JAIST Repository
https://dspace.jaist.ac.jp/

Title	Analyses of Tabular AlphaZero on NoGo
Author(s)	Hsueh, Chu-Hsuan; Ikeda, Kokolo; Nam, Sang-Gyu; Wu, I-Chen
Citation	The 2020 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2020)
Issue Date	2021-03
Type	Conference Paper
Text version	author
URL	http://hdl.handle.net/10119/18244
Rights	This is the author's version of the work. Copyright (C) 2021 IEEE. 2020 International Conference on Technologies and Applications of Artificial Intelligence (TAAI). DOI: 10.1109/TAAI51410.2020.00054. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Description	The 2020 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2020), December 2020

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Analyses of Tabular AlphaZero on NoGo

Chu-Hsuan Hsueh* , Kokolo Ikeda*†, Sang-Gyu Nam*, and I-Chen Wu‡§¶
*School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan
Emails: {hsuehch, kokolo, howzen}@jaist.ac.jp
†Division of Transdisciplinary Sciences, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan
‡Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
Email: icwu@cs.nctu.edu.tw
§Pervasive Artificial Intelligence Research (PAIR) Labs, Hsinchu, Taiwan
¶Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan

***Abstract*—The AlphaZero algorithm has been shown to achieve superhuman levels of plays in chess, shogi, and Go. This paper presents analytic investigations of the algorithm on NoGo, a variant of Go that players cannot capture the opponents' stones. More specifically, lookup tables are employed for learning instead of deep neural networks, referred to as tabular AlphaZero. One goal of this work is to investigate how the algorithm is influenced by hyper-parameters. Another goal is to investigate whether the optimal plays and theoretical values can be learned. One of the hyper-parameters is thoroughly analyzed in the experiments. The results show that the tabular AlphaZero can learn the theoretical values and optimal plays in many settings of the hyper-parameter. Also, NoGo on different board sizes is compared, and the learning difficulty is shown to relate to the game complexity.**

I. Introduction

The AlphaZero algorithm [1] is one of the most important breakthroughs recently in the development of game-playing programs. It is a kind of reinforcement learning algorithm that learns from self-play games and does not need game-specific knowledge other than game rules. Silver et al. [1] successfully applied the algorithm to chess, shogi, and Go, where superhuman levels of plays were achieved.

With the promising results, however, the learning requires a huge amount of computation resources, making it difficult to have a thorough investigation of the AlphaZero algorithm. Especially, the algorithm contains several hyper-parameters, while Silver et al. [1] did not show how each hyper-parameter influenced the algorithm. To thoroughly investigate the hyper-parameters, employing games in smaller scales is a feasible way. Even more, if the employed game is strongly-solved (i.e., knowing the win/loss results of all positions), the knowledge learned by the algorithm can be directly compared with the theoretical values and optimal plays to have an objective evaluation. The analyses on smaller-scaled games may provide insights about normal-scaled ones.

Hsueh et al. [2] analyzed the algorithm on a strongly-solved variant of a two-player stochastic game, Chinese dark chess. They employed lookup tables instead of deep neural networks to learn each position's policy (i.e., the probability distribution of moves) and value. Lookup tables were expected to be able to avoid the possible information loss from feature extraction, though the learning was less efficient since even similar positions were learned separately. Their experiments showed that the tabular AlphaZero learned the optimal plays and the theoretical values under many hyper-parameter settings.

Still, it is worthy of studying more different types of games to understand the influence of hyper-parameters, aiming to provide insights about the hyper-parameter selection. This paper targets on a two-player zero-sum deterministic game called NoGo [3], whose rules are similar to Go. Two players, Black and White, alternatively place stones at the intersections on the board, and Black plays first. Two specific rules for NoGo are (a) moves that capture opponent's stones are forbidden and (b) a game ends when a player on his/her turn cannot put stones on the board, and the player loses. The differences make the strategy to play NoGo quite different from Go.

In this work, NoGo on different board sizes as different problem scales is investigated. The experiments study one hyper-parameter in the tabular AlphaZero called c_{puct}, which determines the degree of exploration in tree search for generating self-play games. The self-play games are then analyzed to figure out how the quality as learning data is influenced by c_{puct}. Besides, different board sizes are compared, showing the learning difficulty generally relates to the game complexity.

The rest of this paper is organized as follows. Section II reviews background knowledge, including NoGo and the AlphaZero algorithm. Section III presents the application of the tabular AlphaZero to NoGo with smaller boards. Section IV describes the experiments and analyses. Finally, Section V makes concluding remarks.

II. Background

This section introduces the game NoGo in Subsection II-A and reviews the AlphaZero algorithm in Subsection II-B.

A. NoGo

(a) (b)

Fig. 1. (a) A capture example and (b) a terminal game of 3×3 NoGo.

NoGo [3] is a two-player zero-sum deterministic game with perfect information. The game is a combinatorial game variant

from Go. The board size can be any $m \times n$ rectangles or squares, usually not bigger than 9×9. The rules generally follow those of Go where Black and White alternatively place stones on the board, and Black is the first player. However, players in NoGo cannot capture the opponent's stones. For example, Black in Fig. 1a cannot play at point A to capture the white stone as in Go. When the player to play cannot put stones on the board, the game ends and the player loses. Fig. 1b shows a terminal game on the 3×3 board. It is White's turn, but both the top-right corner (capture) and the bottom-left corner (suicide) cannot be played. Thus, the winner is Black.

Since NoGo remains many similar properties to Go while the complexity is relatively low, it has been employed as a test bench for research. For example, Hearn and Müller [4] adopted an open-sourced Go program based on Monte-Carlo tree search (MCTS) named Fuego [5] to play NoGo. With only a few modifications, the program was the strongest one in 2011-2012. Lan et al. [6] proposed a new variant of MCTS and showed effectiveness through NoGo. They also incorporated the proposed MCTS into the AlphaZero algorithm.

B. The AlphaZero Algorithm

The AlphaZero algorithm [1] is a kind of reinforcement learning trained by self-play games. Deep neural networks (DNNs) were employed to learn good policies (i.e., the probability distribution of selecting moves) and values for given positions. DNNs were incorporated into tree search to get better policies. The search results were then used as the training data of DNNs. When DNNs got cleverer, the search also produced better results. By repeating these, the algorithm learned to play games well with only knowing the rules.

More specifically, the tree search was a variant of MCTS. The MCTS ran N_{sim} simulations for each move, where each simulation contained three phases, selection, expansion, and backpropagation. In the selection phase, a path was traversed from the root to a leaf following the PUCT algorithm,

$$\underset{a}{\operatorname{argmax}} \{ \frac{W(s,a)}{N(s,a)} + c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \}, \quad (1)$$

where $W(s,a)$ is the total value of move a for node s, $N(s,a)$ the visit count, $P(s,a)$ the prior probability of selecting a at s, and c_{puct} a coefficient influencing the exploration. The expansion phase expanded the leaf node s_L and initialized each move a to $N(s_L,a) \leftarrow 0$, $W(s_L,a) \leftarrow 0$, and $P(s_L,a) \leftarrow p_a$, where p_a is the probability of a obtained from the DNN. In the backpropagation phase, the traversed nodes were updated by $N(s,a) \leftarrow N(s,a)+1$ and $W(s,a) \leftarrow W(s,a)+v$, where v is the value for the position at s_L obtained from the DNN. After all simulations finished, the MCTS's policy $\boldsymbol{\pi}$ for the moves at the root s_0 was calculated, which was in proportion to the visit counts $\pi_a \propto N(s_0,a)$. The moves to be really played in the self-play games during learning were sampled according to $\boldsymbol{\pi}$. The policies $\boldsymbol{\pi}$'s from MCTS and the game outcomes (wins, losses, or draws) in the self-play games were saved and served as the training data for the DNN.

III. AlphaZero for Small NoGo

The methods and the results for solving NoGo on small boards and the application to the tabular AlphaZero are introduced in Subsections III-A and III-B, respectively.

A. Solving Small NoGo

NoGo on the 5×5 board was weakly-solved as Black's win [7], i.e., knowing the winner and the winning strategy from the initial position. Other smaller boards can be strongly solved by retrograde analysis [8], i.e., knowing for each position whether it is a win or loss assuming both players play optimally. The algorithm starts from terminal positions such as Fig. 1b and then gradually updates the win/loss information toward the initial position. A position is a win if one of the successive moves leads to a win from the same player's view; otherwise, it is a loss since all successive moves have been proven to lose. In addition to wins and losses, the shortest wins and longest losses can also be calculated.

TABLE I
Theoretical value and state-space complexity of small NoGo

Board size	1×12	2×6	3×4	1×13	1×14
Theoretical value	B (9)	B (9)	W(10)	B(9)	B(9)
State space	34,747	81,493	87,361	92,996	249,421

This study targets on five board sizes, 1×12, 2×6, 3×4, 1×13, and 1×14. To compare the learning on different problem scales, the board sizes are selected so that they have similar numbers of intersections on the boards and numbers of moves to win. The theoretical value of each board size and the state-space complexity are listed in Table I. State-space complexity is the number of possible positions that can be reached from the initial position, i.e., the empty board. In the row of theoretical value, B and W mean Black's win and White's win, respectively. The number of moves to win from the initial position is also included in the parentheses.

B. Tabular AlphaZero

Following the research line as Hsueh et al. [2], this paper replaces the DNNs in the AlphaZero algorithm with lookup tables, referred to as the tabular AlphaZero. To make the paper self-contained, the mechanism is reviewed as follows. Each position has a unique entry in the lookup table containing the policy and the value. A policy is a vector of real numbers $\hat{\boldsymbol{p}}$. The vector size for NoGo is the same as the board size $m \times n$, similar to the design for Go except that there is no pass move. The probability of each move a is calculated by the softmax function $p_a = e^{\hat{p}_a}/\sum_b e^{\hat{p}_b}$. A position's value is a real number $\hat{v}$, which is scaled to the range of $[0,1]$ by the sigmoid function $v = 1/(1+e^{-\hat{v}})$.

The lookup tables are trained by a synchronous version of the tabular AlphaZero, as shown in Algorithm 1. The first step is to initialize the policies and values in the lookup table to random values. The process of generating self-play games and optimizing policies and values is then repeated for N iterations. In each iteration, M self-play games are generated

Algorithm 1 Tabular synchronous AlphaZero

```
Initialize the lookup table
repeat
    Collect M self-play games
    Optimize by the most recent K × M game (if any)
until N iterations
```

by employing the MCTS as described in Subsection II-B. To optimize the lookup table, the most recent $K \times M$ games, if any, are used to collect groups of (s, $\boldsymbol{\pi}$, z) as training data, where s means a position, $\boldsymbol{\pi}$ the probability distribution of moves in proportion to the visit counts $N(s_0, a)$ in MCTS, and z the outcome of the self-play game (1 and 0 for a win and a loss, respectively, from the view of the player to move). In other words, the self-play games within the recent K iterations are used. For a position s, let $\boldsymbol{p}$ and v be the probability distribution and the scaled value in $[0, 1]$ obtained from the lookup table. The policy $\hat{\boldsymbol{p}}$ and value $\hat{v}$ are updated by $\hat{p}_a \leftarrow \hat{p}_a - \beta \cdot (p_a - \pi_a)$ and $\hat{v} \leftarrow \hat{v} - \beta \cdot (v - z) \cdot v \cdot (1 - v)$, respectively, where β is the learning rate. The formulas were derived based on partial derivatives of the cross-entropy loss $-\boldsymbol{\pi}^T \ln \boldsymbol{p}$ and mean-squared error $(v - z)^2$ [9].

An implementation detail is about $W(s, a)/N(s, a)$ in Eq. (1) when $N(s, a) = 0$. Different ways of implementation are expected to obtain different results. Silver et al. [1] treated the value as 0, i.e., a loss [10], meaning that the MCTS was likely to ignore not-visited moves with low selection probabilities $P(s, a)$. Hsueh et al. [2] also implemented in the same way. In this paper, different implementation is tried, treating $W(s, a)/N(s, a)$ as ∞. Under this implementation, not-visited moves are always selected first. Since the employed games have relatively few legal moves per state, exploring all moves for a state is feasible. One more reasonable way of implementation is to treat the value as 0.5, the middle point between a win and a loss. Comparing different implementation is left as a future research direction.

IV. Experiments and Analyses

In this section, the experiment settings are described in Subsection IV-A. The results of different c_{puct} values and the analyses of MCTS's policies and game outcomes in self-play games are presented in Subsections IV-B and IV-C. Discussions on the influence of c_{puct} are made in Subsection IV-D. Finally, Subsection IV-E compares different board sizes.

A. Experiment Settings

In the experiments, the following settings were the same for all board sizes, selected according to some preliminary experiments. The lookup tables were initialized by a normal distribution with the mean of 0 and the variance of 0.01. The learning lasted for 50 iterations, where each generated 500 self-play games and used the latest 1,000 games to update the lookup table (i.e., $N = 50$, $M = 500$, and $K = 2$ in Algorithm 1). In self-play games, the MCTS ran 800 simulations per move, and the α and the ε for the Dirichlet noise[1] were 1.5 and 0.25, following the settings or the suggestions by Silver et al. [1]. For the optimization of lookup tables, the learning rate β was set to 1.

To know how well the learned policies and values were, two metrics were employed. First, for evaluating policies, the win rates of 10,000 games against the optimal player were collected, denoted by $\mathrm{WR_{OPT}}$. In more detail, the optimal player won immediately when reaching winning positions. For losing positions, since all moves lead to losses, one move was randomly selected. Because of the randomness, even for deterministic games such as NoGo, not all games result in the same ones. $\mathrm{WR_{OPT}}$ could reflect the robustness of the policies. The lookup table player (no search) always played the highest-probability moves based on the policy, regardless of whether the moves were legal. If illegal moves were played, the player lost immediately. Also, it only played as the color of the winner for the initial position. Namely, the lookup table player played as Black for 1×12, 2×6, 1×13, and 1×14 but as White for 3×4. Thus, the highest $\mathrm{WR_{OPT}}$ that could be reached was 100%, meaning that the learned policies without search played as good as the optimal player.

Second, for evaluating learned values, only the values of the initial positions ($\mathrm{V_{init}}$) were considered. It was a fair indicator over different hyper-parameters in the sense that the initial positions were updated for the same number of times. A $\mathrm{V_{init}}$ close to the theoretical value means that the game's theoretical value is learned, though the values of other positions are more important when considering move selection.

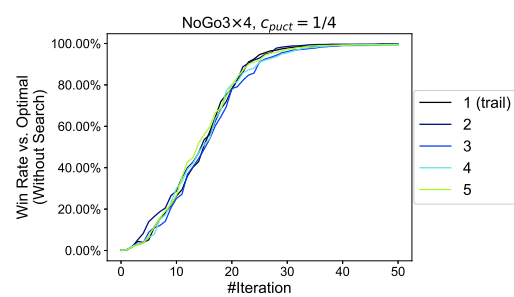


Fig. 2. $\mathrm{WR_{OPT}}$ in different trials for 3×4 NoGo with $c_{puct} = 1/4$.

For each setting, several trials with different random seeds were conducted, but no much differences were observed. An example of $\mathrm{WR_{OPT}}$ training curves for 3×4 NoGo with $c_{puct} = 1/4$ is shown in Fig. 2. Thus, only one trial for each setting is presented in the following subsections.

B. Results on Policies and Values without Search

In this study, the focuses were put on c_{puct}, which was expected more important since it also influenced the effectiveness of Dirichlet noises. A wide range of values, 0, 1/512, 1/256, ..., 1/2, 1, 2, ..., and 16, were tested. The training curves of $\mathrm{WR_{OPT}}$ with some selected c_{puct}'s for board sizes of 2×6, 1×14, and 3×4 are depicted in Fig. 3. The results of

[1] Introduced by Silver et al. [1] to the moves at the root node to ensure all legal moves might be tried, containing two hyper-parameters α and the ε

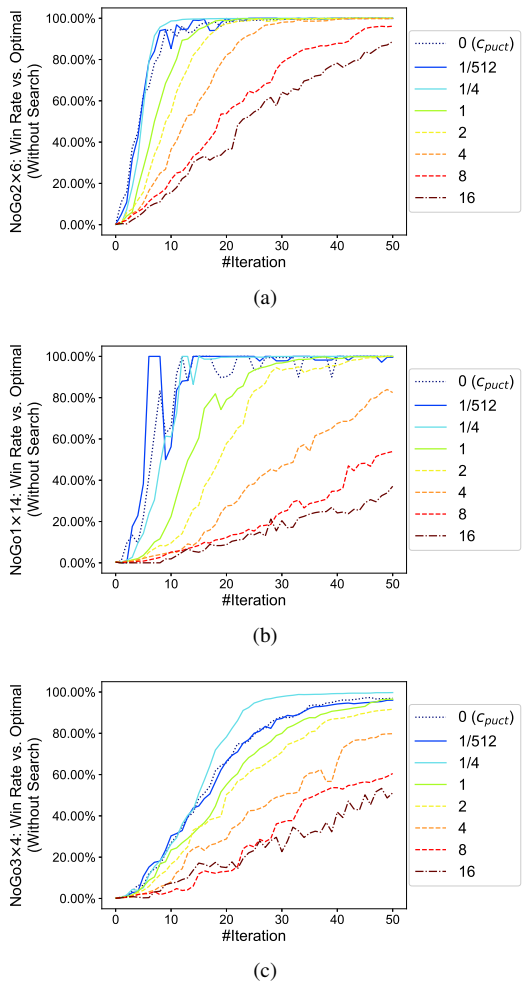


Fig. 3. Training curves of $\mathrm{WR_{OPT}}$ for (a) 2×6, (b) 1×14, and (c) 3×4 NoGo with different c_{puct}.

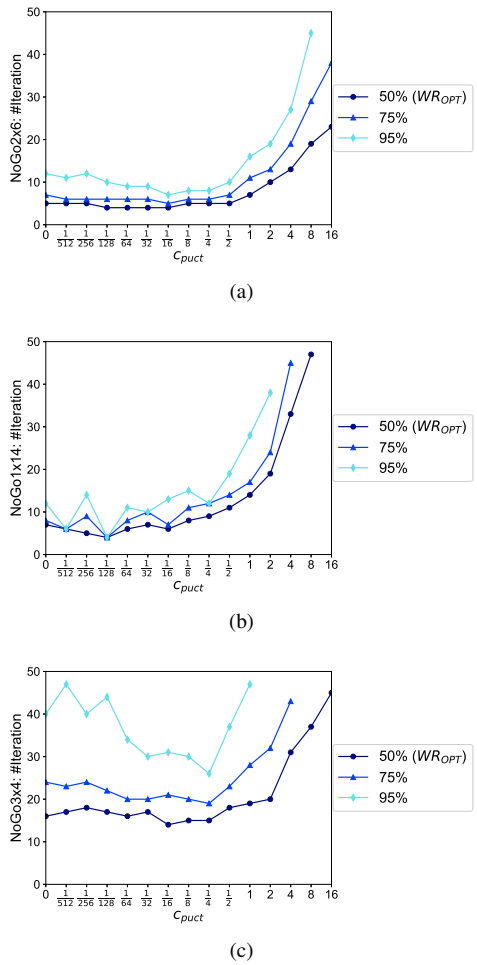


Fig. 4. Learning speed of different c_{puct} for (a) 2×6, (b) 1×14, and (c) 3×4 NoGo.

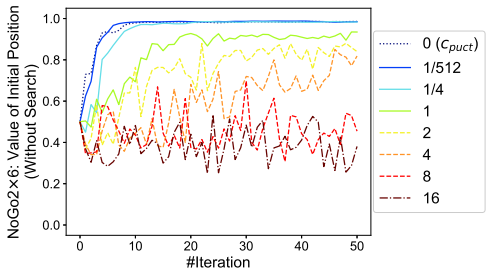


Fig. 5. Training curves of $\mathrm{V_{init}}$ for 2×6 NoGo with different c_{puct}.

1×12 and 1×13 were omitted due to the page limit. From the results, $c_{puct} < 1$ had similar curves. Many settings converged to $\mathrm{WR_{OPT}}$'s of 100%, indicating that the optimal plays were well learned. Interestingly, $c_{puct} = 0$, i.e., no explorations in MCTS, could also learn the optimal plays, though the curve was slightly unstable for 1×14 (Fig. 3b). On the other hand, too-high c_{puct}'s had worse results and learned much slower.

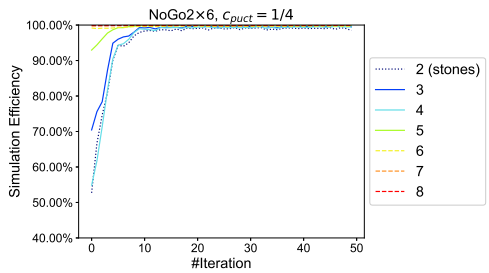
Fig. 4 shows another view of the $\mathrm{WR_{OPT}}$ training curves, namely, learning speeds. More specifically, the learning speed was defined by the number of iterations needed to achieve a given $\mathrm{WR_{OPT}}$. Smaller numbers mean higher learning speed and *vice versa*. For the case that the given $\mathrm{WR_{OPT}}$ was not achieved within 50 iterations, the data point was not drawn. Similar conclusions could be made as to the $\mathrm{WR_{OPT}}$ training curves. First, too-high c_{puct}'s made the learning slow. Second, $c_{puct} < 1$ had similar results, especially for 1×12, 2×6, and 1×13. Third, too-low c_{puct}'s for 1×14 and 3×4 also learned slower, especially 3×4.

Fig. 5 depicts the training curves of $\mathrm{V_{init}}$ with some selected c_{puct}'s for the board size 2×6. The theoretical value is 1 (Black's win). Other board sizes had similar growing trends for c_{puct}, except that $\mathrm{V_{init}}$ for 3×4 went to 0 (White's win). With lower c_{puct}, the learned values converged to the theoretical ones. For $c_{puct} \geq 1$, the convergence trend became even unclearer as c_{puct} increased. Comparing the results of high c_{puct}'s with policies, $\mathrm{WR_{OPT}}$'s still gradually approached to 100%, while nothing was learned for $\mathrm{V_{init}}$'s.

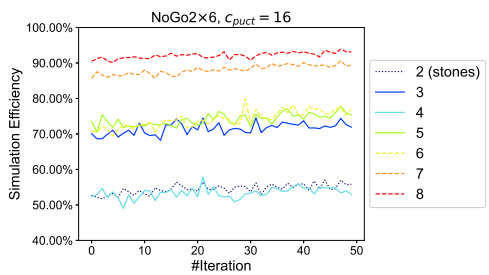
C. Analyses of Self-Play Games

Since the hyper-parameters of MCTS directly influence the self-play games and also the learning on policies and values, it is worthy of studying how MCTS's policies and the

outcomes of self-play games are influenced. This subsection takes c_{puct} as an example and analyzes the self-play games in each iteration. Different board sizes had similar results, and only 2×6 is presented. Also, it was expected that search results would become more accurate as the games got closer to the ends. In NoGo, since stones are not removed from the board as long as they are played, the game progress can be observed by counting the number of stones on the board. In the following analyses, NoGo positions with k stones on the board are denoted by k-stone positions.



(a)



(b)

Fig. 6. Simulation efficiency for 2×6 NoGo with c_{puct} (a) 1/4 and (b) 16.

The first analysis metric for self-play games is the simulation efficiency in searching best moves of given positions, which is the proportion of simulations from the root that are spent on the best moves, i.e., $(\Sigma_{s \in S} \Sigma_{a \in A_s^*} N(s,a))/(\Sigma_{s \in S} \Sigma_{a \in A_s} N(s,a))$ where S is the set of positions in the self-play games, A_s the set of legal moves at state s, and A_s^* the set of best moves at state s. Best moves meant winning moves in winning positions and all moves in losing positions. The simulation efficiency of the self-play games for $c_{puct} = 1/4$ and 16 are shown in Fig. 6. The curves of the 0-stone and 1-stone positions are omitted since all moves in those positions are the best, which made the simulation efficiency be 100% for all iterations. From the figures, it could be observed that the simulation efficiency generally became higher when games were going to end. For $c_{puct} = 1/4$, the simulation efficiency soon converged close to 100%, while some small portions of simulations were still spent on non-best moves. Lower c_{puct}'s had similar results or converged sooner. For $c_{puct} = 16$, too many simulations were spent on non-best moves, making the learning inefficient.

The second analysis metric is the game outcome correctness, which is the proportion of positions that the game outcomes from the positions in the self-play games match the theoretical

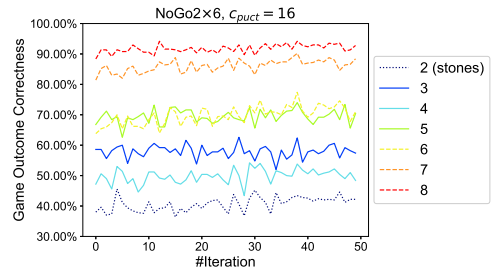
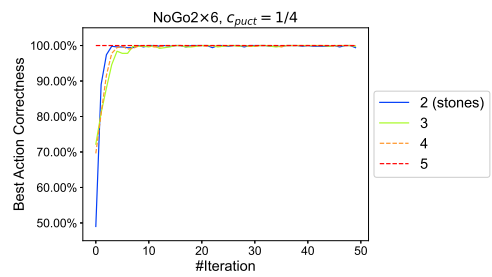
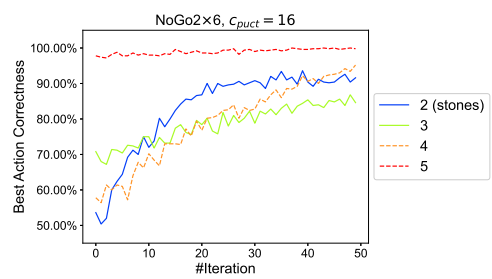


Fig. 7. Game outcome correctness for 2×6 NoGo with $c_{puct} = 16$.

game outcomes, i.e., $|\{s \in S | z_s = v_s\}|/|S|$ where S is the set of positions in the self-play games, z_s the outcome from s in the corresponding self-play game, and v_s the theoretical win/loss of s. The results of game outcome correctness were similar to those on simulation efficiency. For $c_{puct} = 1/4$, the curves were close to Fig. 6a and were omitted. The correctness converged close to 100%, where incorrect ones in later iterations mainly came from exploration. $c_{puct} = 16$, shown in Fig. 7, explored too much and thus failed to provide useful information for learning values.



(a)



(b)

Fig. 8. Best action correctness for 2×6 NoGo with c_{puct} (a) 1/4 and (b) 16.

The third analysis metric is the best action correctness, which is the proportion of positions that the moves most visited in the MCTS belong to the best moves, i.e., $|\{s \in S | a_s \in A_s^*\}|/|S|$ where S is the set of positions in the self-play games, a_s the most-visited move at s in the corresponding self-play game, and A_s^* the set of best moves at s. Fig. 8 depicts the best action correctness for $c_{puct} = 1/4$ and 16. For $c_{puct} = 1/4$, the correctness soon converged to 100%, while $c_{puct} = 16$ approached to 100% much slower. The results of $c_{puct} = 16$ were different from the previous two metrics in that increasing trends could be observed.

D. Discussions on the Exploration Coefficient

Higher c_{puct}'s mean higher degrees of exploration during MCTS. Namely, less-visited moves are more likely to be selected during search even when the expected value $W(s,a)/N(s,a)$ is low. With a too-high c_{puct}, MCTS evenly divides the simulations to the moves, which makes the search inefficient, as shown in Fig. 6b. Since moves in self-play games are selected in proportion to the visit counts, a too-high c_{puct} also increases the probabilities to select non-best moves, which makes the game outcome correctness low, even after learning, as shown in Fig. 7. The analysis provides an explanation to Fig. 5 that V_{init} has no clear trends when c_{puct} is too high. Even so, the MCTS could still successfully identify (one of) the best move(s), as shown in Fig. 8b. As a result, high c_{puct}'s, though learned slower, could still approximate to the optimal plays, as shown in Fig. 3.

On the other hand, with a too-low c_{puct}, MCTS may overlook the best moves due to few simulations of bad results and finally select sub-optimal moves. Fig. 6a shows that most simulations are concentrated on best moves with $c_{puct} = 1/4$. Since the employed games have relatively few moves and small game trees, too-low c_{puct}'s might still be no problems. It might also depend on the implementation of $W(s,a)/N(s,a)$ when $N(s,a) = 0$, as discussed in Subsection III-B. This issue requires further studies and is left as future research.

E. Comparison of Different Board Sizes

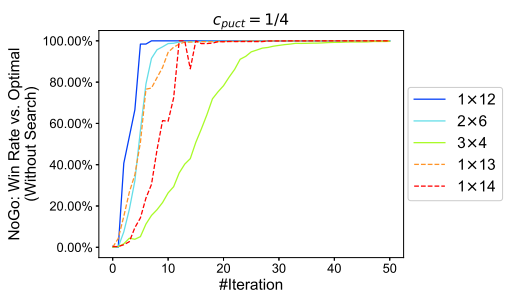


Fig. 9. Training curves of WR_{OPT} for different board sizes ($c_{puct} = 1/4$).

Fig. 9 shows the training curves of WR_{OPT} with $c_{puct} = 1/4$ for different board sizes. Other promising c_{puct}'s had similar results and are omitted. Considering the learning speed, the difficulty of learning from the highest to the lowest was: 3×4 > 1×14 > 1×13 ≈ 2×6 > 1×12. Except for 3×4, the order generally followed their state-space complexity (i.e., 249,421 > 92,996 ≈ 81,493 > 34,747). All these board sizes are Black's 9-move win. The state-space complexity of 3×4 is 87,361, between those of 1×13 and 2×6; however, it was the most difficult one to learn. Since 3×4 is White's 10-move win, it was suspected that the result was still reasonable. One possible explanation is that a 3×4 NoGo can be imagined as twelve sub-problems that win in nine moves. To learn 3×4 NoGo well, all the twelve sub-problems should be properly learned, though it does not need as much as twelve times of self-play games since these sub-problems are dependent.

V. Conclusions and Future Work

In this paper, the tabular AlphaZero is applied and analyzed on strongly-solved variants of NoGo. The hyper-parameter c_{puct} (exploration coefficient in PUCT) is thoroughly investigated through experiments. The results show that the optimal plays and theoretical values can be learned under relatively low degrees of exploration (low c_{puct}), though the phenomena require further study to clarify whether it relates to the problem scales, the implementation of Eq. (1) when $N(s,a) = 0$, or other reasons. On the contrary, when c_{puct} goes too high, the algorithm can learn nothing about the values, though the policies can still slowly approach the optimal plays. In addition, five board sizes are compared, where the learning difficulty is shown to relate to the game complexity generally.

For future research, three promising directions are considered. The first is to investigate other hyper-parameters such as α and ε for Dirichlet noises and N_{sim} (number of simulations for each move). The second is to compare different implementation of $W(s,a)/N(s,a)$ when $N(s,a) = 0$. The third is to replace lookup tables with neural networks and investigate whether the optimal plays and theoretical values can still be learned. The above can be done for games other than NoGo. By comparing different games, the results are expected to provide insights or guidelines about the hyper-parameter selection and different ways of implementation when applying to other games.

Acknowledgment

This research is partially supported by Japan Society for the Promotion of Science (JSPS) under contract number 20K19946.

References

[1] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 352, no. 6419, pp. 1140–1144, 2018.

[2] C.-H. Hsueh, I.-C. Wu, J.-C. Chen, and T.-s. Hsu, "AlphaZero for a non-deterministic game," in *Proc. Conf. Technol. and Appl. of Artif. Intell. (TAAI 2018)*, Taichung, Taiwan, Nov. 2018, pp. 116–121.

[3] "NoGo — Board Game — BoardGameGeek," accessed: Oct. 18, 2020. [Online]. Available: https://boardgamegeek.com/boardgame/151419/nogo

[4] B. Hearn and M. Müller, "The BobNoGo Program," 2011, accessed: Oct. 18, 2020. [Online]. Available: https://webdocs.cs.ualberta.ca/~mmueller/nogo/BobNoGo.html

[5] M. Enzenberger, M. Müller, B. Arneson, and R. Segal, "Fuego—an open-source framework for board games and Go engine based on Monte Carlo tree search," *IEEE Trans. Comput. Intell. AI in Games*, vol. 2, no. 4, pp. 259–270, Dec. 2010.

[6] L.-C. Lan, W. Li, T.-H. Wei, and I.-C. Wu, "Multiple policy value Monte Carlo tree search," in *Proc. the Twenty-Eighth Int. Joint Conf. on Artif. Intell., IJCAI-19*, Aug. 2019, pp. 4704–4710.

[7] P. She, "The designed and study of NoGo program," Master's thesis, National Chiao Tung University, 2013.

[8] K. Thompson, "Retrograde analysis of certain endgames," *ICGA J.*, vol. 9, no. 3, pp. 131–139, 1986.

[9] C.-H. Hsueh, "On strength analyses of computer programs for stochastic games with perfect information," Ph.D. dissertation, Nat. Chiao Tung Univ., Hsinchu, Taiwan, 2019. [Online]. Available: https://hdl.handle.net/11296/ku48z7

[10] "Alphazero news - Page 8 - TalkChess.com," Dec. 2018, accessed: Oct. 18, 2020. [Online]. Available: http://talkchess.com/forum3/viewtopic.php?f=2&t=69175&start=70&sid=8eb37b9c943011e51c0c3a88b427b745