ICIC Express Letters
Volume **18**, Number **3**, March **2024**

 ISSN 1881-803X
pp. **219–227**

USING PARTICLE REPRESENTATION OF BELIEFS IN AN ALPHAZERO-BASED REINFORCEMENT LEARNING ALGORITHM FOR PARTIALLY OBSERVABLE ENVIRONMENTS

Hideaki Itoh[1,*], Yuta Kihara[2], Hisao Fukumoto[1] and Hiroshi Wakuya[1]

[1]Department of Electrical and Electronic Engineering
Faculty of Science and Engineering
[2]Electrical and Electronic Engineering Course
Graduate School of Science and Engineering
Saga University
1 Honjo-machi, Saga 840-8502, Japan
kihara@ace.ec.saga-u.ac.jp; { tokusima; wakuya }@cc.saga-u.ac.jp
*Corresponding author: hideaki@cc.saga-u.ac.jp

Received May 2023; accepted August 2023

Abstract. *The AlphaZero algorithm is a general reinforcement learning algorithm that defeated world-champion level programs in chess, shogi, and Go without prior knowledge. However, the algorithm can only be used for fully observable games, in which all the information regarding the current state of the game is visible to the players. It has not been fully studied how the AlphaZero-type algorithm can be used in partially observable games. Thus, in this study, we modify the AlphaZero algorithm to make it applicable to partially observable environments. We use beliefs, which are posterior probability distributions regarding the hidden state, to enable decision making and learning under partial observability, and examine the use of particles to represent the beliefs. We modify the Monte-Carlo Tree Search (MCTS) and the deep neural network in the AlphaZero algorithm to enable the use of beliefs represented by particles. The performance of the modified algorithm is tested using a popular card game, blackjack. Through numerical experiments, we show that the use of particle representation of beliefs is a valid approach to make the AlphaZero algorithm applicable to the partially observable environment.*
Keywords: AlphaZero, Partially observable environment, MCTS, Deep neural network, Belief, Particle

1. **Introduction.** Artificial intelligence for games, or game AI for short, is an important branch of artificial intelligence. Progresses in game AI benefit not only mastering games but also solving various kinds of real-world problems.

In some games, there have already been developed strong game AIs that can achieve human-level or super-human performances (e.g., [1, 2, 3]). However, each game AI has typically been developed to master the target game only. Such AIs are not easily applicable to other games. General-purpose algorithms which can master a wide range of games had not been extensively studied until recently.

The AlphaZero algorithm [4] has made an important achievement on this regard. It is a general-purpose reinforcement learning algorithm built upon three techniques: self-play, MCTS, and deep neural network. It has defeated world-champion level programs in chess, shogi, and Go. The only required knowledge for the algorithm to master each of the three games was basically the rules of the target game. Therefore, the algorithm can also be applied to many other games.

However, the AlphaZero algorithm can only be used for mastering fully observable games, in which all the information regarding the current state of the game is visible to

DOI: 10.24507/icicel.18.03.219

the players. It has not been well studied how the AlphaZero-type algorithm can be used in partially observable environments, in which some pieces of information regarding the current state are hidden.

There have already been some studies that have modified AlphaZero for use in partial observation environments [5, 6, 7]. There have also been many studies that have used deep neural networks for reinforcement learning in partial observation environments [8, 9, 10, 11, 12]. To select optimal actions in partially observable environments, a sufficient statistic for the hidden part of the state is necessary [13], and the previous studies have used neural networks to create it. However, this approach requires training of a neural network, and it is difficult for humans to understand the neural representation of the sufficient statistic acquired through the training.

Thus, in this paper, we examine an alternative approach, where we use *beliefs represented by particles* as a sufficient statistic. The belief is the posterior probability distribution of the hidden part of the state given available information [13], and the particles are a set of samples drawn from the probability distribution [14]. The use of them is one of the common approaches in decision making in partially observable environments [15, 16, 17], and the belief represented by the particles is easy to understand for humans unlike the neural representation. Their use has already been explored in some studies on MCTS [18, 19, 20, 21]. However, their use has not been examined in combination with the MCTS and deep neural network used in the AlphaZero algorithm.

In this paper, we modify the AlphaZero algorithm so that it can use beliefs represented by particles. We test the performance of the modified algorithm using a popular card game, blackjack. Using numerical experiments, we show the validity of the modified algorithm.

The rest of this paper is organized as follows. In Section 2, we explain the original AlphaZero algorithm. In Section 3, we describe the modification that we made to the original algorithm. In Section 4, we provide experimental results. Section 5 concludes this paper.

2. AlphaZero.

Before providing the modified AlphaZero algorithm in the next section, we review the original AlphaZero algorithm here.

2.1. Self-play.

Self-play means that an AI algorithm plays the target game against the same AI algorithm. This technique is useful in games in which two (or more) players play against each other. It enables the AI algorithm to play a large amount of games by itself, collecting experiences for learning.

The learning takes place as follows. First, the AI algorithm plays the target game N times against itself and collects experiences. At each time step in each game, the algorithm determines the next action to take according to an action probability distribution π. Here, π is a vector whose a-th component, denoted by π_a, is the probability of taking action a. Namely,

$$\pi_a = \Pr(a|s), \tag{1}$$

where s is the current state of the game and a is the next action. The action probability distribution π is obtained by the MCTS method described in Section 2.2. After each game is finished, the game's result z is determined as, e.g., $z = 1$ if the player has won, and $z = -1$ if the player has lost. The action probability distributions π and the results z occurred in the N games played are recorded for being used later to train the neural network (Section 2.3).

After N games are played, the neural network is trained with the recorded π and z. Playing N games for collecting experiences and then training the neural network with the collected experiences are collectively called an *iteration*. Typically, many iterations are performed to master the target game.

2.2. **MCTS.** At each time step during each game, an MCTS is performed to obtain the action probability distribution π in Equation (1). In an MCTS, simulated games are played N_s times, and a search tree is gradually constructed (Figure 1). Each node of the search tree corresponds to a state of the game. The root node of the search tree corresponds to the current state s_0. For example, let us suppose that the target game is blackjack and the current state s_0 is that the dealer has two cards, 2 and 3, and the player also has two cards, 2 and 3. This state is drawn at the root node of the search tree in Figure 1(a). (Details of the blackjack game are explained in Section 4.1. Although one of the dealer's cards is hidden in real blackjack games, let us assume, for the time being, that it is visible.) Beginning from s_0, simulated games are played N_s times.

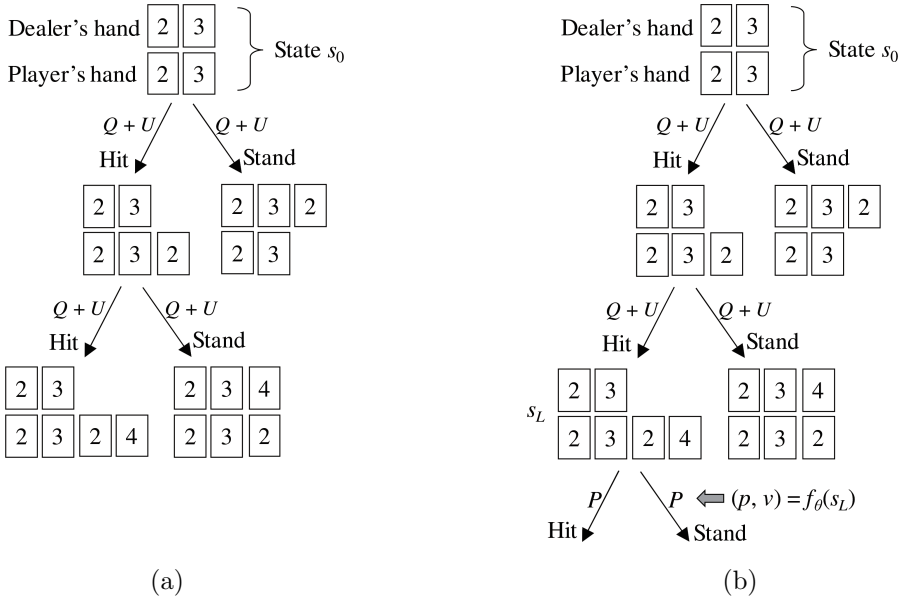

FIGURE 1. Examples of the search tree in MCTS: (a) Search tree for blackjack; (b) expansion of the search tree

In each state s of the simulated games, action a that maximizes

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

is selected as the player's next action, where $Q(s,a)$ is the expected value of game result z after action a is selected in state s, and

$$U(s,a) = cP(s,a)\frac{\sqrt{\sum_{a'} N(s,a')}}{1+N(s,a)} \tag{3}$$

is a bias towards taking actions that have not been selected many times yet. In Equation (3), $P(s,a)$ is the prior probability distribution of the next action a, which is provided by the neural network described in Section 2.3. $N(s,a)$ is the number of times that action a has been selected in s during the MCTS simulations. c is a hyper parameter that balances the two terms, $Q(s,a)$ and $U(s,a)$. In Figure 1(a), for example, there are two actions, "Hit" and "Stand" in each state. Each action in each state has its own $Q(s,a)$ and $U(s,a)$, and the action with a greater value of $Q(s,a)+U(s,a)$ is selected. Each simulation ends at a leaf node of the search tree.

To calculate $Q(s,a)$ and $U(s,a)$, the search tree is gradually constructed during the N_s simulated games (Figure 1(b)). Initially, the search tree consists only of the root node. When a leaf node is reached during a simulation, it is expanded with initial values of

$$N(s_L,a) = 0, \tag{4}$$

$$W(s_L,a) = 0, \tag{5}$$

$$Q(s_L, a) = 0, \tag{6}$$

$$P(s_L, a) = p_a \tag{7}$$

for every action a. Here, s_L is the state corresponding to the leaf node, $W(s_L, a)$ is the sum of predicted game results, and p_a is the action probability obtained by the neural network in Section 2.3. In Figure 1(b), for example, let the bottom-left state (i.e., the state reached after two "Hit" actions from s_0) be the leaf node reached. It is expanded for each action a with the initial values of $P(s_L, a)$ (and also of $N(s_L, a)$, $W(s_L, a)$, and $Q(s_L, a)$, which are omitted in the figure), using the output $(p, v) = f_\theta(s_L)$ of the neural network described in Section 2.3.

After expanding the leaf node, the simulated game is terminated, and the values of each node up to the root node are updated as follows:

$$N(s_t, a_t) = N(s_t, a_t) + 1, \tag{8}$$

$$W(s_t, a_t) = W(s_t, a_t) + v, \tag{9}$$

$$Q(s_t, a_t) = W(s_t, a_t)/N(s_t, a_t), \tag{10}$$

where s_t and a_t are the state and action, respectively, that occurred at the node being updated, and v is the predicted game result obtained by the neural network in Section 2.3. After N_s simulated games are played, π is calculated as

$$\pi_a = \frac{N(s_0, a)^{1/\tau}}{\sum_{a'} N(s_0, a')^{1/\tau}} \tag{11}$$

for each a, where $N(s_0, a)$ is the number that action a was selected in the root node s_0 during the simulated games, and τ is a parameter that controls how directly the differences in the action counts $N(s_0, a)$ become the differences in the action probabilities π_a.

2.3. **Deep neural network.** A deep neural network is trained so that it outputs predictions for the action probability distribution π and game result z given the current state s. This is denoted by

$$(p, v) = f_\theta(s), \tag{12}$$

where p and v are the neural network's prediction for π and z, respectively, and θ denotes the parameters of the neural network.

Training of the neural network is performed to minimize the following loss function:

$$l = (z - v)^2 - \pi^{\mathrm{T}} \log p + k||\theta||^2, \tag{13}$$

where p and v are the outputs of the neural network, π and z are the corresponding teacher signals collected during the self-plays described in Section 2.1, and k is a constant. With the neural network trained, p_a in Equation (7) is obtained as the a-th component of the neural network's output p.

3. **Modified AlphaZero.** In partially observable games, the states are not fully observable to the agent. For example, in blackjack, one of the dealer's cards is hidden (Figure 2). In such a case, if there is some information available on the hidden part of the state, a sufficient statistic for the hidden part of the state is necessary for the agent to correctly predict the future. As mentioned in Section 1, we use a *belief*, which is the posterior probability distribution of the hidden part of the state given the available information, and we let the belief be represented by *particles* [14], which are samples drawn from the probability distribution.

When one of the dealer's cards is hidden (Figure 2), an example of the belief b is

$$b = \{1, 2, 3, 3, 3, 6, 8, 8, 9, 10\}, \tag{14}$$

where each member of the set is a sample drawn from the posterior probability distribution of the hidden card.

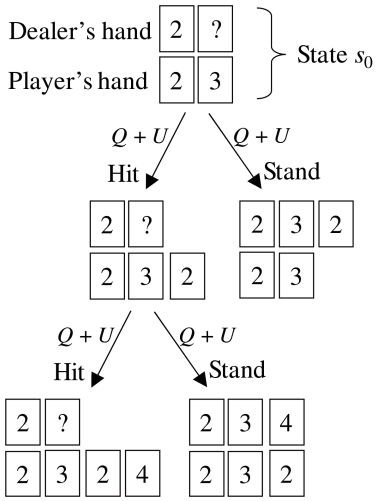

FIGURE 2. Example of an MCTS search tree in which a card of the dealer is hidden. In blackjack, the hidden card is revealed after the player selects the "Stand" action.

We modify the MCTS algorithm so that the hidden part of the state is replaced with belief b. For example, the root state s_0 in Figure 2 is represented as "Dealer's visible card is 2, Player's cards are 2 and 3, and the belief b of the dealer's hidden card is $\{1, 2, 3, 3, 3, 6, 8, 8, 9, 10\}$". All the states s, s_0, s_L, and s_t in Section 2 are replaced with those in which the hidden part of the state is replaced with belief b.

The inputs to the neural network are also modified accordingly. For example, in our experiments of blackjack (Section 4), the input to the neural network is a 3×10 matrix (Figure 3). The top two rows of the matrix contain the dealer's hand and the player's hand. The number of columns of the matrix is set to 10 because the number of the player's cards can be 10 at most in blackjack. The bottom row of the matrix contains the belief (Equation (14)) of the dealer's hidden card.

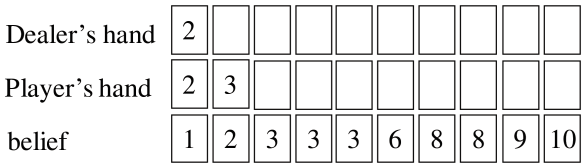

FIGURE 3. Example of the input to the neural network, in the case of the root node in Figure 2

4. **Experiments.** To evaluate the performance of the proposed algorithm, we let the original and modified AlphaZero algorithms learn to play the blackjack game.

4.1. **Rules of blackjack.** The rules of the blackjack that we used for our experiments are as follows. A blackjack game is played by a player and a dealer. A deck of cards (i.e., the standard 52-card pack) is used. The 52 cards are shuffled before each game. When a game starts, the player is dealt two cards. Both of the two cards are face up. Then the dealer is dealt two cards. One of them is face up and the other is face down. Next, the player selects "Hit" (i.e., to receive one more card) or "Stand" (i.e., to stop receiving cards). If the player hits, a card is added to the player's hand. If the player's hand exceeds 21, the player loses; otherwise, the player can select "Hit" or "Stand" again. This is repeated until the player stands. If the player stands, the dealer's face-down card becomes face up. While the dealer's hand does not exceed 17, the dealer continues taking one more card. If the dealer's hand exceeded 21, the player wins. When the dealer's hand exceeds

17 without exceeding 21, the winner of the game is judged as follows: if the hands of the player and the dealer are equally close to 21, then the result of the game is a draw. Otherwise, the player or the dealer whose hand is closer to 21 wins.

A hand of the player as well as the dealer is evaluated as the sum of the values of all the cards in the hand. Each ace card is valued as 1 or 11, which is chosen so that the hand becomes as close to 21 as possible without exceeding 21. Face cards (i.e., Jacks, Queens, and Kings) are valued as 10. Other cards (i.e., cards 2 through 10) are valued as is.

In commonly used blackjack rules, if the player's hand is 21 at the beginning of a game (which is called *natural 21*), the player does nothing, and the result (win or draw) is determined depending on the dealer's hand (i.e., the player wins unless the dealer' hand is also 21). However, in our experiments, when the natural 21 happened, the game is just ignored and the next game is started. This is because there is nothing for the player to learn from those games. The winning rates in the result section (Section 4.3) do not include such winnings.

4.2. **Experimental setups.**

4.2.1. *Self-play.* As explained in Section 4.1, two persons, a player and a dealer, participate in a blackjack game. However, since the dealer selects actions with a fixed rule, learning is unnecessary for the dealer. Thus, in our experiments, we do not conduct the self-play in a strict sense. Nonetheless, the learning of the player takes place from scratch, without any prior knowledge on how to win the game except for the game rules.

4.2.2. *Belief and MCTS.* To apply the modified AlphaZero (Section 3) to blackjack, we need a belief regarding the dealer's hidden card. In real blackjack games, the player can guess what the hidden card is, based on various clues. For example, in a method called *card counting*, the player counts certain values of all the visible cards that have already appeared during the current and previous games in which the same card decks have been used. In the present study, however, we do not consider details on how to make such a guess, and we assume that the player's belief of the hidden card is expressed as a probability distribution in which the true hidden card has a certain probability value of x and the other possible cards are equally likely. The value of x will be changed in our experiments so that we can study the performance of the modified AlphaZero when different amount of information is given to the agent.

More specifically, we assume that the belief is expressed as

$$\bar{b}(i) = x\delta(i - i^*) + (1 - x)U(i) \tag{15}$$

for each $i \in \{1, 2, \ldots, 52\}$, where i $(= 1, 2, \ldots, 52)$ is a unique index assigned to each of the 52 cards, $\bar{b}(i)$ is the posterior probability that the index of the hidden card is i, $\delta(i - i^*)$ is the Kronecker delta function which equals 1 if $i = i^*$ and 0 otherwise, i^* is the index of the true hidden card, and $U(i)$ is the uniform distribution over all the possible cards. The bar in $\bar{b}$ is added to avoid being confused with the belief b represented by particles as in Equation (14). We note that, in Equation (15), a larger value of x means that the agent is making a more accurate guess regarding the index of the hidden card.

At each time step of each game, the belief b (Equation (14)) is constructed by drawing ten samples from the distribution $\bar{b}$ in Equation (15). The MCTS is conducted using b, as mentioned in Section 3. Note that neither $\bar{b}$ nor b allows the agent to know the exact index of the hidden card.

4.2.3. *Neural network.* The inputs to the neural network are also modified to include the belief b, as mentioned in Section 3 (e.g., Figure 3). We used a 6-layered neural network consisting of an input layer, a convolution layer, three densely connected layers, and an output layer. We normalized the inputs to the neural network to values between 0 and 1, by dividing the raw values by ten. When testing the original AlphaZero, we used the

same input matrix as the one shown in Figure 3, except that the bottom row containing the belief information is deleted.

4.2.4. *Conditions.* Experiments were conducted under the following three conditions:

(1) The original AlphaZero learns the blackjack in which one of the dealer's cards is hidden;
(2) The modified AlphaZero learns the blackjack in which one of the dealer's cards is hidden but belief b is given to the player;
(3) The original AlphaZero learns the blackjack in which the dealer's cards are all open.

Condition (1) is for studying the performance of the original AlphaZero in the partially observable game. Condition (2) is for studying the performance of the modified AlphaZero. If the modified AlphaZero can properly use the partial information b about the hidden card, then the winning rate of (2) is expected to be higher than that of (1).

In condition (3), the dealer's cards are all open, unlike in normal blackjack games. The purpose of studying this condition is to see the upper limit of the winning rate. The original AlphaZero can be used in this condition because there is no hidden card.

In the experiment, we set $N = 1000$, $N_s = 3$, and $c = \tau = k = 1$. The game result z was set to 1 for a win of the player, 0 for a draw, and -1 for a loss. The length of b was 10. Each winning rate was calculated from 1000 games played solely for evaluation (i.e., those 1000 games were not used for learning). We used a 4.2 GHz Intel Core i7 7700K CPU and a GeForce GTX1080 GPU.

Each result shown in Figures 4-6 in the next section is the average over 12 experiments with different random seeds. The error bars indicate the standard error.

4.3. **Results.** Figure 4 shows the learning curves, i.e., changes in the winning rate during the iterations of learning, in the three conditions mentioned in Section 4.2.4. We set $x = 0.5$ in this experiment. The winning rates in the figure are lower than 50%, but this is because winnings by natural 21 (Section 4.1) and draws are not included. It can be seen in the figure that the result of condition (2), i.e., the modified AlphaZero with belief b, was better than that of condition (1), i.e., the original AlphaZero. Therefore, it was confirmed that the modified algorithm can properly use the partial information regarding the hidden card.

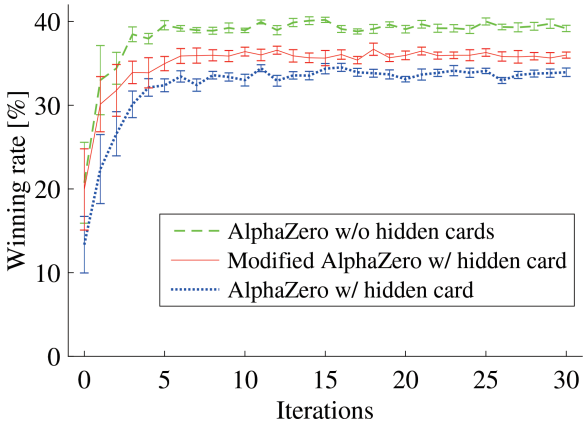

FIGURE 4. Learning curves

Figure 5 shows the time required for the learning in conditions (1) and (2). The results of condition (3) are omitted because they were almost identical to those of condition (1). It is shown that the total time was not very different between conditions (1) and (2). In each condition, the time difference between the total time and the time spent for collecting experiences are the time spent for training the deep neural network. The time spent for training the neural network was a little longer in condition (2) than in condition (1). This

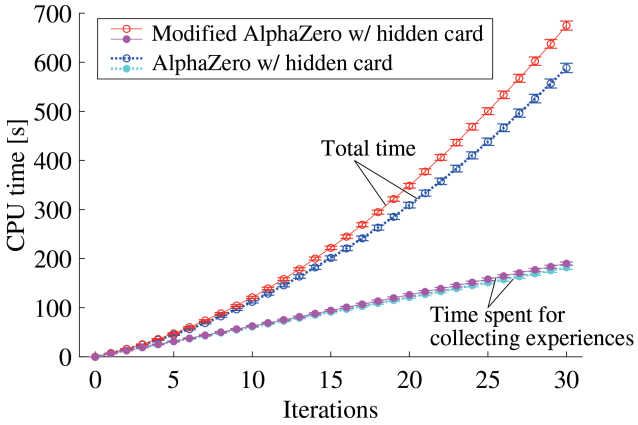

FIGURE 5. Cumulative CPU time required for the iterations in conditions (1) and (2)

is because the size of the input matrix for the neural network is larger in condition (2), since the bottom row is added to contain the belief information (Figure 3).

The winning rates for various values of x are shown in Figure 6. For comparison, the results of the original AlphaZero with and without a hidden card are also shown in the figure. The larger the value of x was, the higher was the winning rate. This result is reasonable because a larger value of x means that the agent's belief is more informative (i.e., the posterior probability is more concentrated on the index of the true hidden card). When x was 0.1, the winning rate was at the same level as that of the original AlphaZero. This means that, in such a case, the modified algorithm could not make use of the belief to improve the winning rate.

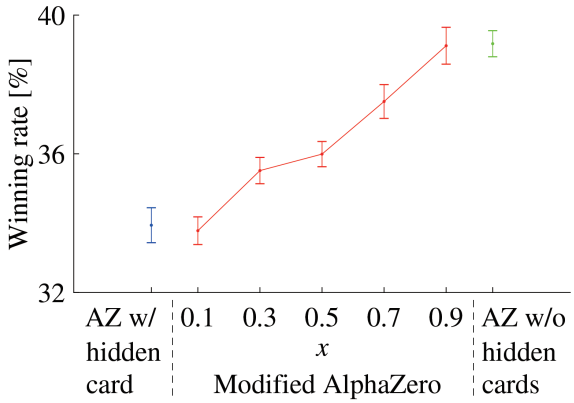

FIGURE 6. Comparison of the winning rates. Each point is the winning rate after 30 iterations.

5. **Conclusion.** In this study, we have used the particle representation of beliefs in order to modify the AlphaZero algorithm. Numerical experiments using the blackjack have shown that the use of the particle belief representation is a valid approach to making the AlphaZero algorithm applicable to the partially observable environment.

An important future study is to investigate how general our algorithm is by applying the algorithm to other partially observable games. It is also important to apply the algorithm to real-world problems other than games.

Another important future issue is to modify the structure of the neural network. In the present study, we have used an ordinary neural network consisting of a convolution layer and densely connected layers. However, the belief b given to the neural network as its input has a unique structure that it consists of many samples. There can be neural

network structures that exploit this input structure for more efficient learning and/or more accurate prediction.

Acknowledgment. This work was supported in part by JSPS KAKENHI Grant Number JP 22K12195.

REFERENCES

[1] M. Campbell, A. J. Hoane Jr. and F.-H. Hsu, Deep blue, *Artificial Intelligence*, vol.134, nos.1-2, pp.57-83, 2002.
[2] T. Takizawa, Contemporary computer shogi (May 2013), *Proc. of Game Informatics*, vol.30-1, Ishikawa, Japan, 2013.
[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., Mastering the game of Go with deep neural networks and tree search, *Nature*, vol.529, no.7587, pp.484-489, 2016.
[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel 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.
[5] T. Kimura, K. Sakamoto and T. Sogabe, Development of AlphaZero-based reinforcement learning algorithm for solving partially observable Markov decision process (POMDP) problem, *Bulletin of Networking, Computing, Systems, and Software*, vol.9, no.1, pp.69-73, 2020.
[6] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert and D. Silver, Planning in stochastic environments with a learned model, *Proc. of International Conference on Learning Representations*, Online, 2022.
[7] Q. Wang, Y. He and C. Tang, Mastering construction heuristics with self-play deep reinforcement learning, *Neural Computing and Applications*, vol.35, no.6, pp.4723-4738, 2023.
[8] M. Hausknecht and P. Stone, Deep recurrent Q-learning for partially observable MDPs, *Proc. of 2015 AAAI Fall Symposium Series*, Arlington, Virginia, 2015.
[9] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev et al., Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, vol.575, no.7782, pp.350-354, 2019.
[10] K. Yamashita and T. Hamagami, Reinforcement learning for POMDP environments using state representation with reservoir computing, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol.26, no.4, pp.562-569, 2022.
[11] T. Sogabe, T. Kimura, C.-C. Chen, K. Shiba, N. Kasahara, M. Sogabe and K. Sakamoto, Model-free deep recurrent Q-network reinforcement learning for quantum circuit architectures design, *Quantum Reports*, vol.4, no.4, pp.380-389, 2022.
[12] S. Xie, Z. Zhang, H. Yu and X. Luo, Recurrent prediction model for partially observable MDPs, *Information Sciences*, vol.620, pp.125-141, 2023.
[13] M. Hauskrecht, Value-function approximations for partially observable Markov decision processes, *Journal of Artificial Intelligence Research*, vol.13, pp.33-94, 2000.
[14] A. Doucet and A. M. Johansen, A tutorial on particle filtering and smoothing: Fifteen years later, *Oxford Handbook of Nonlinear Filtering*, pp.656-704, 2011.
[15] W. Jiang, Y. Lyu, Y. Li, Y. Guo and W. Zhang, UAV path planning and collision avoidance in 3D environments based on POMDP and improved grey wolf optimizer, *Aerospace Science and Technology*, vol.121, 107314, 2022.
[16] Y. Wang, M. Zechner, J. M. Mern, M. J. Kochenderfer and J. K. Caers, A sequential decision-making framework with uncertainty quantification for groundwater management, *Advances in Water Resources*, vol.166, 104266, 2022.
[17] L. Burks, H. M. Ray, J. McGinley, S. Vunnam and N. Ahmed, HARPS: An online POMDP framework for human-assisted robotic planning and sensing, *IEEE Transactions on Robotics*, pp.1-19, 2023.
[18] D. Silver and J. Veness, Monte-Carlo planning in large POMDPs, *Advances in Neural Information Processing Systems*, pp.2164-2172, 2010.
[19] H. Itoh, H. Nakano, R. Tokushima, H. Fukumoto and H. Wakuya, A partially observable Markov decision process-based blackboard architecture for cognitive agents in partially observable environments, *IEEE Transactions on Cognitive and Developmental Systems*, vol.14, no.1, pp.189-204, 2022.
[20] S. Amini, M. Palhang and N. Mozayani, POMCP-based decentralized spatial task allocation algorithms for partially observable environments, *Applied Intelligence*, vol.53, pp.12613-12631, 2023.
[21] A. Tahir, B. Alt, A. Rizk and H. Koeppl, Load balancing in compute clusters with delayed feedback, *IEEE Transactions on Computers*, vol.72, no.6, pp.1610-1622, 2023.