

# Neural Networks for Game Playing

Johannes Omberg Lier

Per-Christian Berg

November 24, 2015

## 1 Discussion

From module five we had gained some insight about how the topology affected the performance of our Artificial Neural Network (ANN). Since the underlying problem of this module is not purely image classification we can't be certain that the topology and parameter specification in the previous module will yield the same results.

From the previous module the most suitable topology and parameter specifications is shown in Table 1.

Layer 1	Layer 2	Activation Function	Learning Rate	Init Weights
1024	720	Rectify	0.001	Random

Table 1: Parameter Specifications

This is the starting point in which we wanted to develop our new network. Since the input layer had decreased in size for this problem, we decided to calculate the ratios between the layers in order to construct a suitable network. The ratios between the input layer and the first hidden layer, and the ratio between the first hidden layer and the next from our previous network is shown below.

$$r_1 = \frac{H_1}{Input} = \frac{1024}{784} \approx 1.3 \quad r_2 = \frac{H_2}{H_1} = \frac{720}{1024} \approx 0.7$$

The 2048 game state is now represented with 192 input nodes. The size of the hidden layers were then calculated as shown below:

$$H_1 = Input \times r_1 = 192 \times 1.3 \approx 250 \quad H_2 = H_1 \times r_2 = 250 \times 0.7 \approx 175$$

Now we could compose a scaled down version of our previous network. After several test runs we found that by adding a third layer and increasing the amount of nodes in each layer the final network consists of the hidden layers [625,300,100].

From some research on how to decide the number of hidden layers and the amount of nodes in each of them we have found that having the first hidden layer to be the size of the input layer times a factor between [1,2] together with a decreasing amount of nodes in the following hidden layers gives the best results.

As for learning rate, we tested different values than 0.001 (default in Theano), but no value were found that gave a better result. According to studies<sup>1</sup>, lower rates would increase accuracy, but would eventually become more costly, and higher rates would result in a steeper but generally lower accuracy curve.

---

<sup>1</sup><http://axon.cs.byu.edu/papers/wilson.ijcnn2001.pdf>

## 2 Two Representation

Representing large amount of data generated by a human player would be costly. Therefore, we decided that the ai solver in module four should generate both the states and the “correct” move for each state. Initially, we started to generate random states as features. The labels were generated by passing the features to the `expectimax` function which would return the estimated best move. A more suitable solution was to change the way we generated features. We went from generating random states to states that occurred in actual games of 2048 run by our algorithm in module four. This discovery made a huge difference in our results, since we now have consistency in our feature set. Our feature set contains a total of 245070 states and the moves made by our algorithm from 150 different runs.

### 2.1 First Version - 16 Nodes

To begin with, the only idea that would be intuitive was to represent the state with 16 input nodes. The representation of each node was  $\log_2(value)$ . A node have a value between range  $[0,12]$  since the highest tile in the feature set has a value of  $\log_2(4096) = 12$ . Figure 1 shows the representation of a state.

**11-9-7-4:0-0-2-3:0-0-0-0:0-0-1-0**

2048	512	128	16
		4	8
		2	

Figure 1: Representation Of A Given State

The colon divides each row in the representation. Before these values are sent into the input layer in ANN every value will be divided by twelve so the range will be between  $[0,1]$ .

### 2.2 Second Version - 12x16 Nodes

Since the input layer in the first version contained only 16 nodes, we wanted to divide the state into more nodes. The highest value of the tile in our feature set was  $2^{12} = 4096$ . Therefore, we can represents every tile with 12 values. 12 ones and no zeros would represent 4096 and eight ones and four zeros would represent 256. Note that this is not the binary of the tile value. This will give a finished preprocessed data with 192 input nodes where the range already is between  $[0,1]$ , since the values is restricted to be either one or zero.

## 2.3 Comparison

To find the method of preprocessing that gave the best results with our network, we ran each version 50 times and compared the highest tile for each run. We compared the two versions with a random player. We believe that the second is the better choice because the increased amount of nodes both in the input layer and the hidden layers. The results displayed in Figure 2 and Table 2 supports our theory.

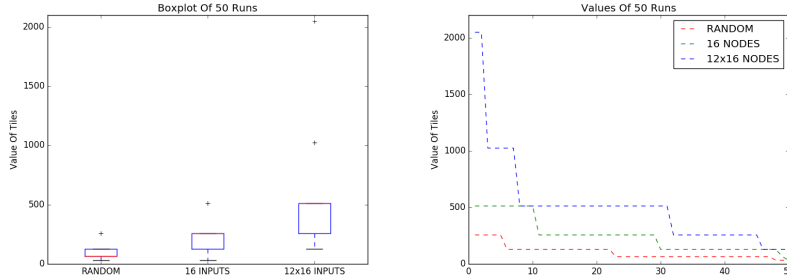


Figure 2: Accuracy and max values after 50 runs

Version	AVG-Tile	P - Value
Random:	103.04	-
16-Nodes:	250	$1.00 \times 10^{-8}$
12x16-Nodes:	514	$2.02 \times 10^{-9}$


Table 2: Evaluation of two ways of preprocessing

## 3 Analysis of Best Ann



Figure 3 shows two scenarios where our ANN does what we consider a bad move. The moves are based on the heuristics we chose in module four. In short, the heuristics tries to make sure that the highest valued tile is as close as possible to the upper left corner and that the other tiles are positioned in a snake-shape with decreasing values. In the first bad example the network does not fill the upper left tile and risks it being filled with a two- or four-tile. In the second example, the network does a move that results in that almost the entire top row, which is valued the most, is moved to towards the bottom, which is valued the least. Reasons for this might be that there are no similar states in the feature sets, or, there are similar states but with a slight difference so that the optimal move is different. A simple solution might be to generate a larger feature set. Also, since our implementation in module four is not perfect, our feature set contains runs where our algorithm makes bad choices and might lose before it

reaches the 2048 tile. We could improve our feature set by excluding features generated from games that did not win the game.


Figure 3 also shows a scenario where our network seemingly displays some intelligence. It “resists” the temptation of merging upwards to the valuable tiles to later be able to merge two pairs of tiles the same direction.

	64	8	32	!	64	8	32
	8	4	2		8	4	4
		2	2			2	2
			2			2	

(a) Left the critical top left corner empty

2	4	64	8				8
2		2	2				2
			8		2	64	8
			4	4	4	2	4

(b) Moved the high valued upper row tiles towards the bottom

256	128	64	8	256	128	64	8
32		32	8			64	8
						2	
		2					2

(c) Delayed the merging of the two eight-tiles to merge the two 32-tiles with the 64-tile

Figure 3: Examples of moves