15-888 Computational Game Solving (Fall 2023) Released on: Oct. 6, 2023

Homework 2

Student: [*** Your Andrew ID here ***] **Due on: Oct. 23, 2023, beginning of class**

Instructions Please typeset your solution in the tex file provided in the homework zip. Attach a readable printout of your CFR code at the end of the document. Turn in your printed solution at the beginning of class on October 23rd. Don't forget to fill in your student ID above.

1 Optimistic multiplicative weights update (50 points)

Multiplicative weights update (MWU) and its predictive variant called *optimistic multiplicative weights update (OMWU)* are popular regret minimization algorithms for the probability simplex

$$\Delta^n \coloneqq \{(x_1, \dots, x_n) \in \mathbb{R}^n_{\geq 0} : x_1 + \cdots + x_n = 1\}. \tag{1}$$

They enjoy many strong theoretical properties, and were involved in a series of important papers in game theory. In this question, you will derive and analyze MWU and OMWU from first principles.

1.1 The negative entropy regularizer (10 points)

OMWU is just a special name for the predictive FTRL algorithm when the regularizer $\omega : \Delta^n \to \mathbb{R}$ is chosen to be the *negative entropy regularizer*

$$\omega(\boldsymbol{x}) \coloneqq \sum_{i=1}^{n} x_i \log(x_i).$$

To avoid annoying issues with the logarithm of 0, we will only ever evaluate and differentiate ω in the relative interior of Δ^n, that is the set

$$\operatorname{relint} \Delta^n = \{(x_1, \dots, x_n) \in \mathbb{R}^n_{>0} : x_1 + \cdots + x_n = 1\}$$

(note the strict inequality $\mathbb{R}_{>0}$, as opposed to $\mathbb{R}_{\geq 0}$ in (1)).

For ω to be a valid choice of regularizer in predictive FTRL, we need to check that ω is 1-strongly convex with respect to some norm. In particular, it turns out that ω is 1-strongly convex both with respect to the Euclidean norm

$$\|\boldsymbol{x}\|_2 \coloneqq \sqrt{\sum_{i=1}^{n} x_i^2} \qquad \forall\, \boldsymbol{x} \in \mathbb{R}^n$$

and with respect to the ℓ_1 norm

$$\|\boldsymbol{x}\|_1 \coloneqq \sum_{i=1}^{n} |x_i| \qquad \forall\, \boldsymbol{x} \in \mathbb{R}^n.$$

The easiest way to verify strong convexity in this case passes through the following well-known characterization.

Lemma 1. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a convex set, $f : \mathcal{X} \to \mathbb{R}$ be a twice-differentiable function with Hessian matrix $\nabla^2 f(\boldsymbol{x})$ at every $\boldsymbol{x} \in \mathcal{X}$, and $\|\cdot\|$ be a norm. If

$$\boldsymbol{u}^\top \nabla^2 f(\boldsymbol{x})\, \boldsymbol{u} \geq \|\boldsymbol{u}\|^2 \qquad \forall\, \boldsymbol{u} \in \mathbb{R}^n, \boldsymbol{x} \in \mathcal{X},$$

then f is 1-strongly convex on $\mathcal{X}$ with respect to norm $\|\cdot\|$.

In the next two exercises you will use Lemma 1 to verify that ω is 1-strongly convex on relint Δ^n with respect to $\|\cdot\|_2$ and $\|\cdot\|_1$.

Problem 1.1 (5 points). Apply Lemma 1 for $\mathcal{X} = \operatorname{relint}\Delta^n$, $f = \omega$, and $\|\cdot\| = \|\cdot\|_2$ and conclude that ω is 1-strongly convex with respect to the Euclidean norm on relint Δ^n.

★ Hint: the Hessian matrix of ω is particularly nice. Start by working that out first.
★ Hint: at some point, it might be useful to argue that $1/x_i \geq 1$ for any $i \in \{1, \dots, n\}$ whenever $\boldsymbol{x} \in \operatorname{relint}\Delta^n$.

Solution. [*** Your solution here ***] □

Problem 1.2 (5 points). Apply Lemma 1 for $\mathcal{X} = \operatorname{relint}\Delta^n$, $f = \omega$, and $\|\cdot\| = \|\cdot\|_1$ and conclude that ω is 1-strongly convex with respect to the ℓ_1 norm on relint Δ^n.

★ Hint: The Cauchy-Schwarz inequality asserts that for any pair of vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$,

$$\left(\sum_{i=1}^n a_i b_i\right)^2 \leq \left(\sum_{i=1}^n a_i^2\right)\left(\sum_{i=1}^n b_i^2\right). \tag{2}$$

Now, let $\boldsymbol{x} \in \operatorname{relint}\Delta^n$ and $\boldsymbol{u} \in \mathbb{R}^n$, and consider the vectors $\boldsymbol{a} := (u_1/\sqrt{x_i}, \dots, u_n/\sqrt{x_n})$ and $\boldsymbol{b} := (\sqrt{x_1}, \dots, \sqrt{x_n})$. What happens if you plug them into (2)? Don't forget that $x_1 + \cdots + x_n = 1$ since $\boldsymbol{x} \in \operatorname{relint}\Delta^n$.

Solution. [*** Your solution here ***] □

1.2 Gradient of ω and of its conjugate (15 points)

In this subsection, you will derive a formula for the gradient of ω and for the gradient of its convex conjugate. Let's start with the gradient.

Problem 1.3 (5 points). Give an expression for the gradient of ω at any point $\boldsymbol{x} \in \operatorname{relint}\Delta^n$.

Solution. [*** Your solution here ***] □

Now, let's focus on the gradient of the convex conjugate of ω, that is, the solution to the optimization problem

$$\nabla\omega^*(\boldsymbol{g}) := \operatorname*{arg\,max}_{\hat{\boldsymbol{x}} \in \operatorname{relint}\Delta^n} \{\boldsymbol{g}^\top \hat{\boldsymbol{x}} - \omega(\hat{\boldsymbol{x}})\}. \tag{3}$$

Problem (3) is a *constrained* optimization problem, since the optimization variable $\hat{\boldsymbol{x}}$ is constrained to satisfy $\hat{\boldsymbol{x}} \in \operatorname{relint}\Delta^n$. Call $\boldsymbol{x}^*$ the optimal solution to (3). As a result of an important theorem in optimization theory (the Lagrange multiplier theorem), there exists a constant (called *Lagrange multiplier*) $\alpha \in \mathbb{R}$ such that

$$\boldsymbol{g} - \nabla\omega(\boldsymbol{x}^*) = \alpha\mathbf{1}, \tag{4}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the vector of all ones.

Problem 1.4 (5 points). Plug in the expression for the gradient of ω that you developed in Problem 1.3 into (4). Note that (4) is a vector equation, and therefore it is equivalent to a system of n scalar equations. Isolate and solve for x_i^* for every $i \in \{1, \dots, n\}$, and show that

$$x_i^* = e^{-1-\alpha} \cdot e^{g_i} \qquad \forall\, i \in \{1, \dots, n\}. \tag{5}$$

Solution. [*** Your solution here ***] □

Problem 1.5 (5 points). Use Equation (5) together with the fact that the sum of the entries of $\boldsymbol{x}^* \in \operatorname{relint} \Delta^n$ must be 1 to solve for the value of the Lagrange multiplier α. Then, plug in the value of α to conclude that for any $\boldsymbol{g} \in \mathbb{R}^n$, $\nabla\omega^*(\boldsymbol{g})$—that is, the solution to the argmax in (3)—satisfies

$$x_i^* = \frac{e^{g_i}}{\sum_{j=1}^n e^{g_j}} \qquad \forall\, i \in \{1, \dots, n\}.$$

Solution. [*** Your solution here ***] □

1.3 OMWU as predictive FTRL (15 points)

Now that we verified that ω is 1-strongly convex, we can safely run predictive FTRL with ω as a regularizer. As a reminder, predictive FTRL is the algorithm recalled in Algorithm 1. In our case, $\mathcal{X}$ will be the relative interior $\operatorname{relint} \Delta^n$ of the probability simplex Δ^n, the regularizer φ will be the negative entropy function ω, and $\eta > 0$ will be a generic stepsize. The resulting algorithm is called OMWU.

Algorithm 1: Predictive FTRL

```
Data: $\mathcal{X} \subseteq \mathbb{R}^n$ convex domain
      $\varphi : \mathcal{X} \to \mathbb{R}_{\geq 0}$ strongly convex regularizer
      $\eta > 0$ step-size parameter
1 $\boldsymbol{L}^0 \leftarrow \boldsymbol{0} \in \mathbb{R}^n$
2 function NextStrategy($\boldsymbol{m}^t$)
      [▷ Set $\boldsymbol{m}^t = \boldsymbol{0}$ for non-predictive version]
3     return $\arg\max_{\hat{\boldsymbol{x}} \in \mathcal{X}} \left\{ (\boldsymbol{L}^{t-1} + \boldsymbol{m}^t)^\top \hat{\boldsymbol{x}} - \frac{1}{\eta}\varphi(\hat{\boldsymbol{x}}) \right\}$
4 function ObserveUtility($\boldsymbol{\ell}^t$)
5     $\boldsymbol{L}^t \leftarrow \boldsymbol{L}^{t-1} + \boldsymbol{\ell}^t$
```

Algorithm 2: Predictive OMD

```
Data: $\mathcal{X} \subseteq \mathbb{R}^n$ convex domain
      $\varphi : \mathcal{X} \to \mathbb{R}_{\geq 0}$ strongly convex regularizer
      $\eta > 0$ step-size parameter
1 $\boldsymbol{z}^0 \leftarrow \arg\min_{\boldsymbol{z} \in \mathcal{X}} \omega(\boldsymbol{z})$
2 function NextStrategy($\boldsymbol{m}^t$)
      [▷ Set $\boldsymbol{m}^t = \boldsymbol{0}$ for non-predictive version]
3     return $\arg\max_{\hat{\boldsymbol{x}} \in \mathcal{X}} \left\{ (\boldsymbol{m}^t)^\top \hat{\boldsymbol{x}} - \frac{1}{\eta} D_\varphi(\hat{\boldsymbol{x}} \,\|\, \boldsymbol{z}^{t-1}) \right\}$
4 function ObserveUtility($\boldsymbol{\ell}^t$)
5     $\boldsymbol{z}^t \leftarrow \arg\max_{\hat{\boldsymbol{z}} \in \mathcal{X}} \left\{ (\boldsymbol{\ell}^t)^\top \hat{\boldsymbol{z}} - \frac{1}{\eta} D_\varphi(\hat{\boldsymbol{z}} \,\|\, \boldsymbol{z}^{t-1}) \right\}$
```

Problem 1.6 (10 points). Use the characterization of $\nabla\omega^*(\boldsymbol{g})$ given in the statement of Problem 1.5 to prove that at times $t = 2, 3, \dots$, for all $i \in \{1, \dots, n\}$, the strategies $\boldsymbol{x}^t \in \Delta^n$ produced by OMWU satisfy[a]

$$x_i^t = \frac{x_i^{t-1} \, \exp\{\eta(\ell_i^{t-1} - m_i^{t-1} + m_i^t)\}}{\sum_{j=1}^n x_j^{t-1} \, \exp\{\eta(\ell_j^{t-1} - m_j^{t-1} + m_j^t)\}},$$

[a]For readability we used the notation $\exp\{\bullet\}$ to mean $e^{\bullet}$.

Solution. [*** Your solution here ***] □

Since OMWU is just predictive FTRL, we can use the known regret bound for predictive FTRL we saw in class—and the following proposition recalls—to give a regret guarantee for OMWU.

Proposition 1. Consider the predictive FTRL algorithm given in Algorithm 1. Let Ω denote the range of φ over $\mathcal{X}$, that is, $\Omega := \max_{\boldsymbol{x},\boldsymbol{x}' \in \mathcal{X}}\{\varphi(\boldsymbol{x}) - \varphi(\boldsymbol{x}')\}$. For any T, the regret accumulated up to time T satisfies

$$R^T \le \frac{\Omega}{\eta} + \eta \sum_{t=1}^{T} \|\boldsymbol{\ell}^t - \boldsymbol{m}^t\|_*^2 - \frac{1}{4\eta}\sum_{t=2}^{T} \|\boldsymbol{x}^t - \boldsymbol{x}^{t-1}\|^2,$$

where $\|\cdot\|$ is any norm with respect to which φ is 1-strongly convex, and $\|\cdot\|_*$ is the dual of the norm $\|\cdot\|$.

Proposition 1 was stated in general for any instantiation of FTRL. In the particular case of OMWU, the negative entropy function ω was proven to be 1-strongly convex with respect to both the Euclidean norm (Problem 1.1) and the ℓ_1 norm (Problem 1.2). So, in principle, either norm can be used in Proposition 1. However, one choice dominates the other.

Problem 1.7 (2 points)**.** The negative entropy function ω is 1-strongly convex with respect to both the Euclidean norm (Problem 1.1) and the ℓ_1 norm (Problem 1.2). So, in principle, either norm can be used when invoking Proposition 1. Which norm do you think leads to a stronger regret bound, and why?

Solution. [*** Your solution here ***] □

Problem 1.8 (3 points)**.** Prove that the range $\Omega = \sup_{\boldsymbol{x},\boldsymbol{x}' \in \operatorname{relint} \Delta^n}\{\omega(\boldsymbol{x}) - \omega(\boldsymbol{x}')\}$ of ω on relint Δ^n is $\Omega = \log n$. Then, use Proposition 1—which was stated in general for any instantiation of FTRL—to argue that OMWU for the simplex Δ^n satisfies the regret bound

$$R^T \le \frac{\log n}{\eta} + \eta \sum_{t=1}^{T} \|\boldsymbol{\ell}^t - \boldsymbol{m}^t\|_\infty^2 - \frac{1}{4\eta}\sum_{t=2}^{T} \|\boldsymbol{x}^t - \boldsymbol{x}^{t-1}\|_1^2.$$

★ Hint: The minimizer $\boldsymbol{x}^*$ of ω over relint Δ^n is $\nabla\omega^*(\mathbf{0})$. You already know how to compute this from Problem 1.5.
★ Hint: The supremum of ω over relint Δ^n is 0 (you should prove this).
★ Hint: You can take for granted the fact that $\|\cdot\|_\infty$ is the dual norm of $\|\cdot\|_1$.

Solution. [*** Your solution here ***] □

1.4 OMWU as predictive OMD (10 points)

It turns out that OMWU—which was defined as the instance of predictive FTRL in which the regularizer is set to the negative entropy function—is equivalent to predictive OMD with negative entropy function, in the sense that the two algorithms produce the same iterates at every time t. Predictive OMD is recalled in Algorithm 2. As a reminder, the Bregman divergence $D_\varphi(\cdot \,\|\, \cdot)$ is defined with respect to any regularizer φ and any two points $\boldsymbol{x}, \boldsymbol{c}$ as

$$D_\varphi(\boldsymbol{x} \,\|\, \boldsymbol{c}) \coloneqq \varphi(\boldsymbol{x}) - \varphi(\boldsymbol{c}) - \nabla\varphi(\boldsymbol{c})^\top(\boldsymbol{x} - \boldsymbol{c}).$$

Problem 1.9 (10 points)**.** Consider the predictive OMD algorithm (Algorithm 2), where $\mathcal{X}$ is set to be the relative interior relint Δ^n of the n-simplex, the regularizer φ is set to be the negative entropy function ω, and $\eta > 0$ is a generic stepsize. Prove that the iterates produced by that algorithm coincide with those produced by OMWU as defined in Section 1.3.

Solution. [*** Your solution here ***] □

2 Counterfactual Regret Minimization (50 points)

In this problem, you will implement the CFR regret minimizer for sequence-form decision problems.

You will run your CFR implementation on three games: rock-paper-superscissors (a simple variant of rock-paper-scissors, where beating paper with scissors gives a payoff of 2 instead of 1) and two well-known poker variants: Kuhn poker [Kuhn, 1950] and Leduc poker [Southey et al., 2005]. A description of each game is given in the zip of this homework, according to the format described in Section 2.1. The zip of the homework also contains a stub Python file to help you set up your implementation.

2.1 Format of the game files

Each game is encoded as a json file with the following structure.

- At the root, we have a dictionary with three keys: `decision_problem_pl1`, `decision_problem_pl2`, and `utility_pl1`. The first two keys contain a description of the tree-form sequential decision problems faced by the two players, while the third is a description of the bilinear utility function for Player 1 as a function of the sequence-form strategies of each player. Since both games are zero-sum, the utility for Player 2 is the opposite of the utility of Player 1.
- The tree of decision points and observation points for each decision problem is stored as a list of nodes. Each node has the following fields

  `id` is a string that represents the identifier of the node. The identifier is unique among the nodes for the same player.

  `type` is a string with value either `decision` (for decision points) or `observation` (for observation points).

  `actions` (only for decision points). This is a set of strings, representing the actions available at the decision node.

  `signals` (only for observation points). This is a set of strings, representing the signals that can be observed at the observation node.

  `parent_edge` identifies the parent edge of the node. If the node is the root of the tree, then it is `null`. Else, it is a pair `(parent_node_id, action_or_signal)`, where the first member is the `id` of the parent node, and `action_or_signal` is the action or signal that connects the node to its parent.

`parent_sequence` (only for decision points). Identifies the parent sequence p_j of the decision point, defined as the last sequence (that is, decision point-action pair) encountered on the path from the root of the decision process to j.

Remark 1. The list of nodes of the tree-form sequential decision process is given in top-down traversal order. The bottom-up traversal order can be obtained by reading the list of nodes backwards.

- The bilinear utility function for Player 1 is given through the payoff matrix $\boldsymbol{A}$ such that the (expected) utility of Player 1 can be written as
$$u_1(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y},$$
where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequence-form strategies for Players 1 and 2 respectively. We represent $\boldsymbol{A}$ in the file as a list of all non-zero matrix entries, storing for each the row index, column index, and value. Specifically, each entry is an object with the fields

  `sequence_pl1` is a pair `(decision_pt_id_pl1, action_pl1)` which represents the sequence of Player 1 (row of the entry in the matrix).

  `sequence_pl2` is a pair `(decision_pt_id_pl2, action_pl2)` which represents the sequence of Player 2 (column of the entry in the matrix).

  `value` is the non-zero float value of the matrix entry.

Example: Rock-paper-superscissors In the case of rock-paper-superscissors the decision problem faced by each of the players has only one decision points with three actions: playing rock, paper, or superscissors. So, each tree-form sequential decision process only has a single node, which is a decision node. The payoff matrix of the game[1] is

$$\begin{array}{c|ccc} & \text{r} & \text{p} & \text{s} \\ \hline \text{r} & 0 & -1 & 1 \\ \text{p} & 1 & 0 & -2 \\ \text{s} & -1 & 2 & 0 \end{array}.$$

So, the game file in this case has content:

```
{
  "decision_problem_pl1": [
    {"id": "d1_pl1", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "decision_problem_pl2": [
    {"id": "d1_pl2", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "utility_pl1": [
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "p"], "value": -1},
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "s"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "r"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "s"], "value": -2},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "r"], "value": -1},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "p"], "value": 2}
  ]
}
```

[1]A Nash equilibrium of the game is reached when all players play rock with probability $1/2$, paper with probability $1/4$ and superscissors with probability $1/4$. Correspondingly, the game value is 0.

2.2 Learning to best respond

Let Q_1 and Q_2 be the sequence-form strategy polytopes corresponding to the tree-form sequential decision problems faced by Players 1 and 2 respectively. A good smoke test when implementing regret minimization algorithms is to verify that they learn to best respond. In particular, you will verify that your implementation of CFR applied to the decision problem of Player 1 learns a best response against Player 2 when Player 2 plays the *uniform* strategy, that is, the strategy that at each decision points picks any of the available actions with equal probability.

Let $\boldsymbol{u} \in Q_2$ be the sequence-form representation of the strategy for Player 2 that at each decision point selects each of the available actions with equal probability. When Player 2 plays according to that strategy, the utility vector for Player 1 is given by $\boldsymbol{\ell} := \boldsymbol{A}\boldsymbol{u}$, where $\boldsymbol{A}$ is the payoff matrix of the game.

For each of the three games, take your CFR implementation for the decision problem of Player 1, and let it output strategies $\boldsymbol{x}^t \in Q_1$ while giving as feedback at each time t the same utility vector $\boldsymbol{\ell}$. As $T \to \infty$, the average strategy

$$\bar{\boldsymbol{x}}^T := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}^t \in Q_1 \tag{6}$$

will converge to a best response to the uniform strategy $\boldsymbol{u}$, that is,

$$\lim_{T\to\infty} (\bar{\boldsymbol{x}}^T)^\top \boldsymbol{A}\boldsymbol{u} = \max_{\hat{\boldsymbol{x}}\in Q_1} \hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{u}.$$

If the above doesn't happen empirically, something is wrong with your implementation.

Problem 2.1 (20 points). In each of the three games, apply your CFR implementation to the tree-form sequential decision problem of Player 1, using as local regret minimizer at each decision point the regret matching algorithm (Lecture 4). At each time t, give as feedback to the algorithm the same utility vector $\boldsymbol{\ell} = \boldsymbol{A}\boldsymbol{u}$, where $\boldsymbol{u} \in Q_2$ is the uniform strategy for Player 2. Run the algorithm for 1000 iterations. After each iteration $T = 1, \ldots, 1000$, compute the value of $v^T := (\bar{\boldsymbol{x}}^T)^\top \boldsymbol{A}\boldsymbol{u}$ where $\bar{\boldsymbol{x}}^T \in Q_1$ is the average strategy output so far by CFR, as defined in (6).

Plot v^T as a function of T. Empirically, what is the limit you observe v^T is converging to?

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: in rock-paper-superscissor, v^T should approach the value $1/3$. In Kuhn poker, the value $1/2$. In Leduc poker, the value 2.0875.

Solution. [*** Your solution here. Your solution should include three plots (one for each game), and three values. Don't forget to turn in your implementation. ***] □

2.3 Learning a Nash equilibrium

Now that you are confident that your implementation of CFR is correct, you will use CFR to converge to Nash equilibrium using the self-play idea described in Lecture 3 and recalled next.

The idea behind using regret minimization to converge to Nash equilibrium in a two-player zero-sum game is to use *self play*. We instantiate two regret minimization algorithms, $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$, for the domains of the maximization and minimization problem, respectively. At each time t the two regret minimizers output strategies $\boldsymbol{x}^t$ and $\boldsymbol{y}^t$, respectively. Then, they receive as feedback the vectors $\boldsymbol{\ell}^t_\mathcal{X}, \boldsymbol{\ell}^t_\mathcal{Y}$ defined as

$$\boldsymbol{\ell}^t_\mathcal{X} := \boldsymbol{A}\boldsymbol{y}^t, \qquad \boldsymbol{\ell}^t_\mathcal{Y} := -\boldsymbol{A}^\top\boldsymbol{x}^t, \tag{7}$$

where $\boldsymbol{A}$ is Player 1's payoff matrix.

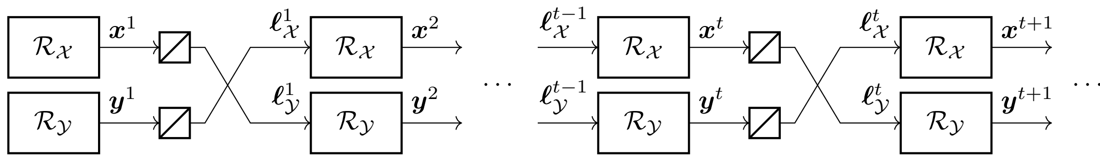
$\mathcal{R}_\mathcal{X}$
$\boldsymbol{x}^1$
$\boldsymbol{\ell}^1_\mathcal{X}$
$\mathcal{R}_\mathcal{X}$
$\boldsymbol{x}^2$
$\mathcal{R}_\mathcal{Y}$
$\boldsymbol{y}^1$
$\boldsymbol{\ell}^1_\mathcal{Y}$
$\mathcal{R}_\mathcal{Y}$
$\boldsymbol{y}^2$
$\cdots$
$\boldsymbol{\ell}^{t-1}_\mathcal{X}$
$\mathcal{R}_\mathcal{X}$
$\boldsymbol{x}^t$
$\boldsymbol{\ell}^t_\mathcal{X}$
$\mathcal{R}_\mathcal{X}$
$\boldsymbol{x}^{t+1}$
$\boldsymbol{\ell}^{t-1}_\mathcal{Y}$
$\mathcal{R}_\mathcal{Y}$
$\boldsymbol{y}^t$
$\boldsymbol{\ell}^t_\mathcal{Y}$
$\mathcal{R}_\mathcal{Y}$
$\boldsymbol{y}^{t+1}$
$\cdots$

Figure 1: The flow of strategies and utilities in regret minimization for games. The symbol ⧄ denotes computation/construction of the utility vector.

We summarize the process pictorially in Figure 1.

A well known folk theorem establish that the pair of average strategies produced by the regret minimizers up to any time T converges to a Nash equilibrium, where convergence is measured via the *saddle point gap*

$$0 \le \gamma(\boldsymbol{x}, \boldsymbol{y}) := \left(\max_{\hat{\boldsymbol{x}} \in \mathcal{X}} \{\hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{y}\} - \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y}\right) + \left(\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} - \min_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \{\boldsymbol{x}^\top \boldsymbol{A}\hat{\boldsymbol{y}}\}\right) = \max_{\hat{\boldsymbol{x}} \in \mathcal{X}} \{\hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{y}\} - \min_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \{\boldsymbol{x}^\top \boldsymbol{A}\hat{\boldsymbol{y}}\}.$$

A point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ has zero saddle point gap if and only if it is a Nash equilibrium of the game.

Theorem 1. Consider the self-play setup summarized in Figure 1, where $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$ are regret minimizers for the sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Let $R^T_\mathcal{X}$ and $R^T_\mathcal{Y}$ be the (sublinear) regret accumulated by $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$, respectively, up to time T, and let $\bar{\boldsymbol{x}}^T$ and $\bar{\boldsymbol{y}}^T$ denote the average of the strategies produced up to time T, that is,

$$\bar{\boldsymbol{x}}^T := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}^t, \qquad \bar{\boldsymbol{y}}^T := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^t. \tag{8}$$

Then, the saddle point gap $\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ of $(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ satisfies

$$\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T) \le \frac{R^T_\mathcal{X} + R^T_\mathcal{Y}}{T} \to 0 \qquad \text{as } T \to \infty.$$

Problem 2.2 (20 points)**.** Let the CFR implementation (using regret matching as the local regret minimizer at each decision point) for Player 1's and Player 2's tree-form sequential decision problems play against each other in self play, as described above.

Plot the saddle point gap and the expected utility (for Player 1) of the average strategies $\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ as a function of the number of iterations $T = 1, \ldots, 1000$.

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: to compute the saddle-point gap, feel free to use the function `gap(game, strategy_pl1, strategy_pl2)` provided in the Python stub file.

★ Hint: the saddle point gap shuold be going to zero. The expected utility of the average strategies in rock-paper-superscissor should approach the value 0. In Kuhn poker it should approach -0.055. In Leduc poker it should approach -0.085.

Solution. [*** Your solution here. Your solution should include six plots (two for each game—one for the saddle point gap and one for the utility). Don't forget to turn in your implementation. ***] □

2.4 CFR+

To achieve better performance in practice when learning Nash equilibria in two-player zero-sum games, people often make the following modifications to the setup of the previous subsection.

- Instead of regret matching, CFR is set up to use the regret matching plus algorithm (see Lecture 3) at each decision point.
- Instead of using the classical self-play scheme described in Figure 1, people *alternate* the iterates and feedback as described in Figure 2, where the utility vector $\boldsymbol{\ell}^t_{\mathcal{X}}$ is as defined in (7), whereas

$$\tilde{\boldsymbol{\ell}}^t_{\mathcal{Y}} := -\boldsymbol{A}^\top \boldsymbol{x}^{t+1}.$$

(Note that at the very beginning, $\boldsymbol{x}^1$ does not participate in the computation of any utility vector).

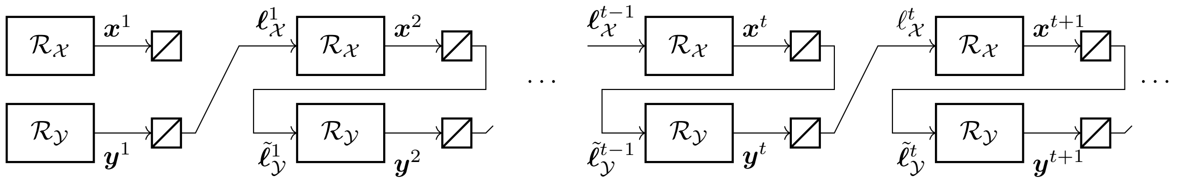
$\mathcal{R}_\mathcal{X}$
$\boldsymbol{x}^1$
$\boldsymbol{\ell}^1_\mathcal{X}$
$\boldsymbol{x}^2$
$\mathcal{R}_\mathcal{Y}$
$\boldsymbol{y}^1$
$\tilde{\boldsymbol{\ell}}^1_\mathcal{Y}$
$\boldsymbol{y}^2$
$\boldsymbol{\ell}^{t-1}_\mathcal{X}$
$\boldsymbol{x}^t$
$\boldsymbol{\ell}^t_\mathcal{X}$
$\boldsymbol{x}^{t+1}$
$\tilde{\boldsymbol{\ell}}^{t-1}_\mathcal{Y}$
$\boldsymbol{y}^t$
$\tilde{\boldsymbol{\ell}}^t_\mathcal{Y}$
$\boldsymbol{y}^{t+1}$
. . .

Figure 2: The alternation method for CFR in games. The symbol ⧄ denotes computation/construction of the utility vector.

- Finally, the *linear average* of the strategies, defined as the weighted averages

$$\bar{\bar{\boldsymbol{x}}}^T := \frac{2}{T(T+1)} \sum_{t=1}^{T} t \cdot \boldsymbol{x}^t, \qquad \bar{\bar{\boldsymbol{y}}}^T := \frac{2}{T(T+1)} \sum_{t=1}^{T} t \cdot \boldsymbol{y}^t$$

is considered instead of the regular averages (8) when computing the saddle point gap.

Collectively, the modified setup we just described is referred to as "running CFR+".

Problem 2.3 (10 points). Modify your implementation of CFR to match the CFR+ self-play setup described above. Run CFR+ for 1000 iterations, plotting the expected utility for Player 1 and the saddle point gap of the linear averages $\gamma(\bar{\bar{\boldsymbol{x}}}^T, \bar{\bar{\boldsymbol{y}}}^T)$ after each iteration T.

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: to compute the saddle-point gap, feel free to use the function `gap(game, strategy_pl1, strategy_pl2)` provided in the Python stub file.

Solution. [*** Your solution here. Your solution should include six plots (two for each game—one for the saddle point gap and one for the utility). Don't forget to turn in your implementation. ***] □

References

H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: opponent modelling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.