# Project 3 - Module 5

by Fredrik Borgen Tørnvall and Kristoffer Dalby

## Deep Learning for Image Classification

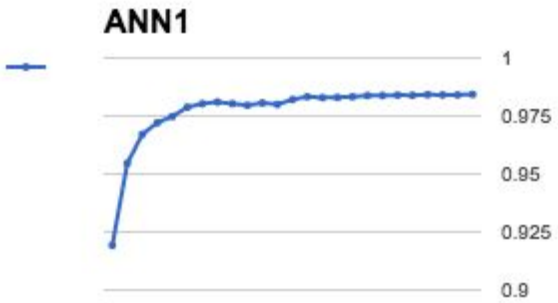
For this project, we were asked to implement an artificial neural network (ANN) using Theano. The ANN should be able to interpret handwritten numbers from 0 – 9. The ANN was trained using the mnist data set and evaluated by the mnist training set, mnist test set and an unknown set on the demo day.

### The Artificial neural networks

Five different ANNs were constructed using Theano. We did several test with different activation functions, different sizes of hidden layers, different number of hidden layers and we tried to change learning rate and number of training epochs as well as change the batch size when training. For backpropagation we use RMSprop for all ANNs. According to Geoffrey Hinton (https://www.youtube.com/watch?v=O3sxAc4hxZU , http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf) and Alec Radford (https://www.youtube.com/watch?v=S75EdAcXHKk) RMSprop will improve accuracy and speed up learning compared to e.g sgd.

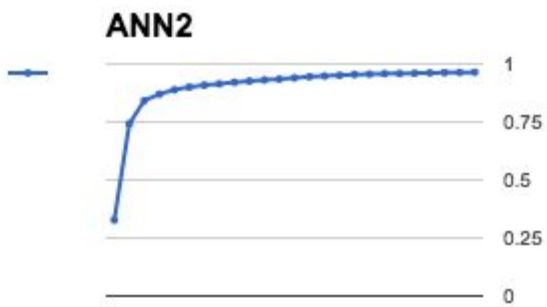
ANN 1
25 epoch
RMSprop is used with, lr = 0.001,  rho = 0.9 and epsilon = 1e-6
Hidden Layers: Two hidden layers
Size of layers: [784, 625, 625, 10]
Activation functions: [rect, rect, soft]
Result:

| | |
|---|---|
| mnist training: | 99.9% |
| mnist test: | 98.3% |
| demo: | 97.0% |




ANN 2
25 epoch
RMSprop is used with, lr = 0.001, rho = 0.9 and epsilon = 1e-6
Hidden Layers: Three hidden layers
Size of layers: [784, 620, 620, 10]
Activation functions: [sig, sig, soft]
Result:

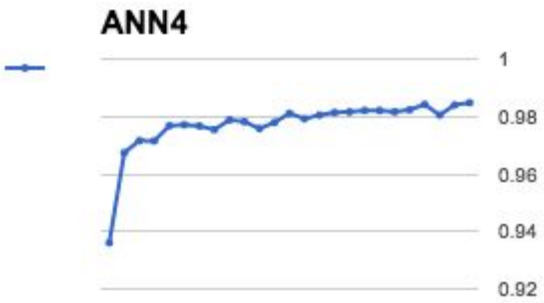
| | |
|---|---|
| mnist training: | 96.95% |
| mnist test: | 96.44% |



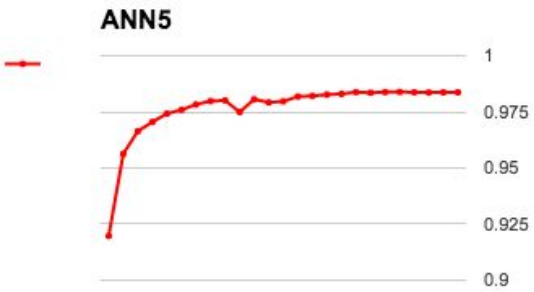
demo: 98.0%

ANN 3
25 epoch
RMSprop is used with, lr = 0.001, rho = 0.9 and epsilon = 1e-6
Hidden Layers: Two hidden layers
Size of layers: [784, 620, 620, 10]
Activation functions: [rect, rect, rect]
Result:
mnist training: 9.87%
mnist test: 9.80%
demo: 10.0%


ANN 4
25 epoch
RMSprop is used with, lr = 0.001, rho = 0.9 and epsilon = 1e-6
Hidden Layers: Two hidden layers
Size of layers: [784, 620, 620, 620, 10]
Activation functions: [rect, rect, rect, soft]
Result:
mnist training: 99.99%
mnist test: 98.36%
demo: 96.00%




ANN 5
25 epoch
RMSprop is used with, lr = 0.001, rho = 0.9 and epsilon = 1e-6
Hidden Layers: Two hidden layers
Size of layers: [784, 620, 620, 10]
Activation functions: [rect, rect, soft]
Result:
mnist training: 99.99%
mnist test: 98.43%
demo: 98.00%



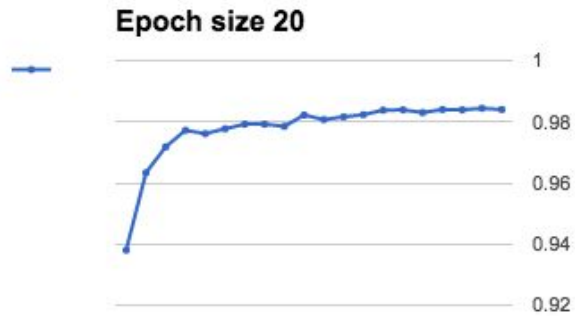

## Accuracy

From the ANN2 and ANN5 graph we see that the initial value for ANN2 is significantly lower than that of ANN5. However they both end up performing quite well. One the demo set they both score 98%, but we see that ANN5 performance ca 2% better on both the mnist sets. In ANN5 there is a dip in accuracy on the 10 epoch. We are uncertain why this is, but it can have something to do with batch size when training and redundancy in the training set. It corrects its predictions and ends up with an accuracy of 98.43% for the training set.

## Training

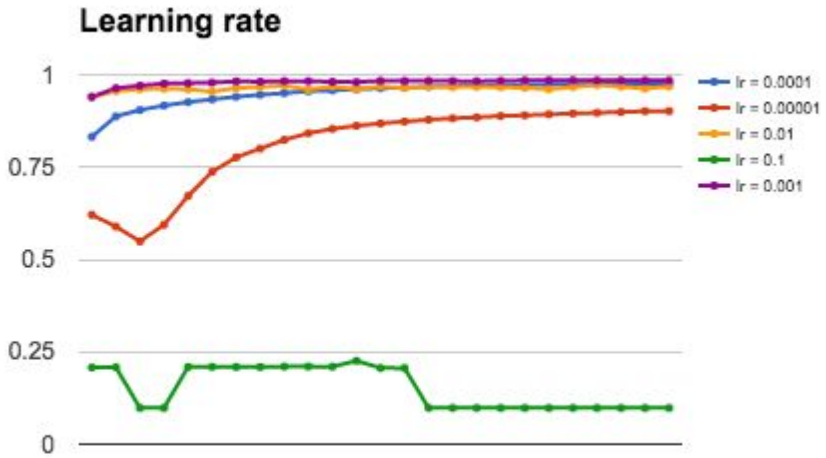
ANN 5 got the best accuracy in our testing when we kept the learning rate, epoch and batch size constant. Testing with batch sizes 128, 256 and 512 only yielded 0.11% accuracy difference.
Testing with different epoch size.






Keeping all the settings from ANN5 changing only the epoch size, we see that the accuracy is over 98% in all cases.

Testing with different learning rates.
Hard to see from the graphs, but the learning rate which resulted in highest accuracy was lr=0.001. It has a good balance between speed and accuracy. With all other variables equal, 98.46% on the training set.



# Project 3 - Module 6

## Deep Learning for Game Playing

## Design

When designing the Artificial Neural Network (NN) in this assignment, we went through a lot of iterations with tweaking, testing and failures until we reached a configuration which could pass the Welch test with a satisfiable score. The initial structure of the code was copied from module 5 and changed. It is important to note that there were orders of magnitude more testing than we were able to describe in this section, almost all of our choices is based on testing and discussions with other students.

When working towards the design of the NN, the initial idea of the layers, activation function, learning rate and momentum was copied directly from the module 5 implementation. That resulted in a network with two hidden layers with rectify and softmax as functions. It did not take a lot of testing until we found this approach to not be optimal for the problem at hand. Instead of using the layer sizes from the last module (784, 620, 620, 10), we used the sizes (16 8 8 4), as initial testing gave us the idea that reducing the hidden layers to the half of the input would yield positive result. This was quickly disproved.

We therefore started to do a series of testing where we tweaked the size of the layers and number of epochs that we ran when building the network. After a lot of testing that yielded little to none positive improvement, we started to turn most of our focus over to the generated training data and the representation of the board within the neural network. After some rounds with different approaches on massaging the data, we started to drastically change the way we structured the hidden layers and the sizes. As mentioned below, we started to experiment with different input sizes as we processed the board in different ways, in addition we expanded the first hidden layer greatly with the idea that it may help the NN to pick up more details from the training and input data. The network was now (48 1024 8 4).

This approach helped us into a good track where we achieved some good scores on the Welch test. The NN now played a little better than the random. After a lot more testing, we decided to remove the second hidden layer.

In the end, our neural network ended with a learning rate of 0.001, the layers 48, 2048, 4 (input, hidden, output), RMS propagation, rectify for the hidden layer and softmax for output.

## Representation of the data

During the testing of different representations we landed on two different possibilities that we would like to elaborate further. The first representation is a combination of the board (16 numbers), the gradient heuristic score for one corner (16 numbers) and a score ranking possible mergers (16 numbers). The second representation was an experiment where we

tried to represent the board as the possible permutations from the current state, after executing a move, summing up to 48 numbers.

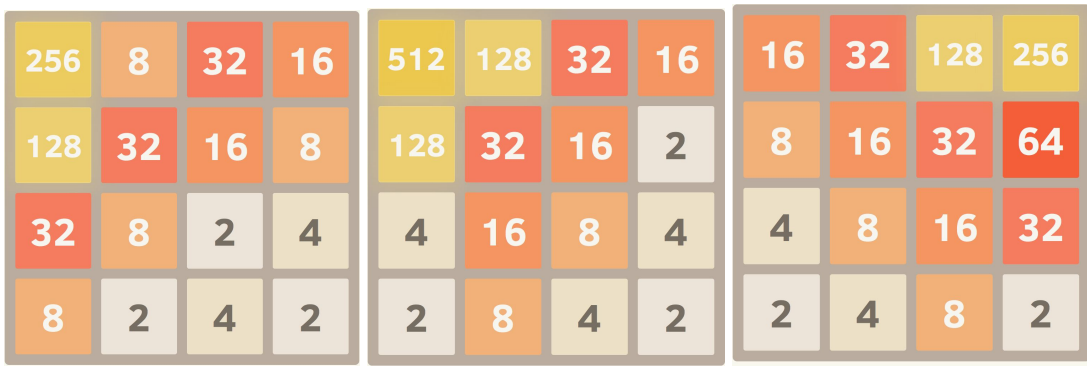
To generate training data for the network, two approches was selected, first we generated multiple sized data sets from a very good AI (https://github.com/nneonneo/2048-ai) and secondly using our own AI from module 4. The difference in the dataset is that the first dataset is based on a AI which gets 16384 most of the times and uses a mixed, very advanced heuristic. The dataset generated from our AI is based on a simple, one corner gradient heuristic. Both datasets has board states with 1024 as the highest tile as we did not expect to achieve a higher tile after initial testing of the neural network. All the training data is represented by a value between 0 and 1.

## Gradient based representation

To construct the gradient based representation, a lot of trying and failing was done, together with discussing possible combinations with each other and other groups. We started by testing out how the NN worked with representing the board as factors between 0 and 1, after poor results, we added a mergeability score. This score is calculated by looking at all the neighbours the tile can merge with, and adding 1 for each possible merge and then dividing it on the amount of neighbours.
After achieving better score, we desiced to test if a board representation using a heuristic would help. As our module 4 approach was gradient, and it achieved a good score, we continued to use it.

After a lot of more testing, failing and tweaking, we achieved a satisfiable average Welch score.

However, even though the network achieved satisfiable on the Welch score, the network was not that smart. To analyze how the NN played we visualized the game. Below a representation of multiple end games is displayed.

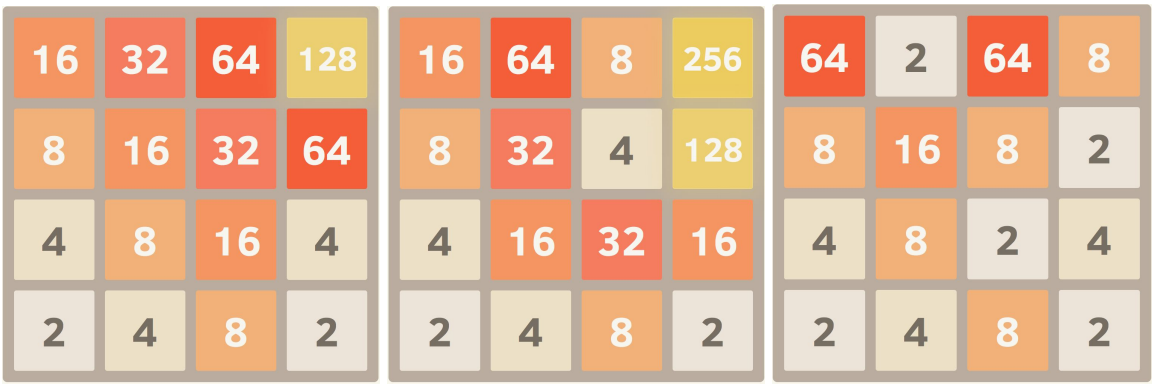



What was observed while the NN played the game was that the gradient was learned. The thing that was different from the Expectiminimax AI is that the NN was not able to get out of tricky situations, like the 8-tile in the upper left corner. This quickly resulted in the NN misplacing tiles it could not recover from, which mostly ended in a 256 as the highest tile. A 1024 tile was also achieved on multiple occasions.

### Move permutation based representation

To try something completely different from the other representation we looked at trying to teach the NN what the next good move may be. This was achieved by taking each state/board in the dataset and perform a move operation in each direction. This will ultimately result in a input data of the size 64.

In addition to using all four moves to generate a representation, we experimented with using just three moves (by removing down), as neglecting a move is a viable tactic for the actual game. This did not result in any notable difference from all the four moves.

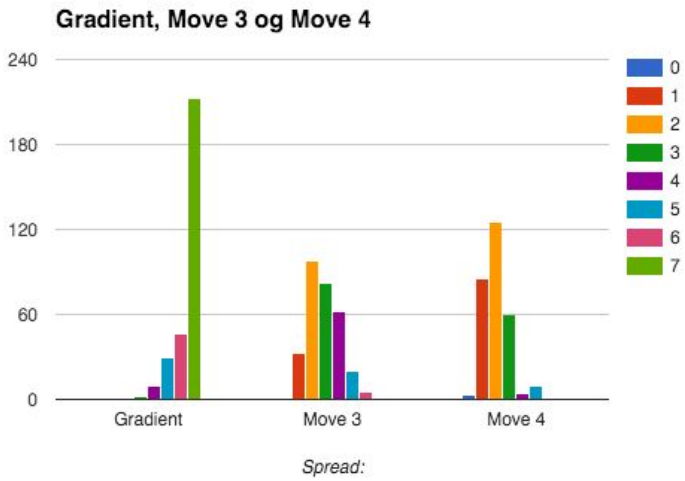



Above, a few end games from the three moves version of the represented is presented. To the left, we can see that the NN tries to build a gradient towards a corner of the map, this is interesting as there is no part of the preprocessing that gives this heuristic. The dataset (nneonneo) does however use some gradient as part of its multi-heuristic strategy. The problem is that this NN does have an even bigger problem with getting out of tricky situation, which again results in even worse score.

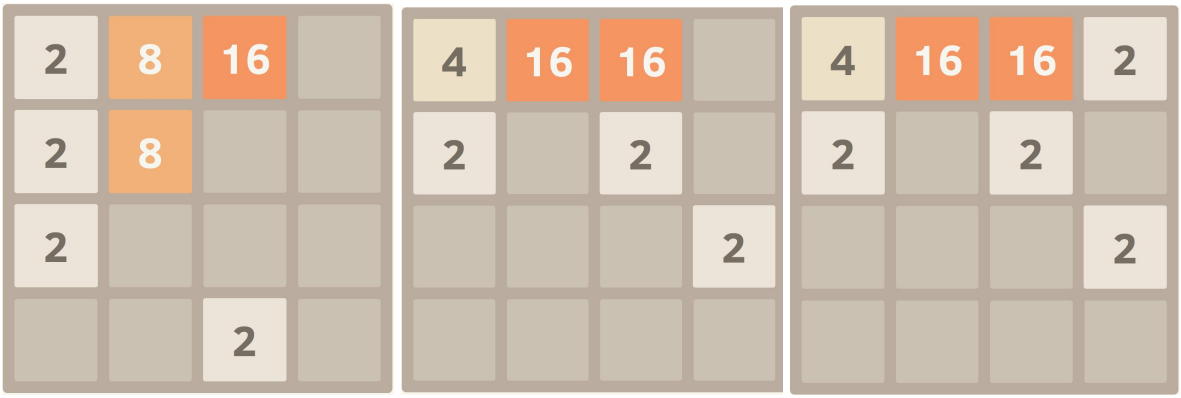
## Analysis of Artificial Neural Network

| | Gradient based | Move 3 based | Move 4 based |
|---|---|---|---|
| Average tile | 206,65 | 143.85 | 133.81 |
| Average random tile | 100,29 | 99.81 | 101.22 |
| Average Welch | 6,63 | 2.84 | 2.106 |

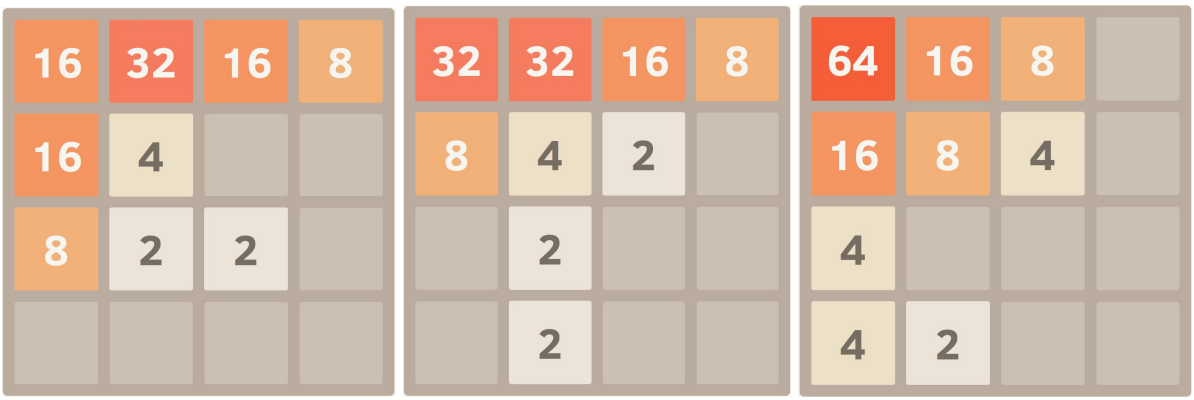
The dataset used in the test run is based on nneonneo's AI and contains 54741 board states represented as a board/value.
After running 300 runs (50 NN and 50 random) with both networks and the different representations, the results on the top and the right were received.



As we can see from both the table and the diagram, the gradient outperforms the different move representations. But even though the Gradient representation was better than the move representation, it still did some move that where not so intelligent.
For example, below is a scenario where the NN has the possibility of getting the board back into a good gradient state, but instead ends up with the big tiles in the middle by not moving to the left in the middle state.




But, in some scenarios, the NN was able to clean up the board and put it back into a nice gradient shape. Note that the board state to the left below is more than one move from the others.




# Conclusion

The Artificial Neural Network has some kind of intelligence, it performance at least twice as good as a random player on average. There is however, some more configuration of the network and dataset that we wished we had explored further. This includes making every state in the dataset unique, testing with more layers and more sizes and how many epochs we run.