Vertical Split Encoding: Enabling Larger N-Tuples for Stronger 2048 Play

Shunsuke Terauchi and Kiminori Matsuzaki[0009−0003−4663−3292]

Kochi University of Technology, Kami, Kochi 782–8502, Japan
295141a@gs.kochi-tech.ac.jp
matsuzaki.Kiminori@kochi-tech.ac.jp

Abstract. N-tuple networks are a simple yet efficient approach to developing computer players for the game 2048, but the tuple size has been limited to six or seven due to the memory constraints. In this study, we break through this limitation by proposing a novel encoding method called Vertical Split Encoding (VSE). VSE enables us to design N-tuple networks with 8-tuples or even 9-tuples. We confirm through experiments that the performance degradation resulting from applying VSE is reasonably small, and that the newly developed 8-tuple networks significantly outperform the 6-tuple baseline, which was used in the state-of-the-art computer player, under the same experimental setting.

Keywords: 2048 · N-tuple networks · Encoding.

1 Introduction

N-tuple networks (a.k.a. pattern-based evaluation functions in Othello [2]) are a simple yet efficient approach to designing evaluation functions for board games and other applications. N-tuple networks compute an evaluation value by sampling local features on the board, retrieving corresponding values from lookup tables, and summing them up (or calculating linear combination of them). The parameters in the lookup tables are often adjusted by supervised learning or reinforcement learning techniques. Several strong computer players have been developed based on N-tuple networks, for example, for Othello [2, 9, 15, 11], connect-4 [20], connect-6 [7], EinStein Würfelt Nicht! [3, 6].

The target game in this study is *2048* [4], a stochastic single-player game developed by G. Cirulli. Several computer players have been implemented [14, 1, 5, 17, 23, 8, 10, 24, 21], among which the most successful approaches employ N-tuple networks as evaluation functions [17] in combination with Expectimax search [23]. For instance, the state-of-the-art player developed by Guei et al. [5] achieved an average score of 625 377 by using two-stage N-tuple networks consisting of 8 × 6-tuples—trained by extended temporal-difference learning with optimistic initialization—, 6-ply Expectimax search, and a game-specific tile-downgrading technique.

In general, larger N-tuple networks yield better performance at the cost of larger memory sizes required and longer training times for tuning [19]. Let N-tuple networks consist of $m \times n$-tuples, then we can consider two approaches to enlarging N-tuple networks. The first approach is to increase the number m of tuples. While this is relatively straightforward to implement, Oka and Matsuzaki [16] demonstrated that the performance saturated only with this approach. The second approach is to increase the tuple size n. In the early stages of research on 2048, computer players improved the performance by enlarging 4-tuples to 6-tuples [17, 23]. However, as Jaśkowski [8] suggested, further enlargement has been considered infeasible for the game 2048 due to the exponential growth in the number of parameters. Note that, in 2048, a board cell can take 18 possible values, and N-tuple networks with $m \times n$-tuples have $m \times 18^n$ parameters. A 6-tuple network has 3.4×10^7 parameters (requiring 13 GB of memory [1]), a 7-tuple network has 6.1×10^8 parameters (235 GB), and an 8-tuple has 1.1×10^{10} parameters (4.2 TB).

In this study, we overcome the aforementioned limitation by proposing a novel encoding method, called *Vertical Split Encoding* (*VSE*). In k-VSE, we encode a board state into k board instances based on k value ranges of interest. Here, for each value range, encoded board instance takes less possible values by equating values outside the range (except for empty cells, represented as E). For example, with 2-VSE with value ranges of interest 2^1–2^8 and 2^9–2^{17}, a board $[E, 2^1, 2^{11}, 2^{12}, \ldots]$ is encoded into $[E, 2^1, L, L]$ and $[E, S, 2^{11}, 2^{12}]$, where L and S are newly introduced special labels denoting *larger* and *smaller* values, respectively. With VSE, the numbers of parameters required for $m \times n$-tuples in 2048 are reduced to $m \times 11^n$ with 2-VSE, $m \times 9^n$ with 3-VSE, and $m \times 7^n$ with 4-VSE. The reduction is drastic: for instance when $n = 8$, the numbers of parameters are reduced to 3.9% with 2-VSE, 1.2% with 3-VSE, and 0.21% with 4-VSE, respectively.

We demonstrate the effectiveness of VSE for 2048 with intensive experiments. In addition to 6-tuple networks without VSE, we developed computer players using larger N-tuple networks: 7-tuple networks with 2-VSE, 8-tuple networks with 3-VSE, and even 9-tuple networks with 4-VSE. All the N-tuple networks are successfully trained on a commodity computer using at most 64 GB of memory, and the 7-, 8-, and 9-tuple networks outperformed the 6-tuple baseline. In particular, the best 8-tuple networks achieved an average score of 407 206 with 1-ply lookahead (greedy) play, 547 365 with 3-ply Expectimax search, and 587 690 with 5-ply Expectimax search, which were significantly better than the 6-tuple baseline.

The main contributions of this study are twofold.

- We proposed Vertical Split Encoding (VSE) to enable employing large N-tuple networks, which had been impractical due to memory constraints.
- We experimentally demonstrated that large N-tuple networks with VSE outperformed the 6-tuple baseline under the same experimental setting.

More broadly, this work introduces a novel approach for reducing parameters in N-tuple networks, which would be widely applied across various games.

[1] The memory size is calculated for the case that $8 \times n$-tuple networks with 2-stage, 64 bits per parameter, are tuned with temporal coherence learning [8].

2 Game 2048

2048 is a single-player stochastic game played on a 4×4 grid. The objective of the original game is to reach a 2048-tile by moving and merging the tiles on the board according to the rules below. In an initial state, two tiles are placed randomly each with a number 2 (with probability 0.9) or 4 (with probability 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. 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 a number 2 (with probability 0.9) or 4 (with probability 0.1). If the player cannot move the tiles in any direction, the game ends.

Fig. 1 depicts the process of the game. Selecting right at the state s_t in Fig. 1, the tiles move to the right and two 2-tiles and two 8-tiles merge adding the score $4 + 16 = 20$. Then, a new tile appears to reach the next state s_{t+1}.

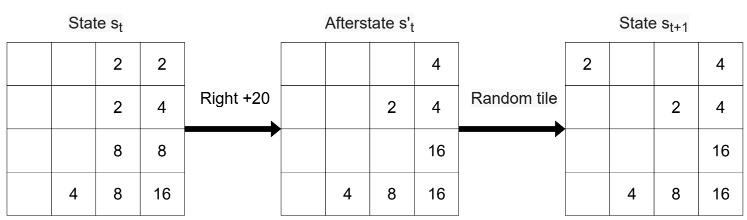


Fig. 1. Process of game 2048

Since a turn in 2048 consists of two steps, we introduce the two notions, called *state* and *afterstate* [17], as shown in Fig. 1.

- A *state* s_t is a board (and score) at which the player selects a move.
- An *afterstate* s'_t is a board (and score) after the slide-and-merge step and before a new tile appears.

For better understanding of the performance of the players, Table 1 summarizes the common score achieved and the common number of moves needed when the player reaches a tile of a specific value.

Table 1. Score and number of moves when a tile is first created

tile	2048	4096	8192	16384	32768
score	20 000	44 000	97 000	210 000	450 000
moves	950	1 900	3 800	7 600	15 200

3 Vertical Split Encoding for N-tuple Networks

The most successful approach to building a strong computer player for 2048 is to use an evaluation function based on N-tuple networks, whose parameters are tuned through reinforcement learning. N-tuple networks are function approximators implemented with (local) sampling of the board and corresponding lookup tables. For each N-tuple (examples of N-tuples are given in Fig. 5), we sample a set of cells corresponding to the tuple elements, retrieve corresponding values from the lookup table, and sum these values up to compute the overall evaluation value. A common practice in 2048 is to perform eight-fold sampling for each N-tuple by exploiting board symmetries [17], and does our study.

Increasing tuple size is expected to yield significant performance improvements. Oka and Matsuzaki [16] investigated the use of 6- and 7-tuples showing that the use of 7-tuples achieved superior performance. We also confirmed for mini2048 (a 3×3 variant of 2048) that increasing the size of N-tuples yielded performance improvement up to 6-tuples [19].

Nevertheless, N-tuple networks require a large amount of memory. When we consider m tuples of size n and each cell on the board can have one of x distinct values, a lookup table consists of $m \times x^n$ entries. In 2048, since the number of possible cell values is 18 ($\text{empty}, 2^1, 2^2, \ldots, 2^{17}$), the memory requirement becomes an issue as n increases. Table 2 summarizes the number of parameters required in a lookup table for a single n-tuple and for eight n-tuples, when n is between 6 and 9, as well as the memory usage when combining the currently dominant methods of multi-staging [23] and Temporal Coherence learning [8]. In practice, it has been believed that 7-tuples or larger could not work on commodity computers (e.g., within 64 GB of memory). Indeed, Jaśkowski [8] experimentally demonstrated that it was more effective to use 6-tuples in combination with the multi-staging technique, rather than using a single 7-tuple without multi-staging.

Table 2. Number of parameters and required memory size. In the calculation of memory size, we assume an implementation of temporal coherence learning with 2 stages and 64 bits per element.

network	parameters	memory	network	parameters	memory
$1 \times$ 6-tuple	34 012 224	1.5 GB	$8 \times$ 6-tuple	272 097 792	12.2 GB
$1 \times$ 7-tuple	612 220 032	27.4 GB	$8 \times$ 7-tuple	4 897 760 256	218.9 GB
$1 \times$ 8-tuple	11 019 960 576	492.6 GB	$8 \times$ 8-tuple	88 159 684 608	3.9 TB
$1 \times$ 9-tuple	198 359 290 368	8.9 TB	$8 \times$ 9-tuple	1 586 874 322 944	70.9 TB

To address the issue of memory requirement and hence to enable large N-tuple networks, this study proposes a novel method called *Vertical Split Encoding* (VSE) [2]. As noted above, the main reason for the huge memory requirement is

[2] The idea behind the term "Vertical": the input space of 2048 is in three dimensions, the first two are for the position on a board, and the third is for the cell values. Vertical Split Encoding works on this third dimension.

that the number of possible cell values on the board is large, $x = 18$. Here, we notice the following characteristics of 2048.

– The relative positions of tiles with *similar* values are important.
– Tiles with *widely different* values do not interact.
– The positions of empty cells are important.

Among these, the second characteristic suggests that, when looking at a particular tile value, it is unnecessary to distinguish tiles whose values are very far apart. For example, when looking at a tile of value 2^1, distinguishing whether the neighboring tiles are 2^{11} or 2^{12} is almost meaningless. Therefore, by equating tiles with widely different values, we can reduce the number of possible values and accordingly the number of parameters in the lookup tables.

Based on this idea, Vertical Split Encoding maps a board into multiple board instances, in each of which some board cells have the same label. Firstly, we define a set of *value ranges* of interest: all the values smaller than this range are replaced with S, and all the larger values are replaced with L. To address the third characteristic mentioned above, we treat empty cells (E) separately.

A concrete example is shown in Fig. 2, where the value ranges of interest are 2^1–2^8 and 2^9–2^{17}. In the first mapping, all the tiles with values larger than 2^8 are replaced by L. In the second mapping, all the tiles with values smaller than 2^9 are replaced by S. After these mapping, the resulting possible values are $[E, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, L]$ in the first mapping and $[E, S, 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, 2^{17}]$ in the second mapping. As a result, the numbers of possible values are $x_1 = 10$ and $x_2 = 11$, respectively, which significantly reduces the memory requirement for the lookup tables.

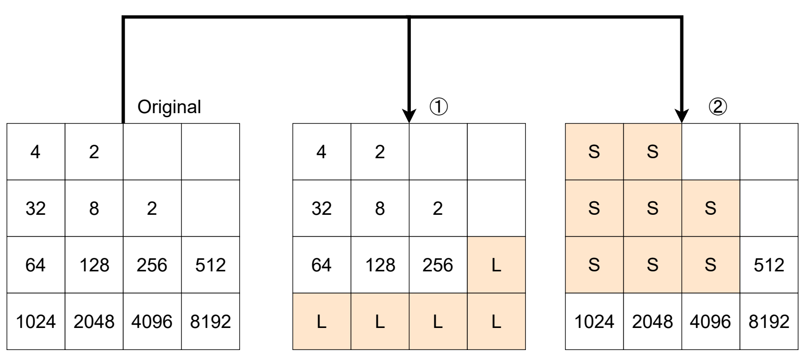


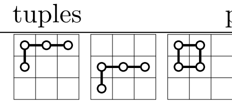
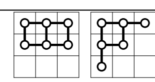
Fig. 2. An example of Vertical Split Encoding with two value ranges 1–8 and 9–17.

We can extend the idea to k-VSE, with k value ranges of interest. In the experiments on 2048 described later, we apply 2-, 3-, and 4-VSE.

4 Preliminary Experiments on Mini2048

We first conducted evaluations of VSE using mini2048, a smaller variant of 2048, to gain insights of how much the application of VSE degrades the performance and of which factors of VSE affect the performance. Mini2048 is a smaller variant of 2048 where the game rules are identical to 2048 except for the board size being 3×3. We chose Mini2048 as the testbed of our preliminary experiments for two reasons: it was strongly solved (a.k.a. perfectly analyzed) [22, 18] and the expected score of the optimal play is known to be 5468.49; we confirmed in our previous work [19] that enlarging the size of N-tuple yielded better performance up to 6-tuples.

Table 3. N-tuple networks used for mini2048.

name	tuples	parameters
mNT4		43 923
mNT6		3 543 122

In the preliminary experiments, we used two manually designed N-tuple networks in Table 3: mNT6 with two 6-tuples and mNT4 with three 4-tuples. We applied multiple VSEs on mNT6. The main questions we wished to examine were the following:

- To what extent does VSE degrade the performance?
- Should we have overlaps between value ranges when splitting?
- Which factor of VSE affects the performance most strongly?

For each configuration, we trained N-tuple networks using Temporal Coherence learning with Optimistic Initialization [5] (initial value 1200) for 5×10^8 steps. After the training, we played 1 000 games with 1-ply lookahead (the greedy play), and evaluate the performance in terms of the average score. To mitigate the effect of randomness, we conducted each training with 10 different random seeds. Table 4 summarizes the configurations of VSEs and the average score of the trained N-tuple networks.

The results in Table 4 were highly promising. All the results from mNT6 plus VSEs outperformed that of mNT4. While applying VSE degrades the performance (for the case of mNT6, <5%) in general, the N-tuple networks with 2-VSE-D was slightly better than the original mNT6 without VSE. From the results of 2-VSE-A, 2-VSE-B, and 2-VSE-C as well as 3-VSE-A and 3-VSE-B, we found that it was not necessary to introduce overlaps between value ranges—this was contrary to the authors' initial assumption. The factor with the greatest impact on performance was the number of elements included in the smallest value

Table 4. Configurations of preliminary experiments for mini2048 and the average score for 1000 games with 1-ply lookahead. The number in parentheses after each value range shows the number of possible values after corresponding mapping. For average scores, mean and standard deviation over 10 training runs (after the ± sign) are given.

tuple	VSE	value ranges	parameters	ave. score
mNT6	no VSE	2^1–2^{10} (11)	3 543 122	4 610.2± 81.1
	2-VSE-A	2^1–2^6 (8), 2^5–2^{10} (8)	1 048 576	4 555.8± 91.7
	2-VSE-B	2^1–2^6 (8), 2^6–2^{10} (7)	759 586	4 584.0± 69.9
	2-VSE-C	2^1–2^6 (8), 2^7–2^{10} (6)	617 600	4 557.3± 74.5
	2-VSE-D	2^1–2^5 (7), 2^6–2^{10} (7)	470 596	4 621.1± 83.5
	2-VSE-E	2^1–2^4 (6), 2^5–2^{10} (8)	617 600	4 441.4± 96.7
	3-VSE-A	2^1–2^4 (6), 2^5–2^7 (6), 2^8–2^{10} (5)	217 874	4 390.9±111.6
	3-VSE-B	2^1–2^4 (6), 2^4–2^7 (7), 2^7–2^{10} (6)	421 922	4 464.9± 64.7
	4-VSE-A	2^1–2^4 (6), 2^5–2^6 (5), 2^7–2^8 (5), 2^9–2^{10} (4)	155 839	4 451.7± 96.7
mNT4	no VSE	2^1–2^{10} (11)	43 923	3 226.0±141.4

range as seen in 2-VSE-C, 2-VSE-D, and 2-VSE-E If the size of the smallest range was the same, then further splitting of higher ranges appeared to make little difference as seen in 2-VSE-E, 3-VSE-A, and 4-VSE-A.

We will use these findings in the design of VSEs for original 2048.

5 Experiments for 2048

5.1 Design of N-tuple Networks and Vertical Split Encoding

Table 5 summarizes the design of N-tuple networks and Vertical Split Encoding used in our experiments for 2048.

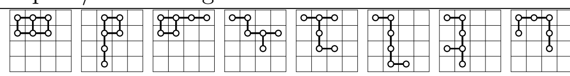
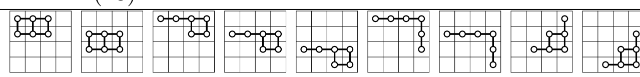
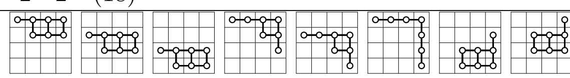
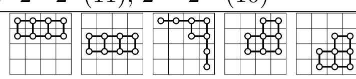
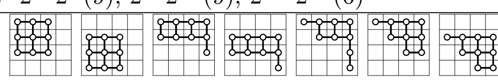
By adopting the heuristics of "manually creating reasonable shapes and generating their translations" [8], we designed new sets of N-tuple networks for 6-, 7-, 8-, and 9-tuple cases.

– NT6: We designed four shapes of 6-tuples and obtained nine tuples by their translations.
– NT7: We designed four shapes of 7-tuples and obtained eight tuples by their translations.
– NT8: We designed three shapes of 8-tuples and obtained five tuples by their translations.
– NT9: We designed four shapes of 9-tuples and obtained seven tuples by their translations.

As a baseline, we used NT6-M, a network consisting of 8×6-tuples, which was first selected through intensive experiments by Matsuzaki [13] and employed in the state-of-the-art player by Guei et al. [5].

For each set of N-tuples, we designed the VSE value ranges under the following assumptions: each element in lookup tables has 64 bits; we use temporal

Table 5. Design of N-tuple networks and Vertical Split Encoding for 2048. The number in parentheses after each value range shows the number of possible values after corresponding mapping.

name	tuples / value ranges	multi-staging / parameters
NT6-M		w/ 2 stages
w/o VSE	2^1–2^{17} (18)	544 195 584
NT6		w/ 2 stages
w/o VSE	2^1–2^{17} (18)	612 220 032
NT7		w/ 2 stages
w/ 2-VSE	2^1–2^9 (11), 2^{10}–2^{17} (10)	471 794 736
NT8		w/ 2 stages
w/ 3-VSE	2^1–2^7 (9), 2^8–2^{13} (9), 2^{14}–2^{17} (6)	877 730 580
NT9		w/ 2 stages
w/ 4-VSE	2^1–2^5 (7), 2^6–2^9 (7), 2^{10}–2^{13} (7), 2^{14}–2^{17} (6)	1 835 939 238

coherence learning [8] that requires three times as much memory with three lookup tables; and we use multi-staging technique with two stages [5] that requires two times as much memory; the whole training program should fit within 64 GB of memory. Our final design of VSEs was as follows.

- For NT6 and NT6-M, the parameters sufficiently fit within the memory without VSE, and so we do not apply VSE.
- For NT7, at least two splits are needed, and thus we use 2-VSE with two value ranges of sizes 11 and 10.
- For NT8, at least three splits are needed, and thus we use 3-VSE with three value ranges of sizes 9, 9, and 6.
- For NT9, at least four splits are needed, and thus we use 4-VSE with four value ranges of sizes 7, 7, 7, and 6.

Note that there exist a few more flexibility in the choice of value ranges of VSE. Exhaustively exploring those possibilities remains as future work.

5.2 Training and Evaluation with 1-ply Lookahead Play

For the N-tuple networks with VSE defined above, we conducted reinforcement learning to tune their parameters. In this study, we employed Temporal Coherence learning with Optimistic Initialization and Restart Strategy.

Temporal Coherence learning [8] Temporal Coherence learning (TC learning) is a variant of TD learning that enables automatic adjustment of the learning rate, which was first introduced to 2048 by Jaśkowski. To keep the effect of the following Optimistic Initialization, we decayed the learning rate from 0.5.

Optimistic Initialization [5] Optimistic Initialization (OI) is a method of encouraging exploration in the training by initializing the parameters with large values. In this study, we use the same initial values as Guei et al. [5]: we initialize the parameters such that all afterstates have values of 320 000.

Restart Strategy [12] The games of 2048 are easy at the beginning but hard when the board is filled with large-number tiles. To address this issue, we applied the restart strategy with constant number 10 of restarts.

Multi-staging [23] Following prior work [5], we also employed the multi-staging method, in which the lookup tables are switched depending on game progression. In this study, we split the game into two stages, switching the lookup tables before and after the generation of a 32 768 tile.

We introduced no additional exploration in the training apart from OI. To avoid convergence to local optima, we reset the parameters controlling the learning rate in TC learning every 50×10^9 steps.

In this study, for each of five types of N-tuple networks above, we conducted the training for 200×10^9 steps. During the training, we output the snapshot of the parameters every 1×10^9 steps and evaluate their performance in terms of the average score of 10 000 games with 1-ply lookahead (greedy) play. To mitigate the randomness, we conducted the training with five different random seeds, and report their means and standard deviations.

Figure 3 shows the training curves. The final average scores after the training with 200×10^9 steps were ranked as $\mathsf{NT8} > \mathsf{NT7} > \mathsf{NT9} > \mathsf{NT6} \approx \mathsf{NT6\text{-}M}$. The standard deviation for NT8 was very large because training with two seeds succeeded achieving average scores of around 400 000, while training with the other three saturated at around 320 000. Even considering these poor cases, NT8 still outperformed NT6 and NT6-M. For NT7 and NT9, the differences from NT6 and NT6-M were sufficiently larger than their standard deviations. These results indicated that increasing the N-tuple size with the help of VSE significantly improved performance.

A closer look at the training progress reveals that when the learning rates were reset in TC learning, networks with larger tuples showed smaller drops in score. This means that the training could be more stable with larger tuples but at the same time they tend to converge local optima. Another interesting observation was that NT7 was the only networks with a learning curve with a different shape. Possible reasons for that slower learning would come from its tuple design or the use of 2-VSE, but we have no clear evidences so far.

5.3 Evaluation with Expectimax Search

We also evaluated the final N-tuple networks in combination with Expectimax search (3-ply and 5-ply). The results are shown in Table 6.

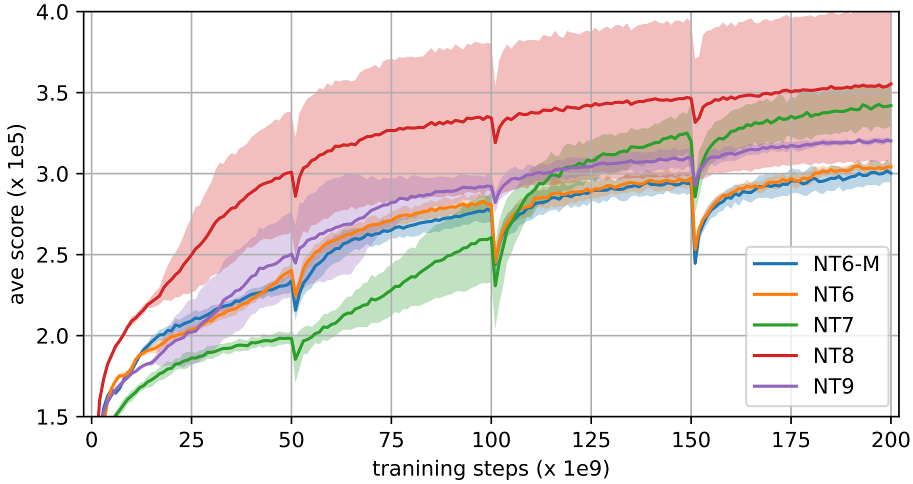


Fig. 3. Training curve evaluated with 10000 games of 1-ply lookahead (greedy) play. The shaded areas show the standard deviations over five training runs.

Table 6. Average scores and the ratios of reaching a 32 768-tile for the networks trained over 200×10^9 steps, evaluated with 1-ply lookahead (greedy) play and with Exxpectimax search with 3-ply and 5-ply lookahead. For each case, mean and standard derivation over five training runs (after ± sign) are given.

	1-ply (10 000 games)		3-ply (1 000 games)		5-ply (100 games)	
	ave. score	32 768[%]	ave. score	32 768[%]	ave. score	32 768[%]
NT6-M w/o VSE	300 241± 5 922	14.3± 1.7	462 677±15 573	44.3± 4.7	508 618± 15 010	55.0± 4.9
NT6 w/o VSE	304 149± 4 361	14.9± 0.4	481 544± 5 497	47.8± 0.4	533 801± 18 084	61.3± 5.0
NT7 w 2-VSE	342 009±12 427	19.1± 2.9	491 425±19 140	46.1± 5.1	550 848± 19 390	60.5± 5.1
NT8 w 3-VSE	355 420±47 102	15.5±15.6	438 702±91 280	29.1±29.1	448 579±102 127	31.5±32.0
NT9 w 4-VSE	320 205± 1 404	0.0± 0.0	360 286± 1 446	0.0± 0.0	363 160± 1 886	0.0± 0.0

Looking only at the average scores when combined with 3-ply or 5-ply Expectimax search, NT7 was the only network that outperformed NT6 and NT6-M. For NT9, the addition of search provided almost no improvement in performance. In fact, the generation rate of the 32 768 tile was 0.0% in all cases of NT9. This is likely due to overfitting: the lookup table contained too many parameters, causing the network to converge to local optima that fail to reach a 32 768 tile. For NT8, if training was successful (achieving around 400 000 points in 1-ply play), the combination of Expectimax search further improved performance. However, when training saturated, NT8 exhibited behavior similar to NT9, failing to reach a 32 768 tile. As a result, both the average score and the 32 768-tile reaching rate showed very large standard deviations.

5.4 Evaluation of Best N-tuple Networks

Since NT8, in particular, produced both successful and unsuccessful training outcomes, we conducted an additional evaluation using the best-performing networks for each N-tuple size, excluding NT9. For each selected network, we ran test plays (10 000 games with 1-ply lookahead, 1 000 games with 3-ply, and 100 games with 5-ply) with five different random seeds. The results are summarized in Table 7.

Table 7. Average scores and the ratios of reaching a 32 768-tile for the *best networks* chosen from ones trained over 200×10^9 steps, evaluated with 1-ply lookahead (greedy) play and with Exxpectimax search with 3-ply and 5-ply lookahead. For each case, mean and standard derivation over five *testplay* runs (after ± sign) are given.

	1-ply (10 000 games)		3-ply (1 000 games)		5-ply (100 games)	
	ave. score	32 768[%]	ave. score	32 768[%]	ave. score	32 768[%]
Best NT6-M w/o VSE	311 354±1 971	16.9±0.3	484 281±5 435	49.0±1.2	517 595±17 160	56.2±5.7
Best NT6 w/o VSE	300 410±1 122	14.4±0.4	484 718±6 182	48.1±1.0	532 599±28 947	58.4±8.5
Best NT7 w 2-VSE	341 636±1 719	19.0±0.2	484 328±6 187	44.2±1.9	541 210±22 938	56.6±5.8
Best NT8 w 3-VSE	407 206± 432	30.0±0.2	547 365±4 938	57.2±1.9	587 690±20 439	66.2±6.2

Using the best networks, the overall ranking was again NT8 > NT7 > NT6 ≈ NT6-M. For the case combined with 5-ply Expectimax search, the performance gap between NT8 and NT6-M was about 70 000, which was sufficiently larger than the standard deviation. These results lead us to conclude that combining VSE with larger tuple sizes is highly effective for 2048.

6 Conclusion

In this study, we have proposed a novel approach to designing powerful N-tuple networks for the game 2048. Increasing the tuple size of N-tuple networks is expected to enhance the players' performance. However, this comes with the severe drawback that the number of parameters and memory requirement grow exponentially with tuple size. As a result, on commodity computers, the practical upper limit on tuple size had been restricted to six or seven for the game 2048. To break through this limitation, we proposed a novel method called Vertical Split Encoding (VSE).

A preliminary experiment using Mini2048 confirmed that applying VSE does not cause substantial performance degradation. We then built and trained 2048 players using N-tuple networks of tuple size six (NT6 without VSE), seven (NT7 with 2-VSE), eight (NT8 with 3-VSE), and nine (NT9 with 4-VSE). The training results showed that all of NT7, NT8, and NT9 networks outperformed the baseline (NT6-M). In particular, the best player developed with NT8 and 3-VSE achieved a high score of 587 690 with 5-ply Expectimax search. Compared under the same conditions, this was about 70 000 points higher than that of NT6-M, the network employed in the state-of-the-art player, and the improvement was far exceeding the standard deviation. These findings demonstrate that VSE enables the design of N-tuple networks with larger tuple sizes and accordingly improvement of players.

Our future work includes the optimization of the design of large N-tuple networks with VSE, instead of manual heuristics and empirical choices. Our experiments suggested that there is room for optimization: for example, the slower learning curve observed for `NT7` with 2-VSE. Another important future work is to address the issue of performance plateau that was seen for NT9. The improvement of learning-rate control and the introduction of exploration strategies during the training will be an important next step. Our ultimate goal is to surpass the current state-of-the-art average score of 625 377 with these improvements.

Acknowledgments. This work was supported in part by JSPS KAKENHI Grant Number JP23K11383.

References

1. Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T.K., Silver, D.: Planning in stochastic environments with a learned model. In: Proceedings of International Conference on Learning Representations (ICLR) (2022).
2. Buro, M.: From simple features to sophisticated evaluation functions. In: Proceedings of First International Conference on Computers and Games, CG'98. Lecture Notes in Computer Science, vol. 1558, pp. 126–145, Springer (1998).
3. Chen, J.C., Hsu, T.Y., Hsu, C.M., Hsu, T.s.: Applying larger N-tuple networks to EinStein Würfelt Nicht! In: Proceedings of the 2023 6th International Conference on Computational Intelligence and Intelligent Systems (CIIS '23). pp. 165–172, ACM (2024).
4. Cirulli, G.: 2048. `https://play2048.co/` (2014).
5. Guei, H., Chen, L.P., Wu, I.C.: Optimistic temporal difference learning for 2048. IEEE Transactions on Games, **14**(3), 478–487 (2022).
6. Hsueh, W.L., Hsu, T.S.: An approximate solution of "Einstein würfelt nicht!". In: Proceedings of 2025 IEEE Conference on Games (CoG), pp. 1–7 (2025).
7. Huy, N.Q., Quoc, D.C., Phong, T.Q.: Selecting pattern shapes to build up an evaluation function for Connect-6 game. In: Proceedings of the 4th International Conference on Virtual Reality, pp. 50–55, ACM (2018).
8. Jáskowski, W.: Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding and carousel shaping. IEEE Transactions on Computational Intelligence and AI in Games, **10**(1), 3–14 (2018).
9. Jaśkowski, W.: Systematic N-tuple networks for othello position evaluation. Journal of the International Computer Games Association, **37**(2), 85–96 (2014).
10. Kao, C., Guei, H., Wu, T., Wu, I.: Gumbel muzero for the game of 2048. In: Proceedings of International Conference on Technologies and Applications of Artificial Intelligence (TAAI 2022), pp. 42–47, IEEE (2022).
11. Kuramitsu, H., Suzuki, K., Matsuzawa, T.: N-tuple network search in othello using genetic algorithms. Games, **16**(1) (2025).

12. Matsuzaki, K.: Developing 2048 player with backward temporal coherence learning and restart. In: Proceedings of Fifteenth International Conference on Advances in Computer Games (ACG2017). pp. 176–187 (2017)
13. Matsuzaki, K.: Systematic selection of n-tuple networks with consideration of interinfluence for game 2048. In: Proceedings of 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 186–193 (2016).
14. Matsuzaki, K.: Developing value networks for game 2048 with reinforcement learning. Journal of Information Processing, **29**, 336–346 (2021).
15. Nguyen, H., Ikeda, K., Le, B.: Extracting important patterns for building state-action evaluation function in Othello. In: Proceedings of 2012 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2012), pp. 278–283 (2012).
16. Oka, K., Matsuzaki, K.: Systematic selection of n-tuple networks for 2048. In: Proceedings of Computers and Games (CG 2016), Revised Selected Papers, Lecture Notes in Computer Science, vol. 10068, pp. 81–92. Springer (2016).
17. Szubert, M., Jaśkowski, W.: Temporal difference learning of N-tuple networks for the game 2048. In: Proceedings of 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–8 (2014).
18. Terauchi, S., Kubota, T., Matsuzaki, K.: Using strongly solved Mini2048 to analyze players with N-tuple networks. In: Proceedings of International Conference on Technologies and Applications of Artificial Intelligence (TAAI 2023) (2023).
19. Terauchi, S., Matsuzaki, K.: Relationship between the size of N-tuple network and learning performance: Experiments and evaluation using mini2048 (in Japanese). IPSJ SIG Technical Reports 2025-GI-55, pp. 1–9 (2025).
20. Thill, M., Koch, P., Konen, W.: Reinforcement learning with N-tuples on the game Connect-4. In: Proceedings of the 12th International Conference on Parallel Problem Solving from Nature — Volume Part I, pp. 184–194, Springer-Verlag (2012).
21. Wang, W., Matsuzaki, K.: Refining evaluation functions for game 2048 by extended temporal difference learning. IEEE Transactions on Games, pp. 1–11 (2025), available as an early access article.
22. Yamashita, S., Kaneko, T., Nakayashiki, T.: Strongly solving 2048 on 3 × 3 board and performance evaluation of reinforcement learning agents. In: Proceedings of the Game Programming Workshop 2022. pp. 1–8 (2022), in Japanese.
23. Yeh, K.H., Wu, I.C., Hsueh, C.H., Chang, C.C., Liang, C.C., Chiang, H.: Multistage temporal difference learning for 2048-like games. IEEE Transactions on Computational Intelligence and AI in Games, **9**(4), 369–380 (2016).
24. Zhou, Y.: From AlphaGo zero to 2048. Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD '19), Available on `http://www.cse.msu.edu/~zhaoxi35/DRL4KDD/2.pdf` (2019).