A Data-Efficient Method of Deep Reinforcement Learning for Chinese Chess

Changming Xu[1, 2, *], Hengfeng Ding[1], Xuejian Zhang[1], Cong Wang[1, 2], and Hongji Yang[2]
[1]Northeastern University at Qinhuangdao, Qinhuangdao, Hebei Province, China
[2]University of Leicester, Leicester, United Kingdom
ChangmingXu@neuq.edu.cn, 1774777097@qq.com, 709257084@qq.com, CongW@neuq.edu.cn, Hongji,Yang@Leicester.ac.uk
* corresponding author

***Abstract*—The computer game is the *Drosophila* in the field of artificial intelligence. Recently, a series of computer game systems, such as AlphaGo and AlphaGo Zero, defeating the world human champion of Go, has greatly refreshed people's understanding of the creativity of machine. This paper applies the deep reinforcement learning method to the computer Chinese Chess. We are committed to decrease the demand for computing resources heavily from multi-perspectives, such as data augmentation and using more intermediate results as labels. The experiment shows that the level of our program is increased rapidly.**

Keywords- Computer Games, Deep Reinforcement Learning, Chinese Chess, Monte Carlo Tree Search, Residual Network

I. INTRODUCTION

Recently, the most successful method to solve a complex chess-like board game should be deep reinforcement learning [1]. In 2016, being the first program that defeated the human master of Go in 19×19 board, AlphaGo [2] took advantage of the ability to approximate any function provided by deep learning and the decision-making ability of reinforcement learning. Then, an enhanced version, called AlphaGo Zero [3], easily defeated AlphaGo 100: 0.

The ideal representation model for the problem of computer board games is a complete game tree containing all possible playing processes. Following the minimax principle, theoretically, the best action in each state can be obtained. Chinese Chess has the simple rules and the finite state space. However, constrained by limited computing resources and available time, the practical solution can only adopt greedy strategy, that is, only access a subtree, and must estimate the leaf nodes for the further reasoning based on the minimax principle. In this way, the difficulty of Computer Chess-Like Board Games focuses on two key aspects: search algorithm and estimation function. The traditional minimax algorithm and the pruning algorithm [4,5] have a large degree of limitations in extending the game tree. In addition, the evaluation of Chinese Chess board positions is relatively complex, and it is difficult to find reasonable evaluation criteria by hand, even for chess master.

The methods of deep reinforcement learning can surpass Chess master, if you have supercomputing resources, which are difficult for most researchers to access. So, it is very important to end up the learning process with fewer resources [6]. To this end, this paper makes the following two improvements to the characteristics of chess on the basis of AlphaGo Zero:

1) The action space is re-represented, which reduces the complexity of representation. We decompose the original action into 2 successive actions, thus reducing the action space from N^2 to N.

2) It re-represents the input of the network and improves the reuse ability of knowledge.

The rest of this article is organized as follows. Section 2 introduces the design of the Chinese Chess self-playing algorithm. Section 3 introduces the design of the game system: the process of the self-playing system, the data augmentation process, and the implementation of the system. Section 4 compares and analyzes the experimental results. Section 5 draws concluding remarks.

II. TYPE STYLE ADESIGN OF CHINESE CHESS SELF-PLAYING ALGORITHM

The Monte Carlo Tree Search (MCTS) [7-9] is an optimization method based on probability and statistics. The Upper Confidence Bound Apply to Tree (UCT) is the popular MCTS algorithm for Computer Chess-like Games so far.

Deep Neural Network is the State-of-the-art for function fitting. It is used for implementing the value function and strategy function in deep reinforcement learning [10]. Given a state, the neural network should estimate its values and the distribution of actions to be played. At first, it will be impossible. However, we can train the neural network through self-playing.

A. UCT

This section introduces how to apply the Monte Carlo method to the Chinese Chess.

The upper confidence of each child node N_i is calculated by Equation 1.

$$Z_i = \overline{X_i} + C\sqrt{\frac{\ln t(N)}{t(N_i)}} \qquad (1)$$

In the formula, node N is the parent node of N_i. Z_i is the upper limit confidence index value of the ith child. $\overline{X_i}$ is the simulated average reward value of child i, which is used to represent the existing reward. $t(N_i)$ is the number of simulations that the ith node is selected, and C is the weighting coefficient. $\sqrt{\frac{\ln t(N)}{t(N_i)}}$ represents the unknown reward that needs to be explored.

MCTS will gradually expand the game tree according to the upper bound confidence strategy value. Based on this upper bound confidence strategy, the game tree can more reasonably find a balance between exploration and utilization, to obtain better results. MCTS are implemented in four steps. The specific example is shown in Figure 1.

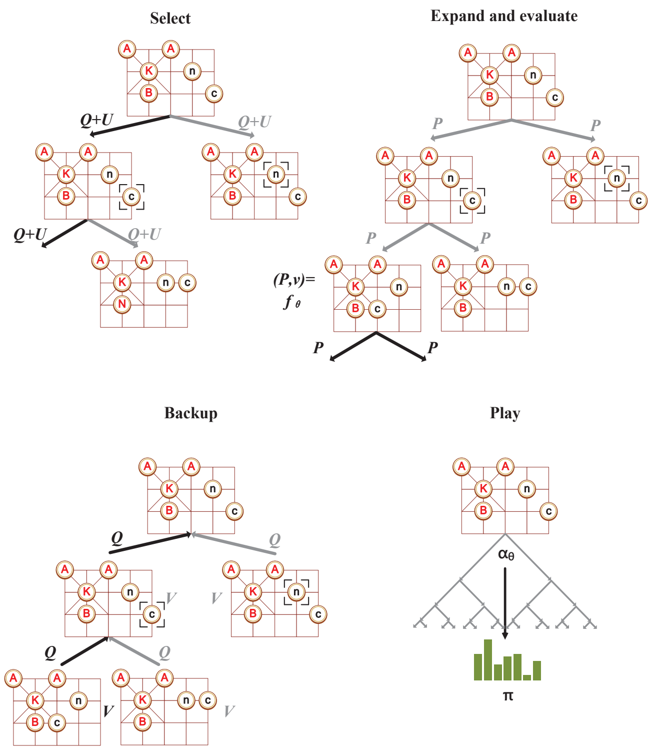


Figure 1. MCTS of Chinese Chess

Selection: In the selection process of the MCTS, the neural network is used to guide the search. The input to this step is the current state of the board, including the position of each piece and which side it is the turn to play. Through the restriction of Chinese Chess rules, the legal position in the current state will be generated by the move generation function. For the first layer of the node, the position coordinates of the pieces that can be selected are first generated. After searching for the prior probability $p(s,a)$ of each edge, the number of visits $N(s,a)$ and the action value $v(s,a)$, it is selected from all legal positions to make the position of the pawn with the largest value of the UCT formula. After the end of the selection stage of the first layer, for the selection stage of the second layer, the program will generate the legal moves of the selected piece, that is, the coordinates of the position where the piece can be moved. The UCT formula, further selects the position of the pawn in the legal position. The formula for UCT is as follows:

$$UCT = Q(s,a) + U(s,a) \tag{2}$$

$$U(s,a) = c_{puct} \cdot p(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \tag{3}$$

c_{puct} is a constant that determines the weight of exploration, s represents the parent node, and a, b are actions leading to the corresponding child nodes. Each time the target action selected has the maximum value of UCT. By the formula, the size of $U(s,a)$ is negatively correlated with the number of times $N(s,a)$ that the current node is explored, that is, when the difference between the winning rate v predicted by the neural network is small between all the candidate nodes, the difference between $Q(s,a)$is small, and this The more times node a is selected, the smaller $U(s,a)$ will be, and the system will be more inclined to select nodes that have been explored less or have not been explored. Similarly, when the number of explorations between different nodes is not very different, the system will be more inclined to select the node with a larger probability v predicted by the neural network, thus making the UCT larger. UCT algorithm achieves a balance between exploration and exploitation.

In the selection process, UCT algorithm will be used for each layer node, until the action selected by a certain layer node is not extended, and the second step of expansion is entered.

Expansion and evaluation: In (1) process selection, after selecting and moving through the two layers of each node, it will explore a certain node along a certain path. At this time, the node should be expanded, and the chess pieces should be selected or moved for the action selected in the selection, that is, a new node is generated, representing a new state. Then use the neural network to predict the new node, and the output results are the prior probability $p(s,a)$ and the winning probability evaluation value $v(s)$ of the current situation. At the same time, the $N(s,a)$, $W(s,a)$, and $Q(s,a)$ of each branch is initialized to zero, and $P(s,a)$ is initialized to p_a, where p_a The prior probability $p(s,a)$ from the neural network prediction.

Backtracking: After obtaining the winning probability evaluation value $v(s)$ predicted by the neural network, add 1 to the number of visits $N(s,a)$ of the node, and backtrack this value and the number of visits of each node layer by layer, and pass it back to all parent nodes, to realize the transformation from Bottom-up is continuously updated until it reaches the root node. And the $N(s,a)$and $Q(s,a)$ of each node and each branch is continuously calculated according to the following formula, to make a more reasonable choice in the next round of exploration. The number of visits backtracking formula 4.

$$N(s,a) = N(s,a) + 1 \tag{4}$$

The total value of the node backtracking formula：

$$W(s,a) = W(s,a) + v(s) \tag{5}$$

Node average value backtracking formula：

$$Q(s,a) = \frac{W(s,a)}{N(s,a)} \tag{6}$$

After the (3) stage is over, the algorithm will return to the (1) selection stage and continue to repeat the process of (1) ~ (3) until the number of executions of the process reaches the specified threshold, and then enter (4) to execute the action.

Execute action: At the end of the search process, the system will generate a probability distribution $\pi(a|s_0)$ of action selection at the position of the root node, and its calculation formula is:

$$\pi(a|s_0) = \frac{N(s_0,a)^{\frac{1}{\tau}}}{\sum_b N(s_0,b)^{\frac{1}{\tau}}} \qquad (7)$$

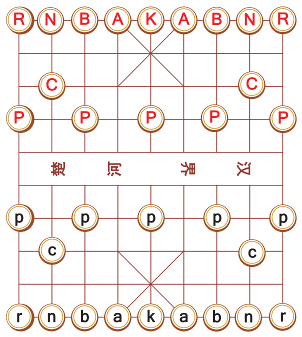
τ is a constant that controls the exploration level and is called the temperature parameter. After the action is executed, the child node corresponding to the selected action is selected as the root node, the subtree under this node maintains the previous statistical characteristics unchanged, and the rest of the tree is discarded. When the value of the root node and the best child node are both less than the specified value threshold, the player to move resigns.

B. *Structure of deep neural network*

This paper uses the residual neural network that combines the strategy network and the value network as the deep learning model. Since each move in Chinese Chess needs to go through two stages of selection and movement, this system needs to train two neural networks, named select network and move network respectively, which are responsible for the combination of the strategic network and value network in the process of selecting pieces and moving pieces.

In the process of self-play, the neural network is used to guide the MCTS for scoring. The input structure of the neural network needs to be able to express the current chessboard state. In the selection stage, the chessboard state specifically refers to the position of the chess piece and stage number 0; in the moving stage, the chessboard state specifically refers to the position of the chess piece, the coordinates of the selected chess piece, and stage number 1. The chessboard can be viewed as a 9×10 matrix, where each element of the matrix represents the intersection point corresponding to a coordinate on the chessboard. The state corresponding to each intersection can be empty, and it may also be king, assistant, bishop, knight, rook, and pawn of the black or red side. Any piece of these 7 types, that is, each element in the matrix has a total of 15 possible values. If the checkerboard intersection is empty, the corresponding element in the matrix is represented by 0; If the intersection points of the chessboard are pawns, cannons, rooks, knights, bishops, assistants, and king on the red side, the corresponding elements in the matrix are 1, 2, 3, 4, 5, 6, and 7 respectively; if the intersection points of the board are the corresponding pieces of the black side, the elements in the matrix are the opposite numbers of the corresponding number numbers of the red pieces. Table 1 shows the correspondence among Chinese characters, letter codes, and digital codes of chess pieces.

According to the above table, the position of the pieces in any chessboard state can be represented in one and only one way. Taking the opening game as an example, the corresponding representation of the chessboard is shown in Figure 2.



$$\begin{bmatrix} 3 & 4 & 5 & 6 & 7 & 6 & 5 & 4 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -4 & -5 & -6 & -7 & -6 & -5 & -4 & -3 \end{bmatrix}$$

Figure 2. Representation of opening chessboard

Table 1. Chinese Chess pieces

Piece name	King	Assistant	Bishop	Knight	Rook	Cannon	Pawn
Piece name (In Chinese)	帅/将	仕/士	相/象	马	车	炮/砲	兵/卒
Red piece letters	K	A	B	N	R	C	P
Red piece ID	7	6	5	4	3	2	1
Black piece letters	k	a	b	n	r	c	p
Black piece ID	-7	-6	-5	-4	-3	-2	-1

The neural network in this system is divided into two parts. The first part is called the select network, which is used to select the pieces to be moved after getting the state of the chessboard; the second part is the Move network, which is used to select the pieces determination of the mobile position. The general structure of the two parts of the network is the same, both are multi-layer deep residual networks. But the structure of the residual network in the first layer is different. This is because the input is different. The number of input channels of the first layer in the move network is 2, that is, two input matrices, while the select network only needs 1 channel.

The rest of the network structure is the same, and the network is divided into three parts: the feature extraction part, the strategy network part, and the value network part. There are four layers of the residual network in the feature extraction part, it will first convert the number of input channels to 32 channels, which will further convert to 64 channels, and in the last layer of the residual network, the number of channels of the feature will be reduced to 32. Intuitively, in this structure from low-dimensional to high-dimensional and then to low-dimensional, since the high-dimensional part has more model parameters, it is expected that the model can learn more knowledge in the high-dimensional part, while the final low-dimensional part is expected to learn more knowledge. The dimensional part compresses it to lower dimensions to generate embeddings that can be processed by downstream tasks. The network of the feature extraction part has the structure on the top part of the figure 3, and its input will be used as the input of the strategic network and the value network respectively. At this time, the output of the feature extraction network represents the chessboard. The embedding generated by the residual network learning, Represents the learned chessboard information, and the dimension of the vector is 32×9×10, because there are 32 channels in total, and each channel is a matrix equivalent to the size of the chessboard.

The structure of the strategic network and the value network are similar. First, there will be a residual block to further process the embedding from the feature extraction network. However, because the final output vectors of the two networks are different, the fully connected layer after the residual block is quite different. As a policy network, the purpose of the final output is to give the probability of selection or movement of 90 points on the chessboard, that is, the output is a 90-dimensional vector, which contains more information. Therefore, in the fully connected layer, the hidden layer in the middle has a higher dimension of 512 dimensions, and after a high-dimensional transformation, the final output is 90 dimensions. For the value network, the output target dimension is the winning or losing value in the current situation, so it is a one-dimensional vector. At this time, a high-dimensional hidden layer is not needed, so the dimension of the hidden layer is only 64 dimensions, which is used for Extracting the value information under the current chessboard, and finally outputting a one-dimensional vector after the dimension transformation.

C. *Neural network training process*

In the stage of self-play, the neural network will take the chessboard state as input, and output the predicted values of $p(s, a)$, and $v(s)$ through the feature extraction network part, the strategy network part, and the value network part. After the end of the self-game stage, the system obtains the result z of the current game process and records the output value π of MCTS in the self-game process. At this time, the back-propagation starts. According to the loss function, each corresponding π and p in the state are calculated by cross-entropy, and z and v are calculated by the mean square errors. The specific calculation process is shown in Figure 3.

III. Design and Implementation of Chinese Chess Computer Game System

A. *Self-play process implementation*

To facilitate the prediction and training, the state of the board fed into the neural network are simply supposed to be the red side in our implementation. When it is the black's turn to play, the position of chess pieces on the board should be flipped up and down symmetrically, and then the color of each chessman will be exchanged. As a result, the same neural network can be used on both the black player and red player.

In the process of self-play, it often happens that some pieces on the board, such as king, rook, knight, and cannon, are chased, and the opponent have been unable to capture through the attack. more than 100 moves have not been captured. Because when the loss in the system returns a value of 1 for victory or -1 for defeat, it is more valuable to learn during training. The implementation of self-game process is shown as Figure 4.

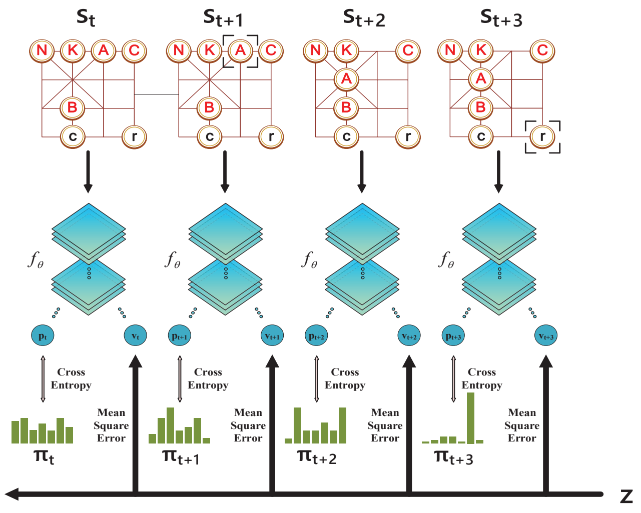


Figure 3. Schematic diagram of neural network training process

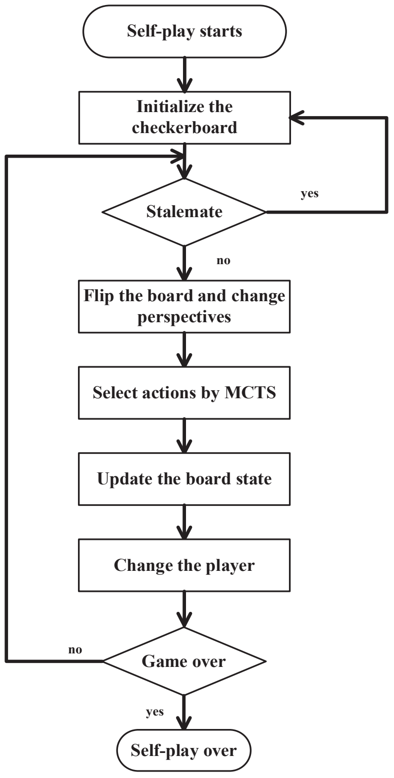


Figure 4. Flow chart of self-playing process

To facilitate the implementation of the algorithm, when the red side starts the game, the algorithm will perform the operations of inverting the chessboard and changing the perspective.

B. *Enhancement*

The most resource-intensive part of deep reinforcement learning is self-learning. The slowest part of the self-playing is the MCTS, which invokes neural network to get value of a state and the strategy distribution of actions at each node in the tree. To save computation, we adopt the following measures:

1) data augmentation. The data augmentation method generates new data and make it available as training data to the neural network. The specific data augmentation method incorporates a characteristic of the Chinese Chess board— symmetry. Since the position of the pieces is symmetrical with the vertical line where the king is located, that is, the five-way position, if you simply do a symmetrical transformation with the vertical line as the axis of symmetry, it will not affect the chess game. The state causes any effect, and the situation before and after the change is the same. The picture below shows the classic opening of the red side's central artillery rushing in seven pawns to deal with the black side's three-step tiger. It can be seen that after the chessboard is symmetrically transformed, the advance of the seventh pawn is changed to the advance of the third pawn, and the advance of the Knight on the eight line is changed to the advance of the Knight on the second line, in essence, two chess games are equivalent.

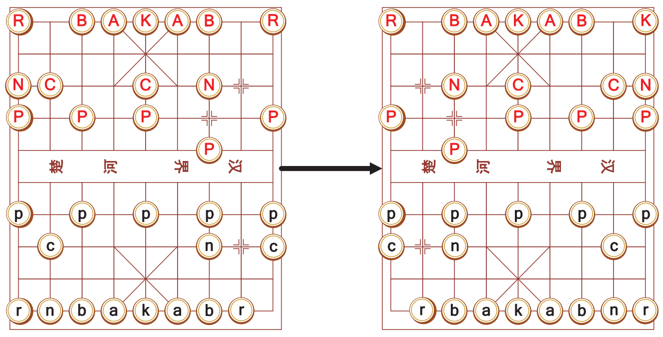


Figure 5. Data Augmentation

2) The action space is re-represented, which reduces the complexity of representation. The discrete action space of Chinese Chess is too large, which is not conducive to the structural representation of neural network output and the numerical representation of the strategy distribution. For example, in the left board of Figure 5, the moving of the red rook at 2nd column from the 1st row to the 5th row can be separated into two steps: first, select the red rook at right among all the red chessmen; second, push forward the rook 4 grids. Thus, we decompose the original action into 2 successive actions, thus reducing the action space from N^2 to N.

3) re-representing the input of the network and improving the reuse ability of knowledge. To emphasize the importance of the order of the move sequence, the input of neural network is a state sequence fragment composed of 8 consecutive moments in AlphaGo Zero. In this paper, the input of the network is changed to the state of only one moment, which increases the hit rate of hash table, thus improving the knowledge reuse ability, and at the same time.

IV. Experiments and Results

Our Chinese Chess computer game system generates training data through self-play using MCTS, trains neural networks with the obtained data, and then generates new data with new networks through self-play and so on... At present, 100 generation models have been iteratively generated.

A. *Training process and experimental analysis*

The losses incurred during training are recorded and analyzed in this paper. From the form of the loss function mentioned above, it is not difficult to see that the loss function consists of three parts, namely the predicted probability p, the predicted winning rate v and the regular term. In the loss curve shown p in Figure 6, due to a large number of action branches and the probability distribution π obtained by MCTS is not necessarily the best action strategy, it is difficult to learn the action strategy probability distribution, even after 100 rounds of iterations, there is still a loss close to 1, which further verifies the complexity of the Chinese Chess game and the diversity of moves. In Figure 7, both the win rate v and the loss of the regularization term are gradually decreasing. Combining the changes of the three in the learning process, it can be concluded that the learning of the model is indeed fruitful.

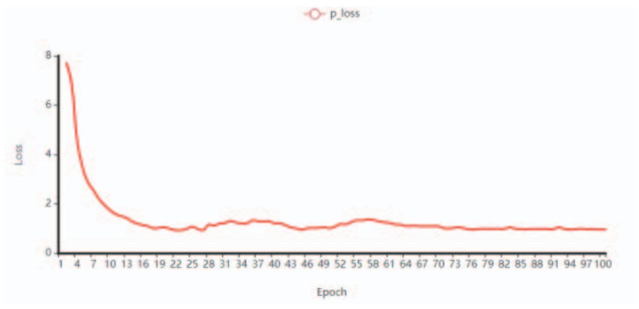


Figure 6. Loss curve for π during training

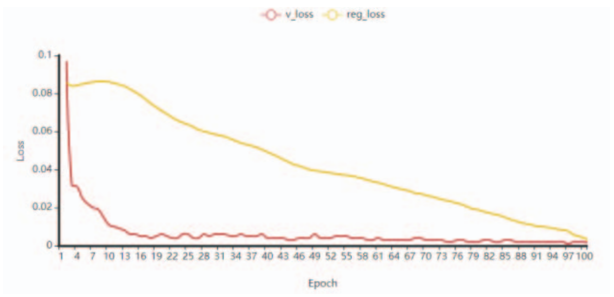


Figure 7. Loss curve of v and regularization term during training

Table 2 shows that, from the initial model to the 100-round model, the average number of steps in a game increased first, and then decreased. It is found that the reason for the small number of steps in the initial model is that after random movement because the opponent cannot well explore the state of the veteran being eaten. As the number of iterations of training increases, the model begins to gradually learn to avoid silly moves. At last, the model knows how to efficiently use rooks, knights and cannons, thus reducing unnecessary movement of sub-forces during the game, making the number of game moves tend to decrease.

Table 2. Comparison of the number of moves in a single game in different stages of the Chinese Chess self-playing

Models	**Avg(#moves)**
Initial Model	63
After 5 iterations	319
After 10 iterations	453
After 50 iterations	284
After 100 iterations	221

Table 3 shows the comparison between the models trained at different rounds, which can see the progress of the model. However, due to the different styles of playing chess between different models, in the process of chess power comparison, even the model of 100th iteration will still lose chess. At the same time, too many draws still mean that the attack and kill the ability of the 100th iteration model still needs to be improved. But on the whole, with the increase of training rounds, the chess power of the model has been improved.

Table 3. improvements of the models during the training

Compared Models	**Win : Draw : Loss**
100 iterations VS 5 iterations	14: 6: 0
100 iterations VS 10 iterations	8: 11: 1
100 iterations VS 50 iterations	4: 15: 1

During the training process of the AlphaGo Zero system, it only took 3 days to generate up to 3.9 million chess data. For most researchers, this is nothing more than an astronomical amount. Because of the limitation of computing resources and time, the number of chess records generated by this system for training is quite small compared to it, so it has not been able to train the chess skills to be able to compete with human master. However, the experimental results show that our program have make significant progress.

B. Detailed Explanations and Examples

It seems to be playing randomly, during the first five rounds of training of the model. Here, the neural network is still in its initial stage, its parameters will basically guide the MCTS for more random exploration. Furthermore, both sides have not yet discovered that capturing is an important value factor to win, nor have they learned the characteristics of the chess pieces with the obvious high mobile value, such as rooks, knights, and cannons, etc. However, at this stage, the data generation speed, as well as training speed are very fast, because in the process of nearly random play, there is often a chance that a player will move its king to be threatened by the enemy's cannons, or rooks. There is a very high probability of directly ending the game early. Therefore, the first five rounds of training are more in the exploration stage.

The model has gradually gained some cognition in the sub selection stage when the model is trained to 10 iterations. In order to facilitate model learning, the neural network simply considers that the player of current state of the board is red. When black chess moves, the positions of black chess and red chess will be exchanged. As a result, the model learned to give higher probability to chessmen above the boundary when selecting a position with a chess piece as the starting point of a move. But there are some typical problems in the model training at this stage. For example, the pieces located on the bottom two lines are difficult to be selected by the neural network, and the probabilities of them close to zero. This leads to the fact that if a pawn or a knight, gets to the position, even if it can threaten the enemy's king within a few steps or even one step, it cannot be selected in the MCTS. Meanwhile, because it is easy to find that moving the king threatened by the enemy is an important strategy to avoid losing the game, so the probability of the king being selected is quite high after learning. Although this effectively avoids the failure of our side, frequently moving the king is the choice wasting attack opportunities. Furthermore, it is difficult to end the game, which will greatly prolong the steps of the self-play process, resulting in slow training speed.

The model has preliminarily learned some basic knowledge after the training reaches 50 iterations. At this time, the probability of six pieces with the high value above the river boundary being selected becomes greater, as shown in Figure 8. The chessboard on the left represents the current chessboard state, and the chessboard on the right represents the probability of selecting pieces.

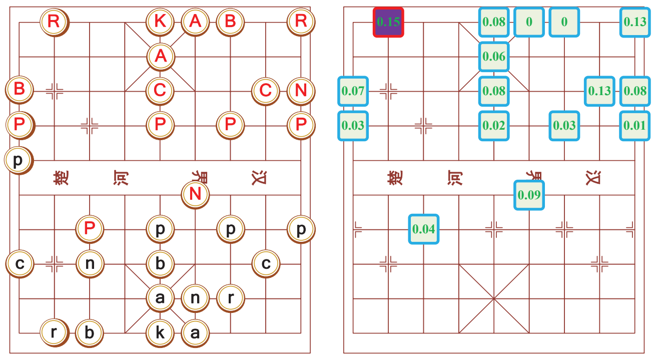


Fig 8. The probability distribution on selecting chess pieces for the state of the left-hand board (hints: red to move) after training the model 50 iterations

It can be seen that the red side's rook at left has the highest probability of being selected, and the probability of the chessmen with high value in other positions is obviously higher than that of other types. The model has recognized that these chessmen pose a greater threat to the enemy's king, so it is more inclined to select these pieces to move at the opening stage, and in addition, the problem of ignoring the pieces that go deep into the enemy's rear has been alleviated.

When the training reaches 100 rounds, as shown in Figure 9, the model learned to use assistants and knights to conduct simple defense when the enemy's aggressive chess pieces threaten our king. It can be seen that the one with the highest probability of selection in the figure is the assistant on column 5. The model hopes to conduct defense through the assistant retreating from column 5 to column 4. The model knows that the pawn should move more laterally rather than blindly rush to the enemy's bottom line after forward crossing the river boundary. More than this, the problem of the models that king being selected frequently and the neglect of selecting the chess pieces on enemy's home will be further alleviated at the same time.

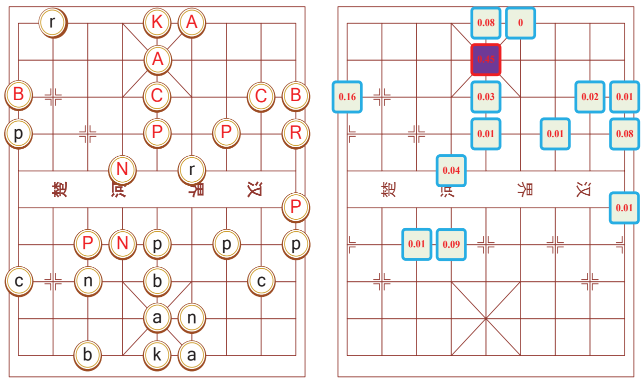


Fig 9. The probability distribution on selecting chess pieces for a state of the left-hand board (hints: red to move) after training the model 100 iterations

V. Conclusion

This paper introduces a data-efficient Chinese Chess deep reinforcement learning method, which utilizes the closed-loop process of Chinese Chess self-play, and continuously updates the network by automatically generating chessboards and training. At present, 100 generations of models have been iteratively generated. Through the analysis of the probability distribution of the action strategy predicted by the system and the loss curve in the training process, it is proved that the chess power of the system has been significantly improved.

References

[1] Y. Li, "Deep reinforcement learning: An overview", arXiv preprint arXiv:1701.07274, 2017.

[2] D. Silver, A. Huang, C. Maddison, A. Guez , L. Sifre, et al., "Mastering the game of Go with deep neural networks and tree search", Nature, London, UK, 2016, 529(7587), pp. 484-489.

[3] D. Silver, J. Schrittwieser, K.Simonyan, I. Antonoglou, A. Huang, A. Guez, et al., "Mastering the game of go without human knowledge", Nature, London, UK, 2017, 550(7676), pp. 354-359.

[4] D. Knuth, R. Moore, "An analysis of alpha-beta pruning", Artificial intelligence, Elsevier, 1975, 6(4), 293-326.

[5] T. Marsland, "A review of game-tree pruning", ICGA Journal, IOS press, Amsterdam, The Netherlands, 1986, 9(1), pp. 3-19.

[6] H. Li, D. Li, W. E. Wong, D. Zeng, and M. Zhao, "Kubernetes Virtual Warehouse Placement based on Reinforcement Learning", International Journal of Performability Engineering, 2021, 17(7), pp. 579–588.

[7] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search", International conference on computers and games, Springer, Berlin, Germany, 2006, pp. 72-83.

[8] G. Chaslot, S. Bakkes, I. Szita, P Spronck, "Monte-carlo tree search: A new framework for game ai", Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Chicago, US, 2008, 4(1), pp. 216-217.

[9] S. Gelly, D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go", Artificial Intelligence, Elsevier, 2011, 175(11), pp. 1856-1875.

[10] Y. Cheng, D. Li, W. E. Wong, M. Zhao, D. Mo, "Multi-UAV collaborative path planning using hierarchical reinforcement learning and simulated annealing", International Journal of Performability Engineering, 2022, 18(7), pp. 463–474.