# IT3105 Module 6

## Neural Networks for Game Playing

This report was written for the 6th and final module in the course IT3105 Artificial Intelligence Programming at the Norwegian University of Science and Technology. The task was to use supervised learning to train an artificial neural network to play the game of 2048, using the situation-action pairs generated by our Expectimax algorithm implemented to solve the 2048 game in module 4 earlier in the course.

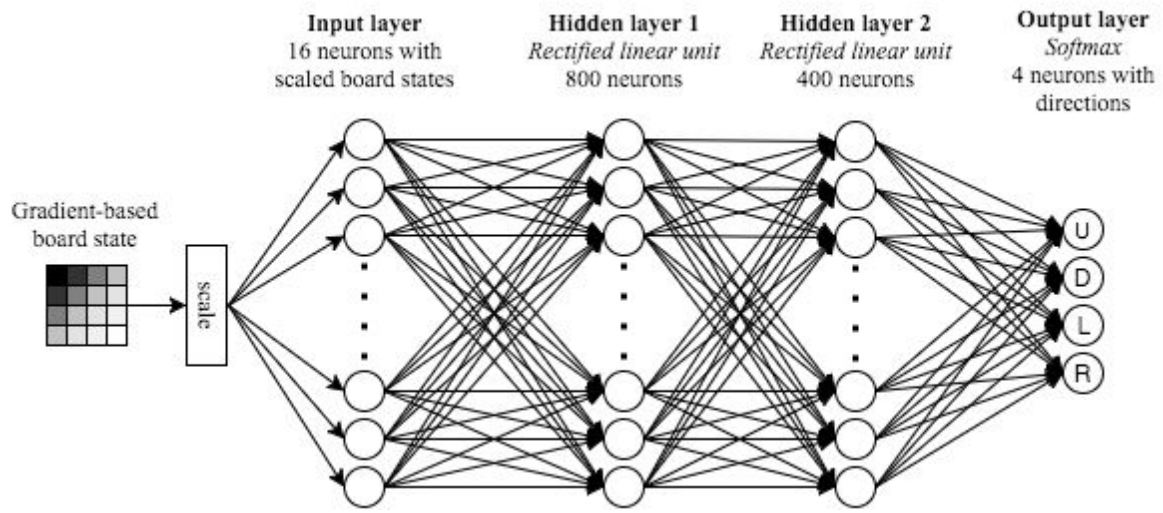
## Design choices

When deciding on the architecture of our artificial neural network, several key factors played a significant role. We knew that there would be at least sixteen units in the input layer, corresponding to the value of each cell of a board state, and depending on how we chose to represent our test cases there could be a few more units corresponding to other features, such as the number of free cells, or whether or not the highest tile was positioned in a corner. We initially tried using a single hidden layer with number of hidden units ranging between $[num_{in},\ num_{in} + num_{out}]$ , and tried every variation with all the different activation functions; *hyperbolic tangent, sigmoid* and *rectified linear units.* We eventually realized that this number of hidden units was not enough for the neural network to capture all the different patterns and actually learn anything significant. We therefore decided to try to use the neural network which yielded best performance in module 5, namely a 2-hidden layer neural network with 800 and 400 hidden units in each layer, and using rectified linear units as the activation function in each layer. Against all rules of thumbs for the sizes of hidden layers with respect to the input and output layer sizes, this topology actually seemed to yield substantially better results than previous topologies. We continued to test this architecture with both the sum of squared errors, and categorical cross-entropy as the cost function used in our stochastic gradient descent algorithm, and strategically lowering the learning rate from $10^{-1}$ to $10^{-4}$. In the end, the architecture which continuously yielded best results was a 2-hidden layer rectified linear units neural network with 800 and 400 units respectively, using categorical cross-entropy as cost function with a learning rate of $10^{-2}$. This neural network is capable of learning complex patterns in our reasonably large datasets because of the amount of hidden neurons it possesses.

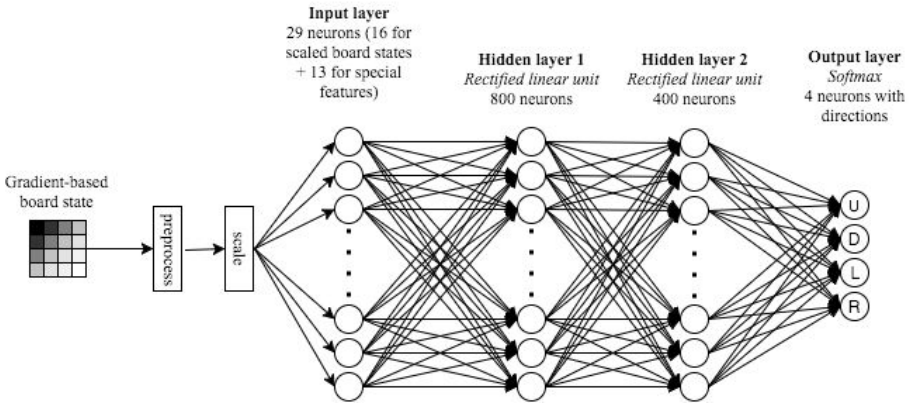
## Representations

We tried several different representations during this assignment, ranging from a lot of preprocessing to simply converting the board to a flat vector. In this section we'll describe the least complex form of preprocessing and the most complex form of preprocessing.

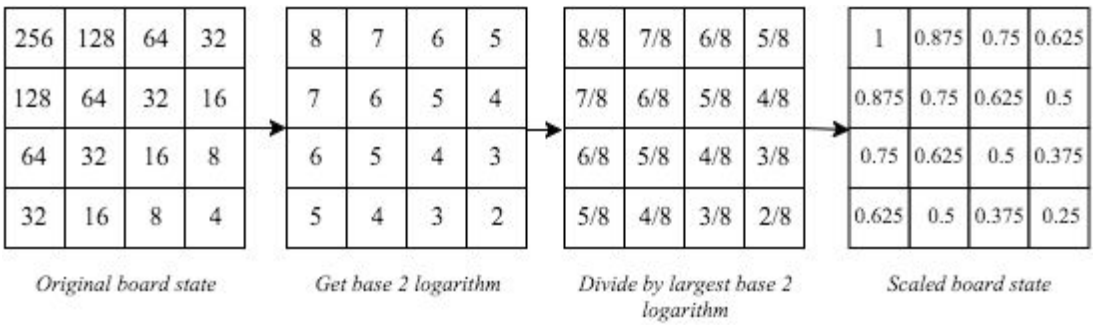
The least complex form of preprocessing is simply the use of a 16 dimensional vector to represent the board state. In addition to this preprocessing we only generated test cases up to the tile 1024, so that they all would be relevant in terms of helping us get a decent score. The justification for this is that with a good architecture we hoped that the network would be able to recognize the patterns by itself without us confusing it by overloading it with data. In addition to this preprocessing we used a scaling function that will be mentioned at the end of this section.




The most complex form of preprocessing resulted in a 29 dimensional vector. The first 16 bits were the same as in the least complex form. Followed by a bit to indicate whether the highest tile was in the bottom right corner (in compliance with the gradient). The next two bits indicate whether the topmost row and leftmost column were full. Then two numbers indicated the number of merges resulting from a horizontal move and a vertical move respectively. Four bits then determined the utility resulting from the moves up, down, left and right. These were scaled by dividing them all on the biggest value. Then another four bits determined the number of free tiles the resulting boards from moves up, down, left and right would have. These values were also scaled by the largest number.




Both methods used scaling for the flat representation of the map. As shown in the figure below all numbers were first divided by their base 2 logarithm, then all the tiles were divided again by the largest base 2 logarithm.




We fed the results from the two network into the welch test. Substituting the first for the random player and using the second as the second parameter

```
Average random tile: 170.88
Average ANN tile: 198
```

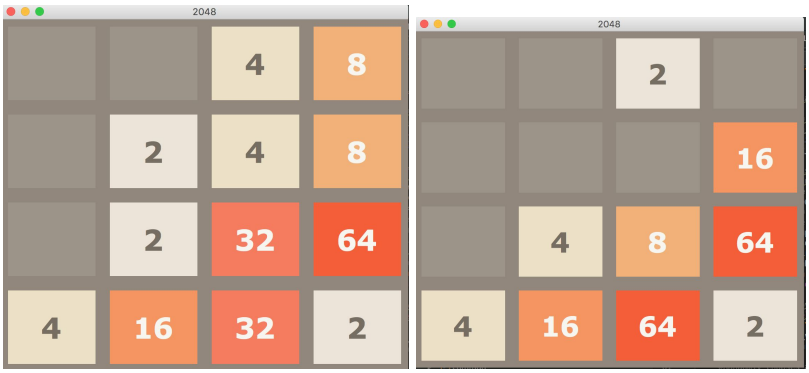
As the results show the second network gets a better score than the first.

## Analysis of game play

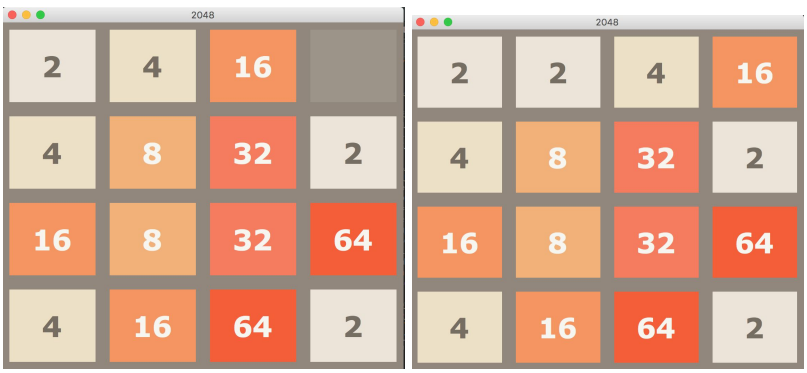
The artificial neural network quickly learns that it is beneficial to keep higher valued tiles in the bottom right corner. The training set we produced using the Expectimax algorithm uses a heavily influenced gradient heuristic in order to gather higher valued tiles in one corner, such that we increase the monotonicity of the board.

On the other hand, since the artificial neural network has only trained on producing an output given a board configuration and some special features, it does not possess the knowledge to actually assess the score of future moves. This leads to the network sometimes being stuck

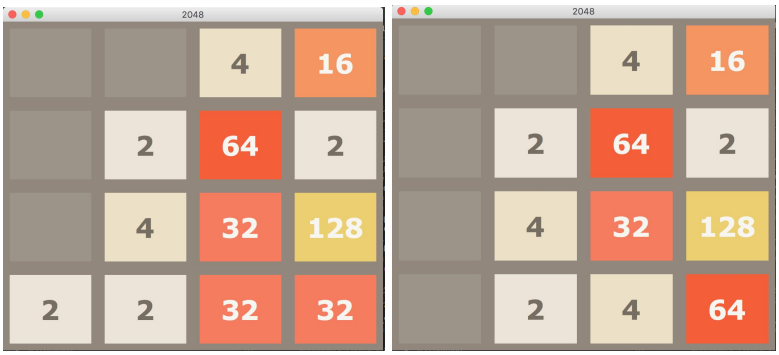
with lower-valued tiles in a corner if a lower-valued tile spawns there, and it does not attempt to fix such a state by looking K steps ahead, such as the Expectimax algorithm would do.




Here, the neural network is already stuck with a lower-valued tile in the bottom right corner, which is unfortunate. However, it decides to move down, which is a wise move resulting in the merging of two 8-tiles, two 4-tiles, two 2-tiles and two 32-tiles. It also results in a rather monotonic board, except the unlucky two in the corner.




Here, the network performs an unwise move - instead of moving down resulting in the merging of two 32-tiles and two 8-tiles, with a potential merging of two 16-tiles and two 64-tiles in the subsequent move, it decides to move right, resulting in zero merges.




In this situation, the neural network conforms with its knowledge of moving higher valued tiles into the bottom right corner is beneficial, but it had the chance to create a 256-tile in two moves by either first moving down then up, or first moving up then down, and then right.