International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2017, Vol.1, No.2, P.21 - 40

An Approach to the Development of a Game Agent based on SOM and Reinforcement Learning

Keiji Kamei * , Yuuki Kakizoe †

Abstract

Recently, several studies have reported that the computer programs are called the game agent exceed to the ability of experts in some board games, e.g., Deep Blue, AKARA, AlphaGO, etc. Meanwhile, human beings have no advantages in terms of numerical ability compared with computers; however, experts often defeat those programs. For this, the aim of many researches for developing agents of board games is to defeat experts in all kinds of computational ways; hence, those depend on the computational capability because those apply deep look ahead search to determination of moves. By contrast to those researches, our final aims are the development of a board game agent does not require the high computational capability and of an "Enjoyable" game agent is tailored skills for a player based on "Simple structure and Algorithms." To realize our aims, we propose to combine Self-Organizing Maps(SOM) with Reinforcement Learning. For more effective learning of the optimal moves of a board game, our proposal modifies the formula of SOM and introduces the tree search with less calculation load to determine moves in the closing stage. We conduct the two experiments; firstly, we examine the availability of our proposals. Secondly, we aim for improving the winning rate. From the results, the game agent that is developed on the basis of our proposal achieved a 60% winning rate against the opponent program by using the general personal computer. Moreover, those suggest the potential of becoming an "Enjoyable" game agent for every player with diverse skills.

Keywords: Game agent, Self-Organizing Maps, Reinforcement learning, Reversi(Othello) game

1 Introduction

Recently, researches that create agents which play board games, e.g., Chess, Shogi, and the others, have been studied actively. "Deep Blue" developed by IBM[1] is known for having won to the champion of Chess. The reason of winning to its champion is that Deep Blue was able to explore the next move deeper than him because that agent was constituted by a supercomputer. It would have to say the feat of strength of a supercomputer.

"AKARA" developed by Information Processing Society of Japan(IPSJ)[2] is famous for strong Shogi agent. AKARA consists of some strong agents for Shogi, and that decides

* Nishinippon Institute of Technology, Fukuoka, Japan
† Nishinippon Institute of Technology, Fukuoka, Japan

a next move based on council system. So, it can be said that AKARA is one of cluster system. “Bonanza”[3], which is one of the agent in “AKARA”, is said that it has the ability of learning. Bonanza was applied the reaction equations of chemistry to evaluation of next move, and that equation was called the ability of learning. The one of factor of learning capability of Bonanza is that there is a gradual change in a board position in Shogi by one move, and the evaluation function is formed in the same manner as alternation of board positions. This fact means that an agent of Shogi is able to approximate an evaluation function directly from changes of board positions. Another example of the transition of board positions as same as Shogi is Backgammon. Tesauro[4][5] have constructed the agent of Backgammon, which was named “TD-Gammon,” and the agent acquired the ability of winning to its champion.

“AlphaGO” developed by Google DeepMind[6] is the agent of Go, and that is the most recent successful research. AlphaGO is constructed by Deep Neural Network(DNN) and reinforcement learning(RL). That proposal uses 1202 CPUs and 176 GPUs for training of DNNs and deep lookahead search; hence, AlphaGO needs the high performance computers.

Human beings have no advantages in terms of numerical ability compared with computers; however, experts sometimes defeat those agents. For this, the aim of many researches for developing agents of board games is to defeat experts in all kinds of ways. By contrast to those researches, our final aims are the development of a board game agent does not require the high computational capability and of an “Enjoyable” game agent is tailored skills for a player based on “Simple structure and Algorithms.”

In this research, we aim the development of the agent that plays Reversi game. The feature of Reversi game is that it has the simple rule and a lot of states, though board positions change drastically. Hence, magnitude at the time of evaluation of a board position do not necessarily be same as evaluation after a move. For this reason, it is difficult to learn the optimal evaluation function from variations of board positions; therefore, the agent has to map variations of board positions to the space which changes gradually from drastically space. Moreover, most of board games have “The curse of dimensionality.” That means state explosion for learning because placements of pieces are expressed as input data which were quantified a board position in general. Common approaches for solving that problem reduce the information by means of such as search tree.

To accomplish our aim, we propose to apply reinforcement learning(RL)[7] to the agent because the algorithm of RL learns the best moves according to situations. Meanwhile, the curse of dimensionality becomes a major problem in RL. To overcome the curse of dimensionality and the problem of drastic variation of board positions in Reversi, we propose to apply Self-Organizing Maps(SOM)[8] to memorizing of board positions. SOM has some abilities below. It maps high dimensional input space to low dimensional space; hence, that ability will solve the curse of dimensionality. Another ability of SOM is that it maps input space to competitive layer with conserving topology of input space. Consequently, it is expected to be possible to reduce of the dimensions of input space for RL and to approximate to the gradual variation space by SOM; therefore, an agent is able to learn efficiently the optimal moves by using RL. Our proposal might look in the same way as the approach of AlphaGO. However, Silver et. al.[6] presents that AlphaGO applies DNN and RL in order to search efficiently and deeply of moves by using Monte Carlo tree search. By contrast to AlphaGO, our proposal uses only simple tree search in the closing stage in order not to rely on search based on computational capability. Hence, we apply SOM to only reduction and mapping of learning space and RL to only learning of the optimal moves. In addition, our proposal is easily applicable to other board games with similar rules of Reversi.

The problem of memorizing of Reversi by SOM is memory retrieval. In our proposal, a competitive layer neuron in SOM presents board positions, and an element in each weight(reference) vector in a competitive neuron expresses the states of a square, i.e., a black disk, a white disk and an empty square. Due to continuous weight values in each competitive neuron, we have to determine the appropriate thresholds for properly retrieval of the state of squares. Otherwise, an agent is not able to make appropriate decisions for a move by RL. We do not, however, have prior information to determine the thresholds; hence, the thresholds for determination of states in each square becomes necessary. In our former study[9][10], we have attempted three methods for determination of that thresholds; hence, we cite the essential part of those papers in this paper. One of the method is fixed values which are set by us. Another one is that thresholds are determined by statistical information. This approach assumes that variation of squares follows Gaussian distribution. Finally, we propose to adopt a genetic algorithm(GA)[13] for determination of appropriate thresholds, since a GA is good at global search in a high dimensional space. Another approach is that we have introduced the modification for training algorithm of SOM[11][12] in order to avoid the problem of threshold determination for states of squares. This paper is a continuation of the [12]; for this, this paper cites from [12] such as the methods. The determination of the winner neuron of the conventional SOM is on the basis of Euclidean Distance. In contrast to the conventional SOM, the determination for the winner neuron in this study is on the basis of the inner product, is called the IP-SOM. This method ignores empty squares because empty squares correspond to 0 values; therefore it is expected that an agent is able to choose the highly reproducible neuron. In addition, the thresholds for determination of state of all of squares are 0 because this modified method considers only the occupied squares. In our previous study[12], we have applied 3 kinds of SOMs to reduction of dimensions, i.e., SOM-1 for beginning, SOM-2 for middle and SOM-3 for closing stage. From the results of that study, we think difficult to memorize board positions using 3 kinds of SOMs. For this, we propose to use 7 kinds of SOMs to memorize board positions and to reduce the dimensions. Especially, we split the closing stage finely because the winning rate rely on accurate retrieval of board positions in closing stage from rule of Reversi.

As mentioned in above this section, we apply RL to learning of the optimal moves. Several algorithms of RL are proposed. We apply TD(0)-learning, MC-learning and TD(λ)-learning algorithm to learning of playing for Reversi due to the importance of evaluations for board positions. The feature of TD(0)-learning is that an agent learns for each time step by predicted rewards from a result of posterior action. In contrast to TD(0)-learning, MC-learning does not apply rewards immediately. An agent that learns by using MC-learning does not learn for each time step in an episode, since rewards are determined after termination of an episode and that are propagated to all of visited states in an episode. TD(λ)-learning is like a method of combining TD(0)-learning with MC-learning. The evaluations of moves are propagated from the closing to the beginning stage due to the algorithm of RL. For this, the determination of moves in the closing stage is key to improve the winning rate; therefore, we introduce the tree search to determine the best move within the last 4 moves in this paper. From the results of the implementation of improvement proposals in above, the agent learned by TD(λ)-learning with the IP-SOM achieved a 60% winning rate against the opponent program by using the general personal computer. Additionally, the agent of our proposal learns the optimal moves from only relations between moves and results, so that the skills of that will be almost equally with a player in the end. Consequently, it can be said that our game agent will become an “Enjoyable” game agent for every Reversi player with diverse skills.

2 Feature of Reversi game

Reversi game is one of the Two-players, Zero-sum, Logical Perfection Information Game, and that is pretty much same as Othello game. The face of a board is split into 8×8, 64 squares in total. 4 squares in central are placed by two white and two black disks in advance. One side of a disk is painted by white, and the another side is by black. Two players place a white and a black disk alternatively, and a placing act is called a "move." A player have to place a disk to the square that is able to hold the disks of opponent player by two disks of a player. Then, the sandwiched disks are inverted. Players are prohibited to move disks to other squares or to take out disks on a board. The condition for the end of a game is that the all of squares are filled by disks, the color of all of disks are same, or a player is not able to hold disks of opponent player. The winner is the player who took more squares than the opponent player. The Figure 1 illustrates an example of a Reversi game.

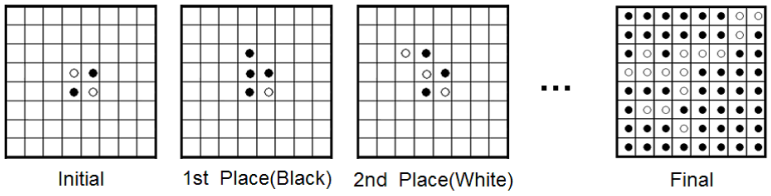

Figure 1: An example of a Reversi game. "Initial" stands for the first board position. "1st Place" and "2nd Place" stand for the move of black and the move of white, respectively. "Final" stands for a board position of the end of a game.

Feinstein presents in [14], the mini-Reversi game that is split into 6×6 squares has been solved. From his research, the possible board positions in mini-Reversi are about 3.6 trillion patterns; hence, it can be said that Reversi game has a tremendous amount of possible board positions. For this reason, we propose to reduce the dimensions for the states of Reinforcement Learning by Self-Organizing Maps(SOM).

3 Reduction of dimensions

We present the method of reduction of input space for learning and retrieval of board positions, in this section.

3.1 Memorizing and reduction of board positions by SOM

Self-Organizing Maps(SOM)[8] is the one of machine learning and clustering method. SOM has two layers that are the input layer and the competitive layer, and SOM maps high dimensional input space to low dimensional competitive layer with preserving topology of input space. This feature is suitable for memorizing of board positions and for reduction of input space for learning of the optimal moves of Reversi by Reinforcement Learning.

The input layer has one neuron, and the competitive layer has some neurons. The all of neurons in the competitive layer are connected to the input layer neuron. The neuron in the input layer has the input vector, $\mathbf{x}$, and each neuron in the competitive layer has the weight(reference) vector, $\mathbf{w}$; additionally, dimension of the input vector and the weight vector is same. The Figure 2 illustrates the overview of SOM.

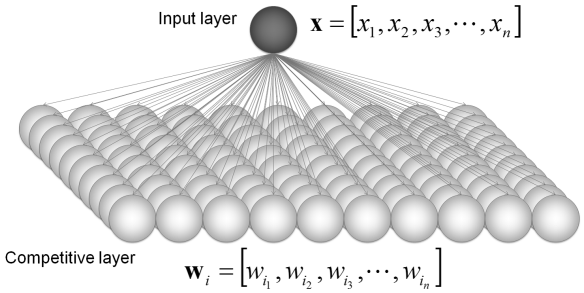

Figure 2: The overview of SOM. n is the number of elements. i is the neuron number in competitive layer.

The algorithm of SOM minimizes the distance norm between each vector of the input pattern and weight vector of the winner neuron that corresponds to each input; then, the neighborhood neurons of the winner neuron are also adjusted based on that input pattern. The conventional SOM uses the Euclidian distance for determination of the winner neuron, and the competitive layer is constructed by 2 dimensional space as the Figure 2.

The method for determination of the winner neuron is discussed in the Section 3.3. The update formula for weight vectors of the winner and the neighborhoods is as follows.

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{w}_i(t)] \tag{1}$$

where i is the neuron number in the competitive layer, t is the number of iterations, and $h_{ci}(t)$ is the function for determination of the area for neighborhoods as the step function in the Equation 2 or as the Gaussian function in the Equation 3.

$$h_{ct}(t) = \begin{cases} \alpha(t) : i \in N_c \\ 0 : i \notin N_c \end{cases} \tag{2}$$

$$h_{ct}(t) = \alpha(t) \cdot \exp\left(-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma_\Lambda(t)^2} \right) \tag{3}$$

where $\alpha(t)$ is a monotonically decreasing function for training rate. N_c is the set of neighborhood neurons, and the size of N_c varies between the initial radius, r, and the its final, r'. $\mathbf{r}_c$ and $\mathbf{r}_i$ are coordinates of the winner neuron and *i-th* neuron, respectively. $\sigma_\Lambda(t)$ is as the Equation 4.

$$\sigma_\Lambda(t) = r \cdot \left(1 - \frac{t}{T_{\max}}\right) + 1 \tag{4}$$

where r is the initial radius for the set of neighborhood as mentioned above, $t_{\max}$ is the sum of training iterations. We apply the Gaussian function in the Equation 3 and 4 to training of SOM in this study.

Each square of Reversi has three states, i.e., occupied by black disk, by white disk and empty. In this study, the black disks, the white disks and the empty squares correspond to 1, -1 and 0, respectively. An element in each weight vector in each competitive neuron of SOM represents the state of corresponding square of a board in Reversi; therefore, a neuron in SOM has 64 dimensional vector because Reversi is played on a board is split into 64 squares. The Figure 3 illustrates the method of memorization of board positions of Reversi by using SOM.

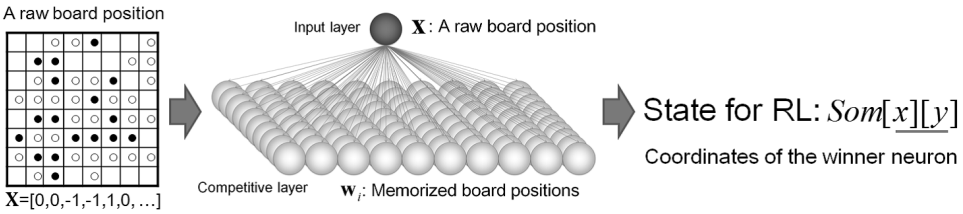

Figure 3: Memorizing of board positions of Reversi. SOM memorizes board positions, and the states for RL are the coordinates of the winner neuron.

We apply 2 dimensional SOM to memorizing of board positions, here. Therefore, our proposal reduces the dimensions of learning space in RL to 2 dimensional vector from 64 dimensional vector.

We prepare some SOMs for memorizing board positions, and each SOM memorizes particular board positions that are assigned depending on the move number. For example, SOM-1 memorizes between the initial board position and 10-*th* move, and SOM-2 memorizes between 11-*th* and 20-*th* move, etc. The reason of separation of training is that the properties of board positions among training stages are so different, e.g., almost no variation in the beginning stage and wide variation after the middle stage. The Figure 4 illustrates an example of separation of training.

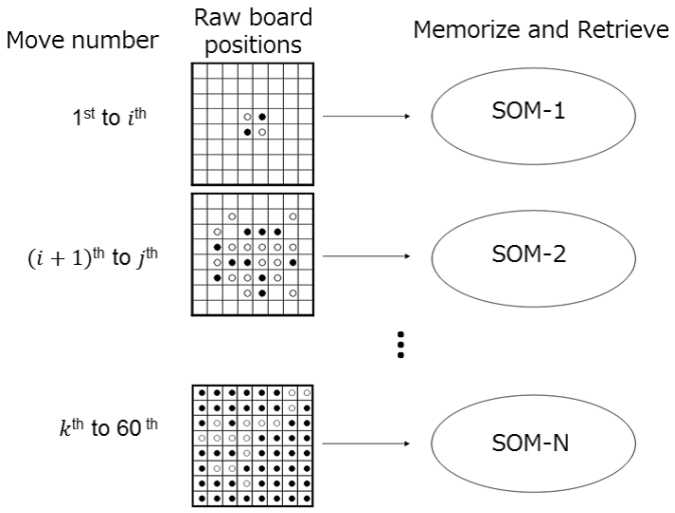

Figure 4: Separation of training. "SOM-1" memorizes the initial to *i-th* moves, "SOM-2" memorizes the $(i+1)$-*th* to *j-th* moves, etc.

3.2 Retrieval of board positions from SOM

We have to determine the appropriate thresholds for which distinguish between a white disk, a black disk and an empty square because elements in each weight vector in each competitive neuron of SOM are real number. The thresholds have to be determined separately; the reason is that elements in each weight vector in each competitive neuron which correspond to the corner squares are almost zero because that elements are changed only once in a game. On the other hand, elements in each weight vector in each competitive

neuron which correspond to the squares of the center of a board vary significantly because the states of that squares change variously from the rule of Reversi. For this reason, it can be said that the thresholds that are determined identically will lead to several undesirable effects on retrieval of board positions.

We have proposed[9][10] the method for determination of thresholds for retrieval of board positions of Reversi from SOM in former research. One of that was statistical method, and another one was determined by a genetic algorithm(GA).

The statistical method assumes that variation in squares on a board follows Gaussian distribution. From the assumption, the average values for each element in each weight vector in each competitive neuron will be 0 from law of large numbers because elements in that take randomly 0, 1, and -1. Moreover, the standard deviation for each element in each weight vector in each competitive neuron will be larger than 1.0 if the rate of empty in each square of board will be less than about 68%; otherwise, the standard deviation for each element in that will be less than 1.0. Hence, we are able to scale each element in weight vectors in each competitive neuron to normal distribution from the assumption, so that we are able to determine the thresholds from the standard deviation.

Another one is that we have proposed to adopt a genetic algorithm(GA) for determination of the appropriate thresholds, since a GA is good at global search in a high dimensional space. A chromosome is binary coded, and that is composed of thresholds which discern states of squares of Reversi. The length of a chromosome is 2048 bits, with 32 bits in each threshold. All thresholds are transcoded to real number by linear scale. In our proposal, it is difficult to efficient search by a GA because the chromosome for each individual is so long. To improve search performance, we have proposed to search independently in each square of a board. For example, one of the individual searches the best threshold of decision of state for a square, and the other individuals search the best thresholds for other squares. The thresholds which are not under consideration are fixed values determined by the statistical method in the first generation or the results of previous search of a GA after that.

3.3 Memorizing and retrieval based on inner product

The main cause of the problem of determination for thresholds is that the conventional SOM focuses on the all of squares in a board position. For this, we attempt to modify of the measure of the distance for SOM. The new measure is intended to focus solely on the placed squares in a board position. The determination of the winner neuron of the conventional SOM is on the basis of Euclidean Distance as equation as follows,

$$c = \arg\min_i \|\mathbf{w}_i - \mathbf{x}\| \tag{5}$$

where c is the winner neuron, i is the neuron number, $\mathbf{w}_i$ is the weight(reference) vector and $\mathbf{x}$ is the input vector. In the conventional SOM, the winner neuron is chosen from the all of squares of a board, including the empty squares.

In contrast to the conventional SOM, the determination for the winner neuron in our proposal is on the basis of the inner product, called the IP-SOM. The determination formula for the winner neuron in the IP-SOM is as follows.

$$c = \arg\max_i [\mathbf{w}_i \cdot \mathbf{x}] \tag{6}$$

where the valuables are same as the Equation 5.

The IP-SOM ignores empty squares because empty squares correspond to 0 values in our proposal; therefore it is expected that the board game agent is able to choose the highly reproducible neuron. In addition, the thresholds for determination of state of all of squares are 0 because the IP-SOM considers only the occupied squares. The Figure 5 illustrates the differences between two methods.

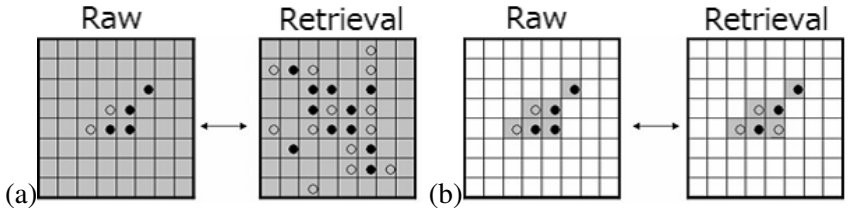

Figure 5: Differences between two methods. (a) is the conventional SOM. (b) is the IP-SOM. The gray squares show the squares are subject to the determination of the winner.

4 Pre-Process of Retrieval

Some board positions of Reversi are often same as the other symmetrical or rotationally-symmetrical board positions. To reduce the number of memorizing, such positions should be treated as the same. In our proposal, an input vector will be rotated if the comparison results between that vector and a retrieved board position do not fully match; and then, an input vector are compared with a retrieval position in the same manner. The upper number of rotation is 8, which an input vector is rotated 90, 180, 270 degrees and invert besides. The Figure 6 illustrates an example of rotation processes.

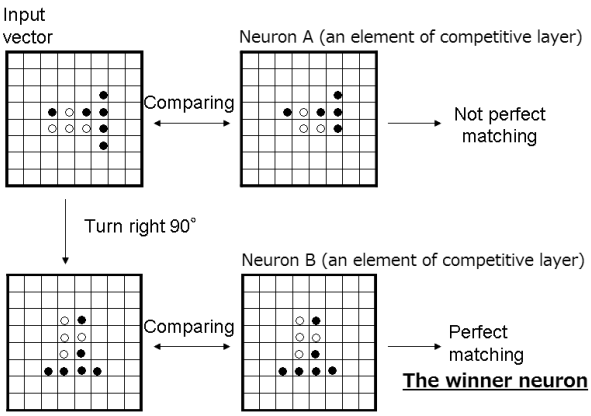

Figure 6: An example of rotation of an input vector(board position)

If all of comparison results do not fully match, the winner neuron is determined by distance from the Equation 5 or 6.

5 Reinforcement Learning

In this research, the agent learns to play Reversi by Reinforcement Learning. Reinforcement Learning(RL)[7] is the one of the machine learning algorithm which does not require

supervised signals. From the rule of Reversi, the agent and a player placed alternately a disk, and then the agent observes rewards or punishments from a situation. The aim of the agent is maximization of the sum of rewards, so that the agent optimizes moves to accomplish that aim. The framework of RL has the reward function, the value function and the policy function. The reward function determines the rewards or punishments, and the value function stores the sum of rewards or punishments. The agent takes a move(action) based on the policy function. In our research, the policy function is set by ε-greedy function. The agent under this policy takes moves(actions) which are the highest value in the value function in general; however, the agent takes random moves(actions) in probability ε. We set that probability, ε, to 0.1. The learning methods of RL in this research are temporal difference learning(TD-learning) and Monte Carlo learning(MC-learning), and the details of those methods is shown as following sections. We compare the differences of performance in learning between the learning algorithms by experiment in the Section 7.

5.1 Temporal Difference learning(TD(0)-learning)

Temporal-Difference (TD) learning is the most popular algorithm in RL. Several algorithms of temporal difference are proposed, e.g., TD, Q, Sarsa, etc. We apply TD(0) and TD(λ) algorithm to learning due to the importance of evaluations for board positions. In this section, we describe the details of the algorithm of TD(0)-learning. The feature of TD(0)-learning is that the agent learns for each time step by the observed rewards and the predicted values of the state from a result of posterior action. For this reason, TD(0)-learning is disadvantageous in that the state values in the beginning stage of an episode fluctuate; on the other hand, the agent is able to explores the optimal moves(actions) widely. The equation for updating of the state values in TD(0)-learning is as follows,

$$V(s_t) = V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \tag{7}$$

where $V(s_t)$ is the state value. t is the time step number, r_{t+1} is the reward, α is the learning rate and γ is the discount rate. In this research, α is 0.4, γ is 0.9998 and rewards is described in the Section7.2.2.

The agent learns step by step from the beginning stage of a game in Reversi game; therefore we think that it is difficult to learn the optimal moves(actions) for the beginning stage of a game by using TD(0)-learning if the evaluation of moves in closing stage is unstable.

5.2 Monte Carlo learning(MC-learning)

In contrast to TD(0)-learning, MC-learning does not apply rewards immediately to learning. The agent that learns on the basis of MC-learning does not learn for each time step in an episode, because the rewards are determined after termination of an episode and that are propagated to all of visited states in an episode. Hence, the agent based on MC-learning learns after the end of each game. In general, it is said that MC-learning is not suitable for learning of the optimal moves(actions) in practice because the termination of an episode is not guaranteed. However, it is not problem for learning because the number of moves in Reversi is at most 60 in one game. In addition, the state values that correspond to the all of the moves in a game are updated concurrently; therefore, the agent based on MC-learning is able to learn the optimal moves(actions) in the beginning stage of a game compared with the

agent based on TD-learning. The equation for updating of the state values in MC-learning of this research is as follows,

$$V(s_t) = V(s_t) + \frac{R_t}{t} \tag{8}$$

where $V(s_t)$ is the value of state at the time t, R_t is the sum of the rewards at the time t.

5.3 TD(λ)-learning

TD(λ)-learning is like a method of combining TD(0)-learning with MC-learning. The state values are updated by the observed rewards, the predicted value of the posterior state and the results of a few prior moves(actions). The equation for updating of the state values in TD(λ)-learning is as follows,

$$\begin{aligned} &V(s_t) = V(s_t) + \alpha\delta e_t(s) \\ &\delta = r + \gamma V(s_{t+1}) - V(s_t) \\ &e_t(s) = \begin{cases} \gamma\lambda e_t(s) & (if\ s \neq s_t) \\ \gamma\lambda e_t(s) + 1 & (if\ s = s_t) \end{cases} \end{aligned} \tag{9}$$

where λ is the decay parameter; $V(s_t)$, r, α and γ are same as TD(0)-learning. The decay parameter, λ, is set to 0.8; so, $e_t(s)$ of former 10 actions is about 0.1.

6 The flow of learning the playing of Reversi

The flow of learning the optimal moves is as the Figure 7. As mentioned in the Section 5, the value function in RL is updated by TD-learning and MC-learning. The agent is learned by TD-learning updates that function for each move; on the other hand, the agent is learned by MC-learning updates that function for each game.

7 Experiments

This section describes the experimental conditions and its results. We conduct the two experiments; firstly, we examine the availability of our proposals. Secondly, we implement the methods for improving of the winning rate.

7.1 Conditions

In first, we glean the massive amount of the record data of board positions from the results of playing Reversi; secondly, the dimensions of learning space for RL are reduced by SOMs as described as the Section 3.1. The training data are generated by observed games that played against between computer programs. The program of playing Reversi for generation of the training data for SOM plays a game according to the rule of Reversi; therefore, the training data do not contain the unrealistic board positions.

The opponent program for the game agent has no strategy to win Reversi; therefore, that chooses a move randomly from within the squares which meet the rule of Reversi. In simulations, the agent is Black(first move), and the opponent program is White(second move). The opponent program is more advantageous than the agent because it is said that White have the advantage from the results of mini-Reversi.

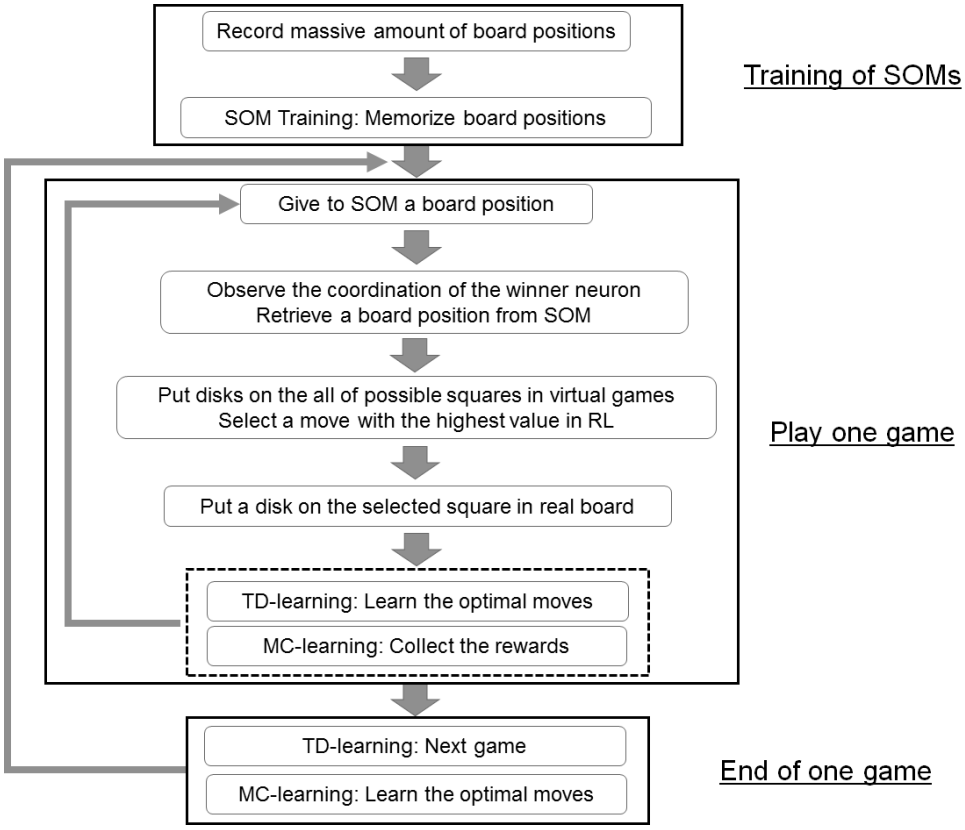

Figure 7: The flow of learning the playing of Reversi.

It is possible that a neuron in the IP-SOM represents the several board positions because only the occupied squares are subject to determination of the winner neuron. The Figure 8 illustrates an example that a neuron wins for several board positions.

From this, we apply the count of moves to states of RL in case of using the IP-SOM. On the other hand, a neuron in the conventional SOM has the temporal information due to the rule of Reversi; hence, the states of RL are only the coordinates of the winner neuron in case of the conventional SOM.

We use the general computer; that has an Intel Core i7-2600 @ $3.4GHz$ and $8.00GB$ memory. That CPU has the ability of the parallel computation based on Hyper-Threading Technology and Multi-Cores. We do not, however, apply multi-threading for SOM and RL.

7.2 Experiment 1: 3 kinds of SOM without tree search

This experiment is same as our previous study; hence, this section cites the experimental results in [12].

In the first experiment, we apply the three kinds of SOM to reduction of the learning space of RL. The number of training data is 100,000 games. A training data is split into 3 part; data of first to 25-th moves give to SOM-1, 26-th to 45-th moves give to SOM-2 and 46-th to 60-th moves give to SOM-3. The size of competitive layers for each SOM are 50 $\times$ 50(2500 neurons). SOMs are trained in twice; SOMs memorize roughly board positions by the first training, then those memorize finely by the second training. The Table 1 shows the parameters of training of SOMs.

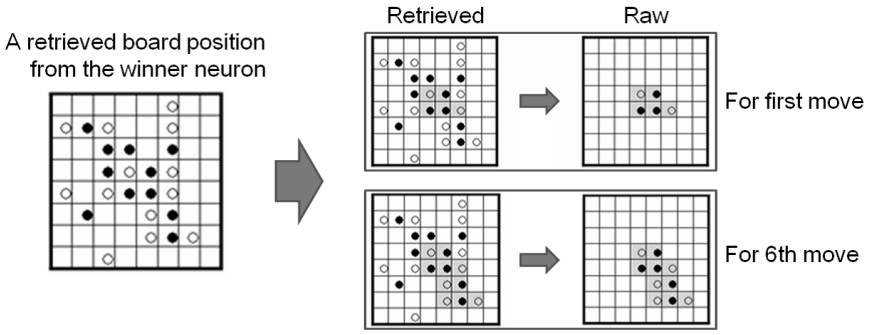

Figure 8: An example that a neuron wins for several board positions. The gray squares show the squares are subject to retrieval of board position.

Table 1: Parameters for learning by 3 SOMs. "# data" stands for the number of data. "# epochs" stands for the number of training iterations. "α" stands for the training rate. "r" and "r'" stands for the initial and final size of neighborhood, respectively.

		# data	# epochs	α	r	r'
1st training	SOM-1	18000	90000	0.1	50	3
	SOM-2	14000	70000	0.1	50	3
	SOM-3	10124	60000	0.1	50	3
2nd training	SOM-1	18000	900000	0.01	3	1
	SOM-2	14000	700000	0.01	3	1
	SOM-3	10124	600000	0.01	3	1

7.2.1 Results of retrieval from 3 SOMs

In this section, we show the experimental results of the retrieval of board positions of Reversi from SOM. The method of validation of the retrieval performance of board positions is illustrated as the Figure 9.

We give a raw board position to SOM, and then find the winner neuron from the manner of the Section 3.1 and 4. A board position is retrieved from the weight(reference) vector based on the Section 3.2 and 3.3. After the retrieval, we compare the retrieved board position with that raw of that. In case of the retrieval based on the IP-SOM(inner product SOM), the retrieved squares are only the occupied squares in raw board positions at that time; hence, we compare retrieved placement of disks with raw placement of that. For comparison, we also prepare the fixed thresholds that are same for all of squares.

Those methods are validated by 300 games, i.e., the number of all of moves are about 18,000. In the experiments, one of the comparison is the number of different squares which include empty squares in a board between raw and retrieved. Another one is the number of different squares which are only the occupied squares. The former one is intended to compare the all of squares in a board, and the latter one is intended to focus on accurately retrieving the color of disks. The Table 2 shows the results of experiments of retrieval, and that shows the rate was retrieved definitely. In case of the IP-SOM, the retrieval rate for all of squares is not able to calculate because that SOM focuses on only occupied squares. For this, the result of the IP-SOM for board positions is not assigned in the Table 2.

From the results of retrieval of board positions, the method of statistical approach is the best for retrieval the all of squares; on the other hand, the worst is the method of a GA. By contrast, the method of a GA and the IP-SOM are the best for retrieval of the color of disks.

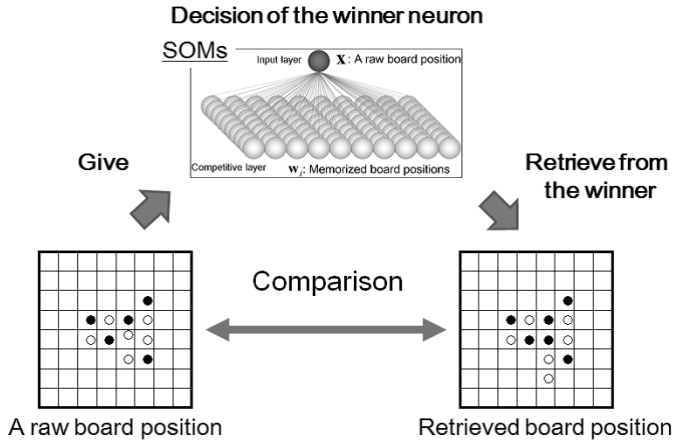

Figure 9: The method of validation for the retrieval performance of Reversi

Table 2: The results of the retrieval experiments. "Statistical" stands for the thresholds determined by statistical approach, "GA" stands for the thresholds determined by a GA, "IP" stands for the IP-SOM, and "Fixed" stands for the thresholds determined by us. "Board Positions" stands for compared the all of squares. "Disks" stands for compared the different disks. "NA" stands for Not Assigned.

	Statistical	GA	IP	Fixed		
				0.1	0.5	0.8
Board Positions	0.76	0.45	NA	0.74	0.72	0.66
Disks	0.68	0.86	0.86	0.68	0.58	0.44

The fixed(set by us) thresholds do not show good performances in the all of experiments.

The average number of different squares between raw and retrieved board positions are 4.76 squares in the IP-SOM. By contrast, those of the conventional SOM with statistical thresholds are 15.33 squares. The IP-SOM shows the better retrieval performance than the conventional SOM because the IP-SOM retrieves only occupied squares.

From the all of experimental results, the thresholds determined by statistical information attach importance to the all of squares; on the other hand, the thresholds determined by a GA attach importance to the occupied squares. In addition, the retrieval performance of the IP-SOM is as same as a GA, even though thresholds are no need to be set individually for each square.

We apply the statistical method and the IP-SOM to the agent to learn the optimal moves, in follow experiments. As these reasons, the statistical method is suitable for learning the optimal moves from board positions, and the IP-SOM is the simple approach because that does not require determination of thresholds of each square.

7.2.2 Results of winning rate of 3 SOMs

We compare the winning rate between the methods of RL in this section. In the comparison, we apply two types of SOM in order to learn board positions, i.e., the conventional SOM with statistical thresholds and the IP-SOM. The Table 3 shows the common parameters for RL. An episode corresponds to one game, i.e., first to last move.

The Table 4 shows the differences of the winning rate between methods of learning. The

Table 3: The common parameters for reinforcement learning.

Number of games	Rewards		
	Win	Lose	1 move
1000000	1000	−1000	−0.0001

results of the winning rate are averaged over the number of learning games, i.e., 1,000,000 games.

Table 4: The comparison of winning rate. TD(0) stands for TD(0)-learning, MC stands for MC-learning, TD(λ) stands for TD(λ)-learning. "Conventional SOM" stands for the method of the conventional SOM with statistical thresholds. "IP-SOM" stands for the inner product SOM.

	TD(0)	MC	TD(λ)
Conventional SOM	0.47	0.56	0.48
IP−SOM	0.47	0.58	0.50

The results show that the agent which is learned by MC-learning is the best results in both methods. The reasons of those results are explained as below. An agent learned by TD(0)-learning updates only the current state value from the predicted values of the next states and the rewards. In contrast to TD(0)-learning, an agent with MC-learning updates the state values that correspond to the all of moves in a game. In our proposal, the agent is given the big rewards at the end of a game, and that is given the small rewards in the other. The differences between TD(0) and MC-learning affect in performance of learning for the beginning stage in a game because the state values are propagated from the last moves. For this, the agent which learns on the basis of the TD(0)-learning is difficult to determine the optimal moves in the beginning stage. In case of TD(λ)-learning can be also said the same as TD(0)-learning. On the other hand, the agent that learns on the basis of MC-learning is able to learn the optimal moves and to decide it in the beginning stage.

The winning rate for a player which plays with Black disks is about 47% from our previous study[11] if players place the disks randomly in each other; in addition to that, Feinstein[14] suggests that White player has the advantage from the results of mini-Reversi. Thus, the agent learned by TD(0)-learning is not able to learn the optimal moves at all, and the agent learned by TD(λ)-learning is able to learn a little bit of the optimal moves because the algorithm of TD(λ)-learning uses few posterior rewards or punishments. By contrast to TD-learning, the agent learned by MC-learning is able to learn the optimal moves to some degree.

The results of the experiment 1 show reasonable efficacy of our proposal. For this, we implement the methods for improving of the winning rate in the next experiment.

7.3 Experiment 2: 7 kinds of SOM with tree search

In the second experiment, our aim is improvement of the winning rate. We only apply the IP-SOM in this experiment because the IP-SOM shows the best score from the result of the experiment in the Section 7.2.2.

The reason for hardly learning by TD-learning is that retrieval of board positions in the closing stage are unstable; hence, the state values are not propagated to the beginning stage correctly. Since, we split a training data for SOMs into 7 part in order to improve

the correctness of retrieved board positions. The training data for SOMs is split equally 3 parts in the previous experiment. In contrast, we split that data into 7 part unequally in this experiment, i.e., data of first to 18-*th* moves give to SOM-1, 19-*th* to 30-*th* moves give to SOM-2, 31-*st* to 40-*th* moves give to SOM-3, 41-*st* to 46-*th* moves give to SOM-4, 47-*th* to 52-*nd* moves give to SOM-5, 53-*rd* to 56-*th* moves give to SOM-6 and 57-*th* to 60-*th* moves give to SOM-7. The reason for separation is that board positions for memorizing are few patterns in the beginning stage; on the other hand, board positions for memorizing increase drastically in accordance with the increase of move number. The Table 5 shows the parameter for training of SOMs.

Table 5: Parameters for learning by 7 SOMs. "# data" stands for the number of data. "# epochs" stands for the number of training iterations. "α" stands for the training rate. "r" and "r'" stands for the initial and final size of neighborhood, respectively.

		# data	# epochs	α	r	r'
1st training	SOM-1	1900000	9500000	0.1	50	3
	SOM-2	1200000	6000000	0.1	50	3
	SOM-3	1000000	5000000	0.1	50	3
	SOM-4	600000	3000000	0.1	50	3
	SOM-5	600000	3000000	0.1	50	3
	SOM-6	400000	2000000	0.1	50	3
	SOM-7	350000	1750000	0.1	50	3
2nd training	SOM-1	1900000	95000000	0.01	3	1
	SOM-2	1200000	60000000	0.01	3	1
	SOM-3	1000000	50000000	0.01	3	1
	SOM-4	600000	30000000	0.01	3	1
	SOM-5	600000	30000000	0.01	3	1
	SOM-6	400000	20000000	0.01	3	1
	SOM-7	350000	17500000	0.01	3	1

7.3.1 Results of retrieval from 7 SOMs

In this section, we examine the effectiveness of fine separation of the training data. The manner of the examination is same as the previous experiment in the Section 7.2. The Table 6 shows the result of retrieval from 7 SOMs.

Table 6: The retrieval rate from 7 SOMs.

Move number	[1, 10]	[11, 20]	[21, 30]	[31, 40]	[41, 50]	[50, 60]
Retrieval rate	0.99	0.85	0.76	0.70	0.65	0.67

From the Table 6, the retrieval rate in the beginning stage is high. That rate between the first move and 20-*th* move is over 85%. By contrast, that rate in the middle and closing stage is low. That rate after 41st move is about only 65%. The reasons of those results are that the number of patterns for board positions in case of 50-*th* move are less than $2^{50} \approx 1Peta$ patterns; hence, it can be said that it is difficult to memorize the board positions after 40-*th* move.

In case of TD-learning, the evaluations of moves are propagated from the closing to the beginning stage due to the algorithm of TD-learning. For this, unstable retrieval in the closing stage is the cause of hardly learning the optimal moves; hence, the determination of moves in the closing stage is the key to improve for the winning rate.

7.3.2 Results of the winning rate of 7 SOMs with tree search

From the results of the Section 7.3.1, the method that uses only SOM occurs inevitably the different squares between raw and retrieved board positions. To overcome the problem of influence of unstable retrieval in the closing stage, we introduce the simple tree search to determine the optimal moves; namely, the agent executes full search for finding the best move within the last 4 moves. Consequently, it is able to solve instability of the evaluation in the closing stage due to the problem that the agent wins or loses from a same board position. The Figure 10 shows an example of a board position for starting full search, and the Table 7 shows the results.

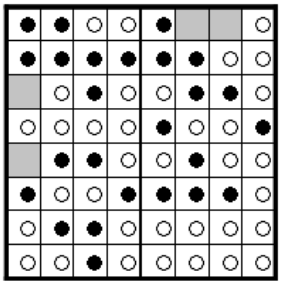

Figure 10: An example of board position for starting full search. The gray squares show the empty.

Table 7: The comparison of the winning rate. MC stands for MC-learning, and TD(λ) stands for TD(λ)-learning.

	MC	TD(λ)
3 SOMs without search	0.58	0.50
7 SOMs with search	0.58	0.60

From the Table 7, although the number of SOMs are increased, the winning rate of MC-learning remain unchanged; moreover, the search within the last 4 moves does not affect to learn of the optimal moves. By contrast to MC-learning, the winning rate of TD(λ)-learning increases by 10% with increasing numbers of SOM and introducing the search within the last 4 moves. As a result of the improvement, TD(λ)-learning is the best performance.

The reasons for those results are below. The algorithm of MC-learning in the Equation 8 splits the sum of rewards in an episode that is first move to last move equally; therefore, the agent takes a move so as to make board positions that won in the past games. For this, the agent follows similar moves, and moves are hardly changed after the agent learns the optimal moves to some degree. Additionally, the evaluation of moves does not change drastically from the Equation 8. In addition to above, the search within the last 4 moves do not affect to performance because the agent learns for each game.

In contrast to MC-learning, the algorithm of TD(λ)-learning updates the some past moves while gradually reducing the update value from the Equation 9. For this, the agent learned by TD(λ)-leaning is able to learn the optimal moves if the evaluation of moves in the closing stage is stable. In the previous experiment, the agent does not use search for the optimal moves in the closing stage; on the other hand, the training data for SOMs are split finely, and the agent uses search in that stage, so that the evaluation of moves in that

stage becomes stable. In addition, the agent explores widely the optimal moves compared with MC-learning because the algorithm of TD(λ)-learning updates the evaluation for each move. Consequently, the winning rate of TD(λ)-learning is higher than MC-learning.

8 Related Works

We presented several examples of board game agents, i.e., chess, shogi, backgammon and Go, in the introduction section. We introduce, here, several proposals of Reversi game agent. There are 2 types of proposal for Reversi game agent as with agents for the other board games. One of them is the method that depends on the computational capability, so that method aims to discover the best moves with searching deeply and widely on search tree. Rocki[15] et. al. proposed to apply the TSUBAME 2.0 supercomputer to discovering the best moves based on Parallel Monte Carlo Tree Search. The TSUBAME 2.0 supercomputer is constructed by many CPUs and GPUs, and so their proposal searches the best moves deeply and widely on search tree by parallel processing with massive threads. Strnad[16] et. al. proposed similar approach to above. Their approach executes the parallel Alpha-Beta pruning based on PV-split algorithm on the GPU with many threads. Olivito[17] et. al. proposed to implement the Reversi agent to Field-Programmable Gate Array(FPGA). The feature of FPGA is that a certain processing such as loop processing is faster than the general computer because it generally executes the processes in parallel. For this, FPGA is good at tree search as with GPU; therefore, their proposal applies FPGA to "Iterative deeping search" to search tree in order to discover the best move in each move.

Another type of proposal is optimization approach of the evaluation function for a move based on the result of moves. Benbassat[18] et. al. proposed to optimize the evaluation function by using Genetic Programming(GP). The individuals in their proposal represent as the evaluation function for board positions based on some parameters. Their proposal, in addition, is not only GP but also using Alpha-Beta pruning with parallel computing to achieve good results; therefore, it can be said the hybrid system which combines GP with parallel computing. Chong[19] et. al. proposed to combine the Evolutionary Algorithms with Multi-Layer Perceptron(MLP) to optimization of the evaluation function of board positions. An individual is a MLP that evaluates the board positions. Their proposal also applies minimax tree search to discovering the best moves.

By contrast to above proposals, our proposal applies SOM to reduction of learning space for RL in order to solve the problem of "Curse of dimensionality," so that our proposal realizes to be able to learn the optimal moves by using RL. Besides, to the best of our knowledge, our proposal that combines SOM with RL to develop Reversi game agent is unprecedented approach.

9 Discussions

The aim of many researches for developing board game agent is to defeat the experts in all kinds of ways. For this, many approaches rely on computational capability for deep search shown as the Section 8. One of our aim is to develop the game agent does not depend on the computational capability. Firstly, we discuss the requirement of computational capability for our proposal. As mentioned in the Section 7.1, we use the general computer, i.e., Intel Core i7-2600 @ 3.4GHz and 8GB memory. Moreover, our proposal does not apply parallel computing. Training period for SOMs requires about 3 hours, and learning period for RL

requires about 1 day. The time to select a move is almost 0 second because our proposal does not use deep and wide search. For this, it can be said that our proposal does not depend on computational capability.

Secondly, we discuss the adaptation abilities of our proposal for the other board games. The evaluations of board positions in our proposal do not depend on the rule of Reversi because the framework of our proposal do not apply rules of Reversi to learning for the optimal moves. Namely, the agent learns the optimal moves from only the relations between moves and results. For this, our proposal is applicable to Reversi-like games without modifications. Furthermore, if board positions are able to be vectrized, our proposal is applicable to the other games with few modifications because of the simple structure and algorithms.

Meanwhile, the agent in this study is not so strong under present circumstances. However, it is enough possible that the agent becomes more strong if that executes simple search for each move from the results of the Section 7.3.2. It is said that the experts of Reversi search about 10 moves in general; hence, we intend to implement the same search for each move. We think that the implementation of that search is possible because the computation cost for simple search is enough small. In addition, our proposal is possible to construct an "Enjoyable" game agent is tailored for skills of a player. The reason is that the agent learns the optimal moves from only relations between moves and results except for last 4 moves, so that the skills of the agent will be almost equally with a player in the end.

10 Conclusions

In this paper, we have proposed to develop the board game agent, and the target game is Reversi(Othello). The feature of proposed agent is that the agent memorizes and retrieves on the basis of SOM and it learns the optimal moves using RL. In retrieval processes, we have to determine the threshold for decision of states of squares because the weight vectors of SOM are real number. Our proposals are that the thresholds are determined by statistical information from the recorded data, by a GA, and by us(fixed). In addition, we have proposed to modify the training algorithm of SOM, called the IP-SOM. The conventional SOM determines the winner neuron based on Euclidean distance. Our proposal, on the other hand, determines that neuron based on the inner product; therefore, our proposal ignore empty squares because those squares correspond to 0. Consequently, the thresholds for decision of states of squares are not required.

Several algorithms in RL are proposed, e.g., Monte Carlo learning(MC-learning) and Temporal Difference learning(TD-learning). The algorithm of MC-learning is that the agent learns for each game. By contrast to MC-learning, the algorithm of TD-learning is that the agent learns for each move, and the final evaluation that is determined by a result of a game is propagated from the last move. From the feature of MC-learning, the agent is able to learn the optimal moves even if the evaluations in the closing stage are not stable. On the other hand, the evaluations and retrieved board positions in the closing stage have to be stable in case of TD-learning.

We conduct the two experiments; firstly, we examine the availability of our proposals. Secondly, we implement the methods for improving of the winning rate. The results of the first experiments are following. The thresholds determined by statistical information are the best for retrieval of board positions. On the other hand, the retrieval rate for the color of disks is that the thresholds determined by a GA and by the IP-SOM are the best performance. In particular the latter result, the performance of decision of disk colors

is improved about over 20% in comparison with the other methods. In the processes of learning of the optimal moves, we have applied TD(0)-learning, TD(λ)-learning and MC-learning to the agent. The results show that the agent which is learned by MC-learning is the best results. In contrast to MC-learning, the agent learned by both of methods in TD-learning is not able to learn the optimal moves at all because the evaluations and retrieved board positions are not stable. From the experimental results, the agent learned by MC-learning achieved a 58% winning rate against the opponent program which chooses moves randomly. The winning rate for a player which plays with black disks is about 47% from our previous study if players place the disks randomly in each other. For this, the results of the first experiment show reasonable efficacy of our proposal.

The results of the second experiments are below. The reason for hardly learning of the optimal moves by TD-learning is that retrieval of board positions in the closing stage is unstable; hence, the state values are not propagated to the beginning stage correctly. Since, we split the training data for SOM into 7 part in order to improve the retrieval of board positions. From the results of experiments, the retrieval rate in the beginning stage is high. That rate between the first move and 20th move is over 85%. By contrast, that rate in the middle and closing stage is low. That rate after 41st move is about only 65%. The method that uses only SOM occurs inevitably the different squares between raw and retrieved board positions. To overcome the problem of influence of unstable retrieval in the closing stage, we introduce the simple tree search to determine the optimal moves; namely, the agent executes full search for finding the best move within the last 4 moves. The winning rate of MC-learning remain unchanged; moreover, the search within the last 4 moves does not affect to learn of the optimal moves. By contrast to MC-learning, the winning rate of TD(λ)-learning increases by 10% with increasing numbers of SOM and introducing the search within the last 4 moves. As a result of the improvement, TD(λ)-learning is the best performance. Finally, the agent learned by TD(λ)-learning achieved a 60% winning rate against the opponent program that has no strategy to win by using the general personal computer. The agent of our proposal learns the optimal moves from only relations between moves and results except for last 4 moves, so that the skills of the agent will be almost equally with a player in the end. Consequently, it can be said that our game agent will become an "Enjoyable" for every Reversi player with diverse skills.

Acknowledgments

This research was supported by Grant-in-Aid for Special Research Equipment from Nishinippon Institute of Technology, 2008.

References

[1] M. Campbell, "Deep Blue," Artificial Intelligence, vol. 134, issues 1-2, Jan., 2002, pp. 57–83

[2] T. Obata, T. Sugiyama, K. Hoki and T. Ito, "Consultation Algorithm for Computer Shogi: Move Decisions by Majority," Computers and Games: 7th Int'l Conf.(CG 2010), Springer, 2011, pp.456–465

[3] K. Hoki, "A New Trend in the Computer Shogi: Application of Brute-force Search and Futility Pruning Technique in Shogi," Joho Shori, vol.47, no.8, 2006, pp.884–889

[4] G. Tesauro, “TD-Gammon, a Self-teaching Backgammon Program, Achieves Master-level Play,” Neural Computation, vol. 6, no. 2, 1994, pp.215–219

[5] J. B. Pollack and A. D. Blair, Why did TD-Gammon Work?, Advances in Neural Information Processing Systems 9, MIT Press, 1997, pp.10–16

[6] D. Silver et al., “Mastering the Game of Go with Deep Neural Networks and Tree Search,” Nature, vol. 529, 2016, pp.484–489

[7] R. S. Sutton and A. G. Barto, Reinforcement Learning, MIT Press, 1998

[8] T. Kohonen, Self-organizing Maps : Second edition, Springer, 1997

[9] Y. Kakizoe and K. Kamei, “Reduction of Dimensions by SOM for Creating a Game Agent,” Proc. The 21st Annual Conference of the Japanese Neural Network Society(JNNS2011), 2011, pp.212–213

[10] Y. Kakizoe and K. Kamei, “Determination of the Appropriate Thresholds for Memory Retrieval of Reversi from SOM,” Proc. The 6th Int’l Conf. on Soft Computing and Intelligent Systems and the 13th Int’l Symp. on Advanced Intelligent Systems(SCIS&ISIS 2012), 2012, pp.473–476

[11] Y. Kakizoe and K. Kamei, “Development of Othello Agent by Reinforcement Learning and SOM,” Proc. The 22nd Annual Conference of the Japanese Neural Network Society(JNNS2012), CD-ROM, 2012

[12] K. Kamei and Y. Kakizoe, “An Approach to the Development of a Game Agent based on SOM and Reinforcement Learning,” Proc. IIAI Int’l Congress on Advanced Applied Informatics 2016(IIAI AAI 2016), 2016, pp. 669–674

[13] R. Pfeifer and C. Scheier, Understanding Intelligence, MIT Press, 1999

[14] J. Feinstein, Perfect play in 6x6 Othello from two alternative starting positions; http://www.feinst.demon.co.uk/Othello/6x6sol.html

[15] K. Rocki and R. Suda, “Large-scale parallel monte carlo tree search on GPU,” Proc. IEEE Int’l Symposium on Parallel and Distributed Processing Workshops and Phd Forum(IPDPSW), 2011, pp. 2034–2037

[16] D. Strnad and N. Guid, “Parallel alpha-beta algorithm on the GPU,” Computing and Information Technology, vol. 19, no. 4, 2011, pp.269–274

[17] J. Olivito, C. González and J. Resano, “FPGA implementation of a strong Reversi player,” Proc. The 2010 Int’l Conference on Field-Programmable Technology(FPT’10), 2010, pp.507–510

[18] A. Benbassat and M. Sipper, “Evolving board-game players with genetic programming,” Proc. the 13th annual conference companion on Genetic and evolutionary computation(GECCO’11), 2011, pp.739–742

[19] S. Y. Chong, M. K. Tan and J. D. White, “Observing the evolution of neural networks learning to play the game of Othello,” IEEE Transactions on Evolutionary Computation, vol.9, no. 3, 2005, pp.240–251