MIT 6.S890 — Topics in Multiagent Learning

Lecture 10

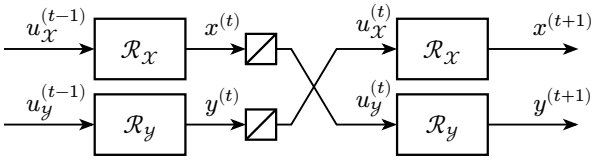
Learning in extensive-form games

Instructor: Prof. Gabriele Farina

1 Learning algorithms for extensive-form games

Several approaches for constructing no-regret algorithms for extensive-form games have been proposed. For one, extensive-form games are a particular instance of combinatorial games for which the multiplicative weights update algorithm can be implemented efficiently in the reduced normal form of the game, despite the exponential size. We will see more details about this in a later class.

As explained in Lecture 9, the natural representation of strategies to define learning in extensive-form games is the *sequence-form representation*. Indeed, in that representation utility functions are linear and the strategy set of each player a convex polytope, aligning with the requirements of the regret minimization framework. Thanks to the sequence form representation of strategiesall the results about external regret minimization we have seen so far apply to extensive-form games as well, including for example the fact that a Nash equilibrium in a two-player zero-sum game can be found by letting two regret minimizers play against each other by exchanging equence-form strategies at every iteration according to the canonical learning setup



Another example is the computation of coarse correlated equilibria in any multiplayer extensive-form game via external regret minimization, or computation of best responses against static opponents.

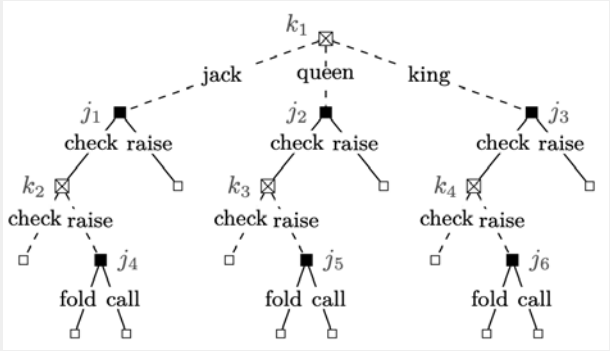
To construct an external regret minimizer that outputs sequence-form strategies, several approaches can be followed. For one, we have seen that one can always use the online projected gradient ascent algorithm, which is a particular instantiation of the online mirror descent (OMD) algorithm. The drawback of such approach is that it requires projecting onto the polytope of sequence form strategies, which might be laborious. Alternative regularizers (*i.e.*, distance-generating functions) that render projection easier have been proposed. However, for today we focus on a different approach, which has been extremely popular in practice: the *counterfactual regret minimization (CFR)* algorithm.

2 The CFR algorithm

The idea of the CFR algorithm is simple: construct a regret minimizer for the whole tree-form problem starting from *local* regret minimizers at each decision point, each learning what actions to play at that decision point.

⋆These notes are class material that has not undergone formal peer review. The TAs and I are grateful for any reports of typos.

Example 2.1. As an example, consider the TFDP faced by Player 1 in the game of Kuhn poker [Kuh50], which we already introduced in Lecture 9. The black nodes are the *decision points* of the player, and the white nodes are the *observation points*.



Since the player has six decision point—denoted $j_1, ..., j_6$ in the figure—the CFR algorithm will use six local regret minimizer, which we denote $\mathcal{R}_1, ..., \mathcal{R}_6$. Each regret minimizer $\mathcal{R}_j$ will be responsible for outputting a local strategy $b_j \in \Delta(A_j)$ for the decision point j.

The local distributions output by the different local regret minimizers is then combined to form a *sequence-form strategy* that plays according to the local distributions at each decision point.

2.1 Where the magic happens: counterfactual utilities

What is the training signal that each local regret minimizer receives? In other words, what is the utility that the regret minimizer at decision point j observes? The answer is the *counterfactual utility*.

Remember that in the sequence form representation, the dimensionality of the strategy vectors matches the number of actions controlled by the players. Hence, the gradient vector received by the regret minimizer has one entry per each action controlled by the player, intuitively representing whether the "probability flow" passing through that action scores well or poorly. The idea of counterfactual utilities is to use as training signal for every $\mathcal{R}_j$ the vector of expected utilities in the subtrees rooted at each of the actions $a \in A_j$.

It can be shown that the regret cumulated by the CFR algorithm satisfies the following bound.

Theorem 2.1. Let $\mathrm{Reg}_j^{(T)}$ $(j \in \mathcal{J})$ denote the regret cumulated up to time T by each of the regret minimizers $\mathcal{R}_j$. Then, the regret $\mathrm{Reg}^{(T)}$ cumulated by Algorithm 1 up to time T satisfies

$$\mathrm{Reg}^{(T)} \leq \sum_{j \in \mathcal{J}} \max\left\{0, \mathrm{Reg}_j^{(T)}\right\}.$$

It is then immediate to see that if each $\mathrm{Reg}_j^{(T)}$ grows sublinearly in T, then so does $\mathrm{Reg}^{(T)}$.

2.2 Pseudocode for CFR

In order to formally introduce pseudocode, we recall a bit of notation to deal with tree-form decision processes.

Notation for tree-form decision processes. We recall the following notation for dealing with tree-form decision processes (TFDPs), which we introduced in Lecture 9. The notation is also summarized in Table 1.

- We denote the set of decision points in the TFDP as $\mathcal{J}$, and the set of observation points as $\mathcal{K}$. At each decision point $j \in \mathcal{J}$, the agent selects an action from the set A_j of available actions. At each

Symbol	
$\mathcal{J}$	Set of decision points
A_j	Set of legal actions at decision point $j \in \mathcal{J}$
$\mathcal{K}$	Set of observation points
S_k	Set of possible signals at observation point $k \in \mathcal{K}$
ρ	Transition function: • given $j \in \mathcal{J}$ and $a \in A_j$, $\rho(j,a)$ returns the next decision or observation point v in $\mathcal{J} \cup \mathcal{K}$ in the decision tree that is reached after selecting legal action $a \in j$, or $\perp$ if the decision process ends; • given $k \in \mathcal{K}$ and $s \in S_k$, $\rho(k,s)$ returns the next decision or observation point $v \in \mathcal{J} \cup K$ in the decision tree that is reached after observing signal s at k, or $\perp$ if the decision process ends
Σ	Set of sequences, defined as $\Sigma := \{(j,a) : j \in J, a \in A_j\}$
p_j	Parent sequence of decision point $j \in \mathcal{J}$, defined as the last sequence (decision point-action pair) on the path from the root of the TFDP to decision point j; if the agent does not act before j, $p_j = \varnothing$

observation point $k \in \mathcal{K}$, the agent observes a signal s_k from the environment out of a set of possible signals S_k.

- We denote by ρ the transition function of the process. Picking action $a \in A_j$ at decision point $j \in \mathcal{J}$ results in the process transitioning to $\rho(j,a) \in \mathcal{J} \cup \mathcal{K} \cup \{\perp\}$, where $\perp$ denotes the end of the decision process. Similarly, the process transitions to $\rho(k,s) \in \mathcal{J} \cup \mathcal{K} \cup \{\perp\}$ after the agent observes signal $s \in S_k$ at observation point $k \in \mathcal{K}$.
- A pair (j,a) where $j \in \mathcal{J}$ and $a \in A_j$ is called a *sequence*. The set of all sequences is denoted as $\Sigma := \{(j,a) : j \in \mathcal{J}, a \in A_j\}$. For notational convenience, we will often denote an element (j,a) in Σ as ja without using parentheses.
- Given a decision point $j \in \mathcal{J}$, we denote by p_j its *parent sequence*, defined as the last sequence (that is, decision point-action pair) encountered on the path from the root of the decision process to j. If the agent does not act before j (that is, j is the root of the process or only observation points are encountered on the path from the root to j), we let $p_j = \varnothing$.

Example 2.2. As an example, consider again the TFDP faced by Player 1 in the game of Kuhn poker [Kuh50], which was also recalled above in Example 2.1. We have that $\mathcal{J} = \{j_1, ..., j_6\}$ and $\mathcal{K} = \{k_1, ..., k_4\}$. We have:

$$A_{j_1} = S_{k_4} = \{\mathsf{check}, \mathsf{raise}\}, \qquad A_{j_5} = \{\mathsf{fold}, \mathsf{call}\}, \qquad S_{k_1} = \{\mathsf{jack}, \mathsf{queen}, \mathsf{king}\}$$
$$p_{j_4} = (j_1, \mathsf{check}), \qquad p_{j_6} = (j_3, \mathsf{check}), \qquad p_{j_1} = p_{j_2} = p_{j_3} = \varnothing.$$

Furthermore,

$$\rho(k_3, \mathsf{check}) = \rho(j_2, \mathsf{raise}) = \perp, \qquad \rho(k_1, \mathsf{king}) = j_3, \qquad \rho(j_2, \mathsf{check}) = k_3.$$

Notation for the components of vectors. Any vector $x \in \mathbb{R}^{\Sigma}$ has, by definition, as many components as sequences Σ. The component corresponding to a specific sequence $ja \in \Sigma$ is denoted as $x[ja]$. Similarly, given any decision point $j \in \mathcal{J}$, any vector $x \in \mathbb{R}^{A_j}$ has as many components as the number of actions at j. The component corresponding to a specific action $a \in A_j$ is denoted $x[a]$.

CFR algorithm. Pseudocode for CFR is given in Algorithm 1. Note that the implementation is parametric on the regret minimization algorithms $\mathcal{R}_j$ run locally at each decision point. Any regret minimizer $\mathcal{R}_j$ for simplex domains can be used to solve the local regret minimization problems. Popular options are the regret matching algorithm, and the regret matching plus algorithm (Lecture 5).

Algorithm 1: CFR regret minimizer

Data: $\mathcal{R}_j$ regret minimizer for $\Delta(A_j)$; one for each decision point $j \in \mathcal{J}$ of the TFDP.

function NextStrategy()
  [▷ Step 1: we ask each of the $\mathcal{R}_j$ for their next strategy local at each decision point]
  for each decision point $j \in \mathcal{J}$
    $b_j^{(t)} \in \Delta(A_j) \leftarrow \mathcal{R}_j$.NextStrategy()

  [▷ Step 2: we construct the sequence-form representation of the strategy that plays according to the distribution $b_j^{(t)}$ at each decision point $j \in \mathcal{J}$]
  $x^{(t)} = \mathbf{0} \in \mathbb{R}^{\Sigma}$
  for each decision point $j \in \mathcal{J}$ in *top-down traversal* order in the TFDP
    for each action $a \in A_j$
      if $p_j = \varnothing$
        $x^{(t)}[ja] \leftarrow b_j^{(t)}[a]$
      else
        $x^{(t)}[ja] \leftarrow x^{(t)}[p_j] \cdot b_j^{(t)}[a]$

  [▷ You should convince yourself that the vector $x^{(t)}$ we just filled in above is a valid sequence-form strategy, that is, it satisfies the required consistency constraints we saw in Lecture 9. In symbols, $x^{(t)} \in Q$]
  return $x^{(t)}$

function ObserveUtility($u^{(t)} \in \mathbb{R}^{\Sigma}$)
  [▷ Step 1: we compute the expected utility for each subtree rooted at each node $v \in \mathcal{J} \cup \mathcal{K}$]
  $V^{(t)} \leftarrow$ empty dictionary [▷ eventually, it will map keys $\mathcal{J} \cup \mathcal{K} \cup \{\bot\}$ to real numbers]
  $V^{(t)}[\bot] \leftarrow 0$
  for each node in the tree $v \in \mathcal{J} \cup \mathcal{K}$ in *bottom-up traversal* order in the TFDP
    if $v \in \mathcal{J}$
      let $j \leftarrow v$
      $V^{(t)}[j] \leftarrow \sum_{a \in A_j} b_j^{(t)}[a] \cdot \left(u^{(t)}[ja] + V^{(t)}[\rho(j,a)]\right)$
    else
      let $k \leftarrow v$
      $V^{(t)}[k] \leftarrow \sum_{s \in S_k} V^{(t)}[\rho(k,s)]$

  [▷ Step 2: at each decision point $j \in \mathcal{J}$, we now construct a local utility vector $u_j^{(t)}$ called *counterfactual utility*]
  for each decision point $j \in \mathcal{J}$
    $u_j^{(t)} \leftarrow \mathbf{0} \in \mathbb{R}^{A_j}$
    for each action $a \in A_j$
      $u_j^{(t)}[a] \leftarrow u^{(t)}[ja] + V^{(t)}[\rho(j,a)]$
    $\mathcal{R}_j$.ObserveUtility($u_j^{(t)}$)

Bibliography

[Kuh50] H. W. Kuhn, "A Simplified Two-Person Poker," *Contributions to the Theory of Games*, vol. 1. in Annals of Mathematics Studies, 24, vol. 1. Princeton University Press, Princeton, New Jersey, pp. 97–103, 1950.

MIT OpenCourseWare
https://ocw.mit.edu

6.S890 Topics in Multiagent Learning

Fall 2024

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms