DOUBLE NEURAL COUNTERFACTUAL REGRET MINIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Counterfactual regret minimization (CRF) is a fundamental and effective technique for solving imperfect information games. However, the original CRF algorithm only works for discrete state and action spaces, and the resulting strategy is maintained as a tabular representation. Such tabular representation limits the method from being directly applied to large games and continuing to improve from a poor strategy profile. In this paper, we propose a double neural representation for the Imperfect Information Games, where one neural network represents the cumulative regret, and the other represents the average strategy. Furthermore, we adopt the counterfactual regret minimization algorithm to optimize this double neural representation. To make neural learning efficient, we also developed several novel techniques including a robust sampling method, mini-batch Monte Carlo counterfactual regret minimization (MCCFR) and Monte Carlo counterfactual regret minimization plus (MCCFR+) which may be of independent interests. Experimentally, we demonstrate that the proposed double neural algorithm converges significantly better than the reinforcement learning counterpart.

1 INTRODUCTION

In Imperfect Information Games (**IIG**), a player only has partial access to the knowledge of her opponents before making a decision. This is similar to the real world scenarios, such as trading, traffic routing, and public auction. Thus designing methods for solving IIG is of great economic and social benefits. Due to the hidden information, a player has to reason under the uncertainty about her opponents' information, and she also needs to act so as to take advantage of her opponents' uncertainty about her own information.

Nash equilibrium is a typical solution concept for a two-player extensive-form game. Many algorithms have been designed over years to approximately find Nash equilibrium for large games. One of the most effective approaches is CFR (Zinkevich et al., 2008). In this algorithm, the authors proposed to minimize overall counterfactual regret and prove that the average of the strategies in all iterations would converge to a Nash equilibrium. However, the original CFR only works for discrete state and action spaces, and the resulting strategy is maintained as a tabular representation. Such tabular representation limits the method from being directly applied to large games and continuing to improve if starting from a poor strategy profile.

To alleviate CFR's large memory requirement in large games such as heads-up no-limit Texas Hold'em, Moravk et al. (2017) proposed a seminal approach called DeepStack which uses fully connected neural networks to represent players' counterfactual values and obtain a strategy online as requested. However, the strategy is still represented as a tabular form and the quality of this solution depends a lot on the initial quality of the counterfactual network. Furthermore, the counterfactual network is estimated separately, and it is not easy to continue improving both counterfactual network and the tabular strategy profile in an end-to-end optimization framework.

Heinrich et al. (2015); Heinrich & Silver (2016) proposed end-to-end fictitious self-play approaches (XFP and NFSP respectively) to learn the approximate Nash equilibrium with deep reinforcement learning. In a fictitious play model, strategies are represented as neural networks and the strategies are updated by selecting the best responses to their opponents' average strategies. This approach is advantageous in the sense that the approach does not rely on abstracting the game, and in theory, the strategy should continually improve as the algorithm iterates more steps. However, these methods do not explicitly take into account the hidden information in a game, and in experiments for games

such as Leduc Hold'em, these methods converge slower than tabular based counterfactual regret minimization algorithms.

Thus it remains an open question whether the purely neural-based end-to-end approach can achieve comparable performance to tabular based CFR approach. In the paper, we partially resolve this open question by designing a double neural counterfactual regret minimization algorithm which can match the performance of tabular based counterfactual regret minimization algorithm. We employed two neural networks, one for the cumulative regret, and the other for the average strategy. We show that care algorithm design allows these two networks to track the cumulative regret and average strategy respectively, resulting in a converging neural strategy. Furthermore, in order to improve the convergence of the neural algorithm, we also developed a new sampling technique which has lower variance than the outcome sampling, while being more memory efficient than the external sampling. In experiments with Leduc Hold'em and One-card poker, we showed that the proposed double neural algorithm can converge to comparable results produced by its tabular counterpart while performing much better than deep reinforcement learning method. The current results open up the possibility for a purely neural approach to directly solve large IIG.

2 BACKGROUND

In this section, we will introduce some background on IIG and existing approaches to solve them.

2.1 REPRESENTATION OF EXTENSIVE-FORM GAME

We define the components of an extensive-form game following Osborne & Ariel (1994) (page $200 \sim 201$). A finite set $N = \{0, 1, ..., n-1\}$ of **players**. Define h_i^v as the **hidden variable** of player i in IIG, $e.g.$, in poker game h_i^v refers to the private cards of player i. H refers to a finite set of histories. Each member $h = (h_i^v)_{i=0,1,...,n-1}(a_l)_{l=0,...,L-1} = h_0^v h_1^v ... h_{n-1}^v a_0 a_1 ... a_{L-1}$ of H denotes a possible **history** (or state), which consists of each player's hidden variable and L actions taken by players including chance. For player i, h also can be denoted as $h_i^v h_{-i}^v a_0 a_1 ... a_{L-1}$, where h_{-i}^v refers to the opponent's hidden variables. The empty sequence $\emptyset$ is a member of H. $h_j \sqsubseteq h$ denotes h_j is a prefix of h, where $h_j = (h_i^v)_{i=0,1,...,n-1}(a_l)_{l=1,...,L'-1}$ and $0 < L' < L$. $Z \subseteq H$ denotes the terminal histories and any member $z \in Z$ is not a prefix of any other sequences. $A(h) = \{a : ha \in H\}$ is the set of available actions after non-terminal history $h \in H \setminus Z$. A **player function** P assigns a member of $N \cup \{c\}$ to each non-terminal history, where c denotes the chance player id, which usually is -1. $P(h)$ is the player who takes an action after history h. $\mathcal{I}_i$ of a history $\{h \in H : P(h) = i\}$ is an **information partition** of player i. A set $I_i \in \mathcal{I}_i$ is an **information set** of player i and $I_i(h)$ refers to information set I_i at state h. Generally, I_i could only remember the information observed by player i including player $i's$ hidden variable and public actions. Therefore I_i indicates a sequence in IIG, $i.e.$, $h_i^v a_0 a_2 ... a_{L-1}$. For $I_i \in \mathcal{I}_i$ we denote by $A(I_i)$ the set $A(h)$ and by $P(I_i)$ the player $P(h)$ for any $h \in I_i$. For each player $i \in N$ a utility function $u_i(z)$ define the payoff of the terminal state z. A more detailed explanation of these notations and definitions is presented in section B.

2.2 STRATEGY AND NASH EQUILIBRIUM

The strategy in an extensive-form game contains the following components. A **strategy profile** $\sigma = \{\sigma_i | \sigma_i \in \Sigma_i, i \in N\}$ is a collection of strategies for all players, where Σ_i is the set of all possible strategies for player i. σ_{-i} refers to all strategies in σ expect σ_i. For play $i \in N$ the **strategy** $\sigma_i(I_i)$ is a function, which assigns an action distribution over $A(I_i)$ to information set I_i. $\sigma_i(a|h)$ denotes the probability of action a taken by player $i \in N \cup \{c\}$ at state h. In IIG, $\forall h_1, h_2 \in I_i$, we have $I_i = I_i(h_1) = I_i(h_2)$, $\sigma_i(I_i) = \sigma_i(h_1) = \sigma_i(h_2)$, $\sigma_i(a|I_i) = \sigma_i(a|h_1) = \sigma_i(a|h_2)$. For iterative method such as CFR, σ^t refers to the strategy profile at t-th iteration. The **state reach probability** of history h is denoted by $\pi^\sigma(h)$ if players take actions according to σ. For an empty sequence $\pi^\sigma(\emptyset) = 1$. The reach probability can be decomposed into $\pi^\sigma(h) = \prod_{i \in N \cup \{c\}} \pi_i^\sigma(h) = \pi_i^\sigma(h)\pi_{-i}^\sigma(h)$ according to each player's contribution, where $\pi_i^\sigma(h) = \prod_{h'a \sqsubseteq h, P(h')=P(h)} \sigma_i(a|h')$ and $\pi_{-i}^\sigma(h) = \prod_{h'a \sqsubseteq h, P(h') \neq P(h)} \sigma_{-i}(a|h')$. The **information set reach probability** of I_i is defined as $\pi^\sigma(I_i) = \sum_{h \in I_i} \pi^\sigma(h)$. If $h' \sqsubseteq h$, the **interval state reach probability** from state h' to h is defined as $\pi^\sigma(h', h)$, then we have $\pi^\sigma(h', h) = \pi^\sigma(h)/\pi^\sigma(h')$. $\pi_i^\sigma(I_i)$, $\pi_{-i}^\sigma(I_i)$, $\pi_i^\sigma(h', h)$, and $\pi_{-i}^\sigma(h', h)$ are defined similarly.

2.3 COUNTERFACTUAL REGRET MINIMIZATION

In large and zero-sum IIG, CFR is proved to be an efficient method to compute Nash equilibrium (Zinkevich et al., 2008; Brown & Sandholm, 2017; Moravk et al., 2017). We present some key ideas of this method as follows.

Lemma 1: The state reach probability of one player is proportional to posterior probability of the opponent's hidden variable, $i.e., p(h_i^v|I_i) \propto \pi_{-i}^{\sigma}(h)$, where h_i^v and I_i indicate a particular h. (see the proof in section E.1)

For player i and strategy profile σ, the **counterfactual value (CFV)** $v_i^{\sigma}(h)$ at state h is define as

$$v_i^{\sigma}(h) = \sum_{h \sqsubseteq z, z \in Z} \pi_{-i}^{\sigma}(h)\pi^{\sigma}(h,z)u_i(z) = \sum_{h \sqsubseteq z, z \in Z} \pi_i^{\sigma}(h,z)u_i'(z). \tag{1}$$

where $u_i'(z) = \pi_{-i}^{\sigma}(z)u_i(z)$ is the expected reward of player i with respective to the approximated posterior distribution of the opponent's hidden variable. The **action counterfactual value** of taking action a is $v_i^{\sigma}(a|h) = v_i^{\sigma}(ha)$ and the regret of taking this action is $r_i^{\sigma}(a|h) = v_i^{\sigma}(a|h) - v_i^{\sigma}(h)$. Similarly, the CFV of information set I_i is $v_i^{\sigma}(I_i) = \sum_{h \in I_i} v_i^{\sigma}(h)$ and the regret is $r_i^{\sigma}(a|I_i) = \sum_{z \in Z, ha \sqsubseteq z, h \in I_i} \pi_i^{\sigma}(ha,z)u_i'(z) - \sum_{z \in Z, h \sqsubseteq z, h \in I_i} \pi_i^{\sigma}(h,z)u_i'(z)$. Then the **cumulative regret** of action a after T iterations is

$$R_i^T(a|I_i) = \sum_{t=1}^{T}(v_i^{\sigma^t}(a|I_i) - v_i^{\sigma^t}(I_i)) = R_i^{T-1}(a|I_i) + r_i^{\sigma^T}(a|I_i). \tag{2}$$

where $R_i^0(a|I_i) = 0$. Define $R_i^{T,+}(a|I_i) = \max(R_i^T(a|I_i), 0)$, the **current strategy** at $T+1$ iteration will be updated by

$$\sigma_i^{T+1}(a|I_i) = \begin{cases} \frac{R_i^{T,+}(a|I_i)}{\sum_{a \in A(I_i)} R_i^{T,+}(a|I_i)} & \text{if } \sum_{a \in A(I_i)} R_i^{T,+}(a|I_i) > 0 \\ \frac{1}{|A(I_i)|} & \text{otherwise.} \end{cases} \tag{3}$$

The **average strategy** $\bar{\sigma}_i^T$ from iteration 1 to T is defined as:

$$\bar{\sigma}_i^T(a|I_i) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I_i)\sigma_i^t(a|I_i)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I_i)}. \tag{4}$$

where $\pi_i^{\sigma^t}(I_i)$ denotes the information set reach probability of I_i at t-th iteration and is used to weight the corresponding current strategy $\sigma_i^t(a|I_i)$. Define $s_i^t(a|I_i) = \pi_i^{\sigma^t}(I_i)\sigma_i^t(a|I_i)$ as the additional numerator in iteration t, then the cumulative numerator can be defined as

$$S^T(a|I_i) = \sum_{t=1}^{T} \pi_i^{\sigma^t}(I_i)\sigma_i^t(a|I_i) = S^{T-1}(a|I_i) + s_i^T(a|I_i). \tag{5}$$

where $S^0(a|I_i) = 0$.

2.4 MONTE CARLO CFR

When solving a game, CFR needs to traverse the entire game tree in each iteration, which will prevent it from handling large games with limited memory. To address this challenge, Lanctot et al. (2009) proposed a Monte Carlo CFR to minimize counterfactual regret. Their method can compute an unbiased estimation of counterfactual value and avoid traversing the entire game tree. Since only subsets of all information sets are visited in each iteration, this approach requires less memory than standard CFR.

Define $\mathcal{Q} = \{Q_1, Q_2, ..., Q_m\}$, where $Q_j \in Z$ is a **block** of sampling terminal histories in each iteration, such that $\mathcal{Q}_j$ spans the set Z. Generally, different Q_j may have an overlap according to the specify sampling schema. Specifically, in the external sampling and outcome sampling, each block $Q_j \in \mathcal{Q}$ is a partition of Z. Define q_{Q_j} as the probability of considering block Q_j, where $\sum_{j=1}^{m} q_{Q_j} = 1$. Define $q(z) = \sum_{j:z \in Q_j} q_{Q_j}$ as the probability of considering a particular terminal history z. Specifically, vanilla CFR is a special case of MCCFR, where $\mathcal{Q} = \{Z\}$ only contain one block and $q_{Q_1} = 1$. In outcome sampling, only one trajectory will be sampled, such that $\forall Q_j \in \mathcal{Q}$, $|Q_j| = 1$ and $|\mathcal{Q}_j| = |Z|$. For information set I_i, a sample estimate of **counterfactual value** is $\tilde{v}_i^{\sigma}(I_i|Q_j) = \sum_{h \in I_i, z \in Q_j, h \sqsubseteq z} \frac{1}{q(z)} \pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h,z)u_i(z)$.

Lemma 2: The sampling counterfactual value in MCCFR is the unbiased estimation of actual counterfactual value in CFR. $E_{j\sim q_{Q_j}}[\tilde{v}_i^{\sigma}(I_i|Q_j)] = v_i^{\sigma}(I_i)$ (see the proof in section E.2)

Define σ^{rs} as **sampling strategy profile**, where σ_i^{rs} is the sampling strategy for player i and σ_{-i}^{rs} are the sampling strategies for players expect i. Particularly, for both external sampling and outcome sampling proposed by (Lanctot et al., 2009), $\sigma_{-i}^{rs} = \sigma_{-i}$. The regret of the sampled action $a \in A(I_i)$ is defined as

$$\tilde{r}_i^{\sigma}((a|I_i)|Q_j) = \sum_{z\in Q_j, ha\sqsubseteq z, h\in I_i} \pi_i^{\sigma}(ha, z)u_i^{rs}(z) - \sum_{z\in Q_j, h\sqsubseteq z, h\in I_i} \pi_i^{\sigma}(h, z)u_i^{rs}(z) \quad , \tag{6}$$

where $u_i^{rs}(z) = \frac{u_i(z)}{\pi_i^{\sigma^{rs}}(z)}$ is a new utility weighted by $\frac{1}{\pi_i^{\sigma^{rs}}(z)}$. The sample estimate for **cumulative regret** of action a after T iterations is $\tilde{R}_i^T((a|I_i)|Q_j) = \tilde{R}_i^{T-1}((a|I_i)|Q_j) + \tilde{r}_i^{\sigma^T}((a|I_i)|Q_j)$ with $\tilde{R}_i^0((a|I_i)|Q_j) = 0$.

3 Double Neural Counterfactual Regret Minimization

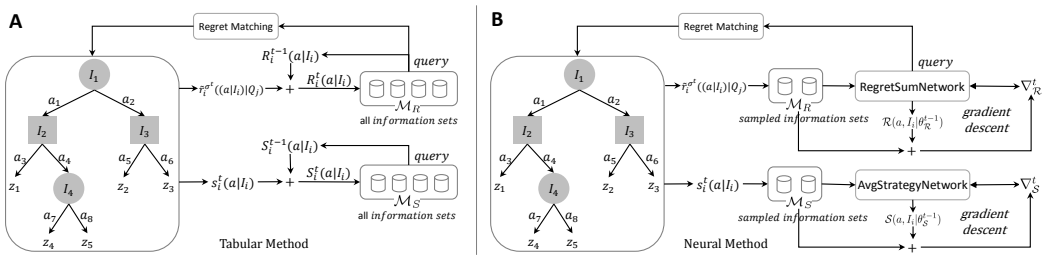

Figure 1: (A) tabular based CRF, and (B) our double neural based CRF framework.

In this section, we will explain our double neural CFR algorithm, where we employ two neural networks, one for the cumulative regret, and the other for the average strategy.

As shown in Figure 1 (A), standard CFR-family methods such as CFR (Zinkevich et al., 2008), outcome-sampling MCCFR, external sampling MCCFR (Lanctot et al., 2009), and CFR+ (Tammelin, 2014) need to use two large tabular-based memories $\mathcal{M}_R$ and $\mathcal{M}_S$ to record the cumulative regret and average strategy for all information sets. Such tabular representation makes these methods difficult to apply to large extensive-form games with limited time and space (Burch, 2017).

In contrast, we will use two deep neural networks to compute approximate Nash equilibrium of IIG as shown in Figure 1 (B). Different from NFSP, our method is based on the theory of CFR, where the first network is used to learn the cumulative regret and the other is to learn the cumulative numerator of the average strategy profile. With the help of these two networks, we do not need to use a large memory to save the key information of the entire game tree. In practice, the proposed double neural method can achieve a lower exploitability with fewer iterations than NFSP. In addition, we present experimentally that our double neural CFR can also continually improve after initialization from a poor tabular strategy.

3.1 Overall Framework

An algorithm in the CFR framework needs to be able to answer two queries:

1. what is the current strategy $\sigma^{t+1}(a|I_i)$ for iteration $t+1$;
2. and what is the average strategy $\bar{\sigma}_i^t(a|I_i)$ after t iterations;

$\forall i \in N, \forall I_i \in \mathcal{I}_i, \forall a \in A(I_i), \forall t \in [1, T]$. Thus, our neural networks are designed to address the needs for these two queries respectively.

For the first query. According to Eq. (3), current strategy $\sigma^{t+1}(a|I_i)$ is computed by the cumulative regret $R^t(a|I_i)$. Given information set I_i and action a, we design a neural network **Regret-SumNetwork(RSN)** $\mathcal{R}(a, I_i|\theta_{\mathcal{R}}^t)$ to learn $R^t(a|I_i)$, where $\theta_{\mathcal{R}}^t$ is the parameter in the network at t-th iteration. As shown Figure 1 (b), define memory $\mathcal{M}_R = \{(I_i, \tilde{r}_i^{\sigma^t}((a|I_i)|Q_j))|\forall i \in N, \forall a \in A(I_i), h \in I_i, h \sqsubseteq z, z \in Q_j\}$. Each member of $\mathcal{M}_R$ is the visited information set I_i and the corresponding regret $\tilde{r}_i^{\sigma^t}((a|I_i)|Q_j)$, where Q_j is the sampled block in t-th iteration. According

to Eq. (2), we can estimate $\mathcal{R}(a, I_i|\theta_{\mathcal{R}}^{t+1})$ using the following optimization:

$$\theta_{\mathcal{R}}^{t+1} \leftarrow \underset{\theta_{\mathcal{R}}^{t+1}}{\operatorname{argmin}} \sum_{(I_i, \tilde{r}_i^{\sigma^t}((a|I_i)|Q_j)) \in \mathcal{M}_R} \left(\mathcal{R}(a, I_i|\theta_{\mathcal{R}}^t) + \tilde{r}_i^{\sigma^t}((a|I_i)|Q_j) - \mathcal{R}(a, I_i|\theta_{\mathcal{R}}^{t+1}) \right)^2. \quad (7)$$

For the second query. According to Eq. (4), the approximate Nash equilibrium is the weighted average of all previous strategies over T iterations. We only need to track the numerator in Eq. (4) since the denominator is used to normalize the summation. Similar to the cumulative regret, we employ another deep neural network **AvgStrategyNetwork(ASN)** to learn the cumulative numerator of the average strategy. Define $\mathcal{M}_S = \{(I_i, \pi_i^{\sigma^t}(I_i)\sigma_i^t(a|I_i)) | \forall i \in N, \forall a \in A(I_i), h \in I_i, h \sqsubseteq z, z \in Q_j\}$. Each member of $\mathcal{M}_S$ is the visited information set I_i and the value of $\pi_i^{\sigma^t}(I_i)\sigma_i^t(a|I_i)$, where Q_j is the sampled block in t-th iteration. Then the parameter $\theta_{\mathcal{S}}^{t+1}$ can estimated by the following optimization:

$$\theta_{\mathcal{S}}^{t+1} \leftarrow \underset{\theta_{\mathcal{S}}^{t+1}}{\operatorname{argmin}} \sum_{(I_i, s_i^t(a|I_i)) \in \mathcal{M}_S} \left(\mathcal{S}(a, I_i|\theta_{\mathcal{S}}^t) + s_i^t(a|I_i) - \mathcal{S}(a, I_i|\theta_{\mathcal{S}}^{t+1}) \right)^2. \quad (8)$$

Relation between CFR, MCCFR and our double neural method. As shown in Figure 1, these three methods are based on the CFR framework. The CFR computes counterfactual value and regret by traversing the entire tree in each iteration, which makes it computationally intensive to be applied to large games directly. MCCFR samples a subset of information sets and will need less computation than CFR in each iteration. However, both CFR and MCCFR need two huge memories to save the cumulative regrets and the numerators of average strategy for all information sets after multiple iterations, which prevents these two methods to be used in large games directly. The proposed neural method keeps the benefit of MCCFR yet without the need for a large memory.

3.2 Recurrent Neural Network Representation for Information Set

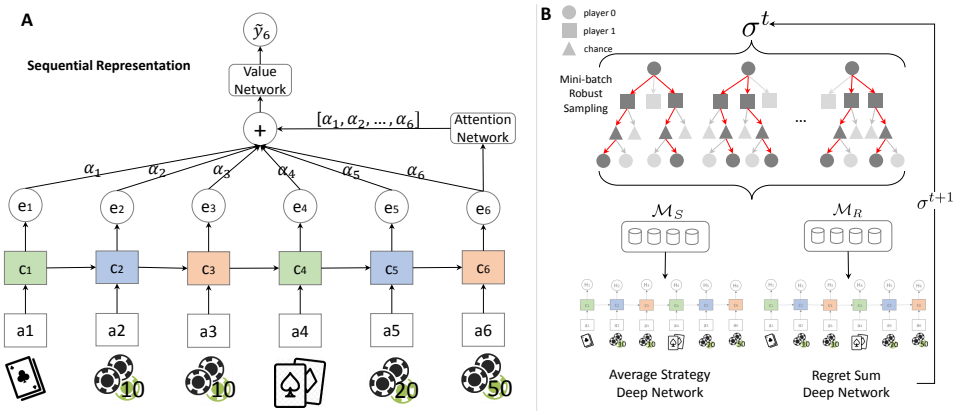

Figure 2: (A) the key architecture of the sequential neural networks. (B) an overview of the novel double neural counterfactual regret minimization method.

In order to define our $\mathcal{R}$ and $\mathcal{S}$ network, we need to represent the information set $I_i \in \mathcal{I}$ in extensive-form games. In such games, players take action in alternating fashion and each player makes a decision according to the observed history. In this paper, we model the behavior sequence as a recurrent neural network and each action in the sequence corresponds to a cell in RNN. Figure 2 (A) provides an illustration of the proposed deep sequential neural network representation for information sets.

In standard RNN, the recurrent cell will have a very simple structure, such as a single tanh or sigmoid layer. Hochreiter & Schmidhuber (1997) proposed a long short-term memory method (LSTM) with the gating mechanism, which outperforms the standard version and is capable of learning long-term dependencies. Thus we will use LSTM for the representation. Furthermore, different position in the sequence may contribute differently to the decision making, we will add an attention mechanism (Desimone & Duncan, 1995; Cho et al., 2015) to the LSTM architecture to enhance the representation. For example, the player may need to take a more aggressive strategy after beneficial public cards are revealed. Thus the information, after the public cards are revealed may be more important.

More specifically, for l-th cell, define x_l as the input vector (which can be either player or chance actions), e_l as the hidden layer embedding, ϕ_* as a general nonlinear function. Each action is represented by a LSTM cell, which has the ability to remove or add information to the cell state with three different gates. Define the notation $\cdot$ as element-wise product. The first **forgetting gate** layer is defined as $g_l^f = \phi_f(w^f \cdot [x_l, e_{l-1}])$, where $[x_l, e_{l-1}]$ denotes the concatenation of x_l and e_{l-1}. The second **input gate** layer decides which values to update and is defined as $g_l^i = \phi_i(w^i \cdot [x_l, e_{l-1}])$. A nonlinear layer output a vector of new candidate values $\tilde{C}_l = \phi_c(w^l \cdot [x_l, e_{l-1}])$ to decide what can be added to the state. After the forgetting gate and the input gate, the new cell state is updated by $C_l = g_l^f \cdot C_{l-1} + g_l^i \cdot \tilde{C}_l$. The third **output gate** is defined as $g_l^o = \phi_o(w^o \cdot [x_l, e_{l-1}])$. Finally, the updated hidden embedding is $e_l = g_l^o \cdot \phi_e(C_l)$. As shown in Figure 2 (A), for each LSTM cell j, the vector of attention weight is learned by an **attention network**. Each member in this vector is a scalar $\alpha_j = \phi_a(w^a \cdot e_j)$. The attention embedding of l-th cell is then defined as $e_l^a = \sum_{j=1}^{l} \alpha_j e_j$, which is the summation of the hidden embedding e_j and the learned attention weight α_j. The final output of the network is predicted by a **value network**, which is defined as

$$\tilde{y}_l := f(a, I_i|\theta) = w^y \cdot \phi_v(e_l^a) = w^y \cdot \phi_v \left(\sum_{j=1}^{l} \phi_a(w^a \cdot e_j) \cdot e_j \right), \tag{9}$$

where θ is the parameters in the defined sequential neural networks. Specifically, ϕ_f, ϕ_i, ϕ_o are sigmoid functions. ϕ_c and ϕ_e are hyperbolic tangent functions. ϕ_a and ϕ_v are rectified linear functions. The proposed **R**SN and **A**SN share the same neural architecture, but use different parameters. That is $\mathcal{R}(a, I_i|\theta_{\mathcal{R}}^t) = f(a, I_i|\theta_{\mathcal{R}}^t)$ and $\mathcal{S}(a, I_i|\theta_{\mathcal{S}}^t) = f(a, I_i|\theta_{\mathcal{S}}^t)$. $\mathcal{R}(\cdot, I_i|\theta_{\mathcal{R}}^t)$ and $\mathcal{S}(\cdot, I_i|\theta_{\mathcal{S}}^t)$ denote two vectors of inference value for all $a \in A(I_i)$.

3.3 Continual Improvement

With the proposed framework of double neural CFR, it is easy to initialize the neural networks from an existing strategy profile based on the tabular representation or neural representation. For information set I_i and action a, in an existing strategy profile, define $R_i'(a|I_i)$ as the cumulative regret and $S^{'}(a|I_i)$ as the cumulative numerator of average strategy. We can clone the cumulative regret for all information sets and actions by optimizing

$$\theta_{\mathcal{R}}^* \leftarrow \underset{\theta_{\mathcal{R}}}{\operatorname{argmin}} \sum_{i \in N, I_i \in \mathcal{I}_i, a \in A(I_i)} \left(\mathcal{R}(a, I_i|\theta_{\mathcal{R}}) - R^{'}(a|I_i) \right)^2. \tag{10}$$

Similarly, the parameters $\theta_{\mathcal{S}}^*$ for cloning the cumulative numerator of average strategy can be optimized in the same way. Based on the learned $\theta_{\mathcal{R}}^*$ and $\theta_{\mathcal{S}}^*$, we can warm start the double neural networks and continually improve beyond the tabular strategy profile.

3.4 Overall Algorithm

Algorithm 1: Counterfactual Regret Minimization with Two Deep Neural Networks

Function `Agent` (T, b)**:**
  For $t = 1$ *to* T **do**
    if $t = 1$ ***and*** *using warm starting* **then**
      initialize $\theta_{\mathcal{R}}^t$ and $\theta_{\mathcal{S}}^t$ from an existing checkpoint
      $t \leftarrow t + 1$ ▷ skip cold starting
    else
      initialize $\theta_{\mathcal{R}}^t$ and $\theta_{\mathcal{S}}^t$ randomly.
    $\mathcal{M}_{\mathcal{R}}, \mathcal{M}_{\mathcal{S}} \leftarrow$ sampling methods for CFV and average strategy. ▷ such as Algorithm3
    sum aggregate the value in $\mathcal{M}_R$ by information set. ▷ according to the **Lemma 5** and Equation 12
    remove duplicated records in $\mathcal{M}_S$.
    $\theta_{\mathcal{R}}^t \leftarrow$ NeuralAgent$(\mathcal{R}(\cdot|\theta_{\mathcal{R}}^{t-1}), \mathcal{M}_R, \theta_{\mathcal{R}}^{t-1}, \beta_{\mathcal{R}}^*)$ ▷ update $\theta_{\mathcal{R}}^t$ using Algorithm2
    $\theta_{\mathcal{S}}^t \leftarrow$ NeuralAgent$(\mathcal{S}(\cdot|\theta_{\mathcal{S}}^{t-1}), \mathcal{M}_S, \theta_{\mathcal{S}}^{t-1}, \beta_{\mathcal{S}}^*)$ ▷ update $\theta_{\mathcal{S}}^t$ using Algorithm2
  return $\theta_{\mathcal{R}}^t, \theta_{\mathcal{S}}^t$

Algorithm 1 provides a summary of the proposed double neural counterfactual regret minimization algorithm. In the first iteration, if the system warm starts from tabular based CFR or MCCFR methods, the techniques in section 3.3 will be used to clone the cumulative regrets and strategy. If there

is no warm start initialization, we can start our algorithm by randomly initializing the parameters in RSN and ASN at iteration $t = 1$. Then sampling methods will return the counterfactual regret and the numerator of average strategy for the sampled information sets in this iteration, and they will be saved in memories $\mathcal{M}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{S}}$ respectively. Then these samples will be used by the NeuralAgent algorithm from Algorithm2 to optimize RSN and ASN. Further details for the sampling methods and the NeuralAgent fitting algorithm will be discussed in the next section.

4 EFFICIENT TRAINING

In this section, we will propose three techniques to improve the efficiency of the double neural method. These algorithms can also be used separately in other CFR-family methods.

4.1 ROBUST SAMPLING TECHNIQUES

Theoretically, outcome sampling is more memory efficient than the external sampling, since in outcome sampling only one trajectory is sampled according to strategy profile while in the external sampling, player i will traverse all actions for all information set $I_i \in \mathcal{I}_i$ and the opponent players including chance sample one action. Therefore many information sets will be visited in a sampling process and block $Q_i \in \mathcal{Q}$ will contains many terminal nodes in external sampling. In outcome sampling method, the weighted utility $u_i^{rs}(z) = \frac{u_i(z)}{\pi^{\sigma_i}(z)}$ for terminal node depends on the concrete reach probability $\sigma_i(z)$ in each iteration, therefore it will lead to a high variance and slow down the convergence of the resulting strategy profile.

In this paper, we proposed a new and robust sampling technique which has lower variance than outcome sampling, while being more memory efficient than the external sampling. In this robust sampling method, the sampling profile is defined as $\sigma^{rs(k)} = (\sigma_i^{rs(k)}, \sigma_{-i})$, where player i will randomly select k actions according to sampling strategy $\sigma_i^{rs(k)}(I_i)$ for each information set I_i and other players will randomly select one action according to strategy σ_{-i}.

Specifically, if player i randomly selects $min(k, |A(I_i)|)$ actions according to discrete uniform distribution $unif(0, |A(I_i)|)$ at information set I_i, *i.e.*, $\sigma_i^{rs(k)}(a|I_i) = \frac{min(k,|A(I_i)|)}{|A(I_i)|}$, then

$$\pi_i^{\sigma^{rs(k)}}(I_i) = \prod_{h \in I_i, h' \sqsubseteq h, h'a \sqsubseteq h, h' \in I_i'} \frac{min(k, |A(I_i')|)}{|A(I_i')|} \tag{11}$$

and the weighted utility $u_i^{rs(k)}(z)$ will be a constant number in each iteration, which has a low variance. In addition, because the weighted utility no longer requires explicit knowledge of the opponent's strategy, we can use this sampling method for online regret minimization. For simplicity, $k = max$ refers to $k = max_{I_i \in \mathcal{I}}|A(I_i)|$ in the following sections.

Lemma 3: If $k = max$ and $\forall i \in N, \forall I_i \in \mathcal{I}_i, \forall a \in A(I_i), \sigma_i^{rs(k)}(a|I_i) \sim unif(0, |A(I_i)|)$, then robust sampling is the same as external sampling.

Lemma 4: If $k = 1$ and $\sigma_i^{rs(k)} = \sigma_i$, then robust sampling is the same as outcome sampling.

Lemma 3 and **Lemma** 4 provide the relationship between outcome sampling, external sampling, and the proposed robust sampling algorithm. The detailed theoretical analysis are presented in Appendix E.3.

4.2 MINI-BATCH TECHNIQUES

Mini-batch MCCFR: Traditional outcome sampling and external sampling only sample one block in an iteration and provide an unbiased estimation of origin CFV according to **Lemma 2**. In this paper, we present a mini-batch Monte Carlo technique and randomly sample b blocks in one iterations. Let Q^j denote a block of terminals sampled according to the scheme in section 4.1 at $j-$th time, then **mini-batch CFV** with b mini-batches for information set I_i can be defined as

$$\tilde{v}_i^{\sigma}(I_i|b) = \frac{1}{b}\sum_{j=1}^{b}\left(\sum_{h \in I_i, z \in Q^j, h \sqsubseteq z} \frac{\pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h,z)u_i(z)}{q(z)}\right) = \sum_{j=1}^{b}\frac{\tilde{v}_i^{\sigma}(I_i|Q^j)}{b}. \tag{12}$$

Furthermore, we can show that $\tilde{v}_i^{\sigma}(I_i|b)$ is an unbiased estimator of the counterfactual value of I_i: **Lemma 5:** $E_{Q^j \sim \text{Robust Sampling}}[\tilde{v}_i^{\sigma}(I_i|b)] = v_i^{\sigma}(I_i)$. (see the proof in section E.4) Similarly, the

cumulative mini-batch regret of action a is

$$\tilde{R}_i^T((a|I_i)|b) = \tilde{R}_i^{T-1}((a|I_i)|b) + \tilde{v}_i^{\sigma^T}((a|I_i)|b) - \tilde{v}_i^{\sigma^T}(I_i|b) \quad , \tag{13}$$

where $\tilde{R}_i^0((a|I_i)|b) = 0$. In practice, mini-batch technique can sample b blocks in parallel and help MCCFR to converge faster.

Mini-Batch MCCFR+: When optimizing counterfactual regret, CFR+ (Tammelin, 2014) substitutes the regret-matching algorithm (Hart & Mas-Colell, 2000) with regret-matching+ and can converge faster than CFR. However, Burch (2017) showed that MCCFR+ actually converge slower than MCCFR when mini-batch is not used. In our paper, we derive mini-batch version of MCCFR+ which updates cumulative mini-batch regret $\tilde{R}^{T,+}((a|I_i)|b)$ up to iteration T by

$$\tilde{R}^{T,+}((a|I_i)|b) = \begin{cases} \left(\tilde{v}_i^{\sigma^T}((a|I_i)|b) - \tilde{v}_i^{\sigma^T}(I_i|b)\right)^+ & \text{if } T = 0 \\ \left(\tilde{R}_i^{T-1,+}((a|I_i)|b) + \tilde{v}_i^{\sigma^T}((a|I_i)|b) - \tilde{v}_i^{\sigma^T}(I_i|b)\right)^+ & \text{if } T > 0 \end{cases} , \tag{14}$$

where $(x)^+ = max(x, 0)$. In practice, we find that mini-batch MCCFR+ converges faster than mini-batch MCCFR when specifying a suitable mini-batch size.

4.3 Neural Agent for Optimizing Neural Representation

Algorithm 2: Optimization of Deep Neural Network

Function `NeuralAgent`$(f(\cdot|\theta^{T-1}), \mathcal{M}, \theta^{T-1}, \beta^*)$**:**
- initialize $optimizer$, $scheduler$ ▷ gradient descent optimizer and learning rate scheduler
- $\theta^T \leftarrow \theta^{T-1}, l_{best} \leftarrow \infty, t_{best} \leftarrow 0$ ▷ warm starting from the checkpoint of the last iteration
- **For** $t = 1$ *to* β_{epoch} **do**
  - $loss \leftarrow []$ ▷ initialize $loss$ as an empty list
  - **For** *each training epoch* **do**
    - $\{x^{(i)}, y^{(i)}\}_{i=1}^m \sim \mathcal{M}$ ▷ sampling a mini-batch from $\mathcal{M}$
    - $batch_loss \leftarrow \frac{1}{m}\sum_{i=1}^m (f(x^{(i)}|\theta^{T-1}) + y^{(i)} - f(x^{(i)}|\theta^T))^2$
    - back propagation $batch_loss$ with learning rate lr
    - clip gradient of θ^T to $[-\epsilon, \epsilon]^d$ ▷ d is the dimension of θ^T
    - $optimizer(batch_loss)$
    - $loss.append(batch_loss)$
  - $lr \leftarrow sheduler(lr)$ ▷ reduce learning rate adaptively when loss has stopped improving
  - **if** $avg(loss) < \beta_{loss}$ **then**
    - $\theta_{best}^T \leftarrow \theta^T$, early stopping. ▷ if loss is small enough, using early stopping mechanism.
  - **else if** $avg(loss) < l_{best}$ **then**
    - $l_{best} = avg(loss), t_{best} \leftarrow t, \theta_{best}^T \leftarrow \theta^T$
  - **if** $t - t_{best} > \beta_{re}$ **then**
    - $lr \leftarrow \beta_{lr}$ ▷ reset learning rate to escape from potential saddle point or local minima.
- **return** θ^T

Define β_{epoch} as training epoch, β_{lr} as learning rate, β_{loss} as the criteria for early stopping, β_{re} as the upper bound for the number of iterations from getting the minimal loss last time, θ^{t-1} as the parameter to optimize, $f(\cdot|\theta^{t-1})$ as the neural network, $\mathcal{M}$ as the training sample consisting information set and the corresponding target. To simplify notations, we use β^* to denote the set of hyperparameters in the proposed deep neural networks. $\beta_{\mathcal{R}}^*$ and $\beta_{\mathcal{S}}^*$ refer to the sets of hyperparameters in RSN and ASN respectively. Algorithm 2 presents the details of how to optimize the proposed neural networks.

Both $\mathcal{R}(a, I_i|\theta_{\mathcal{R}}^{t+1})$ and $\mathcal{S}(a, I_i|\theta_{\mathcal{S}}^t)$ are optimized by mini-batch stochastic gradient descent method. In this paper, we use Adam optimizer (Kingma & Ba, 2014) with both momentum and adaptive learning rate. Some other optimizers such as Nadam, RMSprop, Nadam from (Ruder, 2017) are also tried in our experiments, however, they do not achieve better experimental results. In practice, existing optimizers may not return a relatively low enough loss because of potential saddle point or local minima. To obtain a relatively higher accuracy and lower optimization loss, we use a carefully designed scheduler to reduce the learning rate when the loss has stopped decrease. Specifically, the

scheduler reads a metrics quantity, $e.g$, mean squared error, and if no improvement is seen for a number of epochs, the learning rate is reduced by a factor. In addition, we will reset the learning rate in both optimizer and scheduler once loss stops decrease in β_{re} epochs. Gradient clipping mechanism is used to limit the magnitude of the parameter gradient and make optimizer behave better in the vicinity of steep cliffs. After each epoch, the best parameter will be updated. Early stopping mechanism is used once the lowest loss is less than the specified criteria β_{loss}.

5 EXPERIMENT

The proposed double neural CFR algorithm will be evaluated in No-Limit Leduc Hold'em with stack size 5 and One-Card-Poker game with 5 cards. We will compare it with tabular CFR and deep reinforcement learning based method such as NFSP. The experiments show that the proposed double neural algorithm can converge to comparable results produced by its tabular counterpart while performing much better than deep reinforcement learning method. The current results open up the possibility for a purely neural approach to directly solve large IIG. Due to space limit, we present experimental results for One-Card-Poker and the analysis in section C.

Settings. To simplify the expression, the abbreviations of different methods are defined as follows. **XFP** refers to the full-width extensive-form fictitious play method in (Heinrich et al., 2015), **NFSP** refers to the reinforcement learning based fictitious self-play method in (Heinrich & Silver, 2016). **RS-MCCFR** refers to the proposed robust sampling MCCFR. This method with regret matching+ acceleration technique is denoted by **RS-MCCFR+**. To evaluate the contribution of each neural agent, we replace the tabular based cumulative regret and numerator with RSN and ANS separately. These methods only containing one neural network are denoted by **RS-MCCFR+-RSN** and **RS-MCCFR+-ASN** respectively. **RS-MCCFR+-RSN-ASN** refers to the proposed double neural MCCFR. According to **Lemma 3**, if $k = max$, ES-MCCFR is the same with RS-MCCFR. More specifically, we investigated the following questions.

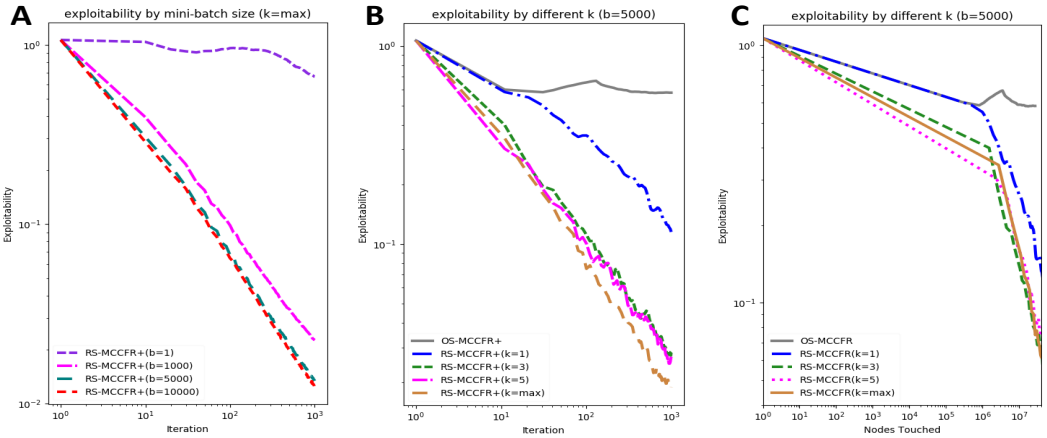

Figure 3: Comparison of different CFR-family methods in Leduc Hold'em. (A) Performance of robust sampling with different batch size. (B) Performance of robust sampling with different parameter k by iteration. (C) Performance by the number of touched node.

Is mini-batch sampling helpful? Figure 3(A) presents the convergence curves of the proposed robust sampling method with $k = max$ under different mini-batch sizes (b=1, 1000, 5000, 10000 respectively). The experimental results show that larger batch sizes generally lead to better strategy profiles. Furthermore, the convergence for $b = 5000$ is as good as $b = 10000$. Thus in the later experiments, we set the mini-batch size equal to 5000.

Is robust sampling helpful? Figure 3 (B) and (C) presents convergence curves for outcome sampling, external sampling($k = max$) and the proposed robust sampling method under the different number of sampled actions. The outcome sampling cannot converge to a low exploitability smaller than 0.1 after 1000 iterations (touch more than 10^7 nodes as shown in Figure 3(C) because of the high variance. The proposed robust sampling algorithm with $k = 1$, which only samples one trajectory like the outcome sampling, can achieve a better strategy profile after the same number of iterations. With an increasing k, the robust sampling method achieves an even better convergence rate. Experiment results show $k = 3$ and 5 have a similar trend with $k = max$, which demonstrates that the proposed robust sampling achieves similar strategy profile but requires less memory than

the external sampling. We choose $k = 3$ for the later experiments in Leduc Hold'em Poker. Figure 3 (C) presents the results in a different way and displays the relation between exploitability and the cumulative number of touched nodes. The robust sampling with small k is just as good as the external sampling while being more memory efficient on the condition that each algorithm touches the same number of nodes.

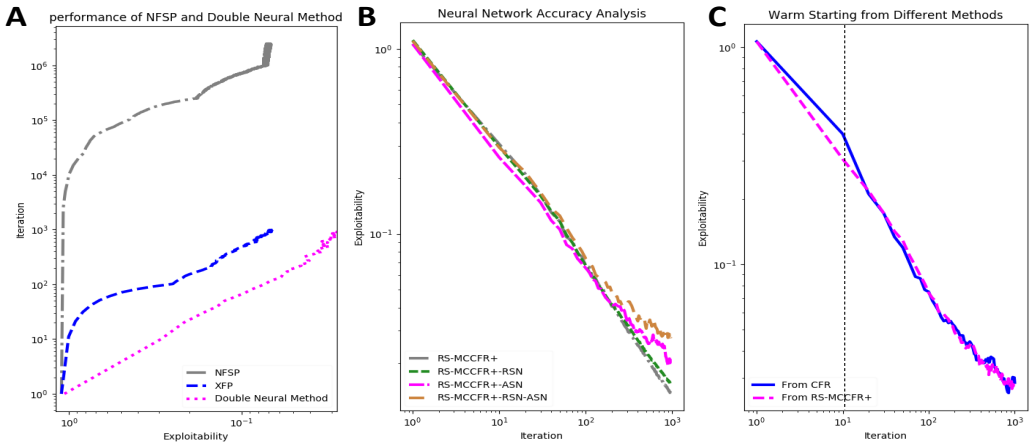

Figure 4: Performance of different methods in Leduc Hold'em. (A) comparison of NSFP, XFP and the proposed double neural method. (B) each contribution of RSN and ASN. (C) continue improvement from tabular based CFR and RS-MCCFR+

How does double neural CRF compared to tabular counterpart, XFP and NFSP? To obtain an approximation of Nash equilibrium, Figure 4(A) demonstrates that NFSP needs 10^6 iterations to reach a 0.06-Nash equilibrium, and requires 2×10^5 state-action pair samples and 2×10^6 samples for supervised learning respectively. The XFP needs 10^3 iterations to obtain the same exploitability, however, this method is the precursor of NFSP and updated by a tabular based full-width fictitious play. Our proposed neural method only needs 200 iterations to achieve the same performance which shows that the proposed double neural algorithm converges significantly better than the reinforcement learning counterpart. In practice, our double neural method can achieve an exploitability of 0.02 after 1000 iterations, which is similar to the tabular method.

What is the individual effect of RSN and ASN? Figure 4(B) presents ablation study of the effects of RSN and ASN network respectively. Both MCCFR+-RSN and MCCFR+-ASN, which only employ one neural network, perform only slightly better than the double neural method. All the proposed neural methods can match the performance of the tabular based method. For RSN, we set the hyperparameters as follows: neural batch size is 256, hidden size is 128 and learning rate $\beta_{lr} = 0.001$. A scheduler, who will reduce the learning rate based on the number of epochs and the convergence rate of loss, help the neural agent to obtain a high accuracy. The learning rate will be reduced by 0.5 when loss has stopped improving after 10 epochs. The lower bound on the learning rate of all parameters in this scheduler is 10^{-6}. To avoid the algorithm converging to potential local minima or saddle point, we will reset the learning rate to 0.001 and help the optimizer to learn a better performance. θ^T_{best} is the best parameters to achieve the lowest loss after T epochs. If average loss for epoch t is less than the specified criteria β_{loss}=10^{-4}, we will early stop the optimizer. We set $\beta_{epoch} = 2000$ and update the optimizer 2000 maximum epochs. For ASN, we set the hidden size as 256, the loss of early stopping criteria as 10^{-5}. The learning rate will be reduced by 0.7 when loss has stopped improving after 15 epochs. Other hyperparameters in ASN are similar to RSN.

How well does continual improvement work? In practice, we usually want to continually improve our strategy profile from an existing checkpoint (Brown & Sandholm, 2016). In the framework of the proposed neural counterfactual regret minimization algorithm, warm starting is easy and friendly. Firstly, we employ two neural networks to clone the existing tabular based cumulative regret and the numerator of average strategy by optimizing Eq. (10). Then the double neural methods can continually improve the tabular based methods. As shown in Figure 4(C), warm start from either full-width based or sampling based CFR the existing can lead to continual improvements. Specifically, the first 10 iterations are learned by tabular based CFR and RS-MCCFR+. The remaining iterations are continually improved by the double neural method, where $b = 5000, k = max$.

REFERENCES

Noam Brown and Tuomas Sandholm. Strategy-based warm starting for regret minimization in games. pp. 432–438. AAAI, 2016.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, pp. eaao1733, 2017.

Neil Burch. Time and space: Why imperfect information games are hard. PhD thesis, 2017.

Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoderdecoder networks. arXiv preprint arXiv:1507.01053, 2015.

Robert Desimone and John Duncan. Neural mechanisms of selective visual attention. Number 18, pp. 193–222. Annual review of neuroscience, 1995.

Geoffrey J. Gordon. No-regret algorithms for structured prediction problems. Number CMU-CALD-05-112. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2005.

David Harris and Sarah Harris. *Digital design and computer architecture (2nd ed.)*, volume ISBN 978-0-12-394424-5. San Francisco, Calif.: Morgan Kaufmann.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, (65(5)):1127–1150, 2000.

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

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. pp. 805–813. International Conference on Machine Learning, 2015.

Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. Number 8, pp. 1735–1780. Neural computation, 1997.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Marc Lanctot, Waugh Kevin, Zinkevich Martin, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In Advances in neural information processing systems, 2009.

Matej Moravk, Schmid Martin, Burch Neil, Lis Viliam, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, (6337):508–513, 2017.

Martin J. Osborne and Rubinstein Ariel. *A course in game theory*, volume 1. MIT Press, 1994.

Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2017.

Finnegan Southey, Michael P. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. arXiv preprint arXiv:1207.1411, 2012.

Oskari Tammelin. Solving large imperfect information games using cfr+. arXiv preprint, 2014.

Martin Zinkevich, Johanson Michael, Bowling Michael, and Carmelo Piccione. Regret minimization in games with incomplete information. Advances in neural information processing systems, 2008.

APPENDIX A GAME RULES

Leduc Hold'em a two-players IIG of poker, which was first introduced in (Southey et al., 2012). In Leduc Hold'em, there is a deck of 6 cards comprising two suits of three ranks. The cards are often denoted by king, queen, and jack. In No-Limit Leduc Hold'em, the player may wager any amount of chips up to a maximum of that player's remaining stack. There is also no limit on the number of raises and bets in each betting round. There are two rounds. In the first betting round, each player is dealt one card from a deck of 6 cards. In the second betting round, a community (or public) card is revealed from a deck of the remaining 4 cards.

One-Card Poker is a two-players IIG of poker described by (Gordon, 2005). The game rules are defined as follows. Each player is dealt one card from a deck of X cards. The first player can pass or bet, If the first player bet, the second player can call or fold. If the first player pass, the second player can pass or bet. If second player bet, the first player can fold or call. The game ends with two pass, call, fold. The fold player will lose 1 chips. If the game ended with two passes, the player with higher card win 1 chips, If the game end with call, the player with higher card win 2 chips.

APPENDIX B DEFINITION OF EXTENSIVE-FORM GAMES

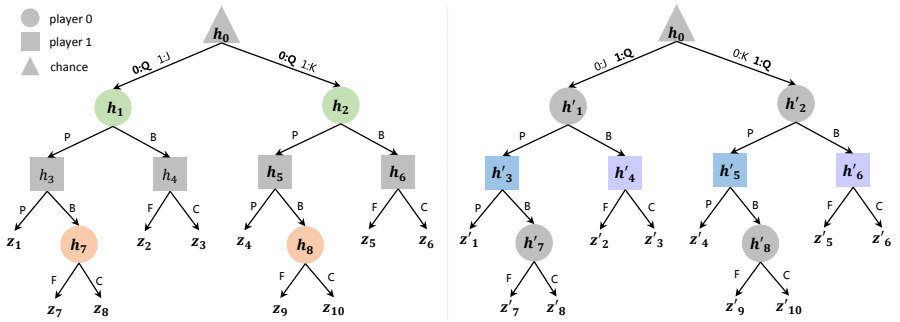

Figure 5: Illustration of extensive-form game. The left and right denote two different kinds of dealt private cards. We use same color other than gray for each state in the same information set. F, C, P, B refer to fold, call, pass, bet respectively.

B.1 ADDITIONAL DEFINITIONS

For player i, the **expected game utility** $u_i^\sigma = \sum_{z \in Z} \pi^\sigma(z) u_i(z)$ of σ is the expected payoff of all possible terminal nodes. Given a fixed strategy profile σ_{-i}, any strategy $\sigma_i^* = \max_{\sigma'_i \in \Sigma_i} u_i^{(\sigma'_i, \sigma_{-i})}$ of player i that achieves maximize payoff against π_{-i}^σ is a **best response**. For two players' extensive-form games, a **Nash equilibrium** is a strategy profile $\sigma^* = (\sigma_0^*, \sigma_1^*)$ such that each player's strategy is a best response to the opponent. An **ϵ-Nash equilibrium** is an approximation of a Nash equilibrium, whose strategy profile σ^* satisfies: $\forall i \in N$, $u_i^{\sigma_i^*} + \epsilon \geq \max_{\sigma'_i \in \Sigma_i} u_i^{(\sigma'_i, \sigma_{-i})}$. **Exploitability** of a strategy σ_i is defined as $\epsilon_i(\sigma_i) = u_i^{\sigma^*} - u_i^{(\sigma_i, \sigma_{-i}^*)}$. A strategy is unexploitable if $\epsilon_i(\sigma_i) = 0$. In large two player zero-sum games such poker, $u_i^{\sigma^*}$ is intractable to compute. However, if the players alternate their positions, the value of a pair of games is zeros, *i.e.*, $u_0^{\sigma^*} + u_1^{\sigma^*} = 0$. We define the exploitability of strategy profile σ as $\epsilon(\sigma) = (u_1^{(\sigma_0, \sigma_1^*)} + u_0^{(\sigma_0^*, \sigma_1)})/2$.

B.2 EXPLANATION BY EXAMPLE

To provide a more detailed explanation, Figure 5 presents an illustration of a partial game tree in One-Card Poker. In the first tree, two players are dealt (queen, jack) as shown in the left subtree and (queen, king) as shown in the right subtree. z_i denotes terminal node and h_i denotes non-terminal node. There are 19 distinct nodes, corresponding 9 non-terminal nodes including chance h_0 and 10 terminal nodes in the left tree. The trajectory from the root to each node is a history of actions. In an extensive-form game, h_i refers to this history. For example, h_3 consists of actions 0:Q, 1:J and P. h_7 consists of actions 0:Q, 1:J, P and B. h_8 consists of actions 0:Q, 1:K, P and B. We have $h_3 \sqsubseteq h_7$, $A(h_7) = \{\text{P}, \text{B}\}$ and $P(h_3) = 1$

In IIG, the private card of player 1 is invisible to player 0, therefore h_7 and h_8 are actually the same for player 0. We use information set to denote the set of these undistinguished states. Similarly, h_1

and h_2 are in the same information set. For the right tree of Figure 5, h'_3 and h'_5 are in the same information set. h'_4 and h'_6 are in the same information set.

Generally, any $I_i \in \mathcal{I}$ could only remember the information observed by player i including player $i's$ hidden variable and public actions. For example, the information set of h_7 and h_8 indicates a sequence of 0:Q, P, and B. Because h_7 and h_8 are undistinguished by player 0 in IIG, all the states have a same strategy. For example, I_0 is the information set of h_7 and h_8, we have $I_0 = I_0(h_7) = I_0(h_8)$, $\sigma_0(I_0) = \sigma_0(h_7) = \sigma_0(h_8)$, $\sigma_0(a|I_0) = \sigma_0(a|h_7) = \sigma_0(a|h_8)$.

APPENDIX C ADDITIONAL EXPERIMENT DETAILS

C.1 FEATURE ENCODING OF POKER GAMES

The feature is encoded as following. As shown in the figure 2 (A), for a history h and player $P(h)$, we use one-hot encoding (Harris & Harris) to represent the observed actions including chance player. For example, the input feature x_l for l-th cell is the concatenation of three one-hot features including the given private cards, the revealed public cards and current action a. Both the private cards and public cards are encoded by one-hot technique, where the value in the existing position is 1 and the others are 0. If there are no public cards, the respective position will be filled with 0. Because the action taking by chance is also a cell in the proposed sequential model. Thus in a No-Limit poker, such as Leduc Hold'em, action a could be any element in $\{\text{fold}, \text{cumulative spent}\} \cup \{\text{public cards}\}$, where cumulative spent denotes the total chips after making a call or raise. The length of the encoding vector of action a is the quantities of public cards plus 2, where cumulative spent is normalized by the stack size.

C.2 ADDITIONAL EXPERIMENT RESULTS

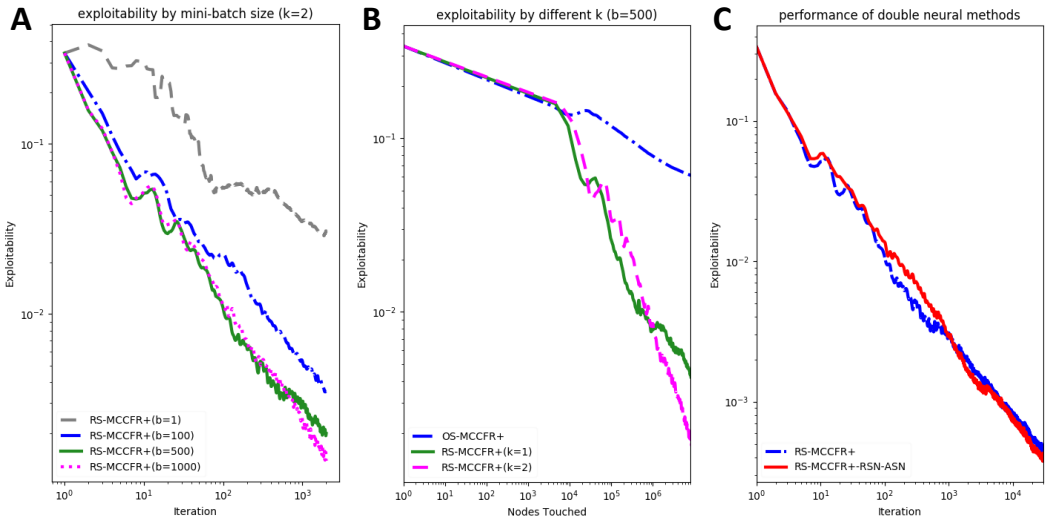

Figure 6: Comparison of different CFR-family methods and neural network methods in One-Card-Poker. (A) Comparison of the robust sampling with different mini-batch size. (B) Comparison of the outcome sampling and the robust sampling with different sample actions k. (C) Comparison of tabular based RS-MCCFR+ and the double neural method.

Figure 6 (A) presents the convergence rate for the proposed robust sampling method of different mini-batch size $b = (1, 100, 500, 1000)$. The experimental results are similar to Leduc Hold'em poker, larger mini-batch size indicates a better exploitability.

Figure 6 (B) demonstrates that the convergence rate for different sampling methods including outcome sampling and robust sampling under $k = 1, 2$. The conclusion is that RS-MCCFR+ converges significantly faster than OS-MCCFR+ after touching the same number of nodes. Experiment results show that $k = 1$ has a similar trend with $k = 2$ (external sampling). Because only one trajectory is sampled, the proposed RS-MCCFR+ will require less memory than the external sampling.

Figure 6 (C) compares the performance between the tabular method and the double neural method. Experimental results demonstrate that RS-MCCFR+-RSN-ASN can achieve an exploitability of less than 0.0004 in One-Card Poker, which matches the performance of the tabular method. For RSN and ASN, we set neural batch size 4, hidden size 32 and learning rate 0.001.

APPENDIX D OPTIMIZE COUNTERFACTUAL REGRET MINIMIZATION WITH TWO DEEP NEURAL NETWORKS

Algorithm 3: Mini-Batch RS-MCCFR with Double Neural Networks

Function `Mini-Batch-MCCFR-NN(t)`**:**
  $\mathcal{M}_{\mathcal{R}} \leftarrow \emptyset, \mathcal{M}_{\mathcal{S}} \leftarrow \emptyset$
  For all $i = 1$ *to* b **do in parallel then**
    MCCFR-NN$(t, \emptyset, 0, 1, 1)$
    MCCFR-NN$(t, \emptyset, 1, 1, 1)$
  return $\mathcal{M}_{\mathcal{R}}, \mathcal{M}_{\mathcal{S}}$

Function `MCCFR-NN(`$t, h, i, \pi_i, \pi_i^{rs(k)}$`)`**:**
  $I_i \leftarrow I_i(h)$ ▷ information set at state h
  if $h \in Z$ **then**
    return $\frac{u_i(h)}{\pi_i^{rs(k)}}$ ▷ return game payoff
  else if $P(h) = -1$ **then**
    $a \sim \sigma_{-i}(I_i)$ ▷ Sample an action from $\sigma_{-i}(h)$
    return MCCFR-NN$(t, ha, i, \pi_i, \pi_i^{rs(k)})$
  else if $P(h) = i$ **then**
    $\hat{R}_i(\cdot|I_i) \leftarrow \mathcal{R}(\cdot, I_i|\theta^t_{\mathcal{R}})$ **if** $t > 1$ **else** $\overrightarrow{0}$ ▷ inference the vector of cumulative regret $\forall a \in A(I_i)$
    $\sigma_i(I_i) \leftarrow$CalculateStrategy$(\hat{R}_i(\cdot|I_i), I_i)$ ▷ calculate current strategy
    $v_i(h) \leftarrow 0, r_i(\cdot|I_i) \leftarrow \vec{0}, s_i(\cdot|I_i) \leftarrow \vec{0}$ ▷ $r_i(\cdot|I_i)$ and $_i(\cdot|I_i)$ are two vectors over $A(I_i)$
    $A^{rs(k)}(I_i) \leftarrow$ sampling k different actions according to $\sigma_i^{rs(k)}$
    For $a \in A^{rs(k)}(I_i)$ **do**
      $v_i(a|h) \leftarrow$MCCFR-NN$(t, ha, i, \pi_i\sigma_i(a|I_i), \pi_i^{rs}\sigma_i^{rs(k)}(a|I_i))$
      $v_i(h) \leftarrow v_i(h) + v_i(a|h)\sigma_i(a|I_i)$ ▷ update counterfactual value
    For $a \in A^{rs(k)}(I_i)$ **do**
      $r_i(a|I_i) \leftarrow v_i(a|h) - v_i(h)$ ▷ update cumulative regret
      $s_i(a|I_i) \leftarrow \pi_i^{\sigma}(I_i)\sigma_i(a|I_i)$ ▷ update average strategy numerator
    Store updated cumulative regret tuple $(I_i, r_i(\cdot|I_i))$ in $\mathcal{M}_{\mathcal{R}}$
    Store updated current strategy dictionary $(I_i, s_i(\cdot|I_i))$ in $\mathcal{M}_{\mathcal{S}}$
  else
    $\hat{R}_{-i}(\cdot|I_i) \leftarrow \mathcal{R}(\cdot, I_i|\theta^t_{\mathcal{R}})$ **if** $t > 1$ **else** $\overrightarrow{0}$ ▷ inference cumulative regret
    $\sigma_{-i}(I_i) \leftarrow$CalculateStrategy$(\hat{R}_{-i}(\cdot|I_i), I_i)$ ▷ calculate current strategy
    $a \sim \sigma_{-i}(I_i)$ ▷ Sample an action from $\sigma_{-i}(I_i)$
    return MCCFR-NN$(t, ha, i, \pi_i, \pi_i^{rs(k)})$

Function `CalculateStrategy(`$R_i(\cdot|I_i), I_i$`)`**:**
  $sum \leftarrow \sum_{a \in A(I_i)} \max(R_i(a|I_i), 0)$
  For $a \in A(I_i)$ **do**
    $\sigma_i(a|I_i) = \frac{\max(R_i(a|I_i), 0)}{sum}$ **if** sum > 0 **else** $\frac{1}{|A(I_i)|}$
  return $\sigma_i(I_i)$

Algorithm 3 presents one application scenario of the proposed double neural method, which is based on the proposed mini-batch robust sampling method. The function MCCFR-NN will traverse the

game tree like tabular MCCFR, which starts from the root history $h = \emptyset$. Define I_i as the information set of h. Suppose that player i will sample k actions according to the robust sampling. Then the function can be defined as follows. (1) If the history is terminal, the function returns the weighted utility. (2) If the history is the chance player, one action $a \in A(I_i)$ will be sampled according to the strategy $\sigma_{-i}(I_i)$. Then this action will be added to the history, $i.e.$, $h \leftarrow ha$. (3) If $P(I_i) = i$, the current strategy can be updated by the cumulative regret predicted by RSN. Then we sample k actions according the specified sampling strategy profile $\sigma_i^{rs(k)}$. After a recursive updating, we can obtain the counterfactual value and regret of each action at I_i. For the visited node, their counterfactual regrets and numerators of the corresponding average strategy will be stored in $\mathcal{M}_{\mathcal{R}}$ and $\mathcal{M}_{\mathcal{S}}$ respectively. (4) If $P(I_i)$ is the opponent, only one action will be sampled according the strategy $\sigma_{-i}(I_i)$.

The function Mini-Batch-MCCFR-NN presents a mini-batch sampling method, where b blocks will be sampled in parallel. This mini-batch method can help the MCCFR to achieve a more accurate estimation of CFV. The parallel sampling makes this method efficient in practice.

APPENDIX E THEORETICAL ANALYSIS

E.1 REACH PROBABILITY AND POSTERIOR PROBABILITY

Lemma 1: The state reach probability of one player is proportional to posterior probability of the opponent's hidden variable, $i.e.$, $p(h_i^v|I_i) \propto \pi_{-i}^{\sigma}(h)$.

Proof: For player i at information set I_i and fixed $i's$ strategy profile σ_i, $i.e.$, $\forall h \in I_i, \pi_i^{\sigma}(h)$ is constant. Based on the defination of extensive-form game in Section 2.1, the cominbation of I_i and opponent's hidden state h_{-i}^v can indicate a particular history $h = h_i^v h_{-i}^v a_0 a_1 ... a_{L-1}$. With Bayes' Theorem, we can inference the posterior probability of opponent's private cards with Equation15

$$
\begin{aligned}
p(h_i^v|I_i) &= \frac{p(h_i^v, I_i)}{p(I_i)} = \frac{p(h)}{p(I_i)} \propto p(h) \\
&\propto p(h_i^v)p(h_{-i}^v)\prod_{l=1}^{L}\sigma_{P(h_i^v h_{-i}^v a_0 a_1 ... a_{l-1})}(a_k|h_i^v h_{-i}^v a_0 a_1 ... a_{l-1}) \\
&\propto \pi^{\sigma}(h) = \pi_i^{\sigma}(h)\pi_{-i}^{\sigma}(h) \\
&\propto \pi_{-i}^{\sigma}(h)
\end{aligned}
\tag{15}
$$

E.2 MCCFR PRESENTS AN UNBIASED ESTIMATION OF COUNTERFACTUAL VALUE

Lemma 2: $E_{j\sim q_{Q_j}}[\tilde{v}_i^{\sigma}(I_i|Q_j)] = v_i^{\sigma}(I_i)$

Proof:

$$
\begin{aligned}
E_{j\sim q_{Q_j}}[\tilde{v}_i^{\sigma}(I_i|Q_j)] &= \sum_{j=1}^{m} q_{Q_j} v_i^{\sigma}(I_i|Q_j) \\
&= \sum_{j=1}^{m}\sum_{h\in I_i, z\in Q_j, h\sqsubseteq z}\frac{q_{Q_j}}{q(z)}\pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h,z)u_i(z) \\
&= \sum_{h\in I_i, z\in Z, h\sqsubseteq z}\frac{\sum_{j:z\in Q_j} q_{Q_j}}{q(z)}\pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h,z)u_i(z) \\
&= \sum_{h\in I_i, z\in Z, h\sqsubseteq z}\pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h,z)u_i(z) \\
&= v_i^{\sigma}(I_i)
\end{aligned}
\tag{16}
$$

E.3 ROBUST SAMPLING, OUTCOME SAMPLING AND EXTERNAL SAMPLING

Lemma 3: If $k = max_{I_i\in\mathcal{I}}|A(I_i)|$ and $\forall i \in N, \forall I_i \in \mathcal{I}_i, \forall a \in A(I_i), \sigma_i^{rs(k)}(a|I_i) \sim unif(0, |A(I_i)|)$, then robust sampling is same with external sampling. (see the proof in section E.3)

Lemma 4: If $k = 1$ and $\sigma_i^{rs(k)} = \sigma_i$, then robust sampling is same with outcome sampling. (see the proof in section E.3)

For robust sampling, given strategy profile σ and the sampled block Q_j according to sampling profile $\sigma^{rs(k)} = (\sigma_i^{rs(k)}, \sigma_{-i})$, then $q(z) = \pi_i^{\sigma^{rs(k)}}(z)\pi_{-i}^{\sigma}(z)$, and the regret of action $a \in A^{rs(k)}(I_i)$ is

$$\begin{aligned}
\tilde{r}_i^{\sigma}((a|I_i)|Q_j) &= \tilde{v}_i^{\sigma}((a|I_i)|Q_j) - \tilde{v}_i^{\sigma}(I_i|Q_j) \\
&= \sum_{z \in Q_j, ha \sqsubseteq z, h \in I_i} \frac{1}{q(z)} \pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(ha, z)u_i(z) - \sum_{z \in Q_j, h \sqsubseteq z} \frac{1}{q(z)} \pi_{-i}^{\sigma}(z)\pi_i^{\sigma}(h, z)u_i(z) \\
&= \sum_{z \in Q_j, ha \sqsubseteq z, h \in I_i} \frac{u_i(z)}{\pi_i^{\sigma^{rs(k)}}(z)} \pi_i^{\sigma}(ha, z) - \sum_{z \in Q_j, h \sqsubseteq z, h \in I_i} \frac{u_i(z)}{\pi_i^{\sigma^{rs(k)}}(z)} \pi_i^{\sigma}(h, z) \\
&= \sum_{z \in Q_j, ha \sqsubseteq z, h \in I_i} \pi_i^{\sigma}(ha, z)u_i^{rs}(z) - \sum_{z \in Q_j, h \sqsubseteq z, h \in I_i} \pi_i^{\sigma}(h, z)u_i^{rs}(z),
\end{aligned} \tag{17}$$

where $u_i^{rs}(z) = \frac{u_i(z)}{\pi_i^{\sigma^{rs(k)}}(z)}$ is the weighted utility according to reach probability $\pi_i^{\sigma^{rs(k)}}(z)$. Because the weighted utility no long requires explicit knowledge of the opponent's strategy, we can use this sampling method for online regret minimization.

Generally, if player i randomly selects $min(k, |A(I_i)|)$ actions according to discrete uniform distribution $unif(0, |A(I_i)|)$ at information set I_i, *i.e.*, $\sigma_i^{rs(k)}(a|I_i) = \frac{min(k,|A(I_i)|)}{|A(I_i)|}$, then

$$\pi_i^{\sigma^{rs(k)}}(I_i) = \prod_{h \in I_i, h' \sqsubseteq h, h'a \sqsubseteq h, h' \in I_i'} \frac{min(k, |A(I_i')|)}{|A(I_i')|} \tag{18}$$

and $u_i^{rs}(z)$ is a constant number when given the sampling profile $\sigma^{rs(k)}$.

Specifically,

- if $k = max_{I_i \in I}|A(I_i)|$, then $\sigma_i^{rs(k)}(I_i) = 1$, $u_i^{rs(k)}(z) = u_i(z)$, and

$$\tilde{r}_i^{\sigma}((a|I_i)|Q_j) = \sum_{z \in Q_j, h \sqsubseteq z, h \in I_i} u_i(z)(\pi_i^{\sigma}(ha, z) - \pi_i^{\sigma}(h, z)) \tag{19}$$

  Therefore, **robust sampling is same with external sampling when** $k = max_{I_i \in I}|A(I_i)|$**.**

- if $k = 1$ and $\sigma_i^{rs(k)} = \sigma_i$, only one history z is sampled in this case,then $u_i^{rs(k)}(z) = \frac{u_i(z)}{\pi_i^{\sigma_i}(z)}$, $\exists h \in I_i$, for $a \in A^{rs(k)}(I_i)$

$$\begin{aligned}
\tilde{r}_i^{\sigma}((a|I_i)|Q_j) &= \tilde{r}_i^{\sigma}((a|h)|Q_j) \\
&= \sum_{z \in Q_j, ha \sqsubseteq z} \pi_i^{\sigma}(ha, z)u_i^{rs}(z) - \sum_{z \in Q_j, h \sqsubseteq z} \pi_i^{\sigma}(h, z)u_i^{rs}(z) \\
&= \frac{(1 - \sigma_i(a|h))u_i(z)}{\pi_i^{\sigma}(ha)}
\end{aligned} \tag{20}$$

  For $a \notin A^{rs(k)}(I_i)$, the regret will be $\tilde{r}_i^{\sigma}((a|h)|j) = 0 - \tilde{v}_i^{\sigma}(h|j)$. Therefore, **robust sampling is same with outcome sampling when** $k = 1$ **and** $\sigma_i^{rs(k)} = \sigma_i$**.**

- if $k = 1$, and player i randomly selects one action according to discrete uniform distribution $unif(0, |A(I_i)|)$ at information set I_i, then $u_i^{rs(1)}(z) = \frac{u_i(z)}{\pi_i^{\sigma^{rs(k)}}(z)}$ is a constant, $\exists h \in I_i$,

for $a \in A^{rs(k)}(I_i)$

$$\begin{aligned}\tilde{r}_i^\sigma((a|I_i)|Q_j) &= \sum_{z \in Q_j, ha \sqsubseteq z, h \in I_i} \pi_i^\sigma(ha, z) u_i^{rs}(z) - \sum_{z \in Q_j, h \sqsubseteq z, h \in I_i} \pi_i^\sigma(h, z) u_i^{rs}(z) \\ &= (1 - \sigma_i(a|h)) \pi_i^\sigma(ha, z) u_i^{rs(1)}(z)\end{aligned} \tag{21}$$

if action a is not sampled at state h, the regret is $\tilde{r}_i^\sigma((a|h)|j) = 0 - \tilde{v}_i^\sigma(h|j)$. **Compared to outcome sampling, the robust sampling in that case have a lower variance** because of the constant $u_i^{rs(1)}(z)$.

E.4 Mini-Batch MCCFR gives an unbiased estimation of counterfactual value

Lemma 5: $E_{Q^j \sim \text{Robust Sampling}}[\tilde{v}_i^\sigma(I_i|b)] = v_i^\sigma(I_i)$.

Proof:

$$\begin{aligned}E_{Q^j \sim \text{Robust Sampling}}[\tilde{v}_i^\sigma(I_i|b)] &= E_{b' \sim \text{unif(0,b)}}[\tilde{v}_i^\sigma(I_i|b')] \\ &= E_{b' \sim \text{unif(0, b)}} \left(\sum_{j=1}^{b'} \sum_{h \in I_i, z \in Q^j, h \sqsubseteq z} \frac{\pi_{-i}^\sigma(z) \pi_i^\sigma(h, z) u_i(z)}{q(z) b'} \right) \\ &= E_{b' \sim \text{unif(0, b)}} \left(\frac{1}{b'} \sum_{j=1}^{b'} \tilde{v}_i^\sigma(I_i|Q^j) \right) \\ &= \frac{1}{b} \sum_{b'=1}^{b} \left(\frac{1}{b'} \sum_{j=1}^{b'} \tilde{v}_i^\sigma(I_i|Q^j) \right) \\ &= \frac{1}{b} \sum_{b'=1}^{b} \left(\frac{1}{b'} \sum_{j=1}^{b'} E(\tilde{v}_i^\sigma(I_i|Q^j)) \right) \\ &= \frac{1}{b} \sum_{b'=1}^{b} \left(\frac{1}{b'} \sum_{j=1}^{b'} v_i^\sigma(I_i) \right) \\ &= v_i^\sigma(I_i)\end{aligned} \tag{22}$$