An Early Attempt at Applying Deep Reinforcement Learning to the Game 2048

Hong Gui, Tinghan Wei, Ching-Bo Huang, I-Chen Wu[1]

[1]Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
tinghan.wei@gmail.com,icwu@aigames.nctu.edu.tw

Abstract. We adapted deep neural networks with reinforcement learning methods to the single-player puzzle game 2048. The deep neural network consists of two to three hidden layers, and a convolutional filter of sizes 2×2 or 3×3. The reinforcement learning component uses either temporal difference learning or Q-learning. A total of 100,000 and 10,000 games were used as the training sample sizes for the two filter sizes. The trained networks were able to perform better than the random and greedy baseline players, but are far from the level of play of players that use only reinforcement learning. We think a redesign of the network may be needed to take into account the reflectional and rotational symmetries involved in 2048 for a higher level of play in the future.

Keywords: Deep learning, reinforcement learning, puzzle games

1 Introduction

2048 is a non-deterministic single-player puzzle game that has been massively popular since it was released in 2014. The game involves sliding number tiles in one of four directions so that identical-valued tiles may combine to create a new tile with twice the value of its components. The goal of the game is to reach the 2048 tile, but it is possible to continue playing past the goal and reach higher tile values, say 32768. The non-deterministic component of the game occurs after each player action, where a new tile of random value and position is added to the board.

There have been several efforts at creating computer programs that play 2048 at a high skill level. Early efforts on the Internet focused on traditional tree search and relied on manually-defined evaluation functions. Szubert and Jaśkowski followed up by using temporal difference learning (TDL) and n-tuple networks to achieve a program that is able to win over 97% of the time [1]. Wu et al. extended this work by partitioning the game into stages, and performing TDL on the stages separately, and is able to achieve a 32768 tile in 10.9% of the games [2].

Meanwhile, deep learning has been demonstrated to work well when applied to game-playing programs. Mnih et al. first presented a learning model based on convolutional neural networks and Q-learning, which was then applied to seven different Atari 2600 games [3]. In their work, the input of the application consists of graphical

information, namely, the pixels on screen, which is similar to what a human player is able to see when playing these games. The method is able to outperform humans in many games, and play at a comparative level for one game. There have been several attempts to adapt neural networks to 2048 [4][5]. Both attempts that were cited in this paper were not able to achieve the 2048 tile reliably.

In this paper, we present our work in applying deep learning to 2048. Our method follows the efforts from using deep Q-learning to the Atari 2600 games [3]. The experiment results show that the training has some effect wherein it outperforms two baselines (random play and a greedy method without search), but has significant room for improvement.

The paper is organized as follows. Section 2 will describe the method used for applying deep learning to 2048. In section 3 we provide the experiment results, followed by a short discussion on why the method does not work as well as previous efforts (such as TDL). Concluding remarks including potential research directions are given in Section 4.

2 Method

We follow the Deep Q-Network method from Mnih et al. [3], with two variations depending on filter size (2×2 and 3×3). The network with 2×2 filters is shown in Figure 1 below. It has two hidden layers for convolution, each followed by a rectified linear unit (ReLU). After the convolution layers, there is a fully connected hidden layer and its ReLU. Finally, the fully connected output layer is designed with four possible move directions as the output. For the network that convolves with 3×3 filters, an additional convolutional hidden layer is added (for a total of three convolutional hidden layers).

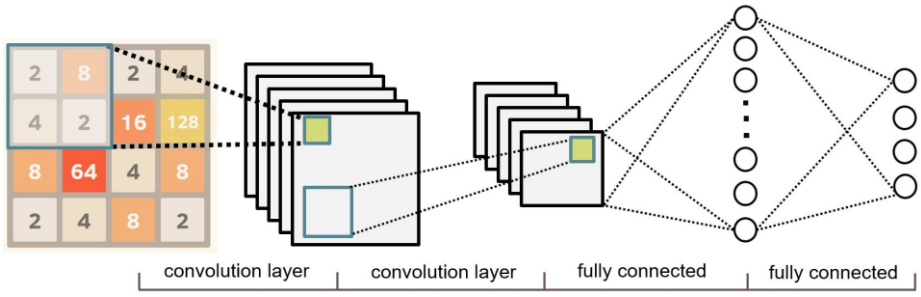

Fig. 1. Applying deep neural networks to 2048 with a 2×2 convolutional filter.

The input board position is encoded into 16 channels, where each channel is a 4×4 binary image; the ith channel marks each grid of the game position that contains 2^i as 1, and 0 otherwise. The game does not contain 1-tiles, so we represent 0-tiles in the first channel. For each hidden layer, we convolve either with a 2×2 or a 3×3 filter. In the 3×3 case, the input image is zero-padded so that the convolution output (input for the

s

next hidden layer) is a 4×4 image; the 2×2 case is convolved as is without zero-padding. In either case, the stride is 1.

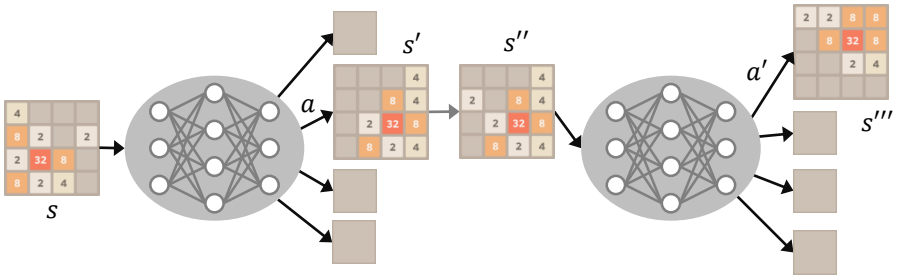

Fig. 2. Using Q-learning neural networks to play 2048.

We used both Q-learning and TD(0) as the reinforcement learning component of our network updates. In a game of 2048, such as the example illustrated in Figure 2, the player chooses an action a for the game state s, where the reward r is the score gained by playing a. The game state after the action is s'. A random tile is added to an empty grid, which leads to the game state s''. In this case, the states s and s'' are referred to as *beforestates*, and s' is referred to as an *afterstate* [1]. For Q-learning, we update the evaluation of performing the action a for a beforestate s using the following equation.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \max_{a' \in A(s'')} Q(s'',a') - Q(s,a)) \quad (1)$$

To do this, the program plays through a game of 2048, and records the tuple (s, a, r, s'') for each action performed. Once the game ends, equation (1) is used for each tuple to train the network.

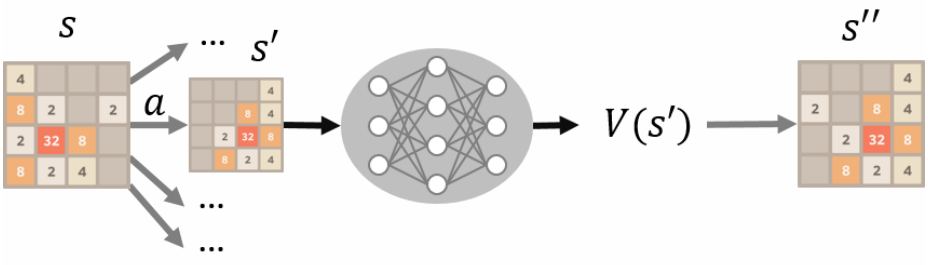

Fig. 3. Using TD(0) neural networks to play 2048. The move a is chosen because $V(s')$ is the largest of all possible moves from s.

For TD(0), the case is illustrated in Figure 3. We update the evaluation of any given afterstate s' using the following equation.

$$V(s) \leftarrow V(s) + \alpha\big(r + V(s'') - V(s)\big) \tag{2}$$

The training process is similar to Q-learning, except the tuple recorded is $(s',\ r, s''')$. The neural network output for the TD(0) case differs from the Q-learning case, since it only has a single value for V(s), which is the evaluated score for state s.

3 Experiments

For the network gradient descent algorithm, we used the AdaDelta method, with a min-ibatch size of 32. Two sets of parameters were set for the experiments, the size of the filters (and its corresponding network), and the type of reinforcement learning. The experiments were performed on a machine with the Intel Xeon E5-2620 v3 and four NVidia 980Ti graphics cards. The neural networks were implemented using the Caffe library.

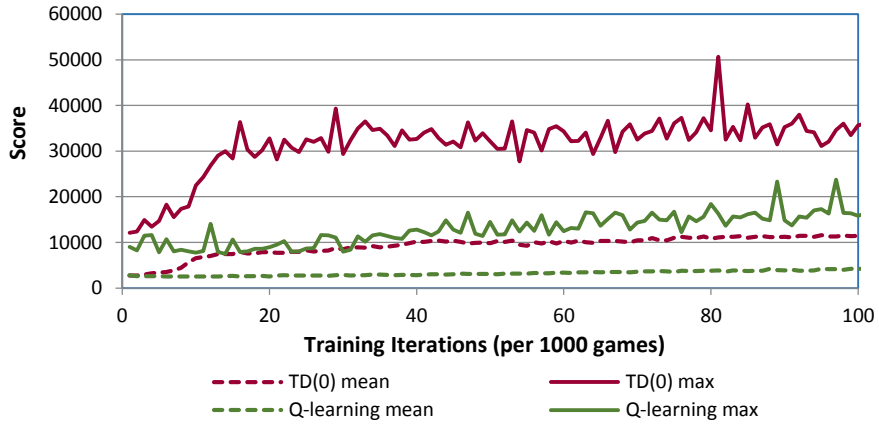

Fig. 4. Deep learning applied to 2048, using 2×2 filters.

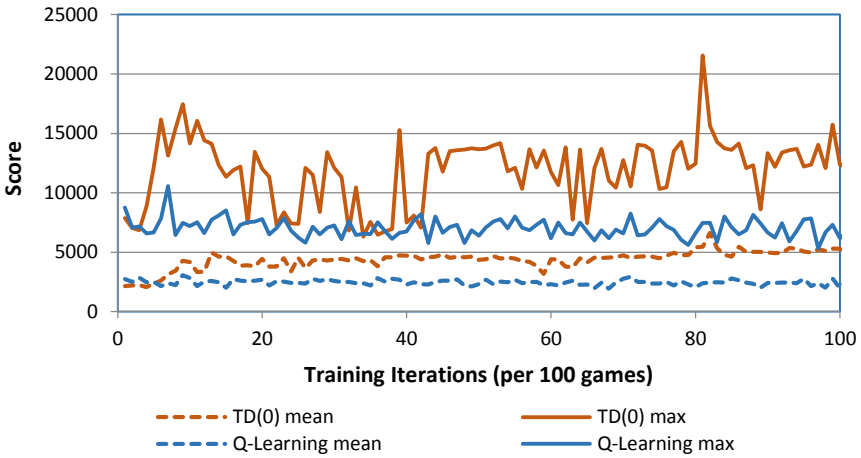

Fig. 5. Deep learning applied to 2048, using 3 × 3 filters.

Figures 4 and 5 show the performance of the deep reinforcement learning methods during the training process. The networks that use 3 × 3 filters require significantly more time than the 2 × 2 filter networks, so a total of 10,000 games were used (as compared to the 100,000 total games for the network using 2 × 2 filters). We can observe that the performance for TD(0) exceeds that for Q-learning in both kinds of networks, which corroborates the findings in [1][2]. The 2 × 2 filter network performs better than the 3 × 3 network, but additional training for the latter needs to be conducted for a more conclusive result. A single game was able to achieve a score above 50,000 in the 2 × 2 filter network, which corresponds to the single case in Figure 6 where the game ended with a maximum tile of 4096. We used the best performing weights for the 2 × 2 filter network to obtain the results in Figure 6. Again, we can see that TD(0) works better than Q-learning. The random baseline chooses a series of random inputs until the game ends, while the greedy baseline chooses whichever move gives the highest immediate score. While both reinforcement learning cases perform worse than the results in [1][2], we can confirm that at the very least the neural networks perform better than the two baselines.

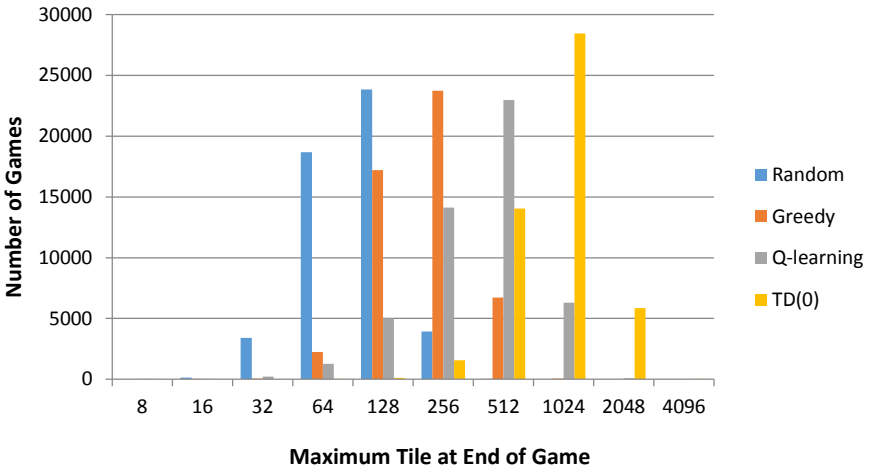

Fig. 6. Reinforcement learning vs. random and greedy baselines.

4 Conclusions

This paper presents an early attempt to adapt deep neural networks to the game of 2048. The method was based on TDL, as described in [1][2], and on deep Q-learning, as in [3]. Of the publicly available cases online, our implementation performs the best among all using (deep) neural networks. That said, the winning rate is much lower than expected, and is much worse than the previous methods for 2048 that relied solely on reinforcement learning [1][2]. From the experiment results, we think convolutional neural networks might not be suitable for a game such as 2048, where the game board is compact and patterns are not meaningful if they are translated along the board. More specifically, a certain arrangement of tiles in the corner will most assuredly have a different effect on the evaluation of a game state if, instead, the same tiles are placed in the center of the board.

Two things may be done to improve the performance of deep learning in 2048 in the future. First, the initial training process can be redesigned as a supervised learning problem, where a previous well-performing TDL program can be used as the training sample. Once a certain level of play is achieved, the program can then use reinforcement learning by itself. Second, the neural network may need to be redesigned to take into account rotational and reflectional symmetry.

References

1. Szubert, M., Jaśkowski, W.: Temporal Difference Learning of N-Tuple Networks for the Game 2048. In: IEEE Computational Intelligence and Games 2014, pp. 1-8. IEEE Press, Dortmund, Germany (2014)

2. Wu, I-C., Yeh, K.-H., Liang, C.-C., Chang, C.-C., Chiang, H.: Multi-stage Temporal Difference Learning for 2048. In: 19th International Conference on Technologies and Applications of Artificial Intelligence (TAAI), pp. 366-378. Springer International Publishing, Taipei, Taiwan (2014)
3. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv: 1312.5602 (2013)
4. Downey, J.: Evolving Neural Networks to Play 2048. Online video clip. Youtube. https://www.youtube.com/watch?v=jsVnuw5Bv0s (2014)
5. Weston, T.: 2048-Deep-Learning. https://github.com/anubisthejackle/2048-Deep-Learning (2015)