Alpha Zero Application for Popular Chess Games

ELENE6885.2018Fall

Fan Yang fy2232, Ming Li ml4076, Wenyi Tao wt2271, Xinlei Cao xc2420, Ziyi Ye zy2302
Columbia University

Abstract

Recently, the Alpha Zero program, which starts from random play and given no domain knowledge except the game rules, achieved superhuman performance in the game of Go. In this project, we generalize this approach and implement the Alpha Zero algorithm to solve two popular game problems, Gomoku and Connect Four and we also explore how to define and solve the game Checker using AlphaZero algorithm. For Gomoku and Connect Four, we solved the problems using several different methods and got good solutions. And we also showed appropriately defining a problem is very important before implement an algorithm in the experiment of checker.

1. Introduction

1.1 Background

Games are an interesting domain for artificial intelligence research. They provide a controlled and well-defined environment, generally straightforward rules, and clear-cut results. However, game-winning formulas are often complicated and nonsingular. These characteristics make games suitable to test out different artificial intelligence approaches.

Traditionally, artificial intelligence program leverage enormous amounts of human expertise and data to learn how to play games. Along with the development of different algorithms such as alpha-beta search, deep neural networks, MCTS, and some new reinforcement learning method, the solution becomes better and better.

AlphaGo Zero [2] is one of the most latest algorithms for the problems in this domain. Based solely on reinforcement learning to learn from scratch, without human data, guidance or domain knowledge beyond game rules, AlphaGo Zero has demonstrated its strong ability in Go game and can be much more easily generalized to other problems without significant human effort.

1.2 Related Work and Literature Review

In the history of artificial intelligence research in the game domain, a large number of approaches have been proposed for board games.

The basic program aims to perform a complete search of all the possible moves in the future using game-tree searching and learn how to play using alpha-beta pruning along with a board evaluation function. In 1997, Deep Blue defeated the human world champion, which is a landmark in the application of artificial intelligence in board game domain. The programs Deep Blue [3] took is to evaluate positions using features handcrafted by human grandmasters and carefully tuned weights, combined with a high-performance alpha-beta search. However, to search to a depth of n moves with N possible movements, p!/(p-n)! board situations have to be evaluated [4], which requires massive computational power.

With the development of reinforcement learning, a new method showed up. Freisleben [5] designed an appropriately designed network to play a series of games against an opponent and trained the network using a reinforcement learning algorithm to evaluate the non-occupied board positions by rewarding good moves and penalizing bad moves. However, this method is also limited to the small board game.

To improve the performance and efficiency of basic reinforcement learning, a common approach is to introduce pre-determined patterns based on human experimentation. But in the meantime, it also limits the method to the area where human expertise is lacking.

It was until 2015 that AlphaGo, which was the first program to achieve superhuman performance in Go, appeared. Instead of using supervised learning systems that are trained to replicate the decisions of human experts, it trains reinforcement learning systems from their own experience, which allows them to exceed human capabilities, and also to be generalized without expert knowledge.

The previous version of AlphaGo combined supervised learning and reinforcement learning.

AlphaGo Fan introduced Monte Carlo tree search (MCTS). It combined two deep neural networks: a policy network helping to have a greater chance to choose the good moves while a value network to evaluated positions. The training process can be divided into two part. In the first part, the author trained policy network by supervised learning to accurately predict human expert moves. In the second part, the author utilizes self-play strategies to train the value network to evaluate positions in the tree and refine the policy network by policy-gradient in the meantime.

AlphaGo Lee is similar to AlphaGo Fan. The key differences are: 1) In AlphaGo Lee the value network was trained from the outcomes of fast games of self-play by AlphaGo, rather than games of self-play by the policy network; 2) It trained larger network using more iterations.

AlphaGo Zero, which is the latest evolution of AlphaGo, started from completely random play. Another two significant differences from the previous version are that AlphaGo Zero uses a single neural network rather than separate policy and value networks, and also a simpler tree search that relies upon the single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts.

2. Methodology

In this part, we introduce the architecture of AlphaZero and the technical details how AlphaGo Zero is trained from games of self-play by a novel reinforcement learning algorithm.

2.1 Deep Neural Network

AlphaGo Zero uses a new deep neural network which combines the roles of both policy network and value network into a single architecture. The input of his neural network is the current position and its history, and the outputs are the move probabilities of each action and the estimated probability of winning (i.e. value) at this state.

The neural network parameters are updated to maximize the similarity of the policy vector p_t to the search probabilities π_t, and to minimize the error between the predicted winner v_t and the game winner.

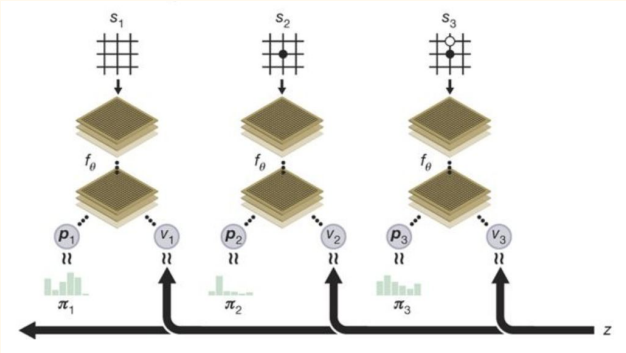

Figure 2-1: Architecture & Training of DNN in AlphaGo Zero.

2.2 Self-play Training

The AlphaGo Zero self-play algorithm can be understood as an approximate policy iteration scheme in which MCTS is used for both policy improvement and policy evaluation.

Policy improvement starts with a neural network policy, executes an MCTS based on the policy's recommendations, and then projects the search policy back into the function space of the neural network.

Policy evaluation is applied to the search policy: the outcomes of self-play games are also projected back into the function space of the neural network.

These projection steps are achieved by training the neural network parameters to match the search probabilities and self-play game outcome respectively.

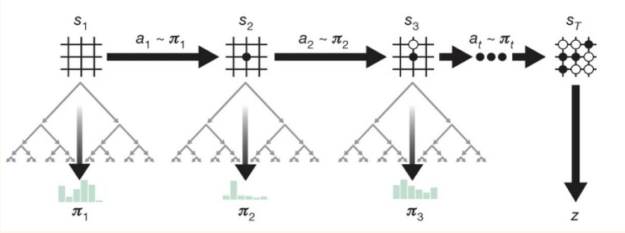

Figure 2-2: Process of Self-play Training

2.3 Search Algorithm

AlphaGo Zero uses a simpler variant of the asynchronous policy and value MCTS algorithm (APV-MCTS).

In the search tree, each node s contains edges (s, a) for all legal actions. Each edge stores a set of statistics: the visit count N(s, a), the total action value W(s, a), the mean action value Q(s, a) and the prior probability of being selected P(s, a).

During each simulation, moves are selected starting from the root state to maximize an upper confidence bound Q(s, a) + U(s, a), where U(s, a) ∝ P(s, a) / (1 + N(s, a)) and the iterations stop if encountering a leaf node.

Multiple simulations are executed in parallel on separate search threads. The algorithm proceeds by iterating over three phases and then selects a move to play.

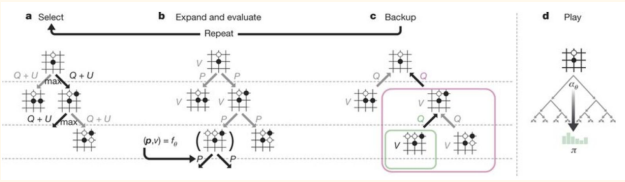

Figure 2-3: Architecture & Training of MCTS in AlphaGo Zero.

3. Implementation

Our project contains two applications of Alpha Zero: Gomoku, Checker and Connect Four.

3.1 Gomoku

3.1.1 Environment

Game Rule

Gomoku, also known as 'five-in-a-row', is played on a square board. In the game, two players place stones alternatively on the intersections of the board and the one who first gets 5 of his stones in a straight line, no matter vertical, horizontal or diagonal, wins the game.

Programming Implement

State s: We used a 3-dimensional matrix to represent our state space. The 3 dimensions are *channels* by *board width* by *board height*.[7] We consider the first two channels of the problem to be the raw board representation. The matrix in the first channel is the representation of white pieces and the matrix in the second channel is for black pieces. The value of the element is 1 if occupied or 0 if empty. The other channels will be well-defined in the following sections.

Action a: Since the state is the raw board representation, we can use all the next possible moves for our action space.

3.1.2 Policy-Value-Network

The Policy-Value-Network is a convolution neural network based model, whose input is the state. The model's output is the probabilities of each action and the estimated value of this state. In our experiment, we built three weight-sharing convolution layers. On the top of these layers, we added one convolution layer and one dense layer separately for both the policy net and value net.

3.1.3 Improvement of Input of Network

For the basic policy-value-network, the input state was defined by three channels, whose shapes were all two dimensions like a chess board (length by width). The first channel was the location of the current player's stones (if there was a stone, the value of its corresponding location was set as 1 otherwise as 0). The second channel was the location of the opponent's pieces. Because we know offense played an important role in the chess situation in Gomoku, we added the third channel into the network to represent the offense information (if the current player was the offense, all the value of this layer was set as 1, otherwise as 0).

In order to make the model converge more quickly and improve the performance, we added one more channel as the input of the network to tell the model the last move location (only the value of the location of the last move was set as 1, others as 0). Because it was more likely to place a stone near the last piece placed by the opponent. This was achieved in our second version algorithm.

In our third version algorithm, we added one more channel based on the version two model to represent the stone move before the last one, i.e. the last move of the current player, cause we thought the last move of the current player is highly related with the intention of the player and the current move.

By comparing the three different versions, we want to show finding a better input of network helps with accelerating the speed of converging and finding a better solution within the same epochs.

3.1.4 Data Augmentation

It is the data simulation part takes the most of the computation cost, in order to save the computation cost and help our model converge, we flipped and rotated the board to generate more data for the network training, because the result remains the same after these transformations.

3.2 Connect Four

3.2.1 Environment

Game Rule

This game is played on a vertical board with seven hollow columns and six rows. Each column has a hole in the upper part of the board, where pieces are introduced. There is a window for every square, so that pieces can be seen from both sides. The aim for both players is to make a straight line of four own pieces; the line can be vertical, horizontal or diagonal.

Programming Implement

State s: Like the architect we used in Gomoku implementation, we also use a 3-dim matrix to represent the state space. The main difference is lied in the channels design in the first channel. The other two channels are the board width and board height, which is the same as the Gomoku game.

Action a: As for each round of Connect 4 game, the player can only place one piece from one of the seven holes on the top, so here in order to accelerate the computeration, I only use a length 7 binary valued vector to represent the action space. And for each move, I use a help function to get the move is valid or not and return the updated board.

3.2.2 Policy-Value-Network

In addition to the Convolutional NN used in Gomoku modelling, in the Connect4 policy value network, we also leverages the Residual Network technology to optimize its network model. The Residual Network is introduced here to overcome the degradation problem of the deep neural network. As the network depth increasing, accuracy gets saturated and then degrades rapidly.[12]

Given a neural network and denote its input is x and its expected output is $H(x)$, if there is an extra edge in the graph which pass the input x directly to the output as one additive value, then the network is supposed to learn will change from $H(x)$ to $F(x) = H(x) - x$. As depicted by

the diagram below, it is a typical Residual Unit of Residual Network.

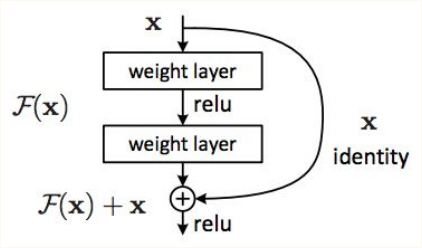

Figure 3-1:typical Residual Unit of Residual Network

Due to the "shortcutting" of the input information to the output, the new model protects the integrity of the information and it also simplifies the learning objective and learning complexity as the entire network just needs to learn the difference between input and output.[12]

4. Results

We compare the model we trained to others' models to see how well AlphaZero performs in the two chess games.

4.1. Gomoku

From the three different versions' result, we can conclude that by adding one more layer to tell the model the last move location and the move before the last one, we can improve the speed of convergence and the capability of the model within the same limited training epochs.

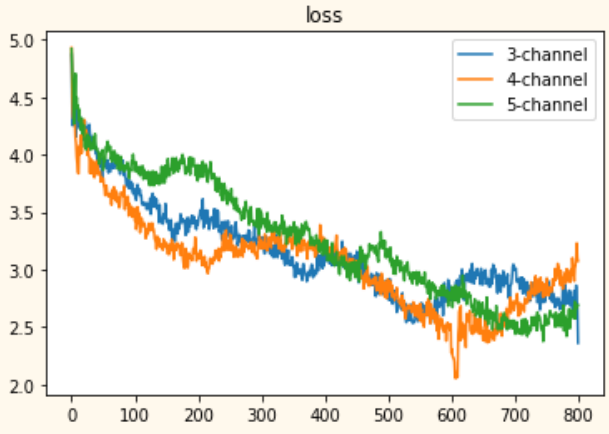

Figure 4-1:The Change of Loss in Training Process for 3 Models.

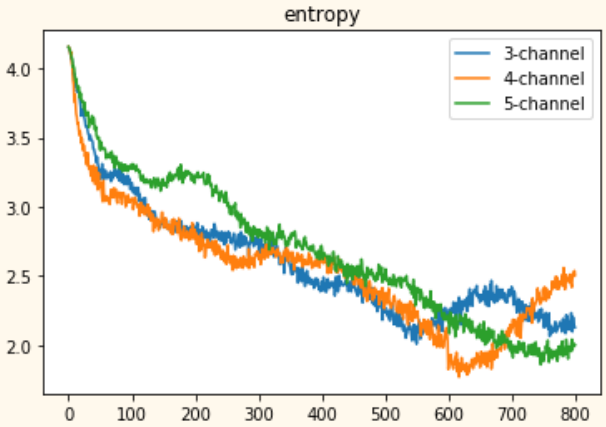

Figure 4-2:The Change of entropy in Training Process for 3 Models.

From the above two figures, we can see the change of loss, entropy for the three different models. When we update the policy network, we set the early stopping criteria to be related to KL divergence of probability over the actions of the same batch prediction. It measures how aggressive we update the model. For each step, we update our model and record the corresponding loss and entropy of our policy network.

5- layer contain more information will work better in the long run but each network are initialized with random weights therefore intuitively contains more garbage compared to 3 and 4 channels. We do find combining one last move from the opponent will help converge faster at first.

Offensive Model	Defensive Model	Winner Model
3-Channel Model	4-Channel Model	4-Channel Model
3-Channel Model	5-Channel Model	DRAW
4-Channel Model	3-Channel Model	4-Channel Model
4-Channel Model	5-Channel Model	4-Channel Model
5-Channel Model	3-Channel Model	5-Channel Model
5-Channel Model	4-Channel Model	5-Channel Model

Table 4-1: Winning Rates between Three Models Against Each Other

From the above table, we can see the result of competition between every two teams from the three different models that the 4-Channel Model and 5-Channel Model have better performance than 3-Channel Model.

4.2 Connect Four

As Connect Four is a really old game and there have already been perfect mathematical solver for this game using optimization method [13, 14, 15], so I try to use my own solver to compete with the classic solver [13] and the really strong optimization solver using Alpha-beta pruning algorithm [14]. By define the player API in the game module, it's easy to achieve the competition between different models even if the model is not developed by us.[10]

The following is the result of 100 rounds of play between the three selected models: Classical Solver, Alpha-beta Solver and our version of Alphago Zero model.

Offensive Model	Defensive Model	Winner Model	Winner Rate
Zero	Classic	Zero	81%
Classic	Zero	Zero	78%
Zero	Alpha-beta Pruning	Zero	63%
Alpha-beta Pruning	Zero	Alpha-beta Pruning	89%

Table 4-2: Winning Rates of Three Models Against Each Other

5. More Games

Apart from the experiment on Gomoku and Connect Four, we further explore a more complicated game Checker.

Checkers is played on a standard 64 square board. Only the 32 dark colored squares are used in play, as shown in Figure 3-1 . The object of the game is to capture all of your opponent's checkers or position your pieces so that your opponent has no available moves. Basic movement is to move a checker one space diagonally forward. You can not move a checker backwards until it becomes a King. If a jump is available, you must take the jump. If one of your opponent's checkers is on a forward diagonal next to one of your checkers, and the next space beyond the opponent's checker is empty, then your checker must jump the opponent's checker and land in the space beyond. Your opponent's checker is captured and removed from the board.

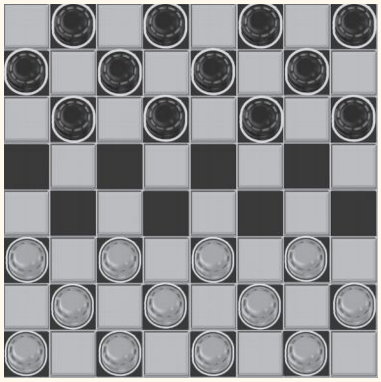

Figure 3-1: Checker Game Board

This game has been solved using TD learning and Monte-Carlo Method [5], in this project we are going to use AlphaZero algorithm to better define and solve this problem.

TD Learning method

Under TD Learning method, the author considered the states to be the raw board representation which is an array signifies the dark tiles of the checker board. Each element represents a tile and the value is determined by whether a tile is occupied. If the tile is empty then the element in the array is 0, if it is occupied by a piece then it is set to -1.0 or 1.0; if it's a king then it is sent to -0.5 or 0.5.

The reward is +1 for a winning move, -1 for a losing move and 0 for a draw. All actions that do not come to a final board position gets a reward of 0.

The first method attempted is temporal difference algorithms involve bootstrapping. Experiments run using replacing eligibility traces and $\lambda=0$, $\lambda=0.9$ didn't show any improvement over the Monte-Carlo Method.

Monte-Carlo Method

In Monte-Carlo method the state value is estimated as the average of the rewards following a visit to this state in an episode. To generate an episode, simply having the agent play against a random player. This method proved to be just as good as any other.

AlphaZero method

Besides the two reinforcement learning algorithms, we implemented AlphaZero method for checker game.

State s: Similar to the application in Gomoku, we used a 3-dimensional matrix to represent our state space. Since there are 32 valid dark squares are used in play, we used a 5 by 8 by 4 matrix to define a particular state. The two matrices in the first two channels represents the raw board representation of white piece and king piece locations. The two matrices in the third and forth channels represents the raw board representation of black piece and

king piece locations. And the last channel stores the last movement.

Action a: We used integer pairs of length two to define an action. The first integer to determine the piece at which location to move; and the second integer can be one of the following four: [1, 2, 3, 4], which represents left forward, right forward, left backward and right backward, respectively.

MCTS: One of our most important improvement is flipping the board. When growing Monte Carlo tree, after set the node state, we intensively flip the board in order to change to the opposite player. Then we grow the action space in the next layer and update the Q + U for all states along Monte Carlo tree trace till current node, by using current policy value network to estimate.

The definition of our environment is the most breakthrough for implementing AlphaZero method. Instead of the raw board representations, we combined the state for both two colored pieces, king pieces and also the last move information. Also the action space definition is a tricky strategy. In order to represents the king's action, we used a 4 dimensional vector for 4 directions' move. And the total action space is a combination of states and actions, which will have a maximum dimension of 32 by 4, which significantly reduces the computational costs.

6. Conclusion

In this project, we generalize this approach and apply the Alpha Zero algorithm to two popular game problems, Gomoku, Connect Four and Checker.

For Gomoku, we implement three versions of deep neural networks with different input channel. By comparison the speed of convergence and the performance (winning rate) of these three models with the same training epochs, we show that defining a good input of deep neural network in AlphaZero algorithm plays an important role in the final model.

For Connect Four, we implement the Alphago Zero algorithm with residual network to solve the game. As connect four's action space and state space is much smaller than the Gomoku or Checker, it converges really fast.(It converages in 500 epochs with 50 MCTS simulations per run). As an old game, there are dozens of solvers from different algorithms, by competition with the two typical solver, we can know that for such a game which has limited state space, the defensive or offensive status plays an vital role in the final outcome.

For Checker, we implemented AlphaZero algorithm to compete with TD learning and Monte-Carlo Method. The most notable work is how we define the environment, this contributed tremendously to our programming implement and also decreases the computational cost.

In a conclusion, we proved the ability of AlphaZero Algorithm to be generalized to other games. And we also showed the importance of the definition of both state space and deep neural network input channel.

7. References

[1] Silver, David, et al. Mastering the game of Go with deep neural networks and tree search, Nature 529.7587: 484-489, 2016

[2] M. Campbell, A. J. Hoane, and F. Hsu. Deep Blue. Artificial Intelligence, 134:57–83, 2002.

[3] Littman, M. L. Markov games as a framework for multi-agent reinforcement learning. In 11th International Conference on Machine Learning, 157–163 (1994).

[4] Gelly, S. & Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. Artif. Intell. 175, 1856–1875 (2011).

[3] T.K. William, S. Pham, Experience-based learning experiments using Gomoku, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Charlottesville, Virginia, USA, vol. 2, October 13–16, 1991, pp. 1405–1410.

[4] B. Freisleben, A neural network that learns to play five-in-a-row, in: Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, 1995, pp. 87–90.

[5] G. Tesauro, Neurogammon: a neural-network backgammon program, in: Proceedings of International Joint Conference Neural Networks, San Diego, California, USA, June 17–21,1990, pp. 33–40.

[6] Schaeffer, J., Hlynka, M. & Jussila, V. Temporal difference learning applied to a high-performance game-playing program. In 17th International Joint Conference on Artificial Intelligence, 529–534 (2001).

[7] Baxter, J., Tridgell, A. & Weaver, L. Learning to play chess using temporal differences. Mach. Learn. 40, 243–263 (2000).

[8] Amco Dubel, Jaap Brandsema, and L. Lefakis, Reinforcement learning project: AI Checkers Player.

[9] https://github.com/junxiaosong/AlphaZero_Gomoku

[10] Müller, M., Enzenberger, M., Arneson, B. & Segal, R. Fuego – an open-source framework for board games and Go engine based on Monte-Carlo tree search. IEEE Trans. Comput. Intell. AI in Games 2, 259–270 (2010)

[11] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. 8, 229–256 (1992)

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Deep Residual Learning for Image Recognition

[13] Victor Allis, A Knowledge-based Approach of Connect-Four, 1988
[14] Knuth, D. E. & Moore, R. W. An analysis of alpha-beta pruning. Artif. Intell. 6, 293–326 (1975)
[15] http://connect4.gamesolver.org/

8. Contributions

All team members contributed equally in all stages of this project. All team members approve our work presented in this report including this contributions statement.

Fan Yang: Responsible for Gomoku game environment implementation, MCTS implementation and related paperwork.

Ming Li: Responsible for Connect Four game environment implement and MCTS implementation and related paperwork.

Wenyi Tao: Responsible for Checker game environment implementation, comparison of TD algorithm, MC algorithm, policy network algorithm and related paperwork.

Xinlei Cao: Responsible for Connect Four policy network implementation, experiment result and related paperwork.

Ziyi Ye: Responsible for Gomoku policy network implementation, experiment result and related paperwork.