Hindawi
Complexity
Volume 2020, Article ID 4708075, 11 pages
https://doi.org/10.1155/2020/4708075

Research Article

Hybrid Online and Offline Reinforcement Learning for Tibetan Jiu Chess

Xiali Li, Zhengyu Lv, Licheng Wu, Yue Zhao, and Xiaona Xu

School of Information and Engineering, Minzu University of China, Beijing 100081, China

Correspondence should be addressed to Xiali Li; xiaer_li@163.com

Received 13 February 2020; Revised 3 April 2020; Accepted 10 April 2020; Published 11 May 2020

Academic Editor: Ning Cai

In this study, hybrid state-action-reward-state-action (SARSA (λ)) and Q-learning algorithms are applied to different stages of an upper confidence bound applied to tree search for Tibetan Jiu chess. Q-learning is also used to update all the nodes on the search path when each game ends. A learning strategy that uses SARSA (λ) and Q-learning algorithms combining domain knowledge for a feedback function for layout and battle stages is proposed. An improved deep neural network based on ResNet18 is used for self-play training. Experimental results show that hybrid online and offline reinforcement learning with a deep neural network can improve the game program's learning efficiency and understanding ability for Tibetan Jiu chess.

1. Introduction

Compared with Go, chess, Shogi, and other games achieving the top level of human beings by deep neural network and reinforcement learning, the research of Tibetan Jiu chess is still in the primary stage. The current chess power of Jiu chess is low, which has not defeated the primary players of human beings. And the standard Jiu chess game data are very few. At present, the complete Jiu chess manual obtained is only about 300 games. Jiu chess game has 2 sequential stages of layout and battle, and the layout stage can only enter into the battle stage after all the blank intersections are filled alternately, which leads to the lengthy search path for an upper confidence bound applied to trees (UCT) search. Considering the useless steps that the algorithm may take during the exploration, the search path will be much longer than that of chess and Go. In this case, how to improve the efficiency of Jiu chess program in self-play learning under the special rules and the limitations of laboratory hardware is our study motivation.

In this study, the state-action-reward-state-action (SARSA (λ)) and Q-learning algorithms are innovatively used in different stages of UCT search for Jiu chess. In the selection, expansion, and simulation section of the UCT search, the SARSA (λ) algorithm is used to update the quality of each action. In the backpropagation stage, the Q-learning algorithm is used to update the quality of each action. The quality of each action is denoted by its Q-value. The Q-learning algorithm is also used globally after the end of each game. Probability distribution function based on two-dimensional (2D) normal distribution matrix is used as feedback to SARSA (λ) and Q-learning for the layout stage. The feedback function for a time difference (TD) algorithm based on important shapes [1, 2] is constructed for the battle stage. ResNet18 structure is improved to be suitable for the deep neural network training considering its lower error rate and better performance in image classification [3]. The contribution of this study is outlined in the following:

(1) Hybrid deep reinforcement learning algorithm is firstly applied to Jiu chess game. The proposed strategy can prevent many worthless or low value nodes generated by the deep learning from wasting computing resources. Using SARSA (λ) and Q-learning algorithms in different stages of UCT search provides guarantee for learning fine chess steps faster and improves the algorithm's learning efficiency; besides, final result can be fed back to all steps to improve the accuracy of the return estimation value.

(2) Constructing probability distribution function based on two-dimensional (2D) normal distribution matrix makes the reinforcement learning model have the ability of prioritizing its learning tactics in the center of the layout.

(3) The proposed improved Resnet18-based network, with very small parameter scale, has achieved a good learning effect, reduced the use of computing power, and significantly shortened playing time by training on a common workstation or graphics processing unit (GPU) server in a short time. The experimental results have demonstrated it.

Section 2 of the paper introduces the rules and difficulties of Tibetan Jiu chess, and Section 3 introduces related works on Tibetan Jiu chess. Section 4 discusses the hybrid SARSA(λ) and Q-learning algorithm applied to different stages of the UCT search and the improved RedNet18-based deep neural network structure. Section 5 details the experiment and data analysis. Finally, we outline the conclusions and propose future work.

2. Rules and Difficulties in Tibetan JIU Chess

There are two kinds of Tibetan chess, mi mang and Jiu, which are widely played in Sichuan, Gansu, Tibet, Qinghai, Yunnan, and other Tibetan areas [4] in China. The process of playing chess can be divided into two successive stages: layout and fighting. In the layout stage, the central area is occupied first in order to construct the dominant formation (square, standard, etc.) for the fighting stage; hence, the quality of the layout has an important impact on the final victory. In the fighting stage, actions include moving chess pieces, jumping capture, or even square capture. The key to victory is to take the lead in constructing a girdling formation, and the necessary condition to constructing this formation is to construct a square.

2.1. Tibetan Jiu Chess Rules. The public Jiu board is 14×14 points. The Jiu game process is divided into two sequential stages: layout and battle. Jiu chess is a two-player game. The white side plays with white stones and the black side uses black stones. Players alternate turns. Stones must be added or moved to empty points on the game board. A Jiu game starts with an empty board. The task in the layout stage is to place one stone on a point at each move until there are no empty points on the board. The goal in the battle stage is moving or capturing stones until one player wins the game. The movements and capturing methods are similar to those of international checkers [1, 2].

2.1.1. Layout Stage. In the layout stage, white plays first, followed by black. The first and second moves must be placed on one of the points of the diagonal line of central grids. Then, each side alternates in placing one stone on one point until no empty points remain on the board. After filling the board, game turns into battle stage.

2.1.2. Battle Stage. In the battle stage, there are three actions to select in each turn.

(1) Move. Usually, a player moves a stone to the up, down, left, or right adjacent empty point (see Figure 1(a)). But there is the exception. If a player has no more than 14 stones left, he can move a stone to any empty cross point he wants. This exception is shown in Figure 1(c). Since the black side only has no more than 14 stones left, the diamond black stone moves to the point where the arrow directs. After this move, the square is constructed. During this move, the point of the diamond black stone is the beginning and the point of the black stone directed by the arrow is the ending. In this case, the black side can capture any one stone of its opponent. The dim diamond marked white stone is taken away by the black side after this move.

(2) Jumping Capture. When the opponent's stone is adjacent to the player's stone and there is an empty point directly behind it, the player will perform a jumping capture. This action can be continued until the player cannot capture stones or the player's turn ends. As shown in Figure 1(b), the diamond marked white stone is placed to the final empty point where the arrows direct after four continuous jumping. This continuous jump begins from the point of the diamond white stone and ends at the point of the white stone directed by the arrow. The white side captures all the four black stones on the path where the arrows direct.

(3) Square Capture. In one turn, if a player constructs a square with four adjacent stones, he will capture one of his opponent's stones located at any point on the board. This is called square capture. There are three important shapes which are called gate, square, and dalian (or chain) associated with square capture. Gate is the basic shape which is shown in (A) in Figure 1(d). Square, one of the important shapes, is shown in (B) in Figure 1(d). Dalian or chain is the most important Jiu shape, which is shown in (C) in Figure 1(d). This shape is critical to the winning of the game. It is comprised by seven stones of the same color and one empty point. The stone adjacent to the empty point is called vital stone which is marked by the circle. By moving this vital stone to any of the two empty points, a player can construct a square and then capture one stone anywhere of its opponent on the board in one turn. A player can capture his opponent's stones by repeatedly moving the vital stone in different turns.

2.1.3. Winning the Game. If a player wants to be a winner, he will make sure one of the following conditions is met:

(1) He must have at least one special shape like chain while his opponent does not have any gates before his opponent has less than 14 stones.

(2) He will win by taking away all stones of his opponent when both sides have no special shapes or gates.

2.2. Difficulties in Tibetan Jiu Chess

(1) Deep and wide tree search space. The layout stage is closely connected with the battle stage. When the

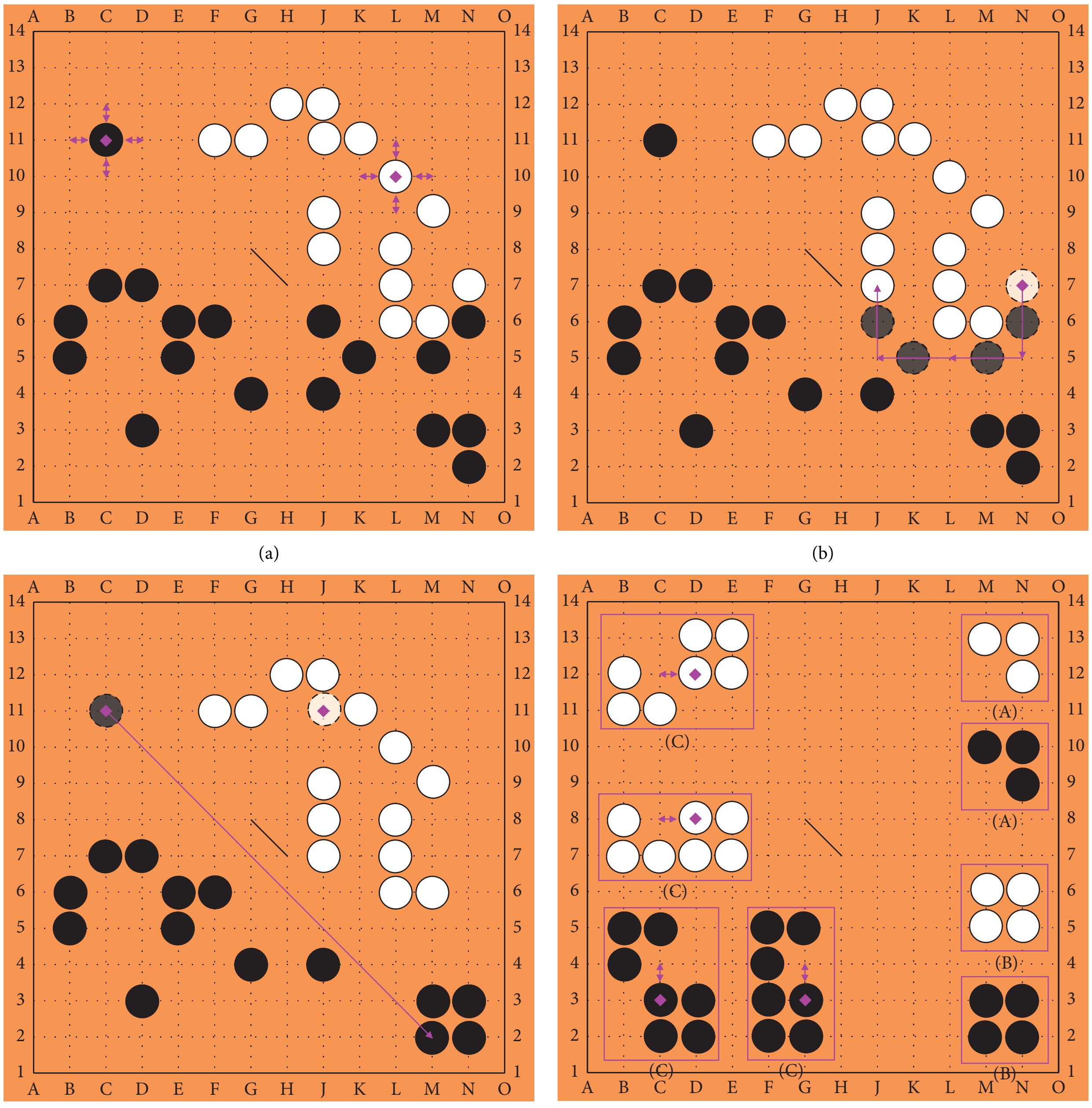

Figure 1: Actions and shapes in Jiu chess. (a) Example state 1. (b) Example state 2. (c) Example state 3. (d) Important shapes.

layout is finished, the battle begins. The search space of the game tree is huge and the search path is far longer than that of general chess games.

(2) Special rules make much more low value states than high value states. The Jiu chess layout first covers the central area and then gradually expands outward. The importance of position gradually decreases from the center to the outside. It takes a long time for ordinary deep neural networks to learn to choose high value states from much more low value states.

(3) Extremely limited research and expert knowledge. Jiu chess players are mostly distributed in Tibetan areas, creating huge difficulties in collecting and processing Jiu chess data. At present, we have only collected and analyzed 300 complete chess record data, represented as SGF files.

3. Related Work

Deep reinforcement learning algorithms used in the Atari series of games, inlcuding Deep Q Network (DQN) algorithm [5], 51-atom-agent (C51) algorithm [6], and those suitable for continuous fieds with low search depth and narrow decision tree width [7–15], have achieved or exceeded the level of human experts. In the field of computer games, pattern recognition [6, 16], reinforcement learning, deep learning, and deep reinforcement learning algorithms are used in Go, including Monte Carlo algorithm and upper confidence bound applied to tree (UCT) algorithm [17–20], temporal difference algorithm [21], the deep learning model combined with UCT search algorithm [22], and DQN algorithm [23, 24], and they have also achieved quite good results in computer Go game, which shows that the idea of deep reinforcement learning algorithm can adapt to the computer game environment. In addition, the application of deep reinforcement learning algorithm in Backgammon [25–27], Shogi [28–30], chess [31–34], Texas poker [35, 36], Mahjong [37], and Star Craft II [38] has achieved human excellence and even exceeded human achievements. There is no doubt that the deep reinforcement learning algorithm will make a good progress in this field when it is applied to Jiu chess.

At present, Jiu chess is little-known to the people because it is mainly spread in Tibetan people gathering areaes. Due to special rules and smaller players compared with Go or other popular games, the research on Jiu chess is very limited. Jiu chess is a complete information game, along with Go and chess. However, completely different from Go or chess, Jiu rules are very special with two consequential stages, which makes the game tree search path very long. There are three Jiu chess programs from all current literatures [1, 2, 39]. And all of them have low chess power although they have different playing levels. In [1], several important shapes of Jiu were first recognized and about 300 playing records were collected and processed. Strategies based on chess shapes were designed to defend the opponent. It was the first Jiu program based on expert knowledge, but it had very low power because of the limited human knowledge. In [39], a Bayesian network model which was designed to solve the problem of small sample data for Jiu playing records estimated the chess board rapidly. This method alleviated the extreme lack of expert knowledge to a certain extent, but the model was only useful in the layout stage. In [2], a time difference algorithm was used to realize the probability statistics and prediction of chess type, which was the first time reinforcement learning was applied to Jiu chess. It was verified that solely applying SARSA(λ) or Q-learning algorithm with special different parameters was helpful to improve the efficiency of recognizing and evaluating chess board. However, it did not make important contributions to the chess power because of not applying a deep neural network or UCT search. Q-learning algorithm is not only used in the games, but also widely used in other fields such as the wireless network to reduce the cost of resources [40]. It is very promising. So in our study, we first combine SARSA(λ) and Q-learning algorithms in different stages of UCT search to help us get the motivation of achieving better performance under low hardware cost.

4. Learning Methodology

UCT search combined with deep learning and reinforcement learning is of high performance for Go, chess, and other games. However, it requires considerable hardware resources to support exploration to the huge state space and training for the deep neural network. To take advantage of reinforcement learning and deep learning while reducing hardware requirements as much as possible, this study proposed an improved deep neural network model combined UCT search, with hybrid online and offline time difference algorithms for Tibetan Jiu chess, which is exquisite and efficient in self-play learning ability.

Hybrid TD algorithms are applied to different stages of the UCT search, which minimizes searching in low-value state spaces. Thereby, the computational power consumption and training time of the exploration state space are reduced. According to the unique characteristics of Jiu chess, a TD algorithm reward function is proposed based on a 2D normal distribution matrix for the layout stage, enabling the Jiu chess reinforcement learning model to more quickly acquire layout awareness of Jiu chess priorities. The reward function for the battle stage is also designed based on Jiu chess shapes. The improved ResNet18-based deep neural network based is used for self-play and training.

4.1. UCT Search

4.1. UCT Search. The reinforcement learning algorithm used in this study is based on a UCT [41, 42] search algorithm (see Figure 2), which combines Q-learning, SARSA(λ), and expert domain knowledge. When the model performs self-play learning, it searches from the root node, uses SARSA(λ) combined with immediate feedback from domain knowledge in the former three stages (selection, extension, and default policy simulation), and uses Q-learning updation in the backpropagation stage.

In the selection, extension, and default policy simulation stages, the board situation is evaluated through expert knowledge and this evaluated value is returned to each node of the path using the SARSA(λ) algorithm updation method (see the bold parts of the routes in the selection, extension, and default policy simulation stages of Figure 2). In the backpropagation stage, the board situation is evaluated through Q-learning algorithm updation (see the bold part of the route in the backpropagation step of Figure 2).

4.2. Hybrid SARSA(λ) and Q-Learning Algorithm. In this study, the node of the UCT search tree is expressed as

$$\{S, \mathbf{W}, \mathbf{N}, \mathbf{P}, V, \mathbf{Q}, \mathbf{E}\}, \tag{1}$$

where S represents the state of the node in the search tree; $\mathbf{W}$ represents the value of each action of the node; $\mathbf{N}$ represents the number of times the action of the node is selected, and for each action a selected, $\mathbf{N}(s,a) \longleftarrow \mathbf{N}(s,a) + 1$; $\mathbf{P}$ represents the probability of selecting each action under the state (calculated by the neural network); V is the winning rate estimated by the neural network; $\mathbf{Q}$ is the winning rate estimated by the search tree; and $\mathbf{E}$ is a variable of auxiliary $\mathbf{Q}$ updation. In the selection, extension, and simulation stages, $\mathbf{E}$ is updated according to the SARSA(λ) algorithm, which is expressed as follows:

$$\mathbf{E}(s,a) \longleftarrow \mathrm{R}(s,a) + \gamma \mathbf{Q}(s_{i+1}, a_{i+1}) - \mathbf{Q}(s,a), \tag{2}$$

where $\mathrm{R}(s,a)$ is the reward value of the current situation which can usually be calculated by the board situation

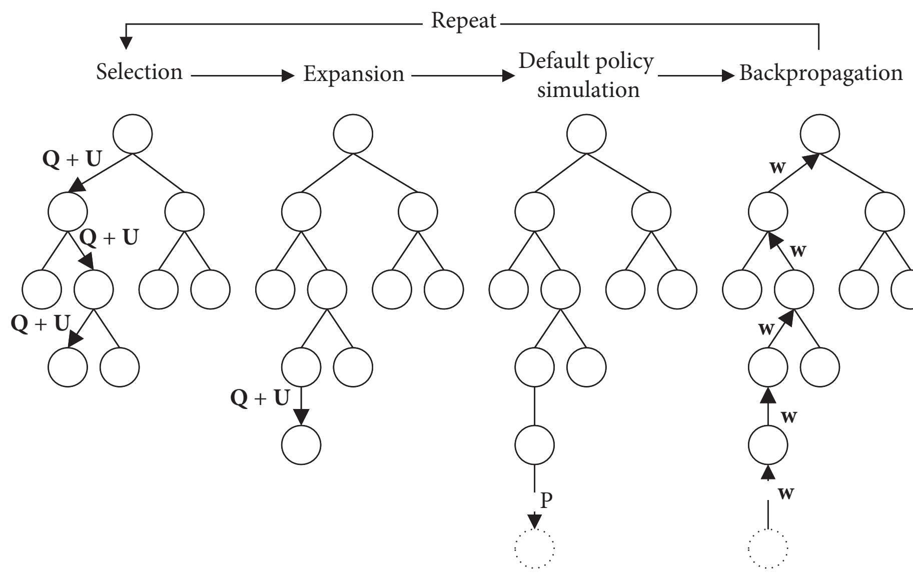

Figure 2: UCT algorithm for Jiu chess.

evaluation, γ is learning rate, s_{i+1} is the next state, searched by taking action a, and a_{i+1} is the action selected in state s_i.

The node (s_i, a_i), which is previously searched in the searching path, is updated in turn by the following equation:

$$\mathbf{Q}(s_i, a_i) \longleftarrow (1-\gamma_1)\mathbf{Q}(s_i, a_i) + \gamma_1 \lambda^{c-i}\mathbf{E}(s,a), \tag{3}$$

where γ_1 is learning rate of SARSA (λ), i is denoted as the state index satisfying $0 \le i \le c$, c is the number of steps from the status of the root node to the status of the search ending at the stage of value return in each turn, $i = 0$ represents the searching is initialized by taking the current situation as the search tree root node, and $i = c$ represents the searching is ended. Especially at the end of the game, c is simply the total number of steps from the beginning to the end of the game.

In the backpropagation stage of UCT search or at the time of each game end, equation (4) is used to update the Q-values of all nodes on the path, where γ_2 is learning rate of Q-Learning:

$$\mathbf{Q}(s_i, a_i) \longleftarrow (1-\gamma_2)\mathbf{Q}(s_i, a_i) + \gamma_2 \max \mathbf{Q}(s_{i+1}, a_{i+1}), \tag{4}$$

and if s_{i+1} is a leaf node, $\mathbf{Q}(s_{i+1}, a_{i+1})$ will be calculated by

$$\mathbf{Q}(s_{i+1}, a_{i+1}) = \frac{\mathbf{W}(s_{i+1}, a_{i+1})}{\mathbf{N}(s_{i+1}, a_{i+1})}. \tag{5}$$

In the selection, extension, and simulation stages of the UCT search algorithm, this hybrid algorithm selects action a with the maximum value from state s through equation (6) by calculating the score of each alternative action:

$$a = \operatorname{argmax}_a (Q(s,\cdot) + U(s,\cdot)). \tag{6}$$

For each alternative action a at state s, there is the comprehensive evaluation function which is represented by the following equation:

$$\mathbf{Q}(s,a) + \mathbf{U}(s,a) = \mathbf{Q}(s,a) + c_{\text{puct}}\mathbf{P}(s,a)\sqrt{\frac{\sum \mathbf{N}(s,\cdot)}{\mathbf{N}(s,a)+1}}, \tag{7}$$

where $\mathbf{Q}(s,a)$ is obtained by equation (3) or (4), which depends on different tree UCT search stages, and $\mathbf{U}(s,a)$ represents the estimation value by the UCB method which is obtained by the following equation:

$$\mathbf{U}(s,a) = c_{\text{puct}}\mathbf{P}(s,a)\sqrt{\frac{\sum \mathbf{N}(s,\cdot)}{\mathbf{N}(s,a)+1}}, \tag{8}$$

where c_{puct} represents the parameter balancing the exploitation and exploration of UCT algorithm.

4.3. Feedback Function Based on Domain Knowledge. Q-value is updated by the combination of the SARSA (λ) and Q-learning algorithms [43, 44]. SARSA (λ) is used to update the nodes on the game path in the selection, expansion, and simulation stages of the UCT [18]. In the backpropagation stage, Q-learning is used to update the values of all nodes in the search path. The hybrid strategy enables the algorithm to learn to prune the huge state space effectively, improving computing speed and reducing the consumption of hardware resources.

The layout stage of Jiu chess plays a key role in the game and its outcome [1]. The value of the board position decays from the center of the board to the outside, which is similar to the probability distribution of a 2D normal distribution matrix. Therefore, the importance of each intersection in the layout stage can be approximated by the 2D discrete normal distribution (see Figure 3).

In the battle stage, constructing the chess shapes discussed in [1, 2] is very important in gaining a victory. The feedback function used by the SARSA (λ) and Q-learning algorithm is shown in equation (9), $\sum_i V_i(s) = V_{\text{chain}} + V_{\text{gate}} + V_{\text{square}}$:

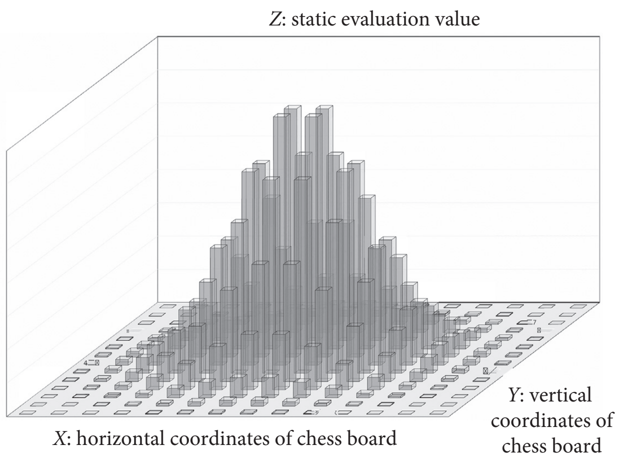

FIGURE 3: Approximate 2D normal distribution matrix.

$$\mathrm{R}(s,a)=\begin{cases} f(x,y), & s \text{ in layout stage,} \\ \sum_i V_i(s), & s \text{ in battle stage,} \end{cases} \tag{9}$$

where $f(x,y)$ is the joint probability density of X and Y is shown in the following equation [39, 45]:

$$f(x,y)=\frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}e^{-\left(1/\left(2\left(1-\rho^2\right)\right)\right)\left(\left(x-\mu_1\right)^2/\sigma_1^2-2\rho\left(\left(x-\mu_1\right)\left(y-\mu_2\right)/\sigma_1\sigma_2\right)+\left(\left(y-\mu_2\right)^2/\sigma_2^2\right)\right)}, \tag{10}$$

where X and Y represent chessboard coordinates and μ represents the mean value of the range of the chessboard $(x, y \in [0,13]$, $\sigma_1=\sigma_2=2$, $\mu_1=u_2=6.5$, $\rho=0)$. V_{chain}, V_{gate}, and V_{square} are approximations summarized through the Jiu chess rules [1, 2] and experience. $V_{\text{chain}}(s)=7c_{\text{chain}}$, where c_{chain} is the sum of the number of squares on the chessboard in the current state; $V_{\text{gate}}(s)=3c_{\text{gate}}$, where c_{gate} is the sum of the number of gates on the chessboard in the current state; and $V_{\text{square}}(s)=4c_{\text{square}}$, where c_{square} is the sum of the number of squares on the chessboard in the current state.

Using the 2D normal distribution approximate matrix probability distribution based on expert knowledge as the feedback function of the TD algorithm in the layout stage, we can produce better search simulations and self-play game performance in the case of deep search depths and large search branches and learn a reasonable chess strategy in the layout stage better and faster [46].

4.4. Improved Deep Neural Network Based on ResNet18. Strategy prediction and board situation evaluation are realized using a deep neural network, which is expressed as a function with parameter θ, as shown in equation (11), where $\mathbf{s}$ is the current state, $\mathbf{p}$ is the output strategy prediction, and v is the value of board situation evaluation:

$$(p,v)=f_{\boldsymbol{\theta}}(s). \tag{11}$$

The RESNET series of deep convolution neural networks shows good robustness in image classification and target detection [3]. In this study, a ResNet18-based network is used to transform the full connection layer. It can simultaneously output strategy prediction $\mathbf{p}$ and board situation evaluation v (see Figure 4).

Unlike ResNet18 [3], a public full connection layer with 4096 neuron nodes is used as the hidden layer. The fully connected layer is replaced by a 196-dimensional output fully connected layer to output drop strategy $\mathbf{p}$ and a one-dimensional fully connected layer to output the estimation v of the current board status. The neural network is used to predict and evaluate the state of nodes that have not previously been evaluated. The neural network outputs $\mathbf{p}, v$ to the UCT search tree node so that $P(s_i)=p, V(s_i)=v$.

The number of input characteristic graphs is two. It means that there are two channels. The first channel is the position and color of the pieces in the chessboard state. White pieces are represented by -1 and black ones are represented by 1. The second channel is the position of the falling pieces in the current state of the chessboard. The positions in which pieces do not fall are represented by 0 and the position of the falling pieces is represented by 1. All convolution layers use ReLu as their activation function and batch normalization is performed after convolution.

At the end of the game, we play back the experience and output the dataset $\{i\le 0\le fc\,|\,(\mathbf{s},\boldsymbol{\pi},z)_i\}$ to adjust the parameters of the deep neural network. The loss function used is shown in the following equation:

$$l=(z-v)^2-\boldsymbol{\pi}^T\ln\mathbf{p}+c\|\theta\|^2. \tag{12}$$

In the i-th state of the path, $\pi_i=N_i/\|N_i\|$ and $z_i=W_i/N_i$. The training of the neural network is carried out

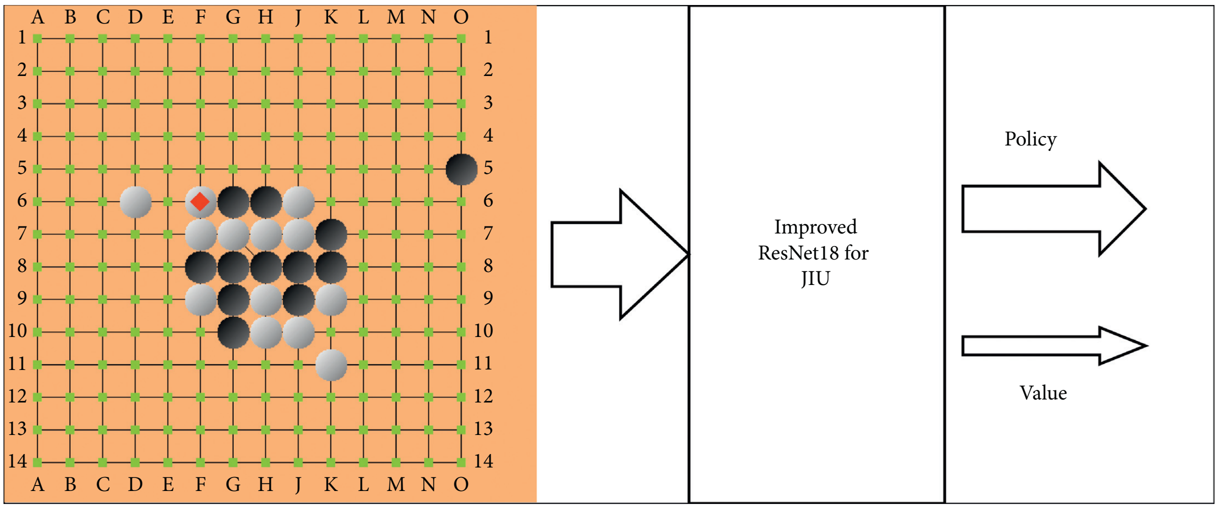

Figure 4: The deep neural network structure.

with a *min-batch* parameter of 100 and the *learning rate* is set to 0.1. At the end of each game, the number of samples (turns) sent to neural network training is about 1×10^3 to 2×10^3.

5. Results

The experiment was performed with the following parameters: c_{puct}, step, and learning rate (see Table 1). We compare the efficiency of three methods of updating the Q value: hybrid Q-learning with SARSA (λ), pure Q-learning, and pure SARSA (λ). The server and hardware configuration used in the experiment is shown in Table 1.

Table 1: The experimental parameters.

Toolkit package	CNTK 2.7
Runtime	.Net 4.7.2
Operating system	Windows 10
Central processing unit	AMD 2700X@4.0 GHz
Random access memory	32 GB
Graphics processing unit	RTX2070
Threads used	16
Development environment	Visual Studio 2017 Community
c_{puct}	0.1
Step	200
Learning rate	0.001
γ(Q-learning)	0.628
γ(SARSA (λ))	0.372

5.1. The Improvement of Learning and Understanding Ability.

It is important to measure the learning ability of a Jiu chess agent that can quickly learn to form squares in the layout stage. Therefore, we calculate the influence of the hybrid update strategy compared to pure Q-learning or SARSA (λ) algorithms on the sum of the squares of 200 steps of self-play game training, as shown in Table 2. When using the hybrid update strategy, the total number of squares in 200 steps is almost the sum of the number of squares occupied when only using pure Q-learning or SARSA (λ), proving that the hybrid update strategy can effectively train a Jiu chess agent to understand the game of Jiu chess and learn it well.

We also counted the number of squares per 20 steps in the 200-step training process. Figure 5 shows that the number of squares occupied when using the hybrid update strategy is significantly more than that when using pure Q-learning or SARSA (λ) algorithms. Especially in first 80 steps, hybrid updating strategy is significantly more effective than pure Q-learning or SARSA (λ) algorithms. And from step 140 to step 200, hybrid updating strategy has a little drop compared to the first 120 steps. It is because parameter c_{puct} at 0.1 must decrease to fit later training to avoid useless exploration. This result indicates that the chess power trained by the hybrid update strategy is stronger than that of pure Q-learning or SARSA (λ). In addition, this strategy produces stronger learning and understanding as well as better stability.

Table 2: Average number of squares per 20 steps in 200-step training.

Q-learning + SARSA (λ)	Q-learning	SARSA (λ)
8604	4248	5571

5.2. Feedback Function Experiment.

In order to improve the reinforcement learning efficiency of Jiu chess, a feedback function based on a 2D normal distribution matrix is added to the reinforcement learning game model as an auxiliary measure for updating value. To test the effect of this measure, we conducted 7 days and 110 instances of self-play training and obtained data with and without the 2D-normal-distribution-assisted feedback mechanism for comparison.

Table 3 and Figure 6 show that the total number of squares occupied when using the 2D normal distribution assistant in the self-play training process is about three times that of programs that do not; the average value is close to

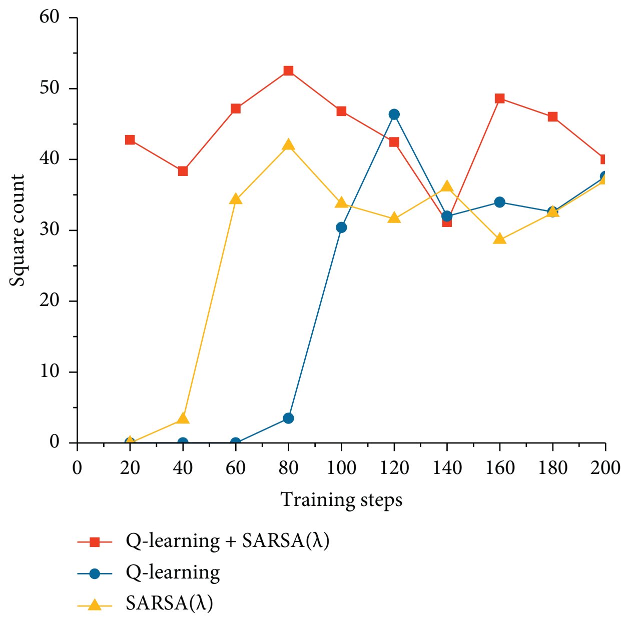

Figure 5: Number of squares employing different algorithms.

Table 3: The number of squares with 2D normalization on or off.

	2D normalization off	2D normalization on
Sum	1524	5005
Average	13	45

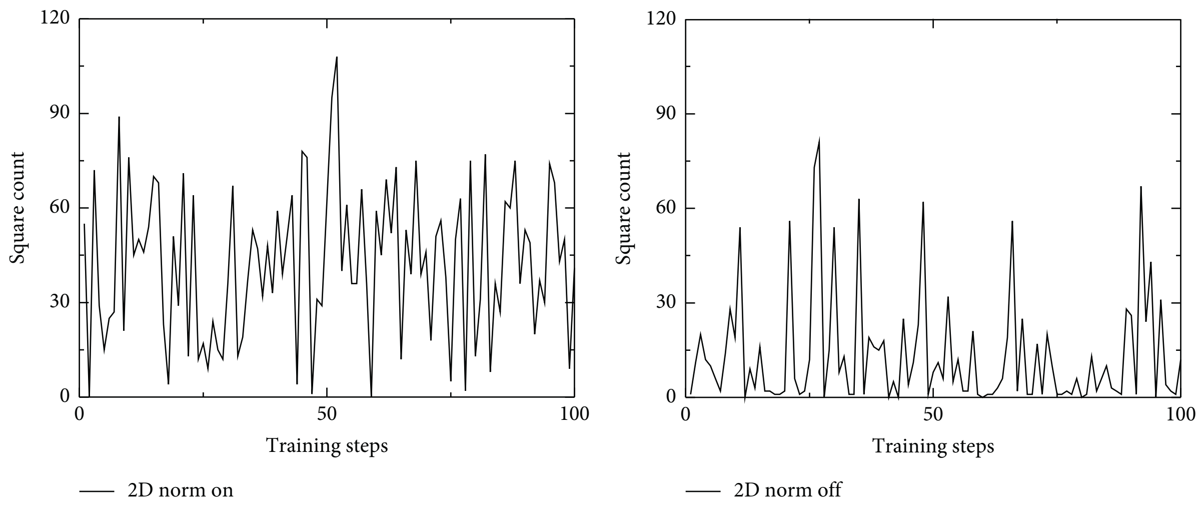

Figure 6: Comparison of the number of squares with or without 2D normal distribution.

three times. The total number of squares occupied when using the 2D normal distribution assistant during training is significantly more than that without the assistant. Two-dimensional normal distribution auxiliary matrix shows the application of expert knowledge in fact. The learning ability to play in the layout has improved about 3 times with the two-dimensional normal distribution auxiliary matrix.

We also compare the learning efficiency of using a 2D normal distribution assistant program from the perspective of the quality of specific layouts. As shown in Figure 7(a), in 10 days of training, the program without the aid of the 2D normal distribution has little knowledge of the layout. After using the 2D normal distribution matrix to assist in training, the layout process can learn specific chess patterns faster,

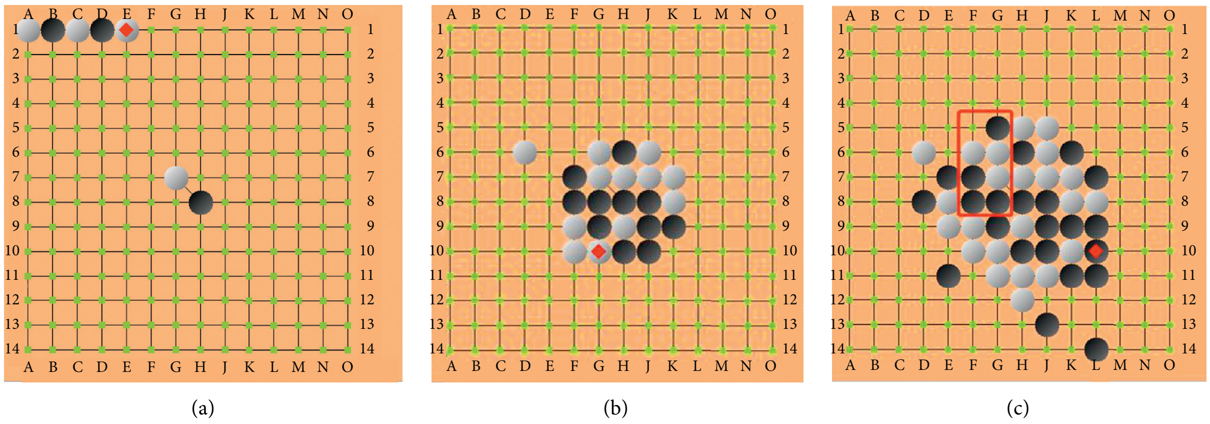

Figure 7: Comparison of learning efficiency with or without 2D normal distribution. (a) 2D normalization off. (b) 2D normalization on. (c) 2D normalization on and making a special board type.

form a triangle (chess gate), and even form squares (the most important chess shape) in the fighting stage of Jiu chess game, as shown in Figures 7(b) and 7(c).

The above experiments show that the feedback function based on the two-dimensional normal distribution matrix can reduce the calculation amount of Jiu chess program and improve the self-learning efficiency of the program because in the layout stage, the importance of the chessboard position is close to the two-dimensional normal distribution matrix. Because Jiu chess can have more than 1000 matching steps and slow search iteration process, the evaluation of layout stage is optimized by using two-dimensional normal distribution matrix, which is actually the knowledge of model experts. The feedback value of the model indirectly reduces the process of blindly exploring the transfer value of chessboard state action, so it takes less time to get better results.

6. Conclusion

In this study, the deep reinforcement learning model, combined with expert knowledge, can learn the rules of the game faster in a short time. The combination of Q-learning and SARSA(λ) algorithms can make the neural network learn more valuable chess strategies in a short time, which provides a reference for improving the learning efficiency of the deep reinforcement learning model. The deep reinforcement learning model has produced good layout results obtained by the two-dimensional normal distribution matrix of expert knowledge modeling, which also proves that deep reinforcement learning can shorten the learning time by combining expert knowledge reasonably. The better performance of ResNet18, which was used to make the deep reinforcement learning model training more effectively at low resources cost, has been verified by experimental results.

Inspired by Wu et al. [47, 48], we consider to improve Jiu chess power not only from the state-of-the-art of technologies but also from the holistic social good perspectives in the future work. We will collect and process much more Jiu chess game data to establish the big data resource which can turn big values for using light reinforcement learning model to reduce the cost of computing resources. We will also explore the possibilities of combining multiagent theory [49] and mechanism [50] to reduce the network delay of the proposed reinforcement model besides using the strategies proposed in [39, 41].

Data Availability

Data are available via the e-mail xiaer_li@163.com.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was funded by the National Natural Science Foundation of China (61873291 and 61773416) and the MUC 111 Project.

References

[1] X. Li, S. Wang, Z. Lv, Y. Li, and L. Wu, "Strategies research based on chess shape for Tibetan JIU computer game," *International Computer Games Association Journal (ICGA)*, vol. 40, no. 3, pp. 318–328, 2019.

[2] X. Li, Z. Lv, S. Wang, Z. Wei, and L. Wu, "A reinforcement learning model based on temporal difference algorithm," *IEEE Access*, vol. 7, pp. 121922–121930, 2019.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[4] D. SuodaChucha and P. Shotwell, "The research of Tibetan chess," *Journal of Tibet University*, vol. 9, no. 2, pp. 5–10, 1994.

[5] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, http://arxiv.org/abs/1312.5602.

[6] D. Stern, R. Herbrich, and T. Graepel, "Bayesian pattern ranking for move prediction in the game of go," in *Proceedings of the 23rd International Conference on Machine Learning*, pp. 873–880, ACM, Pittsburgh, PA, USA, 2006.

[7] W. Dabney, M. Rowland, M. G. Bellemare, and R. . Munos, "Distributional reinforcement learning with quantile regression," 2017, http://arxiv.org/abs/1710.10044.

[8] M. Andrychowicz, W. Filip, A. Ray et al., "Hindsight experience replay," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 5048–5058, Long Beach, CA, USA, December 2017.

[9] F. Scott, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018, http://arxiv.org/abs/1802.09477.

[10] T. P. Lillicrap, J. J. Hunt, P. Alexander et al., "Continuous control with deep reinforcement learning," 2015, http://arxiv.org/abs/1509.02971.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, http://arxiv.org/abs/1801.01290.

[12] M. Babaeizadeh, I. Frosio, T. Stephen, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," 2016, http://arxiv.org/abs/1611.06256.

[13] V. Mnih, A. P. Badia, M. Mirza et al., "Asynchronous methods for deep reinforcement learning," in *Proceedings of the International Conference on Machine Learning*, pp. 1928–1937, New York City, NY, USA, June 2016.

[14] John Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 1889–1897, Lille, France, July 2015.

[15] John Schulman, W. Filip, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, http://arxiv.org/abs/1707.06347.

[16] S.-J. Yen, T.-N. Yang, C. Chen, and S.-C. Hsu, "Pattern matching in go game records," in *Proceedings of the Second International Conference on Innovative Computing, Information and Control (ICICIC 2007)*, p. 297, Washington, DC, USA, 2007.

[17] S. Gelly and Y. Wang, "Exploration exploitation in go: UCT for Monte-Carlo go," in *Proceedings of the NIPS: Neural Information Processing Systems Conference On-Line Trading of Exploration and Exploitation Workshop*, Barcelona, Spain, 2006.

[18] Y. Wang and S. Gelly, "Modifications of UCT and sequence-like simulations for Monte-Carlo go," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 175–182, Honolulu, HI, USA, 2007.

[19] S. Sharma, Z. Kobti, and S. Goodwin, "Knowledge generation for improving simulations in uct for general game playing," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, pp. 49–55, Auckland, New Zealand, December 2008.

[20] X. Li, Z. Lv, S. Wang, Z. Wei, X. Zhang, and L. Wu, "A middle game search algorithm applicable to low-cost personal computer for go," *IEEE Access*, vol. 7, pp. 121719–121727, 2019.

[21] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, "Temporal difference learning of position evaluation in the game of go," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 817–824, Denver, CO, USA, 1994.

[22] D. Silver, A. Huang, C. J. Maddison et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[23] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[24] D. Silver, T. Hubert, J. Schrittwieser et al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[25] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[26] G. Tesauro, "Practical issues in temporal difference learning," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 259–266, Denver, CO, USA, 1992.

[27] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[28] D. F. Beal and M. C. Smith, "First results from using temporal difference learning in shogi," in *Proceedings of the International Conference on Computers and Games*, Springer, Tsukuba, Japan, pp. 113–125, November 1998.

[29] R. Grimbergen and H. Matsubara, "Pattern recognition for candidate generation in the game of shogi," *Games in AI Research*, pp. 97–108, 1997.

[30] Y. Sato, D. Takahashi, and R. Grimbergen, "A shogi program based on monte-carlo tree search," *ICGA Journal*, vol. 33, no. 2, pp. 80–92, 2010.

[31] S. Thrun, "Learning to play the game of chess," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 1069–1076, Denver, CO, USA, 1995.

[32] I. Bratko, D. Kopec, and D. Michie, "Pattern-based representation of chess end-game knowledge," *The Computer Journal*, vol. 21, no. 2, pp. 149–153, 1978.

[33] Y. Kerner, "Learning strategies for explanation patterns: basic game patterns with application to chess," in *Proceedings of the International Conference on Case-Based Reasoning*, Springer, Sesimbra, Portugal, pp. 491–500, October 1995.

[34] M. Campbell, A. J. Hoane Jr., and F.-h. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[35] N. Brown and T. Sandholm, "Safe and nested subgame solving for imperfect-information games," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 689–699, Long Beach, CA, USA, December 2017.

[36] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.

[37] https://www.msra.cn/zh-cn/news/features/mahjong-ai-suphx, 2019.

[38] V. Zambaldi, D. Raposo, S. Adam et al., "Relational deep reinforcement learning.," 2018, http://arxiv.org/abs/1806.01830.

[39] S. T. Deng, "Design and implementation of JIU game prototype system and multimedia courseware," Minzu University of China, Beijing, China, Dissertation, 2017.

[40] X. Chen, J. Wu, Y. Cai, H. Zhang, and T. Chen, "Energy-efficiency oriented traffic offloading in wireless networks: a brief survey and a learning approach for heterogeneous cellular networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 4, pp. 627–640, 2015.

[41] T. Pepels, M. H. M. Winands, M. Lanctot, and M. Lanctot, "Real-time Monte Carlo tree search in ms pac-man," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 245–257, 2014.

[42] C. F. Sironi and M. H. M. Winands, "Comparing randomization strategies for search-control parameters in monte-carlo

tree search," in *Proceedings of the 2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, London, UK, August 2019.

[43] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[44] F. Xiao, Q. Liu, Qi-M. Fu, H.-K. Sun, and L. Gao, "Gradient descent Sarsa (λ) algorithm based on the adaptive potential function shaping reward mechanism," *Journal of China Institute of Communications*, vol. 34, no. 1, pp. 77–88, 2013.

[45] E. Estrada, E. Hameed, M. Langer, and A. Puchalska, "Path laplacian operators and superdiffusive processes on graphs. ii. two-dimensional lattice," *Linear Algebra and Its Applications*, vol. 555, pp. 373–397, 2018.

[46] K. Lloyd, N. Becker, M. W. Jones, and R. Bogacz, "Learning to use working memory: a reinforcement learning gating model of rule acquisition in rats," *Frontiers in Computational Neuroscience*, vol. 6, p. 87, 2012.

[47] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: big data toward green applications," *IEEE Systems Journal*, vol. 10, no. 3, pp. 888–900, 2016.

[48] J. Wu, S. Guo, H. Huang, W. Liu, and Y. Xiang, "Information and communications technologies for sustainable development goals: state-of-the-art, needs and perspectives," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2389–2406, 2018.

[49] B. Liu, N. Xu, H. Su, L. Wu, and J. Bai, "On the observability of leader-based multiagent systems with fixed topology," *Complexity*, vol. 2019, pp. 1–10, 2019.

[50] H. Su, J. Zhang, and X. Chen, "A stochastic sampling mechanism for time-varying formation of multiagent systems with multiple leaders and communication delays," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 12, pp. 3699–3707, 2019.