

Lab assignment 1

Artificial neural networks using back propagation

Examiner	Email
Ning Xiong	ning.xiong@mdh.se
Students name	Email
Robin Calmegård	rcd10002@student.mdh.se
Dennis Stockhaus	dss10002@student.mdh.se

Abstract

This report is written in relation to Assignment 1 in the course 'DVA427 – Learning Systems' at Mälardalens University. The assignment is to develop a competent neural network to predict 'etch-depth' for a Reactive-Ion-Etching (RIE) machine based on predefined selected sensor signals gathered in .txt files. The Neural-Network may be implemented with any programming language as well as framework.

In continuation this report will describe and summarize the following information: the structure of neural-network selected, the learning algorithm used, the evolving of performance (errors with iterations), performance on training data and performance on test data

Implementation

The software for the assignment is built with C# programming language and uses predefined graphical libraries offered in Microsoft's Integrated Development Environment (IDE) Visual Studio 2015 for the user interface (UI).

User Interface

The interface is built to dynamically create, load and save Neural-Networks and configurations, for a wider range of testing. The UI also has a graph component that better visualize the improvement of training and test performance of the neural-network in real time.

Artificial Neural Network

Structure of neural-network selected:

The structure selected for the artificial neural-network is a standard 'Multilayered Perceptron (MLP) – Feed Forward Artificial Neural Network (FFANN)', with one hidden layer as well as one input and output layer. The solution implemented can handle several different configurations with different amount of; input, hidden, output nodes. But for this assignment it is specifically configured with 3 inputs, X hidden and 1 output.

All data for the network is normalized by a Gaussian Normalization function, therefore the computation of data is suitable to be activated with a Hyperbolic tangent function for the input to hidden layer and uses a simple LogSigmoid function for the hidden to output layer.

Learning algorithm used:

The learning algorithm used with this neural-network is a variant of the Backpropagation Algorithm:

```
Repeat
  Shuffle TrainingData sequence (helps with BP algorithm getting "stuck")
  For each training example in data
    1. Compute and store output from training example
    If (t != 0)
      2. Compute error term for output layer
      3. Compute the "error" term for hidden layer
      4. Compute Weight-Update-Gradients
      5. Update Weights
  Check if (Epoch reached or acceptable Accuracy(overfitting stop)) then break iteration.
```

The Backpropagation algorithm has some known issues where the performance of the network might get stuck at local minima, for this we have a function that shuffles the order of training examples between each iteration (although for this application it hasn't been proven to be a very efficient countermeasure).

Results

Performance (error over iterations)

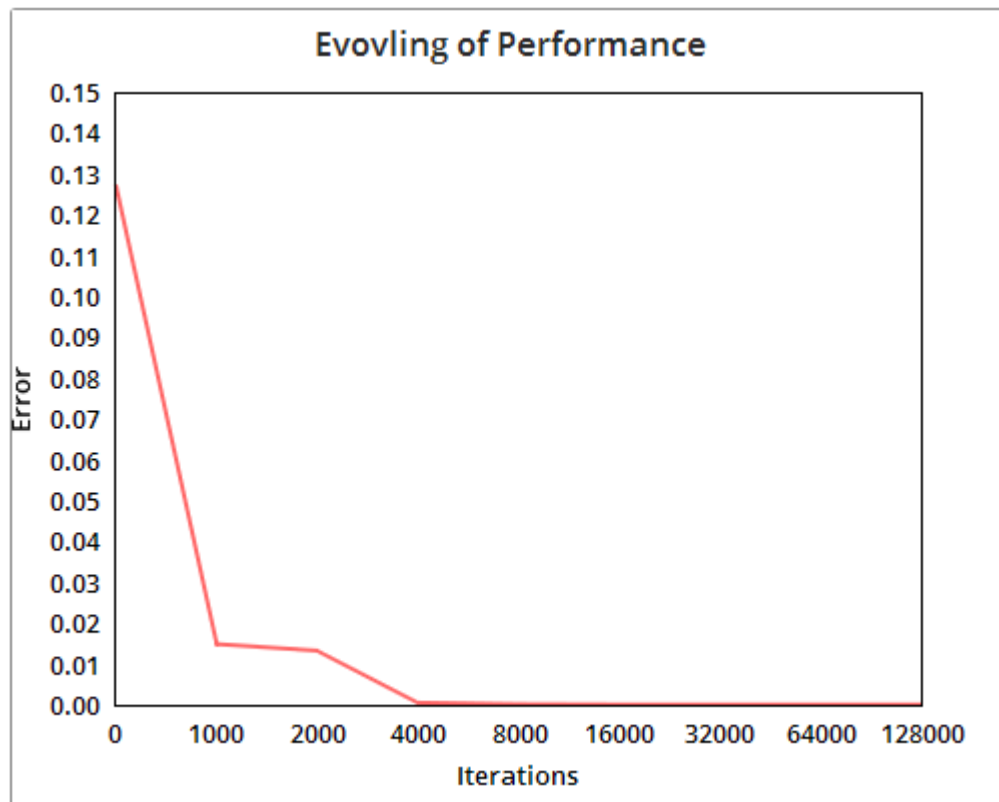


Figure 1. Evolving of Performance Graph.

In the graph above (Figure 1.) the Neural-Network used is configured with:

- One input layer – with 3 nodes

- One hidden layer – with 75 nodes
- One Output layer – with 1 node
- A learning rate of 0.00065 (value that affects the gradients which the weights will be updated with)
- A momentum of 1.1 (value that affects the general speed of learning)

As shown in the graph (Figure 1) the error value (sum of difference between output and target-output squared) becomes smaller as the iterations go on. It is also visible that the improvement of performance gets less significant as the iterations get higher, this is because of the importance of the error-value in the update-weights function, where the rule is: the smaller the error the smaller the adjustment of the weights. For most cases the neural-network can be assumed to merely approximate an error-value of zero but never actually reach it.

Training & Test Performance

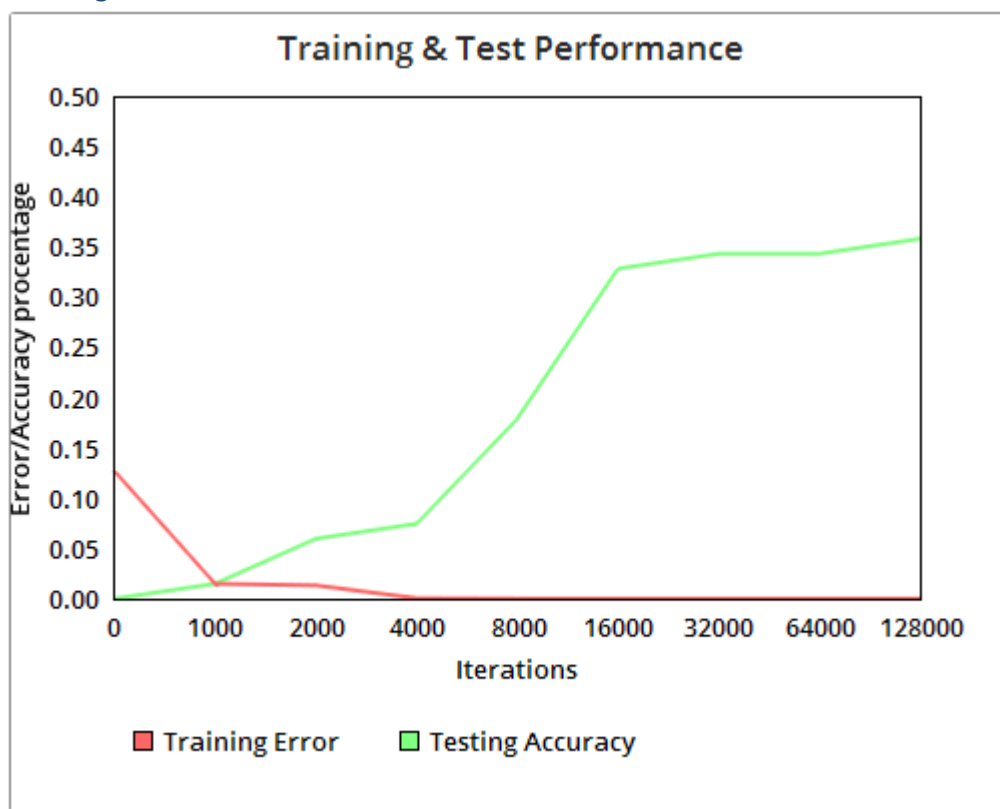
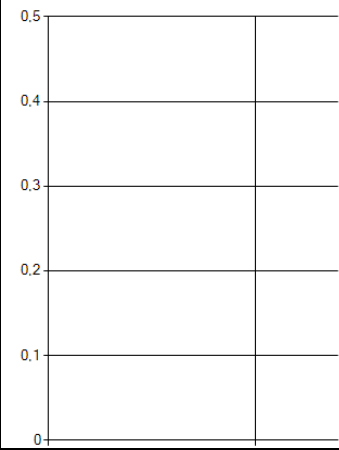
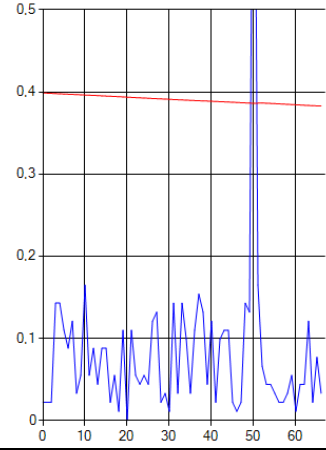
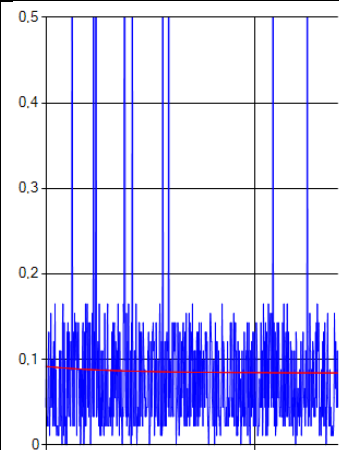
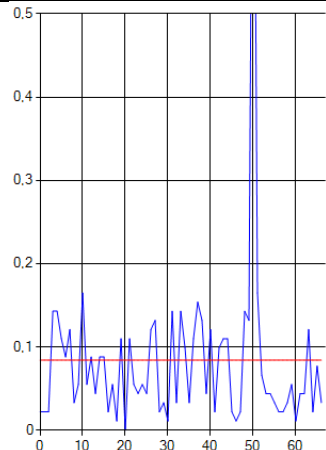


Figure 2. Training & Test Performance Graph.

In this graph (Figure 2) both the 'Performance of Iterations' and the 'Accuracy of testing' is displayed. Vertically the values for the red line shall be read as decimal values just as Figure 1, where for the green line it should be read as a decimal representation of percentage (percentage of correct answers) where 0 – 1 equals 0 – 100%.

As can be seen in the graph (Figure 2), when the iterations gets higher, the error value for training examples becomes smaller as well as accuracy for test examples gets better. This proves that the implemented neural-network is competent at learning and given an arbitrary large amount of training should be able to accurately predict answers to unseen problems.

Shown below is a table containing screen capture images of the applications graph component during runtime. Each graph has a red and blue line, where the red line is the outputted answer from the neural-network and the blue line is the correct/expected answer. For each iteration there is an image showing a comparison of accuracy for both training and testing.

Iteration	Training Performance	Test Performance
0		
1000		
2000	