Multi-Stage Temporal Difference Learning for 2048

I-Chen Wu, Kun-Hao Yeh, Chao-Chin Liang, Chia-Chuan Chang, Han Chiang

Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
{icwu,khyeh,chaochin,jimmy4834,passerby1023}@aigames.nctu.edu.tw

Abstract. Recently, Szubert and Jaskowski successfully used TD learning together with n-tuple networks for playing the game 2048. In this paper, we first improve their result by modifying the n-tuple networks. However, we observe a phenomenon that the programs based on TD learning still hardly reach large tiles, such as 32768-tiles (the tiles with value 32768). In this paper, we propose a new learning method, named multi-stage TD learning, to effectively improve the performance, especially for maximum scores and the reaching ratio of 32768-tiles. After incorporating shallow expectimax search, our 2048 program can reach 32768-tiles with probability 10.9%, and obtain the maximum score 605752 and the averaged score 328946. To the best of our knowledge, our program outperforms all the known 2048 programs up to date, except for the program developed by the programmers, nicknamed nneonneo and xificurk, which heavily relies on deep search heuristics tuned manually. The program can reach 32768-tiles with probability 32%, but ours runs about 100 times faster. Also interestingly, our new learning method can be easily applied to other 2048-like games, such as Threes. Our program for Threes outperforms all the known Threes programs up to date.

Keywords: Stochastic Puzzle Game, 2048, Threes!, Temporal Difference Learning, Expectimax.

1 Introduction

The puzzle game 2048 [7], a single-player stochastic game originated from 1024 [5] and Threes [6], has recently become very popular over the Internet. The author Gabriele Cirulli [11] claimed his estimation: the aggregated time of playing the game online by players during the first three weeks after released was over 3000 years. The game is intriguing and even addictive to human players, because the rule is simple but hard to win. Note that players win when reaching 2048-tiles, the tiles with the value 2048. For the same reason, the game also attracted many programmers to develop artificial intelligence (AI) programs to play it. In [17], the authors also thought that the game is an interesting testbed for studying AI methods.

Many methods was proposed to design AI programs in the past. Most commonly used methods were alpha-beta search [8][12], a traditional game search method for two-player games, and *expectimax search* [1], a common game search method for single-player stochastic games. Recently, Szubert and Jaskowski [17] proposed

S.-M. Cheng and M.-Y. Day (Eds.): TAAI 2014, LNAI 8916, pp. 366–378, 2014.

Temporal Difference (*TD*) *learning* together with *n-tuple networks* for the game. They successfully used it to reach a win rate (the ratio of reaching 2048-tiles) 97%, and obtain the averaged score 100,178 with maximum score 261,526. However, we observe a phenomenon: the TD learning method tends to maximize the average scores, but becomes less motivated to reach large tiles, such as 16384 or 32768.

In this paper, we first improve their result by modifying the n-tuple networks. However, we observe a phenomenon that the programs based on TD learning (together with n-tuple network still hardly reach 32768-tiles (the tiles with value 32768), even with the help of expectimax search.

Furthermore, we propose a new technique, named *multi-stage TD* (*MS-TD*) *learning*, to help reach large tiles more easily. In this learning method, we separate the training into multiple stages similar to those used in [4]. After incorporating shallow expectimax search, our 2048 program can reach 32768-tiles with probability 10.9%, and obtain the maximum score 605752, and the averaged score 328946.

To the best of our knowledge, our program outperforms all the known 2048 programs up to date, except for the one in [10], which heavily relies on deep search heuristics tuned manually. The program can reach 32768-tiles with probability 32%, but ours runs 300-1000 moves/second, about 100 times faster than theirs, 3-4 moves/second.

More interesting, our new learning method can be easily applied to other 2048-like games, such as Threes. Our program for Threes can reach 6144-tiles with probability 10%, which outperforms all the known Threes programs up to date and also won the champion in the 2048-bot tournament [18]. However, due to the limit of paper size, the research on Threes is omitted in this paper.

2 Background

This section reviews some backgrounds for this paper. First, introduce the rules of 2048. Second, we briefly review game tree search. Third, review TD learning, and discuss three different evaluation methods and n-tuple networks for 2048 proposed in [17].

2.1 Rules of 2048

The game 2048 is a single-player game that can be played on web pages and mobile devices with a 4x4 board, where each cell is either empty or placed with a tile labeled with a value which is a power of two. Let v-tile denote the tile with a value v. Initially, two tiles, 2- or 4-tiles, are placed on the board at random. In each turn, the player makes a move and then the game generates a new 2-tile with a probability of 9/10 or 4-tile with a probability of 1/10 on an empty cell chosen at random.

To make a move, the player chooses one of the four directions, up, down, left and right. Upon choosing a direction, all the tiles move in that direction as far as they can until they meet the border or there is already a different tile next to it. When sliding a tile, say v-tile, if the tile next to it is also a v-tile, then the two tiles will be merged into a larger tile, $2v$-tile. At the same time, the player gains $2v$ more points in the score. A move is legal if at least one tile can be moved.

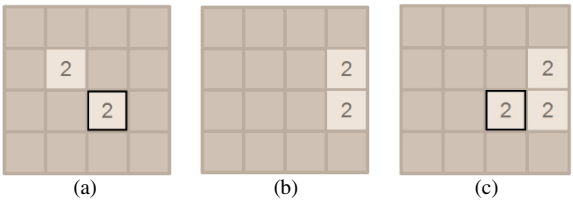


Fig. 1. Examples of 2048 boards

Consider an example, in which an initial board is shown in Fig. 1 (a). After making a move to right, the board becomes the one shown in Fig. 1 (b). Then, a new 2-tile is randomly generated as shown in Fig. 1 (c). The player can repeatedly make moves in this way.

A game ends when the player cannot make any legal move. The final score is the points accumulated during the game. The objective of the game is to accumulate as many points as possible. The game claims that the player wins when a 2048-tile is created, but still allow players to continue playing.

In a 2048-bot tournament held in [18], all the 2048-bot participants play 100 games. Their performances are graded by four factors, the win rates, the *largest tiles*, denoted by v_{max}-tiles, plus the reaching ratios of v_{max}-tiles, the average scores, and the maximum scores, in a formula described in [18].

2.2 Game Tree Search

A common game tree search algorithm used for 2048 bots is expectimax search [1]. Like most game tree search, the leaves are evaluated with values calculated by heuristic functions. An expectimax search tree contains two different nodes, max nodes and chance nodes. At a max node, its value is the highest value of its children nodes, if any. At a chance node, its value is the expected value of its children nodes, if any, each with a probability of instances.

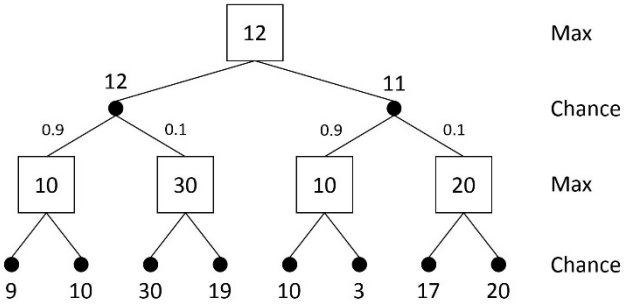


Fig. 2. An expectimax search tree

Consider the example shown in Fig. 2. For the left chance node of the root, its value is derived by calculating the expected value of its two children nodes: 0.9 * 10 + 0.1 * 30 = 12. For the right chance node of the root, its value is 11 which can be calculated in the same way. Thus, the root will choose the left and its value is 12.

Several features were used in heuristic functions for 2048-bot programs [10][21], and three of them are listed as follows. The first is the monotonicity of a board. Most high-ranked players might tend to play the game 2048 with tiles arranged decreasingly such as those described in [15]. The second is the number of empty tiles on board. The more empty tiles, the less likely for the game to end in a few steps. The third is the number of mergeable tiles, since it is a measure of the ability to create empty tiles by merging tiles of same values.

For game tree search, transposition table is a very important technique for speed-up. One common implementation is based on Z-hashing [23]. In Z-hashing, for each position, each possible tile is associated with a unique random number as a key. When looking up table, say 2048, the hash value is calculated by making an exclusive-or operation on 16 keys.

2.3 Temporal Difference (TD) Learning

Reinforcement learning is an important technique in training an agent to learn how to respond to the given environment [16]. *Markov decision process* (*MDP*) is a model commonly used in reinforcement learning, modeling the problems in which an agent interacts with the given environment through a sequence of actions according to the change of the state and the rewards, if any. In terms of MDP, an AI program for 2048-like game thus can be regarded as such an agent, which makes actions (legal moves) on board states and gets points as rewards.

Temporal difference (*TD*) *learning* [16][22], a kind of reinforcement learning, is a model-free method for adjusting state values from the subsequent evaluations. This method has been applied to computer games such as Backgammon [19], Checkers [13], Chess [2], Shogi [3], Go [14], Connect6 [22] and Chinese Chess [20]. TD learning has been demonstrated to improve world class game-playing programs in the two cases, TD-Gammon [19] and Chinook [14]. Since 2048-like games can be easily modeled as MDP, TD learning can be naturally applied to AI programs for 2048-like games.

In TD(0), the value function $V(\mathrm{s})$ is used to approximate the expected return of a state s. The error between states s_t and s_{t+1} is $\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$, where r_{t+1} is the reward at turn $t+1$. The value of $V(s_t)$ in TD(0) is expected to be adjusted by the following value difference $\Delta V(s_t)$,

$$\Delta V(s_t) = \alpha\delta_t = \alpha\big(r_{t+1} + V(s_{t+1}) - V(s_t)\big) \tag{1}$$

where α is a step-size parameter to control the learning rate. For general TD(λ) (also see [22]), the value difference is

$$\Delta V(s_t) = \alpha\left((1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}V(s_{t+n}) + \lambda^{T-t-1}V(s_T) - V(s_t)\right). \tag{2}$$

In this paper, only TD(0) is investigated.

In most applications, the evaluation function of states $V(s)$ can be viewed as a function of features, such as the monotonicity, the number of empty tiles, the number of mergeable tiles [10], mentioned in Subsection 2.2. Although the function was actually very complicated, it is usually modified into a linear combination of features [22] for TD learning, that is, $V(s) = \varphi(s) \cdot \theta$, where $\varphi(s)$ denotes a vector of feature occurrences in s, and θ denotes a vector of feature weights.

In order to correct the value $V(s_t)$ by the difference $\Delta V(s_t)$, we can adjust the feature weights θ by a difference $\Delta\theta$ based on $\nabla_\theta V(s_t)$, which is $\varphi(s_t)$ for linear TD(0) learning. Thus, the difference $\Delta\theta$ is

$$\Delta\theta = \Delta V(s_t)\varphi(s_t) = \alpha\delta_t\varphi(s_t) \tag{3}$$

TD Learning for 2048 In [17], Szubert and Jaskowaski proposed TD learning for 2048. A *transition* from turn t to $t+1$ is illustrated in Fig. 3 (below). They also proposed three kinds of methods of evaluating values for training and learning as follows.

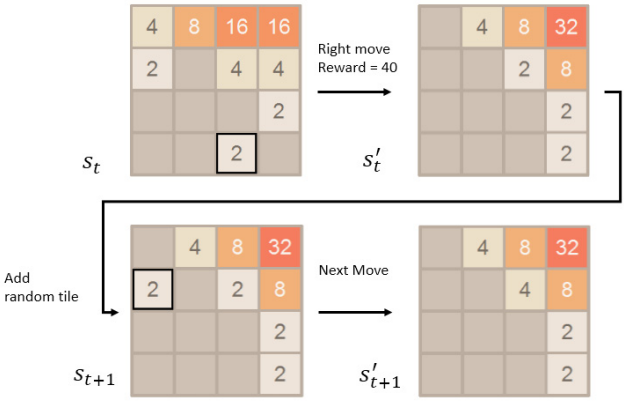


Fig. 3. Transition of board states

1. Evaluate actions. This method is to evaluate the function $Q(s,a)$, which stands for the expected values of taking an action a on a state s. For 2048, an action a is one of the four directions, up, down, left, and right. This is so-called *Q-learning*. In this case, the agent chooses a move with the highest expected score, as the following formula.

$$arg_a \max Q(s,a) \tag{4}$$

2. Evaluate states to play. This method is to evaluate the value function $V(s_t)$ on state s_t, the player to move. As shown in Fig. 3, this method evaluates $V(s_t)$ and $V(s_{t+1})$.The agent chooses a move with the highest expected score on s_t, as the following formula.

$$arg_a \max\left(R(s_t,a) + \sum_{s_{t+1}} P(s_t,a,s_{t+1})V(s_{t+1})\right) \tag{5}$$

where $R(s_t, a)$ is the reward for action a on state s_t, and $P(s_t, a, s_{t+1})$ is the probability of turning into state s_{t+1} after taking action a on state s_t.

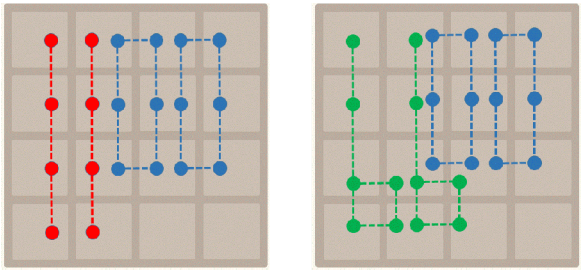
3. Evaluate states after an action. This method is to evaluate the value function $V(s_t')$ on state s_t', a state after an action, also called *after-states* in [17]. As shown in Fig. 3, this method evaluates $V(s_t')$ and $V(s_{t+1}')$.The agent chooses a move with the highest expected score on s_t', as the following formula.

$$arg_a max[R(s_t, a) + V(s_t')]. \tag{6}$$

In [17], their experiments showed that the third method clearly outperformed the other two. Therefore, in this paper, we only consider this method, evaluating after-states. For simplicity, states refers to after-states in the rest of this paper.

2.4 N-Tuple Network

In [17], they also proposed to use *N-tuple network* for TD learning of 2048. An n-tuple network consists of m n_i-tuples, where n_i is the size of the i-th tuple. As shown in Fig. 4, one 4-tuple covers four cells marked in red rectangular and one 6-tuple covers six cell marked in blue rectangular. Each tuple contributes a large number of features, each for one distinct occurrence of tiles on the covered cells. For example, the leftmost 4-tuple in Fig. 4 (a) includes 16^4 features, assuming that a cell has 16 occurrences, empty or 2-tile to 2^{15}-tile.



(a) (b)

Fig. 4. (a) Tuples used in [17] and (b) tuples used in this paper

Based on TD learning, the expected score $V(s)$ is a linear summation of feature weights for all occurred features. For each tuple, since there is one and only one feature occurrence, we can simply use a lookup table to locate the feature weight. Thus, if the n-tuple network includes m different tuples, we need m lookups.

In [17], they used the tuples shown in Fig. 4 (a) as well as all of their rotated and mirrored tuples. All the rotated and mirrored tuples can share the same feature weights. Thus, the total number of features is roughly $2x2^{16}+2x2^{24}$, about 32 million.

Their experiments showed that the tuples shown in Fig. 4 (a) outperformed all other n-tuple networks used in their experiments. The experimental results claimed in [17] include an average score of 100178 and a maximum score of 261526.

3 Our TD Learning Method for 2048

This paper designs our TD learning method for 2048 based on the method in [17]. In our method, we first change the n-tuple network in Subsection 3.1, and then propose MS-TD learning in Subsection 3.2. The experiments for MS-TD learning are described in Subsection 3.4 and some issues are discussed in Subsection 3.5.

3.1 New N-Tuple Network

In this paper, we use the tuples shown in Fig. 4 (b) as well as all of their rotated and mirrored tuples. In brief, we change 4-tuples (1x4 lines) to 6-tuples in a green knife shape as shown in Fig. 4 (a). Apparently, the new 6-tuples cover all the original 4-tuples. The number of features increases only by a factor of two or so.

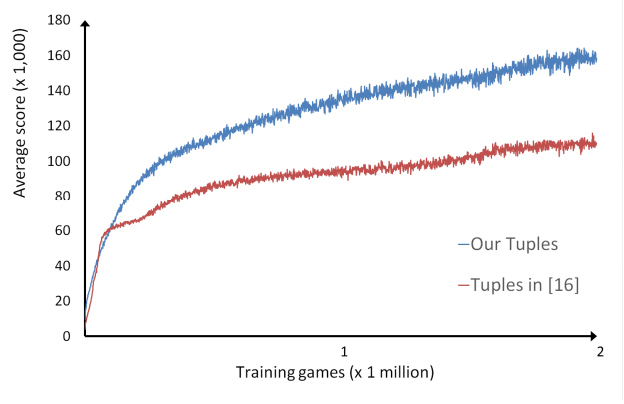


Fig. 5. Average scores in TD learning with different n-tuple networks

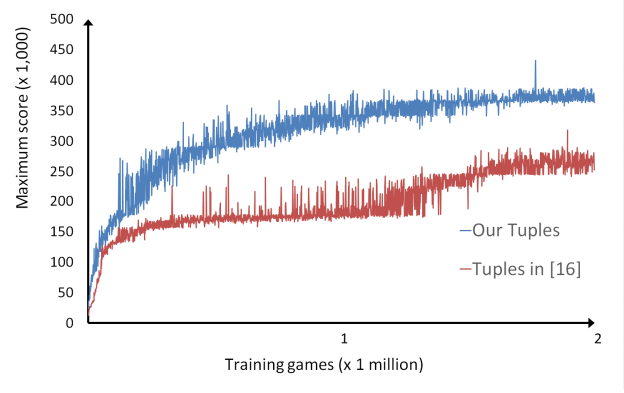


Fig. 6. Maximum scores in TD learning with different n-tuple networks

In our experiments, we further use a set of features representing large tiles, namely v-tiles, where $v \geq 2048$. These features are used to indicate difficulty due to large tiles. Our n-tuple network apparently outperforms the one used in [17] in terms of both average and maximum scores, as shown in Fig. 5 and Fig. 6 respectively. In these figures, the number of training games is up to 2 million and average/maximum scores in y axis are sampled every 1000 games. For simplicity of analysis, we use the n-tuple network in the rest of this paper.

3.2 MS-TD Learning

From above, TD learning is intrinsically to train and learn to obtain as high average (or expected) scores as possible, and the experiments also demonstrate this. However, TD learning does not necessarily lead to other criteria such as high maximum scores, high v_{max}, and the reaching ratios of v_{max}, though it does often.

From the experiences for 2048, we observed that it is hard to push to 32768-tiles or raise the reaching ratios of 32768-tiles from Fig. 6. However, for most players, earning the maximum scores and the largest tiles is a kind of achievement to players, and was also one of the criteria of the 2048-bot tournament [18].

In order to solve this issue, we propose *MS-TD learning* for 2048-like games. In this method, we divide the learning process into multiple stages. The technique of using multiple stages has been used to evaluate game states in [4] for Othello.

In our experiment, we divided the process into three stages with two important splitting times, marked as T_{16k} and T_{16+8k}, in games. T_{16k} denotes the first time when a 16384-tile is created on a board in a game, and T_{16+8k} denotes the first time when both 16384-tile and 8192-tile are created. The learning process with three stages is described as follows.

1. In the first stage, use TD learning to train the feature weights starting from the beginning, until the value function (expected score) is saturated. At the same time, collect all the boards at T_{16k} in the training games. Now, the set of trained feature weights are saved and called the *Stage-1 feature weights*.
2. In the second stage, use TD learning to train another set of feature weights starting from these collected boards in the first stage. At the same time, collect all the boards at T_{16+8k} in the training games. Again, the set of trained feature weights are saved and called the *Stage-2 feature weights*.
3. In the third stage, use TD learning to train another set of feature weights starting from these collected boards in the second stage. Again, the set of trained feature weights are saved and called the *Stage-3 feature weights*.

When playing games, we also divide it into three stages in the following way.

1. Before T_{16k}, use the Stage-1 feature weights to play.
2. After T_{16k} and before T_{16+8k}, use the Stage-2 feature weights to play.
3. After T_{16+8k}, use the Stage-3 feature weights to play.

The idea behind using more stages is to let learning be more accurate for all actions during the second or third stage, based on the following observation. The trained feature weights learned from the first stage (the same as the original TD learning) tend

to perform well in the first stage, but may not in the rest of stages for the following reason. More large tiles in these stages increase the difficulty of playing the game, and therefore the feature weights cannot accurately reflect the expected scores with the difficulty. Thus, using another set of feature weights in the next stage makes it more likely for the feature weights to reflect the expected scores. In next subsection, the observation is justified in the experiments with significant improvements for 2048.

3.3 Experiments for MS-TD Learning

In the experiment for MS-TD learning, 5 million training games was run in each stage, and average and maximum scores are sampled every 1000 games.

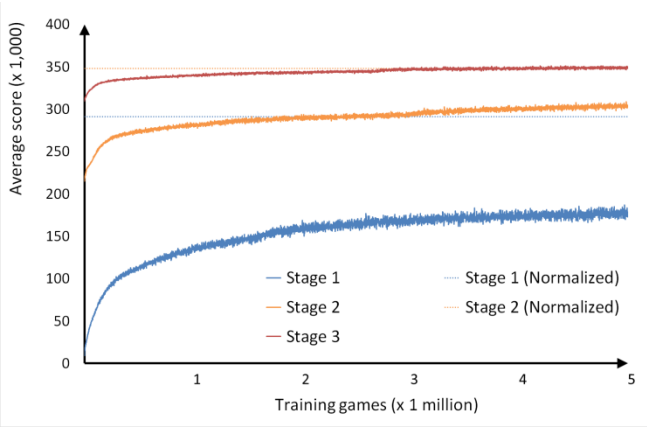


Fig. 7. Average scores in MS-TD learning

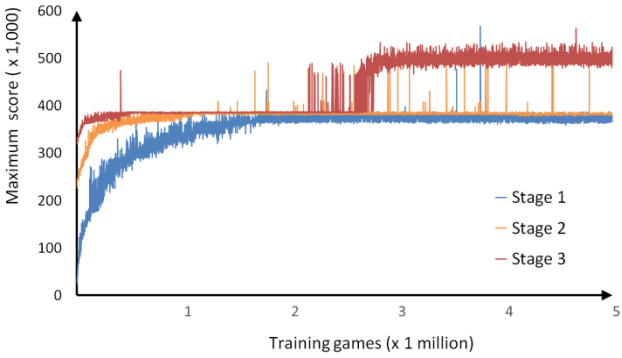


Fig. 8. Maximum scores in MS-TD learning

Fig. 7 shows the learning curves of average scores in the three stages. The learning curve of average scores in the first stage is marked in blue. However, the learning curves in the second and third stages need to be normalized for the purpose of comparisons, explained as follows. In the first stage, all the average scores includes those accumulated points before T_{16k}. In the second stage, if the average scores do not include those points before T_{16k}, then it is unfair for the second stage. However, if the average scores include those points before T_{16k}, it is unfair for the first stage since the average scores in the first stage include those failing to reach T_{16k} (the 16384-tile). For fair comparison, normalize the average scores in the first stage to a fixed value which is the average score of all the boards reaching T_{16k}. In addition, let the average scores in the second stage include those points before T_{16k}. Similarly, we also normalize average scores for the third stage in the same way.

In Fig. 7, the learning curve in the second stage, marked in orange, grows higher than the normalized of the first stage. This shows that the learning in the second stage does slightly improve over the first stage in terms of average scores. The learning curve in the third stage, marked in red, is nearly the same as the normalized of the second stage.

Fig. 8 shows the curves of maximum scores in the three stages. In this figure, the curve for the third stage does go up to 500,000 often, which actually indicate to reach 32,768-tiles. In contrast, the curves for the first and second stages rarely reach 32,768-tiles. This demonstrates that maximum scores can be significantly improved by using MS-TD learning.

3.4 Expectimax Search

Expectimax search fits after-states evaluation well. As described in Subsection 2.2, max nodes are the states, where players to move, and chance nodes are the after-states, where new tiles to be generated after moves. For TD learning, the learned after-state values can be used as the heuristic values of leaves. Thus, choosing the maximum after-state values can be viewed as expectimax search with depth one. Fig. 2 shows an example of expectimax search with depth 3. So, the depth for our expectimax search is an odd number. Then, we compare the performance results as follows.

Table 1. Result of 1000 games for TD learning

\Depth Reaching ratio\	1	3	5	7
2048	97.1%	99.9%	100.0%	100.0%
4096	88.9%	99.8%	100.0%	100.0%
8192	67.3%	96.9%	98.9%	98.5%
16384	18.1%	73.5%	80.7%	80.2%
32768	0.0%	0.0%	0.0%	0.0%
Maximum score	375464	385376	382912	382760
Average score	142727	282827	304631	302288

Table 2. Result of 1000 games for MS-TD learning

\Depth Reaching ratio\	1	3	5	7
2048	97.1%	99.9%	100.0%	100.0%
4096	88.9%	99.8%	100.0%	100.0%
8192	67.3%	96.9%	98.9%	98.5%
16384	18.1%	73.5%	80.7%	80.2%
32768	0.1%	9.4%	10.9%	4.6%
Maximum score	447456	536008	605752	581416
Average score	143473	310242	328946	313776

Table 1 shows the performance results of running 1000 games for the original TD learning in depths 1, 3, 5 and 7, respectively, while Table 2 shows those for MS-TD learning with three stages. Note that all the reaching ratios of all 16384-tiles and smaller tiles are the same, since both uses the same feature weights during the first stage. Besides, the comparisons between the two learning methods in the maximum scores and the average scores are shown in Fig. 9 and Fig. 10, respectively.

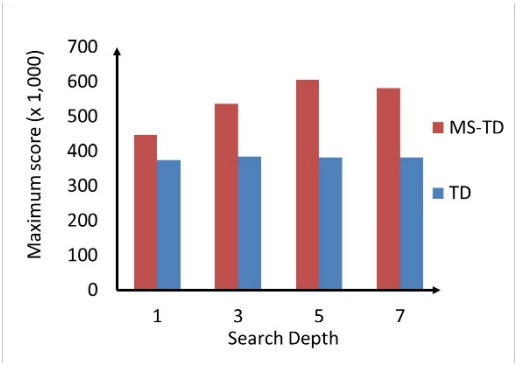


Fig. 9. Comparison of maximum scores

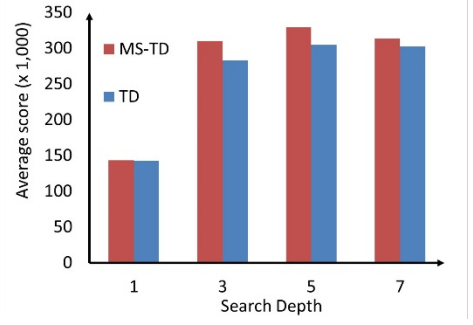


Fig. 10. Comparison of average scores

First, the results shows that the performance for MS-TD learning clearly outperforms that for the original TD learning. Especially, MS-TD learning significantly improves the maximum scores and the achieved ratios of 32768-tiles from 0% to 10.9%.

Second, the results also shows that the expectimax search in depth 5 performs the best, even better than the one with depth 7. The result shows that higher depths larger than 5 do not help much. Our observation is that the trained feature weights include noise which makes it less effective for deeper search.

From above, our 2048 program can reach 32768-tiles with probability 10.9%, the maximum score 605752, and the averaged score 328946. The program outperforms all the known 2048 programs up to date, except for the program in [10], which heavily relies on deep search heuristics tuned manually. Their program can reach 32768-tiles with probability 32%, but ours runs about 100 times faster.

3.5 Discussion

For MS-TD learning, an issue to discuss is what are the optimal number of stages and the splitting times. We did try several different kinds of stages by adding more splitting times. For example, add T_{8k}, the first time when a 8192-tile is created, and $T_{16+8+4k}$, the first time when all of 16384-tile, 8192-tile and 4096-tile are created. However, our experiments showed no better performance than the original three stages.

4 Conclusion

This paper proposes a new learning method, MS-TD (multi-stage TD) learning, which improves the performance effectively, especially for maximum scores and the reaching ratios of 32768-tiles. In our experiments, our 2048 program can reach 32768-tiles with probability 10.9%, the maximum score 605752, and the average score 328946. The program outperforms all the known 2048 programs up to date, except for the one in [10], which heavily relies on deep search heuristics tuned manually and runs about 100 times slower than ours.

Interestingly, MS-TD learning can be easily applied to other 2048-like games, such as Threes. Our program for Threes can reach 6144-tiles with probability 10%, which outperforms all the known Threes programs up to date and also won the champion in the 2048-bot tournament [18]. Furthermore, it is interesting and still open whether the method can be applied to other kinds of applications, such as non-deterministic two-player games, or even the planning applications, which can be separated into several stages.

Acknowledgement. The authors would like to thank Ministry of Science and Technology of the Republic of China (Taiwan) for financial support of this research under the contract numbers NSC 102-2221-E-009-069-MY2 and 102-2221-E-009-080-MY2, and the National Center for High-performance Computing (NCHC) for computer time and facilities.

References

[1] Ballard, B.W.: The *-Minimax Search Procedure for Trees Containing Chance Nodes. Artificial Intelligence 21, 327–350 (1983)

[2] Baxter, J., Tridgell, A., Weaver, L.: Learning to Play Chess Using Temporal Differences. Machine Learning 40(3), 243–263 (2000)

[3] Beal, D.F., Smith, M.C.: First Results from Using Temporal Difference Learning in Shogi. In: van den Herik, H.J., Iida, H. (eds.) CG 1998. LNCS, vol. 1558, pp. 113–125. Springer, Heidelberg (1999)

[4] Buro, M.: Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello. Games in AI Research, 77–96 (1997)

[5] Game 1024, http://1024game.org/

[6] Game Threes!, http://asherv.com/threes/

[7] Game 2048, http://gabrielecirulli.github.io/2048/

[8] Knuth, D.E., Moore, R.W.: An analysis of alpha-beta pruning. Artificial Intelligence 6, 293–326 (1975)

[9] Melko, E., Nagy, B.: Optimal Strategy in games with chance nodes. Acta Cybernetica 18(2), 171–192 (2007)

[10] Nneonneo and xificurk (nicknames), Improved algorithm reaching 32k tile, https://github.com/nneonneo/2048-ai/pull/27

[11] Overlan, M.: 2048 AI, http://ov3y.github.io/2048-AI/

[12] Pearl, J.: The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communications of ACM 25(8), 559–564 (1982)

[13] Schaeffer, J., Hlynka, M., Jussila, V.: Temporal Difference Learning Applied to a High-Performance Game-Playing Program. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence, pp. 529–534 (August 2001)

[14] Silver, D.: Reinforcement Learning and Simulation-Based Search in Computer Go, Ph.D. Dissertation, Dept. Comput. Sci., Univ. Alberta, Edmonton, AB, Canada (2009)

[15] StackOverflow.: What is the optimal algorithm for the game, 2048?, http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048/22674149#22674149

[16] Sutton, R.S., Barto, A.G.: Temporal-Difference Learning, An Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)

[17] Szubert, M., Jaskowaski, W.: Temporal Difference Learning of N-tuple Networks for the Game 2048. In: IEEE CIG 2014 Conference (August 2014)

[18] Taiwan 2048-bot, http://2048-botcontest.twbbs.org/

[19] Tesauro, G.: TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. Neural Computation 6, 215–219 (1994)

[20] Trinh, T., Bashi, A., Deshpande, N.: Temporal Difference Learning in Chinese Chess. In: Tasks and Methods in Applied Artificial Intelligence, pp. 612–618 (1998)

[21] Wu, K.C.: 2048-c, https://github.com/kcwu/2048-c/

[22] Wu, I.-C., Tsai, H.-T., Lin, H.-H., Lin, Y.-S., Chang, C.-M., Lin, P.-H.: Temporal Difference Learning for Connect6. In: van den Herik, H.J., Plaat, A. (eds.) ACG 2011. LNCS, vol. 7168, pp. 121–133. Springer, Heidelberg (2012)

[23] Zobrist, A.L.: A New Hashing Method With Application For Game Playing. Technical Report #88 (April 1970)