Computational Economics 7: 37–53, 1994
© 1994 *Kluwer Academic Publishers. Printed in the Netherlands.*

Identification Environment and Robust Forecasting for Nonlinear Time Series

BERLIN WU
Department of Mathematical Sciences, National Chengchi University, Taiwan

(Received: June 1992)

Abstract. In this paper, the methods of time series for nonlinearity are briefly surveyed, with particular attention paid to a new test design based on a neural network specification. The proposed integrated expert system contains two main components: an identification environment and a robust forecasting design. The identification environment can be viewed as a integrated dynamic design in which cognitive capabilities arise as a direct consequence of their self-organizational properties. The integrated framework used for discussing the similarities and differences in the nonlinear time series behavior is presented. Moreover, its performance in prediction proves to be superior than the former work. For the investigation of robust forecasting, we perform a simulation study to demonstrate the applicability and the forecasting performance.

Key words. nonlinear time series, bilinear, Lagrange multiplier test, neural network, forecasting, robust.

1. Introduction

The analysis in time series models has been concerned with processes which are stationary. Tests for unit roots in time series data have been the subject of attention in econometrics as well as with statisticians in the last two decades. Much of the research has concentrated on the distribution theory that is necessary to develop these tests and the analysis of the power of various tests under different alternative hypotheses. However, in a majority of economics applications the need of a nonlinear test is a priori rather than unit roots test for nonstationarity. As the papers by Tsay (1991), Granger (1991), and Gooijer and Kumar (1992) indicate, the interest in applying nonlinear time series models has considerably increased recently. There also seems to be strong belief among economists that relationships between economic variables are nonlinear; production model being an example. Specifically, given parametric relationship, such as the Cobb-Douglas or CES production models, standard econometrics techniques provide ways of estimation of the parameters as well as its asymptotic properties; see Granger (1991).

The results obtained thus far did not provide efficient rules in testing whether or not the correct nonlinear specification has been achieved or whether there still remains some neglected nonlinearity in the estimated relationship. In other words, when the underlying models are not correctly specified, very few statistical tests can provide a well-designed procedure for model identification and parameters estimation. Since uncertainties usually exist about the correct underlying statistical models, the dynamic form, lag structure and the stochastic assumptions have a heavy impact on the power of the tests and the forecasting performance. Therefore, to propose a *robust procedure* for identifying nonlinear time series, as well as accurate forecasting, is what we are most concerned about.

The problems of *identifying* nonlinear time series, that we propose, can be divided into two groups: model based and model free systems. Up to now, most of the former work has relied on diagnostics such as *Lagrange Multiplier test, Bispectrum test, Likelihood ratio-based tests* and *arranged autoregression test*. . . ; *etc*. Expository papers in *LM* testing are Luukkonen et al. (1988) for testing SETAR/TAR type nonlinearity and Saikonen and Luukkonen (1988) and Guegan and Pham (1992) for testing simple bilinear type nonlinearity. The *LM* test against the ARCH model can be found in Weiss (1986) and Saikonen and Luukkonen (1991). Hinich (1982) used the *Bispectrum test* to identify the bilinear models. Chan and Tong (1986) used *Likelihood ratio-based tests* for SETAR models. Tsay (1989), (1991) presented the *arranged autoregression method* for testing TAR models. Unfortunately, these available tests are based on a particular class of nonlinear time series models. None of the above tests has dominated the others with reasonable power. For instance, the *LM* test is designed to reveal specific types of nonlinearity. The test may also have some power against *incorrect* alternatives. There may, at the same time, exist alternative nonlinear models against which an *LM* test is not powerful. Hence, rejecting $\mathcal{H}_0$ on the basis of this test does not allow very strong conclusions about the type of possible nonlinearity.

On the other hand, an interesting research topic for nonlinear systems has arised in the study of *neural networks*. This is, recently, an area of considerable applications due to its potential for providing insights into the kind of highly parallel computation that is carried out by physiological nervous systems. Recently, many computer-based analytical models have been developed and the neural networks technologies have been extensively studied by many researchers, e.g., Grossberg (1988) and his colleagues provided much of the theoretical foundation for these systems; Lapedes and Farber (1988) at Los Alamos National Laboratory have used *backpropagation* networks for making prediction and system modeling; Ramey (1989) and Kosko (1991) suggest various applications, including fuzzy theory, for neural networks. Detailed expositions of the approximation and network learning theory can be found in the book of White (1992), which also provides a fundamental link between network learning and modern mathematical statistics. Recently, Chatfield (1993) suggests that it is possible that neural nets will outperform standard forecasting procedures when a fair comparison is made, at least for certain types of situations.

The use for neural networks in forecasting is a new approach for the time series analysis and is developed in Section 4. Methodologically, the outlined strategy requires the analysis of the following procedures: (a) constructing the adequate neural system; (b) data pre-processing procedure; (c) network initiation; (d) learning and training; (e) forecasting.

The goal of this paper is to propose a network system which integrates nonlinear testing and neural network identifier techniques and performs a robust forecasting. Neural networks, with its testing processes, can be viewed as an integrated dynamic design in which cognitive capabilities arise as a direct consequence of their self-organizational properties. The robust properties for forecasting performance are discussed by comparing them with other model-based procedures.

2. Integrated Identification Environments

Since a lot of economic time series cannot be linearized, we will consider an alternative technique for model structure and identification. In this section we will give an integrated nonlinear testing environment. Genetic operators allow the system to learn from experience and modify its behavior in reaction to stimuli coming from an external environment which is (possibly) evolving in time.

The principle components of an identifying system are:

- a finite set of rules $\mathcal{D} = \{\mathcal{R}_1, \ldots, \mathcal{R}_1\}$ *(the identifier)*;
- a set of detectors, which provide the system with information about the state of external environment;
- a set of erectors, which send the outputs of the computation to the external environment;
- a message list, where the detectors post their message, and where the erectors pick their activation signals.

An identifier $I_i \in \mathcal{T}$ is a tuple $(I_{i1}, I_{i2}; A_i, S_i)$, where I_{ij} denote the conditions that have to be satisfied for I_i to become active, A_i denote the type of message that will be produced by I_i when it is activated, i.e., A_i is a template that may use some information contained in the message that match the I_{ij}*'s* to produce the new message. s_i is a real number (the strength) which quantifies *how good* the identifier is, i.e., how useful it is, in a sense which will be made clear when we describe the mechanism of the system.

Identification of time series X_{1t} *and* X_{2t} can be based on the threshold values of each data for the category membership. For each data D_i the membership in the category can be the following calculation:

$$If\ sign(output\ D_i) = \begin{cases} 1, \text{then category } I_i \\ 0, \text{then category } I_2 \end{cases} \tag{2.1}$$

Expression (2.1) describes the decision function of the identifier designed by

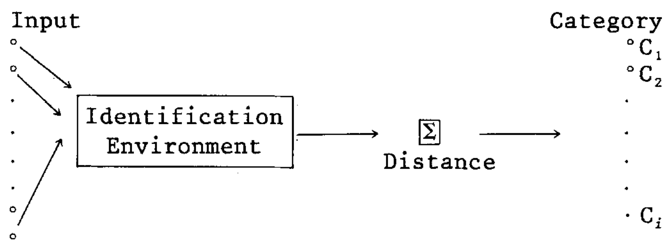


Fig. 1. Integrated identification environment.

inspection of the set that needs to be classified. The dynamic process is shown in Figure 1.

Though the design looks simple, its operation becomes far more involved and intriguing when requirements for membership in categories become complicated.

2.1. IDENTIFYING PROCEDURES

The identifier system performs a sequence of test procedures which include (i) pattern classification of date set (ii) order selection and (iii) parameters estimation. And consist the following steps:

(1) Time series acquisition: ACF, PACF, CCF, cumulant, bispectrum, polyspectrum. etc.

(2) If time series data set passed the linearity and stationarity test: LM test *Bispectrum test*, *Likelihood ratio-based tests* or *arranged autoregression test*, etc. Then go to final step (7); otherwise go to step (3).

(3) Diagnose on correlation dimension, Kolmogorov entropy, and Lyapunov exponents, Lagrange multiplier, etc.

(4) All identifiers with satisfied conditions enter a competition to generate messages for the new message list. The competition consists, in each identifier, of making a bid, and the winning identifiers become *active* and post their messages.

(5) If messages indicate a special nonlinear type, complete this model construction, then go to step (7); otherwise go to step (6).

(6) The effects check the new message list and produce new actions (the output of the identifier system) according the messages they find. It is possible that no action takes place. An extended identification system is considered.

(7) Output the result.

It is clear that the identifier systems are inherently parallel systems. In fact there is an obvious correspondence between the message-passing scheme used by classifiers to communicate and communications between units in a parallel machine. In the above processes, the rules that responded *correctly* to the stimuli of the environment increase their strength, while those that participated in computations which led to wrong answers are weakened. These new identifiers replace the weak ones in an attempt to improve the overall performance of the system. This genetic is essentially random; actually, the selection of the loci for the cross-over and the mutation operations is made following a flat probability distribution, while the prospective parents are usually chosen according to a probability distribution. In practice other genetic operations are introduced to improve the learning performance system.

Another important feature is that one can provide a set of pre-defined rules to help the system in the search for an adequate behavior. In other words, partial information may be provided to shorten the learning stage. This differs from the case of neural networks, where the rules followed by the system are distributed in the connection strengths, and it is hard to give a meaning to the connections in terms of rules *discovered* by the system.

In an identifier system it may happen that different messages generate equal or different messages through the same identifiers in the same time point. In other words, the same identifier can connect different couples of cells and many copies of its are generally brought into play on each major cycle. Cells without connections can be conceived of as being linked to the remaining cells by virtual connections with strength set to zero. Also notice that two distinct nodes n_i, n_j may have several connections, corresponding to different identifiers $\mathcal{T}$ such that $\mathcal{T}(n_j) = n_j$.

3. Neural networks and Model-Free Forecasting

Bilinear time series comprise the simplest class of nonlinear time series. In its most general form a bilinear time series $\{X_t\}$ with discrete parameter is defined by

$$X_t - \sum_{j=1}^{p} a_j X_{t-j} = \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij} X_{t-i} \varepsilon_{t-j}\,, \tag{3.1}$$

where ε_t is a strict white noise process with finite variance $\sigma^2\varepsilon$. This model is called $BL(p, q, m, n)$.

In some practical applications, numerous bilinear models have been studied in physics, seismology, engineering, ecology, etc. These include, e.g., nuclear fission reactors (Mohler, 1963) and biomedical processes (Mohler and Ruberti (1978), and Marchuk (1983)). Heating, ventilating, air conditioning systems, solar panels, and storage tanks, may be similarly modeled by the bilinear state space model. Such analyses result in significant energy saving over the traditional linear optimal control approach. Bilinear models are coupled together so that the total model is more highly nonlinear. Such model coupling may be linear or nonlinear, depending on the particular process under investigation.

Granger and Anderson (1978) pointed out that some simple bilinear series have zero second order autocorrelation and might be mistaken as white noise or an

ARMA model with higher order. Suba Rao and Gabr (1984) have proposed a method about model identification and parameters estimation of $BL(p, O, m, n)$, but it takes a lot of computations and iterations. Wu and Shih (1992) suggest a more efficient process than Subba Rao and Gabr's method for simple bilinear models. However, for general bilinear time series, there is no consistent and efficient method for solving the identification problem so far. Not even for the forecasting of bilinear models.

The neural networks approach, as an area of considerable recent interest due to its potential for providing insights into the kind of highly parallel computation, has found a general approximation for any unknown functions, providing that enough input and output data are given for training the neural networks. The ability to estimate, test, model, forecast and control the underlying system provides great incentive.

Since neural networks are general approximators for the unknown functions, no assumptions are made for the data. That is, no prior models are built for the unknown functions. This characteristic is quite different from that of ARMA models. In ARMA (or vector ARMA) models, it is assumed that the time series should be linear and stationary. While the special capability of neural networks makes themselves a *robust analysis* tool for various patterns of time series.

3.1. PERCEPTRONS

Neural networks have many variations. Recent developments in the neural network theory (see Cybento (1989), Funahashi (1989), Hecht-Nielsen (1989), Amari (1990), Kolen and Goel (1991), and Kosko (1992) show that multilayer feedforward neural networks with one hidden layer of neurons can be used to approximate any Borel-measurable function to any desired accuracy. The concept implies that if the nonlinear law governs the system, then even if the dynamic behavior is chaotic, the future may to some extent be predicted from the behavior of past values that are similar to those of the present.

The outputs of neurons in one layer are transmitted to neurons in next layer through links. The interconnection weights between neurons in two adjacent layers l and $l+1$ are written by *the weight matrix* $W_i = (w_{1,ij})$. The ith row of W_1 corresponds to the input weight vector for the ith neuron in layer $l = 1$. The *input* set of the network contains all the neurons in the first layer and the *output* set of the network, all the neurons in the lst layer. Neural networks act as a function, mapping the input space R^m to the output space R^n. The net input to a neuron i in layer l is

$$I_{1,i} = \sum_{j=1}^{1} w_{1,ij} O_{1,j} \,. \tag{3.2}$$

The output of neuron i in layer l is

$$O_{1,i} = g(I_{1,i}) , \tag{3.3}$$

where g is traditionally the sigmoid function but can be any nonlinear differentiable function. For a sigmoid activation function, we have

$$O_{1,i} = \frac{1}{1 + e^{-\lambda(I_{1,i}+\theta_{1,i})}} , \tag{3.4}$$

where λ, called *the slope of the shape*, is used to control the steepness of the function. The effect of the threshold value $\theta_{1,i}$ is to shift the function along the horizontal axis.

Then, each neuron in layer $l + 1$ performs a summing and applies a threshold function of the following form:

$$O_{1+1,i} = g\left(\sum_{j=1}^{p} w_{1,ij} O_{1,j}\right) , \tag{3.5}$$

where $O_{1+1,i}$ is the output of the ith neurons in the $l + 1$ layer.

3.2. NEUROCOMPUTING FOR THE LEAST MEAN SQUARE LEARNING SYSTEM

We now present a device for the identification of certain sets of bilinear time series. The network weights are to be chosen so that a preselected set of binary N-vectors are obtained by squashing equilibrium solutions. These binary vectors will be called memory traces stored by the network. Any initial condition for the network state vector may be used to probe the network, and the resulting steady-state network output vector is called the *evoked response.*

In the first place, we determine the error function as $\boldsymbol{e}_1 = \boldsymbol{O}_l - \hat{\boldsymbol{O}}_1$ = output vector − estimated output vector. The parameter adaptation of the network is done by adjusting the interconnection weight $w_{1,ij}$ and the thresholds $\theta_{1,i}$ toward the direction of minimizing the mean squared error. The total system error is

$$\mathscr{E} = \sum_{1} \mathscr{E}_1 = \frac{1}{2} \sum_{1} \sum_{i} (o_{1,i} - \hat{o}_{1,i})^2 . \tag{3.6}$$

Using the steepest decent technique, we then update the linkweight as follows:

$$\begin{aligned} w_{1,ij}^{new} &= w_{1,ij}^{old} + \Delta w_{1,ij}^{old} \\ &= w_{1,ij}^{old} + \lambda \nabla_w \mathscr{E}_1 . \end{aligned} \tag{3.7}$$

Clearly, the direction of the maximum decreasing is given by $-\nabla_w \mathscr{E}_1 = -(\partial \mathscr{E}_1 / \partial w_{ij})$. To prove that $\mathscr{E}_1$ is differentiable, interests readers may refer to Hecht-Bielsen (1988). We achieve convergence toward the improved values for the weights and thresholds by taking incremental changes $\Delta w_{1,ij}$ proportional to $-\partial \mathscr{E}_1/\partial w_{1,ij}$, that is,

$$\Delta w_{1,ij} = -\lambda \frac{\partial \mathscr{E}_1}{\partial w_{1,ij}}, \tag{3.8}$$

where λ, called learning rate, is used to control the convergent speed of the learning process. A large λ would make the learning speed faster, but is more likely to cause the system to become unstable.

Using the chainrule, the partial derivative $\partial \mathscr{E}_1/\partial w_{1\,ij}$ becomes

$$\begin{aligned}\frac{\partial \mathscr{E}_1}{\partial I_{1,i}} \frac{\partial I_{1,i}}{\partial w_{1,ij}} &= \frac{\partial \mathscr{E}_1}{\partial I_{1,i}} \frac{\partial}{\partial w_{1,ij}} \left(\sum_{j=1}^{1} w_{1,ij} o_{1,j} \right) \\ &= \frac{\partial \mathscr{E}_1}{\partial I_{1,i}} o_{1,j} \,. \end{aligned} \tag{3.9}$$

Let $\delta_{1,i} = -(\partial \mathscr{E}_1/\partial I_{1,i})$. Combining with (3.8) and (3.9) gives

$$\Delta w_{1,ij} = \lambda \delta_{1,i} o_{1,j} \,. \tag{3.10}$$

If the layer l is not the output layer, then

$$\begin{aligned}\delta_{1,i} &= -\frac{\partial \mathscr{E}_1}{\partial \hat{o}_{1,i}} \frac{\partial \hat{o}_{1,i}}{\partial I_{1,i}} \\ &= \frac{\partial}{\partial \hat{o}_{1,i}} \left[-\frac{1}{2} \sum_{j}^{k} (o_{1,j} - \hat{o}_{1,j})^2 \right] g'(I_{1,i}) \\ &= (o_{1,i} - \hat{o}_{1,i}) g'(I_{1,i}) \,. \end{aligned} \tag{3.11}$$

Thus, we calculate the change of the weight between neuron i and every neuron k (including the pseudo) in layer l, and modify the weights:

$$\begin{aligned} w_{1,ij}^{new} &= w_{1,ij}^{old} + \delta w_{1,ij}^{old} \\ &= w_{1,ij}^{old} + \lambda \delta_{1,i}^{old} o_1^0 \,. \end{aligned} \tag{3.12}$$

3.3. THE PRACTICAL APPLICATION OF BACK-PROPAGATION

As with the original least mean square training procedure, the learning rate parameter λ plays an important role in the practical application of Back-propagation. For a given network and an infinitesimal earning rate, the weights that yield the minimum error can be found. However, they may not be found in our lifetime. While it is possible to get excellent fits to training samples, the application of back-propagation is fraught with difficulties for the forecasting performance on test samples. Unlike most other learning systems that have been previously discussed, there are far more choices to be made in applying this gradient decent method. Even the slightest variation can make the difference between good and bad performance. While we will consider many of these variations, there is no universal answer to what is best. The major route for getting the best results is through repeated experimentation with varying experimental conditions. Steps for building neural networks are listed below.

(*a*) *Deciding the number of systems layers and nodes.* Cybento (1989) proved neural networks with one hidden layer are sufficient for a function approximator.

THEOREM 3.1 (*Backpropagation Approximation Theorem*) *Let g be a bounded, measurable sigmoidal function. Then the finite sums of the form*

$$G(x) = \sum_{j=1}^{N} w_j^{(1,2)} g((W_j^{(0,1)})X - \Theta_j), \tag{3.13}$$

are dense in $C([0,1]^m)$. *In other words, given any* $\varepsilon > 0$ *and* $f \in C([0,1]^m)$, *there exists a three-layer Backpropagation neural network that can approximate f within e mean squared error accuracy.*

This theorem resembles the *Weierstrass approximation theorem*: every continuous function on a compact interval is the limit of a uniformly convergent sequence of polynomials. It is important to note that although this theorem shows that three layers are enough, in processing real-world data it is often designed to have four or more layers. This is because for many problems an approximation with three layers would require an impractical large number of hidden units. The problem of hidden neuron size choice is under intensive study, for example, see Kosko (1992) and Wu, Liou and Chen (1992). The exact analysis is rather difficult because of the complexity of the network mapping and due to the non-deterministic nature of many successfully completed training procedures.

(*b*) *Random initial state.* Unlike most other learning system, the neural network begins in a random state. The initial weight of each link is randomly assigned between −1 and 1. The reason for assigning weights within this interval is that the activation function of each hidden unit is a sigmoid function. The sigmoid function has the most steep slope near zero. To assign weights near zero will have most significant learning speed.

(*c*) *Normalization of data.* Since the input domain of the neural network activation functions are limited, the data are usually normalized to an appropriate interval. The scale/location transformation are performed for the input data and are usually to the interval [0, 1]. The output value of sigmoid function falls into the same interval. An inverse transformed will be taken to get the estimate values of the original form.

(*d*) *The learning rate and local minimum.* The process of tuning the link weight of matrix W to achieve desired result is called *learning*. The selection of a learning rate is of curtailed importance in finding true global minimum of the error distance. Too small of a learning rate will make agonizingly slow progress. Too large of a learning rate will proceed much faster, but may simply produce oscillations between relatively poor solutions. Both of these conditions are generally detectable through experimentation after a number of training epochs. Empirical evidence support the notion that the use of *momentum* conception can be helpful in speeding convergence as avoiding local minimum (see Weiss and Kulilowski, 1990). Momentum towards convergence is maintained by making nonradical revisions to the weights.

(*e*) *Stopping rules.* We pre-assign an accuracy measure, and if the error reaches to the assigned preassigned value the iterative procedure will halt. In practice, for an arbitrary neural net, the convergent rate may be very slow.

(*f*) *Model-free forecasting.* After a neural networks is properly built, we use it to predict future value by inputting the present data *without knowing its specific model.*

4. Performance of Forecasting on Simulated Study

4.1. AN ILLUSTRATED EXAMPLE

Our study is focused on simple bilinear time series because they are frequently encountered in practice, and because they are widely used in the literature.

$$\textit{set } 4.1\text{:}\begin{cases} x_{t+1} = 0.2x_t\varepsilon_t + \varepsilon_{t+1} \\ x_{t+1} = 0.5x_t\varepsilon_t + \varepsilon_{t+1}\,; \\ x_{t+1} = 0.8x_t\varepsilon_t + \varepsilon_{t+1} \end{cases}$$

$$\textit{set } 4.2\text{:}\begin{cases} x_{t+1} = 0.2x_{t-1}\varepsilon_t + \varepsilon_{t+1} \\ x_{t+1} = 0.5x_{t-1}\varepsilon_t + \varepsilon_{t+1}\,; \\ x_{t+1} = 0.8x_{t-1}\varepsilon_t + \varepsilon_{t+1} \end{cases}$$

$$\textit{set } 4.3\text{:}\begin{cases} x_{t+1} = 0.2x_t\varepsilon_{t-1} + \varepsilon_{t+1} \\ x_{t+1} = 0.5x_t\varepsilon_{t-1} + \varepsilon_{t+1}\,; \\ x_{t+1} = 0.8x_t\varepsilon_{t-1} + \varepsilon_{t+1} \end{cases}$$

$$\textit{set } 4.4\text{:}\begin{cases} x_{t+1} = 0.2x_t + 0.2x_t\varepsilon_t + \varepsilon_{t+1} \\ x_{t+1} = 0.5x_t + 0.5x_t\varepsilon_t + \varepsilon_{t+1}\,; \\ x_{t+1} = 0.8x_t + 0.8x_t\varepsilon_t + \varepsilon_{t+1} \end{cases}$$

$$\textit{set } 4.5\text{:}\begin{cases} x_{t+1} = 0.2x_t + 0.2x_t\varepsilon_t - 0.2x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1} \\ x_{t+1} = 0.5x_t + 0.5x_t\varepsilon_t - 0.5x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1}\,; \\ x_{t+1} = 0.8x_t + 0.8x_t\varepsilon_t - 0.8x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1} \end{cases}$$

$t = 1, \ldots, n$.

Sample sizes 500 were obtained as follows. The normally distributed innovations $\varepsilon_t \sim N(0, 1)$, were generated by using `Minitab 8.2 software` on a 486-PC computer. The last 8 observations were kept for the comparison of forecasting performance. Let the initial value $x^0 = 1.0$, the following data sets are realization of bilinear models (i) to (v) and draw in Figure 2.

Neural networks approaches for bilinear time series

According to the procedure stated at section 3, we design a neural networks as follows.

- *Inputs & Outputs.* Because the decision of the layers and nodes numbers relies on proper training data, there are no strict answer for it. It is because an approximator with three layers would require an impractical large number of hidden units. Another question of choosing the hidden neurons' size is under intensive study (Kosko (1992) and Wu, Liou and Chen (1992)). The rules for getting more optimal results are to repeat our experimentation with varying experimental conditions. To sum up, the networks must efficiently integrate the largest possible number of relevant factors. For instance, the AR(p) model used for this type of problem suggests *p-neurons* the number of input. One output is designed for the neural system to give the predictions. We should note that p is just a good suggestion.

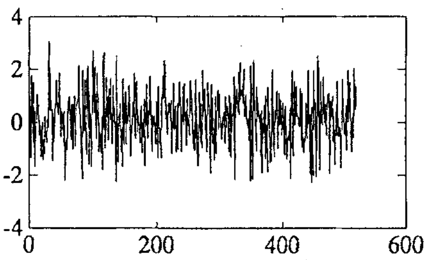
- *Data Standardization Transformation.* We standardized the data by setting

$$x_i^{new} = \frac{x_i^{old}}{max\{x_t\} - min\{x_t\}} \, .$$

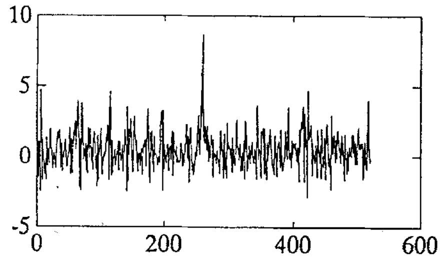
This transformation serves as a dynamic range limiter.

- *The number of hidden layers and hidden neurons.* The hidden layers represent the equivalent of feature space. The difficulty resides in determining the number of hidden layers, for each one the number of neurons and propagation law. According to our experiences, one hidden layer is enough for training linear time series while two hidden layers are appropriate for training a nonlinear time series. Each hidden layer contains about 40 neurons, and generates 1600 links between the hidden layer. Larger networks are more capable. But the computer memory is limited.

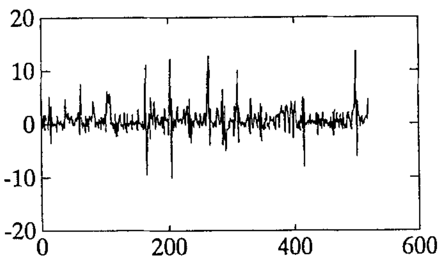
- *Networks initialization.* We train the neural network more than 20,000 times, until the error level is accepted, and the system is approximately close. We set the network initial conditions as:
  `Input range: [-2, 2]`


$x_{t+1} = 0.2x_t\varepsilon_t + \varepsilon_{t+1}$

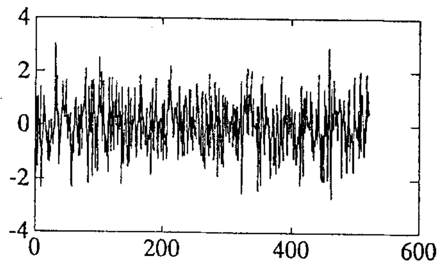
(4.1.1)


$x_{t+1} = 0.5x_t\varepsilon_t + \varepsilon_{t+1}$

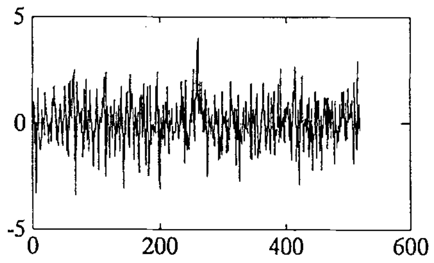
(4.1.2)


$x_{t+1} = 0.8x_t\varepsilon_t + \varepsilon_{t+1}$

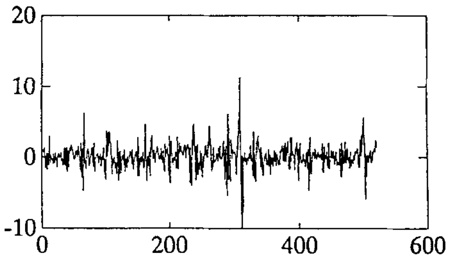
(4.1.3)


$x_{t+1} = 0.2x_{t-1}\varepsilon_t + \varepsilon_{t+1}$

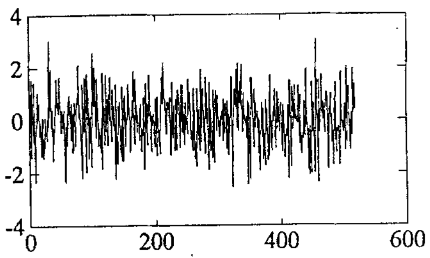
(4.2.1)


$x_{t+1} = 0.5x_{t-1}\varepsilon_t + \varepsilon_{t+1}$

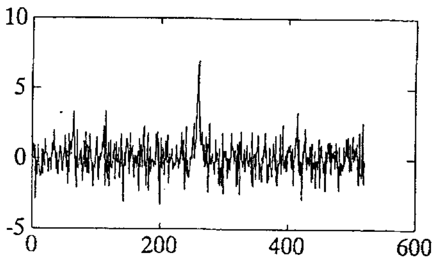
(4.2.2)


$x_{t+1} = 0.8x_{t-1}\varepsilon_t + \varepsilon_{t+1}$

(4.2.3)

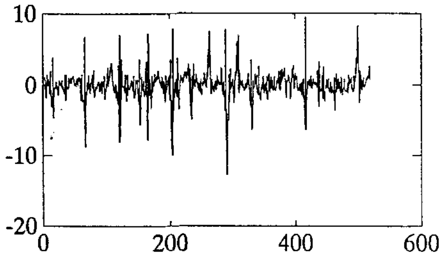

$x_{t+1} = 0.2x_t\varepsilon_{t-1} + \varepsilon_{t+1}$

(4.3.1)


$x_{t+1} = 0.5x_t\varepsilon_{t-1} + \varepsilon_{t+1}$

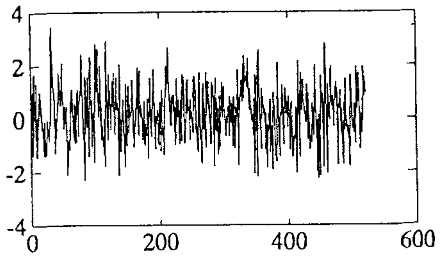
(4.3.2)

Fig. 2. Realizations of bilinear models (4.1) to (4.5).



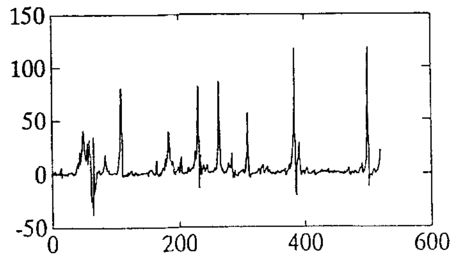
$x_{t+1} = 0.8x_t\varepsilon_{t-1} + \varepsilon_{t+1}$

(4.3.3)



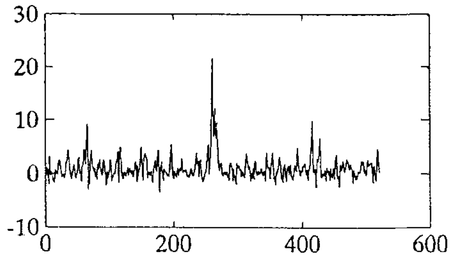
$x_{t+1} = 0.2x_t + 0.2x_t\varepsilon_t + \varepsilon_{t+1}$

(4.4.1)



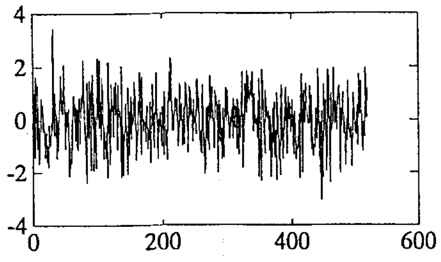
$x_{t+1} = 0.5x_t + 0.5x_t\varepsilon_t + \varepsilon_{t+1}$

(4.4.2)



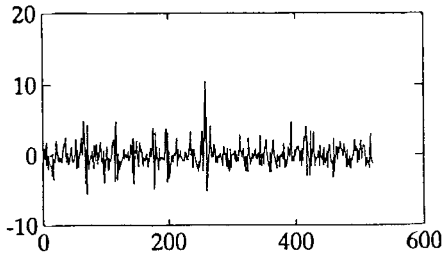
$x_{t+1} = 0.8x_t + 0.8x_t\varepsilon_t + \varepsilon_{t+1}$

(4.4.3)



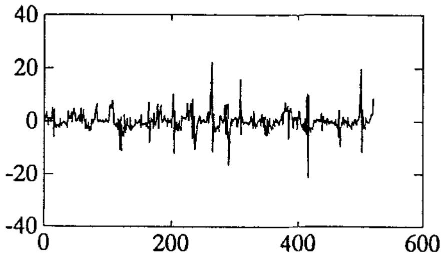
$x_{t+1} = 0.2x_t + 0.2x_t\varepsilon_t - 0.2x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1}$

(4.5.1)



$x_{t+1} = 0.5x_t + 0.5x_t\varepsilon_t - 0.5x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1}$

(4.5.2)



$x_{t+1} = 0.8x_t + 0.8x_t\varepsilon_t - 0.8\ x_{t-1}\varepsilon_{t-1} + \varepsilon_{t+1}$

(4.5.3)

Fig. 2. (*continued*)

```
Output range: [0, 1]
Initial (random) weight range: [-0.1, 0.1]
The slope of sigmoid function: 0.5
Learning rule:Delta rule.
```

- *Training and learning.* The time of training is dependent upon the characteristics of neural networks. While more training time will facilitate the networks remember the pattern of function well, it may ignore the slight local variant in the pattern of the system. On the other hand, less training times correlates to a more sensitive network to the local variance and may get an absurd global pattern for the system.

Stopping rules

Two techniques are presented to determine when to stop:

(i) Limit the number of epochs: training ceases after a fixed upper limit on the number of training epochs. For example, when the training error yields a pre-assigned value 0.01 or the training epochs are more than 20,000, the training procedure will cease, and the link weights would be used as the final solution.
(ii) Measured progress criterion: the error can be sampled and averaged over a fixed number of epochs, e.g., every 100 epochs of training. If the average error distance for the most recent 100 epochs is not better than that for previous 100, one might conclude that no progress is begin made, and training should halt.

Table 1 shows the mean and variance for the simulated data and the fitted values calculated by designed neural networks.

Note that the number of input nodes as well as the number of the nodes each hidden layer are chosen by *experiments*. There are numerous combination methods in constructing a networks to get a robust fitted for the simulated data.

In Table 2, we compare the MSE (mean of square errors) for the neural networks with the best fitted ARMA models, which we find are all AR(1) models according to the Box-Jenkins procedures and *AIC*, *SBC* criterion. An interesting and striking result from Table 2 is that the best linear ARMA models fit the

Table 1. Comparison of the mean and variance given in the bracket for the realization data with the neural networks approach

Data set	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2	4.2.3	4.3.1
Simul	$.19_{(1.04)}$	$.53_{(1.38)}$	$.89_{(2.39)}$	$.00_{(1.01)}$	$.01_{(1.12)}$	$.03_{(1.79)}$	$.00_{(1.01)}$
NN	$.19_{(0.17)}$	$.36_{(0.16)}$	$.94_{(0.31)}$	$-.04_{(.13)}$	$-.02_{(.10)}$	$.18_{(0.15)}$	$.15_{(0.14)}$

4.3.2	4.3.3	4.4.1	4.4.2	4.4.3	4.5.1	4.5.2	4.5.3
$.05_{(1.22)}$	$.03_{(2.24)}$	$.25_{(1.07)}$	$1.1_{(2.29)}$	$5.06_{(13.7)}$	$.00_{(1.06)}$	$.00_{(1.70)}$	$.25_{(3.8)}$
$.17_{(0.19)}$	$.00_{(0.37)}$	$.31_{(0.26)}$	$.91_{(1.54)}$	$4.04_{(9.34)}$	$.62_{(0.21)}$	$-.33_{(0.2)}$	$.57_{(0.91)}$

Table 2. Comparison of MSE for best fitted ARMA models with the neural networks approach

Data set	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2	4.2.3	4.3.1	4.3.2	4.3.3
$AR_{(1)}$	1.08	1.85	5.58	1.03	1.24	3.20	1.03	1.44	5.02
NN	0.74	0.56	11.99	0.89	0.61	4.36	0.91	0.53	13.08
Advantage	NN	NN	AR	NN	NN	AR	NN	NN	AR

Data set	4.4.1	4.4.2	4.4.3	4.5.1	4.5.2	4.5.3
$AR_{(1)}$	1.09	2.77	119.8	1.11	2.76	14.15
NN	0.77	2.65	133.1	0.98	2.27	36.11
Advantage	NN	NN	AR	NN	NN	AR

Table 3. Sum of square error for 8-steps forecasting

Data set	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2	4.2.3	4.3.1	4.3.2	4.3.3
$AR_{(1)}$	5.92	3.30	81.55	7.37	4.81	43.14	7.38	4.81	102.9
NN	4.96	2.49	95.92	6.16	3.93	34.85	5.34	3.21	104.6
Advantage	NN	NN	AR	NN	NN	NN	NN	NN	AR

Data set	4.4.1	4.4.2	4.4.3	4.5.1	4.5.2	4.5.3
$AR_{(1)}$	5.53	10.48	8145	4.70	15.06	212.7
NN	5.05	8.25	3328	3.82	18.21	268.9
Advantage	NN	NN	NN	NN	AR	AR

bilinear samples from (4.1.3), (4.2.3), (4.3.3), (4.4.3) and (4.5.3), whose coefficients of the bilinear term are high, better than the non-linear neural networks.

Forecasting performance

In Table 3, we compare the sum of square error for 8-steps forecasting for the best fitted ARMA model and the selected neural network. Table 4 compares the sum of absolute percent error for 8-steps forecasting errors.

It is shown that the performance of the NN *model* take the advantage to AR as a rate about 2:1. Still we must mention that the neural networks in this research is only a *better* network, the best network structure may be unknown.

Table 4. Sum of absolute percent error for 8-steps forecasting

Data set	4.1.1	4.1.2	4.1.3	4.2.1	4.2.2	4.2.3	4.3.1	4.3.2	4.3.3
$AR_{(1)}$	46.72	2.49	6.51	7.92	7.22	8.04	7.95	7.55	8.69
NN	41.25	1.58	7.79	6.20	11.53	9.13	6.17	9.07	9.78
Advantage	NN	NN	AR	NN	AR	AR	NN	AR	NN

Data set	4.4.1	4.4.2	4.4.3	4.5.1	4.5.2	4.5.3
$AR_{(1)}$	9.54	4.16	52.83	7.26	7.93	9.21
NN	8.90	3.41	16.91	15.95	6.09	8.77
Advantage	NN	NN	NN	AR	NN	NN

5. Conclusions and Further Research

Two methodologies have been presented in this paper for the identification and forecasting of the nonlinear time series. Having outlined the main features of the identifier system, we can concentrate on its most remarkable properties as compared with traditional test criteria. Our aims are to clarify the degree of similarity between the two classes of systems, and to compare their learning strategies.

The result gained in this research has shown practical applications when the underlying model for the observations are uncertain. The advantage of the neural networks proposed in this article is that it provides a methodology for model-free approximation, i.e., the weighted matrix estimation is independent of any model. It liberates us from the modeling-based selection procedure with no assumptions of the sample data being made.

There remains many problems to be improved, such as:

(i) Problems related to identification: (a) there is a need to develop a procedure for identification for the case of interacting noise; (b) the convergence of the algorithm for combined state and parameters estimation needs to be proved; (c) a solution needed to the power problem based on test statistics in section 2.
(ii) Problems relating to network forecasting: (a) Can a particular network be treated as a specific nonlinear design?; (b) In what way does the intial conditions affect the properties of network system?; Does the system become less (or more) sensitive to external signals?; (c) What knowledge is required to obtain prescribed behavior of outlier data?; (d) How to obtain information from chaotic trajectories about the network design?

From the above, it will be evident that the property of identification and robust forecasting are still at the beginning stage. We hope the neurocomputing will be a worthwhile approach and will stimulate future empirical work in time series analysis.

Acknowledgement

The author wishes to thank the referee, whose valuable comments and critique enhanced the clarity of this paper.

References

Ashey, R. and Patterson, D., 1985, Linear versus nonlinear macroeconomics: A statistical Test. Virginia Polytechnic Institute, Blacksburg, VA.

Chan, W.S. and Tong, H., 1986, On test for non-linearity in time series analysis. *J. Forecasting* **5**, 217–28.

Chatfield, C., 1993, Neural networks: Forecasting breakthrough or passing fad? *Int. J. of Forcasting*, **9**, 1–3.

Cybento, G., 1989, Approximation by superposition of a sigmoidal function, *Mathematics of control, signals and system* **2**, 303–14.

Funashi, K.-I., 1989, On the appropriate of continuous mappings by neural networks, *Neural Networks* **2**, 183–92.

Gooijer, J. G. and Kumar, K., 1992, Some recent developments in nonlinear time series modelling, testing, and forecasting. *Int. J. Forecasting* **8**, 135–56.

Granger, C.W.J., 1991, Developments in the nonlinear analysis of economic series. *Scand. J. of economics* **93**(2), 263–76.

Granger, C.M.J. and Andersen, A., 1978, Non-linear time series modelling. Applied Time Series Analysis, 25–38, ed. Findley. Academic Press, N.Y.

Guegan, D. and Pham, T.D., 1992, Power of the score test against bilinear time series models. *Statistica Sininca* **2**, 1, 157–69.

Hecht-Nielsen, R., 1989, Neurocomputing, IEEE Spectrum, March, 36–41.

Hinich, M., 1982, Testing for Gaussianity and linearity of a stationary time series. *J. Time series Analysis* **3**, (3), 169–76.

Kolen, J.F. and Goel, A.K., 1991, Learning in parallel distributed processing networks: Computational complexity and information content. IEEE Transactions on Systems, Man, and Cybernetics, **21**, (2), 359–67.

Kosko, B., 1992, Neural networks and fuzzy systems. Englewood Cliffs, NJ: Prentice Hall.

Luukkonen, R., Saikkonnen P. and Terasvirta, T., 1988, Testing linearity against smooth transition autocorrelation models. *Biometrica* **75**, 491–500.

Marchuk, G.I. 1983, Mathematical models in Immunology. Optimization Software Publication, Springer-Verlag, New York.

Mohler, R.R., 1973, Bilinear control processes, Academic Press, New York.

Mohler, R.R. and Ruberti, A., 1973, Theory and applications of variable structure system. Academic Press, New York.

Ramey, J., 1989, Neural computing. **1**, Neural Ware, Inc., Pittsburgh.

Saikkonen, P. and Luukkonnen, K., 1988, Lagrange multiplier test for testing non-linearities in time series models. *Scand. J. of Statistics* **15**, 55–68.

Saikkonen, P. and Luukkonnen, K., 1991, Power properties of a time series linearity test against some simple bilinear alternatives. *Statistica Sinica* **1**, (2), 453–64.

Subba Rao, T. and Gabr, M.M., 1984, An introduction to bispectral analysis and bilinear time series models. Lecture Notes in statistics, Springer-Verlag, London.

Tong, H., 1990, Non-linear time series. Oxford University Press, Oxford.

Tsay, R.S., 1989, Testing and modeling threshold autoregressive processes. *Journal of the American Statistical Association*, 84 231–40.

Tsay, R.S., 1991, *Detecting and modeling nonlinearity in univariate time series anlaysis. Statistica Sinica* **1**, (2), 431–51.

Weiss, A.A., 1986, ARCH and bilinear time series models: comparison and combination. *J. Business & Economic Statistics* **4**, (1), 59–70.

Weiss, S.M. and Kulikowski, C.A., 1990. Computer systems that learn. Morgan Kaufmann Publishers, Inc. San Mateo, California.

White, H., 1992, Artificial neural networks: Approximation and learning Theory. Blackwell Publisher, Massachusetts.

Wu, B., Liou, W. and Chen, Y., 1992, Robust forecasting for the stochastic models and chaotic models. *J. Chinese Statist. Assoc.* **30**, (2), 169–89.

Wu, B. and Shih, N., 1992, On the identification problem for bilinear time series models. *J. Statist. Comput. Simul.* **43**, 129–61.