Research on Board Position Data Annotation Methods (Postprint)

Authors: Ding Meng, Zhang Yipeng, Li Shuqin

Date: 2018-12-13T00:00:00+00:00

Abstract

The success of AlphaGo has drawn widespread attention to deep learning methods in the field of computer game playing. However, supervised training based on deep learning models relies on large amounts of high-quality labeled data, yet for many niche board games in computer game competitions, there is a lack of human game records to serve as training samples. Therefore, how to generate a reasonably labeled position dataset before using deep learning models is a problem worthy of research and investigation. For the Dots and Boxes game problem, a data hash deduplication and position labeling method is proposed. Based on the characteristics of game positions at different turn stages, Dots and Boxes position samples with different turn counts are collected and labeled via Alpha-Beta exhaustive search, backtracking annotation, parallelized MCTS algorithm annotation, and symmetric expansion techniques. The experiment obtained a dataset containing 15,000,000 labeled Dots and Boxes position samples, providing a guarantee for supervised training of deep learning models for Dots and Boxes. Furthermore, the proposed method also provides valuable reference for the acquisition of training data for other game types.

Full Text

Preamble

Vol. 37 No. 2
Application Research of Computers
Accepted Paper

Study on Chessboard Configuration Data Calibration Methods

Ding Meng a,b, Zhang Yipeng a,b, Li Shuqin a,b
(a. School of Computer Science; b. Joint Laboratory of Sensing & Computa-

tional Intelligence, Beijing Information Science & Technology University, Beijing 100101, China)

Abstract: The success of AlphaGo has drawn widespread attention to deep learning methods in the field of computer games. Supervised training based on deep learning models relies on large volumes of high-quality labeled data. However, many niche computer game competitions suffer from a lack of human game records to serve as training samples. Consequently, generating a reasonably labeled configuration dataset before applying deep learning models represents a significant research problem. This paper proposes a data hashing deduplication and configuration labeling method specifically for the game of Dots and Boxes. By leveraging the characteristics of game states at different stages, we collect and label Dots and Boxes position samples across various move numbers through Alpha-Beta exhaustive search, back-tracing calibration, parallelized MCTS algorithm labeling, and symmetric extension techniques. Our experiments yielded a dataset containing 15 million labeled Dots and Boxes position samples, providing a foundation for supervised training of deep learning models. Additionally, the proposed methods offer valuable reference for training data acquisition in other chess variants.

Keywords: data calibration; Dots and Boxes; chessboard configuration; computer games

---

0 Introduction

Computer game playing aims to enable computers to learn human thinking patterns—reasoning, judging, and making rational decisions to play various board games against human opponents or other computers [**?**]. It represents a challenging problem and important research direction in artificial intelligence. Numerous experts and scholars in China and internationally are actively researching computer game playing. The International Computer Games Association (ICGA) organizes an annual computer game competition and academic symposium, while the Chinese Computer Game Committee of the Chinese Association for Artificial Intelligence holds an annual National College Student Computer Game Competition and National Computer Game Championship, featuring 17 different board and card games including Dots and Boxes, Surakarta, Go, military chess, international checkers, Landlord (a three-player poker game), and bridge [**?**]. These events have significantly advanced the development of computer game playing worldwide.

Following DeepMind' s introduction of deep learning technology to computer game playing in 2015 [**?**, **?**, **?**], ensemble deep learning methods [**?**] have gained extensive attention and substantial progress in this field. The convolutional neural networks used in AlphaGo essentially employ supervised learning, which generally requires large amounts of labeled samples to achieve good generalization performance. Therefore, generating a reasonably labeled configuration

dataset before training CNNs for niche competition games such as Dots and Boxes constitutes a worthwhile research problem. This paper focuses on Dots and Boxes as a case study to investigate chessboard configuration data calibration methods in depth.

1 Dots and Boxes Configuration Representation and Hash Deduplication

Dots and Boxes is a well-known two-player board game that has been included in the International Computer Olympiad and Chinese Computer Game Competition for many years. The rules for arbitrary board sizes are as follows: In a uniformly distributed rectangular grid of points, two players alternately draw horizontal or vertical lines connecting adjacent points during their turn. If a player completes a unit square formed by four points (i.e., all four sides of that square are drawn), the player captures that box and scores one point, then must continue their turn for another move. The game ends when no more lines can be drawn (all boxes are captured), and the player with the most boxes wins.

We represent the 5$×5*DotsandBoxesboardusingan*11×$11 two-dimensional matrix to abstractly and formally encode information about edges, boxes, and box ownership. As shown in the left two diagrams of Figure 1 [Figure 1: see original paper], blocks marked 'R' represent boxes captured by the red player, while blocks marked 'B' represent boxes captured by the blue player. Each box block has four adjacent edge blocks representing the four sides of that box in the actual board. A mark of '1' indicates an occupied edge, while '0' indicates an unoccupied edge.

When collecting Dots and Boxes sample data using various methods, duplicate samples must be removed to improve dataset quality. To save space and improve time efficiency, we need a highly compressed, unique identifier for each board position. We employ a hash table to record collected positions, efficiently querying the table to filter samples and eliminate duplicates while recording only new, unseen positions.

Specifically, each Dots and Boxes position is uniquely described by five attributes: horizontal edge occupation status, vertical edge occupation status, red player score, blue player score, and current turn ownership. A 5$×$5 board contains 6 rows × 5 columns = 30 horizontal edges and 5 rows × 6 columns = 30 vertical edges. By numbering horizontal and vertical edges row by row and using 30 bits to represent the occupation status of each corresponding edge, we can describe a position' s horizontal and vertical edge status using two integers. The red player score, blue player score, and current turn ownership can each be represented by an integer. Consequently, any Dots and Boxes position can be uniquely represented by a 5-tuple of integers, which serves as a key in the hash table for unique identification. The position shown in Figure 1 corresponds to the hash representation (123860205, 430258988, 1, 2, 0).

Given fixed scores for both players, the specific distribution of captured boxes

may vary, but from the perspective of position evaluation and decision selection, different box occupation patterns are completely equivalent.

2 Design and Implementation of Data Calibration Methods

During gameplay, position types and quantities continuously evolve, requiring different strategies at different stages. Dots and Boxes games typically conclude before move 60. Based on the characteristics of positions at various stages, we propose: (1) an exhaustive search calibration method using Alpha-Beta search for moves 28–59, (2) a back-tracing calibration method for moves 23–30, (3) a parallelized MCTS algorithm for position sample calibration for moves 0–24, and (4) a data augmentation method based on symmetric flipping principles.

2.1 Exhaustive Search Data Calibration Method

Alpha-Beta search [**?**], based on Alpha-Beta pruning minimax search, is widely used in computer game tree search. As an exhaustive search strategy, Alpha-Beta search yields absolutely correct results when the search depth reaches terminal game positions. Additionally, extensive experiments reveal that Dots and Boxes positions, especially near endgames, often have nearly unique optimal moves. Therefore, we can use Alpha-Beta search results as the highest-rated decision for the current position, serving as the strategy label, while the evaluation score serves as the position' s value label.

Our Alpha-Beta search implementation provides position evaluation values in the range [0,1], where advantageous positions receive values near 1, disadvantageous positions near 0, and higher values indicate greater advantage. If a specific position' s evaluation value is *value*, then when the turn ownership flips, the flipped position' s evaluation value becomes $1-value$.

The exhaustive search data calibration method proceeds as follows: a) Randomly generate a position as the current position; b) If the current position's move number exceeds the minimum exhaustive search threshold *nlimit*, proceed to step c); otherwise return to step a); c) Perform Alpha-Beta exhaustive search on the current position and record the search depth *depth*; d) If Alpha-Beta search determines the current position is winning, proceed to step e); otherwise proceed to step f); e) Label the current position' s value as $(1-depth)/(2\times turn\_\{sum\})$, label the best move selected by Alpha-Beta search with probability 1, and all other moves with probability 0; f) Label the current position' s value as $depth/(2\times turn\_\{sum\})$, label the best move selected by Alpha-Beta search with probability 1, and all other moves with probability 0.

Here, $turn\_\{sum\}$ represents the maximum number of moves in the game.

2.2 Back-Tracing Data Calibration Method

After evaluating a position' s advantage or disadvantage through exhaustive search, this evaluation can propagate upward along the unique path from the

game tree root to the current position, inverting with each turn ownership flip, recursively labeling and collecting position samples. Furthermore, when a specific position is evaluated as advantageous, if its predecessor position shares the same turn ownership, that predecessor is also advantageous because the predecessor has an opportunity to transition to the advantageous position.

For early-game positions, performing a complete Alpha-Beta search incurs excessive time cost. However, Dots and Boxes games may feature consecutive scoring moves by one player where turn ownership remains unchanged, allowing advantageous position evaluations to propagate continuously toward the root node.

Therefore, we propose back-tracing data labeling algorithms for both advantage and disadvantage scenarios. The pseudocode is shown in Figure 2 [Figure 2: see original paper]. The GET_{PRECURSORS} method obtains all legal predecessor positions according to Dots and Boxes rules. Using the back-tracing calibration algorithm, position evaluations are obtained through propagation rules, and the propagation path corresponds to recommended decisions for the position. This algorithm yields large quantities of labeled Dots and Boxes position samples for moves 23–30.

2.3 Parallel MCTS Data Calibration Method

Positions before move 23 are extremely difficult to evaluate and label through exhaustive search or back-tracing algorithms. We employ Monte Carlo Tree Search (MCTS) [**?**] to fill this gap. It has been proven that as simulation counts increase, MCTS position evaluations and decision recommendations converge to minimax search results [**?**], though this convergence is slow because accurate MCTS evaluations require substantial simulations.

To efficiently label sufficient position samples using MCTS, we improved the naive MCTS algorithm. Our implementation uses a hash table data structure to store all state nodes in the game search tree, with position hash values as keys and search status of corresponding game tree nodes as values. We added table-level and node-level thread mutex locks to the search status index table, launching multiple threads to execute random simulations in parallel while maintaining a shared search status index table across threads.

The table-level thread mutex lock preemption mechanism is illustrated in Figure 3 [Figure 3: see original paper]. The node-level mutex lock read/write preemption mechanism operates as follows: a) When the node-level thread mutex lock is idle, it can be acquired by either write-status threads or read-status threads; b) Both read-status and write-status threads first perform a waiting loop to confirm the mutex lock is idle; c) After confirming availability, read/write threads begin a second loop attempting to acquire the mutex lock; d) If acquisition fails, return to step b); e) If a read-status thread successfully acquires the lock, it uses the mutex lock' s atomic variable as a read-status thread counter, allowing other read-status threads to share the lock while preventing write-status threads

from acquiring it; f) After all read-status threads release the lock (atomic variable returns to 0), the mutex lock returns to idle state, allowing acquisition by read/write threads (returning to state a); g) If a write-status thread successfully acquires the lock, it uses the mutex lock' s atomic variable as an exclusive flag, preventing any other read/write threads from acquiring the lock until the thread releases it; h) If the node-level thread mutex lock is configured as a yielding mutex during instantiation, state node update requests during lock occupancy are directly ignored; i) If configured as a blocking mutex, state node update requests during lock occupancy are blocked until the mutex lock returns to idle state.

When using parallel MCTS to label Dots and Boxes position samples for moves 0–24, the root node' s simulation win rate serves as the position evaluation, and the simulation win rates of all branch nodes serve as decision scores.

2.4 Symmetric Flip Data Augmentation Method

The Dots and Boxes board is a centrally symmetric square, so positions remain equivalent under various symmetric transformations. By applying symmetric flips to labeled positions, we can multiply the labeled position samples. As shown in Figure 4 [Figure 4: see original paper], each Dots and Boxes position has 8 symmetric variants. Through symmetric flipping, we can expand the labeled position samples by a factor of 8. After filtering through the position sample hash table, a substantial number of additional samples are retained.

3 Experiments

Through Alpha-Beta exhaustive search, back-tracing calibration, parallelized MCTS labeling, and symmetric flip augmentation, we obtained a dataset containing 15 million labeled Dots and Boxes position samples. We implemented Alpha-Beta search and CNN-based deep search algorithms in C++, with the deep model using the Caffe framework. The experimental environment consisted of: g++ 5.4.0 compiler, Ubuntu 16.04 x64 system, Intel® Core™ i7-6700HQ CPU @ 2.60 GHz.

3.1 Search Labeling Example

Consider the position sample shown in Figure 5 [Figure 5: see original paper], where it is Blue' s turn to move. This position' s hash representation is (758027384, 113572561, 3, 2, 1). The Alpha-Beta exhaustive search labeling result for this sample is: win rate 77.5%, best move (type: vertical, row: 1, col: 4).

3.2 Back-Tracing Labeling Example

One back-traced sample derived from the Figure 5 position is shown in Figure 6 [Figure 6: see original paper]. This position' s hash representation is (758027384, 113556177, 3, 1, 1). We label this sample with the same Blue win rate as the

original position. Its best move is (type: vertical, row: 2, col: 2), and its MCTS labeling result is: win rate 71.39%.

Based on the original sample' s labeling results, hundreds of new samples can be back-traced and labeled.

3.3 Symmetric Flip Augmentation Example

Through symmetric flip augmentation, we can obtain 7 position samples equivalent to the original. As shown in Figure 7 [Figure 7: see original paper], position 0 represents the original sample from Figure 5. We label these samples with the same win rate as the original position (Blue win rate 77.5%).

4 Conclusion

By analyzing the rule characteristics of Dots and Boxes, this paper proposes a series of methods for sample data calibration, augmentation, and deduplication, including: an exhaustive search calibration method based on compressed-rule Alpha-Beta search, a data augmentation method based on back-tracing labeling algorithms, a position sample calibration method based on parallelized MCTS algorithms, and a data augmentation method based on symmetric flipping principles. These methods effectively acquire and label sample data from different game stages, providing a foundation for supervised training of deep learning models and offering valuable reference for training data acquisition in other niche competition games.

References

[1] Wang Yajie, Qiu Hongkun, et al. Computer game competition and reform of innovative talent training mode [J]. Experimental Technology and Management, 2016, 33(10): 10-14.

[2] China University Students Computer Game Competition Organizing Committee. General rules national computer competition [EB/OL]. [2018-08-12]. http://computergames.caai.cn/jsgz00.html.

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning [J]. Nature, 2015, 518(7540): 529-533.

[4] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search [J]. Nature, 2016, 529(7587): 484-489.

[5] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge [J]. Nature, 2017, 550(7676): 354-359.

[6] Qiu Xueheng, Zhang Le, Ren Ye et al. Ensemble deep learning for regression and time series forecasting [C]//Proc of IEEE Symposium on Computational Intelligence in Ensemble Learning. 2014: 1-6.

[7] Pearl J. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality [J]. Communications of the ACM, 1982, 25(8): 559-564.

[8] Chaslot G, Bakkes S, Szita I, et al. Monte-Carlo Tree Search: A New Framework for Game AI [C]//Proc of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference. AAAI Press, 2008: 216-217.

[9] Bouzy B. Old-fashioned computer go vs monte-carlo go [C]//Proc of IEEE Symposium on Computational Intelligence and Games. 2008.

Note: Figure translations are in progress. See original paper for figures.