ASSIGNMENT 5

TDT4171 - Methods in Artificial Intelligence

Written by:

Lars Liverød Andersen
Håkon Ødegård Løvdal

Spring 2016

NTNU
Norwegian University of Science and Technology

1 Implementation

In order to get a working ranking algorithm we had to extend or modify the code in both the `backprop_skeleton.py` (1) and `data_loader_skeleton.py` (2). We did the entire exercise with pair programming. In order for us both to completely understand the problem at hand we went through one method at the time when implementing it. We cannot clearly tell who implemented which function, since it all were implemented by sitting next to each other and discuss during the implementation. This approach were easily applicable to this problem, due to the scope of the code being so small. The modified or extended code is listed below respectively:

1. `count_misordered_pairs`
   In this method we only iterate every pattern a, b and propagte their features. Due to to patterns already being sorted, the `if A>B` check in the pseudo code is not necessary. The only check required is whether or not activation for $a < b$.

   `train`
   Here we iterate the iterations passed to the function as a keyword argument. For every iteration, we propagate every pattern pair a, b, then we backpropagate. Every iterations error is appended to a list and returned (but we never use this data)

   `compute_output_delta`
   Simply implemented by using the supplied `log_func` and `log_func_derivative` and equation 1-3 in the exercise text.

   `compute_hidden_delta`
   As the previous function, this is implemented by simply using the equation 4 and 5 in the exercise text.

   `update_weights`
   This function is implementing equation 6. First the input weights is updated based upon the input activations. Thereafter the output weights is updated based on the hidden activations.

2. `run_ranker`
   Adds a two calls to `generate_pairs` and appends test/training errors to separate list in order for us to be able to plot the error history.

   `generate_pairs`
   This function exploits two built-in functions in Python, namely `sorted` and `reversed` to create a list of data instances, sorted by rating. Then it creates pairs with a double for-loop and only appends instances where the rating differs.

2 Results

In figure 1 the total average when running the neural net five times, with 20 epochs each. In general we can see that the error rate is decreasing fast, until approx. 15 epochs. At this point the error rate stabilizes around 78% correctness. This shows us that running the network more than 15-20 epochs will not have any positive affect. Most likely, running the network further would yield overfitting on the training data, and therefore generate lower performance on the testing data.

The graphs does not include any surprises in our opinion. They are as expected. Usually, we are accustomed to having a lower error rate, but since the exercise text stated that the expected correctness would be approx. 75% this did not surprise us.

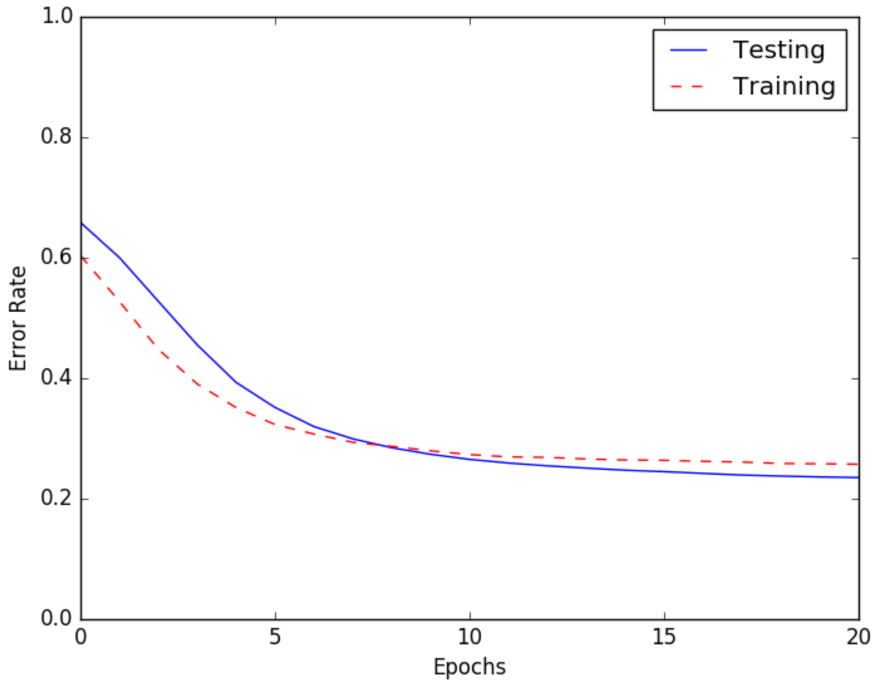

Figure 1: Average over 5 iterations