7 Example II: A Recurrent Network

Recurrent neural networks are especially suited for modelling of temporal sequences (time series). In order to illustrate the use of recurrent networks in NNDT, a simple example was designed: The task is to produce two trigonometric oscillations,

$$f_1 = \sin\left(\frac{\pi}{18} i\right)$$
$$f_2 = \cos\left(\frac{\pi}{18} i\right) \qquad \text{i} = 0, \ldots, 72$$

without using external inputs, i.e., by an autonomous system. The sequence plotted in state space (f_1, f_2) produces a circular trajectory. It can be easily shown [10] that this sequence can be exactly reproduced by a fully recurrent two-node network with linear activation function. Here, however, we study a network with [-1,1] sigmoidal activation (Figure 15, which is part of MLP setup window). The reason for using a nonlinear model is that our goal is to create a periodic attractor.

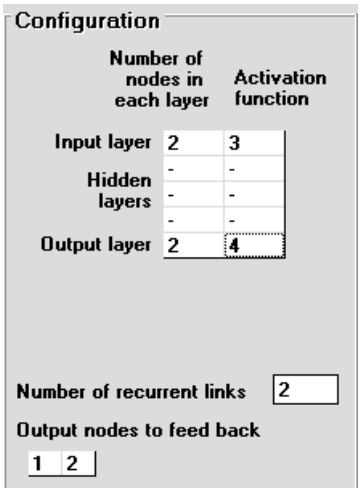

Figure 15.

Figure 16 shows the setup of the network with two (fictitious) input nodes and two feedback nodes in the output layer. We use a scaling of 0.5 on the two *f*'s in order to make it possible to use the activation function of Equation (4), since the nodes will have to operate in their linear ranges (arguments close to zero) to reconstruct the sequence.

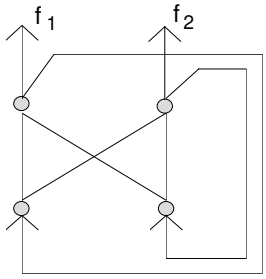

Figure 16.

The result of a network trained from one starting point (seed=40, α=0.1) is depicted in Figure 17. Here, the network has converged to a false minimum, a fixed point at $(-0.07, -0.045)$. Such problems typically occur when recurrent networks are trained. Therefore, it may be advisable to vary the location of the starting point. For instance, using seed=75, the network converges rapidly to the correct

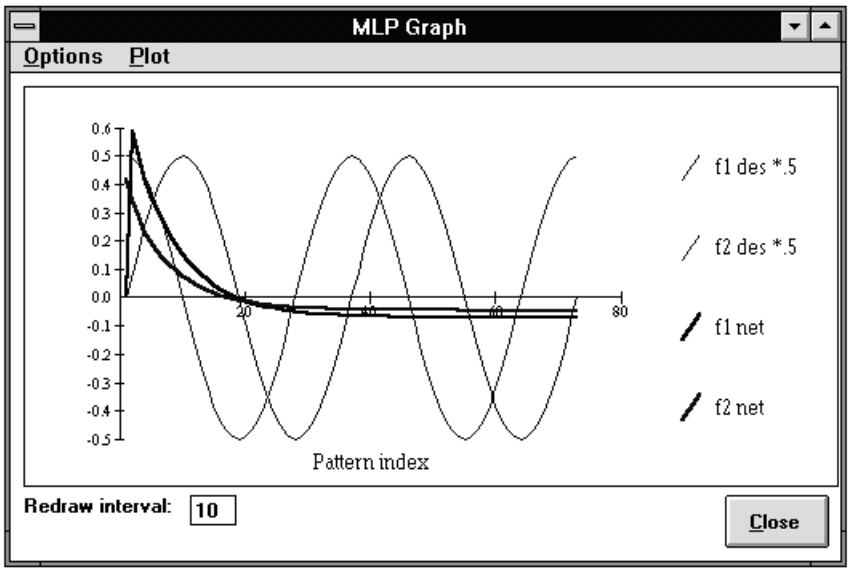

Figure 17.

solution. Another possible remedy is to use so called *teacher forcing*, where the true outputs (targets) are occasionally fed back instead of the networks outputs. This may force the network's state back to the desired trajectory, which facilitates the training. Usually, there is no need to extend the training with teacher forcing beyond the point where the network has captured the main features of the attractor. In the present example, teacher forcing on every 30th pattern was applied. After six iterations from the starting point (seed=40, α=0.1), the network is seen to have created oscillatory sequences, which resemble the desired ones (left graph in Figure 18). The *phase portrait* (right graph) also indicates periodic behaviour.

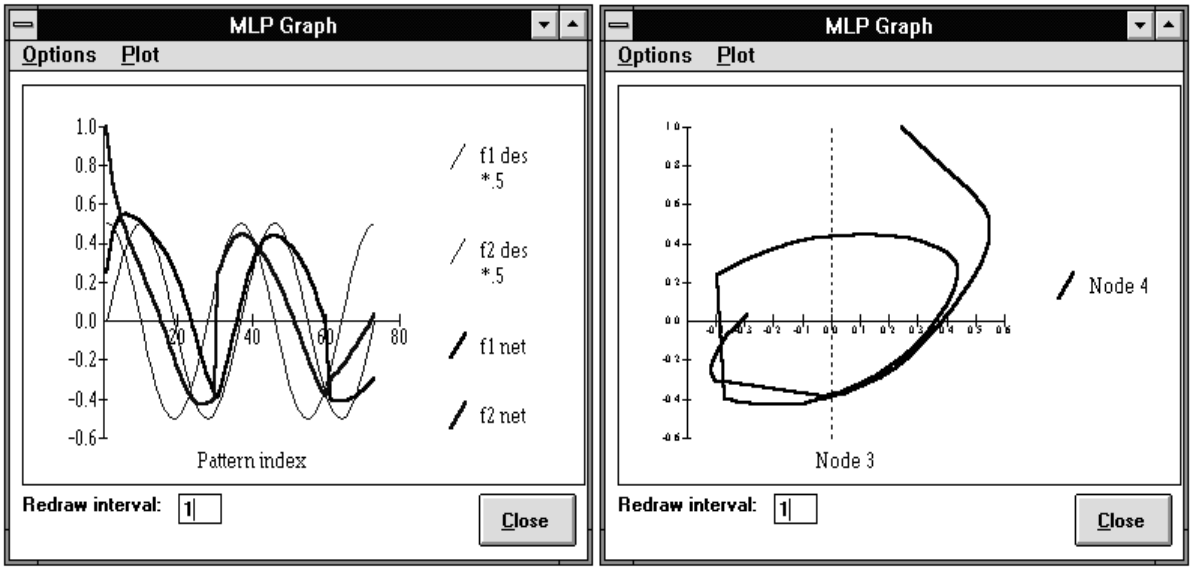

Figure 18.

If the training is continued without teacher forcing from this point onward, the network converges upon the correct solution in 24 iterations, see Figure 19.

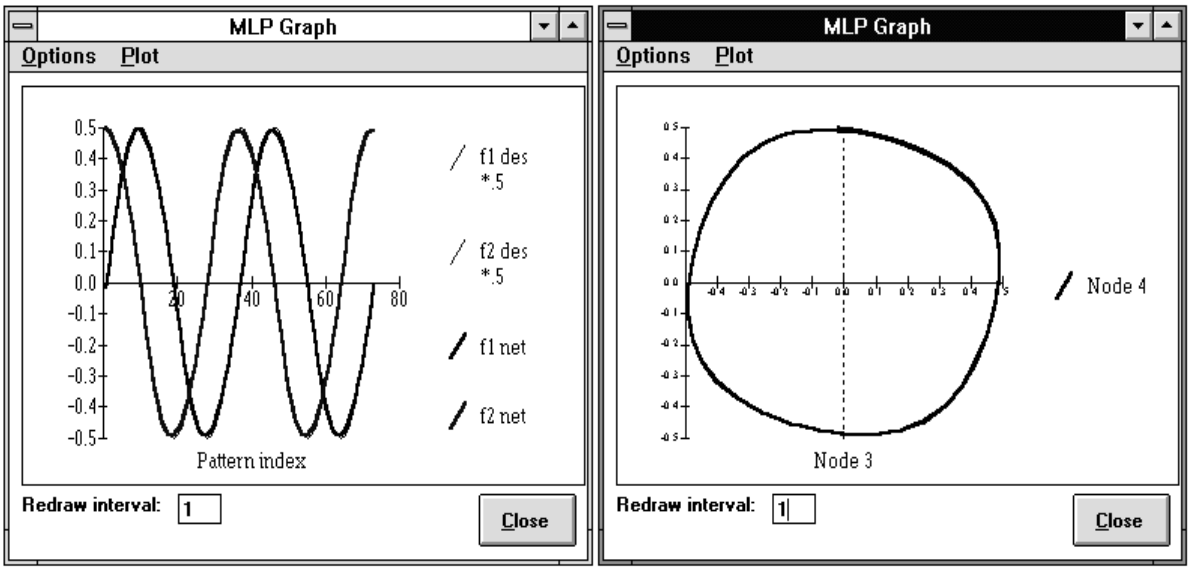

Figure 19.

The optimal network weights are shown in Figure 20, which is an excerpt from the Network state window.

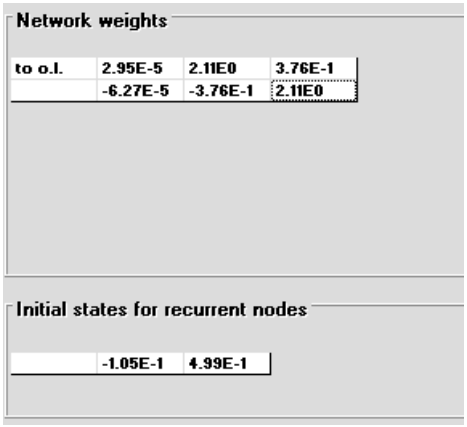

Figure 20.

The fact that NNDT has estimated a proper initial state of the network is illustrated in Figure 21, which shows the evolution of the network from a modified initial state, $\mathbf{y}^0 = (1,1)^T$. The right graph illustrates that there is a circular attractor (the deformation of it being due to the different scaling), while the left graph shows that the oscillations now produced by the network are in wrong phase. This stresses the importance to estimate not only the weights but also $\mathbf{y}^0$ in recurrent networks.

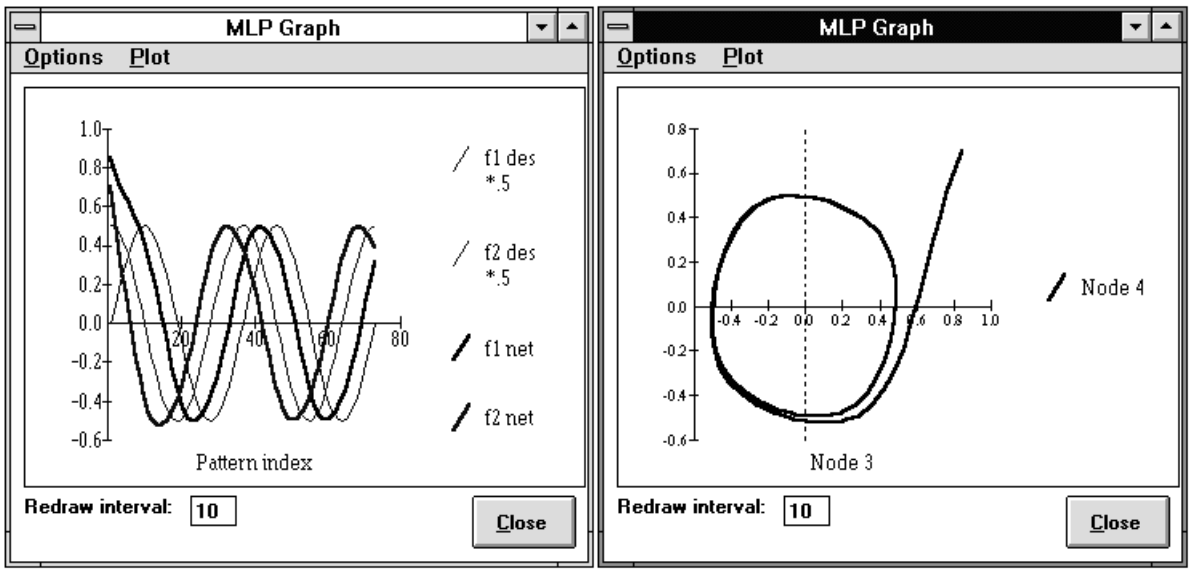

Figure 21.

Because of the obvious symmetry of the weights (cf. Fig. 20) at the optimum, a reduced model is trained in order to illustrate the possibilities to include weight constraints in NNDT. Both bias terms are assumed constant and zero (w_0=w_3=0), while for the weights w_1=w_5 and w_2=-w_4. After a normal initialisation of the network state (e.g., seed=40, α=0.1) and setting w_0=w_3=0, the constraints can be entered through the Equal & constant weights window shown in Figure 22. This two-parameter model can be trained to yield (practically) the same solution as reported above.

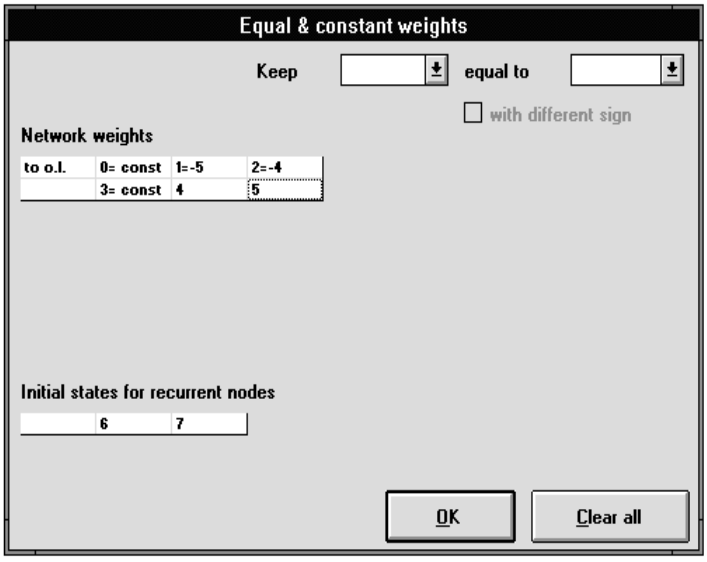

Figure 22.

8 Network Setup Windows: List of Options

The contents of the network setup windows are listed and the options and settings are briefly explained. A specification of the type of the user input is given in brackets. For some options, references to corresponding chapters in this manual are given.

MLP setup window

Filter pattern data [check box]
: Activates the data pre-treatment (next two options).

Number of old val in mean filter [integer]
: Selects the size of the sliding window for the mean value filter. The filter function is disabled if 0 is given.

Number of samples between pattern formation [integer]
: Specifies how many lines to be read (and filtered) before a new pattern is formed. The option is used for reducing the number of patterns in training data. If 1 is given, a pattern is formed for every line in the file.

Number of patterns to be formed before training [integer]
: Specifies the number of patterns to be used in training. If this number is reached before the entire pattern file is read, the remaining lines in the file will be ignored.

Optimisation task [dropdown list]
: All network weights: Normal training where all weights and biases (and initial states) are open parameters.
Gain terms on input signals: Only gain terms for the inputs are adjusted during training, the network weights are kept at their initial values.
Gain terms on output signals: Only gain terms for the outputs are adjusted during training, the network weights are kept at their initial values.
Gain terms on input and output signals: Only gain terms for the inputs and the outputs are adjusted during training, the network weights are kept at their initial values.

Max number of iterations [integer]
: Specifies the maximum number of iterations in training. After this number is reached, the algorithm aborts.

Number of nodes in each layer [column of integers]
: Specifies the number of hidden layers and the number of nodes in each hidden layer. When an element in the table is chosen, a list with the available choices is shown. The numbers of input and output nodes are also shown in the table but these numbers can only be changed in the pattern file setup form or by a modifcation of the recurrent links.

Activation function [column of integers] (Chapter 2)
: Specifies the activation function for the nodes in the hidden layers and in the output layer. When an element in the table is chosen, a list with the available choices is shown. The activation function for the input nodes is always the identity function.

Number of recurrent links [integer] (Chapter 2)
: Specifies the number of feedback connections. For each feedback connection, a fictitious input node is created.

Output nodes to feed back [row of integers] (Chapter 2)
: Specifies the output nodes for the feedback connections. The output nodes are numbered according to the picture in the main window; the leftmost node (first true output node) is number 1. A fictitious output node is created if the number specified for the feed-back node exceeds the number of true output nodes.

MLP advanced setup window

Analytical derivatives [check box] (Chapter 3)
: Switch for selecting analytical expressions to be used for the calculation of the Jacobian. For some special network configurations (e.g. networks with weight equalities), analytical expressions are not available and numerical derivatives are used automatically.

Limit parameter values [check box]
: Activates the parameter limits specified in the two boxes below.

min, max [real values]
: Specifies lower and upper limits for the weights.

Limit relative change of parameters [check box] (Chapter 4)
: Activates the limit for relative parameter change specified in the box below.

Max relative change per iteration (%) [real value, >0] (Chapter 4)
: Specifies the maximum percentage change for each parameter (absolute value) allowed per iteration in the training.

Penalty factor [real value] (Chapter 4)
: Specifies the factor in the penalty term (Equation (9)). The penalty function is switched off if 0 is given.

Number of separate periods [integer] (Chapter 2)
: Specifies the number of separate periods in the training data for a recurrent network.

First patterns in the periods [row of integers]
: Specifies the numbers for the first patterns in the periods. For these patterns, the fictitious input nodes are set to their initial states.

Use equal initial states [check box]
: Specifies that each input node only has one initial state, used for all periods. Otherwise, separate initial states are estimated for each period.

Use teacher forcing [check box] (Chapter 4)
: Activates the teacher forcing option for recurrent networks with true outputs. The option is specified in the box below.

Desired outputs are used each [ ]:th pattern.
: Specifies the interval between teacher forcing actions. Teacher forcing is never used for the first pattern in a period.

9 System Requirements and Installation

NNDT runs under MS Windows 3.1 on personal computers. The program itself (NNDT.EXE), the network routines (NNDTCALC.DLL) and the help file (NNDT.HLP) use approximately 200 K disk space. Together with a number of Visual Basic modules (.DLL and .VBX files) which are required, but in many cases already present on the system, the total use of disk space is approximately 1500 K. The computer should be equipped with a display of at least VGA monochrome class, but a colour monitor is preferable for the graphics.

NNDT is delivered with a setup program which automatically copies the program files to a user-specified directory and the VB modules to the \SYSTEM directory. Two pattern files and corresponding setup files are included for demonstration.

References

[1] Rumelhart, D. E. and J. McClelland (Eds.), *Parallel Distributed Processing*, MIT Press, 1986.

[2] Bulsari, A. B. and H. Saxén, "System identification using the symmetric logarithmoid as an activation function in a feed-forward neural network", *Neural network world* **1** (1991) 221-224.

[3] Bulsari, A. B. and H. Saxén, "A Recurrent Network for Modeling Noisy Temporal Sequences", *Neurocomputing* **6** (1994), in press.

[4] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, Cambridge, 1986.

[5] Marquardt, D. W., "An algorithm for least-squares estimation of nonlinear parameters", *J. SIAM* **11** (1963) 431-441.

[6] Williams, R. J. and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", *Neural Computation* **1** (1989) 270-280.

[7] Jordan, M. I., "Attractor dynamics and parallelism in an connectionist sequential machine", Proceedings of the *Eight Annual Conference of the Cognitive Science Society*, Amherst, 1986, pp. 531-546. Hillsdale: Erlbaum.

[8] Weigend, A. S., B. A. Huberman and D. E. Rumelhart, "Predicting the Future: A Connectionist Approach", *International Journal of Neural Systems* **1** (1990) 193-209.

[9] Finnoff, W., F. Hegert and H. G. Zimmermann, "Improving Model Selection by Nonconvergent Methods", *Neural Networks* **6** (1993) 771-783.

[10] Bulsari, A. B. and H. Saxén, "A Partially Recurrent Connectionist Model", Proceedings of the *10th European Conference on Artificial Intelligence*, (Ed. B. Neumann), Vienna, Austria, 1992, pp. 198-202.