Smart Media Journal / Vol.9, No.4 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2020.9.4.36

Hyper-parameter Optimization for Monte Carlo Tree Search using Self-play

Jin-Seon Lee*, Il-Seok Oh**

Abstract

The Monte Carlo tree search (MCTS) is a popular method for implementing an intelligent game program. It has several hyper-parameters that require an optimization for showing the best performance. Due to the stochastic nature of the MCTS, the hyper-parameter optimization is difficult to solve. This paper uses the self-playing capability of the MCTS-based game program for optimizing the hyper-parameters. It seeks a winner path over the hyper-parameter space while performing the self-play. The top-q longest winners in the winner path compete for the final winner. The experiment using the 15-15-5 game (Omok in Korean name) showed a promising result.

Keywords : Machine learning | Hyper-parameter optimization | Monte Carlo tree search | m-n-k game

I. INTRODUCTION

In game industry, it is not unusual to use pattern analysis or machine learning[1]. Many problems of machine learning that is impossible or difficult to solve using deterministic method uses the Monte Carlo method that relies on a stochastic process using random number generator[2]. The Monte Carlo tree search (MCTS) is one of the most famous instance of the Monte Carlo methods in AI. The MCTS is a heuristic and stochastic tree search algorithm that is used specifically to develop a powerful game program[3,4]. Initially the Monte Carlo idea applicable to game playing was invented under the name of adaptive multi-stage sampling (AMS) to make Go program[5]. In 2006, Coulom coined the MCTS and developed the Go-playing program Crazy Stone[6]. The Deep Mind team took the MCTS as an essential idea in developing AlphaGo program that defeated world Go champion Sedol Lee[7] and in later developing AlphaGo Zero that defeated AlphaGo by 100:0[8].

In implementing an intelligent game playing program, machine learning community has used for a long time the minimax tree search algorithm that needs a good evaluation function[9]. Unlike the minimax that relies on the breadth-first search, the MCTS does not need any evaluation function and is free from the breadth-first search. The computation and memory of the breadth-first search increases exponentially as the search depth increases. When the branching factor is large, the computation becomes infeasible. The branching factors of the Go game played on 19*19 board and the 15*15*5 game that is used for our experiment are 361 and 225, respectively. Due to high branching factors of these games and difficulty of devising reasonable evaluation functions, the traditional minimax algorithm is inapplicable.

The idea of the MCTS is simple. The core operation of the MCTS is to generate lots of random playouts and choose the position with the highest winning rate as the best move [3,4]. Since a playout is made by random choice, the playout generation is very fast and enough number of playouts can be generated under a time constraint. When more playouts are required, parallel processing using GPU or multi-core is easily applicable since playouts can be generated independently.

Three hyper-parameters should be considered.

* Member, Professor, Department of Information and Security, Woosuk University.
** Member, Professor, Division of Computer Science and Engineering, Jeonbuk National University.

Manuscript : 2020. 09. 10
Confirmation of Publication : 2020. 10 25
Corresponding Author : Il-Seok Oh,
e-mail : isoh@jbnu.ac.kr

The playout generation is actually controlled by UCT (upper confidence bound applied to trees) that balances the exploration and exploitation. The first parameter is related to this balancing. The UCT equation has a hyper-parameter λ which gives a more weight on the exploration when it is large. The λ is an important hyper-parameter of MCTS. Another hyper-parameter is related to choosing the best move. We have two options in choosing the best move, taking the move with the highest winning rate or the move with the largest number of playouts generated. The third hyper-parameter is the number of total playouts generated. When allowing a more playouts, more accurate estimation of the best move is possible while requiring more computation time that may violate the game rule constraining the time given to each player. These three hyper-parameters should be optimized in order to obtain the best performance of the MCTS-based game programs.

Generally a machine learning algorithm needs the hyper-parameter optimization[10]. It is known that the random search is better than the grid search [11]. The procedure of the hyper-parameter optimization for the classification models such as SVM, CNN, and RNN is easy since their learning process is deterministic. The only step with a stochastic nature is the weight initialization. The only difficulty is the large amount of computation time due to combinatorial explosion when the number of hyper-parameters is large. On the contrary, the MCTS has only three hyper-parameters, but hyper-parameter optimization is difficult since the process of choosing the best moves is stochastic.

Wang et al. proposed a hyper-parameter optimization algorithm[12]. They experimented on a framework including MCST and other functionalities like neural network. Due to the complexity of the hyper-parameter optimization, they used a very simple 6*6 Othello game. Ruijl et al. proposed a hyper-parameter optimization algorithm for MCST process[13]. The limitation of the work is that only one hyper-parameter controlling the exploration and exploitation (λ in Eq.1) is optimized.

This paper proposes a method for optimizing simultaneously three hyper-parameters of MCTS. The core idea of the proposed method is to use the self-playing capability of MCTS-based game programs. Two program copies with different hyper-parameter settings compete until the winner is decided. The winner is recorded in a *winner path*. Then a copy with newly generated hyper-parameter setting is made. The last winner and the new copy compete until the winner is decided. The winner is recorded in the winner path. This process is iterated for a long sequence and a long winner path is obtained. We regard the winners surviving a long subsequence in the winner path as the competent hyper-parameter settings. The top-q longest subsequences are taken from the winner path. They compete each other as a second round of competition and the final winner in the second round is decided to be the optimal hyper-parameter.

To verify the effectiveness of the proposed method, we use the $m*n*k$ game as testbed problem, in which two players takes turn in placing their stones like O and X on $m*n$ board. The winner is the player who first places k consecutive stones in a row horizontally, vertically, or diagonally. Specifically we used the 15*15*5 game, whose Korean name is Omok meaning five stones. The experiments showed a promising result that identified successfully a competent player with optimal hyper-parameter setting.

Section 2 explains the MCTS algorithm and three hyper-parameters. Section 3 describes our proposed algorithm for optimizing three hyper-parameters. Section 4 presents $m-n-k$ game and Section 5 gives experimental results. Section 6 concludes the paper.

II. MCTS ALGORITHM AND HYPER-PARAMETERS

1. Principle of MCTS

When the artificial intelligence (AI) playing a game against human takes its turn at a game state, it should choose the best move at the state to win the game. The state becomes the root R of the search tree. The children of R are the possible moves from the state and the mission of MCTS is to choose the best one among the possible moves. The MCTS

iterates n times the four steps: selection, expansion, simulation, and backpropagation[3,4]. In each of the iterations, a playout is generated and the data of the nodes in the search tree regarding the number of visits and number of winnings are updated using the information of the playout. The four steps are briefly described as follows.

1. Selection: Starting from R, select successively the child nodes until a leaf node L is reached. The leaf node is a node where the playout has never been started from. The selection is performed using the UCT equation that will be later described in this section. The UCT-based child selection is the core of the MCTS algorithm.

2. Expansion: Among children of L, select randomly one of them, which will be denoted as C. The node C is created and linked to the parent L with downward and upward pointers in order to track down and up the tree. Note that the node C has two fields storing the number of wins and the number of visit. We will denote two fields w/v in Fig.1 where w and v represent win and visit, respectively.

3. Simulation: Starting from C, a playout is made by randomly selecting successively next moves until the winner is decided.

4. Backpropagation: Using the playout, update the w/v information for the nodes on the upward path from C to R. When the O is winner of the playout, the nodes corresponding to O's turn increment both of two fields w and v while the nodes for X's turn increment only v. When the X is winner of the playout, the nodes corresponding to X's turn increment w and v while the nodes for O's turn increment only v. Fig.1 illustrates a cycle of four steps with an example situation.

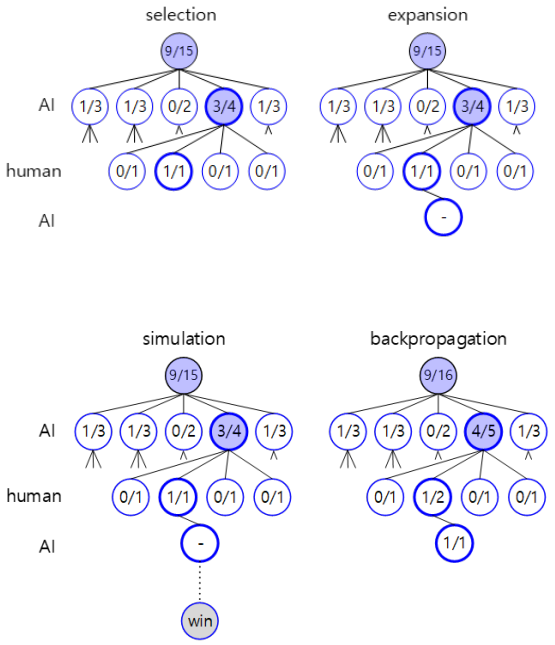


Fig. 1. Illustration of 4 steps of MCTS

In the selection step, the UCT plays an essential role in balancing exploration and exploitation. Eq.1 is the equation used by UCT. The w_i and v_i represents w and v of i-th child and n is the total number of playouts simulated.

$$\frac{w_i}{v_i} + \lambda\sqrt{\frac{\log(n)}{v_i}} \qquad (1)$$

The first term $\frac{w_i}{v_i}$ that is the winning rate forces the exploitation since a child with a better winning rate gets higher value. The second term $\sqrt{\frac{\log(n)}{v_i}}$ forces the exploration since a child being less visited has a lower v_i and results in a higher value. The weight λ controls which term should influence more on the final value.

2. Optimization of hyper-parameters

Three hype-parameters are found in MCTS algorithm. The first one is the weight factor λ in UCT equation (1). A large value of λ encourages the exploration while a small value weighs the exploitation. The usual recommendation is $\sqrt{2}$. However depending on the game, the optimal value may vary. The second hype-parameter is the type

of choosing the best move. Two options are available, one choosing the child with the highest winning rate, i.e., highest w/v and the other with the highest visits, i.e., highest v. The third hyper-parameter is the number of playout simulation. The more the playout is generated, the more accurately the best move is chosen. However more playouts result in more computation time that might violate the game rule of time constraint.

Optimization of hyper-parameters in machine learning is crucial for getting the optimal performance. Two strategies are usually used; grid search and random search. The grid search divides each hyper-parameter in equal intervals or logarithmic intervals. The grid search evaluates each of combinations of hyper-parameters and choose the best combination as optimal value of hyper-parameters. The random search iterates the generation and evaluation of a random combination until the allowed computation time is exhausted. Bergstra's extensive experiments showed a superiority of the random search[11].

The existing algorithms for optimizing hyper-parameters including Bergstra's study work only for the deterministic learning situations. The MCTS works stochastically, so we need a new algorithm for optimizing the above-mentioned three hyper-parameters.

III. ALGORITHM FOR HYPER-PARAMETER OPTIMIZATION OF MCTS

The proposed algorithm is based on the self-playing capability of MCTS. The competitions of two copies of MCTS programs with different settings of hyper-parameters are performed many times to get a long sequence of winner path. The winner at current game participates in the next game. The opponent of the next game is the copy of MCTS with a new hyper-parameter setting randomly generated. The algorithm records the winners during a long sequence of the self-play. Therefore a long consecutive subsequence of a winner appearing in the winner path represents a competitive hyper-parameter combination. The top-q longest winners are identified in the winner path and a league among the q winners is performed as a second round of competition and the winner in the league is regarded as the final optimal hyper-parameter.

Algorithm 1 explains the proposed algorithm for hyper-parameter optimization of MCTS. In lines 2 and 7, three hyper-parameter values are set by random number generation. In line 6, the winner survives and participates to the next game.

Algorithm 1: hyper-parameter optimization for MCTS
Input: MCTS game program
Output: optimal hyper-parameters
1. winner_path=[ ]
2. generate two hyper-parameter combinations c1 and c2 using random sampling.
3. while the allowed time remains
4. compete c1 and c2 and decide a winner cw.
5. append cw to winner_path.
6. c1=cw
7. generate one hyper-parameter combination c2 using random sampling.
8. identify the top-q longest winners in winner_path.
9. perform a league among q winners and take the winner in the league as final output.

In the league of line 9, a pair of winners compete p times and the winning statistics are recorded in a $q*q$ table A on which a_{ij} means the number of winning of the winner i against the winner j. Horizontal sum is calculated and the winner with the maximum sum is regarded as the winner of q winners and hyper-parameter values of the winner is the final output of the algorithm.

IV. $m-n-k$ GAME AS A TESTBED

In our experiment, we use the $m-n-k$ game as a testbed. The game is played on $m*n$ board and the player who first put the k consecutive stones horizontally, vertically, or diagonally wins the game. The tic-tac-toe is a special case of 3-3-3 game. The Omok is 15-15-5 game, which is popular in Asian countries. Fig.2 shows an instance of mini-sized Omok, 10-10-5 game played by AI with the stone X against a human with the stone O. The human is the first mover. At 25th move, the human won the game.

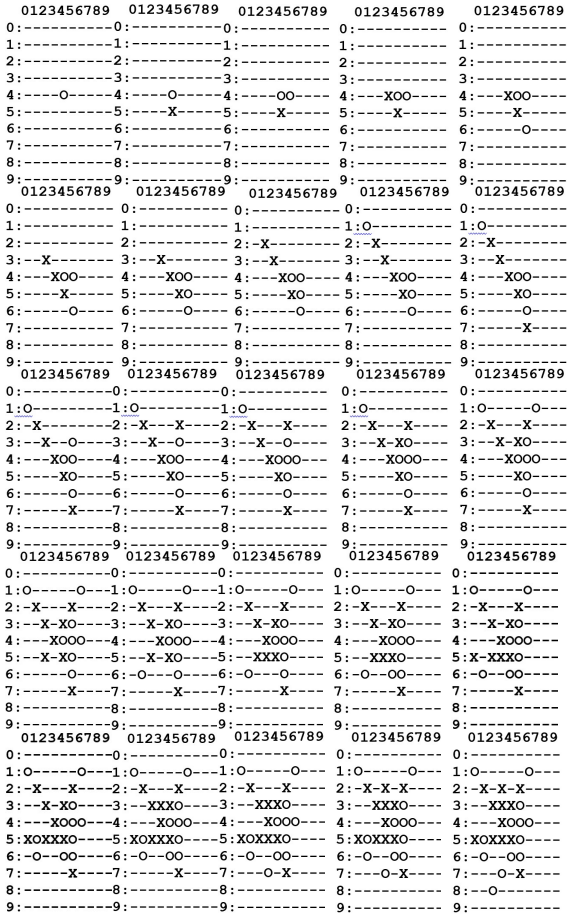


Fig. 2. An instance of 10−10−5 game played by AI (X) against human (O)

We implemented MCTS in Python language. No heuristic rule of the game was implemented in order to evaluate the performance of the pure MCTS and to identify the optimal hyper−parameters of the pure MCTS. The source code is available at https://github.com/isoh24/Omok.

V. EXPERIMENTS

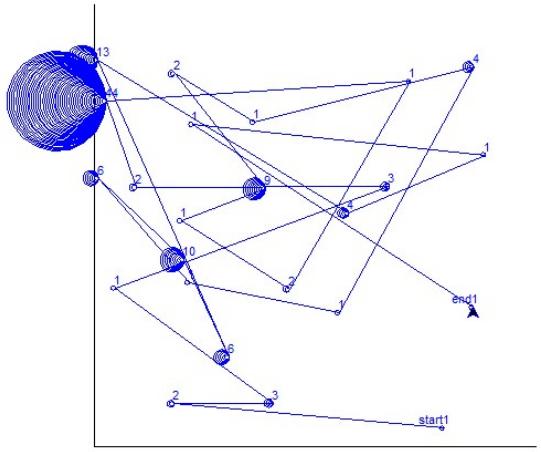
Our experiment was performed on 15*15 board, so 15−15−5 game was used as the testbed problem. The three hyper−parameters in Table 1 are target for the optimization. They are described in Section 2.2 in detail. Table 1 presents the range of values for each hyper−parameter. For λ in Eq.1, it is usual to use $\sqrt{2}$, so we think the range [0.25,4.0] is reasonable. Regarding the second hyper−parameter, our preliminary experiment revealed that two options, *w/v* and *v*, coincide with a high rate. However they chose different child sometimes, so we regarded it as an important hyper−parameter to be optimized simultaneously with other ones. When the computation time doesn't matter, the third hyper−parameter can be excluded in the optimization since a larger value tends to give a higher accuracy. However in many platforms such as desktop computer without GPU or smartphone app, computation time allowed to AI player is limited in playing the game against a human. So we add the number of playouts as third hyper−parameter. Note that the operation of generating playout consumes most of time in running the MCTS. The experiment has been done on a PC with Intel Core i7−8565U CPU (1.80GHz) and 8GB main memory without any parallel programming.

Table 1. Three hyper−parameters for the optimization

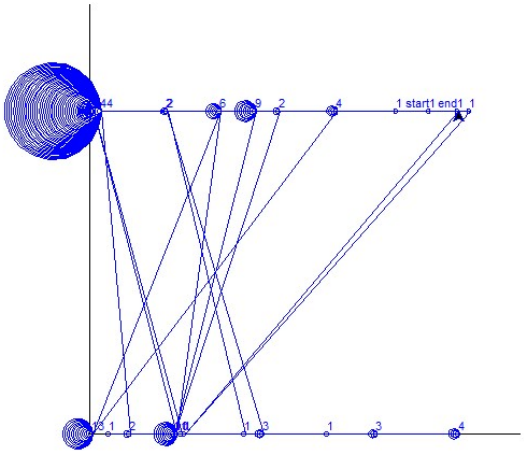
Hyper-parameter (type)	role	range
λ in Eq.1 (floating-point)	Control the trade-off between exploration and exploitation	[0.25,4.0]
b, criterion for choosing optimal child (binary)	Policy using the winning rate (*w/v*) or number of visit (*v*)	0(*w/v*) or 1(*v*)
n, number of playouts (integer)	Controlling the accuracy of deciding the winner	[3000,10000]

1. Visualization of winner path

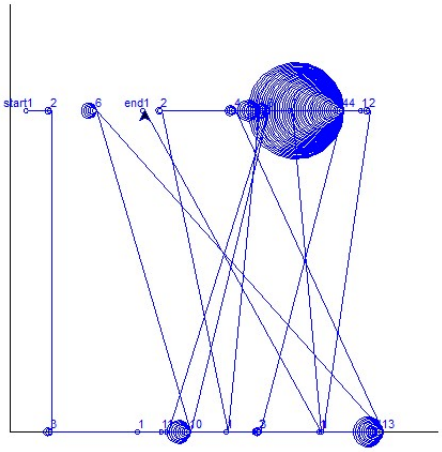
The information visualization is very important in analyzing how the algorithm works[14,15]. This section describes a visualization technique for displaying how our algorithm works. We allowed the loop of line 3~7 in Algorithm 1 to iterate 120 games and collected the winner path data. Since one 3D volume illustration makes a mess, Fig.3 presents three combinations of three hyper−parameters in Table 1. The winner path starts at the point near the lower right corner denoted by 'start' and ends at the point marked by 'end'. The number beside the point and the circle size represent the number of consecutive wins of the point against other hyper−parameter values. The point with the highest winning number is (0.3569,1,9239). It won 44 consecutive games against other hyper−parameter values.



(a) hyper-parameters
(λ horizontal axis and n vertical axis)



(b) hyper-parameters
(λ horizontal axis and b vertical axis)



(c) hyper-parameters
(n horizontal axis and b vertical axis)

Fig.3 A winner path generated by Algorithm 1

Table 2 shows the top-4 winners in the winner path with the values of three hyper-parameters and survival length. The first winner survives 44 games that is believed to be very competitive. The fourth winner survived 9 games which is much lower than the first winner, but are worthy of getting attention as a competitive winner.

Table 2. Top-4 winners in the winner path of Fig.3

Rank	Winner (λ, b, n)	Survival length (number of winning)
1	(0.3569,1,9239)	44
2	(0.2801,0,9973)	13
3	(1.0632,0,6372)	10
4	(1.7969,1,7654)	9

We can observe some patterns in Figure 3(a). The strong winners belonging to the top-4 are located at the upper half region in terms of n, meaning that the higher number of playouts encourages a better performance. The strong winners are located at the left half region in terms of λ, meaning that Algorithm 1 prefers the exploitation. Fig. 3(b) and Fig.3(c) shows that the more competitive winners are located on b=1. Algorithm 1 prefers choosing optimal child with the policy using the number of visit.

2. League of top-4 winners

The line 9 of Algorithm 1 was run for the top-4 winners in Table 2. In the league, each pair of 4 winners plays 20 games. Since the player taking the first move is advantageous in $m*n*k$ game, each of two winners take the first mover for the half of games for the fairness.

Table 3 shows the 4*4 table A on which a_{ij} means the number of winning of the winner i against the winner j. The a_{ij} consists of two numbers f/g where f represent the number of winning when i is the first mover and g the number of winning when j is the first mover. For example, a_{12} is 9/5 that means the winner 1 won 9 games against the winner 2 when the winner 1 is the first mover. The last column has the horizontal sum. The winner 1 has the maximum value and it is regarded as the final winner of 4 winners. Its hyper-parameter value in Table 2, (0.3569,1,9239) is the final output of the algorithm.

Table 3. Winning rate of top-4

	1	2	3	4	sum
1	-	9/5	8/7	10/7	27/19
2	5/1	-	8/5	10/7	23/13
3	3/2	5/2	-	8/3	16/7
4	3/0	3/0	7/2	-	13/2

VI. Conclusions

This paper proposed a hyper-parameter optimization method for MCTS that uses the self-play. By analyzing the winner path, the optimal hyper-parameter is identified. Using 15-15-5 game as a testbed problem, effectiveness of the proposed method was shown. One of the future works is to generate lots of winner paths and to find out some patterns in them. This future work will give a better understanding of the proposed method and eventually leads to a theoretical justification. Another future work is to apply the proposed method to other games.

REFERENCES

[1] GyuHyeok Choi and Mijin Kim, "Analysis of Players' Eye-Movement Patterns by Playing Experience in FPS Game," *Smart Media Journal*, vol. 5, no. 2, pp.33-41, 2016

[2] Walter, J.C. and Barkema, G.T., "An introduction to Monte Carlo methods," *Physica A:Statistical Mechanics and its Applications*, vol. 418, pp. 78-87, January 2015

[3] Browne, c., et al., "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4 Issue: 1, pp. 1-43, March 2012

[4] Fu, M.C., "Monte Carlo tree search: a tutorial," *Proceedings of 2018 Winter Simulation Conference*, pp. 222-236, 2018

[5] Brügmann, B., "Monte Carlo Go, Technical report," *Department of Physics, Syracuse University*, 1993

[6] Coulom, R., "Efficient selectivity and backup operators in Monte-Carlo tree search," *5th International Conference on Computers and Games*, pp. 72-83, 2006

[7] Silver, D., et al., "Mastering the game of Go with deep neural networks and tree search," *Nature 529*, pp. 484-489, January 2016

[8] Silver, D., et al., "Mastering the game of Go without human knowledge," *Nature 550*, pp. 354-359, 2017

[9] Poole, D.L. and Mackworth, A.K., "Artificial Intelligence: Foundations of Computational Agents," *Cambridge University Press*, 2017

[10] Goodfellow, I., Bengio, Y. and Courville, A., "Deep Learning", *The MIT Press*, 2016

[11] Bergstra, J.S., Bardenet, R., "Bengio, Y. and Kegl, B., Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, pp. 2546-2554, 2011

[12] Wang, H, Emmerich, M., Preuss and M., Plaat, A., "Analysis of hyper-parameter for small games: iterations or epochs in self-play?," *arXiv:2003.05988v1*, 2020

[13] Ruijl, B., Vermaseren, J., Plaat, A. and Herik, J., "Combining simulated annealing and Monte Carlo Tree Search for expression simplification," *Proceedings of the 6th International Conference on Agents and Artificial Intelligence*, pp. 724–731, 2014

[14] Woo-Jin Joe, Hyo-Jeong Shin and Hyong-Shik Kim, "A log visualization method for network security monitoring," *Smart Media Journal*, vol. 7, no. 4, pp. 70-78, 2018

[15] Dasom Seo, KangHan Oh, Il-Seok Oh and Tae-Woong Yoo, "Superpixel Exclusion-Inclusion Multiscale Approach for Explanations of Deep Learning," *Smart Media Journal*, vol. 8, no. 2, pp. 39-45, 2019

[16] Li, L., et al., "Hyperband: a novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1-52, January 2017
[17] Rakotoarison, H., Schoenauer, M. and Sebag, M., "Automated machine learning with Monte Carlo tree search," *IJCAI-19 28th International Joint Conference on Artificial Intelligence*, pp. 3296-3303, Macau, China, Aug. 2019

Authors

Jin-Seon Lee

She received her B.S, M.S and Ph.D. from Chonbuk National University, South Korea. She is currently a Professor at the Department of Information and Security, Woosuk University, Jeonbuk, South Korea. Her research interests include pattern recognition and machine learning.

IL-Seok Oh

He received the B.S. degree in computer engineering from Seoul National University, South Korea, in 1984, and the Ph.D. degree in computer science from KAIST, South Korea, in 1992. He is currently a Professor with the Division of Computer Science and Engineering, Jeonbuk National University, Jeonju, South Korea. He was a Visiting Scientist with CENPARMI, Concordia University, Canada, and UCI, USA. He is the author of the books Pattern Recognition, Computer Vision, and Machine Learning (Korean Language). His research interests include computer vision, pattern recognition, and machine learning