# NNDT –
# A Neural Network Development Tool

version 1.1

## User's Guide

B. Saxén and H. Saxén
Department of Chemical Engineering
Åbo Akademi University
Biskopsgatan 8, FIN-20500 Åbo, Finland
E-mail: bjsaxen@abo.fi, hsaxen@abo.fi

January 1995

# 1 Introduction

The use of neural network models normally involves specification and analysis of a large number of parameters as well as large data sets. The availability of software with an interactive graphical user interface facilitates the development of such models. A program for network training and evaluation, the Neural Network Development Tool (NNDT), has been developed. The program includes models for feed-forward and (discrete time) recurrent neural networks and routines for graphical presentation of the results. The user-interface, which runs under Microsoft Windows, is developed with MS Visual Basic 3.0 professional edition while the network calculations and the training method are implemented in C and compiled to dynamic-link library (DLL) routines. In this manual, the features of NNDT (version 1.1) are presented; the network models are described and the user interface is illustrated with two examples.

# 2 Network Models

The program implements feedforward and recurrent multilayer perceptron (MLP) networks [1]. In the model for feedforward networks (Figure 1a), there is an input layer, possible hidden layers, and an output layer. Up to three hidden layers and up to 15 nodes per layer can be used. The elements, or nodes, in the input (lowermost) layer only receive the input signals and distribute them forward to the network. In the upper layers, each node receives a signal which is a weighted sum of the outputs of the nodes in the layer below. To node $i$ in an upper layer, $l$, the *total input* $x$ is given by

$$x_i^l = \sum_{j=0}^{N^{l-1}} w_{ij}^l y_j^{l-1},$$

where $N$ is the number of nodes, $w_{ij}$ is the weight for the connection between node $j$ and node $i$ and $y_j$ is the output of node $j$. Each node $i$ has a bias, represented by the weight $w_{i0}$ from a node with a constant unit activation ($y_0 \equiv 1$). The weights and the bias terms are estimated by the training method described in Chapter 3.

The output of a node in a hidden or output layer is obtained through a nodal activation function, which uses the node input as argument. Four different activation functions can be used; the (standard) sigmoid from 0 to 1

$$y = \frac{1}{1+e^{-x}}, \tag{1}$$

the symmetric logarithmoid [2]

$$y = \frac{x}{|x|}\ln(1+|x|), \tag{2}$$

the linear (identity) function

$$y = x, \tag{3}$$

and the sigmoid from -1 to 1

$$y = -1 + \frac{2}{1+e^{-x}} = \tanh\left(\frac{x}{2}\right). \tag{4}$$

Different node activation functions can be specified for each layer in the network. The signals from the nodes in the output layer form the network output.

In the model for recurrent networks (Figure 1b), the feedforward network is modified by including feedback connections with unit time lag from nodes in the output layer to nodes in the input layer. The feedback signals can either be taken from nodes with target values (true output nodes) or from additional *fictitious output nodes*. For each feedback connection specified, an additional node in the input layer, a *fictitious input node*, is created. Each fictitious input node has a trainable initial state, $y^0$, which corresponds to the node's activation for the first pattern in training (cf. Chapter 3); several initial states can be estimated to utilise training data composed of separate periods. The initial states are estimated by the training method. Since the fictitious output nodes act as internal network elements, the need for hidden layers is reduced for this type of networks. In contrast to feedforward networks, the recurrent ones do not necessarily require input signals, since they can implement autonomous systems. The feedback nodes, and, specially, the fictitious output nodes, may here act as *internal states* of the model [3].

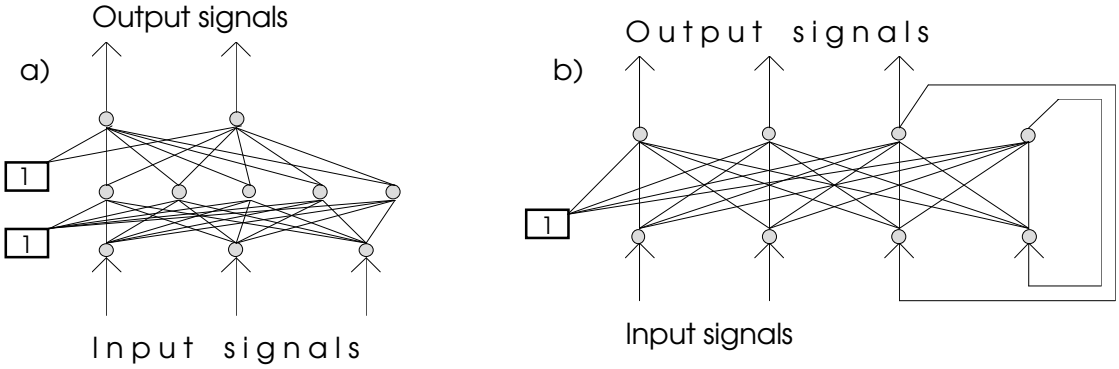



**Figure 1.** Illustration of the network models.
a) A feedforward network with three input nodes, two output nodes and one hidden layer with five nodes.
b) A recurrent network with two true input nodes, three true output nodes and two feedback connections, one from a true output and the other from a fictitious output node.

# 3 Network Training

The network training is the procedure where the unknowns, i.e. weights, bias terms and initial states for fictitious input nodes, are adjusted based on numerical training data. The training data is a set of *patterns* consisting of input signals and corresponding output signals, so called

*target* values. The goal is to find a network which describes the input-output relation represented by the training patterns. For recurrent networks with no true input nodes, the training data represents time-series for the output signals instead of input-output relations. The Levenberg-Marquardt method [4] is used to adjust the unknown network parameters in order to minimise the sum of squares (SSQ) of residuals, calculated as the differences between network outputs and target outputs. The problem can be formulated as

$$\min_{\mathbf{w}} \left\{ E = \mathbf{r}^T \mathbf{r} \right\} \tag{5}$$

where $\mathbf{r}$ is the residual vector and the network unknowns are collected into a vector, $\mathbf{w}$, according to

$$\mathbf{w} = (\overbrace{w_{10}^1, \ldots, w_{1N^0}^1}^{\text{to first node}}, \ldots, w_{N^1 0}^1, \ldots, w_{N^1 N^0}^1, \ldots, \underbrace{w_{10}^{L+1}, \ldots, w_{1N^L}^{L+1}, \ldots, w_{N^{L+1}0}^{L+1}, \ldots, w_{N^{L+1}N^L}^{L+1}}_{\text{weights to output layer}}, \underbrace{y_{11}^0, \ldots, y_{1k}^0, \ldots, y_{m1}^0, \ldots, y_{mk}^0}_{\text{initial states}}) \tag{6}$$

where $N^l$ is the number of nodes in layer $l$, $L$ is the number of hidden layers, $k$ is the number of fictitious input nodes and $m$ is the number of periods with separate initial states. At each iteration, $i$, the optimisation method adjusts the unknowns according to

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \boldsymbol{\delta}^{(i)} \tag{7}$$

where the correction, $\boldsymbol{\delta}$, is obtained from

$$\left( \mathbf{J}^{(i)} \mathbf{J}^{(i)T} + \lambda^{(i)} \mathbf{I} \right) \boldsymbol{\delta}^{(i)} = -\mathbf{J}^{(i)} \mathbf{r}^{(i)} \tag{8}$$

In the equation above, $\mathbf{J}$ is the Jacobian matrix with first derivatives of the residuals with respect to the unknowns and $\mathbf{I}$ is the identity matrix. The Marquardt parameter $\lambda$ is automatically adjusted during the training [5]; the method approaches Gauss-Newton if $\lambda \to 0$ and steepest descent with a small step length if $\lambda \to \infty$. Analytical expressions are derived for calculation of the Jacobian [6].

# 4 Additional Features

The data in the pattern file can be filtered by feeding it through a sliding window where arithmetic mean values are calculated. The size of the training set can be reduced by specifying that only every $n$:th (filtered) observation be used for training. Also, the user can specify the number of lines to be ignored in the beginning of the pattern file and give a limit for the number of patterns to be used in training. In addition to the pattern file with training data, the user can specify a separate test file. The test file option enables evaluation of the generalisation properties of the network throughout training by means of observations which are not included in the training data. The test data is fed through the network after each iteration and the root mean square (rms) error, $\sqrt{E/M}$, where $M$ is the number of residuals, for the test patterns is presented. The result from a training or an evaluation, i.e. input signals, node activations and target values for all patterns can be saved in a file for later use.

For certain problems, the training method may abort at a solution far from optimum due to saturation of nodes caused by single large weight values. To overcome this difficulty, several

methods for limiting the weight values are implemented. In the first method, box constraints for the weight values can be specified. In the second method, an upper limit for the relative weight change, $|\delta_j / w_j|$, is given. In both methods, the step suggested by the search method is restricted if the change in any parameter would exceed the limits. The third method prevents large weight values by introducing a penalty term as an additional residual to be minimised in the training. The penalty residual is calculated as

$$r_{M+1} = \gamma \sum_{i=1}^{n} |w_i|, \tag{9}$$

where $\gamma$ is a (non negative) factor specified by the user, $M$ is the (original) number of residuals and $n$ is the total number of unknowns in the network.

There is a possibility to reduce the number of parameters in the network by keeping some of the network weights constant during training, or, keeping the values of some weights equal. For a weight which is specified as constant, the initial value is used throughout training.

For recurrent networks with feedback connections from true output nodes, the convergence in training may be enhanced if the target values, instead of network outputs, are fed back occasionally. The user specifies the interval between such *teacher forcing* actions [7].

# 5 User Interface

The user interface, developed with MS Visual Basic 3.0, was built to be both flexible and easy to use. All parameters defining the network and its training are presented in setup windows and stored in a file which subsequently can be read by the program. On-line help is available for all windows in the program, the help page is shown by pressing F1. The training can be interrupted at any time retaining all network information. Although the user interface is mainly developed for interactive use, non-interactive runs may be carried out by starting the program with optional command line arguments. Also, several instances of the program may run simultaneously.

During training, the network and its performance can be analysed in several ways. Network outputs and desired outputs are presented in a graph which is updated after each iteration. Alternatively, the residuals (i.e. differences between network outputs and desired outputs), the weights or the activations of internal network nodes can be plotted. The sum of squared errors (SSQ), the rms error and the Marquardt parameter ($\lambda$) are shown after each iteration and written to a log table, which can be analysed later. In a window showing the training progress, the rms error is plotted vs. iteration index. If a test file is used, the rms error for the test patterns is also plotted. The graphs can be copied to the clipboard as Windows metafiles. Network weights, initial states for fictitious input nodes and node activations are easily examined in the network state window.

In the following sections, the user interface is illustrated with two examples. The basic settings for a feed-forward network, file specifications, etc. are explained in the first example whereas the specification of a recurrent network and the use of some options associated with this network type is shown in the second example.

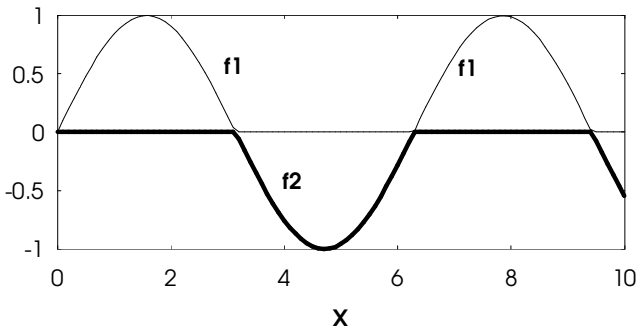
# 6 Example I: A Feed-Forward Network

This example shows how basic settings are made for a feed-forward network and how node saturation is avoided by weight restriction during an initial training phase. The training data is created using the relations

$$f_1(x) = \max(\sin(x), 0)$$
$$f_2(x) = \min(\sin(x), 0)$$

which can be interpreted as the sine function divided into a positive and a negative part. The pattern file holds values for $x$ in a range from 0 to 10 and corresponding values for $f_1$ and $f_2$. The first lines of the file and an illustration of the data is shown below.

```
x       f1(x)         f2(x)
0       0             0
0.1     0.099833      0
0.2     0.198669      0
0.3     0.29552       0
.
.
.
```




The columns in the pattern file are separated with tabs, but space(es) can also be used as column separators. We use a network with one hidden layer for the example problem, $x$ is chosen as network input whereas the desired outputs are $f_1$ and $f_2$.

The main window of the program, which holds a picture of the network specified, is shown in Figure 2. The name of current setup file is displayed in the title bar. During a run, information about the training progress is shown. Other windows are accessed from menus in the main window. A description of the menu items and the other windows of the program is given below.

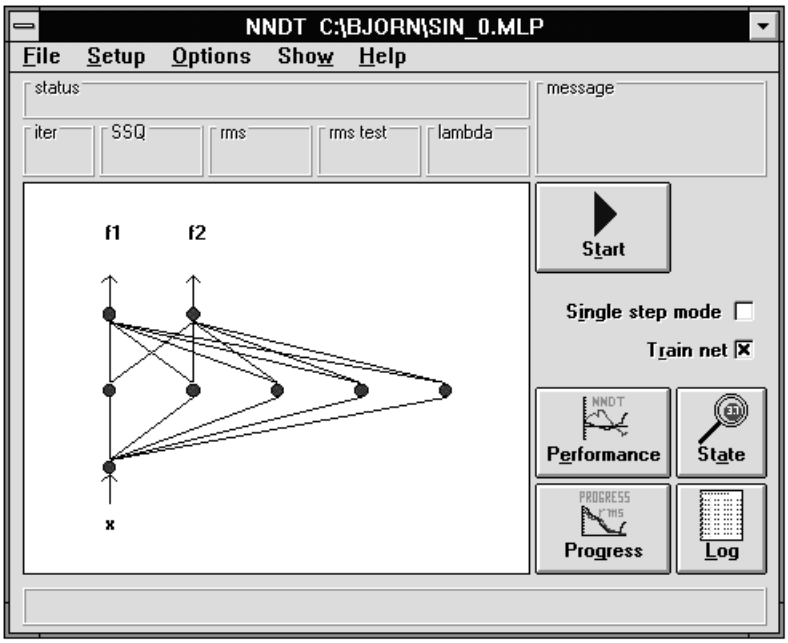
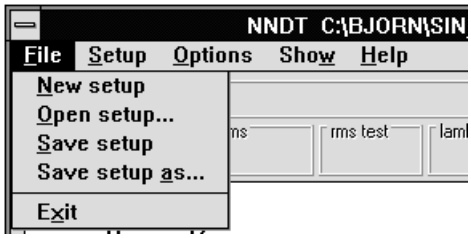



**Figure 2**.

## File Menu




The File menu is used for reading and saving setup information and for exiting the entire program.

- New setup selects a network with only one input node and one output node and sets all configuration parameters to default values. However, the setup of a new network is usually started by opening an existing setup file.
- Open setup... displays a common dialog box for file selection, see Figure 3. In the present example, the setup is read from a file named sin_0.mlp. (Each time NNDT is started, the setup file used in previous session is automatically read.)

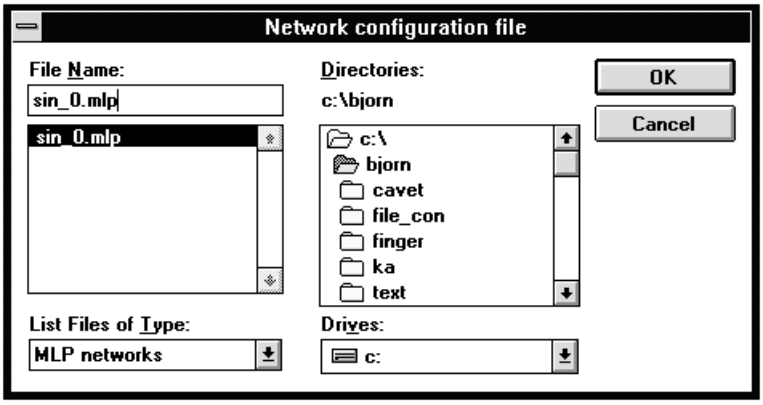
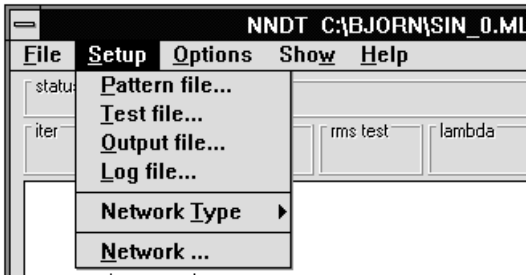



**Figure 3**.

- Save setup writes the actual setup information to the setup file last selected.
- Save setup as... displays a dialog box for setup file selection and writes the actual setup information to the chosen file.

## Setup Menu




- Pattern file... opens a form for specification of the pattern file. The pattern file setup form for the present example is shown in Figure 4.

  The file name is selected in a dialog box. If the user wants to modify the contents of the pattern file between subsequent

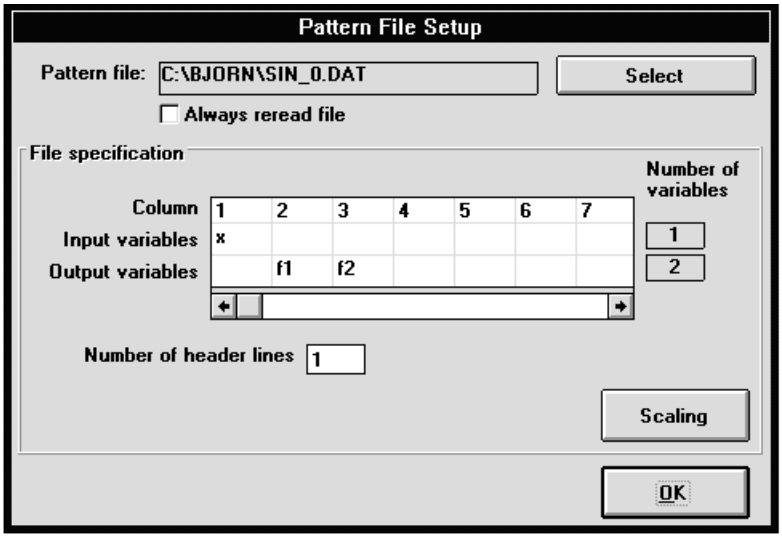



**Figure 4.**