Application of DREAM to the Board Game Geister

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

Abstract: Counterfactual Regret Minimization (CFR) is an effective method to compute approximate Nash Equilibria for large zero-sum, imperfect information games. Deep neural networks extend tabular CFR to Deep CFR, which is able to be applied to larger games. DREAM is a model-free neural CFR variant evaluated on card games with a short history. In this paper, we apply DREAM to the board game Geister where the game length might reach a hundred with necessary enhancements to observe whether it can generate a good policy.

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

1. Introduction

Counterfactual Regret Minimization (CFR) is an effective method to approximate Nash Equilibria for large zero-sum extensive games with imperfect information. CFR variants have achieved great success in solving large poker games [1], [2]. In the research by Bowling et al., the game of heads-up limit hold'em is weakly solved using a variant of CFR called CFR$^+$ [2].

However, tabular CFR methods need to calculate for every game state and often need domain-specific expert knowledge to reduce the game scale, so they are not applicable to many interesting games. Neural CFR methods that mimic tabular CFR methods using neural networks have achieved good performance in popular poker games [3], [4], but they often need a perfect simulator of the game to explore multiple actions at decision points to achieve high performance [5]. In the research by Steinberger et al., a model-free neural CFR variant called DREAM achieves state-of-the-art performance and is even competitive with non-model-free methods in popular poker games Leduc and FHP [5].

CFR variants have made great contributions to card games, however, there is little research on CFR applied to board games. In board games, there are often repetitions in game states, and the game may contain loops that make the game never end, which makes it difficult for CFR to be applied. In our previous research, we applied a variant of Deep CFR to the full game of Geister and acquired an appropriate policy [6], [7]. In this paper, we apply DREAM with enhancements to the board game Geister to observe whether it can compute a good policy for the game.

[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. Background

2.1 Notations

In this paper, we followed a standard notation in the study [5]. In a partially observable stochastic game,

- There are $\mathcal{N} = \{1, 2, \cdots, N\}$ players. For player i, $-i$ stands for all other players.
- A world state w is the exact state of the game world. In each world state w, agent i will receive a reward $\mathcal{R}_i(w)$ that can be zero.
- At world state w, player i's legal actions are denoted by $\mathcal{A}_i(w)$, and $a = \{a_1, a_2, \cdots, a_N\} \in \mathcal{A}$ denotes the players' actions.
- A transition function $\mathcal{T}(w, a)$ determines the next world state w' after players taking actions a. After a transition, an observation $\mathcal{O}_i(w, a, w')$ is given to player i, which is the observation of the world state under the view of player i. A chance player who determines the outcome of a random event in the game is included in the transition function and information about other players' actions is given to player i in the observation.
- A history (also called a trajectory) h is a finite sequence of legal actions and world states.
- An infostate (also called an action-observation history) s_i for player i is a sequence of player i's observations and actions. An infostate contains histories that player i cannot distinguish one from another. Legal actions at infostate s_i are denoted as $\mathcal{A}(s_i)$.
- π is a policy profile consisting of a policy π_i for each player i.
- Expected value $v_i^\pi(h)$ is the expected sum of future rewards for player i in history h when all players act according to π. The expected value for an action in a history is denoted $v_i^\pi(h, a_i)$. Similarly, the expected value for an infostate and for an action in the infostate are denoted $v_i^\pi(s_i)$ and $v_i^\pi(s_i, a_i)$.

- $x^{\pi}(h)$ stands for the probability of reaching history h if players act according to π. $x_i^{\pi}(h)$ is player i's contribution to the probability.

3. Game of Geister

Geister is a two-player board game on a 6×6 game board. Each player has four good ghosts and four evil ghosts, whose type is not revealed to the opponent player. 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 and move it off from the board. Moving into a square containing an ally ghost is not allowed. A player can also move off one of his good ghosts from one of the opponent's corner squares. A player wins when one of the three conditions is satisfied [8]:

- All the player's evil ghosts are captured.
- All the opponent's good ghosts are captured.
- One of the player's good ghosts is moved off the board from one of the opponent's corner squares.

Figure 1 shows a sample board of the game Geister.

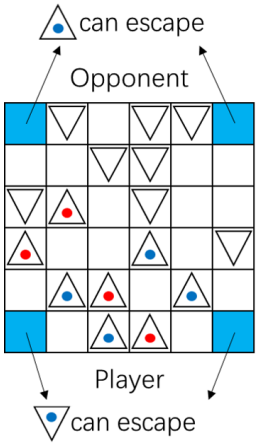

Fig. 1 A sample board of the game Geister. The type of ▽ is not revealed.

4. Related Research

In this paper, we use an algorithm called DREAM [5], which is a combination of outcome-sampling MC-CFR [9], variance reduction [10] and Single Deep CFR [4]. We introduce these works in the following subsections.

4.1 Counterfactual Regret Minimization

CFR 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 traverses the entire game tree repeatedly. The algorithm keeps track of the cumulative counterfactual regret for each action a_i in each infostate s_i and calculates the average policy over all iterations.

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

$$r_i^t(s_i, a_i) = x_{-i}^{\pi^t}(s_i)\left(v_i^{\pi^t}(s_i, a_i) - v_i^{\pi^t}(s_i)\right) \quad (1)$$

and the cumulative conterfactual regret is

$$R_i^T = \sum_{t=1}^{T} r_i^t(s_i, a_i) \quad (2)$$

$\pi(s_i, a_i)$ denotes the probability that action a is chosen at infostate s_i. With regret matching, the policy on iteration $T+1$ is

$$\pi_i^{T+1}(s_i, a_i) = \begin{cases} \frac{R_i^{T,+}(s_i,a_i)}{\sum_{a_i' \in \mathcal{A}_i(s_i)} R_i^{T,+}(s_i,a_i')} \\ \qquad \text{if } \sum_{a_i' \in \mathcal{A}(s_i)} R_i^{T,+}(s_i, a_i') > 0 \\ \frac{1}{|\mathcal{A}(s_i)|} \quad \text{otherwise} \end{cases} \quad (3)$$

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

Now, player i's average policy $\bar{\pi}_i^T$, which is the ϵ-Nash equilibrium policy, for infostate s_i is defined as

$$\bar{\pi}_i^T(s_i, a_i) = \frac{\sum_{t=1}^{T} x_i^{\pi^t}(s_i)\pi^t(s_i, a_i)}{\sum_{t=1}^{T} x_i^{\pi^t}(s_i)} \quad (4)$$

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 policy has outperformed all of the competitors from the bankroll portion of the 2006 AAAI Computer Poker Competition [1].

4.2 Linear CFR

Brown and Sandholm introduced a variant of CFR called Linear CFR [11]. Linear CFR does similar iterations to CFR, but weighs the updates to the regret and average policy on iteration t by t. Linear CFR empirically accelerates CFR by two orders of magnitude [11].

4.3 MC-CFR

Lanctot et al. described Monte Carlo counterfactual regret minimization (MCCFR) based on CFR, which aims to avoid traversing the entire game tree [9]. All possible terminal histories are partitioned into blocks, and on each iteration one of the blocks is chosen and only the terminal histories in that block are considered, while expectation values of counterfactual regret to be updated are unaltered. Two sampling schemes are introduced: external sampling and outcome sampling. In external-sampling MCCFR, only the actions of the opponent and chance are sampled. In outcome-sampling MCCFR, each block contains a single terminal history so that every action is sampled. In OS, a sampling profile $\xi_i^t(s_i)$ is used rather than π^t to choose actions, where $\xi_i^t(s_i, a_i) = \epsilon\frac{1}{|\mathcal{A}_i(s_i)|} + (1-\epsilon)\pi_i^t(s_i, a_i)$ for player i and $\xi_j^t(s_j, a_j) = \pi_j^t(s_j, a_j)$ for player j other than player i. Empirically, MCCFR converges faster than CFR [9].

4.4 VR-MC-CFR

Schmid et al. introduced Variance Reduced Outcome Sampling with Baselines (VR-MC-CFR) [10]. VR-MC-CFR is a variant of MC-CFR that dramatically accelerates the convergence of outcome-sampling MC-CFR by reducing the

variance in each update. It uses a bootstrapped value called baseline-adjusted sampled expected value instead of the estimated value in MC-CFR. The history-action baseline used here is typically the average expected value, but it can be any user-defined scalar funtion closely correlated with the expected value [10].

4.5 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 CFR algorithm. The neural network approximates the advantage rather than regret values. Advantage is the difference in expected payoff between playing a_i and playing according to $\pi_i^t(s_i)$ at infostate s_i.

Deep CFR algorithm keeps reservoir sampling buffers for players' advantage and policy. On each iteration, Deep CFR conducts K times of traversals of the game tree according to external-sampling MC-CFR. The network approximates the advantage value for infostate s_i and generates a policy by a slightly different way of regret matching, which will choose the action with the greatest advantage when the advantage values for all actions are non-positive [3].

During the traversal, the traverser's advantages are added to his advantage buffer and the opponent's policies will be added into the policy buffer. 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 policy buffer. A loss function that satisfies Bregman divergence can be used for the networks, such as mean squared error loss [3].

Deep CFR also incorporates Linear CFR, which weighs the advantage and policy on iteration t by t [11]. Without relying on advanced domain knowledge, Deep CFR shows strong performance in large poker games relative to domain-specific abstraction techniques [3].

4.6 Single Deep CFR

Single Deep CFR (SD-CFR) proposed by Steinberger [4] is a modification of Deep CFR that does iterations in a similar way to Deep CFR, except that it stores all the value networks on each iteration instead of training the policy network. It mimics the average policy by sampling one of the value networks and using its policy for the entire game. It eliminates the approximation error in Deep CFR resulted from training a network to predict the average policy [4].

4.7 DREAM

Steinberger et al. introduced a regret-based deep learning algorithm called Deep Regret minimization with Advantage baselines and Model-free learning (DREAM) [5]. DREAM is trained using CFR iterations similar to SD-CFR and also combines a neural-based baseline as in VR-MC-CFR.

DREAM incorporates outcome-sampling with an adjustable exploration parameter ϵ to collect samples for neural network training. Similar to SD-CFR, the samples for training value networks are stored in a reservoir buffer. The samples are weighted by $\frac{1}{x_i^{\xi^t}(s_i)}$ during neural training [5].

DREAM uses a baseline network $\hat{Q}_i^t(s^*(h), a_i)$ as a baseline, where $s^*(h)$ is the set of infostates for all players. $\hat{Q}_i^t$ is trained using expected SARSA on a circular buffer of collected samples. Using $\hat{Q}_i^t$ as a baseline, the baseline-adjusted sampled expected value for DREAM at history h is defined as

$$\tilde{v}_{i,DREAM}^{\pi^t}(h, a_i|z) = \begin{cases} \hat{Q}_i^t(s^*(h), a_i) \\ \quad + \frac{\tilde{v}_{i,DREAM}^{\pi^t}(h'|z) - \hat{Q}_i^t(s^*(h), a_i)}{x_i^{\xi^t}(s_i, a_i)} \\ \qquad\qquad \text{if } a_i = a_i' \\ \hat{Q}_i^t(s^*(h), a_i) \qquad \text{otherwise} \end{cases} \tag{5}$$

$$\tilde{v}_{i,DREAM}^{\pi^t}(h|z) = \begin{cases} \mathcal{R}_i(h) & \text{if } h = z \\ \sum_{a_i \in \mathcal{A}_i(h)} \pi_i^t(h, a_i) \tilde{v}_{i,DREAM}^{\pi^t}(h, a_i|z) \\ & \text{otherwise} \end{cases} \tag{6}$$

where z is the terminal history finally reached and playing action a_i' leads to the next history h'. Also, we have $\tilde{v}_{i,DREAM}^{\pi^t}(s_i(h), a_i|z) = \tilde{v}_{i,DREAM}^{\pi^t}(h, a_i|z)$ and $\tilde{v}_{i,DREAM}^{\pi^t}(s_i(h)|z) = \tilde{v}_{i,DREAM}^{\pi^t}(h|z)$.

The sampled immediate advantage is defined as

$$\tilde{d}_{i,DREAM}^t(s_i, a_i) = \tilde{v}_{i,DREAM}^{\pi^t}(s_i, a_i) - \tilde{v}_{i,DREAM}^{\pi^t}(s_i). \tag{7}$$

As DREAM also adopts Linear CFR, training loss of $\tilde{d}_i^t$ is weighted by t [5].

To compute the average policy, DREAM follows the way used in SD-CFR, which is to store all value networks on the disk, and sample one of them to determine the policy for the whole game. When implementing Linear CFR, the probability of sampling network on iteration t is proportional to t. As all networks are stored on the disk, it is also possible to calculate the average policy of them [5].

DREAM samples only one action at each desicion point, yet achieved state-of-the-art performance among model-free methods in popular benchmark games and is even competitive with non-model-free algorithms [5].

5. Proposed Methods

In this paper, we apply DREAM for board games — a combination of outcome-sampling MC-CFR [9], variance reduction [10], Single Deep CFR [4] and necessary enhancements — to the full game of Geister. The combination of the first three techniques is proposed as DREAM in the paper [5], and the combination of outcome-sampling [9], Deep CFR [3] and enhancements is proposed in our previous works [6], [7]. This work is the first attempt to apply

DREAM to board games. We will introduce the details of enhancements and implementation in the next subsections.

5.1 Enhancements

Geister is a board game that is not necessarily finite. As the number of infostates increases exponentially to the length of the game, it is hardly possible to deal with such large numbers of infostates due to the fact that all CFR variants need to search until the end of the game. Therefore, we have to make enhancements to address these issues.

Abstraction: While CFR algorithms typically iterate over infostates with perfect recall, containing the full history of a board game will lead to a huge number of infostates, which is difficult to handle. To scale down the number of infostates, we give up perfect recall to abstract the game. We make a similar assumption to the one in our previous study that for an experienced board game player, he is able to obtain enough information from the current board [7]. We extract the information of the current board with additional information about the number of the players' total moves (called *move length* or *history length*) as infostates and input them to our neural networks, whose architectures will be introduced in Section 5.2.

Length limitation: To avoid endless games, we limit the move length to a certain preset number called *length limitation* when training agents, which is the same way as the one in our previous study [7]. When the move length reaches the 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.

Random initialization: In the original game of Geister, players arrange their own ghosts at the beginning of the game. However, in our Geister implementation, the arrangements of the players' ghosts will be generated randomly at the beginning of the game to efficiently explore the game.

Finishing blow assistance: If a player is able to move his good ghost off the board from the opponent's corner, he will be forced to do so and becomes the winner of the game. To keep the game fair, these rules are also applied to random players.

In our implementation, the length limitation is set to 100 in training. We perform self-play to evaluate the trained agents. In our evaluation, the length limitation is set to 300, which is the same value as the one in GPW Geister AI competitions. For the result of self-play, we only focus on the win rate, the ratio of wins in all games.

5.2 Network Architecture

Geister is a board game, and the information from the board can be easily processed by a convolutional layer. We extract a six-channel structure to represent the abstracted infostate similarly to that in our previous study [7]. The detail of the structure is presented in Table 1.

Table 1 The Structure of the Abstracted Infostate

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.
6	The progress of the game.

In channel 1, 2 and 3, for each ghost on the board, the corresponding cell is filled with 1, otherwise 0. In channel 4 and 5, every cell is filled with the number of taken ghosts divided by 3. Channel 6 is a channel representing playing progress, whose every cell is filled with move length divided by length limitation, which are described in the Section 5.1.

We build our network architecture with some similar features to that of Deep CFR [3]. For value networks, the input data first go through a 2×2 and a 1×1 kernel convolutional layers, both containing 16 channels. The convolutional layers are activated by a tanh function. Then the data are flatten and go through 3 fully connected layers. Before the output layer, there is a batch normalization layer. The fully connected layers consist of $x_{i+1} = \mathrm{ReLU}(Ax[+x])$, where the optional skip connection $[+x]$ is the same as that in Deep CFR [3]. It is applied when layers have the same input and output dimension. The output layer is activated by a linear activation function, which outputs the approximated advantage for each of the 32 possible actions. A sketch of the value network architecture is shown in Figure 2.

For baseline networks, they share the same architecture as value networks, expect that the input of baseline networks contains both players' infostates. In the first CNN layer, the player's infostate goes through the first 8 channels and the opponent's goes through the remaining 8 channels to form a 16-channel data for the next layer. The following layers are identical to those of the advantage networks and the final output is the approximated baseline values for 32 actions. Note that the baseline networks are only used in training, so the agents playing the game have no access to the baseline networks nor the opponent's private information.

As it may contain illegal moves, when the networks predict advantage or baseline values, we eliminate the values of illegal moves from the output of the networks. While the architectures are identical, the weights of networks are not shared at all to stablize the training process. Every player possesses an independent baseline network and an independent value network during the training.

5.3 Details in DREAM Implementation

We implemented DREAM algorithm mainly following the description in the DREAM paper [5]. However, we train the networks using a different approach. DREAM trains the baseline network for 1 000 minibatches of 512 samples on each iteration, and trains the value network for 3 000 or 10 000 minibatches of 2 048 samples in different games [5]. In our implementation, we train one epoch of the whole buffer from the weights on the previous iteration for both the baseline network and value network to simplify the im-

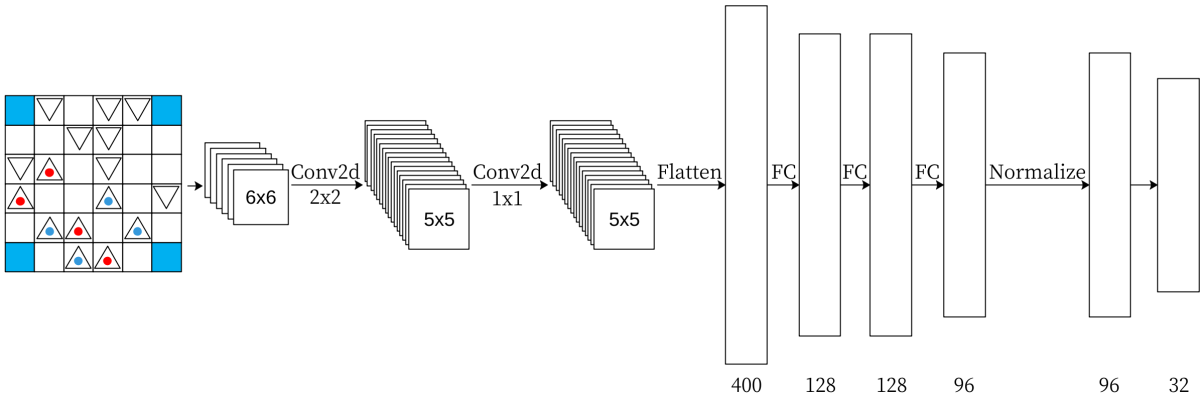

Fig. 2 Advantage Network Architecture

plementation and reduce execution time while making full use of the samples in the buffers.

The game result of Geister can be known only at the terminal of the game, so in our training process, only the terminal state returns a (possibly) non-zero reward, which is 1 for win, -1 for loss and 0 for draw. Other states return a reward equal to zero.

We implemented the CFR iteration in an alternative way, which only updates network weights for one player on each iteration. For a two-player game like Geister, after $2T$ iterations, every player's networks are updated T times. We use T instead of total iterations $2T$ as the weight in Linear CFR.

When generating policy using regret matching, we followed the approach in Deep CFR, where the action with the greatest advantage is chosen at probability 1 when advantage values for all actions are non-positive.

For calculating the average policy, we store all the value networks on the disk, and calculate the policy using regret matching for all networks. We followed the Linear CFR implementation that the policy on iteration t is weighted by t and the weighted average of them derives the average policy.

6. Experiments and Results

6.1 Preconditions

We implemented DREAM with necessary enhancements on board games to the full game of Geister. Our implementation is written in Python 3 language. Neural networks are implemented using PyTorch. Our training programs are run on an AMD Ryzen™ Threadripper™ 1950X 16-Core Processor machine with two NVIDIA GeForce® GTX 1080 Ti GPUs. The Python 3 interpreter version is 3.6.9 and the Pytorch version is 1.5.0. In our previous study [7], the performance suffers from lack of exploration. To cope with this problem, we increase the number of traversals on each iteration and decrease the number of iterations to keep the training time in a reasonable range.

We choose a batch size of 3 584, which is equal to the number of CUDA cores of our GPUs. We set the capacity of both advantage buffers and baseline buffers to be 2 150 400, which is exactly 600 batches. Once the buffer is full, the advantage buffers will be updated according to reservoir sampling while the baseline buffers are circular buffers. We run the algorithm for 200 iterations and all networks' weights are stored on the hard disk. On each iteration, the game is traversed 3 840, 7 680 or 10 240 times. The length limitation is 100. To evaluate the effectiveness of the baseline network, we also trained an agent without baseline network with 3 840 traversals, in total four different settings. Note that 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 0.001 and gradient norm clipping to 1., the same as the settings in Deep CFR [3]. In training value networks, we incorporate Linear CFR [11] so that the losses of samples on iteration t are weighted by t.

The policy used by our agents is the weighted average policy of the value networks on all iterations. When the agent needs to make a move, the 6-channel structure is extracted from the board, and is given to all value networks to predict the advantages. Then a policy is calculated for each value network by regret matching. The policy from iteration t is weighted t to generate the average policy, as is described in Section 5.3. After that, an action is sampled according to the average policy and the agent makes the move.

6.2 Result

We ran the program under the preconditions introduced above and trained 4 agents from stratch in 4 independent executions for each setting. The average execution time of training for each setting is shown in Figure 3.

We can observe from the figure that the execution time is highly dependent on the number of traversals. Skipping training baseline networks also results in a reduction in execution time. Note that our traversals during the training are conducted using 16 processes and that training time will vary depending on the hardware and/or other factors.

We perform self-play between our agents and the random player. We evaluate our agents every 10 iterations. For each evalulation, we conducted 1 000 battles, in which our agent plays first for 500 battles and the random player plays first for the remaining. For each setting, we train our agents

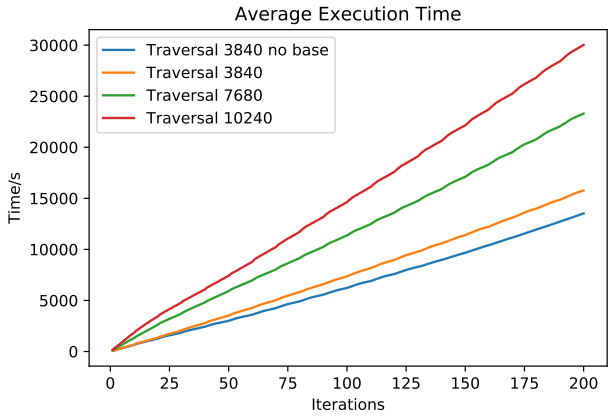

Fig. 3 Average Execution Time

from scratch for 4 independent executions and conduct the evaluations. The result is shown in Figure 4. The win rates are the average of 4 independent executions.

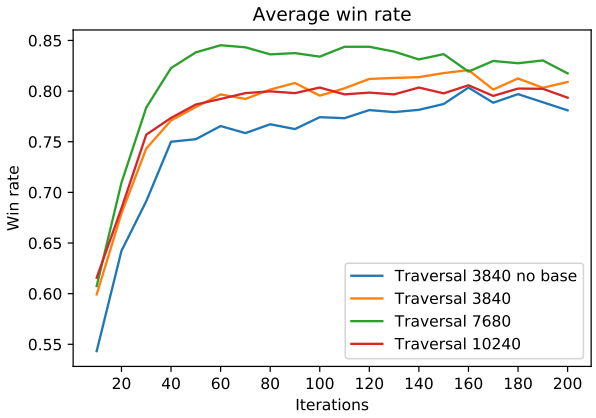

Fig. 4 Average Win Rate

We can observe from the figure that the agents with traversal 7 680 hold the best average performance while the agents without baseline network show a slightly poor performance. Agents with traversal 3 840 and 10 240 perform similarly, but better than that without baseline network. We believe that the baseline network is able to enhance the performance of the agents and accelerate the training process. As for number of traversals, we infer that if the number of traversals is too small, the agent does not have enough exploration, which results in insufficient training. On the other hand, if the number of traversals is too large, the buffer is soon filled up with samples, causing lack of the buffer size in early process of training and adds unsteadibility. We also find that after about iteration 60, all of our agents' performance seem to stop improving or even begin to drop. We suppose that the reason may be lack of the buffer size, as when the traversal is set to be 3 840, the buffer is full after about only 30 iterations. The more the traversal is, the earlier the buffer becomes full. We think that there is a balance between traversal and buffer size, however, it is not fully investigated due to limitation of time. We plan to conduct more experiments in the future work to investigate it.

We also make analysis on the performance of independent executions. The result of independent executions are shown in Figure 5, 6, 7 and 8.

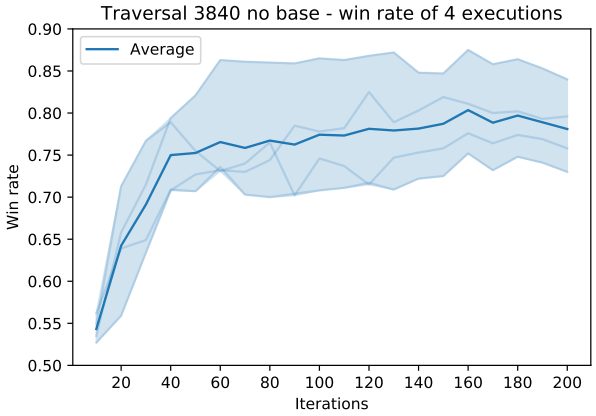

Fig. 5 Win Rate - Traversal 3840, No Baselne Network

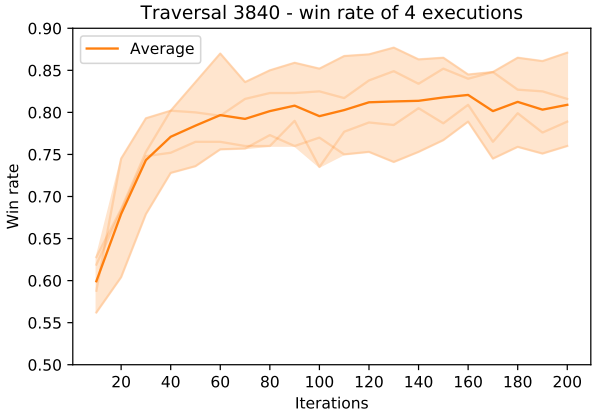

Fig. 6 Win Rate - Traversal 3840

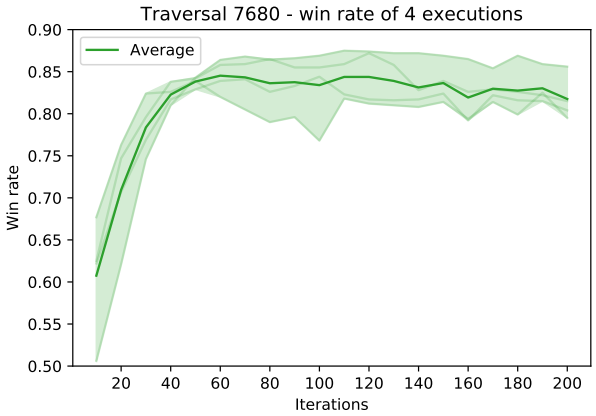

Fig. 7 Win Rate - Traversal 7680

We can observe from the figures that the variance between independent executions under the same setting is still relatively high. As Geister is a huge game, we only explored a

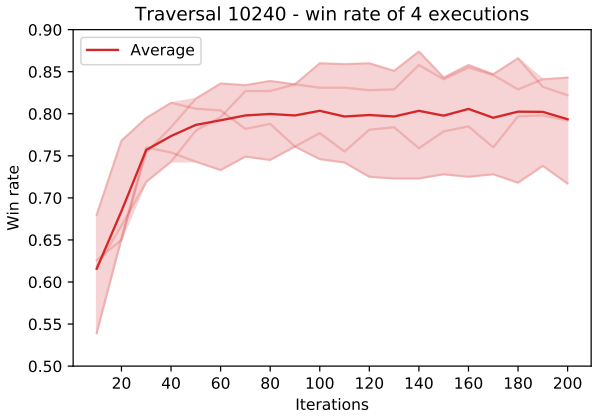

Fig. 8 Win Rate - Traversal 10240

quite small proportion 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 learn a policy strongly dependent on the part of the game explored, which is the similar phenomenon we met in our previous research [7].

The agents with traversal 7 680 show the most stable performance and the best average win rate. We suppose that an appropriate value of traversals will not only derive a better performance but also make the training process more stable. The agents without baseline networks also seem to have the highest variance, which is consistent to our analysis above. However, all the agents achieved a best run of over 85% win rate against the random player in spite of the difference in setting, which is out of our imagination. We will investigate the reason in our future work.

After all, as our agents have an average win rate of about 80%, we conclude that with natural enhancements, DREAM is able to be applied to the board game Geister and achieve an appropriate policy. The baseline network in DREAM adds strength and stablity to the training process and an appropriate number of traversals can also make the training more stable and enhance the performance.

7. Conclusions and Future Work

In this paper, we implemented DREAM and added enhancements to apply it to the board game Geister. We set different numbers of traversals per iteration, and also investigated the effectiveness of the baseline network. We trained different agents under different settings, and evaluated them by self-play against the random player. Our DREAM agents all had an average win rate of about 80%, from which we propose that the DREAM algorithm is able to be applied to the board game Geister with necessary enhancements and generate an appropriate policy for the game. From the anaylsis of the results, we infer that the baseline network accelerates and stabilizes the training progress. We also make an assumption that there is a balance between the traversal and buffer size.

For future work, we plan to investigate the balance between the traversal and buffer size. Also, we will investigate the reason that the best run for each setting performs almost the same. Finally, our network architectures and hyperparameters are not finely tuned due to the limitation of time. We plan to find more efficient architectures and hyperparameters.

References

[1] Zinkevich, M., Johanson, M., Bowling, M. and Piccione, C.: Regret Minimization in Games with Incomplete Information, *Advances in Neural Information Processing Systems*, 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 (2015).

[3] Brown, N., Lerer, A., Gross, S. and Sandholm, T.: Deep counterfactual regret minimization, *International Conference on Machine Learning*, pp. 793–802 (2019).

[4] Steinberger, E.: Single deep counterfactual regret minimization, *arXiv preprint arXiv:1901.07621* (2019).

[5] Steinberger, E., Lerer, A. and Brown, N.: DREAM: Deep Regret minimization with Advantage baselines and Model-free learning (2020).

[6] Chen, C. and Tomoyuki, K.: Acquiring Strategies for the Board Game Geister by Regret Minimization, *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 1–6 (2019).

[7] Chen, C. and Tomoyuki, K.: Utilizing History Information in Acquiring Strategies for Board Game Geister by Deep Counterfactual Regret Minimization, *Game Programming Workshop 2018 Proceedings*, Vol. 2019, pp. 20–27 (2019).

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

[9] 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).

[10] Schmid, M., Burch, N., Lanctot, M., Moravcik, M., Kadlec, R. and Bowling, M.: Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 2157–2164 (2019).

[11] Brown, N. and Sandholm, T.: Solving imperfect-information games via discounted regret minimization, *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 1829–1836 (2019).