15-888 Computational Game Solving (Fall 2025) Wednesday, October 29th, 2025

Lecture 15
Deep learning in Game Solving

Ioannis Anagnostides

In very large games, even performing a handful iterations of CFR can be prohibitive on account of the fact that it requires fully traversing of the game tree. The first part of today's lecture introduces *Monte Carlo counterfactual regret minimization (*MCCFR*)*, a stochastic approximation of regular CFR introduced by Lanctot et al. [2009]. MCCFR is significantly cheaper than CFR in terms of running time per iteration. Many of the state of the art game solving algorithms rely on MCCFR or variants thereof. Indeed, as we saw in an earlier lecture, *Libratus* used MCCFR to compute the blueprint strategy. This lecture dives deeper into the theoretical analysis of MCCFR.

Building on MCCFR, the second part of the lecture centers on more modern game solving algorithms, namely DeepCFR [Brown et al., 2019] and ESCHER [McAleer et al., 2023]. DeepCFR uses deep neural networks *in lieu* of manual game abstraction to deal with large game trees that cannot be handled with the usual tabular methods. As we shall see, the name of the game in stochastic regret minimization is to find a cheap gradient estimator with small variance, so that it tightly approximates the underlying true gradient. ESCHER is a recent innovation that drastically reduces the variance of the estimator, and is thereby often the most effective approach when dealing with large games.

1 Stochastic regret minimization and MCCFR

The main motivation for stochastic regret minimization in general and MCCFR in particular is to reduce the per-iteration complexity. It will be remembered that CFR proceeds by employing an individual, separate regret minimization over each information set. Upon observing a utility $\boldsymbol{u}^{(t)}$, it constructs the *counterfactual utilities* at each information set; the exact form of those counterfactual utilities as a function of $\boldsymbol{u}^{(t)}$ were derived via regret circuits in one of the homework problems. The basic idea of MCCFR is to replace $\boldsymbol{u}^{(t)}$ by a *sparse* estimator $\tilde{\boldsymbol{u}}^{(t)}$, so that one does not need to perform a full game traversal. In particular, a judicious choice of $\tilde{\boldsymbol{u}}^{(t)}$ will confer two main benefits: i) there is no need to exactly compute $\boldsymbol{u}^{(t)}$, which can be expensive; and ii) only a subset of information sets need to be updated in each iteration. So long as $\tilde{\boldsymbol{u}}^{(t)}$ is *unbiased*, in that $\mathbb{E}[\tilde{\boldsymbol{u}}^{(t)}] = \boldsymbol{u}^{(t)}$, one can bound the degradation of the stochastic approximation as a function of the variance of the estimator (Theorem 1.1).

Setup Our focus in this lecture is again on (two-player) zero-sum games in extensive form. Computing an equilibrium in this setting can be phrased as a bilinear saddle-point problem,

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \sum_{z\in\mathcal{Z}} u_2(z)\boldsymbol{x}[\sigma_1(z)]\boldsymbol{y}[\sigma_2(z)]\boldsymbol{c}[\sigma_c(z)], \tag{1}$$

where $\sigma_i(z) \in \Sigma_i$ is the last sequence of player $i \in \{1, 2, c\}$ from the root of the game tree to the terminal node $z \in \mathcal{Z}$ and $\mathcal{X}$ and $\mathcal{Y}$ are the sequence-form polytopes of the two players. The min-max problem in (1) can be cast as $\boldsymbol{x}^\top \mathbf{A} \boldsymbol{y}$ for some matrix $\mathbf{A}$.

In this setting, the utility vector of Player 1 is $\boldsymbol{u}_\mathcal{X} = -\mathbf{A}\boldsymbol{y}$ (the negative sign because Player 1 is the minimizer) and that of Player 2 is $\boldsymbol{u}_\mathcal{Y} = \mathbf{A}^\top \boldsymbol{x}$. Computing these matrix-vector products is relatively expensive, so we will resort to sampling in order to come up with unbiased estimators. In doing so, there is a basic tradeoff one should have in mind: it is desirable to have cheap and sparse estimators but at the same time the variance of the estimator needs to be controlled to limit the regret degradation. This lecture introduces two standard ways of navigating this tradeoff—*external sampling* and *outcome sampling*.

1.1 External sampling

We first introduce external sampling. Let's take the perspective of Player 1; analogous reasoning applies for Player 2. An unbiased estimator of $\boldsymbol{u}_\mathcal{X}^{(t)} = -\mathbf{A}\boldsymbol{y}^{(t)}$ can be obtained by sampling independently pure strategies $\tilde{\boldsymbol{y}}^{(t)}$ for Player 2 and the chance player $\tilde{\boldsymbol{c}}$. It follows from (1) that if $\mathbb{E}[\tilde{\boldsymbol{y}}^{(t)}] = \boldsymbol{y}^{(t)}$ and $\mathbb{E}[\tilde{\boldsymbol{c}}] = \boldsymbol{c}$, where $\boldsymbol{c}$ denotes the fixed strategy of the chance player, then the resulting utility estimator

$$\boldsymbol{x} \mapsto u_1(\boldsymbol{x}, \tilde{\boldsymbol{y}}^{(t)}, \tilde{\boldsymbol{c}}),$$

in which $\boldsymbol{y}^{(t)}$ is replaced by $\tilde{\boldsymbol{y}}^{(t)}$ and $\boldsymbol{c}$ by $\tilde{\boldsymbol{c}}$, is unbiased; it is called "external sampling" because it only samples the opponent and the chance player.

Normal-form games It's worth dwelling first on the power of external sampling in the special case of normal-form games. Let's assume that each player employs a regret minimizer with linear (in the dimension) per-iteration running time; most of the online algorithms we have covered, such as RM and MWU, have this property. Then the per-iteration complexity is dominated by the computation of the matrix-vector products $\mathbf{A}\boldsymbol{y}^{(t)}$ and $\mathbf{A}^\top \boldsymbol{x}^{(t)}$. If $\mathbf{A} \in \mathbb{R}^{m_1 \times m_2}$ has $\text{nnz}(\mathbf{A})$ nonzero entries, then each such computation requires $O(\text{nnz}(\mathbf{A}))$ arithmetic operations. If $\mathbf{A}$ is dense, this running time is quadratic, which makes the computation prohibitive in large instances. External sampling addresses this issue: by replacing $\boldsymbol{y}^{(t)}$ with a pure strategy $\tilde{\boldsymbol{y}}^{(t)}$ sampled from $\boldsymbol{y}^{(t)}$, $\mathbf{A}\tilde{\boldsymbol{y}}^{(t)}$ can now be clearly computed in linear time. In other words, using external sampling, we have managed to reduce the per-iteration running time from $O(\text{nnz}(\mathbf{A}))$ to $O(m_1 + m_2)$; the benefit can be even larger in extensive-form games. A consequence of Theorem 1.1, which will be proved shortly, is that stochastic regret minimization in self-play yields an ϵ-equilibrium in $\tilde{O}(^{(m_1+m_2)}/_{\epsilon^2})$ time, where $\tilde{O}(\cdot)$ hides logarithmic factors. This is essentially the famous *sublinear* algorithm of Grigoriadis and Khachiyan [1995]. It is "sublinear" because, for any $\epsilon > 0$, one can find an ϵ-equilibrium in less time than what is required to read the input—which is $\text{nnz}(\mathbf{A})$! We refer to Clarkson et al. [2012] for a further development of the framework of Grigoriadis and Khachiyan [1995] in other machine learning problems.

In defense of deterministic methods, using optimism *without* sampling leads to an overall running time of $\tilde{O}(^{\text{nnz}(\mathbf{A})}/_{\epsilon})$, which is superior when the precision ϵ is small enough. To push this even further, using linear programming the overall running time would scale polynomially in

$\log(1/\epsilon)$, but that comes at the cost of introducing higher dimension dependencies that prevent its use in large-scale problems. The main takeaway is that, as expected, the best algorithm to use depends on the regime of approximation. The focus of this lecture is on very large games, where one is content with a relatively crude approximation ϵ, so stochastic regret minimization is to be preferred.

Implementing external sampling in extensive-form games Returning to our more general setup in extensive-form games, an external sampling estimator can be constructed as follows. We start at the root and we gradually make our way toward the terminal nodes. Every time we touch a node that belongs to either the opponent or the chance player, we sample according to $\boldsymbol{y}^{(t)}$ (if the node belongs to the opponent) or $\boldsymbol{c}$ (if the node belongs to the chance player). If the node belongs to Player 1, the algorithm branches over all actions and proceeds recursively. The main benefit of this estimator is that every time we encounter a node of Player 2 or the chance player, the algorithm branches only on a single action. As a result, computing this estimator is significantly cheaper than computing the exact utility vector $\boldsymbol{u}_{\mathcal{X}}^{(t)}$. Furthermore, we only need to update information sets that the opponent/chance plays to reach them in *the sampled strategy*.

The algorithm that results from running CFR with this particular choice of utility estimator is known as *external sampling* MCCFR.

1.2 Outcome sampling

Outcome sampling takes a step further in reducing the cost of each iteration. Unlike external sampling, it samples from every player's strategy. For Player 1, it takes as input a fully mixed *reference strategy* $\boldsymbol{r}^{(t)} \in \mathcal{X}$, and samples a terminal node $\tilde{z}^{(t)} \in \mathcal{Z}$ with probability distribution

$$\mathbb{P}[\tilde{z}^{(t)} = z] = \boldsymbol{r}^{(t)}[\sigma_1(z)]\boldsymbol{y}^{(t)}[\sigma_2(z)]\boldsymbol{c}[\sigma_c(z)] \quad \forall z \in \mathcal{Z}. \tag{2}$$

It then produces the estimator

$$\tilde{\boldsymbol{u}}_{\mathcal{X}}^{(t)} = \frac{u_1(\tilde{z}^{(t)})}{\boldsymbol{r}^{(t)}[\sigma_1(\tilde{z}^{(t)})]}\boldsymbol{e}_{\tilde{z}^{(t)}}, \tag{3}$$

where $\boldsymbol{e}_z \in \mathbb{R}^{\Sigma_1}$ is the unit vector such that $\boldsymbol{e}_z[\sigma_1(z)] = 1$. It is a simple exercise to prove that the utility estimator in (3) in unbiased, in that $\mathbb{E}[\tilde{\boldsymbol{u}}_{\mathcal{X}}^{(t)}] = \boldsymbol{u}_{\mathcal{X}}^{(t)}$. Sampling a terminal node per the distribution (2) is computationally very cheap: one can start from the root of the game tree and proceed by sampling in each node according to the strategy of the player acting at that node. This can be done in time proportional to the *depth* d of the game tree, which is typically exponentially smaller than the number of nodes in the game tree. Having sampled a terminal node $\tilde{z}^{(t)}$, executing a CFR step takes time $O(d|\mathcal{A}|)$ since one only needs to update information sets encountered from the path starting from the root and ending at $\tilde{z}^{(t)}$. On the flip side, the main caveat of the outcome sampling estimator defined in (3) is that its variance is high, being proportional to the inverse of $\min_{z\in\mathcal{Z}} \boldsymbol{r}^{(t)}[\sigma_1(z)]$. At this point, we haven't really specified how to come up with the reference strategy $\boldsymbol{r}^{(t)}$. It theory, a sensible approach is to maximize $\min_{z\in\mathcal{Z}} \boldsymbol{r}^{(t)}[\sigma_1(z)]$, so as to minimize the variance of the estimator. It can be shown that there exists a *balanced strategy* $\boldsymbol{r}^*$ such that

$\boldsymbol{r}^*[\sigma] \geq 1/|\Sigma_1|$ for all $\sigma \in \Sigma_1$ [Farina et al., 2020]; but this is not necessarily what works best in practice. The outcome sampling estimator makes use of what's known as *importance sampling*; we will return to this in Section 3.

The algorithm that results from running CFR with the utility estimator given in (3) is known as *outcome sampling* MCCFR.

1.3 Regret analysis

Having introduced two standard ways of estimating the utility vector in extensive-form games, we now analyze stochastic regret minimization algorithms. The following analysis holds beyond extensive-form games and external/outcome sampling. The upshot is that as long as the estimator of $\boldsymbol{u}^{(t)}$ is unbiased, regret can only incur a small degradation.

In more detail, the goal is to design a stochastic regret minimizer $\mathfrak{R}$ that incurs small regret upon receiving the sequence of utilities $(\boldsymbol{u}^{(t)})_{t=1}^T$. $\mathfrak{R}$ will rely on a deterministic regret minimizer $\tilde{\mathfrak{R}}$, such as CFR, which operates by receiving as input the sequence of utilities $(\tilde{\boldsymbol{u}}^{(t)})_{t=1}^T$, each of which is produced by estimating $\boldsymbol{u}^{(t)}$. The output of $\mathfrak{R}$ will simply be the output of $\tilde{\mathfrak{R}}$. We denote by $\mathrm{Reg}^{(T)}(\boldsymbol{x}) = \sum_{t=1}^T \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle$ the regret of $\mathfrak{R}$ with respect to the comparator $\boldsymbol{x} \in \mathcal{X}$, so that $\mathrm{Reg}^{(T)} = \max_{\boldsymbol{x} \in \mathcal{X}} \mathrm{Reg}^{(T)}(\boldsymbol{x})$. Similarly, we define $\widetilde{\mathrm{Reg}}^{(T)}(\boldsymbol{x}) = \sum_{t=1}^T \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle$. The main assumption in the analysis is that the estimator is unbiased, in that

$$\mathbb{E}[\tilde{\boldsymbol{u}}^{(t)} \mid \tilde{\boldsymbol{u}}^{(1)}, \dots, \tilde{\boldsymbol{u}}^{(t-1)}] = \boldsymbol{u}^{(t)}.$$

The following result was established by Farina et al. [2020]; a weaker result was shown by Lanctot et al. [2009]. We stress again that it applies to any stochastic approximation, not just MCCFR with external or outcome sampling.

Theorem 1.1 (Farina et al., 2020)**.** *Suppose that there are constants M and $\tilde{M}$ such that*

$$|\langle \boldsymbol{x} - \boldsymbol{x}', \boldsymbol{u}^{(t)} \rangle| \leq M \text{ and } |\langle \boldsymbol{x} - \boldsymbol{x}', \tilde{\boldsymbol{u}}^{(t)} \rangle| \leq \tilde{M} \quad \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X} \text{ and } t \in [T].$$

Then, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$,

$$\mathrm{Reg}^{(T)} \leq \widetilde{\mathrm{Reg}}^{(T)} + (M + \tilde{M}) \sqrt{2T \log \frac{1}{\delta}}.$$

$\tilde{\mathfrak{R}}$ is a deterministic regret minimizer, so $\widetilde{\mathrm{Reg}}^{(T)}$ can be bounded using the techniques we covered in earlier parts of the course, with the catch that the range of the observed utilities can be as large as $\tilde{M}$. The main takeaway of Theorem 1.1 is that, ignoring lower-order terms, the regret of $\mathfrak{R}$ can only be larger than the regret of $\tilde{\mathfrak{R}}$ by a factor that depends on $\sqrt{T}$ and the range of the estimator $\tilde{M}$. The first consequence of this is that, unlike their deterministic counterparts, stochastic regret minimization algorithms do not beat the $\sqrt{T}$ barrier; this is not merely a limitation of the analysis, but, as we shall explain shortly, happens for a fundamental reason. The second consequence is that it is desirable to minimize $\tilde{M}$; this is where the tradeoff between external versus outcome sampling becomes evident: even if one uses the balanced strategy in outcome sampling, $\tilde{M}$ is a factor $|\Sigma|$ higher, for a total regret bound of $O(|\Sigma|^2\sqrt{T})$ when using CFR to instantiate $\tilde{\mathfrak{R}}$.

Proof of Theorem 1.1. We fix any comparator $\boldsymbol{x} \in \mathcal{X}$. We have

$$\mathsf{Reg}^{(T)}(\boldsymbol{x}) = \widetilde{\mathsf{Reg}}^{(T)}(\boldsymbol{x}) + \sum_{t=1}^{T} d^{(t)},$$

where $d^{(t)} \coloneqq \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle - \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle$ is a martingale difference sequence. For any time $t \in [n]$, we can bound the magnitude of $d^{(t)}$ as

$$|d^{(t)}| = |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle - \langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle| \le |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)} \rangle| + |\langle \boldsymbol{x} - \boldsymbol{x}^{(t)}, \tilde{\boldsymbol{u}}^{(t)} \rangle| \le M + \tilde{M}.$$

Using the Azuma-Hoeffding concentration inequality [Hoeffding, 1963, Azuma, 1967],

$$\mathbb{P}[\mathsf{Reg}^{(T)}(\boldsymbol{x}) \le \widetilde{\mathsf{Reg}}^{(T)}(\boldsymbol{x}) + \epsilon] = 1 - \mathbb{P}\left[\sum_{t=1}^{T} d^{(t)} > \epsilon\right] \ge 1 - \exp\left(-\frac{2\epsilon^2}{4T(M + \tilde{M})^2}\right).$$

Setting $\epsilon = (M + \tilde{M})\sqrt{2T \log(1/\delta)}$ yields the statement. □

There is another interesting consequence of stochastic regret minimization concerning *sparse* equilibria. Let's restrict our attention to normal-form zero-sum games. A strategy $\boldsymbol{x} \in \Delta(\mathcal{A})$ is said to be k-sparse if at most k coordinates of $\boldsymbol{x}$ have a positive value. A well-known result in algorithmic game theory is that, for any $\epsilon > 0$, $\log(m)/\epsilon^2$-sparse equilibrium strategies exist. What's more, running a version of self-play MWU in which in every round we sample each player's strategy provides a constructive proof of that fact. It turns out that $\log(m)/\epsilon^2$ is the best sparsity one can hope for [Feder et al., 2007], so this shows that *any* stochastic algorithm cannot beat the regret barrier of $\Omega(\sqrt{T \log m})$ even in self-play. (This observation is due to Haipeng Luo.)

2 Automated game abstraction and Deep CFR

Stochastic regret minimization is a key component in modern game solving algorithms starting from MCCFR. Another central innovation is the use of *function approximation* via deep neural networks in place of manual game abstraction. This is the main subject of this section. As we have seen, abstraction often requires domain-specific knowledge and may miss important strategic nuances of the game. There is also a chicken-and-egg problem: determining a good abstraction requires knowing the equilibrium, but computing an equilibrium rests on a compact game abstraction.

Neural fictitious self-play Let's begin with the *neural fictitious self-play* (NFSP) of Heinrich and Silver [2016], a precursor of DeepCFR. NFSP is based on the usual fictitious play algorithm, which was introduced in an earlier lecture; we recall that the basic idea is that each player best responds to the *average strategy* of the opponent. What makes fictitious play amenable to deep learning techniques is that, unlike more sophisticated algorithms such as FTRL or MD that require solving a convex optimization problem, its execution only hinges on a best response oracle. It

turns out that there are techniques for implementing a best response oracle even in very large games, which gave the impetus to Heinrich and Silver [2016] to introduce NFSP. From a theoretical standpoint, one should have in mind that even the tabular version of fictitious play can suffer from exponentially slow convergence [Daskalakis and Pan, 2014], although the hope here is that such counterexamples do not arise in typical instances encountered in practice.

NFSP maintains two separate datasets, $\mathcal{D}_{\text{RL}}^{(t)}$ and $\mathcal{D}_{\text{SL}}^{(t)}$, which are gradually grown during training. The high-level idea is as follows.

- It uses $\mathcal{D}_{\text{RL}}^{(t)}$ to train a *value* neural network $Q(\theta_Q^{(t)})$ to predict action values at each state using *Q-learning*, a standard reinforcement learning algorithm. $Q(\theta_Q^{(t)})$ is then used to determine the player's approximate best response strategy by way of maximizing the value at each state, mixed with a small amount of exploration.

- NFSP also maintains a separate *policy* neural network, $\Pi(\theta_\Pi^{(t)})$, designed to imitate the player's past behavior using supervised learning; this is where the dataset $\mathcal{D}_{\text{SL}}^{(t)}$ is used. This policy network intends to capture the average strategy of the player; we recall that it is only the average of (tabular) fictitious play that converges, which justifies the use of the above policy network.[1]

The datasets $\mathcal{D}_{\text{RL}}^{(t)}$ and $\mathcal{D}_{\text{SL}}^{(t)}$ are updated dynamically throughout the execution of the algorithm. $Q(\theta_Q^{(t)})$ and $\Pi(\theta_\Pi^{(t)})$ are continuously updated using stochastic gradient descent. The player chooses actions by mixing between the approximate best response strategy determined through deep reinforcement learning and the average strategy given by the policy network $\Pi(\theta_\Pi^{(t)})$. NFSP is a common benchmark when evaluating deep learning-based approaches in large games.

Deep CFR Similarly to NFSP, the main motivation of DeepCFR is to do away with manual game abstraction. The basic goal of DeepCFR is to approximate CFR without having to compute regrets at each information set. The number of information sets can be very large, so DeepCFR relies on generalization across similar information sets. Neural networks have proven to be powerful function approximators, which justifies their use to estimate regret.

DeepCFR maintains, for each player, a regret neural network $R(\theta_R^{(t)})$ that intends to capture the counterfactual regrets at each information set; if this regret network was exact, DeepCFR would revert to usual MCCFR. In general, the hope is to take advantage of generalization even when the regret network is imprecise. $R(\theta_R^{(t)})$ is trained using a dynamic dataset $\mathcal{D}^{(t)}$; each sample in $\mathcal{D}^{(t)}$ maps an infoset-action pair to the counterfactual regret. Furthermore, as in NFSP, DeepCFR also maintains a policy neural network that intends to approximate the average strategy; again, this step is needed in view of the fact that even tabular CFR only converges in averages.

In each iteration, DeepCFR conducts multiple partial traversals of the game tree in accordance with external sampling MCCFR. (A popular variant of DeepCFR that relies instead on outcome

[1] Incidentally, the additional policy network would not be needed if one had a *last-iterate* guarantee; this provides further justification for focusing on last-iterate convergence, which was discussed in a previous lecture.

sampling is DREAM [Steinberger et al., 2020]; to mitigate the variance of importance sampling, DREAM further relies on a certain variance reduction technique that's not in our scope here.) In each node it encounters that belongs to the opponent, it first obtains an approximation to the counterfactual regrets through the regret network of the opponent, whereupon it determines the opponent's behavioral strategy using RM or any other variant of RM. It then samples from that strategy. Following this process, it computes the sampled counterfactual values $\tilde{\boldsymbol{u}}(a \mid j)$ for each sampled information set j. These counterfactual values are used to compute the instantaneous counterfactual regrets

$$\tilde{\boldsymbol{g}}^{(t)}(a \mid j) = \tilde{\boldsymbol{u}}^{(t)}(a \mid j) - \sum_{a' \in \mathcal{A}_j} \boldsymbol{b}(a' \mid j)\tilde{\boldsymbol{u}}^{(t)}(a' \mid j),$$

where $\boldsymbol{b}(\cdot \mid j)$ is the behavioral strategy of the player at a sampled information set j determined using the regret network of that player. For every such infoset-action pair (j, a), DeepCFR updates the dataset $\mathcal{D}^{(t)}$ by adding the sample $\{(j, a) \mapsto \tilde{\boldsymbol{g}}^{(t)}(a \mid j)\}$. Finally, the regret network is trained through the updated dataset $\mathcal{D}^{(t+1)}$. In other words, the game traversals are used to collect data in order to train the regret network.

A reassuring property of DeepCFR is that it is theoretically sound so long as the function approximation error is small [Brown et al., 2019]. This can be accomplished in principle by making the capacity of the neural network high enough, although practical constraints impose severe limits on the size that can be used. Brown et al. [2019] showed experimentally that DeepCFR is competitive or even outperforms manual abstraction approaches, even though it is a general-purpose algorithm. DeepCFR also outperforms NFSP.

3 ESCHER

The final algorithm that we will cover in today's lecture is ESCHER, introduced by McAleer et al. [2023]. ESCHER lies in the family of outcome sampling MCCFR, but with an improved utility estimator. The main motivation of ESCHER is to obviate the need for importance sampling. As we saw in the first part of the lecture, importance sampling leads to an estimator with a very high variance. This is especially problematic when those estimated utilities are to be produced by deep neural networks, which tend to struggle with outputs of different magnitudes.

ESCHER effectively remedies this issue as follows. It uses a *fixed* reference strategy $\boldsymbol{r}_i$ for each player i; this could be the balanced strategy introduced in Section 1.2, although it doesn't have to be so. As in outcome sampling MCCFR, for each player i, ESCHER proceeds by sampling a node $\tilde{z} \sim (\boldsymbol{r}_i, \boldsymbol{x}_{-i}^{(t)})$, and then updates the regret minimizer at every information set in path from the root to node z. The key difference in ESCHER is that a neural network is used to produce an estimate of the *conditional* values, not the counterfactual values. If $u_i^{(t)}(a \mid j)$ is the counterfactual value at time t, it holds that

$$u_i^{(t)}(a \mid j) = \boldsymbol{x}_{-i}^{(t)}(j) \cdot Q_i^{(t)}(a \mid j),$$

where $\boldsymbol{x}_{-i}^{(t)}(j)$ is the probability that all other players will reach the information set j. Thus,

$$\mathbb{E}[\tilde{u}_i^{(t)}(a \mid j)] = \boldsymbol{r}_i(j) \cdot \boldsymbol{x}_{-i}^{(t)}(j) \cdot Q_i^{(t)}(a \mid j) = \boldsymbol{r}_i(j) \cdot u_i^{(t)}(a \mid j).$$

Now, this is not necessarily an unbiased estimator, but because RM and its variants are scale invariant, the scaling factor—which crucially remains the same throughout, by assumption—doesn't affect the iterates produced. In other words, ESCHER is a theoretically sound algorithm in the tabular setting. It also tends to perform very well *vis-à-vis* other popular algorithms such as DREAM, as demonstrated by McAleer et al. [2023].

References

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael H. Bowling. Monte carlo sampling for regret minimization in extensive games. In *Neural Information Processing Systems*, 2009.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International Conference on Machine Learning (ICML)*, 2019.

Stephen Marcus McAleer, Gabriele Farina, Marc Lanctot, and Tuomas Sandholm. ESCHER: eschewing importance sampling in games by computing a history value function to estimate regret. In *International Conference on Learning Representations (ICLR)*, 2023.

Michael D Grigoriadis and Leonid G Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

Kenneth L Clarkson, Elad Hazan, and David P Woodruff. Sublinear optimization for machine learning. *Journal of the ACM (JACM)*, 59(5):1–49, 2012.

Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Stochastic regret minimization in extensive-form games. In *International Conference on Machine Learning (ICML)*, 2020.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.

Tomás Feder, Hamid Nazerzadeh, and Amin Saberi. Approximating nash equilibria using small-support strategies. In *Electronic Commerce (EC)*, 2007.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv:1603.01121*, 2016.

Constantinos Daskalakis and Qinxuan Pan. A counter-example to karlin's strong conjecture for fictitious play. In *Foundations of Computer Science (FOCS)*, 2014.

Eric Steinberger, Adam Lerer, and Noam Brown. Dream: Deep regret minimization with advantage baselines and model-free learning. *arXiv:2006.10410*, 2020.