MIGA, A Software Tool for 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

JOSEP M. FUERTES
Automatic Control and Computer Engineering Department, Universitat Politècnica de Catalunya, C. Pau Gargallo, 5, 08028 Barcelona, Spain

GABRIELA CEMBRANO
Institut de Robòtica i Informàtica Industrial Universitat Politècnica de Catalunya, Consejo Superior de Investigaciones Científicas, Technological Park of Barcelona, U Building, St. LLorens i Artigas 4-6 2nd Floor, 08028 Barcelona, Spain

Abstract. This paper presents a software tool suitable for dynamic system modelling. The models generated by this tool are modular neural networks, see [1]. 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.

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 [2, 3] 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. [4]. 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

*This paper is a revised and extended version of one originally presented at IFAC Symposium on Computer Aided Control Systems Design—CACSD 2000.

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 (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. [5, 6].

Two important studies on the design and training of modular neural networks are those by Jacobs and Jordan [7] and Happel and Murre [9]. 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. [9].

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 4. 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 [10]) and, more precisely,

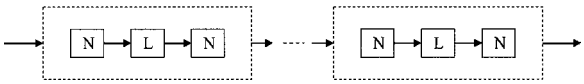

Figure 1. Block diagram of the model structure.

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. Each module has a very easy interpretation.

A neural module MANN 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. Top-Down 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. Both paradigms share here 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, model evolution and for user interaction.

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 Fig. 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 of the graph 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.

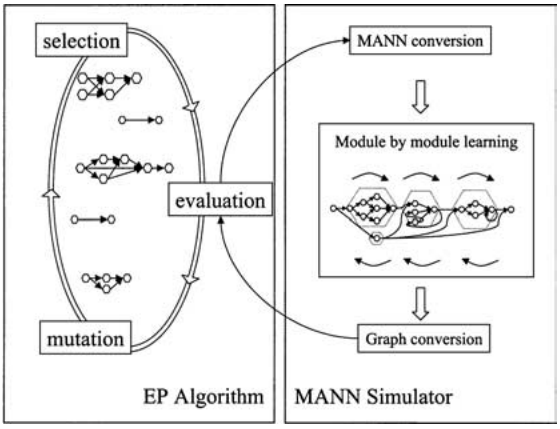

Figure 2. Operation diagram of the modelling tool.

2.2. *EP Algorithm*

The EP algorithm follows the classical outline given by Fogel [11]. It is, however, adapted to the evolution of neural networks as in Angeline [12], that is:

```
Pop := Create initial population at random
Evaluate each individual (x ∈ Pop, f)
while end condition ≠ FALSE do
   NewPop := mutation (Pop)
   Evaluate each individual (x∈NewPop, f)
   Pop := selection (Pop, NewPop)
   Evaluate end condition
end
```

The initial population is a set of N graphs. The nodes in those graphs are chosen according to their a priori 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 $\times$ default value, respectively.

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

The mutation operator is an important function in EP because it is the only function to explore the search space. Other evolutionary techniques rely on 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 function parameter change: a parameter is given its value following a uniform distribution centred at its actual value.
- Node insertion: it is a structural mutation. The inserted node is a unitary static gain. See Figs. 3 and 4 for examples of allowed and illegal insertions.
- Node function change: a new function is chosen for a node according to the a priori probabilities of the available functions.

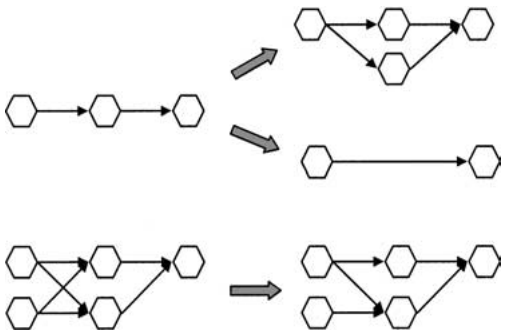

Figure 3. Three examples of allowed structural mutations. The first one corresponds to a node insertion; the second one is a node suppression and the last one is a connection suppression.

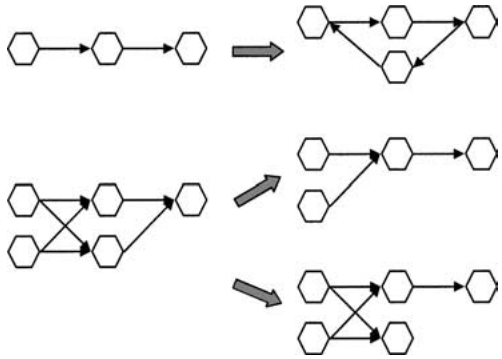

Figure 4. Three examples of illegal structural mutations. The first one is a node insertion; the second one is a node suppression and the last one is a connection suppression.

– Node or connection suppression: it is a structural mutation that can alter the model's behaviour considerably. See Figs. 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 $0.7 \times$ actual value.

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. In elitist selection the $2 \cdot N$ individuals are sorted by their fitness value and the first N are chosen. Competitive selection is based on 'tournaments' between sets of individuals. Each tournament compares the fitness of individuals chosen at random from the set of $2 \cdot N$ candidates. The winner is selected for the following generation. Here it is used the competitive selection, 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 graph into a modular neural network.
– Trains each MANN for a short number of epochs to fine-tune its parameters.
– Tests each MANN to assign it a fitness 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.

Training in modular neural networks can be accomplished in a sequential or a cooperative fashion. Sequential learning consists in training each module separately. When all the modules behave as expected they are connected appropriately and the network is ready for the recognition phase. Cooperative learning consists in training the network as a whole, in such a way that each module learns its corresponding subtask.

In our case, sequential learning is not possible because the structure of the network is unknown beforehand and, consequently, intermediate signals are not available. On the other hand, cooperative learning is more complex because the modules in the network need not be of the same type and combining different learning algorithms is rather laborious.

Most cooperative algorithms are strongly conditioned by the structure of the MANN, see [13] and [14]. There are algorithms, though, that allow arbitrary structures but are computationally expensive or difficult to implement for arbitrary modular structures, see [15] and [16].

The neural simulator implemented within `miga` is *Modular BackPropagation* (MBP), a learning algorithm developed in [17]. 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 handles the information flow between modules. The MANN is considered as a typical neural network being trained with backpropagation, but here the modules correspond to the neurones of the simple network.

MBP is computationally more efficient, see [17], than standard learning algorithms when applied to modular neural networks, specially when the MANN is a combination of feedforward and recurrent modules.

3. Modelling Process with `miga`

The typical modelling procedure is usually described as a sequence of five steps, eventually looping between them. Those steps are:

- Data collection and analysis
- Model structure selection
- Experiment design and execution
- Validation and analysis of results
- Model refinement

The modelling tool described in this paper is designed to aid the user in the three middle steps. The following subsections outline the process of dynamic system modelling with `miga`.

3.1. Data Collection and Analysis

`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.

Input-output sequences are usually obtained from experimental measurements. The procedures to obtain significant data go beyond the scope of this paper, although it is usually useful enough to apply band limited white noise to the plant. The resulting data is divided into 60% for training purposes (identification), 30% for test and the remaining 10% for the validation process. It is always desirable to have more data for validation, but it is also necessary to consider the computational cost of managing longer sequences which may not give any additional information about the system.

A priori knowledge allows biasing the search, which is partially stochastic. This information is represented by a set of functions (neural modules) and their associated probabilities of belonging to the solution. `Miga` provides a user interface to set the a priori probabilities of the module in the library of neural modules. This library is described in Section 4.

3.2. Experiment Design and Execution

The next step in the modelling process would be the selection of the model structure. In this case, the model is always a modular neural network and its internal structure will be obtained as a result.

The design and execution of a modelling-identification experiment is very much dependent on its aim. For instance, one may want to obtain the best model using certain a priori knowledge or minimising a specific error criterion. This tool allows the user to set experiments with different modelling objectives:

- A priori knowledge used. It is possible to use any subset of the neural module library with associated user modifiable probabilities.
- Error criterions. It is possible to search for models that minimise the *Sum of Squared Errors,* the *Final Prediction Error* criterion, the *Minimum Description Length* criterion or the *Akaike's Information Criterion*.
- Tool parameters. The modelling tool can also be parameterised in different ways, e.g. the stop fitness value, the maximum number of iterations, all the parameters explained in the previous section.

Once the user has selected all the necessary parameters the experiment is run.

3.3. Validation and Analysis of Results

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 (see Fig. 5). 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

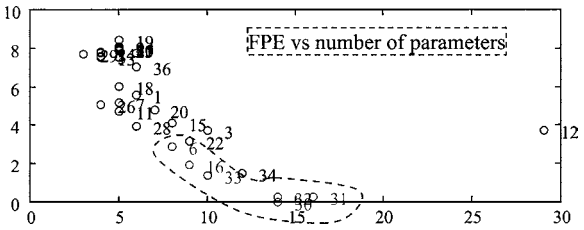

Figure 5. Plot and selection sample of the best 40 models in a run.

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.

4. Neural Module Library

The neural module library is described in detail in [17]. 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 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 6 depicts its input-output representation.

The neural module that approximates this function is represented in Fig. 7. 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.

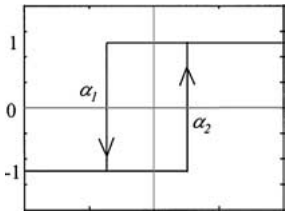

Figure 6. Ideal two-state hysteresis.

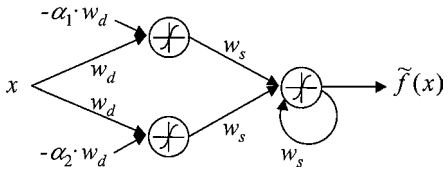

Figure 7. Neural module that approximates the threshold function with hysteresis. The values w_d and w_s force the expected behaviour and the values $\alpha_1 w_s$ and $\alpha_2 w_s$ correspond to the function's parameters.

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.

5. 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 Fig. 8 for block layout and details.

Data sequences, generated using Simulink, are 601 samples long (12 s 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 $\pm 1.5\%$ of the output. There is one sequence for each modelling phase: training, test and validation. Figure 9 shows an example sequence.

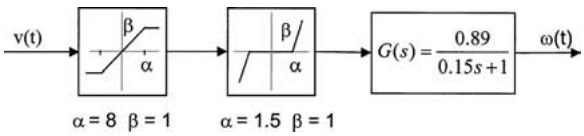

Figure 8. Original block diagram of the dc motor experiment. The first block is a saturation function of unitary slope and limits at ± 8 V. The second block is a dead zone function of amplitude 1.5 V. The third block models the dynamics of the system.

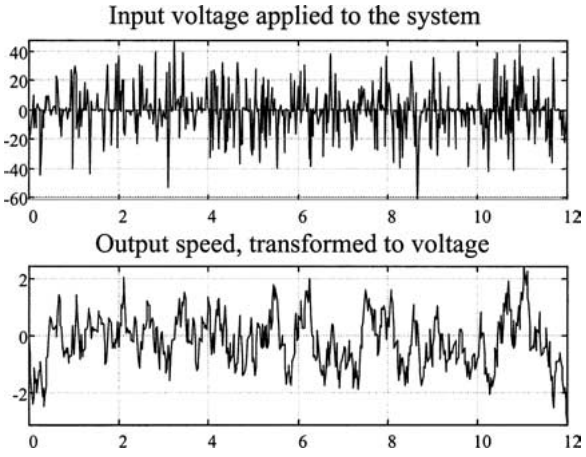

Figure 9. Sample input and output data sequences.

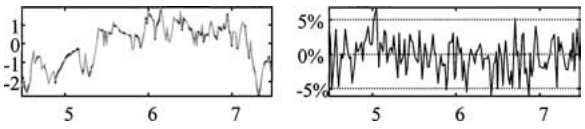

Figure 10. Partial output sequences of test and model output data (left) and their difference (right). The error lies in the 5% band.

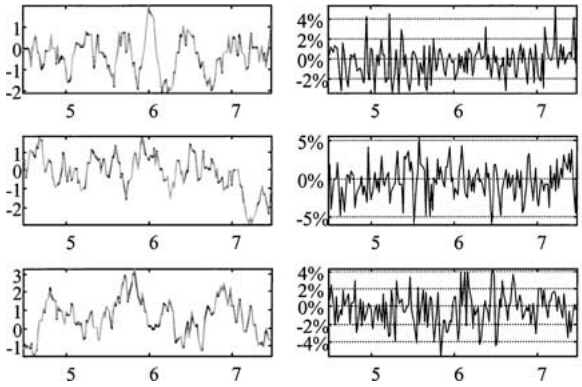

Figure 11. Partial output sequences of validation and model output data (left) and their difference (right). The error is similar to that on Fig. 10.

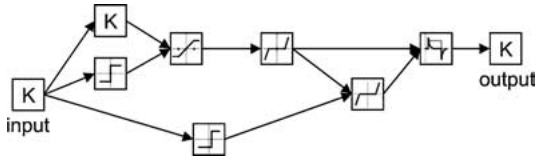

Figure 12. Block diagram of the post-processed final model. It has seven modules.

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 0.1 and 0.4 s.

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 10 shows part of the test and error sequences.

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 11 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 Fig. 10.

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 phase (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.

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 Fig. 12). This model makes explicit two features of the original system: a saturation function in the input path and a linear system of 0.15 s time constant (the module before the output).

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

6. 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.

Acknowledgments

This work is partially supported by the research grant CICYT TAP99-0748 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

1. A.J. Sharkey (Ed.), *Combining Artificial Neural Nets. Ensemble and Modular Multi-Net Systems*, Springer-Verlag, 1999.
2. S.A. Billings, "Identification of nonlinear systems—a survey," IEEE Proceedings vol. 127, no. 6, pp. 272–285, 1980.
3. R. Haber and H. Unbenhauen, "Structure identification of non linear dynamic systems—A survey on input/output approaches," *Automatica* vol. 26, no. 4, pp. 651–677, 1990.
4. K.J. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthrop, "Neural networks for control systems—A survey," *Automatica*, vol. 28, no. 26, 1993.
5. Y. Bennani and P. Gallinari, "Task decomposition through a modular connectionist architecture: A talker identification system," in *Artificial Neural Networks*, edited by I. Aleksander and J. Taylor, vol. 2, Elsevier Sience, 1992.
6. S.H. Chen and Y.F. Liao, "Modular recurrent neural networks for mandarin syllable recognition," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, 1998.
7. R.A. Jacobs and M.I. Jordan, "Learning piecewise control strategies in a modular neural network architecture," *IEEE Transactions on System, Man and Cybernetics*, vol. 23, no. 2, 1993.
8. B.L.M. Happel and J.M.J. Murre, "The design and evolution of modular neural network architectures," *Neural Networks*, vol. 7, pp. 985–1004, 1994.
9. B. Morcego, J.M. Fuertes, and G. Cembrano, "Neural modules: Networks with constrained architectures for nonlinear function identification," in *Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, 1996.
10. B. Morcego, J.M. Fuertes, J. Codina, and G. Cembrano, "Modular neural network framework for nonlinear dynamic systems modeling," *TEMPUS Workshop MODIFY'97*, 1997.
11. L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, Wiley, 1966.
12. P.J. Angeline, G.M. Saunders, and J.B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1. 1996.
13. R.A. Jacobs and M.I. Jordan, "Hierarchical mixtures of experts and the EM algorithm," *Neural Networks*, vol. 6, 1994.
14. T. Catfolis, "Mapping a complex temporal problem into a combination of static and dynamic networks," *SIGART Bulletin*, vol. 5, no. 3, 1994.
15. E.A. Wan and F. Beaufays, "Diagrammatic derivation of gradient algorithms for neural networks," *Neural Computation*, no. 8, 1995.
16. L. Bottou and P. Gallinari, "A framework for the cooperation of learning algorithms," *Advances in Neural Processing Systems* edited by R. Lippmann, J. Moody, and D. Touretzky (eds.), vol. 3, Morgan Kaufman, 1991.
17. B. Morcego, *Study of Modular Neural Networks for Nonlinear Dynamic Systems Modeling*. Ph.D. thesis (in Spanish), Universitat Politècnica de Catalunya, 2000.

Bernardo Morcego is an associate professor at the School of Industrial Engineering at the Technical University of Catalonia, Terrassa. He received his PhD in Computer Science from the Technical University of Catalonia in 2000. His research interests include computer aided or automated control systems in a broad sense, neural network and genetic algorithm applications.

Josep M. Fuertes, Industrial Engineer and PhD (born in Barcelona, 1949), has been Permanent Professor at the Technical University of Catalonia since 1987.

He was *Researcher* (1975–1986) and *Manager* (1982–1986) of the Electronic Design group at the Institut de Cibernètica (Spanish Consejo Superior de Investigaciones Científicas). In 1987 he had a position for a year at the Lawrence Berkeley Laboratory working as *Visiting Scientific Fellow* in the design of the Active Control System of the W.M. Keck 10 meter segmented telescope (Hawaii).

From 1988 he has coordinated projects at national and international level, related with the above areas of expertise and has participated with more than 80 papers on Distributed Control Systems and Advanced Neural Network Control Applications. He acted as Spanish *Representative* at the Council of the European Union Control Association (EUCA).

He is member of the Administrative Committee of the IEEE Industrial Electronics Society and member of other scientific and technical societies. He is director at the Automatic Control and Computer Engineering department at the Technical University of Catalonia.

Gabriela Cembrano received her MSc and PhD from the Polytechnical University of Catalonia (UPC) in 1984 and 1988 respectively. She is a researcher at the Institute of Robotics and Industrial Computing, a joint research center of the Spanish National Research Council (CSIC) and UPC. Her research area is Control Engineering, more specifically Identification and Control of complex systems, with applications in industry. She takes part in several long-term cooperative projects with industry in this area.