Neural Network Modeling of Discrete-Time Chaotic Maps

Homayoun Yousefi'zadeh
Center for Pervasive Communications
Dept. of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92697
hyousefi@uci.edu

Edmond A. Jonckheere
Dept. of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089
jonkhee@usc.edu

***Abstract*— Although chaotic systems have received increasing attention over the past decades, traditional modeling tools have always encountered considerable analytical and numerical difficulties in modeling and predicting the behavior of chaotic systems. Neural networks, on the other hand, seem to be able to introduce a powerful modeling tool relying on their nonlinear nature for the task of modeling. In this paper, we introduce a novel scheme for the modeling task of multi-dimensional discrete-time chaotic maps relying on the capabilities of perceptron neural networks and present some of the related experimental results.**

***Index Terms*— Chaos, Discrete-Time Chaotic Maps, Fractals, Perceptron Neural Networks, Back Propagation Learning Algorithm.**

I. Introduction

CHAOS is a nonlinear phenomenon that manifests itself in many fields of science. Despite being singled out as an important research area only recently, chaos has been around for a long time, and dismissed as physical noise. We now know that chaos can readily occur everywhere nonlinearity is present. In addition, chaos can occur in linear systems with infinite dimensions. Studying chaotic phenomena in discrete-time dynamical systems and modeling such time series is very attractive because they are naturally observed in many physical, social, and economical systems. They also frequently arise in the Poincare analysis of systems modeled by ordinary differential equations. Modeling chaotic time series, however, encounters the significant challenge of analyzing very complex dynamics.

Neural networks are a class of nonlinear systems capable of learning and performing tasks accomplished by other systems. Their broad range of applications includes speech and signal processing, pattern recognition, system modeling, and servo mechanism control. The various kinds of neural networks, generally, have energy functions. The learning procedure of neural networks is, indeed, nothing more than decreasing these energy functions until reaching local minimum levels. Neural networks acquire the required information from the examples supplied to them in their learning procedure. Systems with neural network building blocks are robust in the sense that the occurrence of small errors in the systems does not interfere with the proper operation of the system. This characteristic of the neural networks makes them quite suitable for the time series prediction task discussed in this paper.

An outline of the paper follows. In section [2], we briefly review the qualitative behavior of discrete-time transformations. In section [3], we analyze the modeling tool perceptron neural network and its corresponding learning algorithm back propagation. In section [4], we explain the modeling scheme of discrete-time dynamical transformations with neural networks. In section [5], we summarize the experimental findings.

II. Qualitative Behavior of Discrete-Time Maps

In the present section, we discuss the characteristics of discrete-time maps capable of exhibiting chaotic behavior. The discussion is based on the literature material available in [2], [3], [4], [5], [6], and [7].

Consider a real closed interval $I = [a, b]$ and a nonlinear function $f(.)$ which transforms any point x of I into a point $x\prime$ in the same interval, i.e, $f : I \rightarrow I$. Choose an arbitrarily initial condition x_0 in I and iterate it via the algorithm

$$x_{n+1} = f(x_n) \qquad n = 0, 1, 2, .. \tag{1}$$

Relation 1 is called a discrete map over the interval I. We can generalize this scalar discrete map into an m-

dimensional interval map via the vector equation

$$\boldsymbol{x}_{n+1} = \boldsymbol{f}(\boldsymbol{x}_n) \tag{2}$$

where $\boldsymbol{x}_{n+1}$ and $\boldsymbol{x}_n$ are m-dimensional vectors and $\boldsymbol{f}(.)$ is a nonlinear vector function on $\boldsymbol{x}_n$. Each component x_{n_i} of $\boldsymbol{x}_n$ takes values in a closed interval $I_i = [a_i, b_i]$. We can interpret 1 and 2 as dynamical systems where n plays the role of the time variable. In spite of their apparent simplicity, the dynamics of many simple nonlinear, even polynomial, discrete maps are extremely complicated. They may have stable fixed points, periodic orbits of different periodicity, and chaotic regimes. Moreover, bifurcation among these behaviors can be observed in parameter-dependent family of maps over various nonempty parameter ranges.

As a benchmark example, we study the behavior of the logistic map which is a one-dimensional discrete map. The equation of the map is

$$x_{n+1} = 4W\,.\,x_n\,.\,(1 - x_n) = f(x_n,\, W) \tag{3}$$

where $I = [0, 1]$ and W is a parameter which varies from 0 to 1. The logistic map is an example of a one-parameter family of discrete maps which transforms the interval I into itself.

Although the logistic map is a very peculiar one, its qualitative behavior holds for a broad class of one-dimensional maps as shown in figure 1. Specifically, for those maps where the function $f(.)$ in 1 is a differentiable unimodal function, the same qualitative behavior is observed as the qualitative behavior of the logistic map. More specifically, the rate of decay

$$K_u = \frac{w_{k+1} - w_k}{w_k - w_{k-1}} \tag{4}$$

of the interval width between period doubling for the logistic map has been found to be the same for any differentiable unimodal function. This property is, sometimes referred to as metrical universality.

Higher dimensional discrete maps can be either conservative (area-preserving) or dissipative (area-contracting), invertible or non-invertible. While some results on one-dimensional mappings can be generalized to higher dimensions, the latter are usually much richer in their dynamical behavior than the former. For instance, consider the well known two-dimensional Henon map described by

$$\begin{aligned} x_{n+1} &= 1 - Ax_n^2 + y_n \\ y_{n+1} &= Bx_n \end{aligned} \tag{5}$$

This map which reduces to a one-dimensional quadratic map for $B = 0$ reveals many new phenomena quite typical of multi-dimensional discrete maps. First, the asymptotic behavior of the system depends on the initial point, i.e, different initial points could give rise to different periodic or aperiodic orbits. This behavior cannot occur in one-dimensional unimodal maps, where at most one stable period-1 orbit can exist. Second, for certain parameters and initial points, the system converges to an attractor with a self-similar internal structure. Third, it has been proved [3] that there exist intersections between stable and unstable manifolds of the Henon map. These intersections are called homoclinic points and give rise to extremely complicated dynamics, including chaos.

III. Perceptron Neural Networks and Back Propagation Algorithm.

Perceptron neural networks and their learning algorithm back propagation algorithm (BPA) have been studied extensively in the literature [9], [10],[11], [12], [13], [14], [15], [16]. In this section, we provide a brief overview of the literature work.

A. Perceptron Neural Networks

In an artificial neural network, the unit similar to the neuron is called processing element (PE). A PE has a large number of input paths and combines them by a simple weighted addition. Usually, the combined input is further processed by a transfer function. This transfer function may have the form of a threshold function which passes the information only when the combined input signal has reached a certain level, or it may be a continuous function from the combined input signal.

Normally, the output of the transfer function connects directly to the output path of the PE. The output path of the PE may be connected to the input paths of other PEs by a number of weighting functions. Each of the individual input signals to a PE is adjusted by these weighting functions before combining with other input signals in the transfer function.

Generally, a neural network contains a large number of PEs. PEs are arranged in layers. A network usually contains several layers with complete or random connections between consecutive layers. The input layer takes the data from the outside environment, the output layer returns the data to the outside environment, and the other layers known as hidden layers process the data inside the network. Figure 2 shows a simple network. There are two phases in the operation of a neural network. The first phase is called learning and the second phase is called recalling. It is possible to have one combined learning and recalling phase in a network, although in most of the networks these phases are separate. The learning procedure is, indeed, nothing but the adjustment of weighting functions such that the network can respond suitably to the

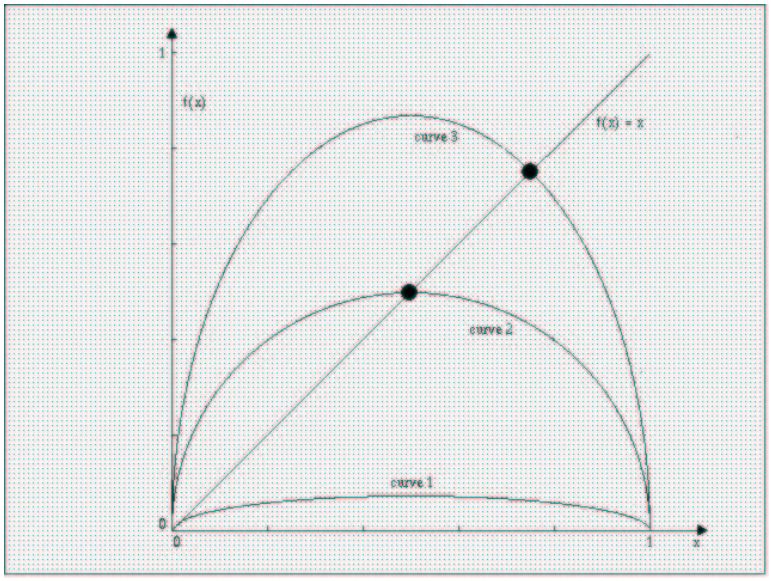

Fig. 1. Fixed points of $f(x)$ for $W < 0.25$ (curve 1) $0.25 < W < 0.75$ (curve 2) and $W > 0.75$ (curve 3)

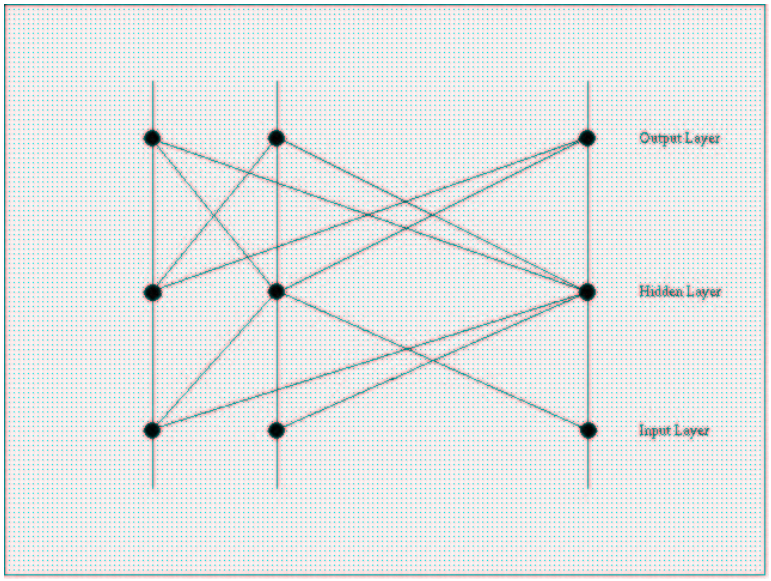

Fig. 2. A simple neural network with one hidden layer

input stimulation. If the desired output(s) is(are) different from the input(s), then the trained network is called Hetero Associative (HA), otherwise it is called Auto Associative (AA). If in the learning process, the network does not have access to the desired output(s), this procedure is called Unsupervised Learning (UL), and if the network does have access to desired output(s) the learning procedure is called Supervised Learning (SL). There is also a combination of the two-mentioned learnings called Reinforced Learning (RL). In the latter case, the supervisor just let the network know whether its output is suitable for a specified input or not.

The number of samples required for training depends on the system which the network attempts to model. There are cases in which the network needs several hundred thousand samples to be trained.

The recalling phase is a situation in which the network is able to create a suitable response after the appearance of a specific stimulation. This phase may be part of the learning phase as this is the case when the network output must be compared to a desired output in order to create an error signal.

A simple network does not have any feedback connection between two different layers or a layer with itself. Such kind of network is called a feedforward network. In this situation, the input data from the input layer appears in the output layer via the interface of hidden layers. Feedforward networks are generally considered because of their

nonlinear properties. The data flow in feedback networks is more complicated.

A network may operate as a synchronous network, i.e, all of the PEs send their outputs at the same time or as an asynchronous network, i.e, every PE may send its output at a time independent of the other PEs.

There are two separate operations for layers. The first one is called normalization. In this case the output of the layer is adjusted at a fixed level. In biological systems, this is done by connecting each PE in a layer to other PEs at the same layer. As the result, every PE of a layer has a criterion about the total output of the layer and can adjust its output corresponding to that. Hence the level of the operation of every layer approximately remains fixed. The second kind is called competitive operation. In this case, the operation of a layer is consequence of interactions of PE operations in that layer. Unlike the first case, now only few PEs are able to create the layer output. As an example, we may mention a situation in which only the PE with the highest level of activity will specify the output of the layer.

Speech and signal processing, pattern recognition, system modeling, and servo mechanism control are among the applications of neural networks. In the specif case of time-series modeling, neural networks with the back propagation training algorithm perform more successfully than linear or polynomial prediction algorithms.

B. Back Propagation Algorithm

Generally, a neural network must learn how to classify input patterns. It has been experimentally observed that as the number of layers of a network increases, it can classify more and more complicated patterns. A significant problem is how a network can determine the error between its output and the desired output The network then overcomes the mismatch between desired and actual outputs by adjusting the weightings of interconnections. This is called Credit Assignment (CA). The back propagation algorithm (BPA), originally introduced by Minsky and Papert [9], solves the CA problem by using all of the PEs and adjusting their total interconnections. This is done by propagating the output layer error to the preceding layer via the existing connections. In fact, this operation is repeated until reaching the input layer. In other words, output error moves from each layer to the preceding layer -just the opposite direction of the movement of the original information- until reaching the input layer.

The classical form of a back propagation network consists of one input layer, one output layer, and one or two hidden layers. Although there is no limit on the number of hidden layers from a theoretical stand point, very complicated problems can be solved with four hidden layers. In a back propagation network, each layer is fully connected to the next layer. In the learning phase, information may come back through the network in order to update the weighting functions. The network may also be hetero associative or auto associative. We use the following notation for the purpose of explaining the back propagation algorithm.

- $x_j[s]$: The present output state of the j-th neuron from the layer s
- $w_{ji}[s]$: Weighting function of the connection between the i-th neuron from layer $(s-1)$ and the j-th neuron from layer s
- $I_j[s]$: The combined input of the j-th neuron of layer s

Hence a PE in a back propagation network transfers its output as

$$x_j[s] = f\{\sum_i (w_{ji}[s].x_i[s-1])\} = f\{I_j[s]\} \qquad (6)$$

where f may be every continuous function. The most popular functions are sigmoid, hyperbolic tangent, and sine function. This is because of their interesting properties as nonlinear functions in vanishing the output error of the network. The sigmoid function is defined as

$$f(z) = \frac{1}{1+e^{-z}} \qquad (7)$$

Suppose that the network has an absolute error function which is differentiable with respect to all of the weighting functions. Then the critical parameter which is back propagated to the network is

$$e_j[s] = -\frac{\partial E}{\partial I_j[s]} \qquad (8)$$

where E is the absolute error function. It will be shown that this value may be obtained as a criterion of the relative error of the j-th PE in layer s. By using the chain rule twice, one may introduce the relationship between the relative error of a specified PE in layer s and the local errors in layer $(s+1)$ as

$$e_j[s] = f\prime\{I_j[s]\} \,.\, \sum_k \{e_k[s+1].w_{kj}[s+1]\} \qquad (9)$$

Note that in relation 9 there must be a layer above the layer s, and hence it is not possible to use this relation for the output layer. If the function f is the sigmoid function defined in 7 then it is possible to express its derivative in terms of itself.

$$f\prime\{I_j[s]\} = f(I_j[s]).\{1 - f(I_j[s])\} \qquad (10)$$

and by using relation 10, one may rewrite relation 9 as follows:

$$e_j[s] = x_j[s].(1-x_j[s]) \, . \, \sum_k \{e_k[s+1].w_{kj}[s+1]\} \quad (11)$$

Note that this relation has been obtained for sigmoid function. Hence the back propagation network is described by the following statement.

First information propagates from the input layer to the output layer. Then the error between the desired output and the network output propagates from the output layer to the input layer in the return path. This procedure materializes itself according to the general relation 9 or in the case of sigmoid function as relation 11.

The objective of the learning procedure is to minimize the absolute error function. In the following, we study the procedure based on the concept of local error. Assume that the weighting functions of the network are given in the form of $w_{ji}[s]$. In order to decrease the absolute error function, one may change the weighting functions in the opposite direction of the gradient vector

$$\Delta w_{ji}[s] = -lc \frac{\partial E}{\partial w_{ji}[s]} \quad (12)$$

where lc denotes the learning coefficient, or it can be said that each of the weighting functions vary according to the magnitude and opposite direction of the gradient vector on the error surface.

Partial derivatives of relation 12 may directly be computed based on the relative error relation. Based on relation 6 and chain rule we have

$$\frac{\partial E}{\partial w_{ji}[s]} = \frac{\partial E}{\partial I_j[s]} . \frac{\partial I_j[s]}{\partial w_{ji}[s]} = - e_j[s].x_i[s-1] \quad (13)$$

Combining relations 12 and 13 will lead to

$$\Delta w_{ji}[s] = lc.e_j[s].x_i[s-1] \quad (14)$$

In order to apply the above relations to back propagation algorithm, it is necessary to define the absolute error function precisely. Assume that o is the present output of the network to the input i, and d is the corresponding desired output. Now we may define the absolute error function as

$$E = \frac{1}{2} \sum_k (d_k - o_k)^2 \quad (15)$$

The index k denotes the various elements of d and o. Here, the raw error function is $(d_k - o_k)$ and from relation 12 it is obvious that the scaled relative error in every PE of output layer is obtained by

$$\begin{aligned} e_k(o) &= -\frac{\partial E}{\partial I_k}(o) = -(\frac{\partial E}{\partial d_k}).(\frac{\partial d_k}{\partial I_k}) \\ &= (d_k - o_k).f\prime(I_k) \end{aligned} \quad (16)$$

In neural networks, the scaled local error which propagates through the network is stored in the error field of each PE. As relation 15 shows, the absolute error function is defined on the pair (i, d). In this case with every new pair (i, d) the BPA adjusts the weighting functions to decrease the absolute error function. Now the standard BPA may be introduced as follows:

- Propagate the input i in the forward direction through the network until reaching to the output o. During propagating this information through the network all of the combined inputs I_j and output states x_j for each PE are set.
- For each PE in the output layer calculate the scaled local error by relation 16, then obtain the variations of weighting functions by relation 14.
- For each PE in layer s located below the output layer and above the input layer obtain the scaled relative error and the variation in weighting functions from relations 11 and 14 respectively.
- Update all of the weighting functions by adding the variations to the old values.

In order to increase the performance of the back propagation algorithm, we have to change the standard algorithm slightly as discussed below.

1) Inserting Momentum Term: Remember that the gradient method updates the weighting functions as linear functions of partial derivatives according to relation 12. The problem is that, this method fails for the large values of the learning coefficient lc. On the contrary, having a small learning coefficient introduces a decrease in the speed of the learning procedure. In order to overcome this problem, we use the concept of momentum term. Inserting this term changes the equation 14 as follows

$$\underbrace{\Delta w_{ji}[s]}_{(k+1)-\text{th step}} = lc.e_j[s].x_i[s-1] + M(\underbrace{\Delta w_{ji}[s]}_{k-\text{th step}}) \quad (17)$$

where M stands for the momentum. This relation is, indeed, a low pass filter omitting the oscillatory behavior and hence providing a reasonable learning speed with a small learning coefficient.

2) Derivatives Correction: Fahlman [10] introduced several methods to increase the convergence speed of the standard BPA. These methods work based on the derivatives correction and nonlinear error functions. One of the simple techniques that he used, was to add a small positive offset to the sigmoid function's derivative before scaling the error. To explain the performance of the method, it

may be said that when the weighting functions of PEs increase, the combined input increases and as a result the operating level go to the saturation. Hence the derivative and the scaled relative error vanish. In this case adding a small offset overcomes this problem.

3) Fast Back Propagation: In this method relation 14 is replaced by the relation

$$\Delta w_{ji}[s] = lc.e_j[s].(x_i[s-1] + e_i[s-1]) \qquad (18)$$

In simple words the error of layer $(s-1)$ is added to the operating level before updating the corresponding weighting functions. It is also possible to insert a coefficient in this new relation.

$$\Delta w_{ji}[s] = lc.e_j[s].(x_i[s-1] + k.e_i[s-1]) \qquad (19)$$

The value of the parameter k allows us to be more flexible in the implementation and get better results. Note that for $k = 0$ standard BPA is obtained and for $k = 1$ relation 18 is obtained.

We close this section by pointing out that other absolute error functions proportional to the higher powers of Euclidean distance between the desired output and the network output can be used instead of the one defined in in relation 15.

IV. Discrete-Time Arrays Modeling with Neural Networks

In the past and present literature work, there are many examples of control and modeling of nonlinear systems exhibiting chaotic behavior by means of using other well-recognized nonlinear systems and control tools such as fuzzy systems, neural networks, and H^∞ control, [1], [17], [18], [21], [22], [23] as well as modeling nonlinear systems relying on the theory of chaos [19], [20].

In this section, we introduce the modeling of discrete-time arrays with neural networks. We focus on two kinds of the arrays namely one- and two-dimensional discrete-time arrays.

A. One-Dimensional Discrete-Time Maps

The first group consists of a broad class of one-dimensional discrete-time maps whose next state is a third order polynomial of the present state. The members of this group are described by the relation

$$x_{n+1} = a + bx_n + cx_n^2 + dx_n^3 \qquad (20)$$

It is obvious that by choosing the parameters a, b, c, and d we can generate a large number of famous one-dimensional discrete maps, e.g, with $a = 0$, $b = 4$, $c = -4$, and $d = 0$ we reach the logistic map with the equation

$$x_{n+1} = 4.x_n.(1 - x_n) \qquad (21)$$

or with $a = V$, $b = 0$, $c = -\frac{1}{2}$, and $d = 0$ relation 20 becomes identical to the parabolic map equation in the form of

$$x_{n+1} = V - \frac{1}{2}\, x_n^2 \qquad (22)$$

Now we proceed to explain the modeling procedure. We have used a fixed structure network with two hidden layers for our modeling task. This network has four neurons in its input layer, ten neurons in each of its two hidden layers, and one neuron in its output layer. The nonlinear transfer function of each of the neurons is the sigmoid function. The interesting property of this function is that its derivative can be described in terms of itself as

$$f\prime(z) = f(z).[1 - f(z)] \qquad (23)$$

where $f(z) = [1+e^{-z}]^{-1}$, this is very useful in the learning procedure of the network and may reduce the cost of computation significantly. Figure 3 shows the structure of the network. As it can be seen in figure 3, this network is a fully connected feedforward perceptron network In our network, the last part of the learning phase is connected to the first part of the recalling phase. The learning phase is, indeed, the adjusting procedure of the weighting functions in response to an input in order to have the suitable output in the output layer. As the network accesses desired output samples in its learning phase, the learning procedure is supervised learning. In addition, because the desired output is different from all of the inputs in each step, the trained network is hetero-associative. Four input samples are four consecutive samples of the discrete-time mapping, and the output that the network must be able to predict is the fifth sample of the mapping. The required number of samples for training of the network based on the steady state behavior of the mapping differs from thirty thousands for simpler mappings to the one hundred thousands for the more complicated mappings, i.e, mappings with chaotic behavior.

Each example consists of four consecutive samples (x_1, x_2, x_3, x_4) as the input and the fifth one, (x_5), as the output. In the next example, the first sample is discarded and the second sample through the fifth sample (x_2, x_3, x_4, x_5) are used as the input samples while the sixth sample (x_6) is used as the output sample and so on and so forth. The network operation is based on the comparison of the generated output and the desired output. Based on this comparison, an error signal is created and by back propagating this signal through the network

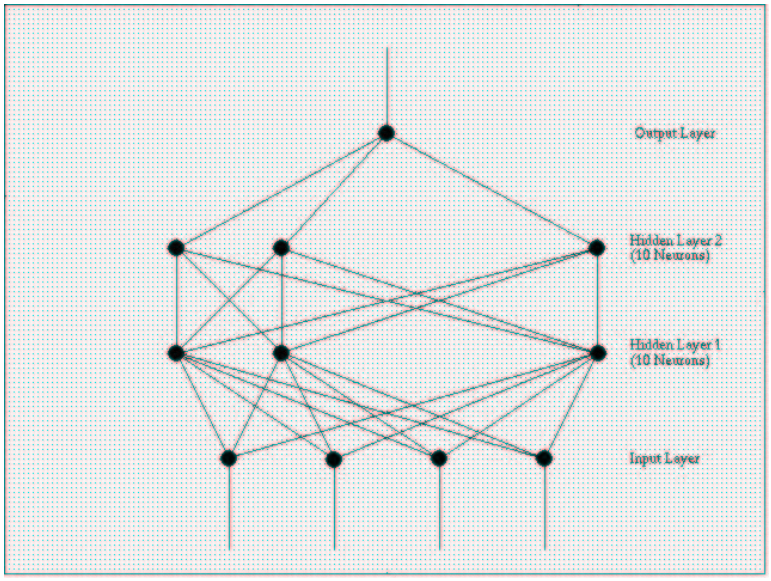

Fig. 3. Fixed structure neural network used for modeling one-dimensional discrete-time arrays

weighting functions are adjusted. In the recalling procedure, the error is approximately zero, i.e, an upper error bound of 10^{-4} for the error is used . The network is a feedforward network with no feedback connection. As the learning procedure is based on the correction of the error, the network is able to have a suitable output when the input is deficient or noisy.
The absolute error function is built on the basis of the second momentum output error

$$E = \frac{1}{2}(y_d - y)^2 \tag{24}$$

By using relation 16 and substituting $f\prime(I_k)$ by $y(1-y)$ from the sigmoid function, the relative error e_o is

$$\begin{aligned} e_o &= -\frac{\partial E}{\partial I_o} = -\frac{\partial E}{\partial y}\frac{\partial y}{\partial I_o} \\ &= (y_d - y).f\prime(I_o) \\ &= (y_d - y).y.(1-y) \end{aligned} \tag{25}$$

where y is the network output, y_d is the desired output, I_o is the compound input of the output neuron, and f is the sigmoid function. It is interesting to know that multiplying the error by the derivative of the transfer function, scales it. In the learning procedure the gradient method is considered. For implementing back propagation algorithm, we use the algorithm discussed in the previous section and for having a better performance we use the improved version of relation 14 as in relation 19. Note again that the objective of the learning phase is to minimize the absolute error function.

B. Two-Dimensional Discrete-Time Maps

The second group consists of a broad class of two-dimensional discrete-time polynomials

$$\begin{aligned} x_{n+1} &= f_{11}(x_n) + f_{12}(y_n) \\ y_{n+1} &= f_{21}(x_n) + f_{22}(y_n) \end{aligned} \tag{26}$$

where the f_{ij} functions are second order polynomials of the corresponding variables, i.e, $f_{ij}(x) = a_{ij} + b_{ij}x + c_{ij}x^2$. Again it shows that the generality of this class is more than special maps which show chaotic behavior from themselves such as Henon map. In fact, by choosing parameters a_{ij}, b_{ij}, and c_{ij} as follow, we have Henon map

$$\begin{aligned} a_{11} &= \frac{1}{2}, \quad b_{11} = 0, \quad c_{11} = -A \\ a_{12} &= \frac{1}{2}, \quad b_{12} = 1, \quad c_{12} = 0 \\ a_{21} &= 0, \quad b_{21} = B, \quad c_{21} = 0 \\ a_{22} &= 0, \quad b_{22} = 0, \quad c_{22} = 0 \end{aligned}$$

in the form of

$$\begin{aligned} x_{n+1} &= 1 + y_n - A\,x_n^2 \\ y_{n+1} &= B\,x_n \end{aligned}$$

Generally, with the appropriate choice of parameters, it is possible to have various kinds of steady state behaviors. We have used again a fixed structure neural network same as before except that present neural network has two neurons in its output layer corresponding to the x_{n+1} and y_{n+1} outputs. Four inputs are two consecutive samples of signals x_n and y_n. Figure 4 shows the structure of the network. The mechanism is completely the same as before.

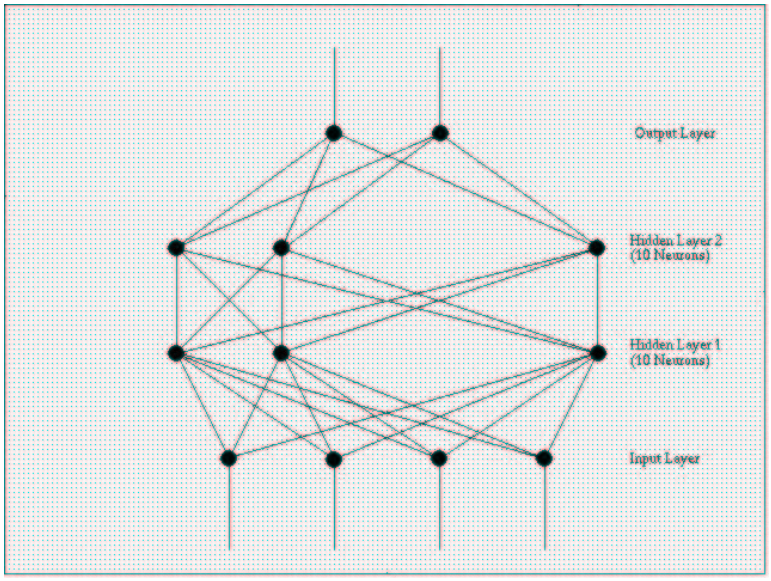

Fig. 4. Fixed structure neural network used for modeling two-dimensional discrete-time arrays

The only difference here is that the absolute error function is now defined as

$$E = \frac{1}{2}\left[(y_d - y)^2 + (x_d - x)^2\right] \tag{27}$$

where x_d, y_d are desire outputs, x, y are network outputs, and the relative output errors are

$$\begin{aligned} e_{ox} &= (x_d - x).x.(1 - x) \\ e_{oy} &= (y_d - y).y.(1 - y) \end{aligned} \tag{28}$$

All of the other steps are the same as the steps implemented in case of the first group.

V. Experimental Results

In this section, first the simulation results of the modeling task are discussed, then some of the practical problems in the implementation of the algorithm are reviewed. For all of the graphs appearing in this section, the initial weighting functions of 0.1 are chosen between every single pair of PEs connected together. Only for the graph appearing in figure 8 a combination of values 0.1, 0.3, and 0.7 was chosen in order to make a significant initial difference.

First, the performance of the modeling tool is evaluated in case of logistic map. As it is observed in figure 5, for the initial condition $x_0 = 0.75$ which yields to a stable fixed point, the network is able to follow the behavior of the map very rapidly. Figure 6 shows the result for the initial condition $x_0 = 0.1$ as an example of modeling chaotic solutions. The fixed structure neural network follows the chaotic map with an absolute error less than 0.001 continuously but it takes almost 94'000 iterations for it to be able to predict the next 40 samples of the map within the specified error bound. After that, the neural network loses the track because it becomes chaotic itself. Since it is not possible to have a zero value for the error, the error is built up as time proceeds and finally exceeds the acceptable error bound. At this stage the learning process needs to be repeated in order to force the network to track the chaotic array within the acceptable error limits. Qualitatively, the same pattern is repeated in case of $x_0 = 0.3$ and a number of other initial conditions. Finally, it is very important to note that the choice of initial conditions both in case of chaotic map and the weighting functions of the neural network greatly affects the speed of convergence.

Figure 7 shows the result of the modeling task for the Henon map with the parameters $(A, B) = (4, 1)$ and the initial condition $(x_0, y_0) = (0.5, 0.5)$. It shows that the network is able to reach to the fixed point very rapidly. Figure 8 shows the result of the modeling task for the Henon map with the parameters $(A, B) = (0.8910, 0.0017)$ and the initial condition $(x_0, y_0) = (0.67, 0.21)$. This is an example of modeling periodic solution and again, the figure shows that the network is able to learn the periodic behavior of the map very rapidly.

It is important to remind that although the dynamical behavior of the map is very much more complicated compare to logistic map, it is relatively difficult in practice to choose the parameters such that the map exhibits chaotic behavior. This is mainly because of the round off errors that shift the map out of the narrow chaotic regions.

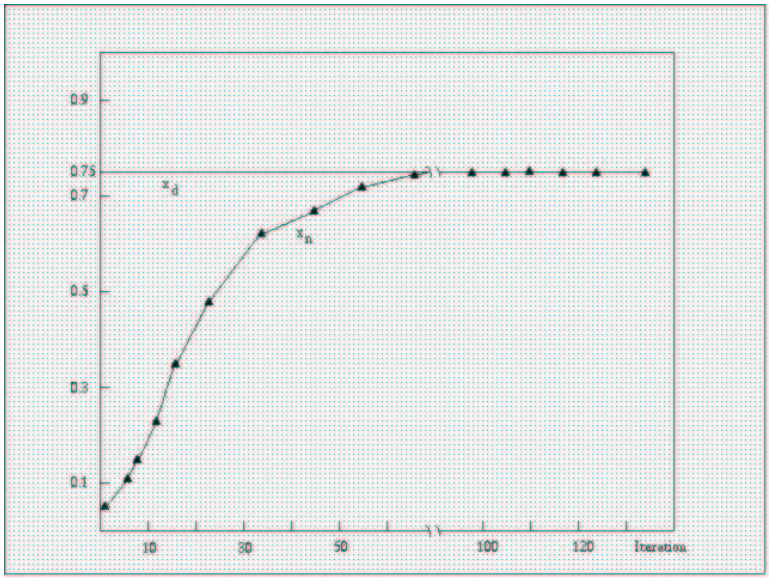

Fig. 5. Logistic map modeling result for initial condition =0.75

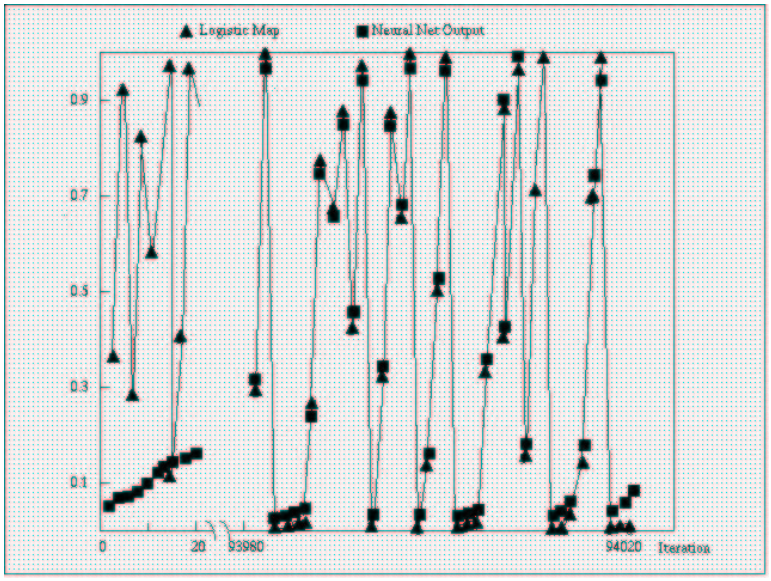

Fig. 6. Logistic map modeling result for initial condition =0.1

A. Numerical Issues

In the following, a number of numerical issues encountered in the implementation phase are mentioned.

1) Time Consuming Learning Algorithm: The learning algorithm is time consuming and needs to be periodically repeated in case of chaotic maps. To explain the issue, it can be said that although it is possible to reach a very small network error at some steps during the learning phase, the error never reaches the absolute value of zero. If the network error is studied for further samples, it is observed that this error begins to grow as time proceeds. The phenomenon is related to the chaotic nature of the system, i.e, since the nonlinear network models a chaotic array, it becomes chaotic itself. In this situation, the small error may be considered as a small difference between two close initial conditions for the desired output and the network output. As a characteristic of a chaotic system, the error will begin to grow soon and this is nothing but high sensitivity to the variations of initial conditions. As a matter of fact, this is interpreted as a sign for a network trying to model a chaotic map and becoming chaotic itself. One way of relieving the effect of having an error component which grows with the time is to repeat the learning phase followed by the recalling phase periodically, otherwise the results are not reliable any more. In case of the maps studied in our work, the neural network would typically be able to predict less than one hundred samples of the chaotic map after the first learning phase with a couple

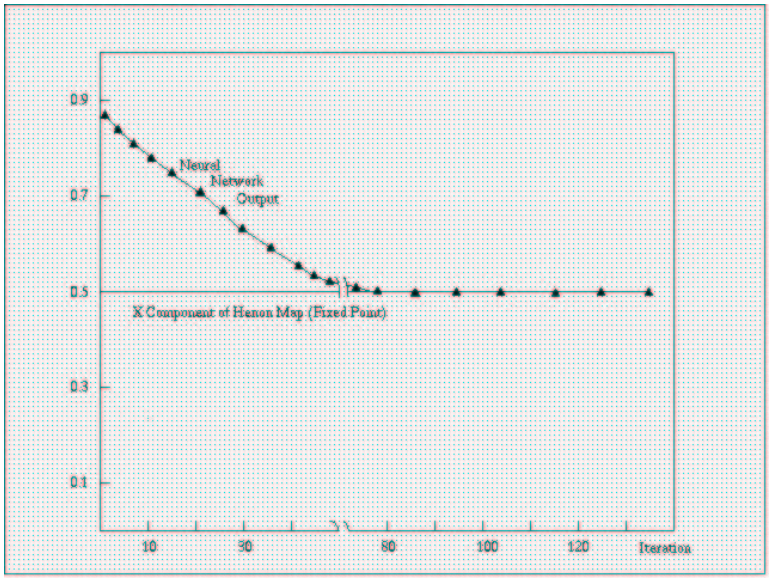

Fig. 7. Henon map x state component modeling result for initial condition $(x_0, y_0) = (0.5, 0.5)$ and parameters $(A, B) = (4, 1)$

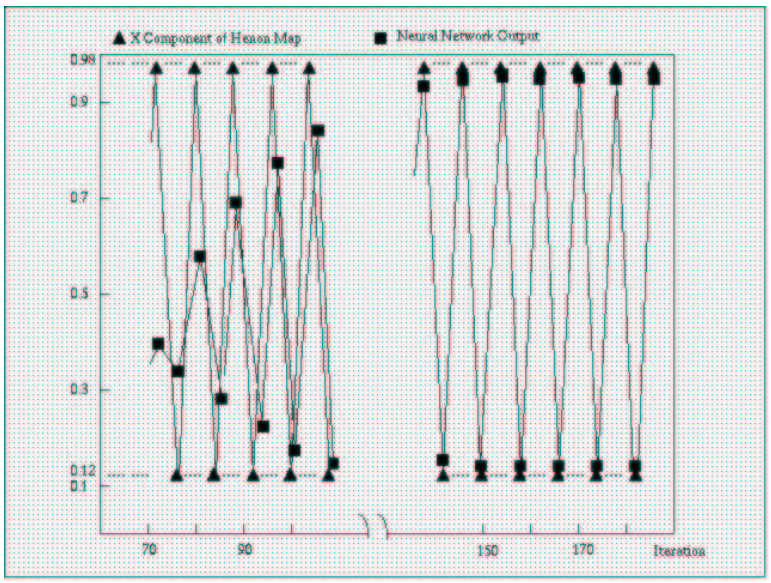

Fig. 8. Henon map x state component modeling result for initial condition $(x_0, y_0) = (0.67, 0.21)$ and parameters $(A, B) = (0.8910, 0.0017)$

of hundreds of thousands of examples and then the learning phase has to be repeated with a comparable number of examples as in the previous phase and so on to have reliable results.

2) Effects of Learning Coefficient and Momentum Term: It was mentioned that the variations of weighting functions are derived by the relation

$$\Delta w_{ji[s]} = lc.e_j[s].x_i[s-1]$$

The network learning speed which is, in fact, the speed of weighting functions adjustment procedure is a function of the value of learning coefficient lc. Although bigger values of the learning coefficient may lead to have faster learning procedure, it is not possible to have very big learning coefficients. This is the direct effect of applying the gradient method, i.e, weighting functions are changed as a linear function of the error surface of the partial derivatives in the gradient method. The approximation is valid as long as the error surface is relatively linear while this assumption is not valid for large values of lc. On the contrary, having small values for lc leads to have a low learning speed. In order to overcome this problem, we have used the improved version of the above relation as in 18. The suitable value of lc is different from case to case but in order not have the effects of nonlinear error surface, the lc must be less than one in most of the cases. Moreover, the second momentum of Δw_{ji} acts more suitably in most of the cases.

3) Low Convergence Speed Near the Optimum Point: The main problem of the gradient method is due to its low convergence speed near the optimum point, i.e, although for a big initial difference moving toward the direction of the gradient vector decreases the difference rapidly, the performance of the method is significantly degraded near the optimum point. In order to omit this problem, a multiple of layer $(s-1)$ error can be added to the weighting functions adjustment algorithm as in relation 29.

$$\begin{aligned} \underbrace{\Delta w_{ji}[s]}_{(k+1)-\text{th step}} &= lc.e_j[s].\{x_i[s-1] + k.e_i[s-1]\} \\ &+ M(\underbrace{\Delta w_{ji}[s]}_{k-\text{th step}}) \end{aligned} \tag{29}$$

where M stands for the momentum. In our experiments, the weighting functions adjustment algorithm has shown the most sensitivity to the variations of the k coefficient. The suitable value of this parameter is again different from case to case but in the most cases a value in the interval $[0, 1]$ has led to a suitable result.

4) Effects of Initial Condition: All of the convergence results are affected strongly by the choice of initial conditions. The choice of initial values of the parameters play a crucial role in the convergence of the algorithm. The algorithm quickly diverges, if the initial values of the parameters are chosen unsuitably. As a practical result, it is better to set the initial values of the parameters as $w_{ji}(0) = 0.01 \quad \forall i, j$.

5) Floating Point Overflow Problems: As the last important factor, the overflow problem is discussed. In the implementation of any numerical computation, one must always consider the probability of having very large or very small numerical values. Storing such kinds of values may lead to produce invalid results because of round off operations. In our special case, significant variations of compound inputs of PEs can create many problems in their storing procedure. Based on the asymptotic behavior of the transfer function which is the sigmoid function in this case, the problem may be avoided. Observe that the sigmoid function reaches its asymptotic value with a very good approximation when the absolute value of its argument is less than or equal 11, i.e

$$\frac{1}{1+e^{-z}} \simeq \begin{cases} 0 & : \quad z \leq -11 \\ 1 & : \quad z \geq 11 \end{cases} \tag{30}$$

In other words, there exist some limits for the compound input and for all of the values beyond these limits the transfer function almost acts the same. Hence, an upper limit of $z = 11$ and a lower limit of $z = -11$ are considered for the compound input to avoid having the overflow problem in the numerical algorithm.

VI. Conclusion

This paper concerned itself with the modeling of such transformations as logistic and Henon maps with a multipurpose fixed structure system, i.e, a neural network relying on back propagation learning algorithm.

We used a fixed structure neural network with two hidden layers in order to model one- and two-dimensional discrete maps. This network had four neurons in its input layer and ten neurons in each of its two hidden layers. The number of neurons in its output layer was one in the case of one-dimensional and two in the case of two-dimensional mappings. In the former case, we applied four consecutive samples of the one-dimensional array as the input of the network and gave the fifth sample as the desired output to the network in each iteration of the learning algorithm. In the former case, we applied two consecutive samples of each of the state elements and gave the third element of state as the desired output to the network. We observed that the number of samples required for the training of the neural network depended on the steady state behavior of the mapping, i.e, the number of required samples increased in the case of more complicated steady state behaviors.

There were several problems in implementing the learning algorithm such as the large number of required examples, the effects of having big learning coefficient and inserting momentum term in order to overcome the corresponding drawbacks, the low convergence speed of the algorithm near the optimum point, the effects of initial conditions in the convergence of the algorithm, and the floating point issues as the result of having to store very large or very small compound inputs.

After inserting a slight improvement in the standard learning algorithm and after completing the learning phase, the network was able to predict the sequential samples of the corresponding discrete array with a reasonable error, i.e, the error was less than 10^{-4}. The number of reliable predicted samples were limited in the case of chaotic arrays and the learning phase needed to be alternatively repeated. This algorithm may be applied for modeling of other classes of discrete-time transformations such as piecewise-linear and higher dimensional transformations. In these cases, the algorithm can be adjusted suitably for the corresponding purpose.

REFERENCES

[1] H. Yousefi'zadeh, M. Shafiee, A. Ziloochian, "Chaotic Arrays Modeling with Neural Networks", In Proceedings of the First International Iranian Conference in Electrical Eng., Vol.4, PP. 667-679, May 1993.

[2] E. Ott, "Chaos in Dynamical Systems", Cambridge University Press, 1997.

[3] J. Guckenheimer, P. Holmes, "Nonlinear Oscillations, Dynamical Systems, and Bifurcation of Vector Fields", Springer-Verlag Co., 1985.

[4] S. N. Rasband, "Chaotic Dynamics of Nonlinear Systems", J. Wiley & Sons Co. 1989.

[5] P. R. Halmos, "Measure Theory", D.Van Nostrand Co., 1950.

[6] R. Mane, "Ergodic Theory and Differential Dynamics", Springer Verlag Co. 1987.

[7] T. S. Parker, L. O. Chua, "Practical Numerical Algorithms for Chaotic Systems", NY, Springer-Verlag, 1989.

[8] M. M. Sushchik, N. F. Rulkov, H. D. I. Abarbanel, "Robustness and Stability of Synchronized Chaos: An Illustrative Model", Circuit & Systems I: Fundamental Theory and Applications, IEEE Transactions on , Volume: 44 Issue: 10 , Oct. 1997.

[9] M. Minsky, S. A. Papert, "Perceptrons: An Introduction to Computational Geometry.", MIT Press, Cambridge, MA, Expanded Edition, 1988/1969.

[10] S. E. Fahlman, "An Empirical Study of Learning Speed in Back-Propagation Networks", Technical Report CMU-CS-88-162, Carnegie Mellon University, June 1988.

[11] A. Van Ooyen, B. Neihuis, "Improving the Convergence of Back Propagation Algorithm", Neural Networks, Vol.5, No.3, 1992.

[12] H. Guo, S. Gelfland, "Analysis of Gradient Descent Learning Algorithms for Multilayer Feed Forward Neural Networks", IEEE Trans. on Circuit & Syst., Vol. 38, No.8, Aug. 1991.

[13] H. Yang, T. Dillon, "Convergence of Self-Organizing Neural Algorithms", Neural Networks, Vol.5, No.3, 1992.

[14] G. Mirchandani, W. Cao, "On Hidden Nodes for Neural Nets", IEEE Trans. on Circuit & Syst., Vol.36, No.5, 1989.

[15] N. J. Dimpoulos, "A Study of Asymptotic Behavior of Neural Networks", IEEE Trans. on Circuit & Syst., Vol. 36, No.5, 1989.

[16] V.Kurkova, "Kolmogrov's Theorem and Multilayer Neural Nets", Neural Networks, Vol.5, No.3, 1992.

[17] K. Tanaka, H. O. Wang, "Fuzzy Control of Chaotic Systems Using LMIs: Regulation, Synchronization and Chaos Model Following", Fuzzy Systems Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on , Volume: 1 , 1998.

[18] K. Tanaka, T. Ikeda, H. O. Wang, "Controlling Chaos via Model-Based Fuzzy Control System Design", Decision and Control, 1997., Proceedings of the 36th IEEE Conference on , Volume: 2 , 1997.

[19] L. Zupeng, D. Yuguo; Y. Peiyang, "A Sensitive Cell Neuron Model Based on Chaos", Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on , Volume: 1 , 2000.

[20] S. Haykin, H. Leung, "Neural Network Modeling of Radar Backscatter from an Ocean Surface Using Chaos Theory", Neural Networks for Ocean Engineering, 1991., IEEE Conference on , 1991.

[21] S. Bohacek and E. A. Jonckheere, "Linear Dynamically Varying LQ Control of Nonlinear Systems over Compact Sets", IEEE Transaction on Automatic Control, volume 46, pp. 840-852, June 2001.

[22] S. Bohacek, E. A. Jonckheere, "Nonlinear Tracking over Compact Sets with Linear Dynamically Varying H^∞ Control", SIAM J. Control and Optimization, vol. 40, No. 4, pp. 1042-1071, 2001.

[23] S. Bohacek, E. A. Jonckheere, "Linear Dynamically Varying H^∞ Control of Chaos", NOLCOS98, Nonlinear Control Systems Design Symposium, International Federation of Automatic Control (IFAC), University of Twente, Enschede, The Netherlands, July 1998.

PLACE PHOTO HERE

Homayoun Yousefi'zadeh was born in Tehran, Iran. He received B.S., M.S., and Ph.D. degrees all in Electrical Engineering from Sharif University of Technology, Tehran Polytechnic Institute, and University of Southern California in 1989, 1993, and 1997 respectively. Dr. Homayoun Yousefi'zadeh research and industry experience spans over teletraffic modeling and analysis, network traffic control, real-time media systems, distributed database systems, storage networking, and enterprise client-server applications. He is currently with the Center for Pervasive Communications in Electrical and Computer Engineering Department of University of California, Irvine. He has been involved with a number of academic and industry initiatives in different capacities including chairperson of systems' management workgroup of Storage Networking Industry Association (SNIA), member of Scientific Advisory Board (SAB) of Integrated Media Services Center (IMSC) at the University of Southern of California, referee for IEEE Communication Letters, main panelist in domestic and international storage networking events such as IDC's European Storage Symposium and Search Storage Online, member of American Management Association (AMA), and member of American Society for Engineering Education (ASEE). His research interests include intelligent modeling of nonlinear dynamics, teletraffic analysis and control, heterogeneous real-time media systems, high-speed broadband networks, and distributed algorithms.

PLACE PHOTO HERE

Edmond A. Jonckheere was born in Belgium in 1950. He received the M.S. degree in electrical engineering from the University of Louvain, Belgium, the Dr. Eng. degree in aerospace engineering from the University of Toulouse, France, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 1973, 1975, and 1978, respectively. From 1973 to 1975, he held a Research Fellowship of the European Space Agency at the Laboratory for Systems Architecture and Analysis (LAAS), Toulouse, France. From 1975 to 1978, he was a Teaching/Research Assistant and subsequently a Research Associate in the Department of Electrical Engineering at the University of Southern California, Los Angeles. In 1979, he was with the Philips Research Laboratory, Brussels, Belgium. In 1980, he returned to the University of Southern California, where he is currently a Professor of Electrical Engineering and a member of the Center for Applied Mathematical Sciences (CAMS). He has also had short-term academic appointments with the Australian National University and the University of Namur, Belgium. His consulting and other activities include the Max Planck Institute, Germany, Memorial Medical Center of Long Beach, CA, Honeywell, Minneapolis, MN, The Aerospace Corporation, El Segundo, CA, Lockheed-Martin, Palmdale, CA, and American GNC Corporation, Simi Valley, CA. His current research interests include modeling complicated nonlinear dynamics, network flow control, and network hyperbolic geometry.