NONLINEAR SYSTEM MODELLING WITH MODULAR NEURAL NETWORKS

Bernardo Morcego

Automatic Control and Computer Engineering Department
Universitat Politècnica de Catalunya
Rbla. Sant Nebridi, 11
08222 Terrassa (Spain)
bernardo@esaii.upc.es

Abstract: This paper presents a software tool suitable for dynamic system modelling. The models generated by this tool are modular neural networks, see (Sharkey, 1999). Each module behaves like a functional block and is connected to the other modules like in classical block diagrams. This tool allows the inclusion of a priori knowledge and, furthermore, to extract physical information from the models, once the system has learned. The modelling tool is capable of automatic model generation, parameter estimation and model validation. *Copyright © 2000 IFAC.*

Keywords: Dynamic system modelling, modular neural networks, software tools, knowledge acquisition.

1. INTRODUCTION

The identification of linear systems has a well-established theory and a wide range of efficient mathematical tools, including a variety of parameter estimation techniques, which provide a good basis for system analysis and controller design. This is not the case, however, for nonlinear systems. Since no generally applicable techniques exist for the analysis of these systems, the identification and controller design are usually performed on a case-by-case basis. The main available techniques for nonlinear modelling can be classified (Billings, 1980; Haber and Unbenhauen, 1990) into three categories: functional series methods, block oriented methods and black box methods.

Artificial Neural Networks (ANN) modelling techniques belong to the class of black box methods. Neural networks are structures used for knowledge synthesis by applying the learning-by-example paradigm. Different ANN structures have been used for system identification, e.g. (Hunt, *et al.*, 1993). The most commonly used neural architectures are feedforward networks with the backpropagation learning rule and recurrent networks with learning algorithms such as the real-time recurrent learning or the backpropagation-through-time algorithms.

One of the most frequent problems in using neural networks as black boxes is that the number of neurones increases rapidly with the order of the system and the learning process needs more time for each training pattern, and more training patterns, to be able to learn. When such problems arise in applied and theoretical sciences, the classical solution is to partition the problem: the divide and conquer approach. However, the ANN research community has carried out very little remarkable studies in this direction. Modular Artificial Neural Networks

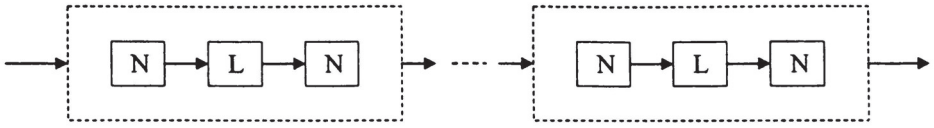

Fig. 1: block diagram of the model structure

(MANN) are popular, in spite of this, in applications in which task subdivision can be made beforehand. A network is trained for each task and then the resulting nets are all combined to give the desired result, e.g. (Bennani and Gallinari, 1982; Chen and Liao, 1998).

Two important studies on the design and training of modular neural networks are those by Jacobs and Jordan (1994) and Happel and Murre (1994). The former describes an ensemble architecture composed of expert and gating networks. Expert networks compete to learn a certain aspect of the behaviour of a dynamic system while the gating network (or networks) decides which of the experts will contribute in the output. The work by Happel and Murre is based on a model of the cerebellar microcolumn. They call it CALM (Categorising Adaptive Learning Module) and it gives successful results in classification tasks when interconnected in a MANN fashion.

1.1 Neural Modules

The building blocks used here are called *Neural Modules* (NM). A neural module is a new concept in the area of ANN, developed by Morcego, *et al.*, (1996).

A neural module is a neural network that, due to structural constraints, behaves inherently like a specified function (or set of functions). The procedure to tune one specific behaviour within the family of functions structurally represented by the NM is the learning mechanism, which only concerns a subset of the weights, while the others are forced to remain constant. For example, a NM can be designed to represent the input-output mapping of a typical nonlinearity in systems theory, like the saturation, the backlash, etc.

Neural modules will be described in more detail in section 3. They can be thought of as models of simple functions (dynamic or static) with as many parameters as the function they model. The learning algorithm adjusts those parameters, which match biunivocally the parameters of the modelled function.

1.2 Model structure

The tool described in this paper was conceived for modelling nonlinear systems, see (Morcego, *et al.*, 1997) and, more precisely, for those systems whose dynamics are essentially linear but some of its components display nonlinearities (usually hard nonlinearities like saturation, dead zone, backlash, etc.) Many physical systems can be arranged in such a way, being the nonlinear block a combination (series, parallel or mixed) of simple hard nonlinearities. Figure 1 shows a block diagram of the system suitable for modelling within the framework presented in this paper. The blocks labelled *N* represent a nonlinear function (dynamic or static) and the blocks labelled *L* represent a linear system.

The models generated by the tool described in this paper are modular neural networks. They consist of simpler ANN that behave like functional blocks, each of which is a neural module. Those modules are connected to one another in a classical block diagram fashion and each one has a very easy interpretation.

A neural module network is a standard ANN in a broad sense. Its neurones are linear or sigmoidal, patterns are fed in a discrete-time fashion and it uses standard learning algorithms. The main difference between a typical three-layered network and a NM is that the connectivity pattern is not restricted in any way. Another difference is that the weights of a NM may take constant values, not modifiable by the learning algorithm. With this approach it is possible to force a particular behaviour in each functional block and, consequently, in the model.

This model structure was selected to benefit from the known advantages of neural networks in dynamic system identification and control while trying to minimise their inconveniences.

2. DESCRIPTION OF THE MODELLING TOOL

The modelling tool described in this paper, called `miga`, is capable of automatic model generation, parameter estimation and model validation.

This tool integrates two paradigms of current search methods, namely Evolutionary Programming (EP) and neural learning algorithms, with the common goal of modelling dynamic systems.

`Miga` is a software tool programmed in C and Matlab. The former is used for efficient neural learning and the latter is used for model creation and evolution and for user interaction.

`Miga` is fed with two types of input data. On the one hand, it needs significant input-output sequences from the system to be modelled. Those sequences must be classified into training, test and validation data. On the other hand, `miga` accepts a priori knowledge about the system. This type of information allows biasing the search, which is partially stochastic. A priori knowledge is represented by a set of functions (neural modules) and their associated probabilities of belonging to the solution. That set is used by `miga` to generate and evolve models as candidate solutions.

This tool is intended for single-input-single-output system modelling. The resulting models are modular neural networks, representing nonlinear discrete time dynamic systems. The modules of the network are immediately interpretable as known static or dynamic functions, e.g.: saturation, first order system, viscous friction.

2.1 Top-down structure of `miga`

The main operation diagram of `miga` is shown in figure 2. It consists in two cooperative procedures, the evolutionary programming algorithm and the neural simulator.

The EP algorithm is used for model creation and for parameter estimation. The neural simulator is used for model parameter fine-tuning (training phase) and for model evaluation (test phase).

The EP algorithm deals with sets of candidate solutions (populations of solutions, in the evolutionary techniques' jargon). Each solution represents a modular neural network, but is implemented as an acyclic directed graph. Therefore, the EP algorithm evolves graphs. The nodes of those graphs contain the essential information to transform each node into a neural module, particularly the name of the function approximated by the module and its parameters. The adjacency matrix gives the appropriate correspondence between a graph's connections and the connections between modules in the network.

The neural simulator is special for two reasons: first, it must be capable of efficiently training and testing large modular neural networks and second, it must carry out the transformation between EP candidate solutions, graphs, and neural networks and viceversa.

2.2 EP algorithm

The EP algorithm follows the classical outline given by Fogel, *et al.*, (1966). It is, however, adapted to the evolution of neural networks as in Angeline, *et al.*, (1994), that is:

Pop = Create initial population at random
Evaluate each individual ($x \in$ Pop, f)
<u>while</u> end condition $\neq$ FALSE <u>do</u>
    NewPop = mutation (Pop)
    Evaluate each individual ($x \in$ NewPop, f)
    Pop = selection (Pop, NewPop)
    Evaluate end condition
<u>end</u>

The initial population is a set of N graphs. The nodes in those graphs are chosen according to their a priori

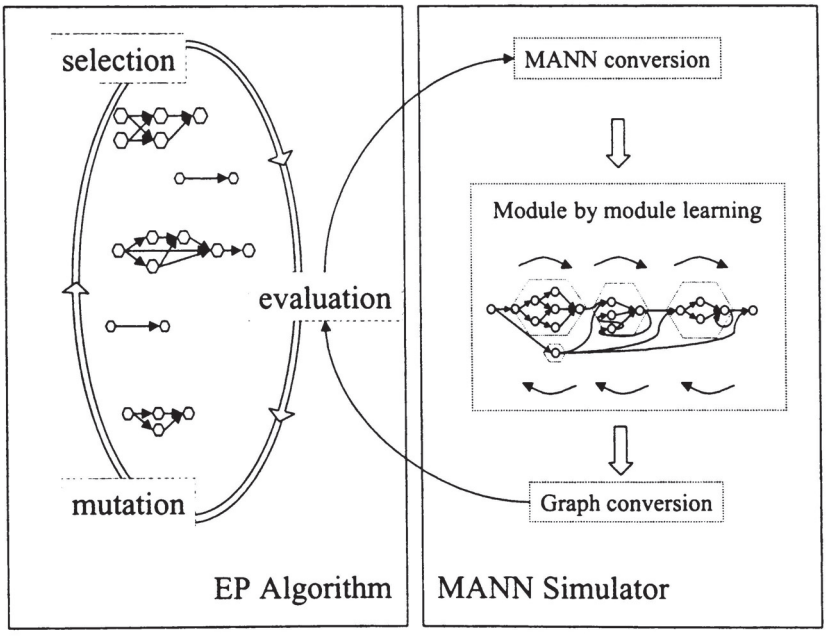

Fig. 2: operation diagram of the modelling tool

probabilities and their parameters are set following a uniform distribution centred at default values. The number of individuals, N, the maximum initial number of nodes in each individual and the maximum parameter deviation are user defined parameters. Those parameters are usually set at 15, 3 and 0.7 × default value, respectively.

Individual evaluation is carried out by the neural simulator and considered in the following section.

The mutation operator is an important function in EP because it is the only search space exploration function. Other evolutionary techniques rely on other operators, such as crossover, to explore the search space, but EP algorithms must include a powerful mutation operator in order to maximise the probability of success.

In this case, mutation is capable of structural and parametric changes. The following list shows the types of mutations implemented in `miga`, sorted by severity:

- Node insertion: it is a structural mutation that does not alter the behaviour of the model because the inserted node is a unitary static gain. See figures 3 and 4 for examples of allowed and illegal insertions.
- Node function parameter change: a parameter is given its value following a uniform distribution centred at its actual value.
- Node function change: a new function is chosen for a node according to the a priori probabilities of the available functions.
- Node or connection suppression: it is a structural mutation that can alter the model's behaviour considerably. See figures 3 and 4 for examples of allowed and illegal suppressions.

The probability of each type of mutation is a user defined parameter, usually set at 0.25, 0.1, 0.4 and 0.25, respectively. The maximum function parameter deviation is usually set at 0.7 × actual value.

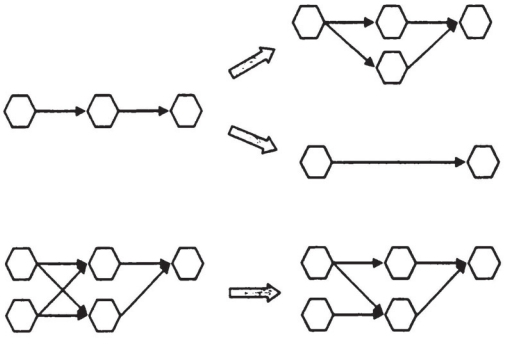

Fig. 3: Three examples of allowed structural mutations. The first one corresponds to a node insertion; the second one is a node suppression and the last one is a connection suppression.

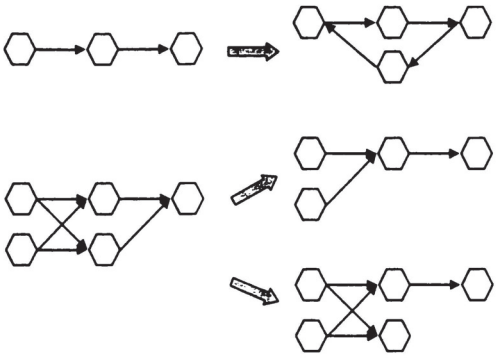

Fig. 4: Three examples of illegal structural mutations. The first one is a node insertion; the second one is a node suppression and the last one is a connection suppression.

The selection phase consists in choosing a new population of N individuals from two evaluated populations. The two classical methods of selection are elitist and competitive selection. Here it is used the second one, with 3 candidate solutions competing in each tournament.

2.3 Neural simulator

As earlier stated, the neural simulator is used for model parameter fine-tuning and for model evaluation.

The neural simulator receives graphs from the EP algorithm and data corresponding to the training and test sequences. For each graph the neural simulator performs the following tasks:

- Transforms the graphs into modular neural networks.
- Trains each MANN for a short number of epochs.
- Tests each MANN to assign it a goodness value.
- Transforms back the MANN into a graph with, possibly, different parameters in each node.

The neural simulator differs from other standard simulators (SNNS, Aspirin/MIGRAINES, PlaNet, etc.) in its special ability to handle modular neural networks. Other simulators do not explicitly support training of MANN. The user must create a single large network from the smaller modules and train it with a recurrent learning algorithm in case any of the modules is recurrent.

The neural simulator implements *Modular BackPropagation* (MBP), a learning algorithm developed in Morcego (2000). MBP is a learning management algorithm. Its main distinctive feature is the local use of standard learning algorithms, e.g. backpropagation or backpropagation through time, in each module. MBP is computationally more efficient

than those standard learning algorithms when applied to modular neural networks.

2.4 Model validation

Model validation is an important feature of `miga` because all the models generated by the tool are potentially interesting. This phase is semi-automated as far as it often depends on the user's subjective criterion.

The first step towards finding the final solution is to choose a reduced set of models to apply the validation data to. `Miga` helps the user with plots of model test error versus complexity. Complexity is characterised as the number of modifiable parameters of the model.

After the user has selected some models, it is possible to proceed with the validation. In this phase, it is possible to apply fresh data (validation sequence) to each chosen model and compare their error and output sequences. It is also possible to perform correlation tests to check the independence between input data and model residuals. Finally, it is also possible to apply an intensive neural learning phase to adjust the parameters of the model, in case it should be necessary.

3. NEURAL MODULE LIBRARY

The neural module library is described in detail in Morcego (2000). It contains nine nonlinear functions and the linear SISO systems. The nonlinear functions are: two state threshold, two state hysteresis, saturation, dead zone, absolute value, friction, backlash, mechanical stop and rate limiter.

For the sake of simplicity, only a brief neural module example will be given. The reader is referred to the previous reference if interested in the design of other types of neural modules.

The two state hysteresis neural module is examined next. The two state hysteresis is a dynamic nonlinearity with its outputs restricted to 1 and -1. Figure 5 depicts its input-output representation.

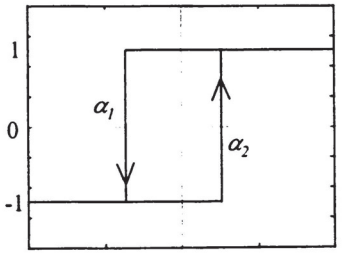

Fig. 5: Ideal two state hysteresis.

The neural module that approximates this function is represented in figure 6. This module has 3 sigmoidal neurones and 7 weights. The two leftmost neurones play the role of detecting the interval the input lies in. For instance, if the input is smaller than α_1 both neurones will fire low. Therefore, no matter what the previous output was, the actual output will be low, too.

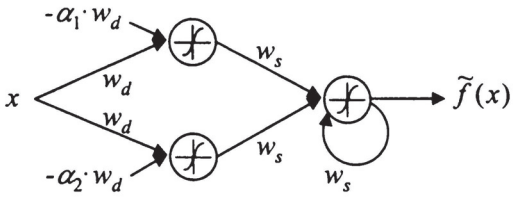

Fig. 6: Neural module that approximates the threshold function with hysteresis. The values w_d and w_s force the expected behaviour and the values $\alpha_1 w_s$ and $\alpha_2 w_s$ correspond to the function's parameters.

The weights labelled w_d and w_s are not learnable and their values provide a means to control the interval discrimination sensitivity and output activation steepness, respectively. The weights labelled $-\alpha_1 \cdot w_d$ and $-\alpha_2 \cdot w_d$ are adjusted by the learning algorithm using input-output examples. Those weights determine the parameters of the specific hysteresis function being approximated.

4. APPLICATION EXAMPLE

In this example a modular model is obtained, using `miga`, for a simulated experiment. This experiment reproduces the behaviour of a dc motor. The input is the voltage applied to the armature of the motor and the output corresponds to rotation speed.

The behaviour of the motor is modelled as a first order system with two nonlinear functions at its input. Those functions are the saturation and the dead zone. See figure 7 for block layout and details.

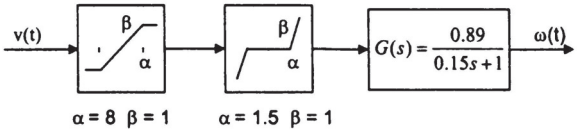

Fig. 7: Original block diagram of the dc motor experiment. The first block is a saturation function of unitary slope and limits at ±8V. The second block is a dead zone function of amplitude 1.5V. The third block models the dynamics of the system.

Data sequences, generated using Simulink, are 601 samples long (12s at 0.02 sampling rate). Input

values are obtained from two white noise generators, chosen at random to produce about 40% saturated outputs, 40% in the dead zone and the rest in between those two regions. The output is perturbed by a band-limited white noise of about ±1.5% of the output. There is one sequence for each modelling phase: training, test and validation. Figure 8 shows an example sequence.

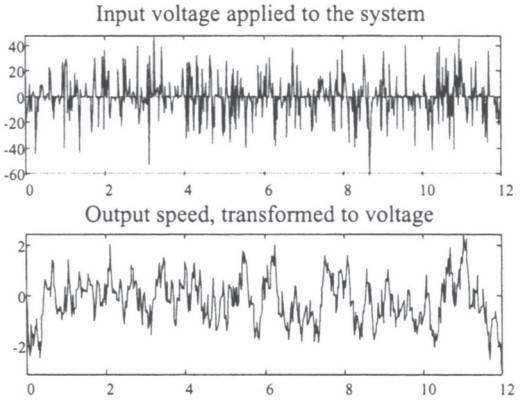

Fig. 8: Sample input and output data sequences.

The a priori knowledge codified into this experiment is: equal probability for static nonlinear functions; zero probability for dynamic nonlinear functions; similar linear systems of first, second and third order, with unitary gain and time constant between 0.1s and 0.4s.

It is impossible to give a comprehensive list of the results obtained. Only the final model will be described. This model uses 17 modules and 19 parameters. It produces a test error of $2.7 \cdot 10^{-3}$ (ISE criterion). Figure 9 shows part of the test and error sequences.

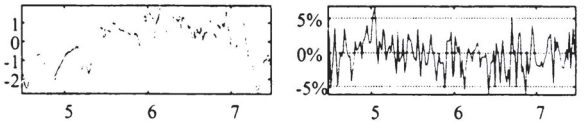

Fig. 9: Partial output sequences of test and model output data (left) and their difference (right). The error lies in the 5% band.

A very interesting quality of this model, observed in many of the dismissed ones, is its great generalising capacity. There is no qualitative difference between test and validation errors. Figure 10 shows partial views of the performance of the final model with three different validation sequences. One can easily see the error sequences are very similar to those on figure 9.

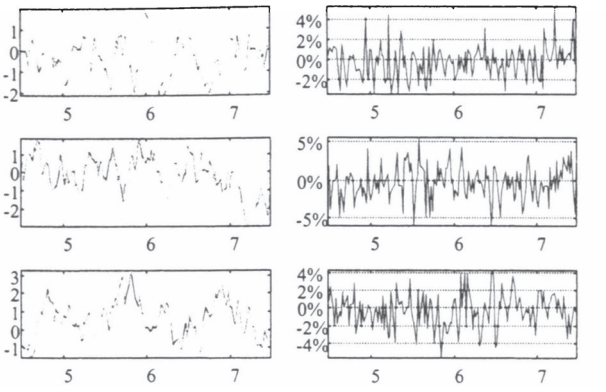

Fig. 10: Partial output sequences of validation and model output data (left) and their difference (right). The error is similar to that on figure 9.

This model was compared with a classical tapped-delay three-layered neural network. The best network has eleven neurones in the middle layer and two tapped-delayed inputs and outputs. After a long training (1000 epochs), with the same data used to obtain the modular model, the test error reached $5.56 \cdot 10^{-3}$. Validation errors are larger and lie in the 10% band. This model has 78 parameters.

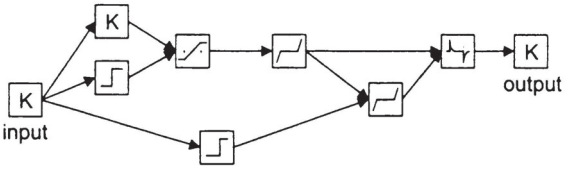

Fig. 11: Block diagram of the post-processed final model. It has seven modules.

The most interesting feature of `miga` is that the final model is easy to analyse. In this case, the model was transformed to Simulink format and its internal signals were analysed. An equivalent model with only seven modules was obtained, see figure 11. This model makes explicit two features of the original system: a saturation function in the input path and a linear system of 0.15s time constant.

All this information may be extremely useful to understand the behaviour of the original system.

5. CONCLUSIONS

This paper presents a software tool, called `miga`, useful for nonlinear system modelling with modular neural networks.

This tool does not rely entirely on neural learning to find a suitable model. It allows the inclusion of a priori structural knowledge about the relationship between the physical system and the model components. Furthermore, as far as highly structured

networks are being used it is easier to extract information from the model, once the system has learned. It is naturally amenable to the representation and adaptive identification of control systems because of the adaptativity properties of ANN. Additionally, the resulting system's properties are easy to study because the neural network parameters are meaningful. Furthermore, the MANN approach overcomes, to a great extent, the problem of "explosion" in the number of neurones and, hence, in the learning time.

This architecture attempts to combine the ability of ANN to approximate any nonlinear function, with the clarity of information in block-oriented methodologies for system identification.

ACKNOWLEDGEMENTS

This work is partially supported by the research grant CICYT TAP98-0585-CO3-01 of the Spanish Science and Technology Council and by the Research Commission of the Generalitat de Catalunya (grup SAC, ref. 1999-SGR-00134). The author is member of CERCA (Collective for the Study and Research of Control and Automation).

REFERENCES

Angeline, P.J., G.M. Saunders and J.B. Pollack (1996). An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transactions on Neural Networks*, vol. 5, nº 1.

Bennani, Y. and P. Gallinari (1992). Task Decomposition Through a Modular Connectionist Architecture: A Talker Identification System, *Artificial Neural Networks, 2*, Aleksander, I. and J. Taylor (Eds.), Elsevier Sience.

Billings, S.A. (1980). Identification of Nonlinear Systems - a survey. *IEEE proceedings* Vol 127, N. 6, pp 272-285.

Chen, S.H. and Y.F. Liao (1998). Modular Recurrent Neural Networks for Mandarin Syllable Recognition. *IEEE Transactions on Neural Networks*, vol. 9, nº 6.

Fogel, L.J., A.J. Owens and M.J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*, Wiley.

Haber, R. and H. Unbenhauen (1990). Structure Identification of Non Linear Dynamic Systems - A survey on input/output Approaches. *Automatica* Vol 26, N. 4, pp651-677.

Happel, B.L.M. and J.M.J. Murre (1994). The design and evolution of modular neural network architectures. *Neural Networks*, 7: 985-1004.

Hunt, K.J., D. Sbarbaro, R. Zbikowski and P.J. Gawthrop (1993). Neural Networks for Control Systems - A Survey. *Automatica*, vol. 28, N. 26.

Jacobs, R.A. and M.I. Jordan (1993). Learning Piecewise Control Strategies in a Modular Neural Network Architecture. *IEEE Transactions on System, Man and Cybernetics*, vol 23, nº2.

Morcego, B. (2000). *Study of Modular Neural Networks for Nonlinear Dynamic Systems Modeling*. Ph.D. Thesis (in spanish), Universitat Politècnica de Catalunya.

Morcego, B., J.M. Fuertes and G. Cembrano (1996). Neural Modules: Networks with Constrained Architectures for Nonlinear Function Identification. *Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*.

Morcego, B., J.M. Fuertes, J. Codina, and G. Cembrano (1997). Modular Neural Network Framework for Nonlinear Dynamic Systems Modeling. *TEMPUS Workshop MODIFY'97*.

Sharkey, A.J. (Ed.) (1999). *Combining Artificial Neural Nets. Ensemble and Modular Multi-Net Systems*. Springer-Verlag.

Yao, X. (1993). Evolutionary Artificial Neural Networks. *International Journal of Neural Systems*, vol. 4, nº 3.