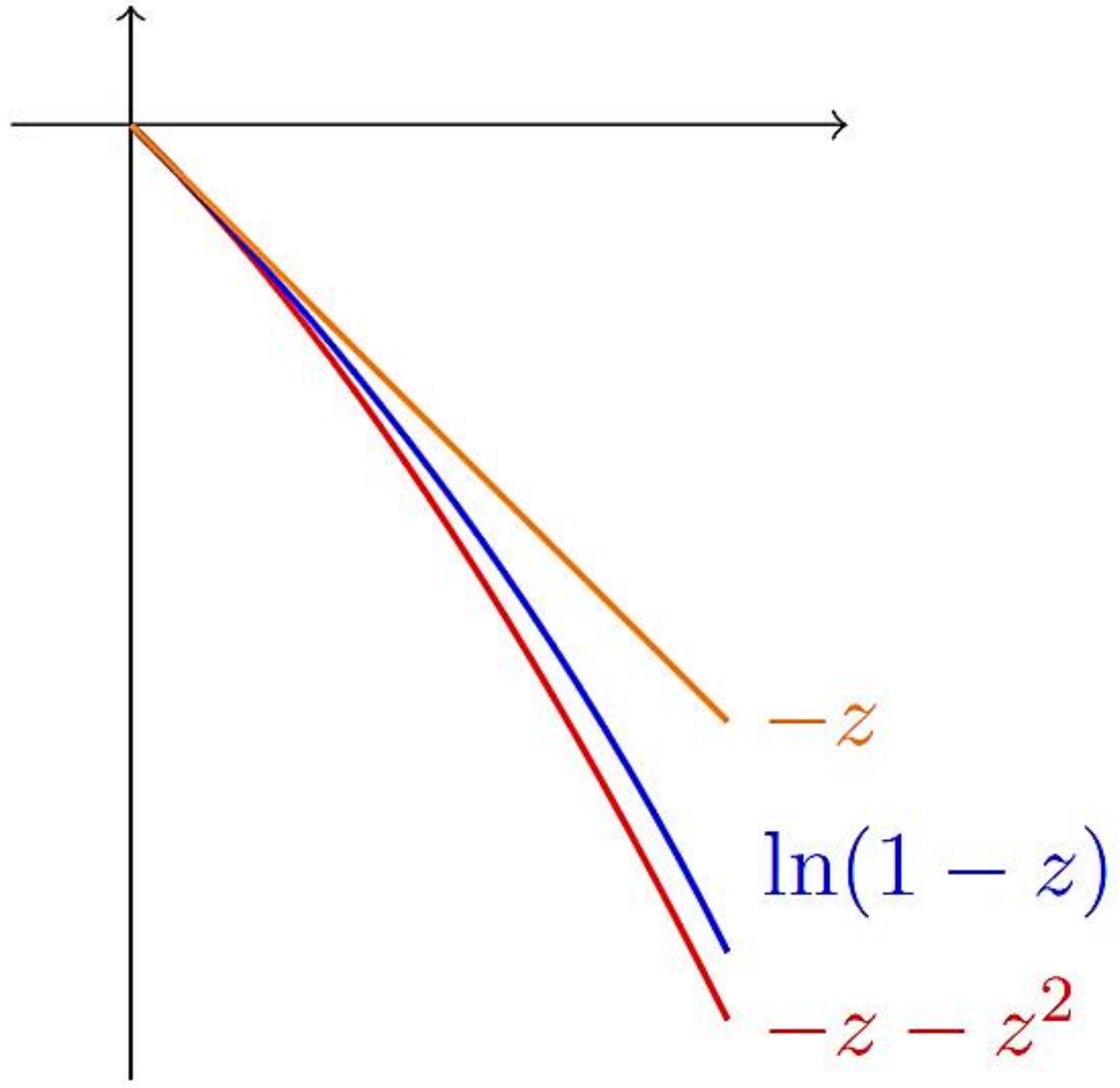
$$W^{t+1} = W^t - \sum_{i=1}^{N} \epsilon w_i^t \ell_i^t = W^t(1 - \epsilon F^t)$$

So by induction, we can write:

$$W^{T+1} = N \prod_{t=1}^{T} (1 - \epsilon F^t)$$

Taking the log, and using the fact that $\ln(1-z) \leq -z$, we can write:

$$\begin{aligned} \ln(W^{T+1}) &= \ln(N) + \sum_{t=1}^{T} \ln(1 - \epsilon F^t) \\ &\leq \ln(N) - \epsilon \sum_{t=1}^{T} F^t \\ &= \ln(N) - \epsilon \mathrm{E}[L_{PW}^T] \end{aligned}$$



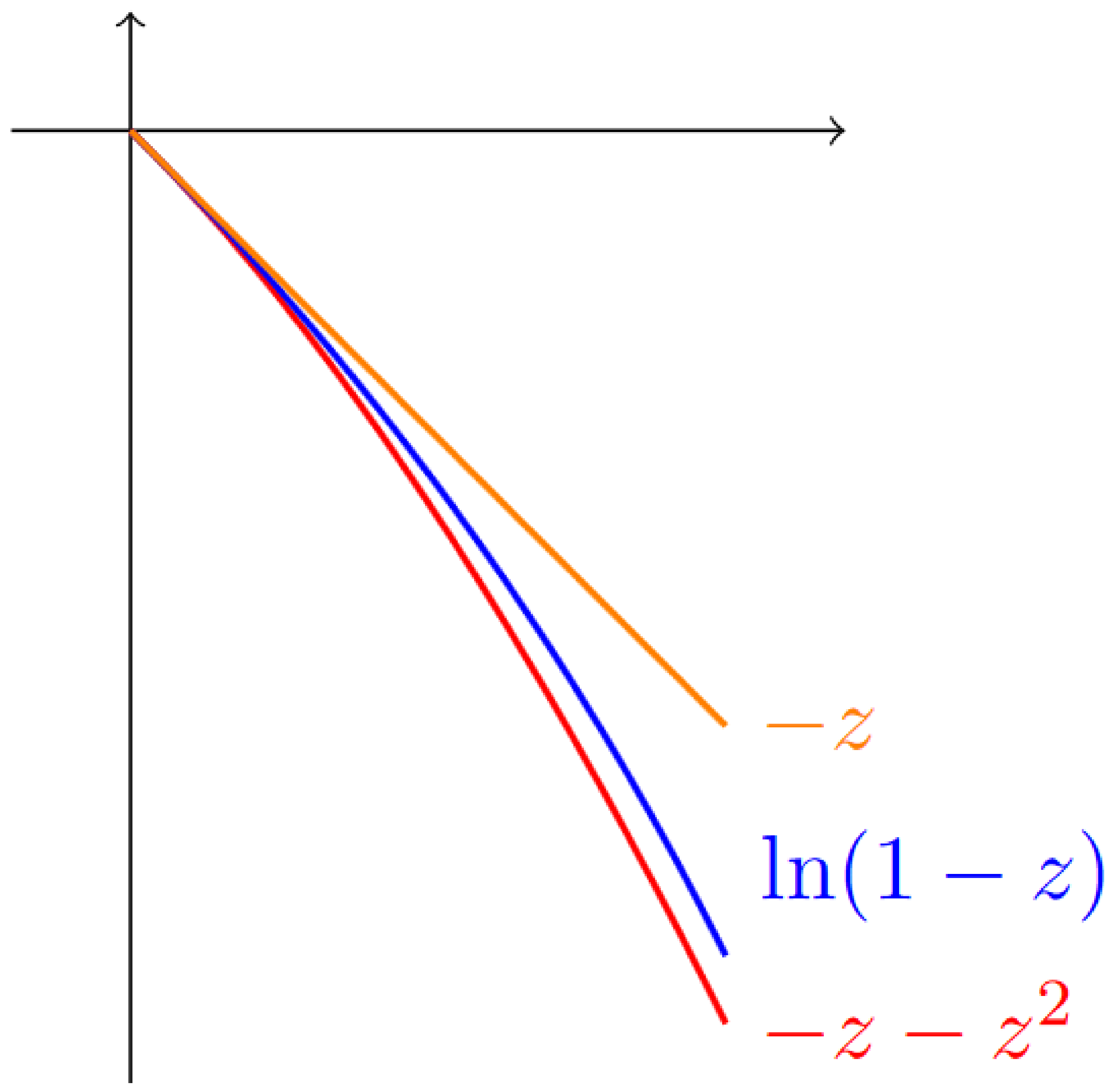
Similarly , we know that for every expert k:

$$\ln(W^{T+1}) \geq \ln(w_k^{T+1})$$

$$w_k^{T+1} = \prod_{t=1}^{T}(1 - \varepsilon \ell_k^t)$$

$$= \sum_{t=1}^{T} \ln(1 - \epsilon \ell_k^t)$$

(using the fact that $\ln(1 - z) \geq -z - z^2$ for $0 < z < \frac{1}{2}$)



$$\geq -\sum_{t=1}^{T} \epsilon \ell_k^t - \sum_{t=1}^{T} (\epsilon \ell_k^t)^2$$

$$\geq -\epsilon L_k^T - \epsilon^2 T$$

$$\ln(W^{T+1}) \leq \ln(N) - \varepsilon \mathbb{E}[L_{PW}^T]$$

Combining these two bounds, we get for all k.

$$\ln(N) - \epsilon\, \mathbb{E}[L_{PW}^T] \geq -\epsilon L_k^T - \epsilon^2 T$$

for all k. Dividing by ϵ and rearranging, we get:

$$\mathbb{E}[L_{PW}^T] \leq \min_k L_k^T + \epsilon T + \frac{\ln(N)}{\epsilon}$$

■

Polynomial Weights Algorithm ➔ Minimax Theorem

Recall *For an $n \times m$ matrix U (think about this as the payoff matrix in a two player zero sum game if you like):*

$$\max\min(U) = \max_{p \in \Delta[n]} \min_{y \in [m]} \sum_{i=1}^{n} p_i \cdot U(i,y)$$

$$\min\max(U) = \min_{q \in \Delta[m]} \max_{x \in [n]} \sum_{j=1}^{m} q_j \cdot U(x,j)$$

Note that we have defined things so that the player is optimizing over mixed strategies, but the opponent is optimizing over pure strategies. We could have let both optimize over mixed strategies, but this is without loss, since any player always has a pure strategy among her set of best responses.

Note that if U is a zero sum game, then $\max\min(U)$ represents the payoff that Max can guarantee if he goes first, and $\min\max(U)$ represents the payoff that he can guarantee if Min goes first.

It is apparent that playing second can only be an advantage: your strategy space is not limited by the first player's action, and you only have more information. In particular, this implies that for any game U:

$$\min\max(U) \geq \max\min(U)$$

Theorem (Von Neumann) *In any zero sum game U (at any NE):*

$$\min\max(U) = \max\min(U)$$

Polynomial Weights Algorithm ➔ Minimax Theorem

- The polynomial weights algorithm can provide a very simple, constructive proof of the minimax theorem!

Proof Suppose the theorem were false: That is, there is some game U for which $\min\max(U) > \max\min(U)$.

Write $v_1 = \min\max(U)$ and $v_2 = \max\min(U)$ (And so $v_1 = v_2 + \epsilon$ for some constant $\epsilon > 0$).

In other words, if Min has to go first, then Max can guarantee payoff at least v_1, but if Max is forced to go first, then Min can force Max to have payoff only v_2.

Lets consider what happens when Min and Max repeatedly play against each other as follows, for T rounds:

1. Min will play using the polynomial weights algorithm. i.e. at each round t, the weights w^t of the polynomial weights algorithm will form her mixed strategy, and she will sample an action at random from this distribution, updating based on the losses she experiences at that round.

2. Max will play the best response to Min's strategy. i.e. Max will play $x^t = \arg\max_x \mathrm{E}_{y \sim w^t}[U(x, y)]$.

Polynomial Weights Algorithm ➔ Minimax Theorem

Consider what we know about each of their average payoffs when they play in this manner. On the one hand, we know from the guarantee of the polynomial weights algorithm that:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}[U(x^t,y^t)] &\leq \frac{1}{T}\min_{y^*}\sum_{t=1}^{T}U(x^t,y^*)+\Delta(T)\\ &= \min_{y^*}\sum_{t=1}^{T}\frac{1}{T}U(x^t,y^*)+\Delta(T)\\ &= \min_{y^*}\mathrm{E}_{x\sim\bar{x}}[U(x,y^*)]+\Delta(T)\end{aligned}$$

where $\bar{x}$ is the mixed strategy that puts weight $1/T$ on each action x^t. $\Delta(T)$ is the regret bound of the polynomial weights algorithm – recall:

$$\Delta(T)=2\sqrt{\frac{\log n}{T}}.$$

But by definition, $\min_{y^*}\mathrm{E}_{x\sim\bar{x}}U(x,y^*)\leq\max\min(U)=v_2$ and so we know:

$$\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}[U(x^t,y^t)]\leq v_2+\Delta(T)$$

Polynomial Weights Algorithm ➔ Minimax Theorem

On the other hand, on each day t we know Max is best responding to Min's mixed strategy w^t. Thus:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\mathrm{E}[U(x^t, y^t)] &\geq \frac{1}{T}\sum_{t=1}^{T}\max_{x^*}\mathrm{E}_{y\sim w^t}[U(x^*, y)] \\ &\geq \frac{1}{T}\sum_{t=1}^{T} v_1 \\ &= v_1\end{aligned}$$

Combining these inequalities, we know:

$$v_1 \leq v_2 + \Delta(T)$$

Recall $v_1 = v_2 + \epsilon$, so:

$$\epsilon \leq \Delta(T)$$

but taking $T = \frac{16\ln(n)}{\epsilon^2}$ we get $\Delta(T) = \frac{\epsilon}{2}$ which implies:

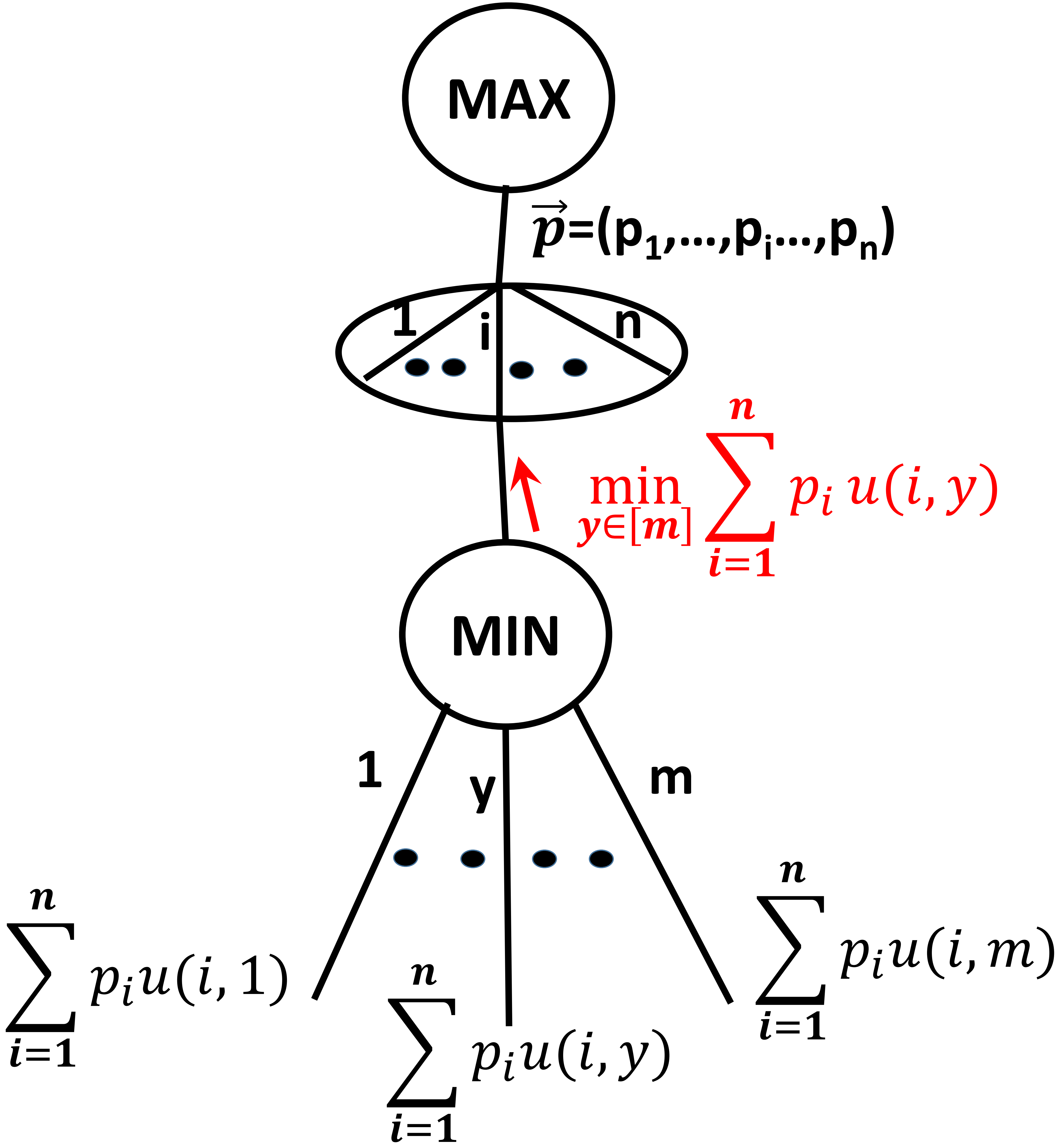
$$\epsilon \leq \epsilon/2$$

Since ϵ is positive, this is a contradiction and concludes the proof. ■

Polynomial Weights Algorithm ➔ Minimax Theorem

- This proof has highlighted the particularly amazing feature of the ***polynomial weights algorithm***:
  - It guarantees that no matter what happens, you do as well as if you had gotten to observe your opponent's strategy, and then best respond after the fact.
  - Using the polynomial weights algorithm guarantees that the player gets payoff quickly approaching the value of the game.
  - What's more, it does so without needing to know what the game is.
  - Note that at no point is the game matrix input to the PW algorithm!
  - The only information it needs to know is what the realized payoffs are for its actions, as it actually plays the game.
    - As such, it is an attractive algorithm to use in an interaction that you don't know much about...

Appendix



To compute max min strategy,
we need to specify the lower envelope of
MAX-player payoffs

		H	T
p	H	1,-1	-1,1
1-P	T	-1,1	1,-1