Module 5 - Deep Learning for Image Classification

Aleksander Skraastad
Håkon Ødegård Løvdal

November 23, 2015

1 Introduction

The purpose of this report is to describe our work with module 5 in IT3105. During this module, we have constructed and implemented several different ANNs in Theano. This report will elaborate details about the five best ANNs and their performance. This will include different combinations of hidden layers, size of hidden layers and activation functions.The result of this work was an ANN capable of classifying hand written digits from the MNIST data set.

2 Network design

The main idea when constructing our ANN was to ensure an easily configurable network, where the ANN would construct itself based on two arrays and a configuration object passed to the ANN upon instantiation. The two arrays describe how many nodes there are in each layer, and the second array declares the activation functions respectively. These two arrays therefore represent the topology specification of the network. The configuration object contains a detailed description of how the network should behave. This included elements such as error function for back-propagation, training batch size, learning rate and much more. Most of these specifications have default values if they are not provided. A deeper elaboration about different configurations tested follows in the next section.

2.1 Configurations

In the following list, the input and output layers are also represented, as it is important to highlight what activation functions were applied to which layer in our configuration. These configurations were among the most successful we generated. All these configurations use categorical crossentropy as their error function.

1. {784, 1156, 784, 10}, {RLU, RLU, RLU, Softmax}, Learning rate: 0.001
2. {784, 1568, 784, 10}, {RLU, RLU, RLU, Softmax}, Learning rate: 0.001
3. {784, 620, 10}, {RLU, RLU, Softmax}, Learning rate: 0.001

4. {784, 620, 10}, {Sigmoid, Softplus, Softmax}, Learning rate: 0.001
5. {784, 392, 10}, {Softplus, Softplus, Softmax}, Learning rate: 0.005

2.2 Error function

During the initial testing, categorical crossentropy seemed to produce a better result over time. On average it gave a result with a slightly higher precision than SSE. These results correspond to the findings by Golik, Doetsch and Ney[1], as we use randomly initialized weights. Therefore we chose to do a more thorough test on configurations using this error function.

2.3 Flavour of back-propagation algorithm

The reason for using RMS-propagation is that it uses the magnitude of previous gradients to normalize the current gradients, by keeping a running average root-mean-square (hence RMS). This has the advantage of working nicely with mini-batch training, instead of having to use full batch training. The end result being faster learning rate, as it is dynamically adjusted.

3 Comparing five ANNs

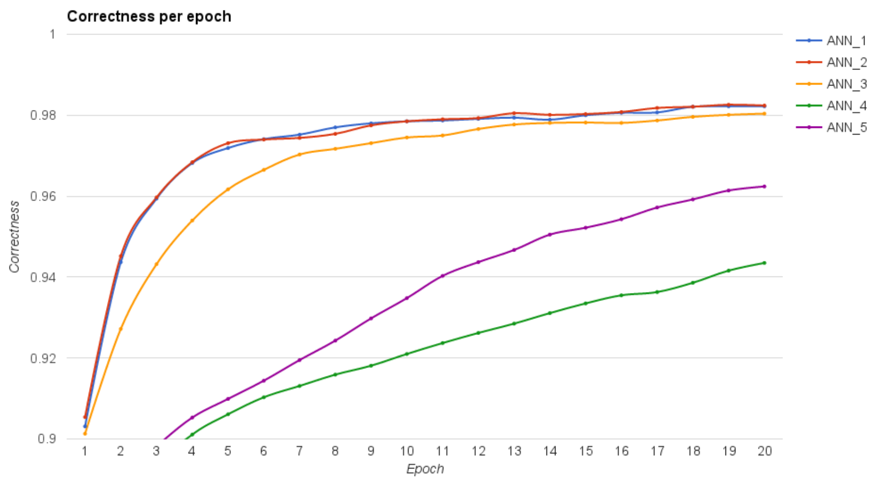


Figure 1: Correctness per epoch

In order to evaluate the different network configurations, all the different configurations was run for 20 epochs, and all statistical data of interest were collected. This data-set included correctness and error rate (crossentropy) per epoch. Also the results from running a classification on the entire MNIST training- and testing-set were included. The correctness data (figure 1) visualizes that ANN_1 and ANN_2 converge quickly after approximately 14-15 epochs. Convergence normally occurs just above 0.98. The error rate data for the same two networks show a steady decrease in the error rate throughout every epoch.

[1]Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison (2014)

We also noticed that ANN_3 continues its increase in correctness after epoch 14-15. Running the entire MNIST training- and testing-set on ANN_1 gives a classification score of 0.9993 and 0.9822 respectively. ANN_2 gives 0.9994 and 0.9824. Due to the similarity of ANN_1 and ANN_2, along with the fact that ANN_3 still seems to increase, we chose to run a longer test on ANN_1 and ANN_3. Each of these networks were run 50 times, where each run created a new instance of the network and trained it for 20 epochs. This test proved that even though ANN_1 had a higher correctness than ANN_3 did after 20 epochs, it gave a lower correctness than ANN_3 on average. ANN_1 seemed to be very good on the MNIST sets, but when testing on the *Demo100*-set handout, ANN_3 was always slightly better. This might be an indication that the training of ANN_1 has been overfitted on the MNIST sets. Since this network has a larger topology, we assume that it may prove better if it was fed with a larger training set than the MNIST sets.

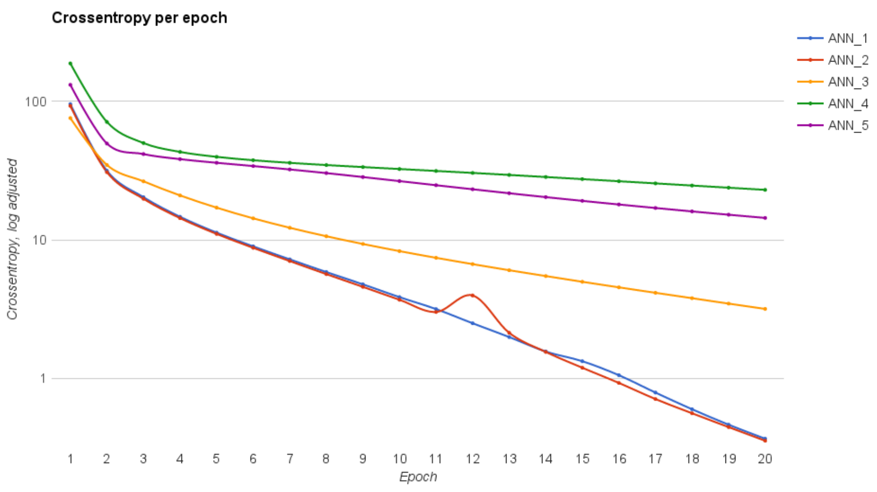
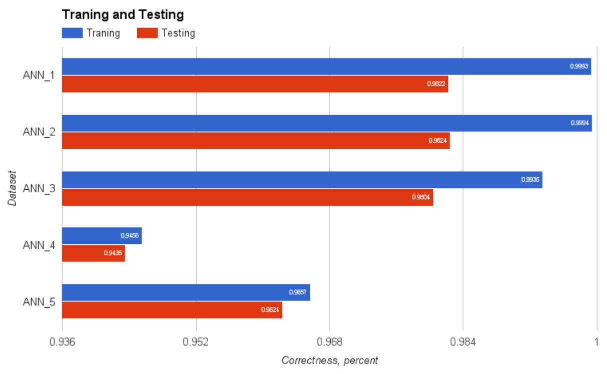
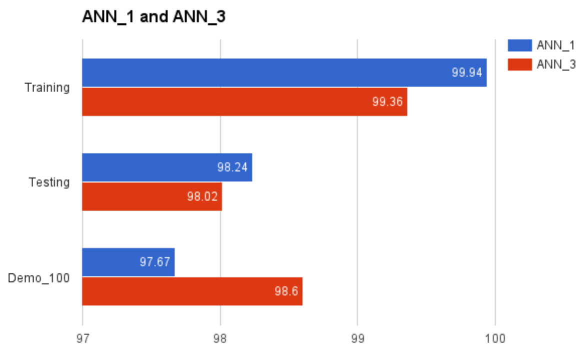


Figure 2: Error rate per epoch

The combination of these two tests prove that ANN_3 seems to be the best network for this exercise. Running the longer evaluation on ANN_1 and ANN_3 differentiated them further. Both networks converge around 0.98 and produces almost the same result, but ANN_3 proves to yield more correct classifications in average. In addition, ANN_3 has significantly less computational complexity than ANN_1.



(a) Training after 20 runs



(b) Training after 50 runs

Figure 3: Training data