Utilizing History Information in Acquiring Strategies for Board Game Geister by Deep Counterfactual Regret Minimization

Chen Chen[1,a)] Tomoyuki Kaneko[2,b)]

Abstract: Counterfactual Regret Minimization (CFR) is an effective method to compute approximated Nash Equilibria for large zero-sum, imperfect information games. With the help of deep neural networks, tabular CFR has been extended to Deep CFR, which is capable to be applied to larger games. In this paper, we propose a variant of Deep CFR algorithm and evaluate its performance on the board game Geister. We train the agents with and without history information and compare their performance.

Keywords: Counterfactual Regret Minimization, CFR, board game, Geister, deep neural networks

1. Introduction

Counterfactual Regret Minimization (CFR) is an effective method to compute Nash Equilibria for large zero-sum extensive games with imperfect information. CFR has been proven to be effective in solving large poker games [1]. In the study [2] by Bowling et al., the game of heads-up limit hold'em is weakly solved by a variant of CFR called CFR$^+$. While most CFR variants remain tabular representation, deep neural networks provide function approximation, which enables CFR to be extended to Deep CFR so as to solve larger games [3]. While CFR variants have made great contributions to card games, there is little research on applying CFR to board games. In this paper, we propose a variant of Deep CFR and apply it to the board game Geister to train game agents. We train the agents with and without history information and evaluate them by self-play against the random player and other agents. We show that our algorithm is able to acquire an appropriate strategy for the game Geister.

2. Background

2.1 Extensive Games

In a finite extensive game with imperfect information,

- N is a finite set of players, and c stands for a chance player. For player i, $-i$ stands for all other players.
- H is a finite set of possible histories h, and $Z \subseteq H$ is a finite set of all terminal histories. $A(h)$ denotes available actions for a non-terminal history h. A prefix h' of history h means that h begins with h'.
- $\mathcal{I}_i$ is a finite set of information sets I for player i. An information set is a set of histories that player cannot distinguish one from another. Available actions at information set I is denoted as $A(I)$.
- σ is a strategy profile consisting of a strategy σ_i for each player i. Σ_i is all strategies for player i. $\sigma_{I \to a}$ represents a strategy profile identical to σ except that action a is always chosen when in information set I.
- u is a utility function. $u_i(z)$ is player i's utility on terminal history $z \in Z$.
- $\pi^\sigma(I)$ stands for the probability of reaching information set I if players act according to σ. $\pi_i^\sigma(I)$ is player i's contribution to the probability.

2.2 Nash Equilibrium

Nash equilibrium is a solution concept of an extensive game. In a Nash Equilibrium, if each player is informed of the equilibrium strategies of other players, no player is able to improve his utility by altering his strategy alone.

For a two-player extensive game, a Nash equilibrium can be represented as a strategy profile σ that satisfies

$$\begin{cases} u_1(\sigma) \geq \max\limits_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) \\ u_2(\sigma) \geq \max\limits_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2') \end{cases}$$

ϵ-Nash equilibrium is an approximation of a Nash equilibrium, which is a strategy profile σ that satisfies

$$\begin{cases} u_1(\sigma) + \epsilon \geq \max\limits_{\sigma_1' \in \Sigma_1} u_1(\sigma_1', \sigma_2) \\ u_2(\sigma) + \epsilon \geq \max\limits_{\sigma_2' \in \Sigma_2} u_2(\sigma_1, \sigma_2') \end{cases}$$

For a two-player game, making decisions according to a Nash equilibrium will not lose the game in expectation. A game can be regarded as solved if its Nash equilibrium strategies are achieved.

[1] Graduate School of Arts and Sciences, The University of Tokyo
[2] Graduate School of Interdisciplinary Information Studies, The University of Tokyo
[a)] chenchen-319@g.ecc.u-tokyo.ac.jp
[b)] kaneko@graco.c.u-tokyo.ac.jp

2.3 The Game of Geister

Geister is a board game with imperfect information designed by Alex Randolph for two players [4]. It is also referred to as Ghosts or Phantoms vs Phantoms.

Geister is a two-player game on a 6×6 game board. Each player has four good ghosts and four evil ghosts assembled in two rows. The good ghosts are represented in blue and the evil ones are represented in red. The types of the player's ghosts are not revealed to the opponent player [4].

Players may assemble their ghosts as they wish at the beginning of the game. Then, in each turn, a player can move one of his ghosts one step vertically or horizontally. Moving into a square containing an opponent's ghost will capture the opponent's ghost. Moving into a square containing an ally ghost is not allowed. A player's good ghosts can escape from the opponent player's corner squares. A player will win if one of the three constraints is satisfied [5]:

- All the player's evil ghosts are captured.
- All the opponent's good ghosts are captured.
- One of the player's good ghosts escapes from one of the opponent's corner squares.

Figure 1 shows a sample board of the game Geister.

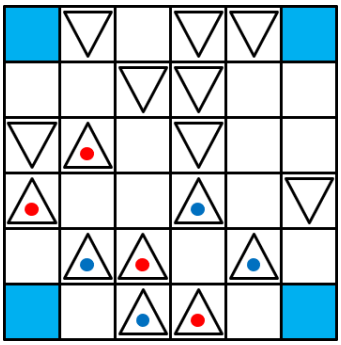

Fig. 1 A Sample Board of the Game Geister

3. Related Research

3.1 CFR

Counterfactual Regret Minimization was first developed by Zinkevich et al. to approximate a Nash equilibrium for very large instances of imperfect information extensive games [1].

CFR is an iterative method that conducts self-play repeatedly. The algorithm keeps track of the cumulative counterfactual regret for each action a in each information set I and calculates the average strategy over all iterations.

Define counterfactual value $v_i(\sigma, I)$ as

$$v_i(\sigma, I) = \sum_{z \in Z_I} u_i(z)\pi^{\sigma}_{-i}(z[I])\pi^{\sigma}(z[I], z)$$

where Z_I is the set of terminal histories with a prefix in I and $z[I]$ is the particular prefix contained in I.

Let σ_i^t be the strategy used by player i on iteration t. The counterfactual regret on iteration t is defined as

$$r_i^t(I, a) = v_i(\sigma^t_{I \to a}, I) - v_i(\sigma^t, I)$$

and the cumulative conterfactual regret is

$$R_i^T = \sum_{t=1}^{T} r_i^t(I, a)$$

Denote the probability that action a is chosen at information set I by $\sigma(I, a)$. With regret matching, the strategy on iteration $T+1$ is

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

where $R_i^{T,+}(I, a) = \max(R_i^T(I, a), 0)$.

Now, player i's average strategy $\bar{\sigma}_i^T$ for information set I is defined as

$$\bar{\sigma}_i^T(I, a) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I, a)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)}$$

Zinkevich et al. applied counterfactual regret minimization to the game of Poker. With abstraction and chance-sampling, which samples a deterministic action at chance nodes, the resulting strategy has outperformed all of the competitors from the bankroll portion of the 2006 AAAI Computer Poker Competition [1].

3.2 Monte Carlo Sampling CFR

Lanctot et al. described Monte Carlo counterfactual regret minimization (MCCFR) based on CFR [6]. This work focuses on avoiding traversing the entire game tree on each iteration while the regret values are kept unchanged in expectation.

On each iteration, the terminal histories to be considered is restricted. Define $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_r\}$ as a number of subsets of Z, and the union of these subsets spans the set of all terminal histories Z. Each subset is called a block. On each iteration, only one of these blocks Q_j is sampled and only terminal histories in that block are considered. On the current iteration, we denote the probability of sampling Q_j as $q_j > 0$ where $\sum_{j=1}^{r} q_j = 1$.

Define $q(z) = \sum_{j:z \in Q_j} q_j$ as the probability that the terminal history z is considered on this iteration. When updating block j, the sampled counterfactual value is defined as

$$\tilde{v}_i(\sigma, I|j) = \sum_{z \in Q_j \cap Z_I} \frac{1}{q(z)} u_i(z)\pi^{\sigma}_{-i}(z[I])\pi^{\sigma}(z[I], z).$$

According to Lemma 1 in the study [6], the sampled counterfactual value equals the counterfactual value in expectation:

$$E_{j \sim q_j}[\tilde{v}_i(\sigma, I|j)] = v_i(\sigma, I).$$

Using the Lemma above, MCCFR algorithm can be performed as follows:

For iteration t:

- Sample a block Q_j from $\mathcal{Q}$;
- For each information set I that contains a prefix in Q_j, calculate the sampled counterfactual regret values as

$\tilde{r}(I,a) = \tilde{v}_i(\sigma^t_{I\to a}, I) - \tilde{v}_i(\sigma^t, I)$ and update them;
- Calculate the strategy according to regret matching for the next iteration $t+1$.

Obviously, if we have $\mathcal{Q} = Z$, then $q_1 = 1$, the resulting MCCFR algorithm is identical to vanilla CFR.

Two efficient MCCFR sampling schemes are proposed in the study of [6]: outcome-sampling and external-sampling.

3.2.1 Outcome-Sampling MCCFR

In outcome-sampling MCCFR, each block contains a single terminal history, and on each iteration, only this terminal history and information sets along this history will be updated, which means $\forall Q \in \mathcal{Q}, |Q| = 1$ [6]. The sampling probability satisfies a specific sampling profile σ', so that $q(z) = \pi^{\sigma'}(z)$. Terminal histories z are sampled according to σ', and $\pi^{\sigma'}(z)$ is stored to compute the sampled counterfactual regret values as

$$\tilde{r}(I,a) = \begin{cases} W \cdot (\pi^\sigma(z[I]a, z) - \pi^\sigma(z[I], z)) & \text{if}(z[I]a) \sqsubseteq z \\ -W \cdot \pi^\sigma(z[I], z) & \text{otherwise,} \end{cases}$$

where

$$W = \frac{u_i(z)\pi^\sigma_{-i}(z[I])}{\pi^{\sigma'}(z)}.$$

When choosing a sampling profile σ', the most straightforward way is called epsilon-on-policy exploration [7]. At information set I, when traversing the game tree, sampling profile has a probability of ϵ to be a uniform distribution, otherwise, it will be the player's current strategy $\sigma^t(I)$.

As there is only one action sampled at each information set, each iteration takes linear time in the depth of the game tree regardless of the number of actions at each information set. However, due to the practice of sampling, exploring is always needed and some actions may never be chosen [7].

3.2.2 External-Sampling MCCFR

In external-sampling MCCFR, only the actions of the opponent and chance player are sampled. We choose the sampling profile that satisfies: for the information sets belonging to the player, every action is explored while in opponent's information sets, a single action is sampled according to the opponent's current strategy $\sigma^t_{-i}(I)$. This sampling profile results in that the sampling probability $q(z)$ for terminal history z is $q(z) = \pi^\sigma_{-i}(z)$, which cancels out the reaching probability of the opponent's player. So the sampled counterfactual regret value can be calculated as

$$\tilde{r}(I,a) = \sum_{z \in Q \cap Z_I} u_i(z) \left(\pi^\sigma_i(z[I]a, z) - \pi^\sigma_i(z[I], z)\right).$$

Since the sampled values don't include terms related to opponent's reaching probabilities, there is no need for passing probabilities to the recursive function, which makes the implementation simple and elegant. However, the sampling only occurs in opponent information sets, so each iteration still takes exponential time to the number of actions. Comparing to vanilla CFR, if each player acts alternately, the exponent will be reduced by half [7].

Empirically, MCCFR requires more iterations, but each iteration has a lower computational cost so that it converges dramatically faster than CFR in various

3.3 Deep CFR

Brown at el. proposed Deep Counterfactual Regret Minimization (Deep CFR) in the study [3]. Deep CFR is proposed to be the first non-tabular CFR variant to be successful in large games [3].

Deep CFR uses deep neural networks to approximate the behavior of a variant of CFR called Linear CFR [3]. Linear CFR does similar iterations with CFR, except that the counterfactual values and strategies generated on iteration t are weighed by t. Empirically, Linear CFR results in faster convergence than CFR and tolerates approximation error well so that it can be approximated by neural networks [3].

In the study [3], Deep CFR with external sampling is introduced. The neural network approximates the advantage rather than regret values. Advantage is the difference in expected payoff between playing a and playing according to $\sigma^t_i(I)$ at information set I.

Deep CFR algorithm keeps reservoir sampling buffers for players' advantage and strategy. On each iteration t, Deep CFR conducts K times of traversals of the game tree according to external-sampling MCCFR. The network approximates the advantage value for information set I and generate a strategy by a slight different way of regret matching, which will choose the action with the greatest advantage when the sum of positive advantages is nonpositive [3].

During the traversal, the traverser's advantages will be added to his advantage buffer and the opponent's strategies will be added into the strategy buffer, with all samples weighed by current iteration t. After all traversals on each iteration, a value network is trained from scratch using the advantage buffer of the traverser. After all iterations, a new policy network, which has the same architecture as the value network except that the last layer applies softmax activation, is trained from scratch using the strategy buffer. A loss function that satisfies Bregman divergence can be used for the networks, such as mean squared error loss [3].

Without relying on advanced domain knowledge, Deep CFR shows strong performance in large poker games relative to domain-specific abstraction techniques [3].

4. Proposed Methods

CFR algorithms have made great contributions to solving games such as Poker and Bluff [1], [8]. However, there is little research on applying CFR algorithms to board games. In our former research, it is proposed that tabular CFR is able to generate an appropriate strategy for a simplified version of Geister [9]. In this research, we apply CFR algorithms to the full game of Geister. Since Geister is an infinite game, instead of approximating the exact Nash equilibrium solution to the game of Geister, we empirically observe whether the algorithm is able to acquire an appropriate strategy.

The full game of Geister is a large game with at most 10^{18} game states, which is impractical for tabular CFR variants to be applied to the game. While using neural networks

is preferable for large games, DeepStack relies on tabular CFR$^+$ to generate training data for the network, which is also difficult to implement. Therefore, we believe that Deep CFR is the most preferable method.

4.1 Sampling Schemes and Sampling Weight

While Deep CFR uses external-sampling when collecting samples, we found it impractical to conduct external-sampling in Geister due to computational cost. In Geister, each player has 4 good ghosts and 4 evil ghosts which can be moved vertically or horizontally, resulting in at most 32 possible actions. At a non-terminal state, the minimal number of possible actions is 3. With a length limit of 100, traversing the game according to external-sampling will result in computational complexity of at least $3^{50} = 7.18 \times 10^{23}$, which is far beyond the ability of our computational resources. Therefore, we have to consider other sampling schemes.

Outcome-sampling MCCFR has computational complexity linear in the depth of the game tree, which means linear time in the move length. While Deep CFR uses external-sampling, it is also proposed that as long as the samples are weighed properly, almost any sampling scheme is acceptable [3]. External-sampling takes advantage by avoiding the weights of reaching probabilities and sampling probabilities, such that the weights of the samples only contain terms about iteration t. In contrast, samples from outcome-sampling have to be weighed by reaching probabilities and sampling probabilities appropriately so that the sampled values approximate the true values in expectation.

We propose that as long as we train the neural network using mean square error loss, we can either contain the probabilities in the value or the training loss weight. Similar to Deep CFR, we use softmax activation on the last layer of the policy network [3], so it is better to contain all the probabilities in the training loss weight. For consistency, for the value network, we also want to contain the reaching probabilities and sampling probabilities in the training loss weight. However, only reaching probabilities are contained in the loss weight and the sampling probabilities are contained in the value for a simpler implementation. Besides probabilities, samples from iteration t are weighed by $\lfloor \frac{t+1}{2} \rfloor$ and the weight is rescaled by $\frac{1}{T}$ when training the network on iteration T. In outcome-sampling, we use an $\epsilon = 0.6$, which is the same value as the one in the study [6]. Similar to Deep CFR, we also conduct K partial traversals on each iteration.

As the neural networks are trained to approximate the weighted average of the training data, the scale of the loss has little practical meaning. However, we observe that if actions with a low probability are sampled multiple times along a single traversal, the probability becomes so low that the loss value becomes NaN and makes the network untrainable. We clip the values if the absolute values get over 10^6, and clip the weights into a range between 10^{-6} and 10^6 to avoid numerical instability.

4.2 Network Architecture

While CFR algorithms generally iterate over information sets with perfect recall, containing the full history of a board game will lead to a huge number of information sets, which is difficult to handle. Therefore, we decide to deal with information sets without perfect recall. We make a similar assumption to the one in our former research [9]. For an experienced board game player, he is able to obtain enough information from the current board. To observe how history information affects the training, we also add a part of history information. As we are using Deep CFR variants, we input the information of the current board and history into the neural network.

Geister is a board game, and the information from the board can be easily processed by a convolutional layer. We extract a five-channel structure to represent the board information. The detail of the structure is presented in Table 1.

Table 1 The Structure of the Extracted Board Information

Channel No.	Contents
1	The good ghosts of the player.
2	The evil ghosts of the player.
3	The ghosts of the opponent player.
4	The status of the opponent's taken good ghosts.
5	The status of the opponent's taken evil ghosts.

In channel 1, 2 and 3, if there is a ghost on the board, the corresponding cell will be filled with 1, otherwise 0. In channel 4 and 5, every cell is filled with the number of taken ghosts divided by 3. Both the current board and history information are represented by this structure. The input of the network consists of 11 channels: a 5-channel structure for current board information, 5-channel structure for the one-step previous board, and a channel for playing progress, whose every cell is filled with move length divided by length limit, which will be described in the Section 4.3. To observe the effectiveness of the history information, we also train agents without the history board, whose input only consists of a 5-channel current board structure and a channel for playing progress, a total of 6 channels.

We build our network architecture with some similar features to that of Deep CFR [3]. The input data first goes through a 2×2 and a 1×1 kernel convolutional layers, both containing 16 channels, and then a flatten layer, and after that 3 fully connected layers are applied. Before the output layer, there is a batch normalization layer. The convolutional layers are activated by tanh and the fully connected layers consist of $x_{i+1} = \mathrm{ReLU}(Ax[+x])$, where the optional skip connection $[+x]$ is applied when layers have the same input and output dimension. The output layer is activated by linear activation for the value network and by softmax for the policy network, which outputs the approximated advantage for each of the 32 possible actions or the average strategy. As it may contain illegal moves, we renormalize the values over legal moves when generating a strategy. A sketch of the network architecture is shown in Figure 2.

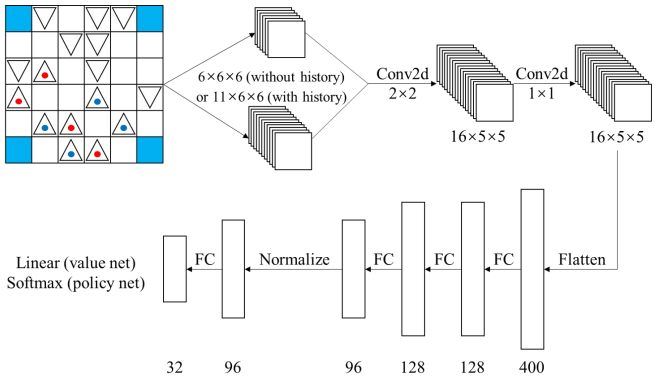

Fig. 2 The Neural Network Architecture

4.3 Bootstrap

Geister is a board game, which means there is a probability that the game never ends. While all CFR variants need to search until the end of the game, for a game with a long history, the number of information sets will explode. Therefore, we have to make efforts to solve the problem.

We consider using the similar way as that in our former research [9], which is to limit the total moves of players and terminate the game to avoid endless games when training the agents. The number of total moves is called *move length* or *history length*. When move length reaches the preset *length limitation*, the game is forcibly terminated with a draw. Although the agents are trained under the length limitation, they are able to play the game with or without it.

We also make other efforts to train the agents more efficiently. Geister is a game without chance players where players arrange their ghosts at the beginning of the game. However, in our Geister version, players will be designated an arrangement of ghosts randomly as a chance node at the beginning of the game. In addition, if a player is able to escape his good ghost from the opponent's corner, his available action is only escaping, making him the winner.

In our implementation, the length limitation is set as 100. We perform self-plays to evaluate the trained agents. The self-play environment limits the number of moves to 300, which is the same value as the one in GPW Geister AI competitions. We only count the wins, losses and draws.

4.4 Proposed Algorithm

Now we propose the algorithm *Deep Counterfactual Regret Minimization with Outcome-Sampling*. The pseudocode is described in Algorithm 1, 2, 3 and 4. We implement the algorithm and conduct several experiments on the board game Geister, whose details are introduced in the next section.

5. Experiments and Results

5.1 Preconditions

We implemented Deep Counterfactual Regret Minimization with Outcome-Sampling and applied to the full game of Geister. Our implementation is written in Python 3 language. Neural networks are implemented using Keras with

Algorithm 1 Deep Counterfactual Regret Minimization with Outcome-Sampling

Initialize reservoir-sampled advantage buffer $B_1 = \emptyset, B_2 = \emptyset$ and strategy buffer $B_s = \emptyset$
Initialize value networks V_1, V_2 and policy networks V_s randomly
for $t = 1, 2, \ldots, N_{iter}$ **do**
  $p \leftarrow 2 - t\%2$
  for $n = 1, 2, \ldots, K$ **do**
    CollectSamples$(\emptyset, p, t, 1, 1, 1)$
  end for
  TrainNetwork(p, t)
  if $t\%N_{checkpoint\_iterval} = 0$ **then**
    TrainNetwork$(-1, t)$
  end if
end for
return V_s

Algorithm 2 Sample Collection Traversal with Outcome-Sampling

Require: CollectSamples$(h, p, t, \pi_i, \pi_{-i}, \pi_{sample})$
if $h \in Z$ **then**
  return $(u_p(h)/\pi_{sample}, 1)$
end if
if $L(h) \geq L_{limit}$ **then**
  return $(0, 1)$
end if
$\sigma(I) \leftarrow$CalculateStrategy$(I(h), p)$
if $P(h) = p$ **then**
  $\sigma'(I) \leftarrow (1 - \epsilon)\sigma(I) + \epsilon \cdot \mathrm{Unif}(I)$
else
  $\sigma'(I) \leftarrow \sigma(I)$
end if
$a \sim \sigma'(I)$
if $P(h) = p$ **then**
  $(u, \pi_{tail}) \leftarrow$
  CollectSamples $(h \cdot a, p, t, \pi_i \cdot \sigma(I, a), \pi_{-i}, \pi_{sample} \cdot \sigma'(I, a))$
  for $i \in A(h)$ **do**
    if $i = a$ **then**
      $\tilde{v}(I, i) = u - u \cdot \sigma(I, a)$
    else
      $\tilde{v}(I, i) = -u \cdot \sigma(I, a)$
    end if
  end for
  Add$\{(I, \tilde{v}(I), \lfloor \frac{t+1}{2} \rfloor \cdot \pi_{-i} \cdot \pi_{tail})\}$ to B_p
else if $P(h) = 3 - p$ **then**
  $(u, \pi_{tail}) \leftarrow$
  CollectSamples $(h \cdot a, p, t, \pi_i, \pi_{-i} \cdot \sigma(I, a), \pi_{sample} \cdot \sigma'(I, a))$
  Add$\{(I, \sigma(I), \lfloor \frac{t+1}{2} \rfloor \cdot \pi_{-i}/\pi_{sample})\}$ to B_s
end if
return $(u, \pi_{tail} \cdot \sigma(I, a))$

Algorithm 3 Strategy Calculation

Require: CalculateStrategy(I, p)
  Calculate $\sigma_{legal}(I)$
  if Network V_p has not been trained for even once **then**
    $sum \leftarrow 0$
    for $a \in A(I)$ **do**
      if $\sigma_{legal}(I, a) =$ True **then**
        $sum \leftarrow sum + 1$
      end if
    end for
    for $a \in A(I)$ **do**
      $\sigma(I, a) \leftarrow \sigma_{legal}(I, a)/sum$
    end for
    return $\sigma(I)$
  end if
  $\hat{R}(I) \leftarrow V_p(I)$
  for $a \in A(I)$ **do**
    if $\sigma_{legal}(I, a) =$ False **then**
      $\hat{R}(I, a) \leftarrow -\inf$
    end if
  end for
  $sum \leftarrow 0$
  for $a \in A(I)$ **do**
    $sum \leftarrow sum + \max\{\hat{R}(I, a), 0\}$
  end for
  if $sum > 0$ **then**
    for $a \in A(I)$ **do**
      $\sigma(I, a) \leftarrow \max\{\hat{R}(I, a), 0\}/sum$
    end for
  else
    for $a \in A(I)$ **do**
      $\sigma(I, a) \leftarrow 0$
    end for
    $\sigma(I, \mathrm{argmax}_a\{\hat{R}(I, a)\}) \leftarrow 1$
  end if
  return $\sigma(I)$

Algorithm 4 Network Training

Require: TrainNetwork(p, T)
  if $p = -1$ **then**
    for $n = 1, 2, \ldots, N_{train}$ **do**
      Sample N_{batch} samples $\{(I_i, \sigma_i, w_i)^{i=1,2,\ldots N_{batch}}\}$ from the buffer B_s.
      Rescale the weight w_i by $\frac{1}{T}$ and train the policy network V_s.
    end for
  else
    for $n = 1, 2, \ldots, N_{train}$ **do**
      Sample N_{batch} samples $\{(I_i, v_i, w_i)^{i=1,2,\ldots N_{batch}}\}$ from the buffer B_p.
      Rescale the weight w_i by $\frac{1}{T}$ and train the value network V_p.
    end for
  end if

Tensorflow backend. Our training programs are run on an Intel® Core™ i7-6950X CPU machine with an NVIDIA® TITAN X or a GeForce GTX 1080 GPU. The Python 3 interpreter version is 3.6.8 and the Keras version is 2.2.4.

We choose a batch size of 3 584, and for advantage buffers and the strategy buffer, we set the capacity to be 1 075 200, which is exactly 300 batches. Once the buffer is full, it will be updated according to reservoir sampling. On each iteration, the game tree is traversed 384 times and the length limitation is 100. For agents with and without moving constraint, we run the algorithm for 1 000 iterations and save the checkpoint every 100 iterations respectively. However, these hyperparameters are not finely tuned. When training the networks, we use a mean square error loss function and update the parameters using the Adam optimizer with a learning rate of 0.01 and gradient norm clipping to 1., the same as the settings in Deep CFR [3].

The strategy used by our agents is generated by the policy network models saved at the checkpoints. When the agent needs to make a move, a 6- (or 11-) channel structure is generated from the board (and the history), and is input into the policy network to predict the average strategy, which is then renormalized over the legal move vector to eliminate illegal moves. After that, an action is sampled according to the strategy and the agent makes the move.

5.2 Self-play against the Random Player

We trained two agents with and without history information, and evaluated them by self-play against the random player. The self-play is performed via the Geister AI Competition server released in Github [10]. The self-play is also performed on an Intel® Core™ i7-6950X CPU machine with an NVIDIA® TITAN X or a GeForce GTX 1080 GPU.

For every agent, we conducted 2 000 battles between the 10 saved checkpoints and the random player to observe the performance, in which our agents play 1 000 battles as the first-hand player and 1 000 as the second. Our agents as well as the random player obey the constraints in Section 4.3. We executed the program from scratch to train the agents for 6 times and collected the average win rate of our agents. The results are shown in Figure 3.

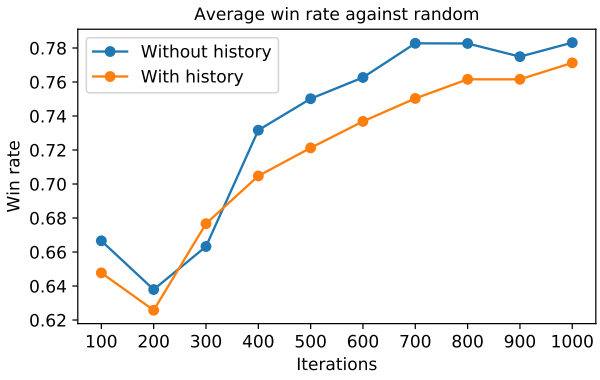

Fig. 3 Average Win Rate against Random Player

As is shown in the figure, both the agents trained with or without history are able to beat the random player by a win rate of over 75%. We can also observe from the figure that the agents are performing better when training continues. While we imagine that training the agent with history information will obtain more information from the board and lead to a more robust and stable learning, the result turns out that the agent without history information outperforms the one with history.

Although we provide the average data, we observe that the win rates from independent executions are quite unstable, which may be caused by insufficient traversal brought by outcome-sampling. The result is shown in Figure 4. As Geister is a huge game with 10^{18} game states, we only explore at most 3.84×10^7 states in each execution, which is a quite small proportion compared to the scale of the game. As a result, due to the randomness, every time the agent only learns a tiny part of the full game and every time the agent may learn a different partition of the full game. Therefore, it is quite possible for the agent to fall into local optima of the explored states or learn a strong overall strategy, basing on the part of the game it traversed. We observe that there is an agent without history from execution earned an unusual high win rate, which greatly affected the average win rate. We infer that the effectiveness of training with history may depend on other constraints. As we still believe there is high possibility for training with history information to bring more benefits, we will investigate more on the reason of unstablity and conduct more experiments to verify the effectiveness of adding history information.

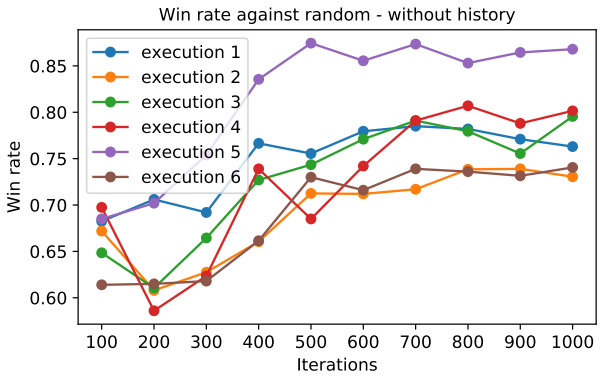

Fig. 4 Win Rate against Random Player - without history

5.3 Self-play against Other Agents

In order to evaluate the true strength of our agents, we conducted self-play with agents developed by other researchers of Geister. Our agents battled with a program called PurpleAI. This program was kindly borrowed from Mr. Kawakami from JAIST Ikeda Laboratory. The techniques of this agent are introduced in the study [11], however, we have no access to the source code so it might contain other unknown techniques.

According to the study of Mr. Kawakami [11], PurpleAI adopted Geister-specific evaluation functions and meta-heuristic calculation methods to perform MinMax search in the game tree. The search is performed under a perfect information situation where the enemy's ghosts are considered as purple ghosts, the ghosts can escape but will become red if taken. It is proposed that the PurpleAI outperforms the first place program of Geister AI Competition in GPW 2017 [11] and has also won second place in Geister AI Competition held in GPW 2018 [12].

Using the agents trained in Section 5.2, we also performed self-play via the Geister AI Competition server. We run the PurpleAI program on Linux using a tool called Wine. The self-play is also performed on an Intel® Core™ i7-6950X CPU machine with an NVIDIA® TITAN X or a GeForce GTX 1080 GPU.

We set the searching depth of PurpleAI to 1 and observed the performance of our agents. For every agent, we also conducted 2 000 battles between the 10 saved checkpoints and the PurpleAI, in which our agents played 1 000 battles as the first-hand player and 1 000 as the second. Our agents obey the constraints in Section 4.3. We also collected average win rate of our agents. The results are shown in Figure 3.

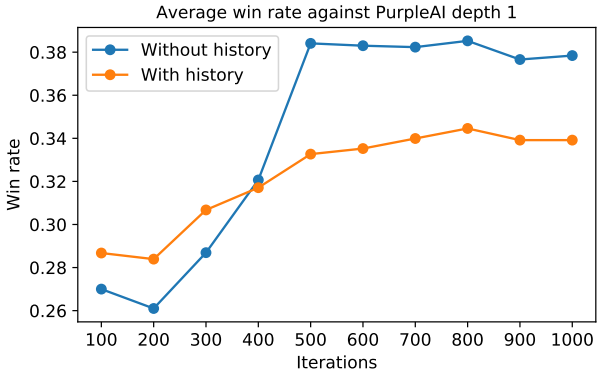

Fig. 5 Average Win Rate against PurpleAI - Depth 1

Unfortunately, our agents failed to compete with PurpleAI, even with the least searching depth 1. Our agents are only able to win about 1 out of 3 battles. This may be caused by the insufficient training problem discussed in Section 5.2.

Although our agents are weak against PurpleAI, we can still observe from the figure that both our agents are gradually learning the correct way to play the game when the training process goes on. The agent trained with history performs slightly better than the one without history in the first stage of training process, but is soon surpassed. Besides the fact that the average result is affected by the unusual data introduced in Section 5.2, we suggest that adding history information in training expands the number of information sets, which leads to a more accurate learning at the beginning, but the buffer size becomes insufficient when the training proceeds and slows down the learning.

To let the good ghosts escape is an important technique for the game Geister. In order to further analyze the result

of self-play, we investigated the details of the results and counted the percentage of our wins by letting good ghosts escape. The result is shown in Figure 6.

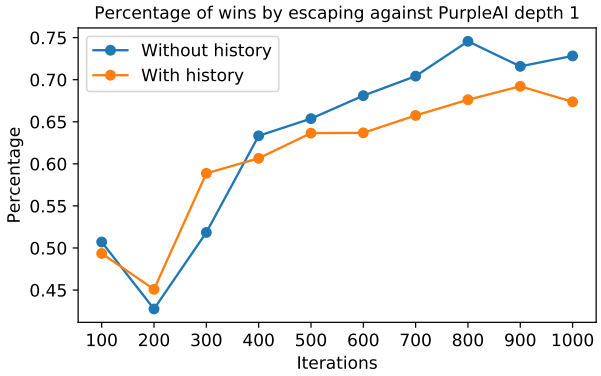

Fig. 6 Percentage of Wins by Escaping against PurpleAI - Depth 1

For comparison, we also performed self-play between the random player and the PurpleAI depth 1. The random player will also certainly let his good ghost escape once possible. We performed 2 000 battles for each searching depth, with the random player playing 1 000 as the first-hand player and 1 000 as the second. The random player wins 399 out of 2 000 battles, in which only 63 wins are letting the good ghost escape.

Comparing with the random player, our agents performed much better. The figure shows that our agents managed to let good ghosts escape, resulting in most of the wins. We can clearly observe from the figure that our agents gradually master the technique of escaping, with a increasing percentage of wins by escaping. We believe that our agents are able to acquire this important skill.

From the results, we can conclude that our agents have learned at least some of the valuable understanding of the game and our algorithm is able to acquire an appropriate strategy for the game Geister. Whether including history information in training board games will enhance the training process remains to be investigate in our future work.

6. Conclusion and Future Works

In this paper, we investigated on applying Counterfactual Regret Minimization algorithms to board games. We proposed an algorithm called *Deep Counterfactual Regret Minimization with Outcome-Sampling* which makes it possible for traversing the game tree in a deep and wide game that is impractical to be traversed by external-sampling. We made bootstrap to train the agents efficiently, and applied the proposed algorithm to the full game of Geister. We trained two agents with and without history information, and evaluated them by self-play with random and other agents. Our agents are able to beat the random player by a win rate of over 75%. However, a strong agent against random player is not necessarily a desirable agent. Our agents failed to compete with the leading Geister agent, PurpleAI, even with the least searching depth of 1. Nevertheless, analysis shows that they are able to gradually manage the correct technique of playing the game, and most of the wins are earned using the learned technique. We predicted that the agent trained with history information should have a stabler learning process and learn the technique better, however, the result was not enough to convince this opinion. We will do future work to investigate on it.

By applying our algorithm, our agents are able to have important understanding and learn correct techniques of the game. There is room for improvements in our implementation. We propose that our algorithm is able to acquire an appropriate strategy for the game Geister.

For future work, as our agents are still very weak, we plan to enhance our agents. As the effectiveness of adding history information is not proved, we would like to add more history moves and do more experiments. Also, our network architecture and hyperparameters are not finely tuned, we plan to tune them to improve the performance. To solve the problem of insufficient traversal caused by outcome-sampling, we would like to have a try on other practical sampling schemes.

References

[1] Zinkevich, M., Johanson, M., Bowling, M. and Piccione, C.: Regret Minimization in Games with Incomplete Information, *Advances in Neural Information Processing Systems 20*, pp. 1729–1736 (2008).

[2] Bowling, M., Burch, N., Johanson, M. and Tammelin, O.: Heads-up limit hold'em poker is solved, *Science*, Vol. 347, No. 6218, pp. 145–149 (online), DOI: 10.1126/science.1259433 (2015).

[3] Brown, N., Lerer, A., Gross, S. and Sandholm, T.: Deep Counterfactual Regret Minimization, *CoRR*, Vol. abs/1811.00164 (online), available from ⟨http://arxiv.org/abs/1811.00164⟩ (2018).

[4] Wikipedia: Ghosts (board game), `https://en.wikipedia.org/wiki/Ghosts_(board_game)`.

[5] BoardGameGeek: Phantoms vs phantoms, `https://www.boardgamegeek.com/boardgame/2290/phantoms-vs-phantoms`.

[6] Lanctot, M., Waugh, K., Zinkevich, M. and Bowling, M.: Monte Carlo Sampling for Regret Minimization in Extensive Games, *Advances in Neural Information Processing Systems 22*, pp. 1078–1086 (2009).

[7] Lanctot, M.: Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games, PhD Thesis, University of Alberta, University of Alberta, Computing Science, 116 St. and 85 Ave., Edmonton, Alberta T6G 2R3 (2013).

[8] Johanson, M., Bard, N., Lanctot, M., Gibson, R. and Bowling, M.: Efficient Nash Equilibrium Approximation through Monte Carlo Counterfactual Regret Minimization, *Proceedings of the Eleventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (2012).

[9] Chen, C. and Kaneko, T.: Counterfactual Regret Minimization for the Board Game Geister, *Game Programming Workshop 2018 Proceedings*, Vol. 2018, pp. 137–144 (2018).

[10] Miyo: Geister Server Java, `https://github.com/miyo/geister_server.java`.

[11] Kawakami, N. and Hashimoto, T.: Research of Geister AI using search for complete information games, *Game Programming Workshop 2018 Proceedings*, Vol. 2018, pp. 35–42 (2018).

[12] GPW2018: Geister AI Competition, `http://www2.matsue-ct.ac.jp/home/hashimoto/geister/`.