ACTOR-CRITIC POLICY OPTIMIZATION IN A LARGE-SCALE IMPERFECT-INFORMATION GAME

Anonymous authors
Paper under double-blind review

ABSTRACT

The deep policy gradient method has demonstrated promising results in many large-scale games, where the agent learns purely from its own experience. Yet, policy gradient methods suffer convergence problems to a Nash Equilibrium (NE) in multi-agent situations. Counterfactual regret minimization has a convergence guarantee to a NE in two-player zero-sum games, but it usually needs domain-specific abstraction techniques and model-based traversing to deal with large-scale games. To inherit merits from both methods, we extend the actor-critic algorithm framework in deep reinforcement learning to solve a large-scale two-player zero-sum imperfect-information game, 1v1 Mahjong, whose information set size and game length are much larger than Poker. In particular, we modify the policy optimization objective from originally maximizing the discounted returns to minimizing a type of weighted cumulative counterfactual regrets. This modification is achieved by approximating the regrets via a deep neural network and minimizing the regrets via generating self-play strategies using Hedge. We name the proposed algorithm Actor-Critic Hedge (ACH) and derive its theoretical connection to CFR. We prove the convergence of ACH to a NE under certain conditions. Experimental results on the proposed 1v1 Mahjong benchmark and benchmarks from OpenSpiel demonstrate that ACH outperforms related state-of-the-art methods. Also, the bot obtained by ACH defeats a human champion in 1v1 Mahjong.

1 INTRODUCTION

Policy gradient methods using deep neural networks as policy and value approximators have been successfully applied to many large-scale games (Berner et al., 2019; Vinyals et al., 2019; Ye et al., 2020). Usually, a score function representing the discounted returns is maximized by the policy, i.e., the actor. In the meantime, a value function, known as the critic, is learned to guide the directions and magnitudes of the policy gradients. This type of actor-critic methods are efficiently scalable with regard to the game size and the amount of computational resources. However, as pointed out in Srinivasan et al. (2018) and Hennes et al. (2020), policy gradient methods have no convergence guarantee to optimal solutions in competitive Imperfect-Information Games (IIGs). The main reason is that the policy gradient theorem (Sutton et al., 1999) is established within the single agent situation, where the environment is Markovian. However, learning becomes non-stationary and non-Markovian when multiple agents learn simultaneously in a competitive environment.

In an IIG, players have differing information about the current situation of the game. An optimal solution to an IIG usually refers to a Nash Equilibrium (NE), where no player could improve by unilaterally deviating to a different policy. Tremendous progress in computing such optimal solutions has been made by a family of tabular methods: Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2008). CFR is a type of iterative self-play algorithm based on regret minimization, and it guarantees to converge to a NE with regard to the average policy in 2-player zero-sum IIGs. CFR is model-based in the sense that a perfect game model is required to sample many if not all actions from any state during self-play. To handle large-scale IIGs with CFR, abstraction techniques, applied to either the action space or the state space, are usually employed to reduce the game to a manageable size (Bowling et al., 2015; Brown & Sandholm, 2018; 2019). The game is then solved in the reduced form, and the corresponding solution is mapped back to the original game. However, abstraction techniques are often manual and require expertise domain knowledge. Besides, some large-scale IIGs may be inherently difficult to be abstracted.

In this paper, we investigate a large-scale IIG, Mahjong. Previous research on Mahjong has mainly been supervised learning and traditional Reinforcement Learning (RL) (Mizukami & Tsuruoka, 2015; Li et al., 2020b). In contrast, we study Mahjong from a game-theoretic perspective, i.e., aiming for a NE. We focus on a 2-player (1v1) zero-sum Mahjong game, whose information set size and game length are much larger than Poker. We are interested in methods that are model-free, i.e., using only trajectory samples to learn, as it is often infeasible to sample multiple if not all actions for each state in large-scale IIGs. Also, we employ deep neural networks to generalize across states without any abstraction, since the state or action abstraction in 1v1 Mahjong is not as straightforward as in Poker, which is explained in the Appendix. To summarize, we make the following contributions.

- To inherit the scalability of deep RL methods and the convergence property of CFR, we develop a new actor-critic model-free algorithm, named Actor-Critic Hedge (ACH), for approaching a NE in 2-player large-scale IIGs, such as 1v1 Mahjong. ACH employs a deep neural network to approximate a type of weighted cumulative counterfactual regrets of state actions. In the meantime, ACH minimizes the regrets via generating self-play strategies using Hedge (Freund & Schapire, 1997).
- We prove that the exploitability of the average strategy of ACH under certain conditions decreases at the rate of $O(T^{-1/2}) + O(1)$, where T denotes the number of iterations in ACH. The term $O(1)$ disappears if the behavioral policy (i.e., the policy for sampling actions in self-play) is constant across iterations and samples any legal action with non-zero probability.
- To facilitate research on large-scale IIGs and Mahjong, we propose a 1v1 zero-sum Mahjong benchmark. The corresponding game enjoys a large population in online games.
- We build a 1v1 Mahjong bot, named JueJong, based on a novel neural architecture design and ACH. In an initial evaluation against human players, JueJong demonstrates superior performance and has an advantage over a Mahjong champion, Haihua Cheng[1].

2 Notations and Background

In this section, we introduce notations in the context of IIGs and NE. Related background on RL (focusing on policy gradient methods) and CFR are briefly described.

2.1 Imperfect-Information Games and Nash Equilibrium

An IIG is usually represented in an extensive-form, i.e., a game tree. A node (history) $h \in \mathcal{H}$ in the tree represents all information of the current situation. Given a history h, $P : \mathcal{H} \to \mathcal{P} \cup \{c\}$ maps h to the unique player $p \in \mathcal{P}$ who acts at h or the chance player c, where the chance player is responsible for taking actions for random events. When $P(h) \in \mathcal{P}$, the player $P(h)$ has to take an action $a \in \mathcal{A}$. We use $\mathcal{A}(h)$ to represent the set of legal actions of $P(h)$. The set of terminal nodes is denoted as $\mathcal{Z}$. At each $z \in \mathcal{Z}$, there is a payoff $u_p : \mathcal{Z} \to \mathbb{R}$ for each player $p \in \mathcal{P}$. In this paper, we focus on 2-player zero-sum games, where $\mathcal{P} = \{0, 1\}$ and $u_0(z) + u_1(z) = 0$ for each $z \in \mathcal{Z}$.

For either player p, the set of histories $\mathcal{H}$ is partitioned into infosets. We denote the set of infosets for player p as $\mathcal{I}_p$ and an infoset in $\mathcal{I}_p$ as I_p. Two histories $h, h' \in \mathcal{H}$ are in the same infoset if and only if h and h' are indistinguishable from the perspective of player p. As a result, a player p's strategy (or policy) π_p maps an infoset to a probability distribution over legal actions. For the same reason, we define $\mathcal{A}(I_p) = \mathcal{A}(h), \forall h \in I_p$ and $P(I_p) = P(h), \forall h \in I_p$. A strategy profile π is simply a tuple of strategies (π_p, π_{-p}), where π_{-p} represents the player p's opponent strategy. The expected payoff for player p under π is then denoted as $u_p(\pi_p, \pi_{-p})$. We use $\Delta(I)$ to denote the range of payoffs reachable from a history h in infoset I. Let $\Delta = \max_{I \in \mathcal{I}_p, p \in \mathcal{P}} \Delta(I)$. $f^\pi(h)$ denotes the joint reaching probability of h under π. $f^\pi_p(h)$ is the contribution of player p to $f^\pi(h)$, and $f^\pi_{-p}(h)$ is the contribution of the opponent and chance: $f^\pi(h) = f^\pi_p(h) \times f^\pi_{-p}(h)$. We focus on the perfect-recall setting, where $f^\pi_p(h) = f^\pi_p(h'), \forall h, h' \in I_p$. We define $f^\pi_p(I_p) = f^\pi_p(h), \forall h \in I_p$, $f^\pi_{-p}(I_p) = \sum_{h \in I_p} f^\pi_{-p}(h)$, and $f^\pi(I_p) = \sum_{h \in I_p} f^\pi(h) = f^\pi_p(I_p) f^\pi_{-p}(I_p)$.

[1] Haihua Cheng is the Competition Mahjong champion of 2014 World Mahjong Master Tournament, 2018 Tencent Mahjong Tournament, and 2019 Tencent Mahjong Tournament.

A best response $BR(\pi_{-p})$ to π_{-p} is a player p's strategy, which satisfies $u_p(BR(\pi_{-p}), \pi_{-p}) = \max_{\pi'_p} u_p(\pi'_p, \pi_{-p})$. A NE is a strategy profile π^*, where each player plays a best response to the other: $u_p(\pi^*_p, \pi^*_{-p}) = \max_{\pi'_p} u_p(\pi'_p, \pi^*_{-p})$, $\forall p \in \mathcal{P}$. The exploitability of a player's strategy, denoted as $e(\pi_p)$, measures the performance gap between π_p and a NE strategy π^*_p: $e(\pi_p) = u_p(\pi^*_p, \pi^*_{-p}) - u_p(\pi_p, BR(\pi_p))$. The total exploitability is $\epsilon = \sum_{p \in \mathcal{P}} e(\pi_p)$.

2.2 Reinforcement Learning and Policy Gradient Methods

RL usually assumes a Markov decision process, where the agent selects an action a_i from the action set $\mathcal{A}(s_i)$ in state $s_i \in \mathcal{S}$[2] at each time step i. The agent then receives a reward r_i from the environment and transitions to a new state s_{i+1}. The objective in RL is to learn a policy that maximizes the expected discounted returns, i.e., the state value, starting from any state s: $V^\pi(s) = \mathbb{E}^\pi[\sum_{j=i}^{\infty} \gamma^{j-i} r_j | s_i = s] = \mathbb{E}^\pi[G]$, with the discount factor $\gamma \in [0, 1)$.

A large family of methods in RL belong to policy gradient methods, in which the parameters θ of policy $\pi(a|s;\theta)$ are updated by performing gradient ascent directly on $\mathbb{E}^{\pi_\theta}[G]$. One early example is the standard REINFORCE algorithm (Williams, 1992) that updates θ in the direction $\nabla_\theta \log \pi(a|s;\theta) G$, which is an unbiased estimate of $\nabla_\theta \mathbb{E}^{\pi_\theta}[G]$. To further reduce the variance of the gradient, an action-independent baseline is often subtracted from G: $\nabla_\theta \log \pi(a|s;\theta)[G - B(s)]$. A recent policy gradient method, named Advantage Actor-Critic (A2C) (Mnih et al., 2016), learns a parameterized value function as the baseline: $B(s) = V^{\pi_\theta}(s; w)$, where w often shares some parameters with θ. Moreover, the value $G - B(s)$ is replaced in A2C with the estimated advantage of action a in state s: $A(s, a) = Q(s, a) - V(s)$. $Q(s, a)$ is usually estimated using sampled rewards and predicted values of future states. A2C updates parameters θ and w in a synchronous manner, while its asynchronous version is Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016).

In an asynchronous training environment, the behavioral policy is usually different from the learning policy. Proximal Policy Optimization (PPO) (Schulman et al., 2017) takes this discrepancy into account by multiplying A2C gradient with an importance ratio $r(a|s;\theta) = \pi(a|s;\theta)/\pi(a|s;\theta_{old})$, which results in $r(a|s;\theta)\nabla_\theta \log \pi(a|s;\theta)A(s, a) = \nabla_\theta r(a|s;\theta)A(s, a)$. Furthermore, PPO constrains the KL divergence between the learning policy π_θ and the old behavioral policy $\pi_{\theta_{old}}$ by clipping the ratio to a small interval around 1.0.

2.3 Counterfactual Regret Minimization

CFR is an iterative algorithm that minimizes the total regret of strategies by minimizing the cumulative counterfactual regrets in every state (infoset). The cumulative counterfactual regret of an action a in state s is defined as $R^c_t(s, a) = \sum_{k=1}^{t} r^c_k(s, a)$, where $r^c_k(s, a)$ denotes the instantaneous counterfactual regret of action a in state s at iteration k. $r^c_t(s, a)$ is equal to $f^{\pi_t}_{-p}(s) A^{\pi_t}(s, a)$ (as proven in Srinivasan et al. (2018)), where $A^{\pi_t}(s, a)$ is the advantage function of player $p = P(s)$ at state s as defined in traditional RL. Intuitively, $r^c_t(s, a)$ represents how regretful player p is that he does not select the action a in state s at iteration t. The term $f^{\pi_t}_{-p}(s) = \sum_{h \in s} f^{\pi_t}_{-p}(h)$ is needed here to reflect the fact that the reaching of state s is also controlled by the opponent $-p$. To minimize the counterfactual regrets, a regret minimizer is utilized in each state at each iteration, generating a series of local policies $\pi_1(s), \ldots, \pi_t(s)$. Once π_t is obtained, $r^c_t(s, a)$ is generated by a model-based game tree traversing using π_t, and $R^c_t(s, a)$ is updated accordingly: $R^c_t(s, a) = R^c_{t-1}(s, a) + r^c_t(s, a)$.

Define the total regret of player p as $R^c_T = \max_{\pi'} \sum_{t=1}^{T} (u_p(\pi', \pi_{-p,t}) - u_p(\pi_{p,t}, \pi_{-p,t}))$. It is proven in Zinkevich et al. (2008) that $R^c_T \leq \sum_{s \in \mathcal{S}} [R^c_T(s)]^+$, where $[\cdot]^+ = \max\{\cdot, 0\}$ and $R^c_T(s) = \max_{a \in \mathcal{A}(s)} R^c_T(s, a)$. As a result, the total regret can be minimized by minimizing the cumulative counterfactual regrets at each state. Moreover, the average strategy $\bar{\pi}_T(a|s) = \sum_{t \in T} f^{\pi_t}_p(s)\pi_t(a|s) / \sum_{t \in T} f^{\pi_t}_p(s)$ converges to a NE if R^c_T grows sub-linearly with T (Zinkevich et al., 2008). There are two commonly used regret minimizers: Regret Matching (RM) and Hedge (Cesa-Bianchi & Lugosi, 2006). In RM, a player selects an action with probability in proportion to its

[2] We use state s and infoset I_p interchangeably in this paper.

positive cumulative counterfactual regret. In Hedge, the strategy $\pi_{t+1}(a|s)$ is decided according to:

$$\pi_{t+1}(a|s) = \frac{e^{\eta(s)R_t^c(s,a)}}{\sum_{a'} e^{\eta(s)R_t^c(s,a')}}. \tag{1}$$

If a player plays according to Hedge in state s and $\eta(s) = \sqrt{8\log(|\mathcal{A}(s)|)/(\Delta^2(s)T)}$, then $R_T^c(s) \leq \Delta(s)\sqrt{\log(|\mathcal{A}(s)|)T/2}$ (Cesa-Bianchi & Lugosi, 2006). In other words, the total regret grows sub-linearly with T, and therefore the average strategy $\bar{\pi}_T(a|s)$ converges to a NE.

3 Actor-Critic Hedge

In this section, we develop a new actor-critic algorithm, i.e., ACH, for approaching a NE in large-scale IIGs, such as 1v1 Mahjong. We make modifications within the actor-critic algorithm framework in deep RL such that ACH remains model-free and highly scalable. Key to ACH is the idea that we change the policy net from predicting action probabilities in deep RL to approximating the expectation of the sum of sampled advantages $R_t^a(s,a) := \mathbb{E}[\sum_{k=1}^t \tilde{A}^{\pi_k}(s,a)]$, where $\tilde{A}^{\pi_k}(s,a) = A^{\pi_k}(s,a)$ if s is visited at iteration k (given that only one trajectory is sampled at each iteration). Otherwise, $\tilde{A}^{\pi_k}(s,a) = 0$. In other words, the expectation $\mathbb{E}[\tilde{A}^{\pi_k}(s,a)]$ depends on both the advantage $A^{\pi_k}(s,a)$ and the sampling strategy μ_k at iteration k. The strategy π_t at each iteration in ACH is obtained via Hedge on the output of the policy net. In the next section, we will deduce that the policy target $R_t^a(s,a)$ in ACH can be viewed as a type of *weighted cumulative counterfactual regret*. As a result, ACH is theoretically connected to CFR with Hedge, more details of which will be given in the next section.

Suppose at the beginning of iteration t, $R_{t-1}^a(s,a)$ is well approximated by the output $y(a|s;\theta_{t-1})$ of the policy net in ACH, where θ_{t-1} represents the parameters of the policy net in ACH. The strategy π_t for iteration t is first obtained via Hedge, i.e., softmaxing on $\eta(s)y(a|s;\theta_{t-1})$. The reason we use Hedge instead of RM is that softmaxing is shift-invariant, which may be more robust to the function approximation error compared to the threshold operation in RM. M trajectories are then sampled into the buffer $\mathcal{B}_v$ via model-free self-play using the strategy profile $\pi_t = (\pi_{p,t}, \pi_{-p,t})$. Those samples in $\mathcal{B}_v$ are used to train the action value net by minimizing the squared loss: $\frac{1}{2}[Q(s,a;\omega) - G]^2$, where ω represents the parameters of the action value net and G is the sampled return. Afterwards, another M trajectories are sampled into the buffer $\mathcal{B}_\pi$ via model-free self-play using another sampling strategy profile $\mu_t = (\mu_{p,t}, \pi_{-p,t})$, where $\mu_{p,t}$ is the behavioral policy of player p and $\pi_{-p,t}$ is the current policy of player $-p$. We use a different behavioral policy $\mu_{p,t}$ for player p other than $\pi_{p,t}$, because $\mu_{p,t}$ can provide more flexibility for the convergence (more details are provided in the next section). The policy θ is then optimized according to the following squared loss function:

$$\mathcal{L}_\pi = \begin{cases} \sum_s \sum_a \{y(a|s;\theta) - [y(a|s;\theta_{t-1}) + \frac{1}{M}\sum_{i=1}^M \mathbb{1}_{s\in\tau_i} A^{\pi_t}(s,a)]\}^2 & \text{if } s \in \mathcal{B}_\pi, \\ \sum_s \sum_a \{y(a|s;\theta) - [y(a|s;\theta_{t-1}) + 0]\}^2 & \text{otherwise,} \end{cases} \tag{2}$$

where τ_i is the ith sampled trajectory in $\mathcal{B}_\pi$. The next iteration of ACH begins after the policy training finishes. The pseudocode of ACH is given in Algorithm 1. Note that we present ACH from the perspective of player p, and the same procedure applies to player $-p$. ACH runs concurrently for both players and synchronizes at the beginning of each iteration. Also, M trajectories are sampled according to $\mu_k = (\mu_{p,k}, \pi_{-p,k})$ at each iteration to estimate $\mathbb{E}[\tilde{A}^{\pi_k}(s,a)]$.

In order to utilize distributed computational resources, we also provide a parallel implementation of ACH by employing a framework of decoupled acting and learning, similar to IMPALA (Espeholt et al., 2018). The parallel implementation trains the network with mini batches and handles asynchronous training with the importance ratio clipping of PPO. Besides, we employ an entropy loss to encourage exploration during training and hopefully the convergence of current strategy to a NE, as done in Srinivasan et al. (2018). The pseudocode of the parallel implementation is presented in the Appendix.

4 Properties of Actor-Critic Hedge

In this section, we link ACH to a type of weighted CFR and prove the convergence of ACH to an approximate NE under certain conditions. To this end, we first deduce that the policy target in ACH is

Algorithm 1: Actor-Critic Hedge

Function ACH $(p, N, M, \theta, \omega)$ **:**

- $\omega_0 \leftarrow \omega, \theta_0 \leftarrow \theta$.
- **for** $t \leftarrow 1$ **to** N **do**
  - **# Obtain the policy**: $\pi_t \leftarrow \text{Softmax}(\eta(s)y(a|s;\theta_{t-1}))$.
  - **# Train the action value net**:
  - Reset $\mathcal{B}_v \leftarrow \emptyset$.
  - **for** $i \leftarrow 1$ **to** M **do**
    - $\tau_i \sim \text{SelfPlay}(\pi_{p,t}, \pi_{-p,t})$, $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \tau_i$
  - Train ω on the loss $\mathbb{E}_{(s,a,G)\sim\mathcal{B}_v}\{\frac{1}{2}[Q(s,a;\omega) - G]^2\}$
  - $\omega_t \leftarrow \omega$
  - **# Train the policy net**:
  - Reset $\mathcal{B}_\pi \leftarrow \emptyset$.
  - **for** $i \leftarrow 1$ **to** M **do**
    - $\tau_i \sim \text{SelfPlay}(\mu_{p,t}, \pi_{-p,t})$, $\mathcal{B}_\pi \leftarrow \mathcal{B}_\pi \cup \tau_i$
  - Estimate $A^{\pi^t}(s,a), \forall a \in \mathcal{A}(s), \forall s \in \mathcal{B}_\pi$ via π_t and $Q(s,a;\omega_t)$.
  - Sum aggregate advantage $A^{\pi^t}(s,a)$ in $\mathcal{B}_\pi$ by state s.
  - Train θ on the loss $\mathbb{E}_{s\sim\mathcal{S}}\{\mathcal{L}_\pi\}$, where $\mathcal{L}_\pi$ is defined in Equation 2.
  - $\theta_t \leftarrow \theta$.

$R_t^a(s,a)$, which is a type of weighted cumulative counterfactual regret. We then define a new variant of CFR algorithm: weighted CFR. We show that ACH is equivalent to a type of weighted CFR with Hedge under certain conditions, and the convergence property of ACH is proven accordingly.

We can rewrite the ACH policy loss (in Equation 2) at iteration t as $\mathcal{L}_\pi = \sum_{s\in\mathcal{S}}\sum_a\{y(a|s;\theta) - [y(a|s;\theta_{t-1}) + \frac{1}{M}\sum_{i=1}^M \mathbb{1}_{s\in\tau_i}A^{\pi_t}(s,a)]\}^2$. In other words, the target of $y(a|s;\theta)$ at iteration t is $y(a|s;\theta_{t-1}) + \frac{1}{M}\sum_{i=1}^M \mathbb{1}_{s\in\tau_i}A^{\pi_t}(s,a)$, whose expectation is

$$\begin{aligned} &\mathbb{E}_{\tau_{k,i}\sim(\mu_{p,k},\pi_{-p,k})}\left[\sum_{k=1}^t\sum_{i=1}^M \frac{1}{M}\mathbb{1}_{s\in\tau_{k,i}}A^{\pi_k}(s,a)\right] \\ &= \mathbb{E}_{\tau_k\sim(\mu_{p,k},\pi_{-p,k})}\left[\sum_{k=1}^t \tilde{A}^{\pi_k}(s,a)\right] = R_t^a(s,a). \end{aligned} \tag{3}$$

In the meantime, $\mathbb{E}_{\tau_k\sim(\mu_{p,k},\pi_{-p,k})}\mathbb{1}_{s\in\tau_k}A^{\pi_k}(s,a)$ is equal to $f_p^{\mu_k}(s)f_{-p}^{\pi_k}(s)A^{\pi_k}(s,a)$, where $f_p^{\mu_k}(s)f_{-p}^{\pi_k}(s)$ is the reaching probability of state s at iteration k. Therefore, $R_t^a(s,a) = \sum_{k=1}^t f_p^{\mu_k}(s)f_{-p}^{\pi_k}(s)A^{\pi_k}(s,a)$. Since $r_k^c(s,a) = f_{-p}^{\pi_k}(s)A^{\pi_k}(s,a)$ (Srinivasan et al., 2018), we have

$$R_t^a(s,a) = \sum_{k=1}^t f_p^{\mu_k}(s)r_k^c(s,a). \tag{4}$$

Hence, $R_t^a(s,a)$ is a type of weighted cumulative counterfactual regret that weighs each instantaneous regret with $f_p^{\mu_k}(s)$. Without loss of generality, we define a new variant of CFR algorithm, weighted CFR, below. Afterwards, we present Theorem 1.

Definition 1. *Weighted CFR follows the same procedure as the original CFR, except that the instantaneous regret $r_t^c(s,a)$ is weighted by some weight $w_t(s)$, $w_t(s) > 0$ and $\sum_{t=0}^\infty w_t(s) = \infty$.*

Theorem 1. *ACH is equivalent to weighted CFR with Hedge when $w_t(s) = f_p^{\mu_t}(s) > 0, \forall s \in \mathcal{S}, t > 0$, given that enough trajectories are sampled and $y(a|s;\theta_t)$ is sufficiently close to $R_t^a(s,a)$. Furthermore, if $\eta(s) = \sqrt{8\ln|\mathcal{A}(s)|/\{[w_h(s)]^2\Delta^2(s)T\}}$ and $w_t(s) \in [w_l(s), w_h(s)] \subset (0,1], t = 1,\ldots,T$, the average strategy $\overline{\pi}$ of ACH with $\overline{\pi}_p(a|s) = \sum_{t=1}^T\left[f_p^{\pi_t}(s)\pi_t(a|s)\right]/\sum_{t=1}^T f_p^{\pi_t}(s), \forall p \in \mathcal{P}$, converges to a ϵ-Nash equilibrium after T iterations, where*

$$\epsilon = |\mathcal{S}|\Delta\sqrt{\frac{2}{T}\ln|\mathcal{A}|} + 2|\mathcal{A}|\Delta\sum_{s\in\mathcal{S}}\frac{(w_h(s) - w_l(s))}{w_h(s)}. \tag{5}$$

In Theorem 1, we proved that the exploitability of ACH is bounded by ϵ when $y(a|s;\theta_t)$ is sufficiently close to $R_t^a(s,a)$. There are two terms in the bound ϵ. The first term converges to zero at the rate of $O(T^{-1/2})$. However, the second term is $O(1)$. In other words, the exploitability may not converge to zero when $T \to \infty$. Nevertheless, the second term is weighted by $\sum_{s\in\mathcal{S}}(w_h(s) - w_l(s))/w_h(s)$, which is the sum of the normalized range of the weights. As a result, it is possible to reduce the exploitability to an arbitrarily small value via tightening the range of the weights.

Corollary 1. *If the behavioral policy $\mu_{p,t}$ for each player $p \in \mathcal{P}$ is constant across iterations, and $f_p^{\mu_t}(s) > 0, \forall s \in \mathcal{S}, t > 0$, ACH is equivalent to CFR with Hedge when $y(a|s;\theta_t)$ is sufficiently close to $R_t^a(s,a)$.*

As shown in Corollary 1, when the behavioral policy is time-invariant, i.e., $w_h(s) = f_p^{\mu_t}(s) = w_l(s), \forall s \in \mathcal{S}, t > 0$, the second term in ϵ disappears, and CFR with Hedge is recovered. All the proofs are given in the Appendix.

5 Experimental Studies

We firstly introduce a 1v1 Mahjong benchmark, on which we compare the performance of ACH with related state-of-the-art methods: PPO, Regret Policy Gradient (RPG) (Srinivasan et al., 2018), and Neural Replicator Dynamics (NeuRD) (Hennes et al., 2020). PPO is an algorithm developed for single-agent environments but has demonstrated strong performance in many large-scale IIGs where multiple agents compete with each other. RPG and NeuRD are two recently proposed actor-critic model-free algorithms that aim for a NE in 2-player zero-sum IIGs. Since our goal is to approximate a NE, the standard and default performance metric, exploitability, is employed. We approximate the exploitability of an agent (i.e., solution) by training a best response against it as suggested in Timbers et al. (2020) and Steinberger et al. (2020), because traversing the full game tree to compute the exact exploitability is intractable in such a large-scale game as 1v1 Mahjong. As a complement, head-to-head performance of different methods on 1v1 Mahjong is also presented. Moreover, the agent obtained by ACH is further evaluated against practised Mahjong human players. To further validate the performance of ACH in IIGs other than 1v1 Mahjong, experimental results on three commonly used IIG benchmarks in OpenSpiel[3] (Lanctot et al., 2019) are presented. Note that results are reported for the parallel implementation of ACH, where $\eta(s)$ is set to a constant η. All the agents we evaluated below use the current policy instead of the average policy, as done in Srinivasan et al. (2018) and Hennes et al. (2020). We use TensorFlow 2.2.0 in all experiments.

5.1 A 2-Player Zero-Sum Mahjong Benchmark

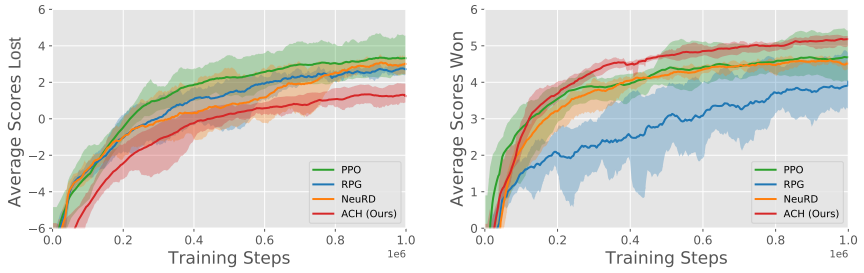
Mahjong is a tile-based game that is played world wide with many regional variations, such as Japanese Riichi Mahjong and Competition Mahjong. Like Poker, Mahjong is an IIG and is full of strategy, chance, and calculation. To facilitate Mahjong research from a game-theoretic perspective, we propose a 2-player zero-sum Mahjong benchmark[4], whose game rules are similar to Competition Mahjong. The corresponding game, "2-player Mahjong Master", is played by humans in Tencent mobile games. A full description of the game rules is in the Appendix. Apart from being the first benchmark for the 1v1 Mahjong game, our 1v1 Mahjong benchmark has a larger infoset size and a longer game length, compared with existing Poker benchmarks (Lanctot et al., 2019). The infoset size (i.e., the number of distinct histories in an infoset) in 1v1 Mahjong is around 10^{11}, compared to 10^3 in Poker. This is due to the fact that only two private cards are invisible in Poker, while there are 13 invisible tiles in 1v1 Mahjong. In addition, players can decide up to about 40 sequential actions in 1v1 Mahjong, whereas most 1v1 Poker games end within 10 steps. More details about the 1v1 Mahjong benchmark are given in the Appendix.

5.2 Results on Our 1v1 Mahjong Benchmark

All methods run in an asynchronous training platform with overall 800 CPUs, 3200 GB memory, and 8 M40 GPUs in the Ubuntu 16.04 operating system. Each method shares the same novel neural network architecture, a full description of which is given in the Appendix. We performed a mild

[3] https://github.com/deepmind/open_spiel.

[4] Our benchmark and related algorithms will be open sourced.



(a) Approximate Exploitability (b) Head-to-Head performance

Figure 1: **(a)**: The training curves of the best response against each agent. As stated in Timbers et al. (2020), the exploitability of an agent is estimated by the average scores the agent loses to its best response. **Lower is better. (b)**: The training curves of each agent. The performance of an agent is evaluated by the average scores the agent wins against a common rule-based bot. **Higher is better**. We report the mean as solid curves and the range of the average scores across 5 independent runs as shaded regions.

hyper-parameter search on PPO and shared the best setting for all methods. The advantage value is estimated by the Generalized Advantage Estimator (GAE(λ)) (Schulman et al., 2016) for all methods. An overview of the hyper-parameters is listed in the Appendix.

Approximate Exploitability. To approximate the exploitability of the agents obtained by each method, we train a best response against each agent. The agent of each method is selected at the $1e6$th training step. Note that the agent is fixed as a part of the 1v1 Mahjong environment when training the best response. We train the best response using PPO with the same hyper-parameters that were used to train the PPO agent. During the training of the best response, we evaluate the best response every 500 training steps using $10,000$ head-to-head plays against each agent. According to the average scores each agent loses to its best response in Figure 1(a), we can conclude that ACH is significantly less exploitable than other methods in the large-scale 1v1 Mahjong environment.

Head-to-Head Evaluation. We then compare the head-to-head performance of PPO, RPG, NeuRD, and ACH in the 1v1 Mahjong environment. First, we compare the training process of each method by evaluating each agent every 500 training steps against a common rule-based bot[5] using $10,000$ head-to-head plays, the results of which are shown in Figure 1(b). As we can see, all methods beat the common rule-based bot significantly, while ACH has a clear advantage over other methods in terms of stability and final performance. The relatively slow convergence of RPG may due to the threshold operation on the advantage, which could reduce sample efficiency in large-scale IIGs. Second, the superior performance of ACH is further validated in head-to-head evaluations with other agents in Table 1, where all the agents are selected at the $1e6$th training step. The agent of ACH wins all other agents by a significant margin.

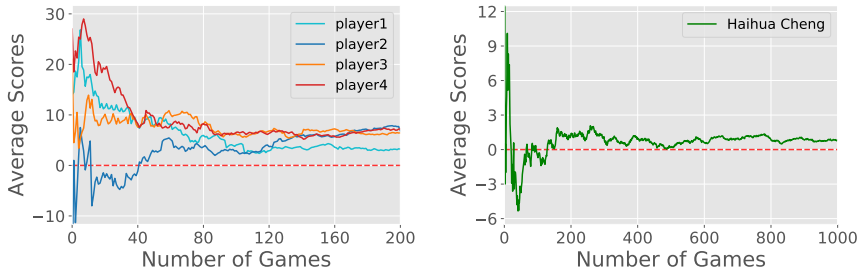
	PPO	RPG	NeuRD
RPG	**-0.21** $\pm$ 0.05	-	-
NeuRD	**-0.03** $\pm$ 0.02	**0.04** $\pm$ 0.10	-
ACH	**0.39** $\pm$ 0.02	**0.66** $\pm$ 0.05	**0.41** $\pm$ 0.07

Table 1: Mean ($\pm$ standard deviation) of the average winning scores of the row agents against the column agents. The mean and the standard deviation are estimated by 5 independent runs. In each run, the average winning scores are obtained via $10,000$ head-to-head plays.

Human Evaluation. We evaluate the agent, selected at the $1e6$th training step, of ACH against human players. First, the agent, named JueJong, is roughly evaluated by playing over $7,700$ games against 157 practiced Mahjong players, where JueJong won an average of 4.56 scores per game. Second, we select the top 4 out of 157 players according to their performances against JueJong and

[5]The rule-based bot is implemented such that it selects Hu, Ting, Kong, Chow, and Pong in descending priority whenever available and discards the tile that has the fewest neighbours.

play JueJong against the four players for 200 games each. As shown in Figure 2(a), the average winning scores of JueJong oscillate in the first 120 games but all plateau above 0 afterwards. More importantly, we evaluate JueJong against the Mahjong champion Haihua Cheng for $1,000$ games, as shown in Figure 2(b). After playing $1,000$ games, JueJong won the champion by a score of 0.82 ± 0.96 (mean $\pm$ standard deviation), with a p-value of 0.19 under one-tailed t-test. Hence, we may conclude that Haihua Cheng failed to exploit JueJong effectively within $1,000$ games.



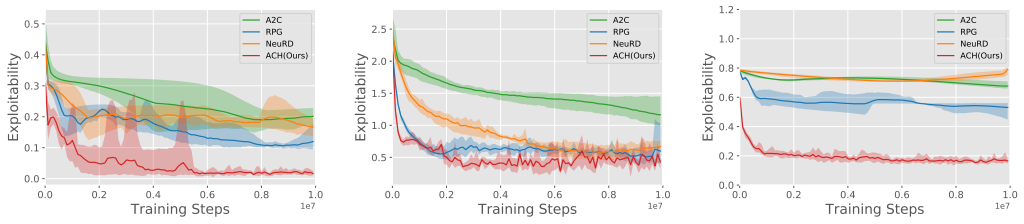
(a) Against 4 practiced Mahjong players. (b) Against the Mahjong champion.

Figure 2: Performance of JueJong against human players. The average scores JueJong won across N games against the 4 practiced players ($1 \leq N \leq 200$) and the Mahjong champion ($1 \leq N \leq 1000$) are plotted respectively.

5.3 Results on Three IIG Benchmarks in OpenSpiel

We further evaluate ACH and compare it with A2C[6], RPG, and NeuRD on three benchmarks from OpenSpiel: Kuhn poker, Leduc Poker, and Liar's Dice. All the experiments were run single-threaded on a 2.24GHz CPU. We use the network architecture in OpenSpiel, which has a 128-neurons fully-connected layer followed by ReLU and two separate linear layers for the policy and the state/action value. For A2C, RPG, and NeuRD, we use the default hyper-parameters in OpenSpiel. ACH shares most of the hyper-parameters with A2C. All the hyper-parameters are listed in the Appendix.

The exploitability of an agent is exactly calculated using tools in OpenSpiel. For each method, we compute the exploitability of each agent every $1e5$ training steps, the results of which are plotted in Figure 3. Clearly, ACH converges significantly faster and achieves a lower exploitability than other methods across the three benchmarks. There is still some gap between 0 and the exploitability ACH converges to. This may due to the neural network approximation error and the fact that we use the current strategy instead of the average strategy for the evaluation. A2C has the worst performance, since it is designed for single-agent environments. Moreover, the superiority of ACH is most significant on the Liar's Dice benchmark, which is the most complex one of the three benchmarks.



(a) Kuhn Poker (b) Leduc Poker (c) Liar's Dice

Figure 3: The training curves of each agent on the three benchmarks from OpenSpiel. We report the mean as solid curves and the range of the exploitability across 8 independent runs as shaded regions.

[6]PPO is replaced with A2C in a synchronous training environment.

6 RELATED WORK

To obviate the need of abstraction in large-scale IIGs, various neural forms of CFR methods have been developed. An early work of this direction is regression CFR (Waugh et al., 2015), which calculates weights for a number of hand-crafted features to approximate the regrets. Deep CFR (Brown et al., 2019) is similar to regression CFR but employs a neural network to approximate the regrets. Also, deep CFR traverses a part of the game tree using external sampling, in comparison to the full traversal of the game tree in regression CFR. Double neural CFR (Li et al., 2020a) is another method that approximates the regrets and the average policy using deep neural networks. The learned cumulative target in Li et al. (2020a) is scaled by the inverse of sampling probability, which varies significantly and may cause problems in games with long episodes. Instead, the advantage, which tends to 0, is accumulated and learned in ACH. All the above methods are model-based in the sense that multiple if not all actions are sampled in a state. For this reason, we do not include these methods for comparison with ACH, which is model-free and uses only trajectory samples to learn.

There are also some model-free neural forms of CFR methods: DREAM (Steinberger et al., 2020) and ARMAC (Gruslys et al., 2020), both of which may be memory or computation intensive for large-scale IIGs. DREAM needs to store self-play samples at each iteration. Although a reservoir buffer can alleviate the problem, this may lower the performance when the reservoir buffer size is not matched with the game size. ARMAC stores strategies at previous iterations instead. As a result, the current policy needs to play against previous strategies at each iteration to generate samples for learning, which may be computationally expensive in large-scale IIGs. Other model-free neural extensions, which are inspired by game-theoretic approaches other than CFR, include neural fictitious self-play (Heinrich & Silver, 2016), policy space response oracles (Lanctot et al., 2017), and exploitability descent (Lockhart et al., 2019), all of which require to approximate the average strategy and train a best response against it at each iteration. In summary, all the above methods are either memory-intensive or computation-intensive at each iteration, which may not be efficiently scalable. For this reason, we do not include these methods for comparison with ACH, which uses only samples at the current iteration (i.e., without saving samples from previous iterations) and requires no computation of a best response at each iteration.

The most related methods to ACH are RPG (Srinivasan et al., 2018) and NeuRD (Hennes et al., 2020), both of which employ the actor-critic framework and thus have similar computation and memory complexities as ACH. RPG minimizes a loss that is an upper bound on the regret after threshold, and the corresponding policy gradient is $\nabla_\theta^{RPG}(s) = -\sum_a \nabla_\theta [Q(s,a;w) - \sum_b \pi(b|s;\theta) Q(s,b;w)]^+$. However, RPG requires a l_2 projection after every gradient step for the convergence to a NE, while such projection is not required in ACH. NeuRD (Hennes et al., 2020) is inspired by the replicator dynamics, a well studied model in evolutionary game theory (Gatti et al., 2013). The policy gradient in NeuRD is $\nabla_\theta^{NeuRD}(s) = \sum_a [\nabla_\theta y(a|s;\theta)][Q(s,a;w) - \sum_b \pi(b|s) Q(s,b;w)]$, where $y(a|s;\theta)$ is the output of the policy net. There are important differences between ACH and NeuRD in how the algorithm is motivated and how the policy net at each iteration is optimized. Also, the convergence analysis was given only for the single-state, all-actions, tabular NeuRD (Hennes et al., 2020), while we link ACH to a type of weighted CFR with Hedge and present its convergence analysis in full extensive-form games.

7 CONCLUSIONS

In this paper, we investigated the problem of adapting policy gradient methods in deep RL to solve a large-scale IIG, i.e., 1v1 Mahjong. To this end, we developed a new model-free actor-critic algorithm, i.e, ACH, for approximating a NE in large-scale IIGs. ACH is memory and computation efficient, as it uses only trajectory samples at the current iteration and requires no computation of best response. As a result, ACH is efficiently scalable. We proved the convergence of ACH to an approximate NE in 2-player zero-sum IIGs under certain conditions, by linking ACH to a type of weighted CFR with Hedge. The superior performance of ACH was validated on both our 1v1 Mahjong benchmark and benchmarks in OpenSpiel. Secondly, to facilitate research on large-scale IIGs, we proposed the first 1v1 zero-sum Mahjong benchmark, whose infoset size and game length are much larger than Poker. Finally, based on the novel neural architecture design and ACH, we obtained the 1v1 Mahjong bot JueJong, which has demonstrated stronger performance against the Mahjong champion Haihua Cheng.

REPRODUCIBILITY STATEMENT

The experiments on the 1v1 Mahjong benchmark were run in a large cluster of thousands of machines, on which we have developed an efficient actor-learner training platform, similar to IMPALA (Espeholt et al., 2018). The codes of both the platform and the 1v1 Mahjong benchmark are not released currently but are planned to be open sourced in the near future. Yet, the codes for both the actor part and the learner part of each algorithm (PPO, RPG, NeuRD, and ACH) are included in the supplementary materials. All the hyper-parameters are listed in the Appendix.

The experiments on the benchmarks in OpenSpiel were run using the OpenSpiel environment. We include the necessary codes, rand seeds, and instructions to reproduce all the results on OpenSpiel in the supplementary materials. All the hyper-parameters are listed in the Appendix.

All the theoretical results with regard to ACH are presented in the main text, with all the proofs included in the Appendix.

REFERENCES

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, and et al. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International Conference on Machine Learning (ICML)*, pp. 793–802, 2019.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge university press, 2006.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*, volume 80, pp. 1406–1415, 2018.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

Nicola Gatti, Fabio Panozzo, and Marcello Restelli. Efficient evolutionary dynamics with extensive-form games. In *AAAI Conference on Artificial Intelligence*, 2013.

Audrunas Gruslys, Marc Lanctot, Rémi Munos, Finbarr Timbers, Martin Schmid, Julien Pérolat, Dustin Morrill, Vinícius Flores Zambaldi, Jean-Baptiste Lespiau, John Schultz, Mohammad Gheshlaghi Azar, Michael Bowling, and Karl Tuyls. The advantage regret-matching actor-critic. *CoRR*, abs/2008.12234, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

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

Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Rémi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duéñez-Guzmán, et al. Neural replicator dynamics: Multiagent learning via hedging policy gradients. In *International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*, pp. 492–501, 2020.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

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

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4190–4203, 2017.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinícius Flores Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, and et al. Openspiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019.

Hui Li, Kailiang Hu, Shaohua Zhang, Yuan Qi, and Le Song. Double neural counterfactual regret minimization. In *International Conference on Learning Representations (ICLR)*, 2020a.

Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *CoRR*, abs/2003.13590, 2020b.

Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 464–470, 2019.

Naoki Mizukami and Yoshimasa Tsuruoka. Building a computer mahjong player based on monte carlo simulation and opponent models. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 275–283, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1928–1937, 2016.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations, (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3422–3435, 2018.

Eric Steinberger, Adam Lerer, and Noam Brown. DREAM: deep regret minimization with advantage baselines and model-free learning. *CoRR*, abs/2006.10410, 2020.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1057–1063, 1999.

Finbarr Timbers, Edward Lockhart, Martin Schmid, Marc Lanctot, and Michael Bowling. Approximate exploitability: Learning a best response in large games. *CoRR*, abs/2004.09677, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, and et al. John P. Agapiou and. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, 2019.

Kevin Waugh, Dustin Morrill, James Andrew Bagnell, and Michael H. Bowling. Solving games with functional regret estimation. In *AAAI Conference on Artificial Intelligence*, pp. 2138–2145, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, Yinyuting Yin, Bei Shi, Liang Wang, Tengfei Shi, Qiang Fu, Wei Yang, Lanxiao Huang, and Wei Liu. Towards playing full MOBA games with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1729–1736, 2008.

A INTRODUCTION OF THE 1V1 MAHJONG GAME

Mahjong is a tile-based game that is played world wide with many regional variations, such as Japanese Riichi Mahjong and Competition Mahjong. Like Poker, Mahjong is an IIG and is full of strategy, chance, and calculation. In this paper, we investigate a 1v1 Mahjong game, whose information set size is much larger than Poker, as shown in Figure 4. The game rules of 1v1 Mahjong are similar to Competition Mahjong. The corresponding game, “2-player Mahjong Master", is played by humans in Tencent mobile games.

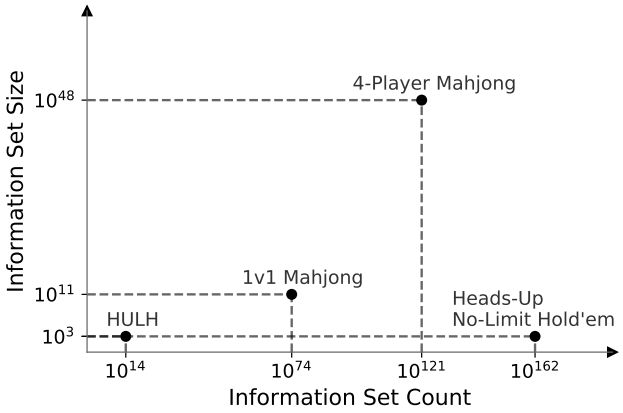


Figure 4: The game complexity of Heads-Up Limit Texas Hold’em (HULH), Heads-Up No-Limit Texas Hold’em, 1v1 Mahjong, and 4-Player Mahjong.

A.1 THE GAME RULES

There are 24 unique tiles and 72 tiles in total in the 1v1 Mahjong game, as shown in Figure 5. At the beginning of the 1v1 Mahjong game, each player is dealt with 13 tiles, the content of which is invisible to the other. Afterwards, each player takes actions in turn. Typically, the first player draws a tile from the deck wall and then discards a tile, and the next player takes the same types of actions in sequence. There are exceptional cases where the player does not draw a tile from the deck wall but Chow, Pong, or Kong the tile the opponent just discarded. Afterwards, the player discards a tile, and the game proceeds. Also, after drawing a tile from the deck wall, there are cases where the player could Kong, after which the player draws and discards a tile in sequence. There are 10 types of actions with 105 different actions in total, a full description of which is listed in Table 2.

Category	Name	Type	Copy	Count
Simples	Characters		4	36
Honors	Winds		4	16
Honors	Dragons		4	12
Bonus	Flowers		1	4
Bonus	Seasons		1	4

Figure 5: A list of all tiles in the 1v1 Mahjong game.

Type	**Description**	**#Action**
Discard	Discard one of the tiles in the hand.	16
Pong	Make a meld of 3 identical tiles by seizing the opponent's latest discarded tile.	16
Concealed-Kong	Make a meld of 4 identical tiles when they are in the hand, and the Concealed-Kong is not revealed to the opponent.	16
Kong	Make a meld of 4 identical tiles by seizing the opponent's latest discarded tile.	16
Add-Kong	Make a meld of 4 identical tiles by adding a tile to his own exposed Pong.	16
Chow	Make a meld of 3 character tiles in a row by seizing the opponent's latest discarded tile.	21
Draw	Give up the action of Pong, Chow, or Kong, and draw a tile from the deck wall.	1
Ting	When there is only one tile away from a legal hand, the player can declare Ting.	1
Hu	Form a legal hand by drawing a tile or seizing the opponent's latest discarded tile. The Hu action ends the game.	1
Pass-Hu	Give up the action of Hu. Afterwards, the player should select another legal action.	1

Table 2: Different actions in the 1v1 Mahjong game.

The goal of each player is to complete a legal hand prior to the opponent, by drawing a tile or using the tile the opponent just discarded. A legal hand is generally in the form of four melds and a pair, with an exception of 7 pairs. Different categories of legal hands, 64 in total, come with different points, which is fully described in Table 3. Besides, a legal hand can belong to multiple categories, and the score of a legal hand is the sum of points of corresponding categories. Legal hands with higher points are generally more difficult to form in terms of either luck or strategy, and it is critical for a player to trade off the winning probability and the corresponding winning points. If no player completes a legal hand before the tiles are exhausted, the game is tied. A flow chart of the game is illustrated in Figure 6.

	English name	**Points**	**Pattern descriptions**	**Conflicts**
1	Pure Double Chow	1	Two identical sequences of 3 character tiles.	
2	Short Straight	1	A hand with two successive sequences.	

3	Two Terminal Chows	1	A hand with two sequences 123 and 789.	
4	Melded Kong	1	A hand with exposed quads.	
5	Edge Wait	1	A hand completion with the situation there is only one tile name to complete because you have the incomplete sequence in edge (12 or 89).	
6	Closed Wait	1	A hand completion with the situation there is only one tile name to complete because you have the incomplete sequence lacking center (like 24 and 79).	
7	Single Wait	1	A hand completion solely waiting on a tile to form a pair.	
8	Self Draw	1	A hand completion by draw.	
9	Flower Tile	1	Whenever a player draws a flower or season tile, this is counted as 1 point.	
10	Ready Hand	2	A hand that is one tile away from winning.	
11	Dragon Pung	2	A hand with a triplet or quad of dragon tile.	
12	Concealed Hand	2	A hand without exposed melds and completed by seizing the discarded tile.	
13	All Chows	2	A hand with four sequences and a pair in characters.	
14	Tile Hog	2	A hand with four same tiles, other than a melded quad.	
15	Two Concealed Pungs	2	A hand with two concealed triplets or quads.	
16	Concealed Kong	2	A hand with a concealed quad.	
17	All Simples	2	A hand consisting of no character 1, 9 and honor tile.	
18	Outside Hand	4	A hand that includes terminals and honors in each meld, including the pair.	
19	Fully Concealed Hand	4	A hand without exposed melds and completed by drawing a tile.	Self Draw
20	Two Melded Kongs	4	A hand with two exposed quads.	Melded Kong
21	Last Tile	4	Winning on a tile that is the last of its kind. Three of the other tiles on the table already revealed to all players.	
22	Little Three Winds	6	A hand with two wind triplets or quads and a wind pair.	
23	All Pungs	6	A hand with four triplets or quads.	
24	Half Flush	6	A hand consisting of character and honor tiles.	
25	Two Dragon Pungs	6	A hand containing two dragon triplets or quads.	Dragon Pung
26	Two Concealed Kongs	6	A hand containing two concealed quads.	Concealed Kong
27	Melded Hand	6	Every set in the hand must be completed with tiles discarded by other players.	Single Wait
28	Out with Replacement Tile	8	Hand completion with supplemental tile when you melding quad.	Self Draw

29	Rob Kong	8	Winning off the tile that opponent adds to a melded triplet(to create a Kong).	Last Tile
30	Last Tile Claim	8	Winning off another player on the last tile (of the game).	
31	Pure Straight	16	A hand with three sequences 123, 456 and 789 in characters.	Short Straight, Two Terminal Chows
32	Pure Shifted Chows	16	Three sequences in characters each shifted either one or two numbers up from the last, but not a combination of both.	
33	All Flowers	16	A player has or draws all flower tiles.	Flower Tiles
34	Full Flush	16	A hand consisting of only characters.	
35	Three Concealed Pungs	16	A hand with three concealed quads or triplets.	Two Concealed Pungs
36	Four Honour Pungs	24	A hand with four honor triplets or quads.	All Pungs
37	Big Three Winds	24	A hand with three winds be triplets or quads.	Little Three Winds
38	Seven Pairs	24	A hand with seven pairs.	Concealed Hand, Single Wait, Fully Concealed Hand.
39	Pure Triple Chow	24	A hand with three identical sequences.	Pure Double Chow
40	Pure Shifted Pungs	24	A hand with three number triplets or quads with successive numbers.	Pure Triple Chow
41	Four Pure Shifted Chows	32	Four shifted number sequences, each shifted either one or two numbers up from the last, but not a combination of both.	Pure Shifted Chows, Short Straight, Two Terminal Chows, Pure Double Chow
42	Three Kongs	32	A hand with three quad melds.	Two Melded Kongs, Melded Kong, Two Concealed Kongs, Concealed Kong
43	All Terminals and Honours	32	A hand consisting of only terminal and honor tiles.	All Pungs, Outside Hand
44	Heavenly Ready Hand	32	A hand that is one tile away from winning at the beginning of the game.	Ready Hand
45	Quadruple Chow	48	A hand with four identical sequences.	Pure Triple Chow, Tile Hog, Pure Double Chow, Two Terminal Chow, Pure Shifted Pungs

46	Four Pure Shifted Pungs	48	A hand with four character triplets or quads with successive numbers.	Pure Shifted Pungs, Pure Triple Chow, All Pungs
47	Four Winds Seven Pairs	48	Seven pairs hand with four different wind pairs.	Seven Pairs, Concealed Hand, Single Wait, Fully Concealed Hand
48	Three Dragons Seven Pairs	48	Seven pairs hand with three different dragon pairs.	Seven Pairs, Concealed Hand, Single Wait, Fully Concealed Hand
49	Little Four Winds	64	A hand with three winds be triplets or quads, and the last wind be pair.	Big Three Winds, Little Three Winds
50	Little Three Dragons	64	A hand with two dragons be triplets or quads, and the last dragon be pair.	Two Dragon Pungs, Dragon Pung
51	All Honours	64	A hand consisting of only honor tiles.	All Terminals And Honours, All Pungs, Four Honour Pungs, Outside Hand
52	Four Concealed Pungs	64	A hand with four concealed triplets or quads.	Concealed Hand, All Pungs, Three Concealed Pungs, Two Concealed Pungs, Fully Concealed Hand
53	Pure Terminal Chows	64	A hand with two Two Terminal Chows and a pair of number 5 in character.	All Chows, Seven Pairs, Full Flush, Pure Double Chow, Two Terminal Chows
54	Big Four Winds	88	A hand with triplets or quads of all four winds and an pair.	All Pungs, Little Three Winds, Big Three Winds, Four Honor Pungs
55	Big Three Dragons	88	A hand with triplets or quads of all three dragons.	Dragon Pung, Two Dragon Pungs

56	Nine Gates	88	Collecting number tiles 1112345678999 without melding, and completing with any tile of characters.	Full Flush, Concealed Hand, Fully Concealed Hand
57	Four Kongs	88	A hand with four quad melds.	Three Kongs, Two Melded Kongs, Melded Kong, Single Wait, Concealed Kong, Two Concealed Kongs, All Pungs
58	Seven Shifted Pairs	88	Seven pairs hand with successive seven numbers in characters.	Seven Pairs, Single Wait, Concealed Hand, Full Flush, Fully Concealed Hand
59	Upper Four	88	A hand consisting of character tiles of 6, 7, 8 or 9	Seven Pairs
60	Lower Four	88	A hand consisting of character tiles of 1, 2, 3 or 4	Seven Pairs
61	Big Seven Honours	88	Seven pairs hand with four different wind pairs and three different dragon pairs.	Seven Pairs, Four Winds Seven Pairs, Three Dragons Seven Pairs, Outside Hand, Single Wait, Concealed Hand, Fully Concealed Hand, All Honours
62	Heavenly Hand	88	The dealer draws a winning hand at the beginning of the game.	Self Draw, Concealed Hand
63	Earthly Hand	88	A player completes a winning hand with the dealer's first discard and in most variants, provided the dealer does not draw a quad.	Self Draw, Concealed Hand
64	Humanly Hand	88	A player completes a winning hand with the opponent player's first discard. And before that any action of Chow, Pong or Kong is not available.	

Table 3: The categories of legal hands in ascending order of corresponding points. The winning points of a legal hand is calculated by summing the points of the matched categories in reverse order while excluding the conflict categories.

A.2 The State and Action Space

The state space size of 1v1 Mahjong, shown as the information set count in Figure 4, is approximately 10^{74}. Yet, the state space of 1v1 Mahjong is not as easily abstracted as in Poker. The primary reason

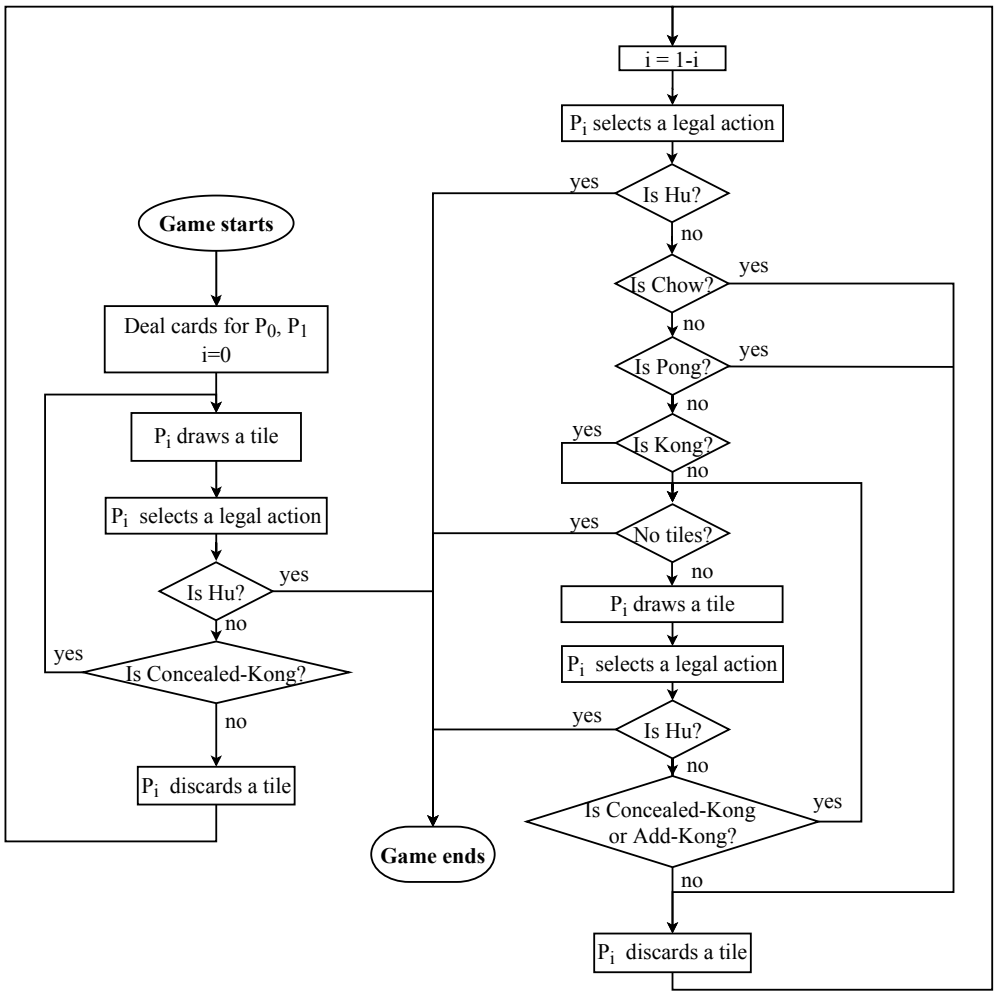


Figure 6: A flow chart of the 1v1 Mahjong game.

is that a single tile difference in the state could significantly impact the strategy, e.g., making a legal hand illegal and vice versa. In contrast, states that have similar strength in Poker could share a common strategy. For instance, the optimal preflop strategy could be very similar for “Ace-Four" and “Ace-Three" in Poker. Another reason is that a state in 1v1 Mahjong is divided into different information groups, as demonstrated in Figure 7(a). Different information groups have significantly different meanings. For instance, one group denotes the player’s hand, which is invisible to the opponent, while another group denotes the player’s discarded tiles, which are visible to both players.

There are 105 different actions in total, as demonstrated in Table 2. The number of legal actions in a state is relatively small compared to Poker. Yet, the game length in 1v1 Mahjong is larger than that in Poker. Players can decide up to about 40 sequential actions in 1v1 Mahjong, whereas most 1v1 Poker games end within 10 steps. As a result, the reaching probability of states in 1v1 Mahjong may vary more significantly than that in Poker.

B THE MODEL DESIGN OF OUR 1V1 MAHJONG BOT JUEJONG

The model of JueJong is an end-to-end neural network that takes all relevant information as input and outputs both the probabilities of all actions and the state value. This is different from five separated neural networks in Suphx (Li et al., 2020b), representing five different types of actions. Also, we train JueJong from zero by pure self-play using ACH, while the five neural networks in Suphx were

Figure 7: **(a)** The graphical user interface of 1v1 Mahjong with marked regions representing different groups of information, which are encoded in either image-like features or one-hot features. **(b)** The image-like feature encoding scheme. **(c)** The model architecture design.

trained by supervised learning on human data, with only the "Discard" network further enhanced by RL. Figure 7 gives an overview of the model design of JueJong.

The marked regions in Figure 7(a) summarize the information an agent can observe. Regions 1 to 7 are encoded in image-like features, which represent the player's hand, the player's Chow, Pong, and Kong, the player's concealed-Kong, the player's Discard, the opponent's Chow, Pong, and Kong, the opponent's concealed-Kong, and the opponent's Discard respectively. A feature map, as shown in Figure 7(b), of height 4 and width 17 is employed, where a 0 or 1 in the ith row and jth column means whether there are i tiles of the jth type for the input set of tiles. Therefore, a single or two successive convolutional layers with 3×3 kernels efficiently derive row and column combinations. Note that, in Suphx (Li et al., 2020b), the feature maps are one-dimensional vectors, where 1×3 kernels are used. The black tile in Figure 7(b) is designed exclusively for Region 6 to indicate how many concealed-Kongs the opponent has. For discarded tiles in Region 4 or 7, 24 such feature maps are used to indicate the latest 24 discarded tiles in order, with one feature map encoding one discard tile. All the feature maps are concatenated in the channel dimension, and therefore the final image-like features are in the shape of $4 \times 17 \times 53$ ($h \times w \times c$).

Regions 8 to 11 are encoded in one-hot features, which represent the player's position (0 or 1), the is_ready state of both players, and the bonus tiles of both players. Additionally, the last action (not shown as a region in Figure 7(a)) for each player is encoded in a one-hot feature vector as well.

We apply residual blocks (He et al., 2016) to transform the image-like features. There are totally 3 stages with 3 residual blocks and 1 transition layer in each stage, as shown in Figure 7(c). Each block contains two convolutional layers with kernel size 3×3 and number of output feature maps 32, 64, and 128 for each stage. The transition layer is point-wise convolution 1×1, and the one in the final stage scales the input to 32 channels. Subsequently, the transformed image-like features are reshaped to a vector, which is concatenated with the one-hot feature vectors. A fully-connected layer (dimension 1024) is then used to transform the concatenated feature vector, and two branches with two fully-connected layers (dimension 512×512) each output the action probabilities and the state value respectively. Besides, we apply batch normalization (Ioffe & Szegedy, 2015) and ReLU non-linearity after all convolutional layers and fully-connected layers.

C THEORETICAL PROPERTIES OF ACH

C.1 PROOF FOR THEOREM 1

In order to prove Theorem 1, we first prove the following Lemma.

Lemma 1. *For a weighted CFR, if* $w_t(s) \in [w_l(s), w_h(s)], 0 < w_l(s) \leq w_h(s) \leq 1$*, then*

$$\max_{a \in \mathcal{A}(s)} \sum_{k=1}^{t} r_k^c(s,a) \leq \frac{\max_{a \in \mathcal{A}(s)} R_t^w(s,a)}{w_h(s)} + \frac{(w_h(s) - w_l(s))|\mathcal{A}(s)|\Delta t}{w_h(s)}. \tag{6}$$

Proof. First, for any state $s \in S$ and action $a \in \mathcal{A}(s)$, we have

$$\begin{aligned} R_t^w(s,a) :=& \sum_{k=1}^{t} w_t(s) r_k^c(s,a) \\ =& \sum_{k: r_k^c(s,a) \geq 0} w_k(s)|r_k^c(s,a)| - \sum_{k': r_{k'}^c(s,a) < 0} w_{k'}(s)|r_{k'}^c(s,a)| \\ \geq& \sum_{k: r_k^c(s,a) \geq 0} w_l(s)|r_k^c(s,a)| - \sum_{k': r_{k'}^c(s,a) < 0} w_h(s)|r_{k'}^c(s,a)|. \end{aligned} \tag{7}$$

So,

$$R_t^w(s,a) \geq w_h(s) \sum_{k=1}^{t} r_k^c(s,a) - (w_h(s) - w_l(s)) \sum_{k': r_{k'}^c(s,a) \geq 0} |r_{k'}^c(s,a)|. \tag{8}$$

In other words,

$$\sum_{k=1}^{t} r_k^c(s,a) \leq \frac{R_t^w(s,a)}{w_h(s)} + \frac{(w_h(s) - w_l(s))}{w_h(s)} \sum_{k': r_{k'}^c(s,a) \geq 0} |r_{k'}^c(s,a)| \tag{9}$$

So,

$$\max_{a \in \mathcal{A}(s)} \sum_{k=1}^{t} r_k^c(s,a) \leq \frac{\max_{a \in \mathcal{A}(s)} R_t^w(s,a)}{w_h(s)} + \frac{(w_h(s) - w_l(s))|\mathcal{A}(s)|\Delta t}{w_h(s)}. \tag{10}$$

□

Then we can prove the Theorem:

Proof. For ACH, the policy at iteration t is generated according to Hedge:

$$\pi_t(s,a) = \frac{e^{\eta(s) R_{t-1}^a(s,a)}}{\sum_{a'} e^{\eta(s) R_{t-1}^a(s,a')}}, \tag{11}$$

where

$$R_{t-1}^a(s,a) = \sum_{k=1}^{t-1} f_p^{\mu_k}(s) r_k^c(s,a). \tag{12}$$

On the other hand, weighted CFR with Hedge generates policy at iteration t according to

$$\pi_t(s,a) = \frac{e^{\eta(s) R_{t-1}^w(s,a)}}{\sum_{a'} e^{\eta(s) R_{t-1}^w(s,a')}}, \tag{13}$$

where

$$R_{t-1}^w(s,a) = \sum_{k=1}^{t-1} w_k(s) r_k^c(s,a). \tag{14}$$

So, when $w_k(s) = f_p^{\mu_k}(s) \geq w_l(s) > 0$ for any $s \in \mathcal{S}$ and $k > 0$, ACH and weighted CFR with Hedge generates the same policy at each iteration, and therefore ACH is equivalent to weighted CFR with Hedge.

Then, we can prove the convergence property of ACH. Let

$$S_t^w(s) = \sum_{a \in \mathcal{A}(s)} e^{\eta(s) R_t^w(s,a)}. \tag{15}$$

We have

$$\begin{aligned} \ln \frac{S_t^w}{S_0^w} =& \ln \sum_{a \in \mathcal{A}(s)} e^{\eta(s) R_t^w(s,a)} - \ln |\mathcal{A}(s)| \\ \geq& \ln \left(\max_{a \in \mathcal{A}(s)} e^{\eta(s) R_t^w(s,a)} \right) - \ln |\mathcal{A}(s)| \\ =& \eta(s) \max_{a \in \mathcal{A}(s)} R_t^w(s,a) - \ln |\mathcal{A}(s)|. \end{aligned} \tag{16}$$

On the other hand, for each $k = 1, \ldots, t$,

$$\begin{aligned} \ln \frac{S_k^w}{S_{k-1}^w} =& \ln \left(\frac{\sum_{a \in \mathcal{A}(s)} e^{\eta(s) R_{k-1}^w(s,a)} e^{\eta(s) w_k(s) r_k^c(s,a)}}{\sum_{a \in \mathcal{A}(s)} e^{\eta(s) R_{k-1}^w(s,a)}} \right) \\ =& \ln \left(\sum_{a \in \mathcal{A}(s)} \pi_k(s,a) e^{\eta(s) w_k(s) r_k^c(s,a)} \right). \end{aligned} \tag{17}$$

Since

$$\ln \mathbb{E}[e^{sX}] \leq s\mathbb{E}X + \frac{s^2(b-a)^2}{8}, \tag{18}$$

$$\ln \frac{S_k^w}{S_{k-1}^w} \leq \eta(s) \sum_{a \in \mathcal{A}(s)} \pi_k(s,a) w_k(s) r_k^c(s,a) + \frac{\eta^2(s) \Delta^2(s) w_k^2(s)}{8}. \tag{19}$$

Note that $r_k^c(s,a) = v_k^c(s,a) - \sum_{a \in \mathcal{A}(s)} \pi_k(s,a) v_k^c(s,a)$. So

$$\sum_{a \in \mathcal{A}(s)} \pi_k(s,a) w_k(s) r_k^c(s,a) = 0. \tag{20}$$

Therefore,

$$\ln \frac{S_k^w}{S_0^w} \leq \frac{\eta^2(s) \Delta^2(s) \sum_{k=1}^t w_k^2(s)}{8}. \tag{21}$$

So,

$$\eta(s) \max_{a \in \mathcal{A}(s)} R_t^w(s,a) - \ln |\mathcal{A}(s)| \leq \frac{\eta^2(s) \Delta^2(s) \sum_{k=1}^t w_k^2(s)}{8}, \tag{22}$$

i.e.,

$$\max_{a \in \mathcal{A}(s)} R_t^w(s,a) \leq \frac{\ln |\mathcal{A}(s)|}{\eta(s)} + \frac{\eta(s) \Delta^2(s) \sum_{k=1}^t w_k^2(s)}{8}. \tag{23}$$

According to Lemma 1,

$$\max_{a \in \mathcal{A}(s)} R_t^c(s,a) \leq \frac{\ln |\mathcal{A}(s)|}{\eta(s) w_h(s)} + \frac{\eta(s) \Delta^2(s) \sum_{k=1}^t w_k^2(s)}{8 w_h(s)} + \frac{(w_h(s) - w_l(s)) |\mathcal{A}(s)| \Delta t}{w_h(s)}. \tag{24}$$

When $\eta(s) = \sqrt{8 \ln |\mathcal{A}(s)| / \{[w_h(s)]^2 \Delta^2(s) T\}}$ and $w_t(s) \in [w_l(s), w_h(s)] \subset (0, 1], t = 1, \ldots, T$, we have

$$\begin{aligned} \max_{a \in \mathcal{A}(s)} R_T^c(s,a) \leq& \sqrt{\frac{\ln |\mathcal{A}(s)| \Delta^2(s) w_h^2(s) T}{2 w_h^2(s)}} + \frac{(w_h(s) - w_l(s)) |\mathcal{A}(s)| \Delta T}{w_h(s)} \\ \leq& \Delta \sqrt{\frac{T}{2} \ln |\mathcal{A}(s)|} + \frac{(w_h(s) - w_l(s)) |\mathcal{A}(s)| \Delta T}{w_h(s)}. \end{aligned} \tag{25}$$

According to Theorem 2 in Zinkevich et al. (2008), the total regret

$$
\begin{aligned}
R_T \leq & \sum_{s \in S} \max_{a \in \mathcal{A}(s)} [R_T^c(s,a)]^+ \\
\leq & \sum_{s \in S} \left(\Delta \sqrt{\frac{T}{2} \ln |\mathcal{A}(s)|} + \frac{(w_h(s) - w_l(s))|\mathcal{A}(s)|\Delta T}{w_h(s)} \right) \\
\leq & |\mathcal{S}|\Delta \sqrt{\frac{T}{2} \ln |\mathcal{A}|} + |\mathcal{A}|\Delta T \sum_{s \in \mathcal{S}} \frac{(w_h(s) - w_l(s))}{w_h(s)}.
\end{aligned} \tag{26}
$$

As a result, according to the folk theorem in Zinkevich et al. (2008), the average strategy converges to an ϵ-equilibrium, where

$$
\epsilon = \frac{2R_T}{T} = |\mathcal{S}|\Delta \sqrt{\frac{2}{T} \ln |\mathcal{A}|} + 2|\mathcal{A}|\Delta \sum_{s \in \mathcal{S}} \frac{(w_h(s) - w_l(s))}{w_h(s)}. \tag{27}
$$

□

C.2 Proof for Corollary 1

Proof. When the behavioral policy $\mu_{p,k}$ of each player $p \in \mathcal{P}$ across iterations $k > 0$ is constant, the reaching probability $f_p^{\mu_k}(s)$ of any state $s \in \mathcal{S}$ is also constant. Assume $f_p^{\mu_k}(s) = w(s)$, then,

$$
R_{t-1}^a(s,a) = \sum_{k=1}^{t-1} f_p^{\mu_k}(s) r_t^c(s,a) = w(s) \sum_{k=1}^{t-1} r_t^c(s,a) = w(s) R_{t-1}^c(s,a). \tag{28}
$$

In other words, $R_{t-1}^a(s,a)$ is equal to the cumulative counterfactual regret scaled by a time-invariant weight $w(s)$. Hence, the policy at iteration t is

$$
\pi_t(a|s) = \frac{e^{\eta(s)w(s)R_t^c(s,a)}}{\sum_{a'} e^{\eta(s)w(s)R_{t-1}^c(s,a')}} = \frac{e^{\eta'(s)R_t^c(s,a)}}{\sum_{a'} e^{\eta'(s)R_{t-1}^c(s,a')}}, \tag{29}
$$

where the new $\eta'(s)$ is set to $\sqrt{8 \ln |\mathcal{A}(s)| / [\Delta^2(s)T]}$. So, when $\mu_{p,k}$ is constant, ACH is equivalent to CFR with Hedge when $y(a|s;\theta_t)$ is sufficiently close to $R_t^a(s,a)$. Furthermore, since $w_l(s) = f_p^{\mu_k}(s) = w_h(s)$, the second term in (27) disappears, i.e.,

$$
\epsilon = |\mathcal{S}|\Delta \sqrt{\frac{2}{T} \ln |\mathcal{A}|}. \tag{30}
$$

As a result, the exploitability bound of CFR with Hedge is recovered. □

C.3 Continuous-Time Dynamics of ACH

We illustrate the convergence dynamics of ACH in the game of Rock-Paper-Scissors, in comparison with other actor-critic methods (PPO/A2C, RPG, and NeuRD), in Figure 8.

D A Parallel Implementation of ACH

In order to utilize distributed clusters, we provide a parallel implementation of ACH, which employs a framework of decoupled acting and learning, similar to IMPALA (Espeholt et al., 2018). The parallel implementation of ACH maintains a policy net $y(a|s;\theta)$ and a value net $V(s;\omega)$, where θ and ω share a large portion of parameters (see Figure 7). Both players use the same θ and ω. We do not use an additional time-invariant behavioral policy for sampling actions. Instead, we use the current policy π_t. As a result, we can use the same samples to train both the value net and the policy net. $\eta(s)$ is incorporated into the learned target value in the policy net, so the policy $\pi(a|s)$ is obtained by directly softmaxing on $y(a|s;\theta)$.

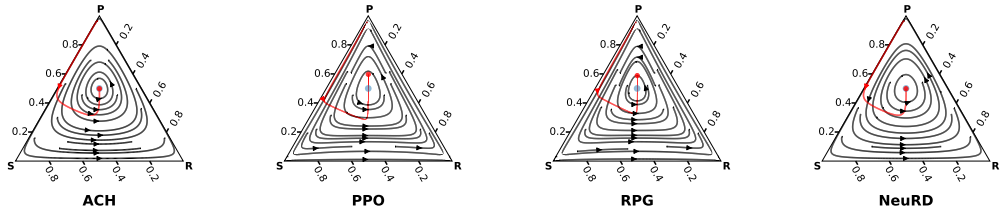


Figure 8: Continuous-time dynamics (black lines) and time-averaged convergence trajectories (red lines) of different methods in a biased version of Rock-Paper-Scissors, with the NE denoted as the blue dot.

The advantage $A(s, a)$ is estimated by the GAE(λ) (Schulman et al., 2016), using sampled rewards and $V(s; \omega)$, for only sampled states and actions. The value and policy nets are updated as soon as a mini batch of samples is available. In other words, we update θ and ω once using a single mini batch at each iteration. As a result, the policy loss reduces to $\mathcal{L}_\pi = \eta(s)\frac{y(a|s;\theta)}{\pi_{old}(a|s)}A(a, s)$, where $\frac{1}{\pi_{old}(a|s)}$ accounts for the fact that a was sampled using $\pi_{old}(a|s)$. The parallel implementation handles asynchronous training with the importance ratio clipping of PPO (Schulman et al., 2017). To avoid numerical issues, the mean is subtracted from the policy output, which is then clipped within a range. The pseudocode of the parallel implementation is given in Algorithm 2.

Algorithm 2: A Parallel Implementation of ACH

Initialize the policy and critic parameters: θ and ω.
Start multiple actor and learner threads in parallel.
Actors:
while *true* **do**
  Fetch the latest model from the learners.
  Generate samples via self-play in the form: $[a, s, A(s, a), G, \pi_{old}(a|s)]$.
  Send the samples to the replay buffer.
Learners:
for $t \in 1, 2, 3, ...$ **do**
  Fetch a mini batch of samples from the replay buffer.
  $\mathcal{L}_{sum} = 0$.
  for *each sample* $[a, s, A(s, a), G, \pi_{old}(a|s)] \in$ *the mini batch* **do**
    Let c be the clipping mask:
    $$c = \begin{cases} \mathbb{1}\{\frac{\pi(a|s;\theta)}{\pi_{old}(a|s)} < 1+\varepsilon\}\mathbb{1}\{y(a|s;\theta) - \bar{y}(\cdot\,|s;\theta) < l^{th}\} & \text{if } A(s,a) \geq 0, \\ \mathbb{1}\{\frac{\pi(a|s;\theta)}{\pi_{old}(a|s)} > 1-\varepsilon\}\mathbb{1}\{y(a|s;\theta) - \bar{y}(\cdot\,|s;\theta) > -l^{th}\} & \text{if } A(s,a) < 0. \end{cases}$$
    $\mathcal{L}_{sum} \mathrel{+}= -c\eta(s)\frac{y(a|s;\theta)}{\pi_{old}(a|s)}A(a, s) + \frac{\alpha}{2}[V(s;\omega) - G)]^2 + \beta\sum_a \pi(a|s;\theta)\log\pi(a|s;\theta)$.
  Update θ and ω once using gradient on $\mathcal{L}_{sum}$.

We employ an entropy loss to encourage exploration during training and hopefully the convergence of current strategy to a NE (Srinivasan et al., 2018). The parallel implementation updates θ and ω simultaneously, and the overall loss is:

$$\mathcal{L}_{ACH} = -c\eta(s)\frac{y(a|s;\theta)}{\pi_{old}(a|s)}A(a, s) + \frac{\alpha}{2}[V(s;\omega) - G)]^2 + \beta\sum_a \pi(a|s;\theta)\log\pi(a|s;\theta). \tag{31}$$

Theoretically, $y(a|s';\theta)$ for unsampled states s' should also be trained with the target $y(a|s';\theta_{old})$. Yet, in the parallel implementation, we only update θ once using a single mini-batch at each iteration. Therefore, θ is equal to θ_{old} before the update. In other words, the policy loss for unsampled states is 0 in the parallel implementation of ACH.

E HYPER-PARAMETERS

E.1 HYPER-PARAMETERS FOR THE 1V1 MAHJONG EXPERIMENT

We used the Adam optimizer (Kingma & Ba, 2014) for all experiments on the 1v1 Mahjong benchmark. We performed a mild hyper-parameter search on PPO and used the best setting for the shared hyper-parameters of all methods (PPO, RPG, NeuRD, and ACH). For each setting in the hyper-parameter search, we conducted 3 independent runs with different random seeds. Table 4 gives an overview of hyper-parameters for each method. Also, we report the performance of current strategy, instead of the average strategy, for each method.

Parameter	**Range**	**Best**
Shared		
Ratio clip (ε)	-	0.5
GAE (λ)	-	0.95
Learning rate	{2.5e-3, 2.5e-4}	2.5e-4
Discount factor (γ)	-	0.995
Value loss coefficient (α)	-	0.5
Entropy coefficient (β)	{1e-1, 1e-2}	1e-2
Batch size	{4096, 8192}	8192
NeuRD		
Logit threshold (l^{th})	-	6.0
ACH		
Logit threshold (l^{th})	-	6.0
Hedge coefficient ($\eta(s)$)	{1.0, 1e-1}	1.0

Table 4: The hyper-parameters used for the 1v1 Mahjong experiment.

E.2 HYPER-PARAMETERS FOR THE EXPERIMENT ON OPENSPIEL

We used stochastic gradient descent with constant learning rate for all experiments on benchmarks from OpenSpiel. For A2C and RPG, we used the default implementations and hyper-parameters in OpenSpiel. NeuRD was originally implemented using counterfactual regrets in OpenSpiel. We re-implemented NeuRD using predicted advantages and set the shared hyper-parameters of NeuRD identical to those in RPG. All methods employ an entropy loss to encourage exploration during training and hopefully the convergence of current strategy to a NE. Also, we report the performance of current strategy, instead of the average strategy, for each method.

Parameter	**Range**	**Best**
Learning rate	{1e-3, 5e-3}	1e-3
Value loss coefficient (α)	{1.0, 2.0}	2.0
Hedge coefficient ($\eta(s)$)	{1.0, 1e-1, 1e-2}	1.0

Table 5: The hyper-parameter search ranges and best settings of ACH for the OpenSpiel experiment.

In OpenSpiel, the value parameters are updated separately and more frequently compared to the policy parameters for A2C, RPG, and NeuRD. Yet, the value loss and the policy loss are combined in ACH, and all parameters are updated simultaneously, as illustrated in Algorithm 2. Also note that, in the single-threaded OpenSpiel training environment, the actor and the learner run in sequence. As a result, $\pi(a|s;\theta)$ is always equal to $\pi_{old}(a|s)$ in Algorithm 2.

We performed a mild hyper-parameter search for ACH, which is illustrated in Table 5. For each setting in the hyper-parameter search, we conducted 4 independent runs with different random seeds. The final hyper-parameters used for each method are listed in Table 6, with 3 additional hyper-parameters of ACH listed in the "Best" column in Table 5.

Parameter	A2C	RPG	NeuRD	ACH
Batch size	4	64	64	64
Critic learning rate	1e-4	1e-2	1e-2	-
Policy learning rate	1e-4	1e-2	1e-2	-
# Critic updates per policy update	32	32	32	-
Entropy coefficient (β)	1e-2	1e-2	1e-2	1e-2
Logit threshold (l^{th})	-	-	2.0	2.0

Table 6: The hyper-parameters used for the OpenSpiel experiment.

F Additional Results on Benchmarks in OpenSpiel

As a complement, we present the head-to-head performance of A2C, RPG, NeuRD, and ACH on the three benchmarks in OpenSpiel: Kuhn Poker, Leduc Poker, and Liars Dice. As demonstrated in Table 7, the agent of ACH won all other agents across all the three benchmarks. The superior performance of ACH is most significant on the Liars Dice benchmark, which is the most complex one of the three benchmarks.

	Kuhn Poker			Leduc Poker			Liars Dice		
	A2C	**RPG**	**NeuRD**	**A2C**	**RPG**	**NeuRD**	**A2C**	**RPG**	**NeuRD**
RPG	**-0.024**±0.20	-	-	**0.521**±0.23	-	-	**0.192**±0.14	-	-
NeuRD	**-0.096**±0.16	**-0.087**±0.16	-	**0.569**±0.16	**-0.108**±0.17	-	**-0.226**±0.05	**-0.355**±0.12	-
ACH	**0.104**±0.05	**0.115**±0.05	**0.118**±0.05	**0.495**±0.22	**0.050**±0.14	**0.117**±0.15	**0.275**±0.05	**0.122**±0.05	**0.407**±0.06

Table 7: Mean ($\pm$ standard deviation) of the average winning scores of the row agents against the column agents. The mean and the standard deviation are estimated by 8 independent runs. In each run, the average winning scores are obtained via $10,000$ head-to-head plays. All the agents are selected at the $1e7$th training step.

G The Relationship Between ACH and Suphx

Recently, Suphx (Li et al., 2020b) has achieved stunning performance on Japanese Riichi Mahjong. The development of Suphx is enabled by a novel integration of existing supervised learning and RL methods in addition to some newly developed techniques. Three new techniques were introduced in Suphx: global reward prediction, oracle guiding, and run-time policy adaptation. The global reward prediction technique is to handle the multi-round game situation, which is irrelevant to our 1v1 Mahjong setting (only one round per game in our test setting). The oracle guiding technique decays the invisible feature during training, which is of independent interest in dealing with imperfect-information. The run-time policy adaptation technique adapts the trained policy at test time, and this can be combined with ACH, which is a training algorithm. In summary, Suphx is more of a novel system than a new algorithm. For this reason, we did not implement and compare Suphx with ACH in our 1v1 Mahjong experiment. Nonetheless, the oracle guiding technique may be complementary to ACH in handling imperfect-information, and we will investigate this in our future work.