AI for Game 2048

Tianchen Zhong

December 13, 2017

Abstract

2048 is a very popular online game. It is very easy but hard to achieve its goal. Therefore we decided to develop an AI agent to solve the game. We explored two strategies in our project, one is ExpectiMax and the other is Deep Reinforcement Learning. In ExpectiMax strategy, we tried 4 different heuristic function and combined them to improve the performance of this method. In deep reinforcement learning, we used sum of grid as reward and trained two hidden layers neural network. We achieved 98% in 2048 with setting depth limit to 3 in ExpectiMax method. But we didn't achieve a good result in deep reinforcement learning method, the max tile we achieved is 512.

1 Introduction

Game 2048 is a popular single-player online game developed by Italian programmer Gabriele Cirulli.

2048 is an online addictive single-player video game invaded the Internet in 2014. One of reasons for the popularity of this game is that it is so easy but hard to win. It is not easy for a human to get the final tile 2048 to win the game. However, we could build an Artificial Intelligence game solver to help us get the highest score that human could not achieve.

In terms of the problem topic for 2048, we could view it from two aspects. Some people believe it is a search problem and implement algorithms, such as expectimax, minimax [1][2]. On the other hand, some view it as a learning problem and build a game solver with reinforcement learning method, such as temporal difference learning, Q-learning[3][4].

In particular, the method used expectimax in [1] is to treat the game as adversarial. The opponent is the computer and it will place the new tile on the board randomly. AI will choose the maximum of the expected scores of the next states. However, we need to choose weight matrix for evaluation function wisely, since it is very important for the next decision. According to the result for [2], which uses the number of open tiles and heavy weight on the edge as heurists, it achieved highest tile of 32768 for 36% of trials under limited search

depth of 8. But it need to think 150ms each move, which means the runtime for this method is very large if we want to get high performance. In[1], the author compares Monte-Carlo tree-search and expectimax depth-limited search. Since Monte-Carlo search do not have any domain knowledge, it performs worse than expectimax search, which is given evaluation function that rewards board properties.

We can also view 2048 as Markov decision process, Szubert[3] employed temporal difference learning to learn the weights of an N-tuple network and won 97% game, scoring 100178 on average. Their best network contained nearly 23 million weights parameters. Recently, Wojciech [4] improved this method with temporal coherence learning and multi-stage function, achieving average 589355 with spending 100ms in each move. Their performances were much better than others, but they also have a longer training time.

2 Approaches

In the first part of projecrt, we implemented ExpectiMax search methods and improved them with human heuristics. Since the search tree would expand exponentially, we tried to use the depth-limited search to max performance in a limited computation resource. At last, we attempted to use deep reinforcement learning with simple neural network. The neural network was used to learn Q value for each state. However, training model is computationally expensive for my laptop and the limited time, my performance is not very well. Although it is better than human player, but not better than search trees.

2.1 Game 2048

Player play this game on a 4 * 4 grid board, and each tile having an even number(Fig.1). Player can make four direction movement, which are up, down, left and right. Then all ties will move in that direction until it stopped by another tile or the border of the grid. If two tiles having the same number collide while moving, they will merge into a new grid with their sum on it. After that, this new tile will not merge with its neighbors again during this move. A new tile with number 2 or 4 will randomly appear on the empty tiles and player need to makes a new move.

If there is no empty tiles on the grid, the game would end. If player could get 2048 before the game end, he wins. The score for each player starts at zero, and it adds by the value of the new tile when two tiles collide together.

2.2 ExpectiMax

ExpectiMax search is a classic algorithm in two-player game. In our Game 2048, we can view computer as an opponent and it randomly pick two and four with probability 9:1 ratio

Figure 1: 2048 game.

in the empty tile. Before describing the specific algorithm, we need to clarify the notations we used in the following part

1. $players : \{agent, opponent(computer)\}$
2. s: the current state, in 2048 is the whole grid.
3. a: action taken.
4. $actions$: possible actions at state s.
5. s' : resulting states after taking action a at states.
6. s'' : state after picking an empty tile.
7. $eval(s)$: evaluation function for state s.
8. d: the current depth of the search tree.

Using these notations help us to represent algorithm presudo code, which is shown as below.

Algorithm 1 ExpectiMax

```
if state
    s is a chance node then
    for actiona ∈ actions do
        s ← s'
        for emptytile ∈ grid do
            add_2_to_tile : s' ← s''_1
            score[a] ← score[a] + 0.9 * eval(s''_1)
            add_4_to_tile : s' ← s''_2
            score[a] ← score[a] + 0.1 * eval(s''_2)
```

For the evaluation function we used in 2048, it needs heuristics and domain knowledge from human player. The performance of our algorithm depends on how good our evaluation function is. There were several heuristics we could used in this Game.

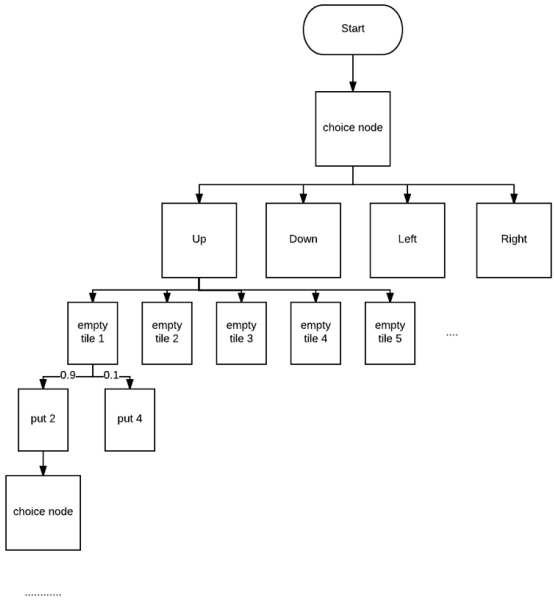


Figure 2: ExpectiMax Flow chart

1. Total sum of all number in the grid: In this game, our goal is to achieve 2048, which means we need two 1024. Therefore, making sum of all grids as much as possible to help agent achieve higher tile value.

2. Empty tile: the empty space is the key to have high score. In many failure circumstance, all the grid was full of random number and they cannot merge together. Therefore, if we could get many empty tiles in each state, it is more possible to have space to merge tiles together to get a higher score.

3. Possible merge: when we see same number in the adjacent position, it is very easy to merge them after single action. Therefore we could have more big possible tiles to get a higher value tile.

4. Monotonicity in a line: This heuristic is not obvious but after playing some games, you could see that how great if you get a monotonic line. For example, in the Figure 3 we could just swipe left three times, and we can achieve our goal. For a monotonic line, it is the most possible way to achieve the high goal. Put tile in an ascending or descending order help tiles merge consecuatively.

For evaluation function, we use these heurstics to evaluate the current state. We tried to use them separately and combine them with different wieht to make them have equal weight in the final score. For implement part, it is not wise to caluate them value again and again,

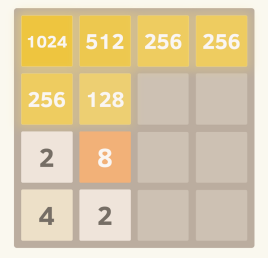


Figure 3: One possible game state.

therefore we build a heuristic table for each row and calculate evaluation value in advance. The number of all possible state is 16^4, which is 65536. When we need to evaluate the current state, we just need to look up the table and add values for all rows and columns.

2.3 Deep Reinforcement Learning

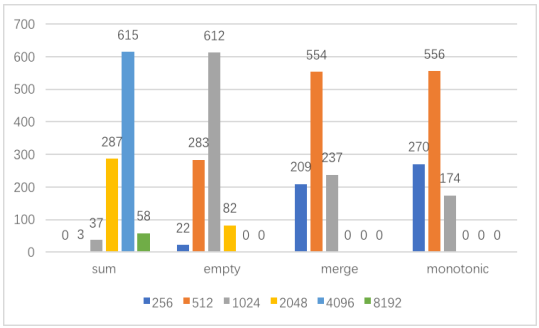
We can also view this problem as a MDP process, and we implement a reinforcement learnign to let agent learning policy to play this game. However, the game state is very large, which is $16^{16} \approx 1.8 * 10^{19}$. It is impossible to store all states in my laptop. So we decided to use neural network to learn Q value for each action in the sepcific state. We implement a neural network with 3 layers. The two hidden layers we chosen of size 256 and size 64. The input is a vector with size 16, which represents all tiles value, the output is a vector of size 4, and each is associated with Q value after taking 4 different actions on this state.

In the two hidden layers, we use ReLu, $h : x- > (x)^+$ activation function. Linear transformation is used in the output layer to get Q value. When we need to decide which action to take, we just compare all four Q value output and choose the maximum value. It is possible that the best move cannot change the state, therefore we will use the second max output value as our action to execuate. If all four moves cannot change game state, game is over.

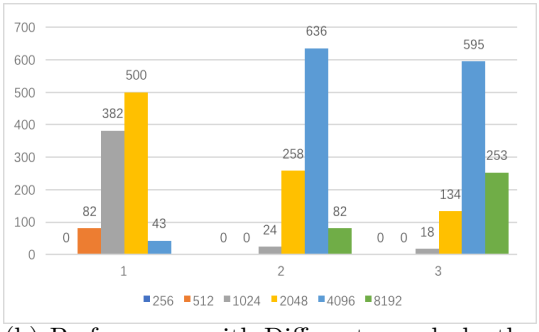
For the back propogation part, we use Q learning formulation to calculate the new Q value after each action.

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(R(s) + \gamma max(Q(s'',a))$$

Reward function is critical in this method, because we want our neural network could know which action is better than others. Therefore, we tried all possible move at that state and calculate reward for this action. Then we mean-center and change value in to a range in $[-50, 50]$, therefore we can give a positive value to a good action and give a negative value to a poor action. Reward function is defined as the value for all merges after taking action.



(a) Comparation between different heuristic functions



(b) Performance with Different search depth

Figure 4: ExpectiMax Result

3 Result

We tried 1000 trials for each hehuristic methods and we set a depth limit to 2 for ExpectiMax search. The reason for that is because the limitation of computation resource. From the graph we could see that only using sum for all tiles as heuristic method is much better than other heuristic functions, which could achieve 287 in 2048, and get 615 times in 4096, which gets 90% to achieve goal. We think the sum of all tiles could really tell the state of the grid, while other heuristic methods could only tell you one part of grid state. And they could not tell you wheter this action is better than others in the long run.

We also combine these heuristic methods with weight 5 for sum, 2 for empty tiles, 1 for merge, 1 for monotonic, which defined after getting the result of justing using single heurstic function. We set depth from 1 to 3 and run 1000 tries. We could see there is an increasing trend with depth-limit value and achieve 98% to get 2048 with depth 3. It is possible for us to get higher performance when we increase depth-limit. We also compared our program with the best ExpectiMax program on the Internet. The results are both based on the depth with 2. We could see that our result is close to the best one. The improvement could be from these two ways: 1. Change parameters for each heurstic methods. 2. Using more heurstic methods to achieve higher score.

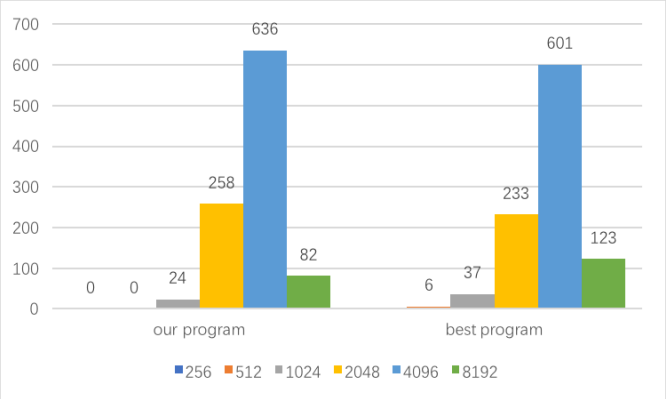


Figure 5: Compare with best program.

For the reinforecement learning algorithm result, we didn't achieve 2048. The highest score we achieve is 512. The result is better than human player but not as good as search tree method. Compared to the Expectimax, this method use less time to make decision but it could not get the goal. However, since just using neural network to predict Q value for each action, which means we could not see the state in advance. Therefore using neural network maybe not a good way to solve this game. From [5] we could use MCTS and neural network to implement on 2048 in the future to see whether it could help us to achieve the goal.

Table 1: Deep Reinforcement Learning result

max tile	64	128	256	512
times	90	408	457	45

References

[1] P. Rodgers and J. Levine, "An investigation into 2048 ai strategies," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–2, Aug 2014.

[2] robert xiao, "Ai for the 2048 game," 2014.

[3] M. Szubert and W. Jakowski, "Temporal difference learning of n-tuple networks for the game 2048," in *2014 IEEE Conference on Computational Intelligence and Games*, pp. 1–8, Aug 2014.

[4] W. Jaskowski, "Mastering 2048 with delayed temporal coherence learning, multi-stage weight promotion, redundant encoding and carousel shaping," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2017.

[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.