Module 6 Report - Johannes Moskvil

23. november 2015 16:43

This report will detail my findings and attempts at creating a working solution for module 6.

INTRODUCTION

For this module there are a lot of variables to tweak that has an effect on the final performance.

- Neural network configurations:
  - Topology
  - Number of nodes
  - Learning rate
  - Activation function
  - Number of epochs to train on
- Board preprocessing
- Acquisition of training and test data

DATA SETS

Before discussing my choice and comparison of parameters, I will mentioned the different data sets I experimented with. I collected two sets of data sets that each tries to be true to a specific strategy, this was important and highly coupled to the preprocessing phase in order to learn anything useful from the examples:

1. **Acquisition method:** Myself playing 2048 naturally.
   Strategy:
   - Highest tile in top-left corner
   - Always keep top row filled in monotonic decreasing order from left to right
   - Try to build next block in [1,3] (0-indexed) to merge with [0,3]. (Kind of a semi-snake strategy)
   - Never press down
2. **Acquisition method:** My module 4 AI playing 2048.
   Strategy:
   - Does not prioritize any corner
   - Tries keep board monotonic and smooth
   - Prioritize highest tile in a corner

Testing data for each contained roughly 2000 training examples and 300 testing examples.

PREPROCESSING

After attempting to learn from raw examples (or scaled to [0,1]) without learning anything at all, I attempted at preprocessing the board into a state that would mimic the properties of the strategy used to generate the examples.

SNAKE PREPROCESSING

One of the easiest and successful heuristics from module 4 was the snake heuristc (though I did not use it). It generally fit s very well with how people play the game, as well as how I play it myself. This way of preprocessing takes the input state (board representation) and element-wise multiplies the lists. The neural network input is still 16 nodes, one for each tile.

EXAMPLE

$$Board = \begin{matrix} 256 & 128 & 64 & 32 \\ 0 & 2 & 4 & 16 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}, snake_{matrix} = \begin{matrix} 10 & 8 & 7 & 6.5 \\ 0.5 & 0.7 & 1 & 3 \\ -0.5 & -1.5 & -1.8 & -2 \\ -3.8 & -3.7 & -3.5 & -3 \end{matrix}$$

The input to the network after the board has been preprocessed would then look like this

Ann input = [2560, 1024, 448, 208, 0, 0.7, 4, 48, 0, 0, 0, 0, 0, 0, 0, 0]

COMBINED HEURISTICS PREPROCESSING

For the other preprocessing tactics I tried to preprocess the board in terms of the heuristics used by the AI in generating the testing data. This preprocessing takes the 16 tile board as input, and returns 7 nodes, each representing the following properties

- Smoothness vertically
- Smoothness horizontally
- Monotonicity from left to right
- Monotonicity from right to left
- Monotonicity from bottom up
- Monotonicity from top down
- Highest valued corner tile

COMPARISON

In order to do a valid comparison, only one parameter can be changed at a time. For this part, that parameter will be the choice of preprocessing, but in order to test the network, a topology is needed. To compare the two forms of preprocessing I will present statistical data of the average max tile value over a 50 run test on three different topologies, to see if any preprocessing is better than the other, regardless of the topology. The three network configurations that will be considered are:

- [20,5] - Hyperbolic tangent activation
- [20,5] - Sigmoid activation
- [50] - Hyperbolic tangent activation

The snake preprocessing will be ran on the data set generated by me playing the game, while the combined heuristic preprocessing will be ran on data generated by the ai. For each of the networks I computed the P-value from Welch's T-test (50 simulated ai games compared to 50 simulated random games) 10 times. The table below summarizes the results. It shows the average P-value of the 10 repetitions for each configuration.

Network:	[20, 5] - tanh	[20, 5] - sigmoid	[50] - tanh
P-value (snake)	0.1185	0.142	0.140
P-value (combined)	0.348	0.456	0.195

This comparison is about which preprocessing method is better given the corresponding input data, not the network topology. The table above gives a unified conclusion that the *snake* preprocessing gives the better P-value (even though it is still poor and yield few actualy demo points). Going forward, I will be using the *snake* preprocessing.

DESIGNING THE ANN

This section contains discussion and comparison of several different neural network configurations attempted for this module. The network really just tries to find the solution to a multi-unknown variable equation set, where weights are the unknown variables, and the examples constitues an equation. Hence there must be more equations than variables. My data set is around 2000 examples, so the total number of weights should be less than that.

Instead of the classic way of rating neural networks, by having an independent testing set to evaluate the network, I chose instead to test the network by doing the same test as above; run 50 simulated ai games, and 50 simulated random games, and rate the networks in terms of the average P-value returned by Welch's test. Each network is trained 10 times on the whole training data, then the 50+50 run test is performed. This process is reapeated 10 times (a new network is trained every time). The table below summarizes the averaged P-values for four different network configurations.

Network	[19,4] - tanh - lr=0.001	[19,4] - sigmoid - lr=0.001	[200,100] - tanh - lr=0.001	[50] - sigmoid - lr=0.001
Average P-value	0.01644	0.1705	0.2539	0.1745

These are just a few of the many configurations I tried, but the point to be made is that [19, 4] - tanh performed best of all the possibilities I tested. From the table, and from additional experimentation, it seems that the hyperbolic tangent activation function performs better than the sigmoid function. The reason the [200,100] performed so poorly could be related to the statement above; it contains 16*200 + 200*100 + 100*4 = 23 600 weights to adjust, while there are only 2000 examples to learn from. But again, I got results closer to the performance of [19, 4] tanh with networks exceeding 2000 weights. So it does not seem to play such a vital role in this setting. Could be because the best networks I got, still is far from the best solution. But to sum up - smaller networks with tanh activation worked better for me in this module.

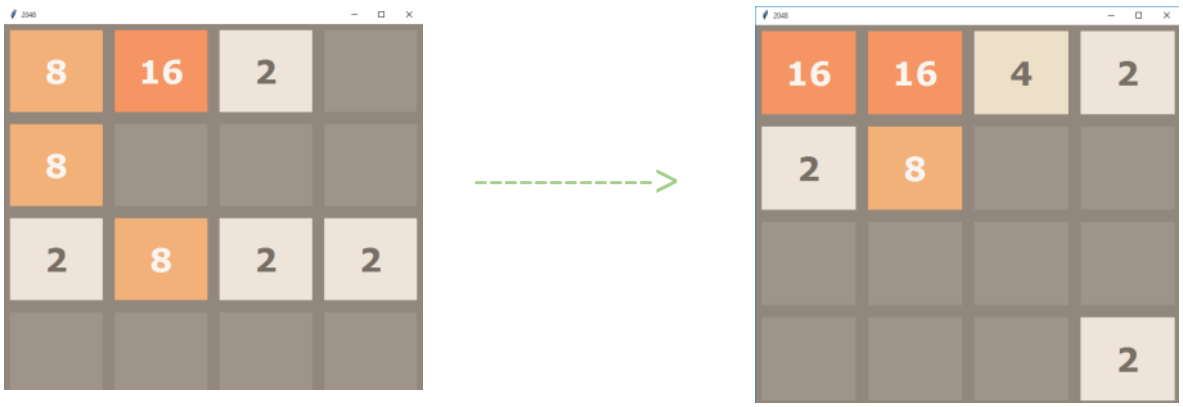
ANALYSIS OF BEST CONFIGURATION

The above discussion resulted in my best ANN having the following configurations:

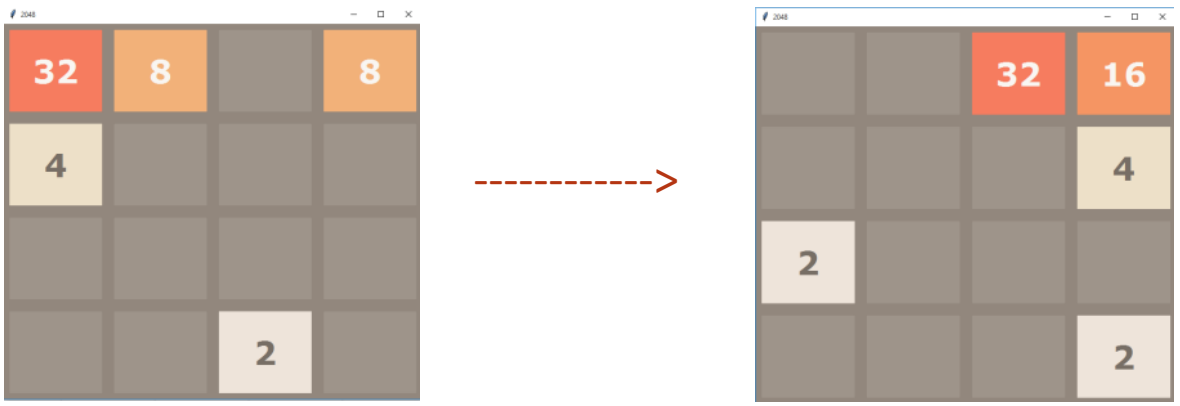
- Network: [19, 4] - Hyperbolic Tangent activation.
- Preprocessing: Multiply every tile with a snake pattern
- Use data set from myself playing 2048

It plays 2048 far from perfectly, but still always outperforms random. To be honest, it rarely plays the move any logical person would do (or a good minimax AI), but I guess there are those few logical moves that seperates it positively from all the random runs.

Examples



The most logical move here is to go up. You set up the merging of the two 16 tiles into a 32 in the corner, while mainting a full and monotonic upper row. In this case the ANN chose to do this.



The behaviour the ANN are trying to learn is to always keep the board smooth, monotonic and highest tile in a corner. This move fails on all accounts. Moving right here is a bad move that hurts the structure.

I think the most important thing the ANN has learned from the examples is to eliminate the down move. If you are trying to build your highest tile in the top-left corner, the most destructive move is to move down. Should the ANN learn to do this, it should already be doing better than random.