Journal of Information Processing Vol.27 340–347 (Apr. 2019)

[DOI: 10.2197/ipsjjip.27.340]

Regular Paper

Playing Game 2048 with Deep Convolutional Neural Networks Trained by Supervised Learning

Naoki Kondo[1,a)] Kiminori Matsuzaki[2,b)]

Received: July 30, 2018, Accepted: January 15, 2019

Abstract: Game 2048 is a stochastic single-player game and development of strong computer players for Game 2048 has been based on N-tuple networks trained by reinforcement learning. Some computer players were developed with (convolutional) neural networks, but their performance was poor. In this study, we develop computer players for Game 2048 based on deep convolutional neural networks (DCNNs). We increment the number of convolution layers from two to nine, while keeping the number of weights almost the same. We train the DCNNs by applying supervised learning with a large number of play records from existing strong computer players. The best average score achieved is 93,830 with five convolution layers, and the best maximum score achieved is 401,912 with seven convolution layers. These results are better than existing neural-network-based players, while our DCNNs have less weights.

Keywords: Game 2048, neural network, supervised learning

1. Introduction

Neural networks (NN) are now widely used in development of computer game players. Among these, deep convolutional neural networks (DCNN) have been studied actively in recent years and played an important role in the development of master-level computer players, for example, for Go (AlphaGo [22] and AlphaGo Zero [24]), Chess (Giraffe [13] and DeepChess [5]), Shogi (AlphaZero [23]), Poker (Poker-CNN [31] and DeepStack [19]), and Atari games [18].

The target of this study is Game "2048" [4], a stochastic single-player game. Game 2048 is a slide-and-merge game and its "easy to learn but hard to master" characteristics have attracted quite a few people. According to its author, during the first three weeks after its release, people spent a total time of over 3000 years on playing the game.

Several computer players have been developed for Game 2048. Among them, the most successful approach is to use N-tuple networks (NTNs) as evaluation functions and apply a reinforcement learning method to adjust the weights of NTNs. This approach was first introduced to Game 2048 by Szubert and Jaśkowski [25], and several studies were then based on it. The state-of-the-art computer player developed by Jaśkowski [11] combined several techniques to improve NTN-based players, and achieved an average score of 609,104 within a time limit of 1 second per move.

NN-based computer players, however, have not achieved a success yet. The only published work by Guei et al. [9] proposed a player with two convolution layers followed by two full-connect layers, but the average score was about 11,400. The player by tjwei [26] used two convolution layers with a large number of weights trained by supervised learning and achieved an average score 85,351. There are other implementations of NN-based players [1], [6], [21], [27], [28], but the scores of these players were not so good or were not reported.

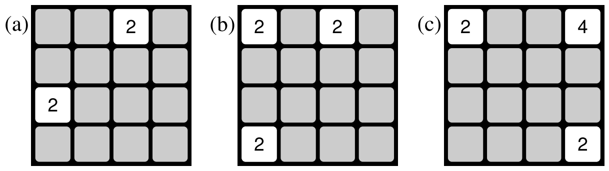
In this paper, we try to improve the performance of DCNN-based players by increasing the number of convolution layers. We designed DCNNs with 2–9 convolution layers and applied supervised learning with play records from existing strong players [15]. As the result, we achieved better results than existing NN-based players. The best player with five convolution layers achieved an average score of 93,830. The player with seven convolution layers achieved maximum score 401,912. These results suggest that DCNNs with 5–7 convolution layers have great potential to develop strong Game 2048 players.

Contributions in this paper are summarized as follows.
- We designed DCNNs with two to nine convolution layers while keeping the total number of weights almost the same (Section 4).
- From the experiment results, we found that we can improve the performance of DCNN-based players by increasing the number of convolution layers from two (Section 5.2).
- The best results by the DCNN with five convolution layers were comparable to NTN-based players with a similar number of weights (Section 5.2). We analyzed the DCNN-player by changing the number of weights and by feeding more training data (Section 5.3).
- We discussed our findings in terms of advantages and disadvantages of DCNN-based players. We also tested supervised learning of value networks (both an NTN and a smaller variant of tjwei's network) with the same set of training data, but we failed to obtain good players (Section 6).

1 Graduate School of Engineering, Kochi University of Technology, Kami, Kochi 782–8502 Japan
2 School of Information, Kochi University of Technology, Kami, Kochi 782–8502 Japan
a) 225119q@gs.kochi-tech.ac.jp
b) matsuzaki.kiminori@kochi-tech.ac.jp

Table 1 Summary of players in terms of number of weights and average score of greedy (1-ply search) play.

	authors	description	number of weights	ave. score
	Szubert and Jaśkowski [25]	17×4-tuples, TD learning	860,625	51,320
	Szubert and Jaśkowski [25]	2×4-tuples & 2×6-tuples, TD learning	22,882,500	99,916
	Wu et al. [29], [32]	4×6-tuples, TD learning, 3 stages	136,687,500	143,958
N-tuple	Oka and Matsuzaki [20]	40×6-tuples, TD learning	671,088,640	210,476
network	Oka and Matsuzaki [20]	10×7-tuples, TD learning	2,684,354,560	234,136
	Matsuzaki [14]	8×6-tuples, TD learning	134,217,728	226,958
	Matsuzaki [14]	8×7-tuples, TD learning	2,147,483,648	255,198
	Matsuzaki [15]	4×6-tuples, backward TC learning, 8 stages	536,870,912	232,262
	Jaśkowski [11]	5×6-tuples, TC learning, 16 stages, redundant encoding, etc.	1,347,551,232	324,710
	This work (comparison)	5×4-tuples, TC learning, 3 stages	983,040	50,120
	Guei et al. [9]	2 convolution (2 × 2), 2 full-connect, TD learning	N/A	≈11,400
	Guei et al. [9]	3 convolution (3 × 3), 2 full-connect, TD learning	N/A	≈ 5,300
	tjwei [26]	2 convolution (2 × 1 & 1 × 2), 1 full-connect, supervised learning	16,949,248	85,351
Neural	tjwei [26]	2 convolution (2 × 1 & 1 × 2), 1 full-connect, reinforcement learning	16,949,248	≈33,000
network	Allik et al. [1]	1 convolution (3 × 3), 20 residual, 2 full-connect, supervised learning	≈2,727,000	≈31,000
	Virdee [27]	2 convolution (2 × 1 & 1 × 2), 2 full-connect, reinforcement learning	1,975,296	≈16,000
	This work	2 convolution (2 × 2), 1 full-connect, supervised learning	817,068	25,669
	This work	5 convolution (2 × 2), 1 full-connect, supervised learning	818,074	93,830



(a) An initial state. Two tiles are placed randomly.
(b) After the first move: *up*. A new 2-tile appears at the lower-left corner.
(c) After the second move: *right*. Two 2-tiles are merged to a 4-tile, and score 4 is given. A new tile appears at the upper-left corner.

Fig. 1 Process of Game 2048.

The rest of the paper is organized as follows. Section 2 briefly introduces the rule of Game 2048. Section 3 reviews existing computer players for Game 2048, categorized in terms of N-tuple networks and neural networks. Section 4 shows the design of our DCNN players and explains how we applied supervised learning. Section 5 reports the experiment results. We discuss the findings in this study in Section 6, and conclude the paper in Section 7.

2. Game 2048

Game 2048 is played on a 4×4 grid. The objective of the original Game 2048 is to reach a 2048 tile by moving and merging the tiles on the board according to the rules below. In an initial state (**Fig. 1**), two tiles are placed randomly with numbers 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$). The player selects a direction (either up, right, down, or left), and then all the tiles will move in the selected direction. When two tiles of the same number collide, they create a tile with the sum value and the player gets the sum as the score. Here, the merges occur from the far side and newly created tiles do not merge again on the same move: move to the right from `222␣`, `␣422` and `2222` results in `␣␣24`, `␣␣44`, and `␣␣44`, respectively. Note that the player cannot select a direction in which no tiles move nor merge. After each move, a new tile appears randomly at an empty cell with number 2 ($p_2 = 0.9$) or 4 ($p_4 = 0.1$). If the player cannot move the tiles, the game ends.

When we reach the first 1024-tile, the score is about 10,000. Similarly, the score is about 21,000 for a 2048-tile, about 46,000 for a 4096-tile, about 100,000 for an 8192-tile, about 220,000 for a 16384-tile, and about 480,000 for a 32768-tile.

3. Current Game 2048 Players

Several computer players have been developed for Game 2048. The most widely used and successful approach is based on N-tuple networks (NTNs) trained by reinforcement learning methods [11], [15], [25], [29]. Neural networks (NN) are also popularly used in the development of Game 2048 computer players [1], [6], [9], [21], [26], [27], [28]. In this section, we review current Game 2048 players for NTN-based and for NN-based players. Other approaches include those based on evolutionary algorithm [2], [3], [7] and combining tree search techniques with human-designed evaluation functions [30].

Table 1 summarizes existing computer players in terms of their features, the number of weights, and the average score with greedy (1-ply search) plays.

3.1 Players based on N-Tuple Networks

The most successful approach to Game 2048 computer players is based on N-tuple networks trained by reinforcement learning methods which was first introduced by Szubert and Jaśkowski [25]. NTNs consist of a set of N-tuples and associated tables of (feature) weights. Given NTNs, we compute the evaluation value of a state simply by looking up weights corresponding to the tiles where the N-tuples cover and computing their sum.

Thanks to the simple design and implementation of the NTNs, we can improve the performance of players by increasing the number of weights. The following are three existing ways in this direction.

(1) *Enlarge the size of tuples*. Szubert and Jaśkowski [25] reported that the computer player performed significantly better by introducing 6-tuples instead of 4-tuples. Some studies used larger 7-tuples [11], [14], [20]. Note that a 4-tuple requires $16^4 = 65{,}536$ weights, a 6-tuple does $16^6 = 16{,}777{,}216$ weights, and a 7-tuple does $16^7 = 268{,}435{,}456$ weights.

(2) *Increase the number of N-tuples*. Though several studies have used four 6-tuples designed by Wu et al. [29], we can use more N-tuples if memory size permits. Oka and Matsuzaki [20] and Matsuzaki [14] analyzed the perfor-

mance of players that utilizes many 6-tuples or 7-tuples. Jaśkowski's redundant encoding [11] is also a technique to increase the number of N-tuples (and we can save the number of weights by using smaller additional tuples).

(3) *Multi-staging*. Multi-staging is a technique to divide a game into multiple stages and to use different tables of weights for each stage. This idea was first introduced for Game 2048 by Wu et al. [29].

The feature weights are adjusted by reinforcement learning methods. For Game 2048 players, the temporal difference learning (TD learning) was commonly used [25], [29], [32], and then a learning-rate-free variant (temporal coherence learning; TC learning) was introduced [11], [15]. Due to the characteristics of the game, biasing the training actions to learn sometimes improves the performance such as carousel shaping [11] and restart strategy [15].

The state-of-the-art player by Jaśkowski [11] was based on a network with five 6-tuples, five 4-tuples and two 3-tuples which was adjusted by the temporal coherence learning with some other improvements. It achieved an average score 324,710 with the greedy (1-ply search) play and 609,104 with the expectimax search within a time limit of 1 second. Though NTNs have worked fine, a weakness remains or namely missing generalization. Since the weights are basically independent from each other, NTNs do not obtain some important property of the game (for example, the similarity among 1024–2048–4096 tiles and 2048–4096–8192 tiles). Weight promotion [11], [15], which initializes a first-accessed weight with a certain existing one, can be considered as a human-aided solution to this issue. A more affirmative reuse of feature weights achieved even a 65536-tile [10]. Another heuristic approach is to use numbers relative to the maximum number on the board [7], but as far as the authors know there were no NTNs that utilized this approach.

3.2 Players based on Neural Networks

Behind the success of NTNs, (deep) neural networks have not been greatly studied or utilized for the development of Game 2048 players. As far as the authors know, the work by Guei et al. [9] was the only study published on the subject. Some open-source programs have been developed, for instance, with a multilayer perceptron [7], [28], with a convolutional neural network [26], [27], with a residual network [1] and with a recurrent neural network [21], but the performance of these players was not very good or not analyzed well (at least from the documents provided). In the following of this section, we review the networks and their training methods by Guei et al. [9] and tjwei [26].

Guei et al. [9] first tried to develop Game 2048 players based on convolutional neural networks. They developed two networks, one with 2×2 filters and the other with 3×3 filters. The first network consists of two convolution layers and two full-connect layers. The input board was encoded to a 4×4 16-channel image, where each channel corresponds to either empty cells, 2-tiles, 4-tiles, ..., or 32768-tiles. Then, 2×2 filters are convoluted twice followed by ReLU, which result in a set of 2×2 images (the number of filters for these convolution layers was not described in the paper). The pixel values are flattened and then processed with two full-connect layers. The output consists of a single value (for TD learning) or four values (for Q learning). The second network is different from the first one in terms of the size of filters and the number of convolution layers or namely 3×3 filters are convoluted (with zero padding) three times. The weights in these networks are adjusted by TD-learning and Q-learning methods with the results of selfplays. The best average score achieved with the first network was about 11,400 and with the second network about 5,300.

The neural network developed by tjwei [26] consists of two convolution layers followed by a full-connect layer. The input is a 4×4 16-channel image. In the first convolution layer, 2×1 filters and 1×2 filters are applied concurrently and then ReLU, yielding a 3×4 512-channel image and a 4×3 512-channel image. In the second convolution layer, 2×1 filters and 1×2 filters are applied concurrently to both intermediate images, yielding four 4096-channel images. The pixel values are finally processed in a full-connect layer to a single output value. The output values are a so-called *afterstate* value, and they are adjusted to expected rewards until the game end. Two training methods are used: reinforcement learning with selfplays and supervised learning from simulation of an existing strong player. In the case of reinforcement learning, the average score achieved was about 33,000. In the case of supervised learning, a player with tree search and heuristic evaluation function (Ref. [30], the median score 387,222) was used for simulation, and the average score achieved was 85,351.

4. Design

In this study, we designed deep convolutional neural networks (DCNNs) that had more convolution layers than prior work [9], [26]. Our DCNNs took an input board and computed four values, each of which represented the probability for selecting a direction (i.e., policy networks). We adjusted the weights of DCNNs by supervised learning with play records of existing strong players.

Though it was shown by Szubert and Jaśkowski [25] that we could develop good Game 2048 players based on value networks (of *afterstate*) trained with reinforcement learning methods, we used policy networks with supervised learning for the following reasons. First, we obtained much better results with a supervised learning method than a reinforcement learning method in a previous work by tjwei [26]. Secondly, the training time would be too long if we did reinforcement learning for DCNNs. Thirdly, we had some experience showing that the play records from an expectimax search player were not well suited to value network training [8], [16]. Due to these reasons, in this study we investigated use of a policy network and supervised learning combination and left other combinations as a topic for future study.

In this section, we show the structure of designed DCNNs, the method of supervised learning, and the play method with the trained DCNNs.

4.1 Structure of Our DCNN

As stated in the previous section, most of existing NN-based players consisted of two (or three) convolution layers. Since the

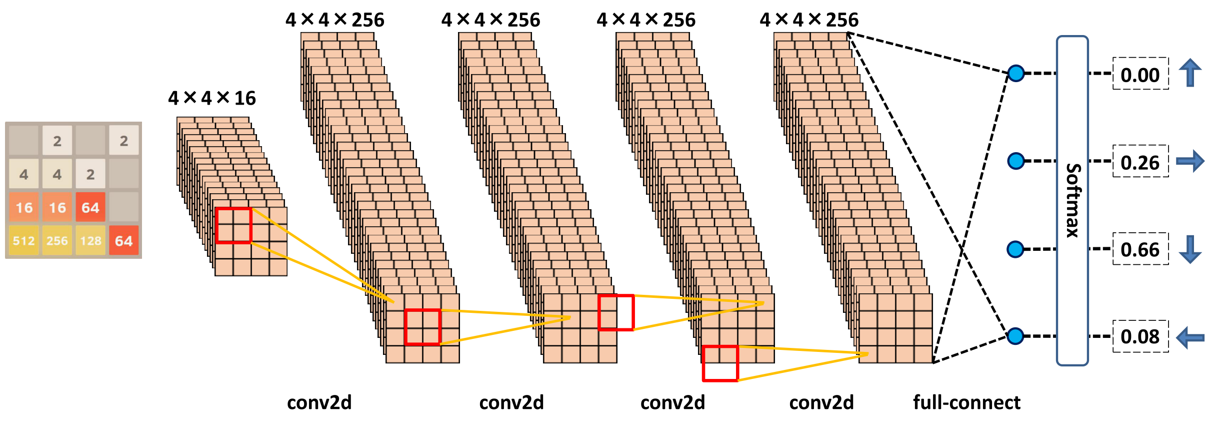


Fig. 2 Overview of our deep convolutional neural network.

Table 2 Numbers of convolution layers, channels and weights.

layers k	channels $ch(k)$	weights
2	436	817,068
3	312	819,628
4	256	820,228
5	222	818,074
6	200	826,804
7	182	819,550
8	168	813,124
9	158	820,498

game is played on a small 4×4 board, two convolution layers might suffice to cover the board. We, however, considered that those networks were too shallow to obtain good (and versatile) knowledge in the games, and designed DCNNs with more convolution layers. **Figure 2** depicts the structure of our DCNN for the case of four convolution layers. In general, our DCNNs have k convolution layers, a full-connect layer, and a softmax.

The input board is encoded to a 4×4 16-channel image as the work by Guei et al. [9]. Each channel represents the positions of empty cells, 2-tiles, 4-tiles, ..., and 32768-tiles, respectively. Then, 2×2 filters are convoluted k times. The stride width is one and zero padding is applied to keep the size of result images being 4×4 (`padding="SAME"` in TensorFlow). We used the same number $Ch(k)$ of filters (equal to the number of channels of intermediate images) for all the convolution layers. Note that the convolution is asymmetric in our case because the zero padding is applied only on the right and the bottom sides [*1]. After each convolution, bias is added and ReLU is applied. After the convolution layers, all the pixel values are processed in the full-connect layer that outputs four values. After the softmax, the four values represent probabilities for selecting the four directions.

The number of weights in our DCNNs is

$$\begin{aligned}&(2 \cdot 2 \cdot 16 + 1) \cdot Ch(k)\\ &\quad + (2 \cdot 2 \cdot Ch(k) + 1) \cdot Ch(k) \cdot (k - 1)\\ &\qquad + (4 \cdot 4 \cdot Ch(k) + 1) \cdot 4\,.\end{aligned}$$

We selected the number $Ch(k)$ so that designed DCNNs have almost the same number of weights in order to determine the importance of depth of DCNNs. **Table 2** shows the number of layers k, the number of channels $Ch(k)$ and the number of weights in the DCNNs. Note that the number of weights is between 810,000–830,000, which is much smaller than that of current players (comparable to that of Ref. [25]).

[*1] We can make the convolution symmetric if we permit changes in board size, for example $4 \times 4 \Rightarrow 3 \times 3 \Rightarrow 4 \times 4 \Rightarrow \cdots$.

4.2 Supervised Learning

Since the authors developed strong computer players for Game 2048 [15], we used play records of existing players as the training data of supervised learning.

Since our DCNNs output probability $P(i)$ for each move i, the supervised learning adjusts the weights to maximize the probability of the desired move. To this end, we defined the error E (loss function) based on the cross entropy as follows.

$$E = -\sum_{i=0}^{3} t(i)\, \ln P(i)$$

$$\textbf{where}\quad t(i) = \begin{cases} 1 & \text{if the existing player selected move } i \\ 0 & \text{otherwise} \end{cases}$$

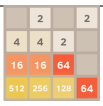
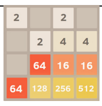
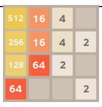
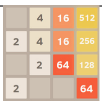
We selected three computer players from the artifacts of our previous work [15] to generate the training data. These players utilized N-tuple networks as the evaluation functions: the networks consisted of four 6-tuples and the game was split into eight stages based on the maximum number of tiles. The weights of the networks were adjusted by backward temporal coherent learning with restart strategy. These players selected moves by the 3-ply expectimax search. The difference in computer players was only the weights. The average scores of the players were 459,455, 463,660 and 460,069.

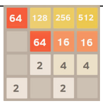
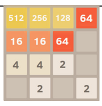
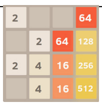
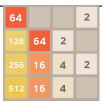
For the training data, we selected 6×10^8 actions from play records of these players (these actions were extracted from 54,000 games). Each action consists of the board status to play and the direction the player selected [*2]. Note that the training data were shuffled before fed to supervised learning.

We would like to add short remarks about symmetry. The board and the rule of Game 2048 is rotation and reflection-symmetric but the moves of a player are not usually symmetric. The NTN-based players used in this study were trained in a completely symmetric manner, and play records were not biased in terms of symmetry. To save training time, we did not feed symmetric boards in the training of DCNNs.

[*2] We extended the action data with the position of randomly appearing tiles and the score to the end of the game in supervised training of the N-tuple network and tjwei's network.

Table 3 Example of state and its symmetries.

				
up	0.000	0.000	0.188	0.095
right	0.256	0.178	0.001	**0.644** (↓)
down	**0.660** (↓)	**0.560** (↓)	**0.444** (→)	0.252
left	0.084	0.261	0.367	0.009

				
up	**0.673** (↓)	**0.649** (↓)	0.317	0.248
right	0.083	0.147	**0.372** (↓)	0.001
down	0.000	0.001	0.312	0.196
left	0.244	0.202	0.000	**0.556** (↓)

4.3 Playing Method

As we discussed above, the structure of DCNNs was asymmetric due to the asymmetric zero padding and we did not fed symmetric boards in the training. Instead, we take the symmetry into account in play with our DCNNs, that is, we generate eight symmetric boards and feed each of the eight boards to our DCNNs. **Table 3** gives an example. For each board, our DCNN returns the probabilities of the moves. We pick up the move with the largest probability and compute the corresponding move in the original board (written in the parentheses). We finally select the move to play with a majority vote. If two or more moves become the majority, we select the move based on the sum of probabilities.

Since the training data were not biased, we had considered this use of symmetric boards worked insignificantly, but in fact it improved the score to a degree. Another design choice of selecting a move from symmetric ones was simply based on the sum of probabilities. In our preliminary test, majority vote performed better than sum of probabilities.

5. Experiments

5.1 Implementation and Experiment Settings

We implemented DCNN players using the TensorFlow framework [*3]. The supervised learning was executed with a batch of 1,000 actions. We used `tf.train.AdamOptimizer` [*4] for the optimization algorithm with the learning parameter 0.001. The initial values of weights were set randomly between -0.1 and 0.1.

During the training phase, we observed the progress of learning through the error E and the accuracy. Error (loss) was averaged over a batch. The accuracy was calculated during the training using the training data itself.

After each training with 2×10^7 actions, we took a snapshot of the weights and performed test plays of 1000 games. After the test plays, we calculate the average score, the maximum score, and the ratio for reaching a 2048 tile.

[*3] The program code and training data used in this study are available at authors' webpage: http://ipl.info.kochi-tech.ac.jp/matsuzaki-lab/repos/JIP2018Supplements/.

[*4] This optimizer implements Adam algorithm [12]. We followed tjwei's implementation [26] for the selection of the optimization algorithm.

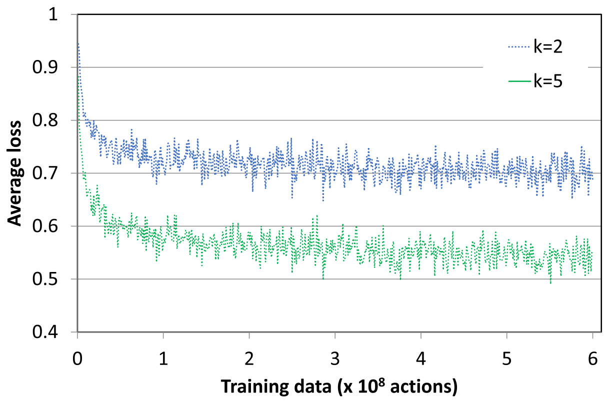


Fig. 3 Transition of error over learning.

Table 4 The average error and accuracy during 5.9×10^8–6.0×10^8 actions.

layers	error	accuracy
2	0.698	0.692
3	0.543	0.749
4	0.537	0.750
5	**0.528**	**0.758**
6	0.532	0.755
7	0.544	0.748
8	0.551	0.751
9	0.552	0.744

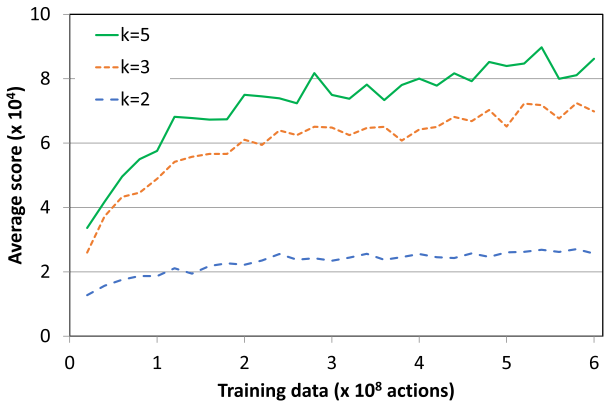


Fig. 4 Transition of average score over learning.

5.2 Comparing Networks with Different Number of Convolution Layers

The progress of training was plotted in **Fig. 3**. We plotted the cases with two convolution layers and five convolution layers only, because the graphs for three to nine convolution layers were quite similar. From Fig. 3, the training proceeded quickly up to 1×10^8 actions, and did not stop even at 6×10^8 actions. We observed rather large statistical variance of the error (and also the accuracy). We considered that the large variance was caused by wide variety of states (games often have more than 20,000 moves) compared with the batch size (1,000). **Table 4** summarized the average error and the accuracy of selecting moves during training with 5.9–6.0 $\times 10^8$ actions. Generally speaking, the smaller the error the larger the accuracy. The smallest error and highest accuracy were achieved with five convolution layers, and the results of three to seven convolution layers would be within the variance.

We then plotted the average scores of test plays in **Fig. 4**. We selected those cases with two, three, and five convolution layers: the graphs for five and six convolution layers were close and the graphs for four, six, seven, eight, and nine convolution layers

Table 5 Average score, maximum score, and ratio for reaching a 2048 tile.

	average score			maximum	2048
layers	2×10^8	4×10^8	6×10^8	score	ratio
2	22,189	25,530	25,669	175,628	45.6
3	61,037	64,212	69,840	332,868	79.4
4	65,022	73,054	80,284	343,496	83.3
5	**74,996**	**80,030**	**86,203**	386,972	**86.5**
6	69,482	74,441	83,791	387,376	83.5
7	67,874	77,448	79,812	**401,912**	83.1
8	64,465	74,737	74,787	363,916	81.1
9	66,732	73,484	68,129	358,736	75.9

Table 6 Maximum tiles out of 1000 test plays.

layers	≤256	512	1024	2048	4096	8192	16384
2	109	134	301	307	148	1	0
3	39	39	128	188	412	186	8
4	46	36	85	172	387	263	11
5	29	30	76	154	412	288	11
6	55	33	77	152	378	284	21
7	27	39	103	166	382	275	8
8	35	51	103	190	363	245	13
9	50	54	137	187	348	216	8

were between those of three and five convolution layers at most of the points. **Table 5** summarized the average scores of the test plays after the training of 2×10^8, 4×10^8 and 6×10^8 actions, the maximum score over all test plays (up to training 6×10^8 actions), and the ratio for reaching a 2048 tile after the training with 6×10^8 actions. From Fig. 4 and Table 5, we claim that the player with two convolution layers performed apparently worse than those with more convolution layers. The best average score was 86,203 with five convolution layers, which was a bit higher than the average score of tjwei's player [26]. Note that the difference was within the standard deviation, which was 1441 for ten trials. The best maximum score was 401,912 with seven convolution layers. Note that the average score were still increasing in training with 6×10^8 actions as we can see in Fig. 4.

Table 6 summarized the number of test plays categorized by the maximum number of tiles at the game end. Unfortunately, the players could not reach a 32768-tile. The best player with six convolution layers reached a 16384-tile in 2% of the games. This ratio was higher than that achieved by tjwei's player [26], while the ratios for reaching 8192, 4096 and 2048 tiles were lower.

5.3 Analysis of DCNN with Five Convolution Layers

We conducted two additional sets of experiments to investigate our DCNN with five convolution layers.

One was to study the relationship between the number of weights and the strength of the player. We let the number of channels *Ch*(5) be 110, 156, or 192, so that the number of weights became about 25%, 50%, and 75% of the original case, respectively. **Table 7** shows the average score and maximum score of test plays of 1000 games after training with 6×10^8 actions. From these results, we confirm that we can improve the performance of DCNN-based players by increasing the number of weights.

The other was to continue the training with more actions. To this end, we performed the training with the same set of training data again (i.e., second epoch), instead of preparing more training data. **Table 8** shows the average score and maximum score of test plays of 1000 games, until the training with 12×10^8 actions. The performance of the DCNN player improved after the training

Table 7 Experiment results by changing the number of channels (of DCNN with five convolution layers, after learning 6×10^8 actions).

channels	number of weights	average	maximum
110	208,234	44,056	255,620
156	410,128	66,133	284,916
192	615,364	77,426	289,968
222	818,074	86,203	332,496

Table 8 Experiment results of DCNN with five convolution layers up to 12×10^8 actions.

number of actions		average	maximum
1st epoch	1×10^8	57,564	294,016
	2×10^8	74,996	333,228
	3×10^8	74,965	330,600
	4×10^8	80,030	354,172
	5×10^8	83,939	384,648
	6×10^8	86,203	332,496
2nd epoch	7×10^8	82,845	378,476
	8×10^8	86,660	333,912
	9×10^8	90,725	342,900
	10×10^8	93,413	346,540
	11×10^8	87,060	343,112
	12×10^8	93,830	332,984

with 6×10^8 actions, and the best average score was 93,830 after the training with 12×10^8 actions [*5].

6. Discussion

The most interesting result in this study is that the performance of the player improved significantly from two convolution layers to three convolution layers. Since the size of board is just 4×4, applying 2×2 filters twice would cover the whole board. One possible reason is related to generalization of knowledge obtained through the training phase. Let us consider combinations of tiles on an edge: [128, 64, 32, 16], [256, 128, 64, 32], and [512, 256, 128, 64]. We can easily notice that these combinations have a same pattern. However obtaining generalized knowledge with just two convolution layers is not practical (we need to encode combinations independently in the weights). If we have three (or more) convolution layers, we could encode the knowledge in the additional layer(s). In our following study [17], we tried to unveil the internal behavior of our DCNN-players for networks with two and three convolution layers. We found two important facts in our DCNNs. First, we obtained a kind of generalized knowledge even with the two-convolution layers, but it was based on mixing (independent) features. Secondly, the knowledge obtained with the three convolution layers was clearer. This could be a reason why the players with three or more convolution layers performed almost at the same level as the existing NN-based player [26], while the number of weights is much smaller.

It is also interesting that the results in this study are much better than those in the work by Guei et al. [9], even in the case that the network consists of two convolution layers. There could be several reasons for the improvement: the difference of learning method (we used supervised learning instead of reinforcement learning); the number of weights available in the networks might be too small, etc.

Under the condition of a similar number of weights, the pro-

[*5] We did not obtain much improvements in the maximum score, because we ran into a brick wall when creating a 32768-tile after creating a 16384-tile and a 8192-tile (about 320,000).

posed DCNN players performed better than existing players including NTN-based ones. The first NTN-based player developed by Szubert and Jaśkowski [25] achieved an average score 51,320. We also generated an NTN-based player with the techniques in Ref. [15], and the average score was 50,120. We also trained an NTN by supervised learning with the same set of training data. Since the designed NTN was for the value of afterstates, we adjusted the weights to be close to the score from the state to the game end. However, the average score of the NTN trained by supervised learning was 999. We also implemented a smaller variant of tjwei's network [26] by halving the number of filters (256 filters for the first layer and 2048 filters for the second layer. Total number of weights was 2,189,312.) and trained by supervised learning with the same set of training data. The average score of the smaller tjwei's network was 917 [*6].

There is still a large gap in the performance of the state-of-the-art computer players based on NTNs [11]. We consider the following two causes for the performance gap. Firstly, as we discussed in Section 3.1, we can easily increase the number of weights due to the simple design and implementation of the NTNs. The Jaśkowski's player [11] had 1,600 times as many weights as our DCNNs. Secondly, it is easy to integrate search methods with the value network. It is reported that with the expectimax search within a time limit of 1 second, the average score increased from 324,710 to 609,104.

One drawback of the NN-based method is the long training time and long playing time. It took about 4.8 hours for training our DCNN-based players with 6×10^8 actions on a computer equipped with an NVIDIA GeForce GTX 1080 Ti GPU. This training time is about 11 times longer than that of an NTN-based player [*7]. For the playing time, our DCNN-based players select a move in 1.8 ms on a computer equipped with an NVIDIA GeForce GTX 1080 Ti GPU. This playing time is 1,400 times longer than that of an NTN-based player [*8].

7. Conclusion

In this paper, we developed computer players for Game 2048 based on deep convolutional neural networks trained by supervised learning. We changed the number of convolution layers from two to nine while keeping the total number of weights. These networks were trained with play records from existing strong computer players.

The experiment results showed some interesting findings. The computer player with two convolution layers did not perform well, and computer players with three or more convolution layers did much better, even with similar number of weights. The best player with five convolution layers achieved an average score 93,830 without combining any search techniques, which was higher than existing NN-based players. The average score was also higher than that of NTN-based players under a similar number of weights. The computer player with seven convolution layers achieved the maximum score 401,912, and this suggested that a deeper network would perform better if we could use more weights and training data.

One topic for future work is to identify the knowledge that our DCNN players have obtained by investigating the weights in the networks. A part of this work will be reported in Ref. [17]. We assume that the DCNNs successfully encoded some generalized knowledge, which is hard to obtain in N-tuple networks. We also want to increase the number of weights and training data and improve the performance of DCNN players for Game 2048.

Acknowledgments The training data used in this study were generated with the support of the IACP cluster in Kochi University of Technology.

[*6] The poor results of supervised learning for these value networks are consistent to our past results on NTN-based players [8]. We expect from our previous results [16] that the play results of expectimax search with NTNs are not suitable for learning.

[*7] In the case of our NTN-based player, the training time was 0.44 hours on an Intel Xeon E5-2620 v3 CPU.

[*8] In the case of our NTN-based player, the playing time was 1.3 μs per move.

References

[1] Allik, K., Rebane, R.-M., Sepp, R. and Valgma, L.: 2048 Report, available from ⟨https://neuro.cs.ut.ee/wp-content/uploads/2018/02/alphago.pdf⟩ (2018).

[2] Boris, T. and Šuković Goran: Evolving Neural Network to Play Game 2048, *Proc. 24th Telecommunications forum* (*TELFOR 2016*), IEEE (2016).

[3] Chabin, T., Elouafi, M., Carvalho, P. and Tonda, A.: Using Linear Genetic Programming to Evolve a Controller for the game 2048 (2015), available from ⟨http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/Treecko.pdf⟩.

[4] Cirulli, G.: 2048, available from ⟨http://gabrielecirulli.github.io/2048/⟩ (2014).

[5] David, O.E., Netanyahu, N.S. and Wolf, L.: DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess, *International Conference on Artificial Neural Networks and Machine Learning* (*ICANN 2016*), pp.88–96 (2016).

[6] Dedieu, A. and Amar, J.: Deep Reinforcement Learning for 2048 (2017), available from ⟨http://www.mit.edu/˜amarj/files/2048.pdf⟩.

[7] Downey, J.: Evolving Neural Networks to Play 2048 (2014), available from ⟨https://www.youtube.com/watch?v=jsVnuw5Bv0s⟩.

[8] Fujita, R. and Matsuzaki, K.: Improving 2048 Player with Supervised Learning, *Proc. 6th International Symposium on Frontier Technology*, pp.353–357 (2017).

[9] Guei, H., Wei, T., Huang, J.-B. and Wu, I.-C.: An Early Attempt at Applying Deep Reinforcement Learning to the Game 2048, *Workshop on Neural Networks in Games* (2016).

[10] Guei, H. and Wu, I.-C.: personal communication (2018).

[11] Jaśkowski, W.: Mastering 2048 with Delayed Temporal Coherence Learning, Multi-Stage Weight Promotion, Redundant Encoding and Carousel Shaping, *IEEE Trans. Computational Intelligence and AI in Games*, Vol.10, No.1, pp.3–14 (2018).

[12] Kingma, D.P. and Ba, J.: Adam: A Method for Stochastic Optimization, *Proc. 3rd International Conference on Learning Representations* (*ICLR*) (2015).

[13] Lai, M.: Giraffe: Using Deep Reinforcement Learning to Play Chess, Master's thesis, Imperial College London, arXiv, Vol.1509.01549v1 (2015).

[14] Matsuzaki, K.: Systematic Selection of N-tuple Networks with Consideration of Interinfluence for Game 2048, *Proc. 2016 Conference on Technologies and Applications of Artificial Intelligence* (*TAAI 2016*) (2016).

[15] Matsuzaki, K.: Developing 2048 Player with Backward Temporal Coherence Learning and Restart, *Proc. 15th International Conference on Advances in Computer Games* (*ACG2017*), pp.176–187 (2017).

[16] Matsuzaki, K.: Semi-Greedy Reinforcement Learning Algorithms for 2048, *Proc. 59th Programming Symposium*, pp.105–117 (2018). In Japanese.

[17] Matsuzaki, K. and Teramura, M.: Interpreting Neural-Network Players for Game 2048, *Proc. 2018 Conference on Technologies and Applications of Artificial Intelligence* (*TAAI 2018*), pp.136–141 (2018).

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M.: Playing Atari With Deep Reinforcement Learning, *NIPS Deep Learning Workshop* (2013).

[19] Moravcík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M. and Bowling, M.H.: DeepStack: Expert-level artificial intelligence in heads-up no-limit poker, *Science*,

Vol.356, No.6337, pp.508–513 (2017).

[20] Oka, K. and Matsuzaki, K.: Systematic Selection of N-tuple Networks for 2048, *Proc. 9th International Conference on Computers and Games* (*CG2016*), Vol.LNCS 10068, pp.81–92, Springer (2016).

[21] Samir, M.: 2048 Deep Recurrent Reinforcement Learning, available from ⟨https://github.com/georgwiese/2048-rl⟩.

[22] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.: Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol.529, No.7587, pp.484–489 (2016).

[23] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D.: Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, arXiv, Vol.1712.01815 (2017).

[24] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D.: Mastering the game of Go without human knowledge, *Nature*, Vol.550, pp.354–359 (2017).

[25] Szubert, M. and Jaśkowski, W.: Temporal Difference Learning of N-Tuple Networks for the Game 2048, *2014 IEEE Conference on Computational Intelligence and Games*, pp.1–8 (2014).

[26] tjwei: A Deep Learning AI for 2048, available from ⟨https://github.com/tjwei/2048-NN⟩.

[27] Virdee, N.: Trained A Convolutional Neural Network To Play 2048 using Deep-Reinforcement Learning (2018), available from ⟨https://github.com/navjindervirdee/2048-deep-reinforcement-learning⟩.

[28] Wiese, G.: 2048 Reinforcement Learning, available from ⟨https://github.com/georgwiese/2048-rl⟩.

[29] Wu, I.-C., Yeh, K.-H., Liang, C.-C., Chang, C.-C. and Chiang, H.: Multi-Stage Temporal Difference Learning for 2048, *Technologies and Applications of Artificial Intelligence*, Lecture Notes in Computer Science, Vol.8916, pp.366–378 (2014).

[30] Xiao, R.: nneonneo/2048-ai (2015), available from ⟨https://github.com/nneonneo/2048-ai⟩.

[31] Yakovenko, N., Cao, L., Raffel, C. and Fan, J.: Poker-CNN: A Pattern Learning Strategy for Making Draws and Bets in Poker Games, arXiv, Vol.1509.06731 (2015).

[32] Yeh, K.-H., Wu, I.-C., Hsueh, C.-H., Chang, C.-C., Liang, C.-C. and Chiang, H.: Multi-stage temporal difference learning for 2048-like games, *IEEE Trans. Computational Intelligence and AI in Games*, Vol.9, No.4, pp.369–380 (2016).

Naoki Kondo received his B.E. degree from Kochi University of Technology in 2018. His research interest is in game programming, in particular Japanese Chess (Shogi).

Kiminori Matsuzaki is a Professor of Kochi University of Technology. He received his B.E., M.S. and Ph.D. from The University of Tokyo in 2001, 2003 and 2007, respectively. He was an Assistant Professor (2005–2009) in The University of Tokyo, before joining Kochi University of Technology as an Associate Professor in 2009. His research interest is in parallel programming, algorithm derivation, and game programming. He is also a member of ACM, JSSST, and IEEE.