

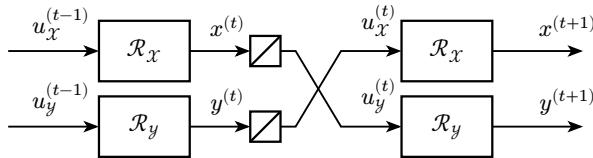
Lecture 11

Learning in extensive-form games

Instructor: Prof. Gabriele Farina (gfarina@mit.edu)*

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 10, 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 strategies, all 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 sequence-form strategies at every iteration according to the canonical learning setup



Another example is the computation of coarse correlated equilibria in any multi-player 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

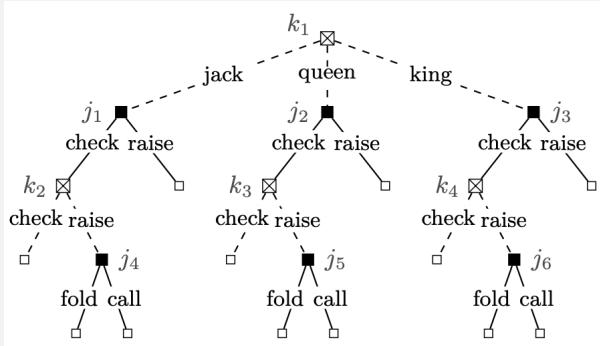
*These notes are class material that has not undergone formal peer review. The TAs and I are grateful for any reports of typos.

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.

L11.1 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.

Example L11.1. As an example, consider the TFDP faced by Player 1 in the game of Kuhn poker [1], which we already introduced in Lecture 10. The black nodes are the decision points of the player, and the white nodes are the observation points.



Since the player has six decision points—denoted j_1, \dots, j_6 in the figure—the CFR algorithm will use six local regret minimizers, which we denote R_1, \dots, R_6 . Each regret minimizer 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.

L11.1.1 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 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 L11.1. *Let $\text{Reg}_j^{(T)}$, for $j \in \mathcal{J}$, denote the regret cumulated up to time T by each of the regret minimizers R_j . Then, the regret $\text{Reg}^{(T)}$ cumulated by Algorithm L11.1 up to time T satisfies*

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

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

In order to formally introduce counterfactual utility, 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 10. The notation is also summarized in [Section L11.1.1](#).

- 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 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 $p_j = \emptyset$.

Example L11.2. As an example, consider again the TFDP faced by Player 1 in the game of Kuhn poker [1], which was also recalled above in [Example L11.1](#). We have that $J = \{j_1, \dots, j_6\}$ and $K = \{k_1, \dots, k_4\}$. We have:

$$\begin{aligned} A_{j_1} &= S_{k_4} = \{\text{check, raise}\}, & A_{j_5} &= \{\text{fold, call}\}, & S_{k_1} &= \{\text{jack, queen, king}\} \\ p_{j_4} &= (j_1, \text{check}), & p_{j_6} &= (j_3, \text{check}), & p_{j_1} &= p_{j_2} = p_{j_3} = \emptyset. \end{aligned}$$

Furthermore,

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

► **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 $j \in J$, 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 $x[a]$.

Symbol	Description
\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: <ul style="list-style-type: none"> 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 $v \in \mathcal{J} \cup \mathcal{K}$ 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 $\Sigma := \{(j, a) : j \in \mathcal{J}, a \in A_j\}$
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 , $p_j = \emptyset$.

L11.1.2 Pseudocode for CFR

Pseudocode for CFR is given in [Algorithm L11.1](#). Note that the implementation is parametric on the regret minimization algorithms R_j run locally at each decision point. Any regret minimizer 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 8).

Algorithm L11.1: CFR regret minimizer

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

```

1 function NEXTSTRATEGY()
2   |▷ Step 1: ask each of the  $R_j$  for their next strategy local at each decision point]
3   |for each decision point  $j \in \mathcal{J}$  do
4   |  | $b_j^{(t)} \in \Delta(A_j) \leftarrow R_j.\text{NEXTSTRATEGY}()$ 
5   |▷ Step 2: we construct the sequence-form representation of the strategy that
6   |  |plays according to the distribution  $b_j^{(t)}$  at each decision point  $j \in \mathcal{J}$ 
7   |  | $x^{(t)} = 0 \in \mathbb{R}^{\Sigma}$ 
8   |for each decision point  $j \in \mathcal{J}$  in top-down traversal order in the TFDP do
9   |  |for each action  $a \in A_j$  do
10  |  |  |if  $p_j = \emptyset$  then
11  |  |  |  | $x^{(t)}[ja] \leftarrow b_j^{(t)}[a]$ 
12  |  |  |else
13  |  |  |  | $x^{(t)}[ja] \leftarrow x^{(t)}[p_j] \cdot b_j^{(t)}[a]$ 
14  |  |▷ You should convince yourself that the vector  $x^{(t)}$  we just filled in above is a
15  |  |  |valid sequence-form strategy, that is, it satisfies the required consistency
16  |  |  |constraints we saw in Lecture 10. In symbols,  $x^{(t)} \in \mathcal{Q}$ 
17  |  |return  $x^{(t)}$ 
18 function OBSERVEUTILITY( $g^{(t)} \in \mathbb{R}^{|\Sigma|}$ )
19   |▷ Step 1: we compute the expected utility for each subtree rooted at each node
20   |  | $v \in \mathcal{J} \cup \mathcal{K}$ 
21   |  | $V^{(t)} \leftarrow$  empty dictionary      |▷ eventually, it will map keys  $\mathcal{J} \cup \mathcal{K} \cup \{\perp\}$  to real
22   |  |  |numbers|
23   |  | $V^{(t)}[\perp] \leftarrow 0$ 
24   |for each node in the tree  $v \in \mathcal{J} \cup \mathcal{K}$  in bottom-up traversal order in the TFDP do
25   |  |if  $v \in \mathcal{J}$  then
26   |  |  |Let  $j \leftarrow v$ 
27   |  |  | $V^{(t)}[j] \leftarrow \sum_{a \in A_j} b_j^{(t)}[a] \cdot (g^{(t)}[ja] + V^{(t)}[\rho(j, a)])$ 
28   |  |else
29   |  |  |Let  $k \leftarrow v$ 
30   |  |  | $V^{(t)}[k] \leftarrow \sum_{s \in S_k} V^{(t)}[\rho(k, s)]$ 
31   |▷ Step 2: at each decision point  $j \in \mathcal{J}$ , we now construct a local utility vector
32   |  | $g_j^{(t)}$  called counterfactual utility
33   |for each decision point  $j \in \mathcal{J}$  do
34   |  | $g_j^{(t)} \leftarrow 0 \in \mathbb{R}^{A_j}$ 
35   |  |for each action  $a \in A_j$  do
36   |  |  | $g_j^{(t)}[a] \leftarrow g^{(t)}[ja] + V^{(t)}[\rho(j, a)]$ 
37   |  | $R_j.\text{OBSERVEUTILITY}(g_j^{(t)})$ 
```

L11.1.3 Learning using self-play

The CFR algorithm can be used to learn a Nash equilibrium in a two-player zero-sum game by letting two regret minimizers play against each other. The two regret minimizers exchange their sequence-form strategies at every iteration according to the canonical learning setup.

L11.2 Bibliography for this lecture

- [1] H. W. Kuhn. “A Simplified Two-Person Poker”. In: *Contributions to the Theory of Games*. Ed. by H. W. Kuhn and A. W. Tucker. Vol. 1. Annals of Mathematics Studies, 24. Princeton, New Jersey: Princeton University Press, 1950, pp. 97–103.

Changelog

- 2025-10-09: Fixed typos (thanks Josh Rountree!).