IT3105 Module 6

Johan Slettevold
Iver Egge

November 2015

1 Artificial Neural Network

Each representation had 10 epochs, and one hidden layer with 1000 weights. We discovered that 10,000 boards and moves as training data was enough to be good enough to train our network.

1.1 Topology

Our network had only one hidden layer. This was based on the task description and justified after we tried one to four hidden layers and found that the one-layer approach performed the best.

The number of weights was crucial in determining the performance of our ANN. When the ANN was operating with significantly more weights than the size of our training data, it performed very badly. We believe this is caused by the ANN remembering the outcomes of specific moves very well and not adjusting to allow good guesses when an unknown board was presented.

Time consumption was also a factor that influenced our choice for the number of weights. 1000 weights seemed to be a good trade-off between time consumption and performance.

1.2 Activation function

We tried to run our ANN using both the Sigmoid activation function as well as rectifier linear units (RLUs). We clearly saw a improvement using RLUs, as proved by the graphs produced from 50 runs of each (figure 1, figure 3).

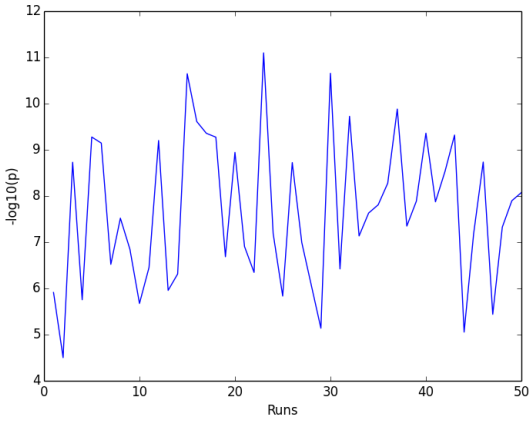

Figure 1: Sigmoid pattern used for activation function

The average P-value for the Sigmoid activation function was 1.2557384768182824e-06. Using RLUs this average P-value decreased to 10^{-8}, as seen below figure 3.

1.3 Learning rate

We experimented with a handful of learning rates in between 1 and 0.00001. While the values 1 and 0.00001 were hopelessly bad, 0,01 and 0.0001 did somewhat good. However, 0.001, the same rate as in module 5 achieved the best results.

2 Representations

The board was represented as a one-dimensional array containing all the tile values (16 elements). Before being fed to Theano all the tile values were replaced with the logarithmic value in order to normalize the input.

The training data was produced by letting our Expectimax algorithm output values for moves based on different boards. Our Expectimax algorithm used a heuristic based on a ”snake”-pattern.

2.1 Normalized with snake pattern score

In this representation there were an additional 16 elements in the board representation. These values where the product of the tile and the weight of the ”snake”-score for that position. The following graph shows 50 runs with this representation (figure 2):

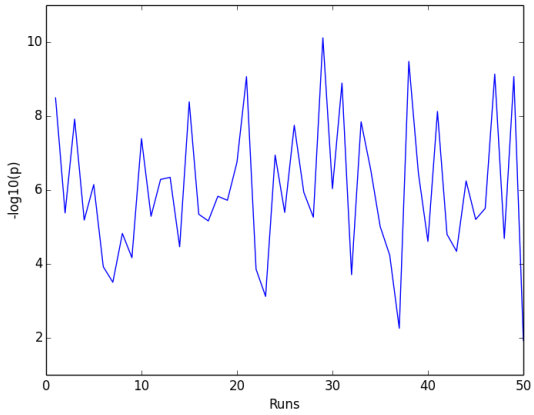

Figure 2: Representation with snake pattern score

The average P-value for this run was 3.856979865558812e-04.

2.2 Plain normalized

In this representation, there were no additional data given to Theano apart from the tile values and the desired move. The following graph shows 50 runs with this representation (figure 3):

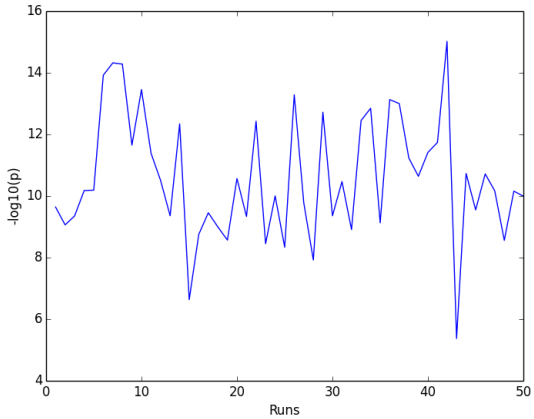

Figure 3: Representation without snake pattern score

The average P-value for this run was 9.015711086488738e-08.

2.3 Analysis

The representation using plain normalized values was four orders of magnitude better that the one with snake pattern score.

2.4 Discussion

Although preprocessing probably could have increased our results, we were not successful in our experiments. We could possibly have tried several other representations, as further exploring the connections between tiles, expanding the input weights from 16 to reasonable values between 32 and 128 - but since our 16 weight representation turned out to be both tidy and effective, we ended choosing this solution.

3 Analysis of ANN

Our ANN plays the game better than the random player at almost every run (96%). It shows that it moves the tiles in a "snake"-like pattern just as our Expectimax algorithm, most of the times.

3.1 Highlights

We used "term2048" to run our ANN against a game. "term2048" is a CLI python game for 2048. The boards shown below are to be read left to right.

Here our ANN shows that is does not confront to our "snake"-pattern. At the third move it does not merge the two 16s in order to place a 64 at the second row. Our Expectimax would be forced to do so as the heuristic value of this move would be much greater than what our ANN produces:

```
64   2 128  32        64   2 128  32        64   2 128  32
16 128  32   8        16 128  32   8        16 128  32   8
 4   8  16   4         4  16  16   4         4  32   4   2
 2   8   2   2         2   2   2   2         4   4   .   .
```

Here our ANN shows that it is able to merge tiles together, following the "snake"-pattern heuristic of our Expectimax:

```
 8 128 128 256        2   8 256 256        .   2   8 512
32   8   2   2        .  32   8   4        .  32   8   4
16   2   .   .        .   .  16   2        .   2  16   4
 2   .   .   .        .   .   .   2        .   .   .   2
```