Improving DNN-based 2048 Players with Global Embedding

Wang Weikai
Graduate School of Engineering
Kochi University of Technology
Kami, Japan
258010i@gs.kochi-tech.ac.jp

Matsuzaki Kiminori
School of Information
Kochi University of Technology
Kami, Japan
matsuzaki.kiminori@kochi-tech.ac.jp

***Abstract*—2048 is a popular game for which plenty of computer players have been created. However, many created 2048 players, especially all DNN-based ones, only implicitly use tile values as inputs and access tile position information. In this study, we take one of the best DNN-based 2048 players as a baseline and propose a 2048 player directly using both tile values and tile positions as inputs. Additionally, we explore the possibility of embedding all tile values and positions that we then concatenate with the network's regular value inputs.**

We first train these variations in a short session and then select the best two models with the baseline to be further trained in a long session. Our best two methods performed better than the baseline DNN player in both short and long training sessions.

***Index Terms*—game 2048, DNN, position embedding**

I. Introduction

In recent years, Deep Neural Networks (DNNs) have had great success in the development of computer game players. Several of them have achieved expert-level results, such as: AlphaGo Zero (Go) [1], Alpha Zero (Chess and Shogi) [2], AlphaStar (StarCraft II) [3], DeepStack (Poker) [4].

2048 is an "easy to learn but hard to master" game which made it popular globally on mobile. It is played on a 4×4 grid board. In an initial state, two tiles are placed randomly with values 2 ($p = 0.9$) or 4 ($p = 0.1$). After the player selects a direction, all tiles will move to it. Colliding tiles of the same value merge into a double valued tile. The player's score increases by the merged tile's value, and a random empty cell generates a tile of value 2 ($p = 0.9$) or 4 ($p = 0.1$). The game ends if the player cannot move the tiles in any direction. The original goal is to create a tile with a value of 2048, but it has already been trivialized using intelligent agents. The current goal is to reach as high score as possible before the game ends.

2048 agents mainly use N-Tuple Networks and DNNs to evaluate board states. So far, The best DNN-based 2048 agents [5] have not been able to outperform the state-of-the-art agent based on N-Tuple Networks [6]. Given the success of DNNs in other games, we expect that it is still possible to improve their performance in 2048.

In most board games, a game state consists of:

- pieces (in 2048, each piece is a tile with a value), and
- meta information (in 2048, this includes the turn number and the score).

The pieces are usually encoded as arrays of piece values that are fed to neural networks. Convolution layers in neural networks are good at capturing local features in their input. Then, positional relationships between pieces over the board are *implicitly* captured through multiple layers.

The success of Transformer networks [7] in Natural Language Processing (NLP) inspired us to design networks so that they *explicitly* handle the relationships of pieces. The Transformer networks are based on two main techniques: attention and position embedding. Both of these techniques allow a neural network to capture and handle relationships over the entire input.

In this paper, we focus on position embedding techniques applied to 2048. Starting with one of the best CNN (CNN22 [8]) as a baseline, we extend the network to explicitly take the information of piece (tile) positions.[1] We encode the tile positions with one-hot vectors (we call them *position vectors*). The position vectors are concatenated with the array of tile values and fed directly or indirectly to the CNN22 network.

The main contributions of the paper are summarized as follows.[2]

- We designed three networks that take tile values and tile positions as inputs (Section III). In all cases, we kept the baseline CNN22 structure and extended it to handle position vectors.
- To test our concept, we first evaluated the three proposed networks and the baseline with a short-term training session (about one day with a commodity GPU) (Section IV-A). The results showed that our proposed networks play better than the baseline with the help of position vectors.
- We further conducted a long-term training experiment (about ten days) for the best two networks and our baseline method (Section IV-B). In this experiment, we evaluated the networks using greedy (1-ply lookahead) and expectimax (3-ply lookahead) search, and compared them with the baseline DNN agent. Our best agent achieved an average score of 226,171 with greedy play

[1] We consider that 2048's board size is too small to benefit from the power of the attention mechanism.

[2] Programs are available at https://github.com/wwk1397/Improving-DNN-based-2048-Players-with-Global-Embedding

and 389,288 with expectimax (3-ply) search. These two results in long-term training are 51,961 and 9,374 better than the best method in [8].

II. CNN22: Baseline Neural Network

CNN22 [8] is a 5-layer convolutional neural network trained using TD-learning with some game-specific tricks. The input of CNN22 is an array of size (4,4,16) consisting of 4×4 one-hot vectors encoding tile values. CNN22 applies two convolution layers with 2×2 kernels followed by three fully-connected layers as shown in Fig. 1(a). The network's output is an evaluation value of the input state, where the value is trained to be an expected score from the state.

The training in [8] used the following settings and tricks:

- **5 generator threads:** Training data is generated with 5 computational threads that perform game plays with the latest neural network.
- **Batch1024:** Afterstates (states after moving tiles and before a random tile appearing) are sampled in groups of 1024.
- **S-jump:** Initial boards may have tiles with bigger value than rules dictate upon termination.
- **Restart:** Instead of resetting, a generator thread rewinds an episode by a half and resumes playing from there.

III. Neural Network Structures

As discussed in Introduction, convolution layers capture local spatial relationships effectively and find global relationships over multiple layers.

In 2048, as in other games, tile positions are important. One major heuristic in 2048 focuses on keeping the tile with the largest value in a board corner. Knowing explicitly the position of a tile can thus be useful to understand and learn this kind of strategy effectively.

A. Extended Input

We develop three networks by extending CNN22. The input of the network is extended to the global input with a value array and position array, which will be detailed in the following subsections.

Here is the definition of value array and position array. Parts of these arrays are shown in Fig. 2.

- **Value array**: Size (4, 4, 16). Includes 4×4 one-hot vectors of tile values.
- **Position array**: Size (4, 4, 4 + 4). Includes:
  - 4×4 one-hot vectors of row coordinates (4, 4, 4).
  - 4×4 one-hot vectors of column coordinates (4, 4, 4).

B. Direct Encoding of Positions

This input part directly feeds the concat array to the CNN22 network. Instead of only using the value array as the input, we add the position array into the input part by bypassing it. Therefore, we call this modified version the Bypass-CNN (BP-CNN). It is illustrated in Fig. 1(b).

TABLE I
Neural networks used in the paper

Network	number of weights	speed (pos./hour)
CNN22	2,902,273	2.41×10^6
BP-CNN	2,910,465	2.21×10^6
FC-CNN	2,927,009	2.22×10^6
CNN-CNN	2,930,577	2.00×10^6

We keep the number of channels after the convolution layers, and thus only the first convolution layer has more weights. Since the number of channels after the first layer is small, the number of weights has increased by 0.28%, as shown in Table I.

C. Extended Encoding of Board Information

We extend the idea of providing global information as input to a CNN network by adding different submodels, which mean learnable network structures of analyzing the global information, to the CNN22's input.

We create two networks with different submodules:

- **FC-CNN:** This network's submodule consists of two fully connected layers with 32 and 128 neurons respectively. The output of the second layer is reshaped to a $4 \times 4 \times 8$ array, concatenated with the Index input, and fed to CNN22. The structure of FC-CNN is presented in Fig. 1(c).
- **CNN-CNN:** This network's submodule contains two convolution layers using 2×2 kernels with 16 filters and 3×3 kernels with 128 filters respectively. As with FC-CNN, the submodule's output is then reshaped, concatenated, and fed to the CNN22 part of the network. The structure of CNN-CNN is presented in Fig. 1(d).

IV. Evaluation

A. Short Training Experiment

We first train each network for 5×10^7 actions. The progress is measured with greedy play.

During the training part, we use computers with an Intel Core i7-9800X CPU and two NVIDIA GeForce RTX 2080Ti GPUs. Each program in this study uses only one GPU.

The networks are trained using the S-jump and restart methods (introduced in section II). The speed of training and the number of weights of each network are shown in table I. To test the effect of adding position embedding, we do not increase the number of parameters of the original model more than 1%.

The snapshots with the network weights were taken for every 10^6 training actions. After training 50×10^6 actions, we got 50 snapshots for each network. For each snapshot, 100 greedy games are played for the evaluation.

Figure 3 shows training progress for four networks plotted with respect to the number of training actions.

FC-CNN and CNN-CNN get a better average score than CNN22. BP-CNN, however, only performs similarly to CNN22. This shows that adding the global input directly

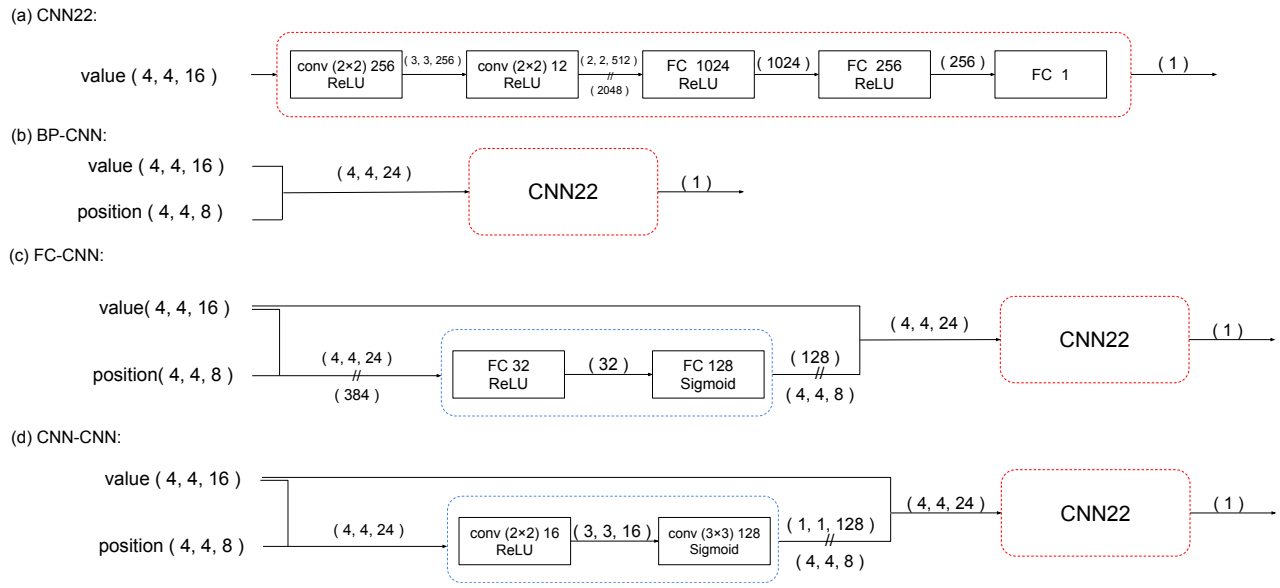

Fig. 1. Structures of (a) CNN22, (b) BP-CNN, (c) FC-CNN, and (d) CNN-CNN. "//" means the operation of tensor reshaping. The red dotted parts have the same structure. The additional blue dotted parts analyze global information.

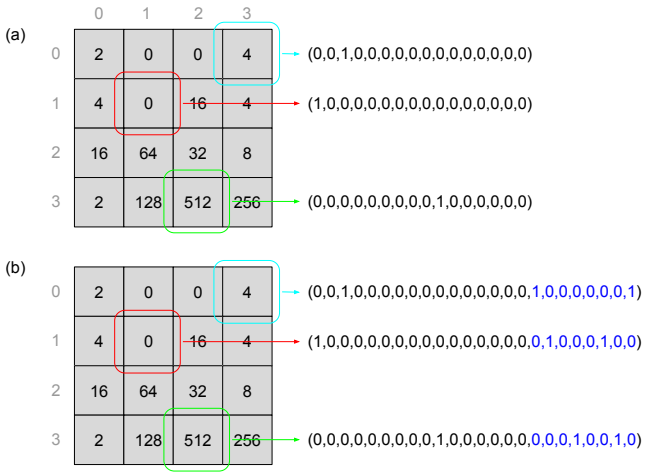

Fig. 2. Parts of the value arrays and the arrays concat with both value and position at (0,3), (1,1), and (3,2). The blue characters consist of row and column position one-hot vectors.

may not make the DNNs-based model analyze the global information effectively.

B. Long Training Experiment

Based on the short training experiment, we select the two best networks and the baseline network to be further trained for a total of 5×10^8 actions. The snapshots with the weights of the networks were taken every 10^7 training actions. Besides the number of training actions, we keep the same training parameters as for the short training.

We measure training progress with greedy (1-ply search), and expectimax (3-ply search) plays. In the greedy play, we play 300 games for each snapshot. In the expectimax play, we play 15 games for each snapshot.

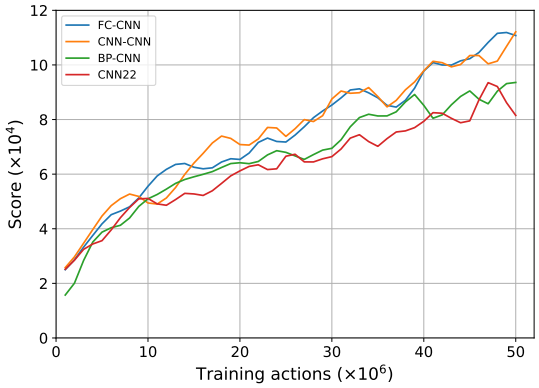

Fig. 3. Experiment 1. Average scores for different networks in greedy play.

As we can see in Table II and Fig. 4, with the expectimax (3-ply search) play, FC-CNN achieves its highest average score 389,288 at 400×10^6 actions. CNN-CNN gets its highest average score 375,794 at 450×10^6 actions. CNN22 gets its highest average score 379,914 at 450×10^6 actions.

With the greedy play, FC-CNN achieves its highest average score 226,171 at 500×10^6 actions. CNN-CNN gets its highest average score 195,816 at 450×10^6 actions. CNN22 gets its highest average score 174,210 at 450×10^6 actions.

Compared with our baseline model CNN22, the best average score of FC-CNN increased by 51,961 with greedy play, however, only 9,374 with expectimax (3-ply search) play. This may be caused by the greedy training method. In the training part, the data is generated by the neural network with

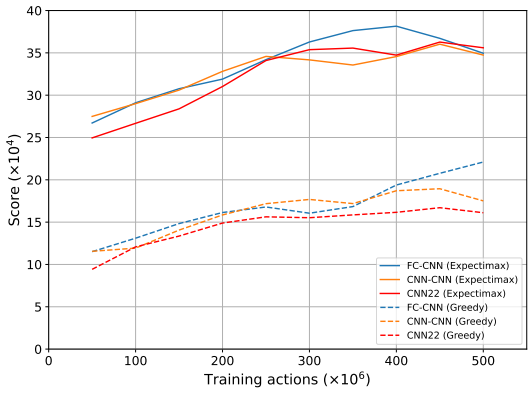

Fig. 4. Experiment 2. Average scores of greedy (1-ply) and expectimax (3-ply) play.

the greedy method. Therefore, the improvement of the neural network's effect may be more reflected in the greedy play result than the expectimax (3-ply search) play result.

V. Conclusion

In this paper, we focused on explicitly providing global information to CNN networks rather than implicitly. To do so, we created BP-CNN, FC-CNN, and CNN-CNN to utilize this extended information and added it to the input of CNN22, the previously established DNN-based 2048 player.

Using this method, we improved the results of the previous agent in both short and long term training. FC-CNN achieved the best performance with an average score of 389,288 with expectimax (3-ply) play and 226,171 with greedy play.

The highest average score of expectimax (3-ply) play of FC-CNN is higher than that of our baseline method CNN22 at 9,374, which is much smaller than the improvement of greedy play at 51,961. We speculate that changing training methods could yield improvements. It is also possible that different methods of encoding global information could deliver better results in the future.

Acknowledgment: Part of this work was supported by JSPS KAKENHI Grant Number 20K12124.

TABLE II

Result of Experiment 2. The column of the average score shows the mean (before ±) and the standard deviation (after ±) of average scores for four runs after 300×10^6 actions.

	average score	achievement ratio 2,048	4,096	8,192	16,384	32,768
Greedy play for FC-CNN						
300×10^6 actions	154,443± 18,465	90.8%	82.8%	62.5%	25.8%	1.3%
350×10^6 actions	160,329± 23,366	90.8%	81.5%	61.4%	29.7%	1.4%
400×10^6 actions	202,613± 20,547	95.9%	89.7%	74.1%	42.8%	3.1%
450×10^6 actions	203,371± 10,555	94.8%	89.1%	75.2%	41.3%	3.8%
500×10^6 actions	226,171± 9,747	96.0%	92.3%	80.8%	50.7%	3.7%
Expectimax 3-ply play for FC-CNN						
300×10^6 actions	365,134± 28,463	100.0%	100.0%	100.0%	83.3%	16.7%
350×10^6 actions	376,312± 24,369	100.0%	100.0%	95.0%	86.7%	23.3%
400×10^6 actions	389,288± 27,789	100.0%	100.0%	100.0%	88.3%	25.0%
450×10^6 actions	367,742± 9,486	100.0%	98.3%	98.3%	83.3%	18.3%
500×10^6 actions	343,985± 45,077	100.0%	98.3%	95.0%	83.3%	11.7%
Greedy play for CNN-CNN						
300×10^6 actions	184,937± 15,461	94.6%	89.2%	73.1%	37.1%	1.1%
350×10^6 actions	158,651± 31,244	92.7%	86.8%	66.8%	27.3%	1.3%
400×10^6 actions	194,519± 30,559	94.5%	89.9%	75.8%	40.0%	2.5%
450×10^6 actions	195,816± 11,764	93.3%	88.3%	73.5%	40.3%	2.4%
500×10^6 actions	169,130± 30,870	92.3%	82.7%	65.9%	31.0%	1.9%
Expectimax 3-ply play for CNN-CNN						
300×10^6 actions	341,023± 48,990	100.0%	98.3%	95.0%	81.7%	15.0%
350×10^6 actions	331,047± 31,102	100.0%	100.0%	98.3%	85.0%	6.7%
400×10^6 actions	340,149± 28,935	100.0%	100.0%	98.3%	71.7%	15.0%
450×10^6 actions	375,794± 28,195	100.0%	100.0%	96.7%	80.0%	21.7%
500×10^6 actions	339,600± 15,638	100.0%	98.3%	98.3%	78.3%	13.3%
Greedy play for CNN22						
300×10^6 actions	151,469± 3,926	93.8%	83.9%	63.2%	24.9%	0.7%
350×10^6 actions	161,216± 6,163	95.5%	88.7%	68.7%	27.6%	0.6%
400×10^6 actions	157,426± 8,645	95.1%	88.2%	66.2%	26.1%	0.1%
450×10^6 actions	174,210± 21,724	94.7%	85.8%	67.3%	33.6%	1.7%
500×10^6 actions	157,528± 31,473	91.4%	82.3%	63.9%	28.7%	1.4%
Expectimax 3-ply play for CNN22						
300×10^6 actions	351,799± 38,013	100.0%	100.0%	98.3%	80.0%	15.0 %
350×10^6 actions	366,480± 41,653	100.0%	100.0%	100.0%	91.7%	13.3%
400×10^6 actions	328,556± 21,488	100.0%	96.7%	96.7%	76.7%	11.7%
450×10^6 actions	379,914± 40,432	100.0%	100.0%	100.0%	90.0%	18.3%
500×10^6 actions	349,683± 25,745	100.0%	100.0%	98.3%	83.3%	11.7%

References

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

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm," *arXiv*, vol. 1712.01815, 2017.

[3] 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.

[4] M. Moravcík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. H. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[5] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver, "Planning in stochastic environments with a learned model," in *International Conference on Learning Representations*, 2021.

[6] H. Guei, L.-P. Chen, and I.-C. Wu, "Optimistic temporal difference learning for 2048," *IEEE Transactions on Games*, 2021.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] K. Matsuzaki, "Developing value networks for game 2048 with reinforcement learning," *Journal of Information Processing*, vol. 29, pp. 336–346, 2021.