Learning the game of Renju with Neural Network and Tree Search

Guandao Yang, Naxin Chen GY46,NC352@CORNELL.EDU
Ransen Niu, Vaidehi Patel RN329,VHP25@CORNELL.EDU
Yiwei Ni, Hanqing Jiang, Kilian Weinberger YN254,HJ284,KQW4@CORNELL.EDU

Abstract

Most of top-performing AIs in the game of Renju based on tree search with hard-coded heuristics, which requires domain specific knowledge. In this report, we adopt ideas from the neural network architectures of AlphaGo on the game of Renju to alleviate the effort of handcrafting features. We modify the training pipeline in AlphaGo Zero to include supervised signals, which largely ease the computational burden. Additionally, we explore different ways to combine neural network with traditional tree search algorithms. Experiment results show that combining neural network with Monte Carlo tree search is competitive with the top-10 Renju AIs.

1. Introduction

Renju is an extension of the board game five-in-a-row. Some rules are used to control first-hand advantages. First, Renju starts with a curated set of opening to ensure the fairness of the game. Second, the first-hand player has a series of forbidden move as illustrated in Figure 1. With these rules, Renju becomes a much more sophisticated games, and has gained interests from AI developers for years (gom, 2017).

Most top performing Renju AIs are based on tree search with hand-crafted heuristics, which requires lots of domain knowledge. In this report, we adopt ideas from AlphaGo (Silver et al., 2017b; 2016; 2017a) to combine neural network and tree search to create a Renju AI without hand-crafted heuristics. In Section 3, we give detailed explanation of our system, especially how we modify the system from AlphaGo Zero (Silver et al., 2017b) to fit the game of Renju. In particular, we modify the purely reinforcement learning based training pipeline to incorporate supervised learning signals (Section 3.2), and also explore Principal Variation search as an alternative of the Monte Carlo tree search (Section 3.3). Finally, we provide detailed experiments in Section 4 to show the effectiveness of different components in our system, and provide statistics to show that our AI is competitive with the top Renju AIs in the world.

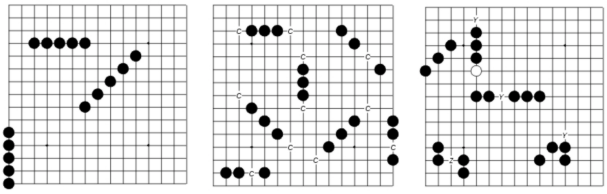

Figure 1. left: Renju possible winning lines; middle: possible open fours with move c (typical winning moves); right: forbidden move example, marked as Y; Different from five-in-a-row, Renju forbids the first-hand player to play the following winning moves: 4-4, 3-3, or overline.

The main contributions of this work are:

1) We show key components of how to successfully combine deep neural networks with tree search to play the game of Renju without hand-crafted heuristics.

2) We conduct extensive experiments to show the effectiveness of different components of our AI system. And we show that our AI is competitive with the strongest heuristic based AIs in the world.

2. Related Work

Currently, there are 7 Renju AIs in the Renju tournament, such as Yixin. To the best of our knowledge, they are all based on tree search or heuristic algorithms. On the other hand, our AI uses a deep neural network to replace the domain-specific heuristic. There has been one previous work on move prediction using deep learning in Gomoku, a simpler version of Renju without forbidden moves (Shao et al., 2016). Several course projects from Stanford use ConvNets and RNNs on similar tasks (Zhao). All of these projects have demonstrated successful training results. Our project uses a more sophisticated network structure adapted from AlphaGo Zero (Silver

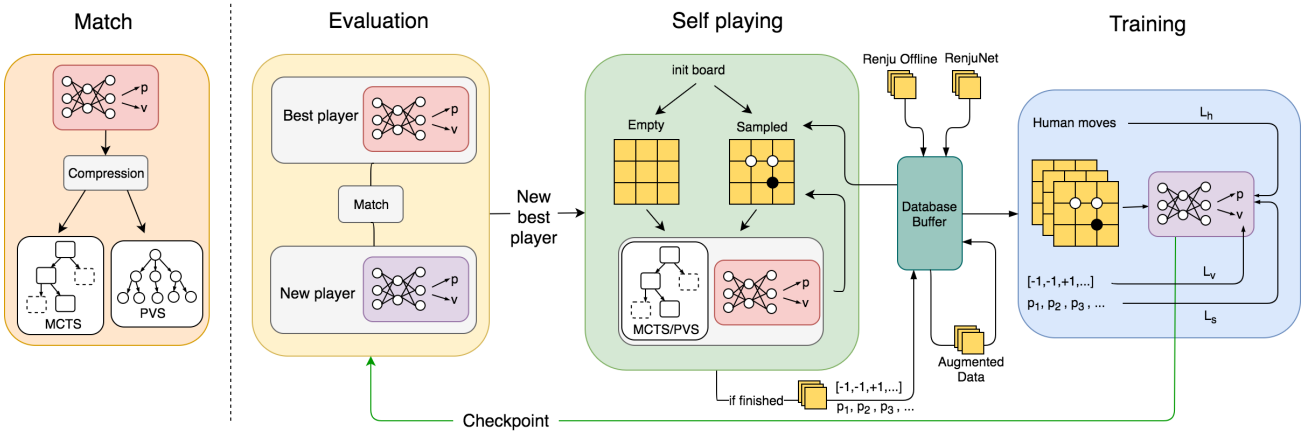

Figure 2. The overall training pipeline of our system contains three stages. Evaluation stage will run matches between two players and select the best one for the self-playing stage. The selected player will generated self-played data and push it into the database buffer. The training stage will update the parameters of the neural network. The best player after the training will be compressed and combined with tree searches while playing with other AI agents.

et al., 2017b) and comprehensive datasets to achieve better training results. There have been several similar attempts to reproduce AlphaGo Master or Zero in Renju or Gomoku(AI-Zone; welkincrowns). None of those projects show successful results competing with heuristic-base AIs.

3. Methods

Our model architecture are inspired by AlphaGo Zero (Silver et al., 2017b). In this section, we will focus on where our methods differ from AlphaGo Zero's.

3.1. Double Dual ResNet

Similar to AlphaGo Zero, we use a dual-head residual network to predict both the policy and the value. We refer to the AlphaGo Zero paper for more detail (Silver et al., 2017b). Due to the fact that there are different restrictions for black stones in Renju, we split our training dataset into two parts : one for the black player (first-hand player) and the other one for the white player. We train two dual ResNet without weight sharing, one on each of dataset. During playing, the network trained in black dataset only predicts the black move, while the network trained in white dataset is used for white move.

3.2. Training Pipeline

Different from the training pipeline of AlphaGo Zero, our neural network is trained by both supervised and unsupervised signals. The unsupervised signal resembles to that of AlphaGo Zero where a tree search is combined with previous best version of neural network to generate training data. But generating such training data requires astronomical amount of computational resources. To our best estimation, it takes 4 1080 Ti GPUs roughly 1700 years to generate the amount of data used in AlphaGo Zero.

To solve this problem, we bootstrap our dataset with human expert data. With these data, we can warm-up the training process with first supervise learning. During supervised learning, the networks is learned to predict human expert's move and the results of human expert's game. We use soft-max cross entropy loss (L_h) for the prediction of human expert's move. At the same time, the pipeline will generate self-playing data for reinforcement learning. The self-playing data contains a policy output from the tree search, so we use a cross entropy loss to teach our networks match the tree search policy. In particular, $L_s = \pi \log(\hat{p})$ where π is the policy ouput from the tree search and $\hat{p}$ is the predicted policy. To train the value function, we use simple mean square error $L_v = (v - z)^2$, where v is the network prediction. z will be +1 if current player wins the game and -1 otherwise.

3.3. Tree Search

Our method use the following two tree searches: monte carlo tree search as used in AlphaGo Zero (Silver et al., 2017b), and Principal Variation search as used in Yixin (Kai, 2017). We refer to the AlphaGoZero paper (Silver et al., 2017b) for details about how to combine Monte Carlo tree search (MCTS) with dual

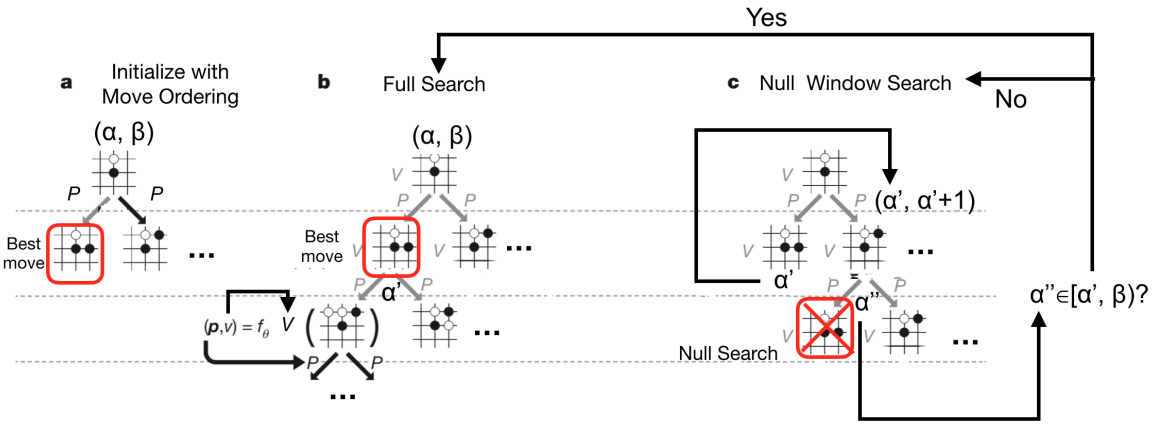

Figure 3. This picture illustrates how we use Principal Variation Search (PVS) with neural network. PVS relies on a move ordering of the current game state. (a) PVS will first perform a full search into the best-move. (b) Once this full search returns, PVS will use the best value obtained so far to perform a null-window search in the second-to-the-best node. (c) If the null window search returns a value that's slightly better than the current best value, then a full search will be performed in the second node. If not, the tree search will go on to perform null window search in the next node in the move ordering. We used our network's predicted policy as our move ordering in PVS.

ResNet. The only modification is that while expanding tree node of one specific stone color, we use the dual ResNet trained with dataset of that color.

Principal Variation search (PVS) is an improved version of mini-max search tree with Alpha-Beta pruning, using move-order and null-window search. It relies on the quality of move ordering, as a good move ordering reducing the chance to perform full search. In our case, we use the probability output p of the policy network as the move ordering. With move ordering, PVS does a full search in the best move/node according to the policy and update α in the search range (α, β) with the best value, $\alpha' = v$, of the child moves/nodes. Afterwards, PVS simply does a null window search in the second-to-the-best move/node with search range $(\alpha', \alpha' + 1)$. If the null window search returns a better value α'' than α', then a full search will be performed. Otherwise, PVS moves to the next best move/node. Figure 3 shows the details of PVS.

4. Experiments

Due to the fact that self-playing data is time-consuming to obtain, self-playing data only contribute to a small fraction of our database. As a result, all of our experiments are done using supervised learning part of the pipeline only, and we postpone reinforcement learning into future work.

4.1. Data

We used two datasets: RenjuNet(Ren) and RenjuOffline(Ren, 2017). RenjuNet contains 50,064 games while RenjuOffline contains 104,948 games. RenjuNet's games are played by human experts across the world in real time. On the other hand, the games in RenjuOffline are slow games played online without time restriction, which is known to have better quality. We removed the first 6 moves from all the games, and generated game states from all the game records. With this method, we obtained a dataset with 3,146,736 game states, all of which annotated with a human expert's move and its game results.

4.2. Evaluation of Policy Network

To evaluate the quality of our policy networks, we conducted an internal tournament among the following models: 1) models trained in the combined dataset with 0,2,4,or 8 residual blocks; 2) models trained in dataset separated by color of the move with 0, 2, or 4 residual blocks. During the tournament, we add a simple move pruning by having the AI to defend an immediate losing move and take an immediate winning move. We report both the validation accuracy of the policy network, the normalized mean square error (i.e. divided by 4 to make it within the range of [0,1]), and the Elo score from the tournament in table 1.

Table 1 shows that deeper models have better performance in validation accuracy, mean square error, and Elo score. Models trained with separated colors in general perform much better than the models trained with two colors combined, which could be a result of the fact that human expert usually have different strategy playing the black stones from playing the white stones.

Datasets	Models	Accuracy	MSE	Elo
Separate color	0 rblocks	44.99%	0.2199	1125
	2rblocks	59.25%	0.1993	1643
	4rblocks	**60.85%**	**0.1919**	**1898**
Both colors	0 rblocks	41.67%	0.249	1097
	2rblocks	45.69%	0.2425	1354
	4rblocks	56.17%	0.2082	1573
	8rblocks	56.88%	0.2153	1809

Table 1. This table contains validation accuracy on policy network, normalized mean square error for value network, and the Elo Score from an internal tournament of all of these models. The model trained with data-sets separated in stone colors obtains higher performance in general.

Models	MCTS	Policy	PELA	PVS
MCTS	-	0 - 10	2 - 8	0 - 10
Policy	10 - 0	-	3 - 7	5 - 5
PELA	8 - 2	7 - 3	-	4 - 6
PVS	10 - 0	5 - 5	6 - 4	-
Total	**28 - 2**	12 - 18	11 - 19	9 - 21
Ratio	**14.00**	0.67	0.58	0.43
Elo Score	**2203**	1831	1803	1763

Table 2. Internal tournament among AIs equipped with or without tree search. We see that MCTS can significantly improve the performance from pure policy network. Our pure policy player also showed better performance against heuristic based AI. PVS Search failed to even achieve the performance of pure policy, which we suspect is due to our noisy value network prediction that greatly impaired PVS Search method's search speed. Note that Policy here is the pure policy network. PELA is restricted to 0.01s per move.

4.3. Evaluation of Tree Search Algorithm

To evaluate whether adding tree search algorithm on-top of our neural network improve the performance, we ran a internal tournament of time restricted fast-games under Renju rules among the following players: 1) pure policy network (i.e. making the move with maximum predicted probability); 2) PELA(gom, 2017) under restriction of 0.01s, as a representative of moderate heuristic based policy; 3) Double dual ResNet (4 residual blocks) with PVS; 4) Double dual ResNet (4 residual blocks) with MCTS using 200 simulations.

The results are shown in table 2. We see that Monte Carlo tree search algorithm can work nicely with the double dual residual network, improving the performance from policy network by a large amount. We have two hypothesis why PVS failed to improve the performance : 1) the value estimation from dual ResNet is too noisy, yet PVS isn't resilience to such noise; 2) bugs in our PVS implementation.

Opponents	400	800	1600	Elo Rating
YIXIN	0-10	2 - 8	1 : 9	2725
RENJUSOLVER	-	-	-	2334
SLOWRENJU	4 - 6	5 - 5	5 : 5	2316
CARBON	5 - 5	4 - 6	8 : 2	2155
XL	5 - 5	9 - 1	9 : 1	1982
PELA	6 - 4	9 - 1	10 : 0	1970
WHOSE	6 - 4	9 - 1	10 : 0	1927
PURE POLICY	10 - 0	9 - 1	10 : 0	-
Wining Rate	1.059	2.478	3.118	-

Table 3. This table records our informal tournament with GomokuCup. Each cell presents the number of winning game and the number of losing game. It shows that increasing the number of simulation can increase the performance against the lower ranked AIs.

4.4. GomokuCup

Finally, we used the best version of our AI to run an informal tournament with the top seven Renju AIs in the world according to GomokuCup(gom, 2017). We ran a one-versus-all tournament using Monte Carlo tree search with different numbers of simulations (400, 800, and 1600). Since our tree search isn't optimized, we restricted other AIs with 1 second per move. The result is presented in Table 3.

Observe that the winning ratio (i.e. number of winning games divided by number of losing games) is increasing as the number of iterations go up, which indicate that the tree search does contribute significantly in the quality of our AI. Our AI with 400-1600 simulations in Monte Carlo tree search can match the performance of SLOWRENJU AI, which is ranked number 3 (out of 7) in the Renju leader-board and 4 (out of 54) in the Gomoku leader-board. Note that this is an estimate of the performance of our method, given that we restricted the thinking time of the Renju AIs, and we haven't optimized our tree search.

5. Conclusion and Discussion

In this report, we demonstrate how to build a Renju AI by integrating double dual ResNet with tree search. Detailed experiments show neural networks can simplify the Renju AI architecture, and achieve reasonably competitive performance among other Renju AIs. In the future, we will include parallelism to speed up tree searches with parallelism and the training process. Truely utilize the reinforcement learning signals will be another future work direction.

References

The renju international federation. URL http://www.renju.net/.

Renju offline, 2017. URL http://renjuoffline.com/main.php.

Gomocup the gomoku ai tournament, 2017. URL http://gomocup.org/.

AI-Zone. Alphagobang - implement alphazero in gobang game. URL https://github.com/AI-zone/AlphaGoBang.

Kai, Sun. Yixin, the strongest gomoku/renju engine in the world, 2017. URL http://www.aiexp.info/pages/yixin.html.

Shao, Kun, Zhao, Dongbin, Tang, Zhentao, and Zhu, Yuanheng. Move prediction in gomoku using deep learning. In *Chinese Association of Automation (YAC), Youth Academic Annual Conference of*, pp. 292–297. IEEE, 2016.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Silver, David, Hubert, Thomas, Schrittwieser, Julian, Antonoglou, Ioannis, Lai, Matthew, Guez, Arthur, Lanctot, Marc, Sifre, Laurent, Kumaran, Dharshan, Graepel, Thore, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

Silver, David, Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, Hubert, Thomas, Baker, Lucas, Lai, Matthew, Bolton, Adrian, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017b.

welkincrowns. Alphago-unlimitedgomokuworks. URL https://github.com/welkincrowns/AlphaGo-UnlimitedGomokuWorks.

Zhao, Rongxiao. Convolutional and recurrent neural network for gomoku.