ISTANBUL UNIVERSITY –
JOURNAL OF ELECTRICAL & ELECTRONICS ENGINEERING

YEAR : 2003 (905-915)
VOLUME : 3
NUMBER : 2

A MEMORY ANN COMPUTING STRUCTURE FOR NONLINEAR SYSTEMS EMULATION IDENTIFICATION

Georgi M. DIMIROVSKI[1] **Cvetko J. ANDREESKI** [2]

[1] Department of Computer Eng., Dogus University, 34722 – Kadikoy / Istanbul, Turkey, and. Institute of ASE-FEE, SS Cyril and Methodius University, Skopje, Republic of Macedonia

[2] Computing Lab. of Faculty of Tourism, St. Clement Univ., 6000 Ohrid, R. of Macedonia

[1]E-mail: gdimirovski @ dogus.edu.tr [2]E-mail: cipuslju @ mt.net.mk

ABSTRACT

Currently, almost all efforts for using artificial neural networks for control oriented process identification are based on feed-forward networks. Provided the system order or the upper limit of the order is known, a neural network design is feasible for which all the collection of previous values of the inputs and outputs of the system to be identified can be used as input data to train in the network computing structures to learn the input-output map. This work reports on a novel technique that makes use of memory artificial neural network architecture that can learn and transform so as to emulate any non-linear input-output map for multi-input-multi-output systems when no prior knowledge on specific system features exists.

Keywords: *Artificial neural-net computing structures, memory artificial neural networks, non-linear input-output maps, system emulation identification.*

I. INTRODUCTION

In engineering and technology, model identification of certain system structures requires searching for some class of functions for approximation of the system input-output law of behavior, generally highly nonlinear, in the "best" possible way [4]. In most situations, such as in control process identification, pattern recognition etc., the value of the output of the representation model of the observed system is some function of the previous values of outputs and inputs.

By and large, the input-output view on model identification, in contrast to the input-state-output one (see Figure 1), has been exploited in industrial applications for identifying and control design of complex nonlinear plants [4]. Hence the formalism of artificial neural networks (ANN) has attracted much attention by researchers in systems and control engineering so

Received Date : *31.10.2002*
Accepted Date: 02.06.2003

that dedicated monographs have appeared [12], [14] in addition to the standard books as [5], [10].

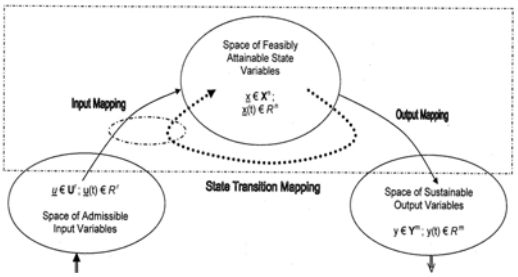

Figure 1. The input-output view on general system assumes only spaces of admissible inputs and sustainable outputs are available, most adequate in emulation modeling by ANN computing.

It has been shown by Narendra and Parthasarathy in [13] that some huge class of models can be constructed by using neural networks and linear filters interconnected in cascades and/or feedback configurations. Neural networks does match non-linear part of the system, and, linear filter gives dynamics to the system representation model. They worked on method for backpropagation-through-time of the error signal, so called dynamic back propagation. Using simulations they showed these kinds of models could identify some complex non-linear dynamic systems. This is the elegant way for understanding model dynamics, but there are some disadvantages. Namely, the knowledge of the system structure is prerequisite in order of the system to develop design combination of non-memory transformation and linear filters that are designed appropriately for emulating the representation model [2], [7].

In the course of development of artificial neural network theory, in parallel, there have been present considerable interests and research efforts about engaging neural networks in model identification for control and the software design and implementation of such soft-computing structures [5], [9], [10], [17]. This paper reports on such a designed realization based on a class of recurrent neural networks employing the neuronal architecture of memory ANN's [16] as a control oriented computational model for emulation identification of inherently generic nonlinear input-output (I/O) maps of either controlled objects or controller laws [2]. The aim of this paper is present an innovated design of a memory ANN computing structure for neural emulation identification of controller or object system nonlinear I/O maps for the purpose of complex systems control a well as to demonstrate the potential computational models with recurrent neural networks.

The memory artificial neural networks were invented and introduced in the literature on neural networks by Sastry and co-authors [16]. These represent a class of recurrent networks created via adding new elements in classic feed-forward ANN's, which can be trained by means of time sequences and which are able to memorize its previous activities. By making use of these abilities, these neural networks can identify dynamic systems with no need for previous values of the inputs and outputs. Thus they able even to identify dynamic systems with unknown specific features such as order or time delay [3], [8] , [16].

The organization of the paper is as follows. In Section II, in two subsections, an outline presentation of memory ANN's along with some of the implementation details on our memory ANN computing structure are given as well as a brief discussion on details of the training algorithm as adjusted for the implemented memory ANN computing structure. Section III, also in two subsections, presents an analysis of the emulation identification strategy along with the results for two examples of highly nonlinear I/O maps known as benchmark cases in the literature. Section IV discusses the performance of this ANN computing structure for emulation identification of nonlinear I/O maps via two characteristic examples. Conclusions are presented in Section V, and references follow thereafter.

II. AN OUTLINE OF MEMORY ANN COMPUTING STRUCTURES

To implement dynamics direct into the network structure, the network to be endowed with some "internal memory" and a learning algorithm for recurrent network. As for the learning algorithm, there is no much detail to present about possible strategies for system model emulation or for training a controller system. Both least-square and back-propagation learning alternatives have

been explored with similar success [6], [8], [11], [15], [18]. However, in this work the rather common strategy of back-propagation-through-time [18] has been found more adequate and employed in our memory ANN based technique [2]. Nonetheless, we emphasize that back-propagation combined with feed-forward neural networks can give erroneous results when current value of the output of the system depends on more than one of the previous inputs and especially when the case is an multi-input-multi-output (MIMO) system. And, MIMO systems precisely are of main concern in our work. That is to say, our focus was put on exploring the emulation of MIMO non-linear I/O maps of dynamical processes, where the values of some or of all the outputs essentially depend on more than one of the inputs.

II. 1. On Memory ANN Architecture

In the memory ANN architecture [16], each neuron has its own memory neuron and its output summarize the history of the previous actions of the neuron. These memory neurons or more precisely weight of the links presents the elements of dynamics for training of the model. Because links between neurons and their memory neurons are back-propagation links, the network from global point of view is indeed recurrent as pointed out in [1]. Also, it has been shown through a number of simulation experiments that the amount of internal memory and elements for training during the time are enough for identification of arbitrary non-linear I/O maps [16], [2].

The architecture of memory ANN based systems, proposed by Sastry et al. in [16], is presented in Figure 1 above. It is seen that their structure is similar to feed-forward networks with a new element - memory neurons, presented with little solid circles, connected on every network neuron, presented by big outlined circles. In order to make difference between these two kinds of neurons we should use titles network and memory neuron.

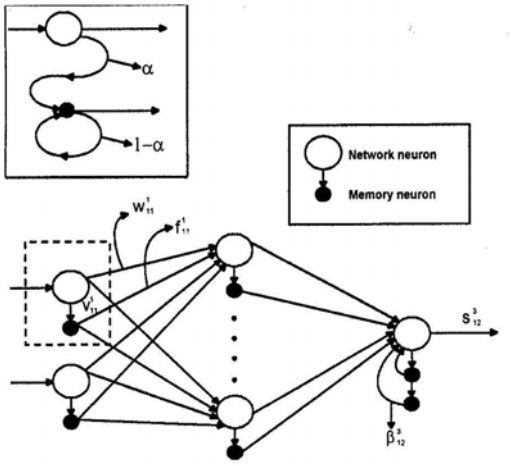

Figure 2. Directed graph representation of the architecture of memory artificial neuron networks.

As shown in Figure 2, on every level of the network, except the output level, every network neuron has one memory neuron connected. Memory neuron gets their input from its network neuron, and also they have own feedback. With this kind of propagation, data that are computed at the network neurons can be stored in memory neurons. Network, and memory neurons, from every level they send their output into the network neurons of the next level. In the output level, every network neuron can have cascade of memory neurons and each of them sends its output to the network neuron of the output level. Figure 2 presents a memory ANN with 2 input and 1 output node and one "hidden" level. This ANN computing structure is self-explained through the notation and comments given below:

- L - number of the network level with the level l as input and level L as output levels;
- N_l - number of network neurons on level l;
- $x_j^l(k)$ - network input in j network neuron on level l in the time period following instant k;
- $v_j^l(k)$ - output of the memory neuron from j network neuron of level l in the time period following instant k;

- $w_{ij}^{l}(k)$ - "weight" of the connection between *i* network neuron of level l and *j* network neuron on level $l+1$ in the time period following instant k;
- $f_{ij}^{l}(k)$ - "weight" of the connection between memory neuron of the *i* network neuron of level l and *j* network neuron of $l+1$ level in the time period following instant k;
- $\alpha_{j}^{l}(k)$ - "weight" of the connection between *j* network neuron of level l and its memory neuron in time period following instant k; $1 \le l < L$,
- $\alpha_{j}^{L}(k)$ - "weight" of the connection of *(j-1)* memory neuron to *j* memory neuron of *i* network neuron in output level in the time period following instant k;
- $v_{j}^{L}(k)$ - output of *j* memory neuron of *i* network neuron of the output level in the time period following instant k;
- $\beta_{ij}^{L}(k)$ - "weight" of the connection between *j* memory neuron of *i* network neuron and *i* network neuron of the output level in the time period following instant k;
- M_j - number of the memory neurons connected to j network neuron of the output level;
- $g(.)$ - activation function of the network neurons.
- Coefficients $\alpha_{j}^{l}, \alpha_{ij}^{L}, \beta_{ij}^{L}$ should be considered as memory coefficients.

Network input in *j* network neuron from level l, $1 \le l < L$, in the time period following instant k;
is given as:

$$x_{j}^{l}(k) = \sum_{i=0}^{Nl-1} w_{ij}^{l-1}(k) s_{i}^{l-1}(k) + \sum_{i=1}^{Nl-1} f_{ij}^{l-1}(k) v_{i}^{l-1}(k) \quad (1)$$

In equation (1) we should accept $s_{0}^{l} = 1$ for every *l*, and w_{oj}^{l} is the beginning condition for j network neuron of level *l*+1. These zero neurons of each level are given in accordance with beginning condition and that network neuron should not have memory neuron. That's why v_o^l does not exist and index of second sum begins from 1. The output of the network neuron is given with the following equation (2):

$$s_{j}^{l}(k) = g(x_{j}^{l}(k)), \; 1 \le l < L. \quad (2)$$

We use two different types of activation functions

$$g_1(k) = c_1 \frac{1}{1+\exp(-k_1 x)}, \qquad g_2(k) = c_2 \frac{1-\exp(-k_2 x)}{1+\exp(-k_2 x)} \quad (3)$$

in our ANN computing emulation and simulation technique. In here, g_1 is implemented for all nodes in hidden level, and g_2 for output nodes. In equations (3) c_1, c_2, k_1 and k_2 are parameters of activation function.

Network neurons from output level, gets their input values according to following equation (4):

$$x_{j}^{L} = \sum_{i=0}^{Nl-1} (k) s_{i}^{L-1}(k) + \sum_{i=1}^{Nl-1} f_{ij}^{L-1} v_{i}^{L-1}(k) + \sum_{i=1}^{M_j} \beta_{ji}^{L}(k) v_{ji}^{L}(k) \quad (4)$$

Output of all memory neurons except memory neurons from output level is computed as following:

$$v_{j}^{l}(k) = \alpha_{j}^{l}(k) s_{j}^{l}(k-1) + (1-\alpha_{j}^{l}(k)) v_{j}^{l}(k-1) \quad (5)$$

The output of the memory neurons from output level is computed as (6):

$$v_{ij}^{L}(k) = \alpha_{ij}^{L}(k) v_{ij-1}^{L}(k-1) + (1-\alpha i_{ij}^{L}(k)) v_{ij}^{L}(k-1), \quad (6)$$

where according to accepted notation $v_{i0}^{L} = s_{i}^{L}$.
For safe stability of the network dynamics following constraints are implemented:

$$0 \le \alpha_{ij}^{L}, \; \alpha_{i}^{l}, \; \beta_{ij}^{L} \le 1. \quad (7)$$

From the previous text and this equations it is easy to note the similarities between feed-forward and memory neural networks. At the memory neural networks, network neurons are connected through weights w_{ij}^{l} creating feed-forward network. Every memory neuron

memorizes combination of all of the previous actions that happened in the network neuron.

The output value of the memory neuron is computed as output from a first-order filter. By keeping memory coefficients into the interval (0, 1) the stability of this filter is ensured. These outputs are easy to compute locally, by storing values of one time interval, using equations (5) and (6). Because outputs from memory neurons gives contribution to the input of the network neurons of the next level, interval memory of the network plays significant role in determining the output of the network in every time interval. Memory neurons of the output level allow direct dependence of the current input of the network from its previous output values.

According what is presented above, these kinds of artificial neural networks are recurrent, and the computation of the output values is similar to the one of the "feed-forward" networks. Therefore there is no waiting for the network to get in stable condition. So, the output of the network in any time interval is dependent of all previous network inputs (as we can see from (5) and (6)). The degree of dependence should be determined by means of the values of the memory coefficients.

II. 2. On Training algorithms for Memory ANN Computing Structures

As it is explained earlier in this text, on every new time interval network gets new input value and computes output using equations (1)-(6). On the output level network gives signal for adjusting which is involved in the error computation of the output level and according to error values, network makes adjustment on all "weights" in the network. It uses common criteria for square error given as

$$e(k) = \sum_{j=1}^{Nl} (s_j^L(k) - y_j(k))^2 , \qquad (8)$$

where $y_j(k)$ is the training signal of the *j-th* output node in time instant *k*.

This network uses training algorithm of error back- propagation type [18]. Everything that the network need is the error derivation $e(k)$, with respect to the network weights. According to attendance of the memory neurons, it is not an easy task to compute precise partial derivation through back-propagation in only one time interval. The strategy is approximately to develop a network with connected input of one time step, and the computed error to put in back propagation. That means that all the weights can be adjusted in time interval k with no need for additional input of the previous actions of the lines that are the essence for implementation of (5) and (6).

The final equations for adjusting the weights are given with the following equations:

$$w_{ij}^l(k+1) = w_{ij}^l(k) - \eta e_j^{l+1}(k) s_i^l(k) , \; 1 \le l < L, \qquad (9)$$

where η is the step

$$e_j^L(k) = (s_j^L(k) - y_j(k)) g'(x_j^L(k)) , \qquad (10a)$$

$$e_j^l(k) = g'(x_j^l(k)) \sum_{p=1}^{Nl+1} e_p^{l+1}(k) w_{jp}^l(k) , \; 1 \le l < L, \qquad (10b)$$

The previous equations are given for the standard error back-propagation processing with no respect to the memory neurons. Function $g'(.)$ is the derivation of the activation function of the network neurons, and we should use proper function according to the indicating number of the level.

The adjustments that needed for f are similar to ones for w, with some differences for the connected memory neuron. Namely, in this case the following equation

$$f_{ij}^l(k+1) = f_{ij}^l(k) - \eta e_j^{l+1}(k) v_i^l(k) , \; 1 \le l < L, \qquad (11)$$

is employed. Different memory coefficients can be adjusted according to following equations:

$$\alpha_j^l((k+1) = \alpha_j^l(k) - \eta' \frac{\partial e}{\partial v_j^l}(k) \frac{\partial v_j^l}{\partial \alpha_j^l}(k), \; 1 \le l < L, \qquad (12)$$

$$\alpha_{ij}^L((k+1) = \alpha_j^L(k) - \eta' \frac{\partial e}{\partial v_{ij}^l}(k) \frac{\partial v_j^l}{\partial \alpha_{ij}^L}(k), \; 1 \le l < L, \qquad (13)$$

$$\beta_{ij}^L(k+1) = \beta_{ij}^L(k) - \eta' e_i^L(k) v_{ij}^L(k) , \qquad (14)$$

where:

$$\frac{\partial e}{\partial v_j^l} = \sum_{s=1}^{Nl+1} f_{js}^l(k) e_s^{l+1}(k), \quad (15)$$

$$\frac{\partial v_j^l}{\partial \alpha_j^l}(k) = s_j^l(k-1) - v_j^l(k-1), \quad (16)$$

$$\frac{\partial e}{\partial v_{ij}^L}(k) = \beta_{ij}^L(k) e_i^L(k), \quad (17)$$

$$\frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L} = v_{ij}^L(k-1) - v_{ij}^L(k-1). \quad (18)$$

In the previous equations, two different parameters are used as step parameters: η' for memory coefficients and η for all of the other weights. If the memory coefficients after previous computations went out of interval (0, 1), these are put back in that interval to ensure stability of the network.

III. AN INNOVATED MEMORY ANN COMPUTNG STRUCTURE FOR NONLINEAR SYSTEM EMULATION

The overall conceptual model of this system identification structure [2], [12]-[14] is depicted in Figure 3. The presented schematic, for the sake of simplicity, depicts the case of SISO systems, where $u(k)$ and $y_p(k)$ are the respective scalar-valued system input and output signals. However, it is valid for the MIMO case when $u(k)$ and $y_p(k)$, respectively are vector-valued time sequences. Also note that, for simplicity, we observe memory neural network with one input, one output, and one hidden layer levels.

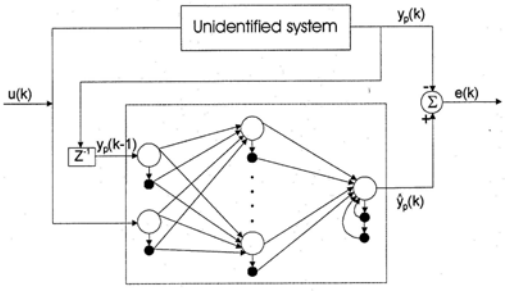

Figure 3. Schematic diagram of emulation simulation of the unidentified system.

III.1. An Analysis of the Algorithm and Conceptual Model of System Identification

In time period between two consecutive time instances, let the input value be connected to $u(k)$ and the output of the network be $\hat{y}_p(k)$. $y_p(k)$ should be used as a training signal in time period implied by time instant k. Accordingly, the following approximation expression applies

$$\hat{y}_p(k) = F(u(k), u(k-1), ..., \hat{y}_p(k-1), \hat{y}_p(k-2)....), \quad (19)$$

where F is the nonlinear transformation performed by the memory ANN structure. As it is written, $\hat{y}_p(k)$ depends of the previous input values (from the memory neurons of input and hidden level and previous values of the outputs). Model presented with (19) is known as a model with parallel identification.

Now, if the input of the system is the same as then current network input and if system output is used as a training signal, then the memory neural network is a model with parallel identification. This is not case in the networks based on feed-forward strategy, where one should decide how many of the previous values should be connected on the network input and how many of the previous outputs should be put in backward connection to have model with parallel identification.

Should there be allowed that value of the model output depends on several previous values of the system output, then the so called serial-parallel model for identification is obtained. In this paper we present serial-parallel model by using network with two inputs $u(k)$ and $\hat{y}_p(k-1)$. The network output is $\hat{y}_p(k)$. Writing the network output with two inputs as function of all of the previous inputs, similarly to (19), one obtains:

$$\hat{y}_p(k) = F(u(k), u(k-1), yp.(k-1), ..., \hat{y}_p(k-1), \hat{y}_p(k-2)....) \quad (20)$$

It is an easy task to see that $\hat{y}_p(k)$ will depend on previous values of the inputs and outputs of

the system and on the current input too. Network output depends on its previous values. This dependence can be cast off by simple disconnecting of the memory neurons from the output level. We should emphasize once more that, to get serial-parallel model, we do not need to know the system order. The network will automatically learn the relative "weights" which should be assigned to different previous values and these "weights" are available through memory neurons. It is well known that serial-parallel model can be engaged in generating of the stable adaptive laws. For identification of the system with m inputs and p outputs we should use network with $m+p$ inputs and p outputs. Current outputs of the system in every time interval are taken as training signals.

In proving the convergence of the given identification algorithm, it is essential to determine if the derivations computed with the algorithm are valid and can we expect good following of the gradient according to given algorithm. "In this case we can say that approximations made by construction of the network in time period of one step should be satisfied. More analysis are necessary to find what class of systems these approximations do not satisfy.

Second and vary important aspect of convergence encompass basic techniques for identification using gradient methods and models of neural networks. All these methods compute the gradient of the current error $e(k)$ using it for calibration of the "weights". For justification of the identification model, we need to show that following of the gradient of current error will result by algorithm, which will minimize some important errors. It can be proved that algorithm that follows gradient of the current error results in minimization of the expected value of the error, if the system input is independent and uniform distributed sequence and the unknown system is stable in sense limited input limited output and controllable. It should be noted that here we do not prove the level of accuracy of the identification method.

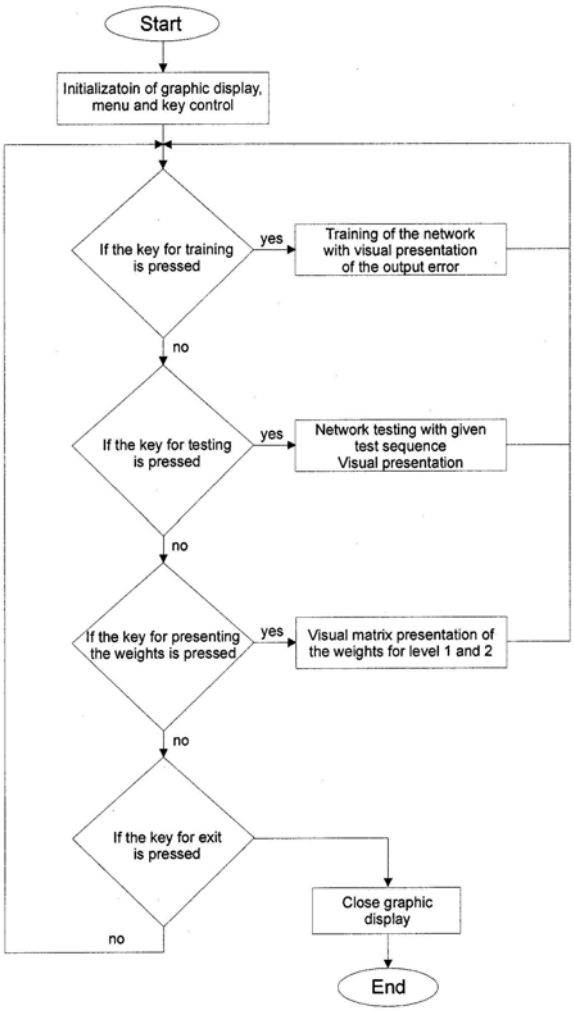

Figure 4. A simplified flow diagram representation of the designed software implementation of memory ANN computing structures.

Let assume that an independent and uniform distributed input time sequence in interval [-1, 1] is used for training the ANN structure. Then, proved [2] that algorithm minimize the expected value of the error between system to be emulated and the memory ANN for this normalized input. For better network representation of the system with other signals it is necessary, however, to have "enough amount" for input during the training time. So, this way, the training procedure will converge to an ANN computing structure that is the closest one to the system to be emulated.

If the training parameter is small, than algorithm has the same asymptotic behavior as algorithm for the gradient following, which makes use of average error rather than current error. The input sequence should be stationary, in order to get average effects. In addition, stability is needed in sense limited input, limited output and controllable of the system just to insure that all other signals in the network should be stationary and according to that to provide existence of invariant act necessary for proving convergence of the error.

III.2. On the Application Software Implementation

This application has been implemented on a standard PC platform and works with minimum requirements of a VGA graphic card [2]. Figure 4 depicts a simplified operational flow diagram of the package. The overall constructive composition of modules with the respective information processing engines can be inferred from this figure. It has been written in C+ language.

Upon activation of the application software, all weights in the memory ANN structure are set to have value 0.5. After the start initialization of this application software, the execution control of information processing performed is managed by the user via appropriate pressed keys within the sequence of user's actions. By pressing key '1' network training is activated, key '2' is for network testing, key '3' displays the weights matrix for layer 1 and 2 of the network, and key 'Esc' is aimed at terminating the current investigation experiment.

IV. EXAMPLES: PERFORMANCE OF MEMORY ANN SYSTEM EMULATION

In the sequel we present two rather characteristic examples of nonlinear systems identified by memory ANN computing structure. In these examples we use serial-parallel conceptual model for identification. The overall system identification structure is presented in Figure 3.

In its present implementation on a standard PC platform, memory ANN structure employs only one hidden layer. The standard notation *m:n* denotes a memory ANN having *m* hidden network neurons and *n* memory neurons on each node of the output layer. The number of input and output nodes is determined according to the system to be emulated; e.g., for a systems with one input and one output we should have two inputs and one output. However, the number of the inputs in this conceptual model for ANN emulation of unknown systems does not depend on the system order. In contrast to other methods, this emulation identification algorithm does not need to 'know' the structure of the system or of any of its subsystems.

The two examples taken into consideration for illustrating the performance of this technique represent complex nonlinear systems, which are stable in sense limited input, limited output. These are known in the literature and may well be said that serve some kind of benchmark purpose. First example system is a SISO system model of an executive control law for manipulating a robot arm. The second one, the MIMO system model, is found among the examples in [ ] used by Narendra and Parthasarathy (1990), and represents a system with two inputs and two outputs. In both cases, the same ANN computing structure and training sequence were applied so that the obtained results are amenable for comparison discussion. For the both examples network structure is an 6:1 memory ANN computing structure.

As it was explained earlier in this text, memory neurons in the output layer do not play any significant role. The network makes use of one or zero memory neurons. In the following text we present results of one considerably large memory ANN computing structure. This network is a 6:1 structure, meaning that it has six network neurons in hidden layer and one memory neuron in the output layer. The speed of learning for both of the systems is by step $\eta' = 0.1$ for the memory coefficients, and step $\eta = 0.2$ for all of the rest weights. Also, both networks employ the same activation functions: g_1 for the hidden nodes, and g_2 for the output nodes, with parameters $c_1=c_2=1$ and $k_1=k_2=1$, respectively. The memory ANN has an embedded constant for the system output that keeps the network signal in the interval [-1, 1]. However, this has no influence whatsoever on the identification process. It is an easy matter to note that the main purpose is to explain in general the structure of

the network. So, each problem, any calibration will be within in the framework of the numbers of the training iterations.

For training the network we use 62000 or 77000 time intervals, or long sequence for training for complex systems. Network training begin with 200 iterations for zero input, after that 2/3 of remaining time for training the input value is uniform sequence in the interval [-1, 1] and the rest of the training time input value is the sinusoid given by sin(πk/45). Upon the completion of the training session, we made s comparison of the output of the memory ANN emulation structure with that of math-analytical based system output by making use of test-signal time sequence of 1000 time steps. Typically, the test signal used is a blended mix of sinusoidal and constant input values generators.

Example 1. First example is a single-input-single-output (SISO) system model, taken from the literature, represents a given manipulation law of an robot arm. The system model is given with following equation:

$$y_p(k+1) = 0.8\sin(2 \cdot y_p(k)) + 1.2u_1(k) \qquad (21)$$

In order to insure operating stability of the network, output is scaled to be in [-1, 1] interval. Through the training procedure, network computes the proper values of the memory coefficients in order to reproduce the behavior of the system for the input sequence (22).

As a test-signal sequence we have used a typical time sequence for exploring the performance of ANN based identification methods. In order to generate the test signal a standard dynamic neural network employing first-order filter has been used. The test-signal sequence is generated by the following mathematical representation:

$$u(k) = \begin{Bmatrix} \sin(\pi k/25),\ k < 250 \\ 1.00,\ 250 \le k < 500 \\ -1.00,\ 500 \le k < 750 \\ 0.3\sin(\pi k/25) + 0.1\sin(\pi k/32) + \\ +0.6(\pi k/10),\ 750 \le k < 1000 \end{Bmatrix} \quad (22)$$

The computed outputs (relative to the time) of this example system, Eq. (21), via both MAN based simulation (blue graphic) and memory ANN based identification (red graphic) are presented in Figure 5 (a) below. These are superimposed on each other, hence it is clearly seen that there is almost no discrepancy among them indeed, demonstrating the achieved very high accuracy of the implemented memory ANN computing structure.

Example 2. The other test example is a multi-input-multi-output (MIMO) system model, notably with 2 inputs and 2 outputs, also taken from the literature. For this example, a memory ANN computing structure employing 4 neurons in the input layer and two neurons in the output layer has been used. The example of a MIMO system is represented by equations:

$$y_{p1}(k+1) = 0.5 \cdot \left[\frac{y_{p1}(k)}{1 + y_{p2}^2(k)} + u_1(k) \right], \quad (23)$$

$$y_{p2}(k+1) = 0.5 \cdot \left[\frac{y_{p1}(k) \cdot y_{p2}9k)}{1 + y_{p2}^2(k)} + u_2(k) \right]$$

.

In Figure 5 (b), there is depicted a comparative presentation of superimposed systems outputs computed via the math-analytical model (red graphics) and the memory ANN emulation structure (blue graphics) for the test signal given by Eq. (22). It is easy to see that emulation identification of the system with two inputs and two outputs is as good as for the SISO case. In the sequel, we discuss possible errors that can happen during the programming of the memory ANN computing structures [2].

Input values of the memory neural networks should be in the interval [-1,1], and so we should check the network output to be in specified interval. If the system formula leads the system output out of the interval [-1,1] then we project that values back to the specified interval. Also we should take care for the values of the memory coefficients $\alpha_j^l, \alpha_{ij}^L$ and β_{ij}^L. If any of these coefficients went out of the (0, 1) interval they should be projected back to that interval. According to their values current actions of the network depends more or less on the previous ones, so that's why projection procedure is very

important in order to keep ratio between the values of these memory coefficients.

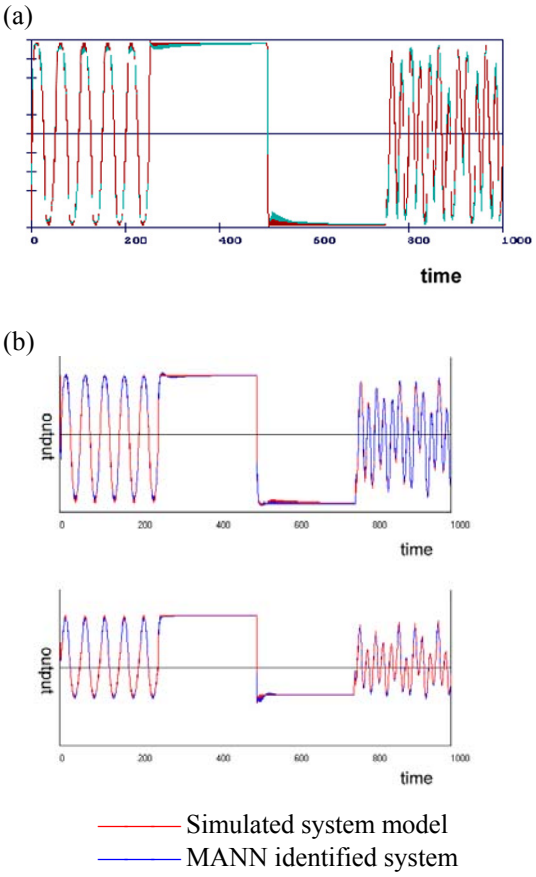

Figure 5. Computational performance: Output results of math-analytical numerical and memory ANN computing for the nonlinaer SISO and MIMO system examples.

IV. CONCLUSION

An memory neural-network computing structure for emulation identification of highly nonlinear unknown systems that makes use of the so called serial-parallel model of system identification [2] has been presented in this paper. First, a brief outline the original results on memory ANN architecture [16] and the respective representation equations has been given, and a discussion on the learning-training algorithm has been presented. Thereafter a summary reference on the applications software design by making use of a simplified flow diagram has been presented. The details the performance of this ANN computing structure have been discussed along with results on two well-known example system models, one for the SISO and another for MIMO cases of unknown systems..

In this research we have worked with the typical test signal time sequence, given by Eq. (22) to emulate nonlinear system models. It should be noted that dynamic neural networks are the closest to memory neural networks. Advantage of the memory neural networks in comparison to those with dynamics is the fact that knowledge of the system order is not needed a-priori for the system identification.

The results achieved in emulation identification of two examples of notoriously nonlinear system models have demonstrated a high-quality performance. The is no noticeable discrepancy between the computed responses of the systems models and of the trained memory ANN computing structure. Memory ANN structures can be successfully trained to learn and emulate nonlinear input-output system mappings of arbitrary complexity

ACKNOWLEDGEMENT

Professor Dimirovski would like to express his gratitude to Prof. Dr. Kurt Schlacher, Head of Department of Automatic Control & Control Systems Tcehnology at Johennes Kepler University in Linz, Austria, for providing excellent working and living environment inviting during his visiting position there when the theoretical part of this work was elaborated.

REFERENCES

[1] F. Albertini and E. D. Sontag, “For neural networks, function determines form.” *Neural Networks,* vol. 6, pp. 975-990, 1992.

[2] C. J. Andreeski, C.J. and G. .M. Dimirovski, Theory, Design and Implementation of Memory ANN Based Computational Technique for Identification of MIMO Non-linear Dynamical Processes, *ASE-FEE Technical Research Rep. NNSID-03/ASE00.* Skopje, MK: SS Cyril & Methodius University, 2000.

[3] V. I. Arnold, "Some questions of approximation and representation of functions." In: *Proceedings of the International Congress of Mathematicians*, pp. 339-348. (Translation in English: American Mathematical Society Translations, Vol. 53). Moscow, RUS: MEI Press, 1958.

[4] K. J. Aström, P. Albertos, M. Blanke, A. Isidori, W. Schaufelberger, and R. Sanz, Eds., *Control of Complex Systems*. London, UK: Springer-Verlag, 2001.

[5] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Clarendon, 1995.

[6] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks." *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 302-309, 1991.

[7] T. Chen and H. Chen, "Approximation of continuous functionals by neural networks with applications to dynamic systems." IEEE Trans. Neural Networks, vol. 4, no. 3, pp. 910-918, 1993.

[8] G. M. Dimirovski, I.I. Ivanoska, M.J. Stankovski, R.A. Pessu, A. Zakeri and N.E. Gough. "Life-like systems: ANN emulation modelling in intelligent control systems." In: *Proceedings of the 3rd APCA Intl. Conf. on Automatic Control,* A. Dourado et al., Eds., vol. 1, pp. 169-176. Coimbra, PT: Assoc. Portuguesa de Controlo Automatico and University of Coimbra, 1998.

[9] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd ed.).* New York, NY: Macmillan, 1999.

[10] H. Katsuura and D. A. Sprecher, "Computational aspects of Kolmogorov's theorem." *Neural Networks,* vol. 7, pp. 455-461, 1994.

[11] S. Mcloone and G. W. Irwin, "Fast parallel of-line training of multilayer perceptrons." *IEEE Trans. Neural Networks,* vol. 8, no. 3, pp. 646-653, 1997.

[12] W. T. Miller, R. S. Sutton, and P. J. Werbos, *Neural Networks for Control*. Cambrdige, MA: MIT Press, 1992.

[13] K. S. Narendra and K. Parthasarathy, "Identification and control using of dynamical systems neural networks." *IEEE Trans. Neural Networks,* vol. NN-1, no. 1, pp. 4-27, 1990.

[14] D. T. Pham and L. Xing, *Neural Networks for Identification, Prediction and Control.* London, UK: Springer-Verlag, 1997.

[15] R. Parisi, E. D. Diclandid, G. Orlandi, and B. D. Rao, "A generalized learning paradigm exploiting the structure of feedforward neural networks." IEEE Trans. Neural Networks, vol. 7, no. 6, pp. 1450-1460, 1996.

[16] P. S. Sastry, G. Santharam, and K. P. Unnikrishnan, "Memory neuron networks for identification and control of dynamical systems." *IEEE Trans. Neural Networks*, vol. NN-5, no. 2, pp. 306 –319, 1994.

[17] D. A. Sprecher, "A numerical implementation of Kolmogorov's superposition II." *Neural Networks,* vol. 10, pp. 447-457, 1997.

[18] P. J. Werbos, "Backpropagation through time: What it does and how to do it." *IEEE Proceedings*, vol. 78, pp. 1550-1560, 1990.