24-787 Artificial Intelligence and Machine Learning for Engineering Design
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
Project Report 2014

LEARNING TO PLAY 2048

Aashir Gajjar
Carnegie Mellon University
Pittsburgh, PA, USA
ajaggar@andrew.cmu.edu

Joel Feske
Carnegie Mellon University
Pittsburgh, PA, USA
jfeske@andrew.cmu.edu

Rohan Krishnan
Carnegie Mellon University
Pittsburgh, PA, USA
rohankri@andrew.cmu.edu

ABSTRACT

2048 is a puzzle game that has grown popular in the past few months. In this study, neural networks were used to learn strategies from human and AI players using both raw and processed data, and genetic algorithms were used to generate strategies from scratch. Games were simulated, and average performance was compared to an AI player which generated random moves. Neural networks were trained on varying numbers of games depending on the available data, and in general were able to successfully emulate the strategy they were trained on, with larger data sets producing more faithful emulation, as expected. Neural networks trained on processed data representing abstract qualities of the board out-performed networks trained only on board states. Genetic algorithms were also implemented to find novel strategies by attempting to predict the best next 6 moves based on the state of the current board. Both strategies learned by neural networks and strategies learned via the genetic algorithm were able to out-perform the random AI player.

INTRODUCTION

2048 is an online game created by Gabriele Cirulli, in which a player must use their arrow keys to move numbered tiles across a 4x4 board. When two tiles of the same value merge, they form a new tile of double the value of the original two tiles. With every move, a new tile will spawn on a random location on the board (this tile will usually be a 2, although the game will occasionally spawn a 4). At the start of the game, the player is given a mostly empty board with two low-valued tiles placed in random locations on the board; from this point on, the player must merge tiles in an attempt to reach the highest tile they can, until no more moves can be made (i.e. until there are no more empty tiles on the board and no two tiles can be merged). The name of the game represents what is usually the final target tile; however, the game can continue to be played beyond the 2048 tile should the player wish to attempt higher scores.

Since new tiles spawn randomly on the board as the game is being played, the problem is non-deterministic in nature and so makes for an interesting AI/ML problem. In our work, we explore two fundamentally different approaches to solving the problem of learning to play 2048. In our first approach, we attempt to have an algorithm learn to play the game by first watching others play and then emulating their strategies. Our second approach attempts to find these strategies on its own, without requiring any user input or data.

RELATED WORK

As the game gained popularity, there have been many attempts to solve the game by implementing different algorithms. The most popular AI algorithms are search based. A blog named *MATLAB Central* has posted a GUI for the game programmed in MATLAB [1] and many AIs have been submitted as attempts to solve the game; we use the same code to generate the game and implement our own algorithms. We have also used *myAI.m* [2] and *oliverAI.m* [3] – two AI programs submitted on the same forums. *myAI* executes a random sequence of moves while *oliverAI* is a brute-search algorithm which dynamically looks forward to every possible board between two and five moves ahead to select the move with the smallest chance of ending the game.

It is quite evident that the search space for 2048 is quite huge and increases exponentially with depth. This requires a huge amount of computational space and time. One approach is to use alpha-beta pruning to reduce this search space and use heuristics for evaluation. This approach has already been used to solve the puzzle by Matt Overlan [4] and it seems quite successful; for our work, however, we look to find a simpler, less computationally expensive approach to the problem.

TECHNICAL APPROACH

Artificial Neural Network (ANN) – Raw Approach

Our first ANNs were trained only on the raw data from the board. They had 16 inputs where they were given the value of the tile at a certain grid point, 1000 hidden nodes, and 4 outputs, representing the four possible moves that could be made.

This approach was successful to some degree, but the gameplay was crude, and the neural network was not able to abstract the patterns of play to higher valued boards for which there was less data. For example, having a row with the sequence 8-4-2-2 would not be seen by the neural network as at all similar to a row with the sequence 64-32-16-16, even though they are both collapsible to the left. For this reason, we speculated that the network would perform better if trained not on the board states themselves, but on more abstract qualities of those board states.

Artificial Neural Network – Heuristic Approach

For choosing the abstract qualities (heuristics) we kept the goal state of the game in mind and some obvious factors that a player considers while making a move. In order to feed the neural network with inputs, we generate four new boards corresponding to the four possible moves and evaluate all of these heuristics on all the boards. (Here we ignore the randomness of the new boards but it can be taken into account by generating all possibilities of the new boards and calculating the average of the heuristics). For every board that is used to train the ANN, we preprocess the data and form four new random boards corresponding to every move on that board. We then calculate the heuristics from these four new boards as well as from the current board and feed it into the neural network as inputs. We assign the move made in the training data for the particular board as an output/label to the network. Hence we have 17 inputs and 4 outputs for the ANN. We trained new neural networks on the following heuristics of the board:

```
Number of empty tiles in the next board (4)
Number of adjacent tiles of the same value (4)
Degree of monotonicity in all four directions on
the current board (4)
Value of the highest tile in the next board (4)
Value of highest tile in the current board. (1)
```

Genetic Algorithm

Thus far we have discussed using a neural network to learn from the behavior of an existing player of the game. An altogether different approach to solving this problem is to have an algorithm attempt to find the best solution by itself; we attempt to complete this task using a genetic algorithm (GA). The GA takes in an input board from a game in progress and runs a series of simulations on each member of its population in an attempt to predict what the board will look like *n* moves later. By running this GA every *n* moves until no more moves can be executed, we can observe how its performance compares to both humans and existent AI strategies.

Each integer in the representation of an *n*-length chromosome corresponds to a single move: up (1), right (2), down (3) or left (4). The fitness function returns a weighted version of the predicted score achieved by executing the move list described in any single chromosome; this score uses the same scoring mechanism seen in 2048 – when two tiles are combined, the score is increased by the value of the resulting tile. The fitness function applies a lighter weightage to scores returned by moves towards the end of a chromosome than it does to those at the beginning – this gives a preference to "good" moves occurring earlier in the move list, which is desirable since it becomes harder to assume that a simulation is accurate as more moves are simulated. Since the GA simply attempts to maximize the fitness over *n* moves, it can often continue iterating until it hits its maximum iteration threshold. However, if it finds that the best solution remains unchanged over some number of consecutive iterations, it will accept that move list as a final solution (note that over these iterations, the fitness of this chromosome will likely not remain constant due to the inherent randomness of the simulation in each iteration).

The biggest challenge of this approach is to find a set of parameters for the GA that allow the algorithm to find desirable move lists in a timely fashion. The length of the move list, *n*, is the most important of these parameters; some parameters, such as population and offspring size, fall out of this value to some extent, while others, such as mutation rates and termination conditions, require more thought.

RESULTS

Neural Network

Histograms were created to show the average performance of each approach over a number of simulated games. All performances were compared to the performance of the random AI player, which chooses a move based on the generation of a pseudo random number between 1 and 4. Figure**s 1a** and **1b** show a comparison of the performance of *myAI* (the random strategy algorithm) and the neural network trained on *myAI*. This neural network was the only one to outperform the data that it was trained on.

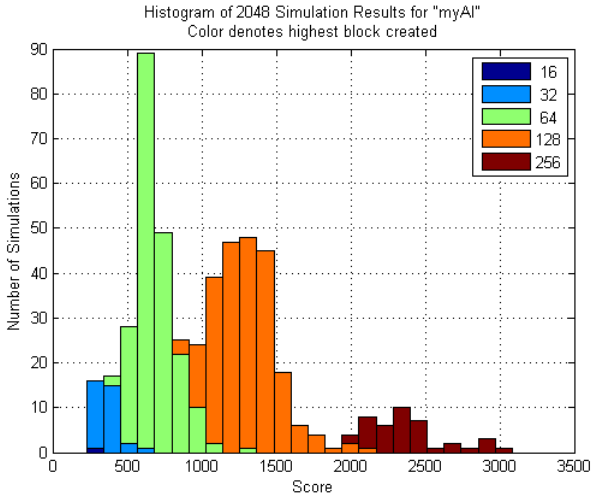

Figure 1a – Score distributions and highest tile achieved for *myAI*

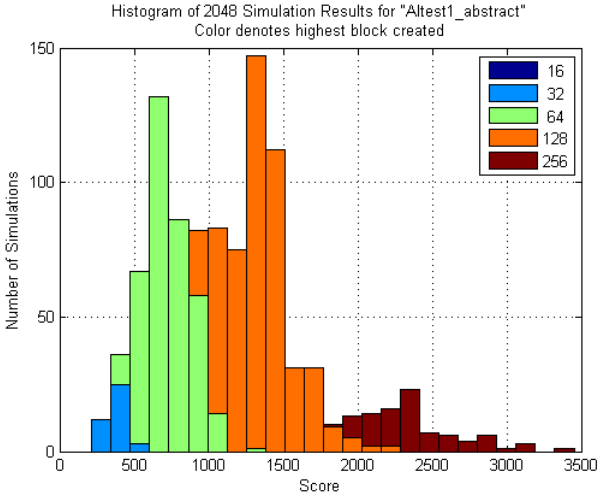

Figure 1b – Score distributions and highest tile achieved for a neural network trained on *myAI*

Figure 2 shows the performance of the neural network trained on 11 games that Joel played, in which he reached high tiles of 2048 twice, 1024 4 times, and 512 5 times. Unfortunately, it would have been much too time consuming to play hundreds of games manually. Nonetheless, this neural network was able to outperform the random AI. The most notable thing about the performance of this network in particular is its imitation of the play style that it was trained on, as it tends to build up tiles with the highest value in the bottom left corner with decreasing values along successive diagonals. Despite this pattern, the network never got to a tile higher than 512.

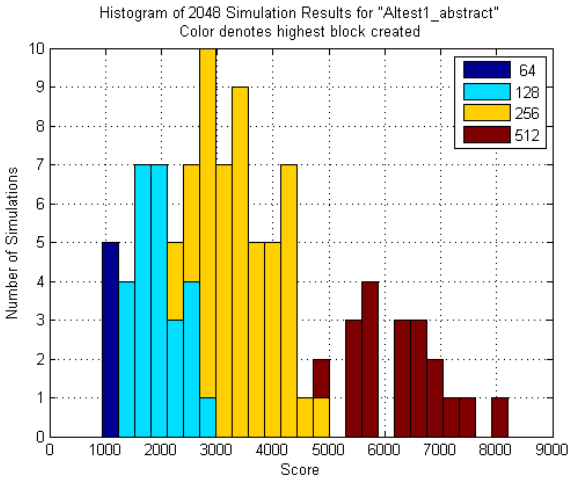

Figure 2 – Score distributions and highest tile achieved for neural networks trained on a human player

Figures **3a** and **3b** shows a comparison of the performance of *oliverAI* with the neural network trained on the algorithm. The source algorithm's strategy is very successful, achieving a highest tile of 2048 80% of the time. The neural network trained on Oliver AI performed the best out of any of the neural networks, but did not approach the performance of the AI itself.

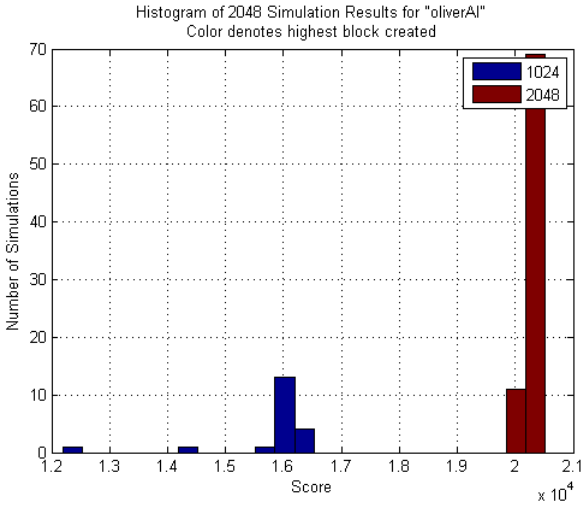

Figure 3a – Score distributions and highest tile achieved for *oliverAI*

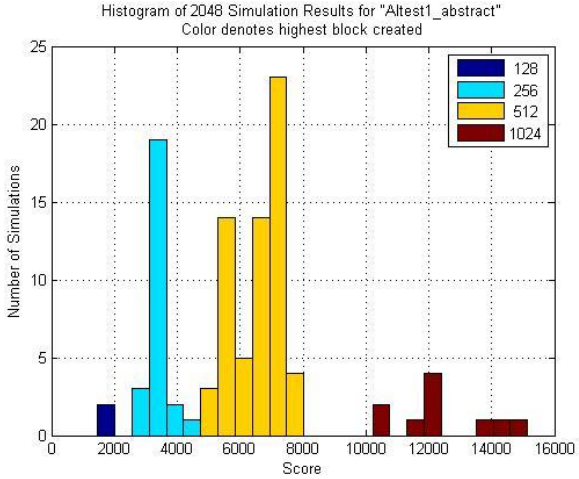

Figure 3b – Score distributions and highest tile achieved for a neural network trained on *oliverAI*

Genetic Algorithm

We found that the GA performed best when $n = 6$ (i.e. when it attempted to predict the next 6 moves). Values of n too much lower than this, and not enough will have happened in the simulated games to be able to choose a "best" move list; much higher values of n, and the simulated predictions became inaccurate due to the inherent randomness of the game. These higher values also result in a much higher computation time – ideally, the algorithm should be able to play a single game of 2048 in its entirety in a few minutes.

Representation	Integers (1-4)
Move list length (n)	6
Population size	90
Number of offspring	90
Crossover method	Uniform
Crossover rate	0.8
Mutation method	Random resetting
Mutation rate	0.8
Parent selection	Tournament (k = 5)
Survivor selection	Generational
Initialization	Random
Termination condition	50 iterations OR no change in best solution after 10 iterations

Table 1 - Key Parameters of the final GA

Table 1 shows the key parameters of the final GA. These parameters went through an iterative tweaking process, where the qualitative performance of the resultant GA could be observed and evaluated. The final values represent those which gave a GA that was able to achieve relatively high scores (reaching tiles of 512 or higher regularly) in an acceptable timeframe – that is, in a playing time comparable to that of a human. Using this final GA, we were able to evaluate its performance over 200 simulations of 2048. The results of these simulations can be seen in **Figure 4**; the algorithm finished with a highest tile of 512 49.5% of the time, and a tile of 1024 14% of the time.

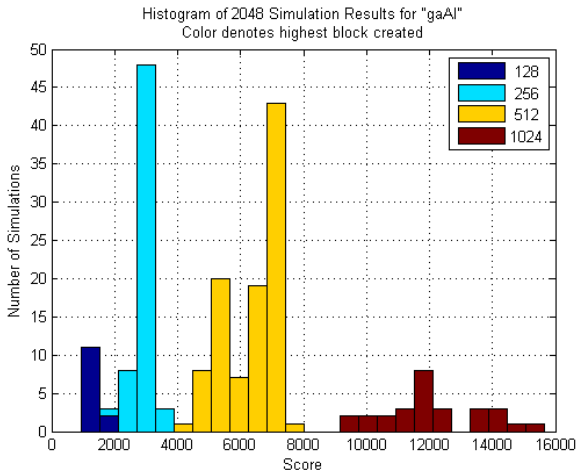

Figure 4 - Score distributions and highest tile achieved for the GA after 200 simulations

DISCUSSION

The most noticeable result is that the neural networks tend to underperform compared to the AI or human player they were trained on. The exception to this is the neural network trained on the random AI which did slightly outperform the random AI itself, however this is likely due to chance alone. It may have been the case that the random AI happened upon some successful patterns of movement that the neural network internalized and executed, but if we were to greatly increase the number of games it was trained on, this discrepancy would likely disappear. The fact that the neural network trained on *oliverAI* performed than the one trained on Joel's input, which in turn performed better than that trained on the random strategy, suggests that our neural network method is in fact learning well from its source data, considering the performance of the source data maintains that same order.

The underperformance of the networks trained on data gathered from Joel and *oliverAI* (hereafter referred to as NNJ and NNO respectively) is likely due to several factors. The first is simply the size of the data sets that the networks were trained on. NNJ in particular was only trained on 11 games, and so its lack of performance was probably due in large part to a simple lack of training data. The performance of NNO drastically improved as the number of games it was trained on increased from 10 to 20, and finally to 30, and it was only in this final iteration that it was able to score 1024 with some consistency.

The second reason the neural networks were outperformed by AI and human players is that they essentially make snap judgments of the board based on vague heuristics learned from data, but they do not anticipate the consequences of their moves. Human players are able to look ahead to the consequences of their moves, and to modify their pattern of play in certain situations where it may not be advantageous. Similarly, *oliverAI* actually looks ahead to every possible eventuality between two and five moves ahead to select the move with the smallest chance of ending the game. The neural networks, on the other hand, are trapped in the present, and must make moves based on rules that are only present implicitly in the pattern of weights in the network. Consequently, they play quickly but with poorer results.

In the future, it would be interesting to see if a simple increase in the amount of training data could significantly improve the performance of NNO above its current level, and if it would approach the performance of *oliverAI* or plateau at some lower level. It would also be interesting to attempt to filter data fed into each neural network in some way, such that the network only learns from moves which resulted in a "successful" game – i.e. one that resulted in some acceptably high score. The performance of each network may then increase dramatically, since it would no longer train itself on "poor" choices in the training data.

The results over the 200 simulations of the game using our GA are promising, as they far out-perform *myAI* and are even comparable to human performance. However, the algorithm was never able to achieve a 2048, suggesting a fundamental flaw in our methodology that prevents the GA from finishing the game. This flaw could be down to our parameter selection; future work in this regard could include optimizing these parameters in a more exact manner. This would require far more simulations across several GAs with varying parameters, a time-consuming but ultimately extremely useful task. The GA's fitness function may also be worth looking at; a more successful function may look to evaluate more measures of the chromosome based on the simulated board after the 6 moves. These measures could include the number of adjacent and equal tiles, the number of empty spaces or other such qualities, and would use a Pareto front to optimize these various measures simultaneously.

CONCLUSIONS

Artificial neural networks are able to approximate a non-linear function which encodes the strategy of various 2048 game players. Fidelity of emulation is directly related the amount of training data, and is also heightened when the inputs represent abstract qualities of the board rather than simple board states.

Genetic algorithms were also able to find relatively successful strategies for the game. However, the inability to finish the game in any of our simulations may suggest a fundamental limitation in their usefulness for this task. Nonetheless, the algorithm demonstrates a promising, simple approach to solving this non-deterministic problem that may provide a base on which to develop a more complete solution.

REFERENCES

[1] Doke, Jiro. *2048 MATLAB Edition*. *MATLAB Central*. Mathworks, 4 Apr. 2014. Web. 24 Apr. 2014. <http://www.mathworks.com/matlabcentral/fileexchange/index?term=2048+game>

[2] Doke, Jiro. *myAI*. Computer Software. Vers. 1. N.p. 4 Apr. 2014. Web. 4 Apr. 2014. <http://www.mathworks.com/matlabcentral/fileexchange/46124-2048-matlab-edition/content/myAI.m>

[3] Woodford, Oliver. *oliverAI*. Computer software. Vers. 1. N.p., 9 Apr. 2014. Web. 25 Apr. 2014. <http://blogs.mathworks.com/pick/2014/04/04/submit-your-algorithms-to-solve-2048/>

[4] Overlan, Matt. *2048-AI*. Computer software. Vers. 1. GitHub, 2014. Web. 11 Mar. 2014. <https://github.com/ov3y/2048-AI>