Available online at www.sciencedirect.com
SCIENCE DIRECT®
Applied Soft Computing 5 (2005) 333–355
Applied Soft Computing
www.elsevier.com/locate/asoc

Prediction of a Lorenz chaotic attractor using two-layer perceptron neural network

Sanjay Vasant Dudul*

Department of Applied Electronics, Amravati University, Amravati 444602, Maharashtra, India

Received 9 January 2003; received in revised form 9 March 2004; accepted 20 July 2004

Abstract

This paper investigates the prediction of a Lorenz chaotic attractor having relatively high values of Lypunov's exponents. The characteristic of this time series is its rich chaotic behavior. For such dynamic reconstruction problem, regularized radial basis function (RBF) neural network (NN) models have been widely employed in the literature. However, author recommends using a two-layer multi-layer perceptron (MLP) NN-based recurrent model. When none of the available linear models have been able to learn the dynamics of this attractor, it is shown that the proposed NN-based auto regressive (AR) and auto regressive moving average (ARMA) models with regularization have not only learned the true trajectory of this attractor, but also performed much better in multi-step-ahead predictions. However, equivalent linear models seem to fail miserably in learning the dynamics of the time series, despite the low values of Akaike's final prediction error (FPE) estimate. Author proposes to employ the recurrent NN-based ARMA model with regularization which clearly outperforms all other models and thus, it is possible to obtain good results for prediction and reconstruction of the dynamics of the chaotic time series with NN-based models.

Keywords: Chaotic time series; Multi-layer perceptron neural network; Neural Network based AR and ARMA model; System identification; Regularization

1. Introduction

A time series is a series of observations of a dynamic system, which are taken at regular time intervals. If the future behavior of the dynamic system should be determined, we speak of forecasting or predicting the time series. In time series analysis, an experiment is not conducted as it is in system identification since the system cannot be influenced (i.e., no external input). The results of a phenomenon over time form a time series. Although, sometimes outcomes of processes described through mathematical closed forms (known deterministic functions) are also viewed as time series, most common time series are the result of unknown or not completely understood processes. Its main characteristic is that its evolution cannot be described exactly as in the case of a known deterministic function of t. It is human nature

* Tel.: +91 721 2666127; fax: +91 721 2662135.
E-mail addresses: dudulsv@rediffmail.com, svdudul@indiatimes.com.

1568-4946/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2004.07.005

to have the desire to know in advance what is likely to happen in the future. The observation of past outcomes of a phenomenon in order to anticipate its future behavior represents the essence of forecasting (prediction). If a mathematical model describing a studied phenomenon is known, forecasting becomes a trivial and degenerate task. However, if a model of the phenomenon is either unknown or incomplete, forecasting is not so easy.

The problem of learning from examples is essentially ill-posed, i.e., there are infinitely many models, which fit the training data well, but few of these generalize well. Dynamic reconstruction may be defined, as the identification of a mapping that provides a model for an unknown dynamical system, which generates a time series. This problem is, in reality, an ill-posed inverse problem for one or more of the following reasons. First, for some unknown reason the existence condition may be violated. Second, there may not be enough information in the observable time series to reconstruct the non-linear dynamics uniquely; hence, the uniqueness criterion is violated. Third, the unavoidable presence of additive noise or some form of imprecision in the observable time series adds uncertainty to the dynamic reconstruction. In particular, if the noise level is too high, it is possible for the continuity criterion to be violated. In view of this, the dynamic reconstruction problem should be made well posed. It is essential to include some form of prior knowledge about the input–output mapping. In other words, some form of constraints (e.g., smoothness of input–output mapping) need to be enforced on the predictive model designed for solving the dynamic reconstruction problem. One effective way in which this requirement can be satisfied is to use Tikhonov's regularization theory [1]. Differential equations are an important type of a priori knowledge available in many engineering problems and physical systems. They are the principal technique for modeling dynamic systems, because the causal relations between physical variables can be described well by differential equations.

One of the primary reasons for studying neural networks (NN) was to create a computer program that was able to learn from experience. When the experience is interpreted as knowledge about how certain inputs affect a system, it is evident that NNs must have something in common with the techniques applied in system identification and control. They are instruments that in a broad sense can learn non-linear mappings from a set of observations. A traditional application of NNs is time series forecasting. A hierarchical neural model is reported using self-organizing feature map for short-term load forecasting, where it is compared with MLP [2]. Suitability of MLP and radial basis function NNs is explored in forecasting engine systems reliability [3]. Neural networks can be used as a black-box model of the unknown process generating the time series, and the network can be trained to forecast the future behavior, given the past observations of the series. In this context, NN design algorithms can be used to construct NNs for forecasting the future behavior of time series. In case the knowledge of the generating process is complete, the network will be fully specified by the design algorithm. If the knowledge is incomplete, however, the network will contain free parameters, which can be determined by using common NN algorithms. In this way, any knowledge in the form of differential equations can be incorporated into the NN and the discrepancies with respect to the real process can be reduced or even removed by additional learning. Taylor series method for generating polynomial approximations of differential equations can be generalized to non-polynomial function, φk. This method permits the construction of NNs for the approximation of differential equations. NN architectures can be found for φk that satisfies solvability conditions (triangulation and invertibility) [4–5].

In particular, the multilayer perceptron (MLP) network has gained an immense popularity. From numerous practical applications published over the past decade, there seems to be sizeable evidence that MLP indeed possesses an impressive ability. Lately, there have also been some theoretical results that attempt to explain the reasons for this success [6,7]. However, so far as solution to the prediction and dynamic reconstruction problem of chaotic time series is concerned, it is common to see the regularized RBF NN in the literature [8–12]. This is because, in practice, as an integral part of the design of RBF NN, regularization theory has been included in a mathematically tractable manner [5,13].

The present study deals with the modeling of a Lorenz chaotic attractor using a two-layer MLP NN.

First, very well established linear system identification theory has been used to estimate all possible linear models. Second, NN-based Auto Regressive (AR) model and Auto Regressive Moving Average (ARMA) model, which constitute a two-layer perceptron NNs are used. These network models are trained in both regularized and unregularized mode. In this paper, we propose to use an innovative approach where a recurrent NN model using the ARMA coefficients derived from the chaotic time series as inputs; built around a two-layer MLP is trained in a regularized mode. This approach uses a NN with a single hidden layer (two-layer MLP NN) with hyperbolic tangent units and a linear output unit, because it can be shown that an MLP NN that has only one hidden layer, with sufficient number of neurons, acts as a universal approximator of non-linear mappings [14]. For initialization of synaptic weights, approach suggested by Nguyen and Widrow [15] is used, because this significantly improves the speed of network training. This approach is shown to predict and reconstruct the dynamics of chaotic time series with reasonable accuracy.

Two-layer MLP NN with a single hidden layer is the simplest possible configuration of the NN, which is shown to solve the dynamic reconstruction problem. Finally, a comparison has been made between linear model and NN-based models including the proposed recurrent NN-based ARMA model with regularization and subsequently, it is shown that using our approach, a significantly better models are estimated in view of the performance in multi-step-ahead prediction of validation data.

2. Lorenz chaotic attractor

In science, chaos is used as a synonym for irregular behavior, whose long-term development is essentially unpredictable [16]. Chaotic differential equations show not only irregular behavior, but they are also unstable with respect to small perturbations of their initial conditions. Consequently, it is difficult to forecast the future of the time series based on chaotic differential equations, and they should be a good benchmark for NN applications.

When we think of chaotic or unpredictable behavior, an example that evidently comes to mind is the weather. Because of the economic importance of having accurate weather predictions, a good deal of effort has been dedicated to this problem. While much of this effort has gone into computer modeling of Earth's atmosphere, much has also been devoted to understanding the weather problem from a more basic point of view. It was work of this kind by the atmospheric scientist Lorenz (1963) [17] that gave a major contribution to the modern field of chaos.

Lorenz was studying the basic equations of fluid mechanics, which are known as the Navier–Stokes equations: they can be thought of as Newton's laws written in a form suitable for a fluid. These are a complex set of differential equations that describe the velocity, temperature, density, etc., as functions of position and time, and they are very difficult to solve analytically in cases of practical interest. Certainly, this is just the type of problem where a computational approach can be useful, and that is precisely what Lorenz did. The particular situation he considered was the Raleigh–Benard problem, which concerns a fluid in a container whose top and bottom surfaces are held at different temperatures. It had long been known that as the difference between these two temperatures is increased, the fluid could undergo transitions from a stationary state (no fluid motion) to steady flow (non-zero flow velocities that are constant in time, also referred to as convection) to chaotic flow. Lorenz did his work 41 years ago, so the computational power available to him was not very imposing by today's standards. This encouraged him to consider a greatly simplified version of the Navier–Stokes equations as applied to this particular problem. Indeed, he grossly oversimplified the problem as he reduced it to only three equations.

$$\begin{aligned} \frac{\mathrm{d}x(t)}{\mathrm{d}t} &= \sigma[y(t) - x(t)] \\ \frac{\mathrm{d}y(t)}{\mathrm{d}t} &= x(t)[r - z(t)] - y(t) \\ \frac{\mathrm{d}z(t)}{\mathrm{d}t} &= x(t)y(t) - bz(t) \end{aligned} \tag{1}$$

These are now known as the Lorenz equations (or equivalently, the Lorenz model). Here σ, r, and b are dimensionless parameters. Typical values for these parameters are $\sigma = 10$, $b = 8/3$, and $r = 28$. As noted

above, the Navier–Stokes equations, from which those in Eq. (1) are derived, involve the state of the fluid as a function of position and time. Hence, a complete description of the Raleigh–Benard problem must contain a large number of variables. The Lorenz variables x, *y,* and *z* are derived from the temperature, density, and velocity variables in the original Navier–Stokes equations, and the parameters σ, *r*, and *b* are measures of the temperature difference across the fluid and other fluid parameters. However, it is not particularly useful to insist on interpreting *x*, *y,* and *z* in that manner, since the simplifications made in reducing the problem to only three variables means that we cannot expect our results to apply to any real system. Rather, the behavior shown by these equations is indicative of the type of behavior that could be expected of the Raleigh–Benard problem or any other problem involving the Navier–Stokes equations. For conciseness, the latter problem can be referred to as the weather problem. Any behavior we find in the Lorenz model will certainly be found in the weather problem.

This study is aimed at predicting the Lorenz system using NN. Lorenz's strange attractor demonstrates a rich chaotic behavior that is very complex. On the one hand, Lorenz strange attractor is deterministic because its operation is managed by fixed rules. On the other hand, it shows a complicated behavior that looks random. However, randomness does not vanish by acquiring more information. As a rule, the future behavior of a chaotic system is entirely decided by the past, but practically any small uncertainty in the selection of initial conditions increases exponentially with time. As a result, although short-term predictability of the dynamic behavior of a chaotic system is possible, long-term predictability is not possible. A chaotic time series is therefore ironic, because its generation is ruled by a deterministic dynamical system and still it has a random like look.

The Eq. (1) has been evaluated using Runge Kutta's fourth order numerical integration method. Following data have been assumed to generate the chaotic time series.

Initial conditions : x_0

$= 1.000,\ y_0 = 1.000,\ z_0 = 1.000,$ step size Δt

$= 0.05,$ number of samples obtained $= 100,000$

For dynamic reconstruction of the system, delay coordinate embedding of one of the dimensions, *x* is used.

2.1. Computation of normalized embedding delay and embedding dimension

Embedding allows one to do a delay coordinate embedding on a data set, whether it is one produced from a sample system or from previously loaded data. The concept of delay coordinates is simple. If we can only view one variable from a system, $X(t)$, and we want to reconstruct a higher-dimensional attractor, we can consider $(X(t), X(t + \tau), ..., X(t + n\tau))$ to produce a $(n + 1)$-dimensional coordinate which can be plotted. It is important to select a good value for τ, the delay. If the delay is too short, then $X(t)$ is very similar to $X(t + \tau)$ and when plotted all of the data stays near the line $X\{t\} = X\{t + \tau\}$. If the delay is too long, then the coordinates are in essence independent and no information can be obtained from the plot. In order to decide good settings for an embedding we can employ the autocorrelation function and mutual information function to get a good choice of delay time, and we can use the false near neighbors' method to find a good choice of embedding dimension.

The autocorrelation is defined from the standard deviation, sigma_x = sqrt(< (x− < x >)^2 >), of a time series x, where < > denotes the cumulant. The correlation *r* between two time series *x* and *y* is then defined as: r = < (x − < x >) (y − < y >) >/(sigma_x sigma_y).

The autocorrelation $r(\tau)$ is given by the correlation of the series with itself; use $x(t)$ and $x(t + \tau)$ as the two time series in the correlation formula. It measures how well correlated *x* and *y* values are under different delays. A good choice for the delay time to use when embedding the data should be the first time the autocorrelation crosses zero. This signifies the lowest value of delay for which the *x* and *y* values are uncorrelated. However, since the autocorrelation function is a linear measure it does not offer accurate results in all situations. This autocorrelation function is plotted in Fig. 1(a), from which it is seen that τ is between 85 and 90. Clearly, this cannot be the lowest delay. Hence, in this case, we do not rely on the results based on autocorrelation.

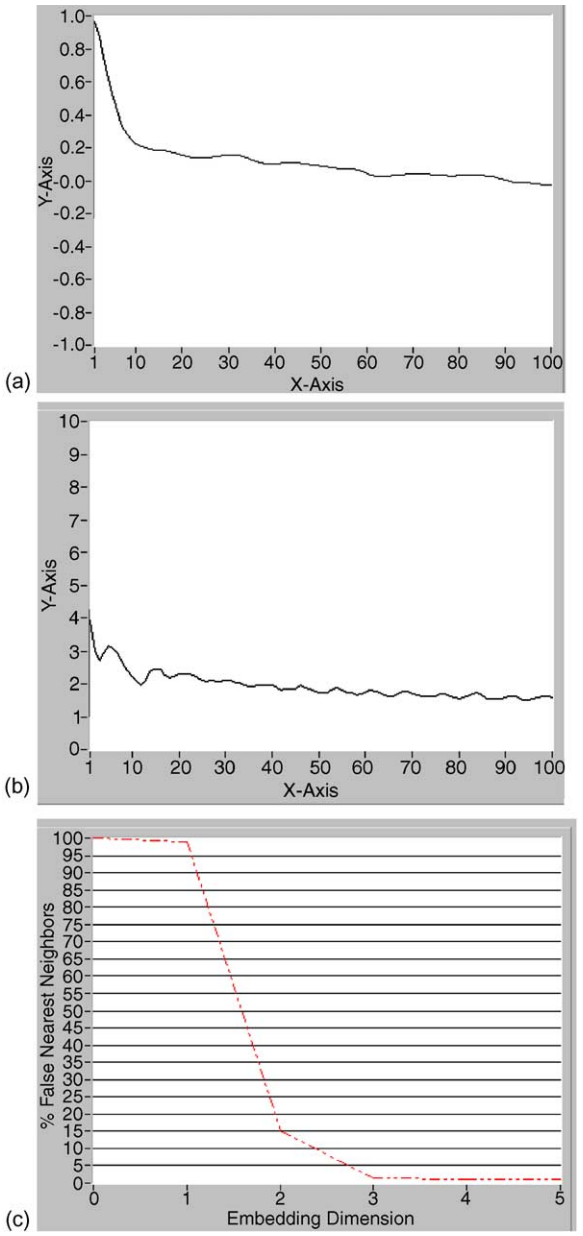

Fig. 1. (a) Autocorrelation. (b) Mutual information. (c) %False nearest neighbors vs. embedding dimension.

The method of "mutual information" is a technique to determine a useful delay time for plotting attractors. The method of mutual information for finding the delay τ was proposed in an article by Fraser and Swinney [18]. The idea is that a good choice for τ is one that, given $X(t)$, provides new information with measurement $X(t + \tau)$. To quote from the article, mutual information I is the answer to the question, "Given a measurement of $X(t)$, how many bits on the average can be predicted about $X(t + \tau)$?" We want $I(\tau)$ to be small. Practically speaking, the way this method works is to calculate $I(\tau)$ for a given time series. A graph of $I(\tau)$ starts off very high (given a measurement $X(t)$, we know as many bits as possible about $X(t+0) = X(t)$). As τ is increased, $I(\tau)$ decreases, then usually rises again. Fraser and Swinney suggest using the first minimum in $I(\tau)$ to select τ. When we specify a minimum and maximum delay, the routine computes the mutual information for all values between those delays. One should locate a reasonable value of the delay to use when embedding at the first minimum of the mutual information.

This is superior to the autocorrelation function, which is another method of determining delay times. However, unlike the autocorrelation, the mutual information function takes non-linear correlations into account. Fig. 1 (b) shows the plot of mutual information. It is observed that the first minimum of the mutual information occurs when $\tau = 3.8$, which is rounded to 4. Therefore, the normalized embedding delay, $\tau = 4$.

The false near neighbors dimension is a method of choosing the minimum embedding dimension of a one-dimensional time series. This method finds the nearest neighbor of every point in a given dimension, then checks to see if these are still close neighbors in one higher-dimension. The percentage of false near neighbors should fall to zero when the correct embedding dimension has been attained. The appropriate embedding dimension is a bit of a judgment call. If the number of false near neighbors is not zero but is very small, the embedding dimension may still be suitable for most forms of analysis.

For the determination of the embedding dimension, we set the criterion to 10 and the delay to 4. The resulting plot of percentage false nearest neighbors is depicted in Fig. 1 (c), from which it is seen that the percentage of false nearest neighbors has dropped sharply to less than about 2.5% for the embedding dimension of three. For a given Lorenz data, an embedding of three should clearly reveal that a two-dimensional embedding is appropriate. If the data is noisy, then it will not reach zero, but it should reach a

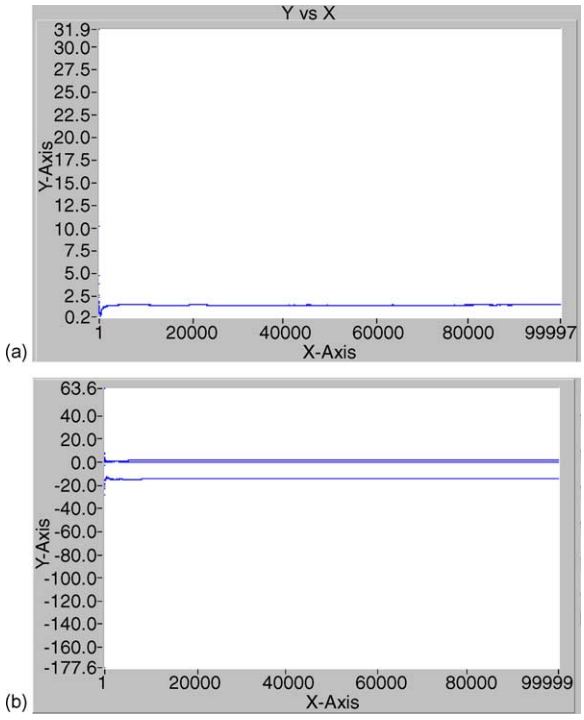

Fig. 2. (a). Dominant Lypunov exponent. (b). Lypunov spectrum.

low plateau at the appropriate embedding dimension. An n-dimensional neighbor is considered false if the distance between the point and its neighbor in the n + 1st-dimension is greater than a criterion $\times$ the average distance between the point and its neighbor in each of the previous n-dimensions [19].

Thus, using the mutual information function, normalized embedding delay is calculated to be τ = 4 and from the percentage of false nearest neighbors embedding dimension as, D_E = 3. In the delay embedding theorem due to Takens (1981), it is shown that if the dynamical system and the observable are generic, then the delay coordinate map from a d-dimensional smooth compact manifold to $\Re^{2d+1}$ is a diffeomorphism on that manifold, where d is the dimension of the state space of the dynamical system [20,21].

Lypunov exponents are statistical measures that explain the doubt about the future state of the attractor. More distinctively, they quantify the exponential rate at which nearby trajectories separate from each other while moving on the attractor. As our system is three-dimensional, it has a total of three Lypunov exponents that can be positive, negative, or zero. Positive exponents give an explanation for the instability of the orbit throughout the state space. Negative ones, on the other hand, control the decay of the transients in the orbit. A zero denotes the fact that the underlying dynamics responsible for the generation of chaos are describable by a coupled system of non-linear differential equation, that is, the chaotic process is a flow. Following Fig. 2 (a) and (b), show the plot of dominant Lypunov exponent and Lypunov spectrum, respectively. From Fig. 2 (a), the dominant Lypunov exponent is computed as 1.35252. All Lypunov exponents can be computed from Fig. 2 (b).

From Fig. 2 (a) and (b), for our system, Lypunov exponents are computed as 1.35252 (dominant Lypunov exponent), 0.38263, and −14.149821 using 100,000 samples of dimension x of the chaotic time series. The first 1000 samples (1:1000) of the time series are used for training purpose and another 400 samples (1101:1500) for validation or testing. The training data and the data used for testing the estimated model are shown in Fig. 3 (a) and (b), respectively. Fig. 3 (c) depicts the phase-space plot of this Lorenz's attractor. Any slight error in any of the initial conditions of Lorenz model would rapidly lead to an enormous error in the predictions. This situation led to the butterfly metaphor that appeared in the title of a talk by Lorenz, "Predictability: Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?" given at a meeting of the American Association for the Advancement of Science in 1972. It seems fitting that the trajectory in Fig. 3 (c) bears some resemblance to a butterfly.

2.2. Estimation of a linear model

Prediction error method is used for estimation of the linear parametric models. The parameter estimation routines require an iterative search for the minimum of the mean square error function. This search uses a special start-up procedure based on least squares and instrumental variables [22]. Least Squares minimizes the sum of squares of the right side minus the left side of the expression for AR, with respect to a Eq. (4). Singular Value Decomposition is the method of choice for solving most linear least-squares problems. Formally, the condition number of a matrix is defined as the ratio of the largest singular value (in

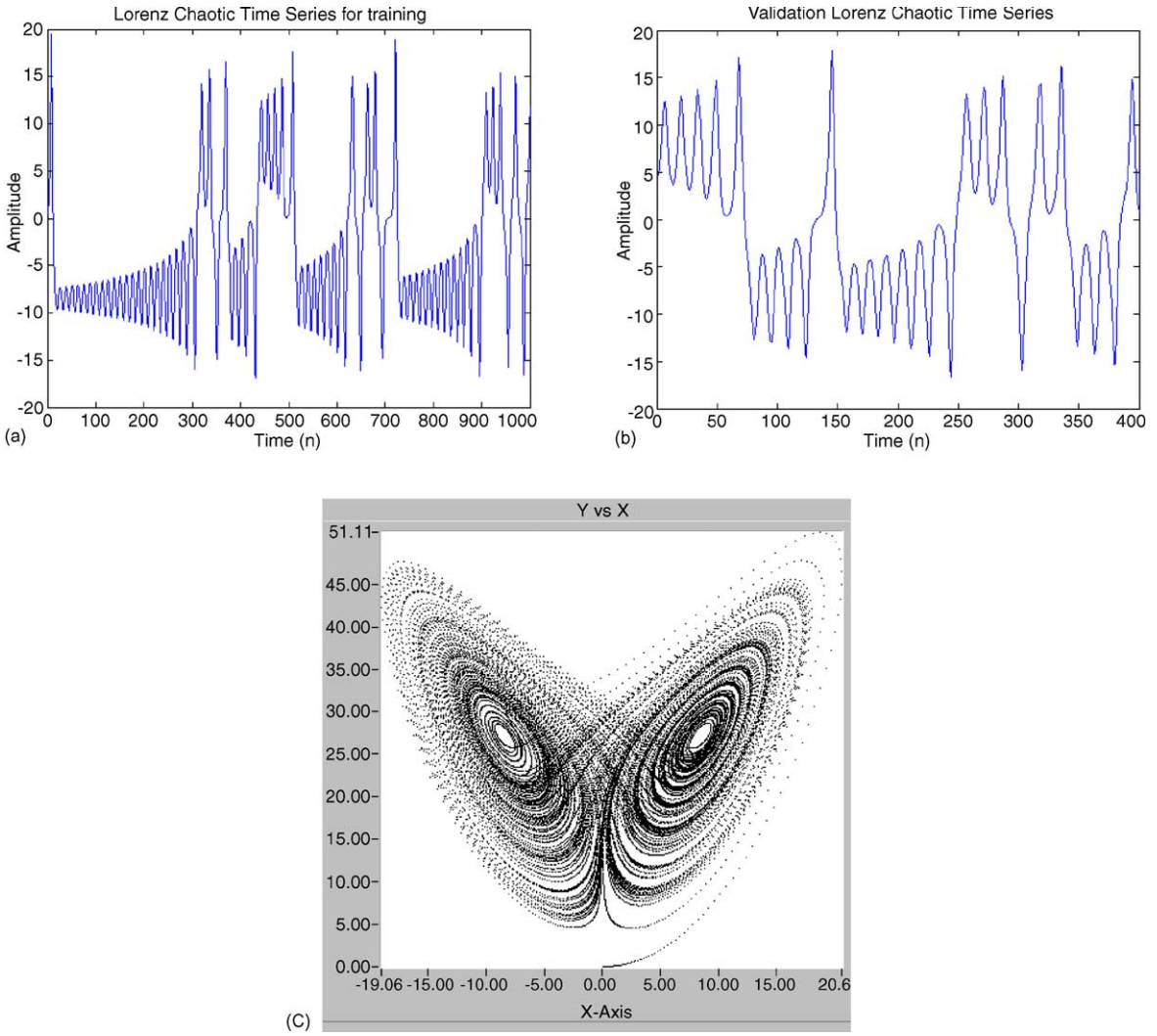

Fig. 3. (a) Lorenz chaotic time series used for training—1000 samples (1:1000) are used to estimate the model; (b) test set for Lorenz chaotic time series—400 samples (1101:1500) are used for validation; and (c) phase space plot of Lorenz's attractor.

magnitude) to the smallest one. A matrix is singular if its condition number is infinite, and it is ill-conditioned if its condition number is too large, that is if its reciprocal approaches the machine's floating-point precision. Instrumental Variables determines a so that the error between the right and left sides becomes uncorrelated with certain linear combinations of the inputs.

From the initial estimate, a Gauss–Newton minimization procedure is carried out until the norm of the Gauss–Newton direction is less than a certain tolerance. On the basis of current and previous estimates, the associated loss functions, and the Gauss–Newton update vector and its norm, the proper model order is estimated.

As a first step, a linear model will be estimated. This is always a good idea because the linear model is useful as a reference against which the performance of NN-based model structure can be compared. Models such as AR, ARMA, and state-space models will be attempted. Moving average (MA) model or the FIR (finite impulse response) model is the simplest type of model structure, where the predicted output is a function of past control inputs. However, as time series

do not have external inputs, the past inputs are simply omitted. In time series analysis, an experiment is not conducted as it is in system identification since the system cannot be influenced. Although, MA model is always stable because there is a pure algebraic relationship between prediction and past inputs that are absent in time series analysis, we have not used it.Fit given by Eq. (2) is used to evaluate the performance of model. It is expressed in percentage. It indicates the model's ability to predict.

$$\text{Fit} = \left[1 - \left\{\frac{\text{Norm}\,(y - \text{yhat})}{\text{Norm}\,(y - \text{mean}\,(y))}\right\}\right] \times 100, \qquad (2)$$

where y denotes the actual or true output, yhat denotes the estimated or predicted output, (norm) denotes the 2-norm or L_2 norm (Euclidean norm), and (mean) denotes the mean value. When y = yhat, Fit = 100%. This implies that when predicted output very closely follows the actual or true output, model is able to predict the output to the extent of 100%. Any deviation between y and yhat decreases the model's ability to predict. This is computed as the percentage of the output variation that is reproduced by the model. So, a model that has a fit of 0% gives the same mean square error as just setting the model output to be the mean of the measured output. Thus this fit function can be a good measure on the basis of which the performance of the model can be assessed. Ljung (1999) uses this fit function to express closeness of actual output with predicted output [23]. Formally, we can pick that model, for which this fit value is the lowest. In practice, it is better to be more pragmatic, and also take into account the model complexity, and whether the important features of the output response are captured.

Linear models such as AR, ARMA, and state-space models will be attempted. For real data there is no such thing as a ''correct model structure.'' However, different structures can give quite different model quality. The only way to find this out is to try out a number of different structures and compare the properties of the models obtained. Next, for each of these models, the sum of squared prediction errors is computed as they are applied to the data set. The resulting loss functions are computed with the corresponding structures. We select the structure that has the smallest loss function for the validation set. Such a procedure is known as cross-validation and is a good way to approach the model selection problem. The selected model should be properly assessed. Model validation is the process of gaining confidence in a model. Essentially this is achieved by ''twisting and turning'' the model to scrutinize all aspects of it. Of particular importance is the model's ability to reproduce the behavior of the validation data sets.

If the model is validated on the same data set from which it was estimated, the fit always improves as the flexibility of the model structure increases. We need to compensate for this automatic decrease of the loss functions. There are several approaches for this. Probably the best-known technique is Akaike's final prediction error (FPE) criterion [24,25]. It simulates the cross-validation situation, where the model is tested on another data set. Different linear models can be compared on the basis of Akaike's FPE and more distant predictions (10-step-ahead, 20-step-ahead and 50-step-ahead). In fact, mean square error on the test set; i.e., a data set not used for training, is an important quantity, since it can be viewed as an estimate of the generalization error. This should not be too large compared to the error on the training data set, in which case one must suspect that the model is over-fitting the training data. If a test set is not available, the average generalization error can be estimated from the training set alone by Akaike's FPE. The FPE is formed as

$$\text{FPE} = \left[\frac{(l + d/N)}{(l - d/N)}\right] V, \qquad (3)$$

where d is the total number of estimated parameters and N is the length of the data record. V is the loss function (quadratic fit) for the structure in question. According to Akaike's theory, in a collection of different models, we should choose the one with the smallest FPE.

An experiment is done, where the model order of AR is gradually increased from 1 to 15. The selection of a model order is based on different criteria such as the magnitude of loss function, Akaike's FPE, fit between the simulated and measured output for the validation data, etc. The following Fig. 4 (a–c) show the magnitude of the loss function versus model order, Akaike's FPE versus model order, and 1-step-ahead predictions versus model order, respectively.

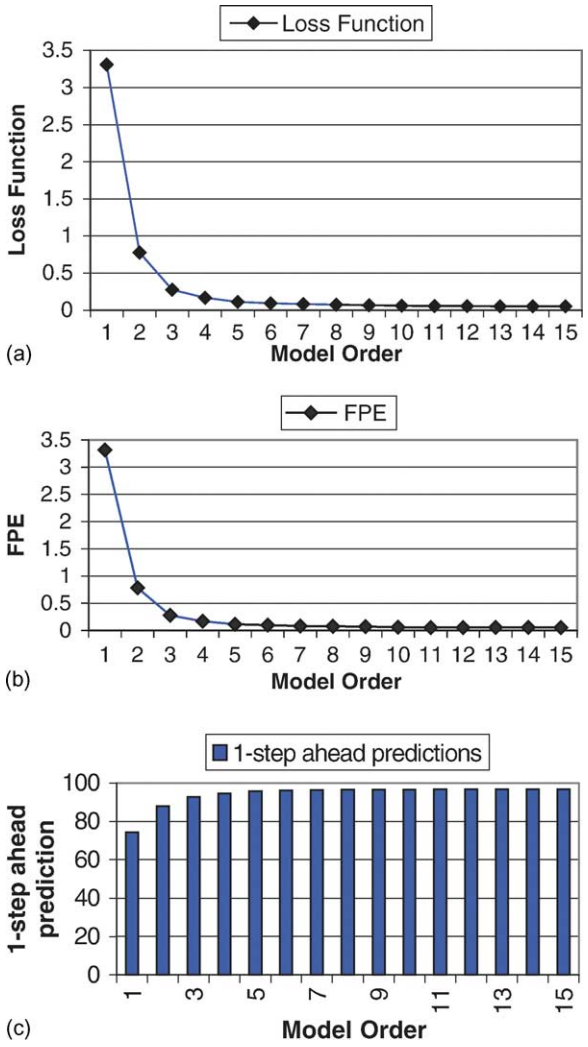

Fig. 4. Selection of an AR model structure.

Careful examination of Fig. 4 (a) and (b) shows that the magnitude of the loss function as well as the FPE decreases rapidly as the order of the AR model is gradually increased from one to eight. However, when model order is increased above eight, the rate of decrease of both the magnitude of the loss function and the FPE has become very slow. Thus, from Fig. 4 (a) and (b), it is noticed that the magnitude of the loss function as well as the FPE are reasonably low (0.0743,101 and 0.0755,087, respectively) for model order set to eight. From Fig. 4 (c), it is observed that

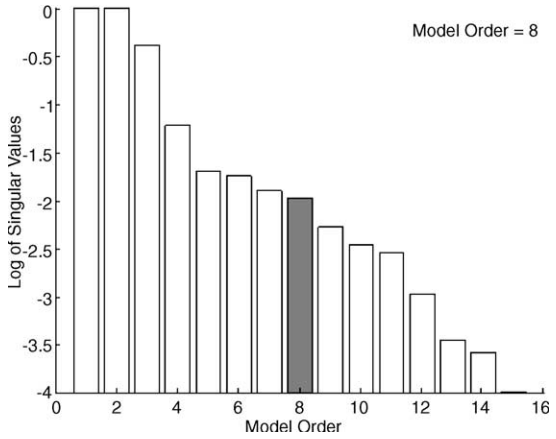

Fig. 5. Model singular values.

the 1-step-ahead prediction fit increases initially when order of the AR model is gradually increased from one to eight. However, when model order is increased beyond eight, there have not been any significant improvements in the fit values. For AR(8) model, the 1-step-ahead prediction fit is seen as 96.5%.

The order of the state space model can be selected by a subspace method. Here evaluation of condition number of the matrix can be used as an additional criterion to determine the proper model order. The Fig. 5 shows the range of singular values versus model order. It is noticed that the log of singular values has dropped from 0 to −2 (the ratio of maximum to minimum singular value decreases from 10^4 to 10^2) for a model order chosen as eight, which is reasonable. Therefore, satisfactory choice of model order should be about eight. In the noise-free case, it is sometimes possible to determine the lag space (number of past inputs and outputs) automatically with the so-called Lipschitz coefficients [26]. This method is not always equally successful but sometimes-reasonable performance is observed.

For a single signal $y(t)$, the AR model is given by

$$A(q)y(t) = e(t), \tag{4}$$

$$\text{where } A(q) = 1 + a_1 q^{-1} + a_2 q^{-2} + \cdots + a_{na} q^{-na}. \tag{5}$$

$e(t)$ indicates the noise source. Here the number n_a denotes the order of the respective polynomial. For the

estimated AR(8) model,

$$A(q) = 1 - 3.834q^{-1} + 7.179q^{-2} - 8.922q^{-3} + 8.3064q^{-4} - 6.04q^{-5} + 3.349q^{-6} - 1.271q^{-7} + 0.241q^{-8} \quad (6)$$

First AR(8) model has been estimated. Here output can be considered as a function of the 8 past output observations.

$$\hat{y}\left(\frac{t}{\theta}\right) = -a_1 y(t-l) - \cdots - a_8 y(t-8), \quad (7)$$

where θ is the parameter vector and a_i 's are the set of adjustable parameters (coefficients) of AR(8) model.

After scaling the data to zero mean and unit variance, an AR (8) model is estimated. The following Fig. 6 shows the 50-step-ahead, 20-step-ahead, 10-step-ahead predictions, and 1-step-ahead predictions. The one-step-ahead predictions appear to be accurate. However, one must be careful not to rely too heavily on a visual inspection of the one-step-ahead predictions. In order that the performance of the estimated model be properly assessed, it is better to have forecast for time horizons longer than one time step-ahead. This has been a standard approach of iterating a one-

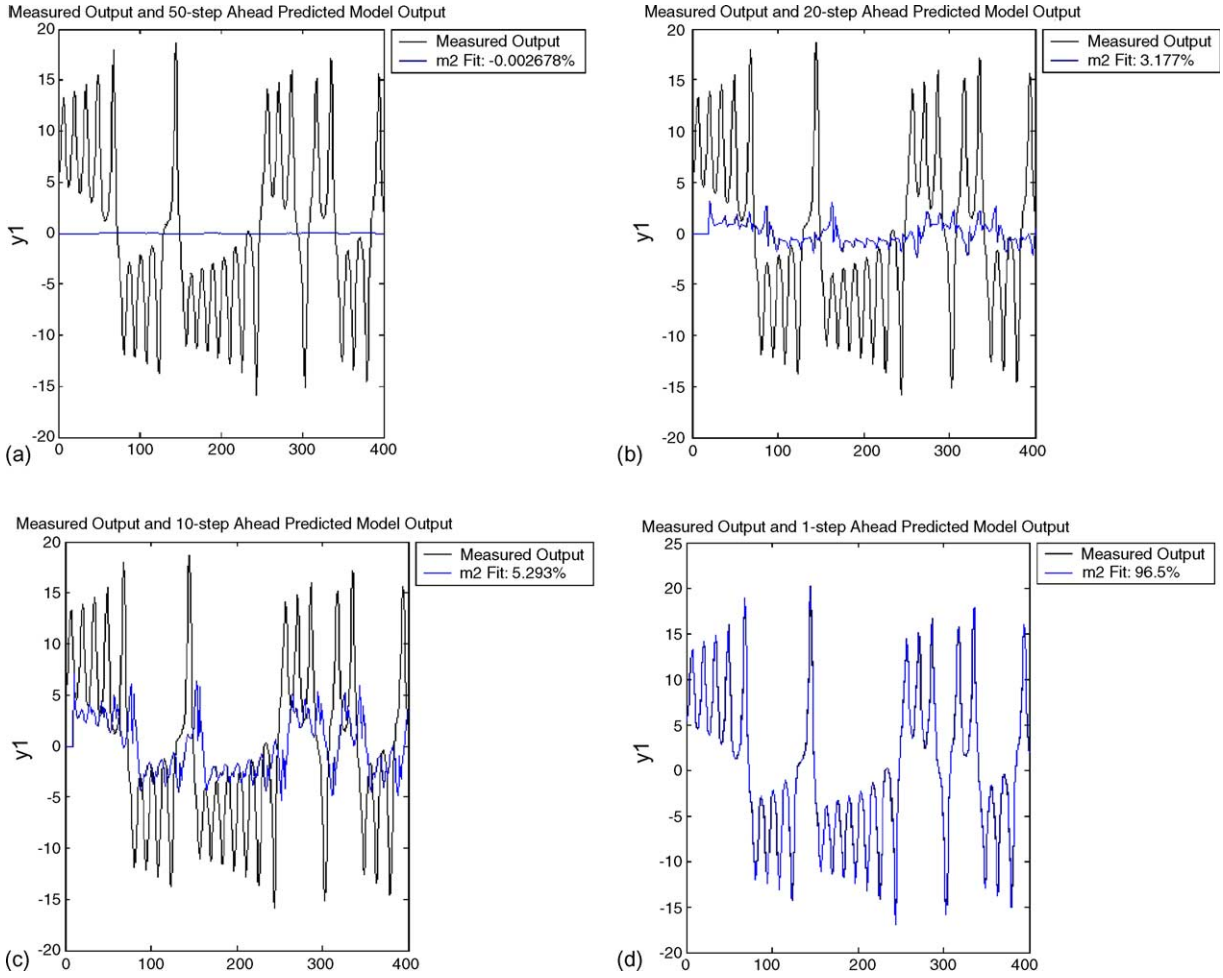

Fig. 6. (a) 50-step-ahead prediction for AR(8) model (Fit = −0.002678%); (b) 20-step-ahead prediction for AR(8) model (Fit = 3.177%); (c) 10-step-ahead prediction for AR(8) model (Fit = 5.293%); (d) 1-step-ahead prediction for AR(8) model (Fit = 96.5%).

step-ahead forecasting procedure. It is always a good idea to consider more distant predictions. Therefore for the validation of the model 50-, 20-, and 10-step-ahead predictions are considered.

Visual inspection of Fig. 6 and the associated fit values for the iterated predictions clearly depicts that linear AR (8) model is not able to make distant predictions of the Lorenz chaotic time series. In addition, this model has not learned the dynamics of the system. Therefore, this model cannot be accepted.

2.3. Other linear\models

Other linear models such as ARMA (8,1) and eighth order state-space innovations form are also estimated. ARMA(8,1) model is given by

$$\begin{aligned} A(q)y(t) = C(q)e(t) \text{ where, } A(q) &= 1 - 3.492q^{-1} \\ &+ 5.539q^{-2} - 5.401q^{-3} + 3.715q^{-4} \\ &- 1.959q^{-5} + 0.8084q^{-6} - 0.2385q^{-7} \\ &+ 0.03258q^{-8} \text{ and} \\ C(q) &= 1 + 0.8222q^{-1} \end{aligned} \tag{8}$$

$$\text{where } C(q) = 1 + c_1 q^{-1} + c_2 q^{-2} + \cdots + c_{nc} q^{-nc}. \tag{9}$$

State-space models are common representations of dynamical models. They describe the same type of linear difference relationship between the inputs and the outputs as in the ARX model, but they are rearranged so that only one delay is used in the expressions. To achieve this, some extra variables, the state variables, are introduced. They are not measured, but can be estimated from the measured input–output data. The order of the state-space model relates to the number of delayed inputs and outputs used in the corresponding linear difference equation. The state-space representation (for a time series) looks like

$$\begin{aligned} x(t + Ts) &= Ax(t) + Ke(t) \\ y(t) &= Cx(t) + e(t) \end{aligned} \tag{10}$$

Here $x(t)$ is the vector of state variables. The model order is the dimension of this vector. The matrix K determines the disturbance properties. Notice that if $K = 0$, then the noise source $e(t)$ affects only the output, and no specific model of the noise properties is built. The first value of the state variable vector $x(0)$ reflects the initial conditions for the system at the beginning of the data record. When dealing with models in state-space form, a typical option is whether to estimate K, and $x(0)$ or to let them be zero. For the estimated 8th order state-space innovations form model

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -0.24788, & 1.8001, & -6.2311, & 13.381, & -19.415, & 19.411, & -12.996, & 5.2991 \end{bmatrix} \tag{11}$$

$$C = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0];$$

$$x(t) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}; \begin{bmatrix} 4.2782 \\ 9.3617 \\ 14.379 \\ 17.727 \\ 18.851 \\ 18.066 \\ 16.06 \\ 13.504 \end{bmatrix}; x(0) = \begin{bmatrix} 4.1169 \\ 4.6045 \\ 5.4085 \\ 6.7845 \\ 9.1743 \\ 13.285 \\ 17.205 \\ 16.725 \end{bmatrix} \tag{12}$$

The results are shown in Table 1. It is seen that none of the linear models is able to learn the true trajectory of the Lorenz chaotic attractor. Although, the values of FPE appear to be reasonably low, linear models perform poorly in iterated predictions.

From Fig. 6 and Table 1, it is observed that all possible linear models are unable to make predictions, because of very low values of prediction fits and

Table 1
Comparison of different linear models

Model	Fit for predictions (%)				FPE	Loss function
	1-Step-ahead	10-Step-ahead	20-Step-ahead	50-Step-ahead		
AR(8)	96.5	5.293	3.177	−0.002678	0.0755087	0.0743101
ARMA(8,1)	96.66	3.135	3.177	−0.5621	0.0783282	0.0757068
State-space 8th order	96.68	4.32	3.177	−6.069	0.0777771	0.0741313

inability to learn the true dynamics of the chaotic time series. Although, the values of Akaike's FPE are fairly low, all linear models fail to make multi-step-ahead predictions. This is not surprising, because for the dynamic reconstruction problem, which is an ill-posed problem, low prediction error does not normally entail good model. In view of this, it is essential to generalize the solution. This can be achieved either by imposing some smoothness constraint on the prediction model or using some a priori knowledge about the process. Moreover, none of them has been able to learn the dynamics of the time series even for a short-term. These facts can also be confirmed from the visual inspection of Fig. 6. In view of this, it turns out that the linear models are not perfect and in this context, the use of a NN to build the predictive model is appropriate. In particular, the universal approximation property of a MLP means that we can take care of the issue of reconstruction accuracy by using this NN with an appropriate size. Typically, to solve a real-world problem using MLP NN, we need to train relatively large neural network architecture. Having a large number of units in the hidden layers guarantees good network performance when it is presented with input patterns that belong to the training set. However, an ''overdesigned'' architecture will tend to ''overfit'' the training data [27–29]. In view of this, we need the solution to be regularized to overcome overfitting. Regularization, in some sense, attempts to remove the superfluous weights in the overdesigned NN by imposing smoothness constraints on the weights of the NN. Consequently, generalization ability of a MLP NN can be improved. Thus, there is a need to explore the possibility of using complex models such as regularized MLP NN-based models for estimation.

3. Modelling of Lorenz chaotic attractor based on NN

Multi-layer perceptron NN-based model is used in this study, because MLP has solid theoretical foundation [30–32]. The main reason for this is its ability to model simple as well as very complex functional relationships. This has been proven through a large number of practical applications [33]. Therefore, the focus is placed on two-layer perceptron NNs with hyperbolic tangent hidden units and linear output units. In [34] it is shown that all continuous functions can be approximated to any desired accuracy, in terms of the uniform norm, with a network of one hidden layer of sigmoidal (or hyperbolic tangent) hidden units and a layer of linear output units. This paper does not explain how many units to include in the hidden layer. This issue is addressed in [6] and a significant result is derived about the approximation capabilities of two-layer perceptron networks when the function to be approximated shows certain smoothness. Unfortunately, the result is difficult to apply in practice for choosing the number of units. Just knowing that any continuous function can be approximated reasonably well by an MLP network is, however, reassuring since this covers most functions of practical interest.

The configuration of the MLP NN is determined by the number of hidden layers, number of the neurons in each of the hidden layers, as well as the type of the activation functions used for the neurons. While it has been proved that the performance of the network does not depend much on the type of the activation function (as long as it is non-linear), the choice of the number of hidden layers and the number of units or neurons in each of the hidden layers is critical. In practice, it is very difficult to determine a sufficient number of neurons necessary to achieve the desired degree of approximation accuracy. Frequently, the number of units in the hidden layer is determined by trial and error. To determine the weight values, one must have a set of examples of how the outputs should relate to the inputs. The task of determining the weights from these examples is called training or learning, and it is basically a conventional estimation problem. That is, the weights are estimated from the examples in such a

way that the network, according to some metric, models the true relationship as accurately as possible.

The biggest advantage of using the MLP NN for approximation of the mapping from input to the output of the system resides in its simplicity and the fact that it is well suited for online implementation. The class of MLP-networks considered for estimation of model is confined to those having only one hidden layer and only hyperbolic tangent and linear activation functions. The weights (specified by the vector θ, or alternatively by the matrices w and W) are the adjustable or free parameters of the network, and they are determined from a set of examples through the process called training. The examples, or the training data as they are usually called, are a set of inputs, $u(t)$ and corresponding desired outputs, $y(t)$. The training set is specified by Eq. (13).

$$Z^N = \{[u(t), y(t)] | t = 1, \ldots, N\}, \tag{13}$$

where N denotes number of samples in training data set. The objective of the training is then to determine a mapping from the set of training data to the set of possible weights given by Eq. (14):

$$Z^N \rightarrow \hat{\theta}, \tag{14}$$

so that the network will produce predictions $\hat{y}(t)$, which in some sense are "close" to the true outputs $y(t)$. $\hat{\theta}$ denotes the set of candidate models. The prediction error approach is based on the introduction of a measure of closeness in terms of a mean square error (MSE) criterion Eq. (15):

$$\begin{aligned} V_N(\theta, Z^N) &= \left(\frac{1}{2}N\right) \sum_{t=1}^{N} [y(t) - \hat{y}(t|\theta)]^{\mathrm{T}} [y(t) - \hat{y}(t|\theta)] \\ &= \left(\frac{1}{2}N\right) \sum_{t=1}^{N} \varepsilon^2(t, \theta) \end{aligned} \tag{15}$$

The weights are then found as Eq. (16):

$$\hat{\theta} = \arg \min_{\theta} V_N(\theta, Z^N) \tag{16}$$

by some kind of iterative minimization scheme Eq. (17):

$$\theta^{(i+1)} = \theta^{(i)} + \mu^{(i)} f^{(i)} \tag{17}$$

where $\theta^{(i)}$ specifies the current iterate (number "i"), $f^{(i)}$ is the search direction, and $\mu^{(i)}$ the step size.

During training the weights and biases of the network are iteratively adjusted to minimize the network performance function, i.e., the average squared error between the network outputs and the target outputs. Different training algorithms for MLP NN use the gradient of the performance function to determine how to adjust the weights to minimize performance. The gradient is determined using a technique called backpropagation, which involves performing computations backwards through the network. The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly the negative of the gradient. One iteration of this algorithm can be written as

$$\theta_{k+1} = \theta_k - \alpha_k g_k, \tag{18}$$

where θ_k is a vector of current weights and biases, g_k is the current gradient, and α_k is the learning rate.

We have used this algorithm in batch mode, where all of the inputs are applied to the network before the weights are updated. Here, the weights and biases of the network are updated only after the entire training set has been applied to the network. The gradients calculated at each training example are added together to determine the change in the weights and biases.

For fast convergence, Newton's method can be used. The basic step of this method is

$$\theta_{k+1} = \theta_k - R_k^{-1} g_k, \tag{19}$$

where R_k is the Hessian matrix (second derivatives) of the performance index at the current values of the weights and biases. Though, Newton's method converges faster, it is complex and expensive to compute the Hessian matrix for MLP NN. There is a class of algorithms that is based on Newton's method, but which does not require calculation of second derivatives. These are called quasi-Newton methods. They update an approximate Hessian matrix at each iteration of the algorithm. The update is computed as a function of the gradient.

Like the quasi-Newton methods, the Levenberg–Marquardt (LM) algorithm was designed to approach second order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical

in training MLP NN), then the Hessian matrix can be approximated by

$$\theta_{k+1} = \theta_k - [J^{\mathrm{T}}J + \lambda I]^{-1} J^{\mathrm{T}} e, \tag{20}$$

and the gradient can be computed as

$$g = J^{\mathrm{T}} e, \tag{21}$$

where I stands for identity matrix, J is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, λ is a LM parameter, and e is a vector of network errors. The Jacobian matrix can be computed through a standard backpropagation technique, that is much less complex than computing the Hessian matrix. The LM algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$\theta_{k+1} = \theta_k - [J^{\mathrm{T}}J + \lambda I]^{-1} J^{\mathrm{T}} e, \tag{22}$$

When the scalar λ is zero, this is just Newton's method, using the approximate Hessian matrix. When A, is large, this becomes gradient descent with small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift towards Newton's method as quickly as possible. Thus, λ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function is always reduced at each iteration of the algorithm. Thus, the LM method is the standard method for minimization of MSE criterion, due to its rapid convergence properties and robustness. This method is derived especially for MSE type criterion. It provides a fast convergence, it is robust, simple to implement, and it is not necessary for the user to initialize any strange design parameters. This method has been used as a training algorithm for the present case study [35].

One of the problems that occur during neural network training is called overfitting. The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situation. One method for improving network generalization is to employ an optimal configuration of a NN that is just large enough to provide an adequate fit. Unfortunately, it is difficult to know beforehand how large a NN should be for a specific application. As a result, a slightly larger configuration (more number of neurons in the hidden layer) of the NN may be selected, so that it has larger number of extra weights and biases (free parameters) than what is actually required. Unlike the difficulty faced in determining the most optimum configuration of the NN, it is relatively much simpler to choose an oversized NN with respect to larger number of neurons in the hidden layer. This large number of hidden neurons entails large number of connection weights in the NN and a possibility of overfitting (training error decreases but test error increases). To deal with this train error/test error dilemma, the performance criterion is augmented by a complexity term (regularization term) as (under the assumption that the model structure is ''large enough'' to describe the true system)

$$W_N(\theta, Z^N) = \left(\frac{1}{2}N\right) \sum_{t=1}^{N} (y(t) - \hat{y}(t|\theta))^{\mathrm{T}} (y(t) - \hat{y}(t|\theta)) + \left(\frac{1}{2}N\right) \theta^{\mathrm{T}} D \theta. \tag{23}$$

The matrix D is a diagonal matrix, which is commonly selected to $D = \alpha I$ and I denotes an identity matrix. Here α denotes the weight decay. It causes exponential decay to connection weights, i.e. the amount of decay is proportional to the corresponding connection weight. Sometimes a different weight decay is used for the input-to-hidden layer and hidden-to-output layer, respectively, or each weight is assigned with an individual weight decay parameter. It is noticed that the influence of the weight decay in general decreases as $N \to \infty$, where N denotes number of samples in the training data set; hence the bias will vanish for large data sets. Regularization enforces a bias towards zero, hence, will increase the bias error for finite data sets. In Sjoberg and Ljung (1995), the FPE estimate has been derived for networks trained according to Eq. (23) with $D = \alpha I$

$$\hat{V}_M = \left(\frac{1}{2}\right) \left[\sigma_e^2 \left(1 + \frac{P_1}{N}\right) + \gamma\right], \tag{24}$$

where $\hat{V}_M$ denotes the FPE estimate and σ_e^2 the noise

variance [36].

$$P_1 = tr\left\{R\left[R+\frac{\alpha}{N}I\right]^{-1}R\left[R+\frac{\alpha}{N}I\right]^{-1}\right\} \text{ and} \tag{25}$$

$$\gamma = \frac{\alpha^2}{N^2}\theta_0^T\left[R+\frac{\alpha}{N}I\right]^{-1}R\left[R+\frac{\alpha}{N}I\right]^{-1}\theta_0 \le \frac{\alpha}{4N}|\theta|^2, \tag{26}$$

where $tr(\,)$ stands for the trace operation, θ denotes the parameter vector, R signifies the Gauss–Newton Hessian, I represents the identity matrix and

$$R = E\{\psi(t,\theta_0)\psi^T(t,\theta_0)\} \simeq \frac{1}{N}\sum_{t=1}^{N}\psi(t,\hat{\theta})\psi^T(t,\hat{\theta}) \tag{27}$$

where $E\{\ \}$ stands for expectation operation, $\psi(t,\theta) = d\hat{y}(t/\theta)/d\theta$ the denotes the derivative of the estimated output with respect to the parameter vector (weights). Since the trace of a matrix equals the sum of its eigenvalues, p_1 can be written as

$$P_1 = \sum_{i=1}^{p}\frac{\delta_i}{(\delta_i+\alpha/N)^2}, \tag{28}$$

where δ_i specifies the ith eigenvalue of the Gauss–Newton Hessian matrix, R. The modifications of the FPE estimate for regularized models are also analyzed in [37].

If a weight θ_j is superfluous, then $V_N(\theta^* + \Delta\theta_j, Z^N) = V_N(\theta^*, Z^N)$, where θ^* specifies the minimum parameter vector. Such a weight will thus manifest itself as a singular direction in the Hessian. In the system identification problem, the Hessian should never become singular in the minimum, partly because the system typically will be affected by noise and partly because the selection of model structure is always insufficient in reality. However, in practice the selected network architecture can be so large that a subset of the weights is not very significant. This causes the Hessian to be nearly singular. Thus, each superfluous weight will lead to an eigenvalue in the Hessian that is zero. In practice, no weight will be completely superfluous, because the NN always represents an under-parametrization. The Hessian is therefore always positive definite. However, a weight of little importance will generally lead to a small eigenvalue and vice versa. This can also be seen by recognizing, $\psi(t,\theta)$, as a sensitivity matrix. If weight i is insignificant, the derivative of this particular weight will be small for all t. this has the effect that the diagonal element as well as all elements in row i (and column i) in matrix R will be small, leading to a small eigenvalue. For more significant weights the opposite can be expected. It is therefore reasonable to split the eigenvalues into two separate groups: a group of small eigenvalues that are due to weights of little importance and a group of large eigenvalues that are due to more significant weights. If it is assumed that α/N is much bigger than the smallest eigenvalues and much smaller than the biggest eigenvalues, the quantity p_1 counts the number of significant weights. Hence, p_1 is referred to as the effective number of weights in the NN. With this interpretation, and by ignoring γ, the new FPE estimate is indeed similar in structure to the FPE estimate for an unregularized criterion. In this way, using the weight decay as a tuning knob one can in essence determine the effective size of the network architecture. The challenge is then to choose the weight decay such that the average generalization error is minimized. In case of NN-based models, the average generalization error can be estimated by

$$\hat{V}_M = \frac{1}{2}\sigma_e^2\left(1+\frac{p}{N}\right) \tag{29}$$

If the network has been trained to minimize an unregularized mean square error criterion, where p denotes the total number of weights in NNs, N denotes number of training data, and σ_e denotes the noise variance. In addition to this estimate Ljung (1999) in [23] also provides an estimate of the noise variance

$$\hat{\sigma}_e^2 = 2\frac{N}{N-P}V_N(\hat{\theta},Z^N), \tag{30}$$

where $V_N(\hat{\theta},Z^N)$ denotes a MSE type of criterion Eq. (15), $\hat{\theta}$ denotes a set of candidate models, and Z^N denotes training data set When this is inserted in Eq. (29), the following formula appears (Ljung, 1987) [22]

$$\hat{V}_M = \frac{N+P}{N-P}V_N(\hat{\theta},Z^N). \tag{31}$$

Although a test set is available, the FPE estimate might still offer some valuable insights. When the regularized criterion is used, the expression gets somewhat more complex [38]

$$\hat{V}_M = \frac{N+\gamma_1}{(N+\gamma_1-2\gamma_2)}V_N(\hat{\theta},Z^N), \tag{32}$$

where

$$\gamma_1 = tr\left[R(\hat{\theta})(R(\hat{\theta}) + \frac{1}{N}D)^{-1}R(\hat{\theta})(R(\hat{\theta}) + \frac{1}{N}D)^{-1}\right] \quad (33)$$

$$\text{and } \gamma_2 = tr\left[R(\hat{\theta})R(\hat{\theta}) + \frac{1}{N}D)^{-1}\right] \quad (34)$$

where $\gamma_1 \approx \gamma_2$ specifies the so-called effective number of parameters in the network. Here D is a diagonal matrix and is most often selected as $D = \alpha I$, where α denotes the weight decay. In the above equation, $R(\hat{\theta})$ denotes the Gauss–Newton Hessian. Complete derivation is given in [38].

In reality, the training set is finite, and it is more relevant to consider what to expect of the trained NN in this situation. An estimate of the generalization error is commonly obtained by evaluating the trained NN model on a test (or validation) set; a data set, Z^T (where T denotes number of samples in the test data set), containing data that was not used for training. The average generalization error has two components: the bias error and the variance error. The bias error is the portion of the error that is due to insufficient model structure and an insufficient sample size. Bias captures the notion of systematic error for all training sets of a given size; it is that part of the error that depends only on the learning model (in case of neural networks, the architecture—or more precisely, the number of free parameters). An incorrect or inflexible model leads to high bias: in the limit, such a biased model will take no notice of the training data. Its contribution to the error can be diminished only by adjusting model complexity.

The variance error is the portion of the error that is due to the fact that the function implemented by the network obtained on a specific data set deviates from the average function. It depends on the training sample; a model that is too complex or too flexible in relation to the training data leads to large variance; in the limit, it will not generalize but simply (over) fit the data points. It is found that the bias error is decreased as more weights are added since the network will be able to describe the system more accurately. The reason for not just selecting a massive NN architecture is that one must expect the variance error to work in the opposite direction: as more weights are estimated on the same data set, the variance on the estimated weights will increase. This difficulty is often referred to as bias/variance dilemma [39–40].

The trade-off between bias and variance is such that reducing bias often increases variance and vice versa. To reduce bias, we add structure to the model, e.g. by adding hidden units; but increasing model complexity may lead to high variance. The problem, then, is to find a compromise between the conflicting requirements of bias and variance in order to minimize the total generalization error. Various techniques such as regularization and cross-validation have been proposed to balance bias and variance. Regularization by simple weight decay has been discussed in [36,38].

Regularization theory is credited to Tikhonov. This method is especially suitable for solving ill-posed problems. The basic philosophy of regularization is to stabilize the solution by means of some auxiliary nonnegative functional that embeds prior information about the solution. The most common form of prior information includes the assumption that the input–output mapping function (i.e. solution to the dynamic reconstruction problem) is smooth, because similar inputs correspond to similar outputs.

When a NN has been trained, the next step according to the procedure is to evaluate it. It is never a good idea to assess the generalization properties of a NN-based on the training data alone. Using the training data to assess the final performance quality of the network can lead to overfitting. This can be avoided by using a standard method in statistics called independent validation. This method divides the available data into a training set and a test set. The entire data set is usually randomized first. The training data are next split into two partitions; the first partition is used to update the weights in the network, and the second partition is used to assess (or validate) the training performance. The test data are then used to assess how well the network has generalized. The most common method of validation is to investigate the residuals (prediction errors) by cross-validation on a test set. The test error is an important quantity, since it can be viewed as an estimate of the generalization error. This should not be too large compared to training error, in which case, one must suspect that the network is over-fitting the training data. As a matter of fact, the visual inspection of the plot comparing predictions to the actual measurements is probably the most important validation tool. However, one has to be

careful with one-step-ahead predictions. Often, they may look very accurate, even though the estimated model is quite far from what it should be. This is particularly true when the system is sampled rapidly compared to its (fastest) dynamics. An additional check, which [23] has recommended, is to perform a more distant predictions of the model. Since it is very likely that one ends up in a ''bad'' local minimum, the network should be trained a couple of times, starting from different initial weights. Regularization has a tremendous smoothing effect on the criterion, and several of the local minima are hence often removed by this. Sometimes another regressor structure or another network architecture is tried. Neural Network Toolbox and System Identification Toolbox for MATLAB (version 6.1 release 12.1) are specifically used for obtaining results. Like linear AR model, the predictors in NN-based AR model are always stable because there is a pure algebraic relationship between prediction and past measurements (predicted output is a function of past outputs). As, linear AR model and NN-based AR model don't have any feedback connections from output to input, they are always stable. On the other hand, in NN-based ARMA model, the predicted output is a function of the past outputs as well the past prediction errors. The past prediction errors in turn depend on the model output and consequently they establish a feedback. A network model with feedback is usually referred to as a recurrent network [41]. This feedback might lead to instability in certain regimes of the network's operating range, and it can be very difficult to determine whether or not network is stable. Therefore, the recurrent NN-based ARMA models should be used cautiously, because these are not as stable as the feedforward NN-based AR models.

3.1. Modeling of Lorenz chaotic attractor using NN-based AR model

Assuming that the appropriate regressors have been selected, a NN-based AR model, that is, NNAR (8) model will now be estimated in the same way as before. An experiment has been made where a network model with eight hidden units initially is trained using the weight decay parameter, $\alpha = 0.99$. The weight decay is slowly decreased up to zero and for each value a new network is trained (with 10 different initializations of the weights) and evaluated. It is seen that the performance is quite satisfactory for $\alpha = 0.5$. In addition, it is also seen that 20-step-ahead and 50-step-ahead predictions appear to improve with regularization. For one-step-ahead and 10-step-ahead predictions, there have not been any significant gains as compared to unregularized networks. It is also seen that the test error and the Akaike's FPE estimate don't have any relevance for assessing the validation performance of the model, because we are dealing with an ill-posed problem. The NN is shown in Fig. 7. This feedforward NN uses eight neurons in its input layer, eight neurons in the hidden layer and one neuron in the output layer, so that the architecture can be represented as 8–8–1. Input to the NN consists of eight past output samples ranging from $y(t-1)$ to $y(t-8)$.

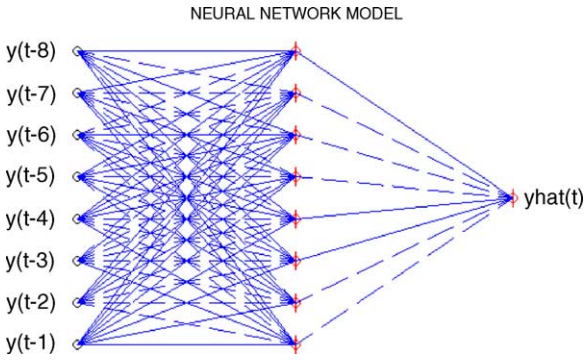

Fig. 7. NNAR(8) model.

In order to appreciate the advantages of regularization, first let us have a look towards the performance of unregularized network model. For the unregularized NN trained with 1000 iterations of LM algorithm, the plot from shown in Fig. 8 illustrates the 50-step-ahead predictions of the neuronal model.

From the above figure, it is seen that the model fails to make more distant prediction. In order to compare the performance of the various models, a fit between the actual output and the predicted output has been computed for the first 100 samples. Here 50-step-ahead prediction of the Neural Network model swings from about -150 to 150 in contrast with the actual output that varies from approximately -15 to 15. In order to fit both the variations concurrently on the same plot, the y-scale is multiplied by a factor of 10. This precisely happened because the unregularized

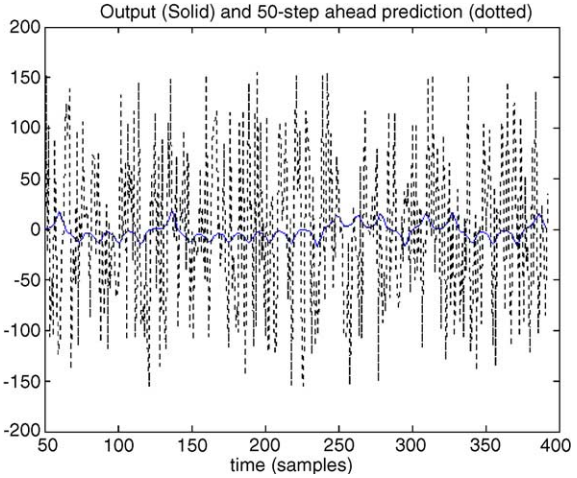

Fig. 8. Unregularized AR (8) model; Fit for the first 100 samples = −114.7%.

NNAR(8) model did fail miserably in making 50-step-ahead prediction.

Now the NNAR(8) model is trained with regularization parameter varied from 0.9 to 1e-4. It is seen that the performance is optimal for $\alpha = 0.5$. For the regularized NN trained with 1000 iterations of LM algorithm, the Fig. 9 depicts the 50-step-ahead prediction.

From this figure, it is seen that the model attempts to make more distant predictions. In addition, it has learned some of the dynamics. Various parameters of the NN-based AR model with and without regularization are shown in Table 2. Figs. 8 and 9 clearly demonstrate the practical importance of regularization. Without regularization, the solution to the dynamic reconstruction problem presented in Fig. 8 is unacceptable as it fails to approximate the true trajectory of the Lorenz attractor; the unregularized system is just a predictor. On the other hand, the solution to the dynamic reconstruction problem presented in Fig. 9 using a regularized form of NN-based AR model has learned the dynamics to some extent, in the sense that the output of the model under iterated prediction (50-step-ahead prediction) tries to reconstruct the actual trajectory of the Lorenz attractor in the short-term. However, the solution to the dynamic reconstruction problem is unacceptable as it fails to approximate the true trajectory of the Lorenz attractor.

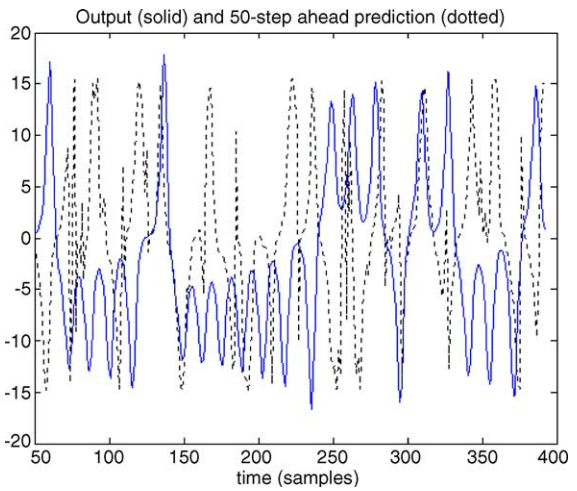

Fig. 9. Regularized NNAR(8) model, Fit for the first 100 samples = 23.396%.

From Table 2, it is seen that the regularized model has learned the dynamics, so far as multi-step-ahead predictions ($k < 20$) are concerned, in the sense that the output of the NN model closely approximates the actual trajectory of the Lorenz attractor in the short-term. It is seen that regularized form of NN-based AR model has clearly outperformed the unregularized one.

3.2. NN-based ARMA model

Now the recurrent NN-based model, NNARMA-(8,1) will be implemented as shown in Fig. 10 (a) and (b). Here past prediction error, $e(t-1)$, which is calculated as $e(t-1) = y(t-1) - \hat{y}(t-1)$ is fed back to the NN. In this way, a feedback from the output layer to the input layer is incorporated in a NN. Similarly, for NNARMA(8,2) model, two past prediction errors, $e(t-1)$ and $e(t-2)$ will be fed back to the NN. In the recurrent NN model shown in Fig. 10 (a), we see nine neurons in the input layer, eight neurons in the hidden layer and one neuron in the output layer, so that the architecture can be denoted as 9–8–1. Input to the NN is comprised of eight past output samples ranging from $y(t-1)$ to $y(t-8)$ and one past prediction error or residual, $e(t-1)$.

As before, first the network is trained in unregularized manner. Following Fig. 11 shows the 50-step-ahead predictions of this model.

Table 2
Neural network based AR (8) model, NN-based AR (8) model; architecture 8–8–1; N (number of training patterns = 1000); p (number of connection weights = 81); N/p = 12.34; number of iterations = 1000; algorithm used: Levenberg–Marquardt

α	Train error	Test error	Fit for predictions for the first 100 samples (%)				FPE
			1-Step-ahead	10-Step-ahead	20-Step-ahead	50-Step-ahead	
0.0	0.0	0.0	46.066	44.179	19.289	−114.7	0.0
0.5	0.001503	0.00934	46.21	46.18	44.557	23.396	0.0108

From this figure, it is seen that even this unregularized model elegantly makes more distant predictions. In addition, it has learned the dynamics. This performance is definitely better than the NNAR(8) model. Now the NNARMA(8,1) model is trained with regularization parameter varied from 0.9 to 10^4. The weight decay is slowly decreased up to 10^4 and for each value a new network is trained (with 10 different initializations of the weights) and evaluated. It is seen that the performance is optimal for $\alpha = 0.9$. For the regularized NN trained with 400 iterations of LM algorithm, the various plots from Figs. 12–14 illustrate the validation of the neuronal model.

Careful visual inspection of Fig. 13 reveals that the short-term predictability of the reconstructed time series (20-step-ahead prediction) is about 50 samples. Once the period of short-term predictability is over, the reconstructed time series starts to move away from the actual Lorenz attractor. Similarly, from the scrupulous examination of Fig. 14, it is noticed that the output of the NN under iterated prediction (10-step-ahead prediction) closely approximates the actual trajectory of the Lorenz attractor in the short-term. This short-term predictability of the reconstructed time series in Fig. 14 is about 100 samples. From these figures, it is seen that this regularized recurrent model elegantly makes more distant predictions like 20-step-ahead and the 10-step-ahead. In addition, it has learned the dynamics over short-term so far as more

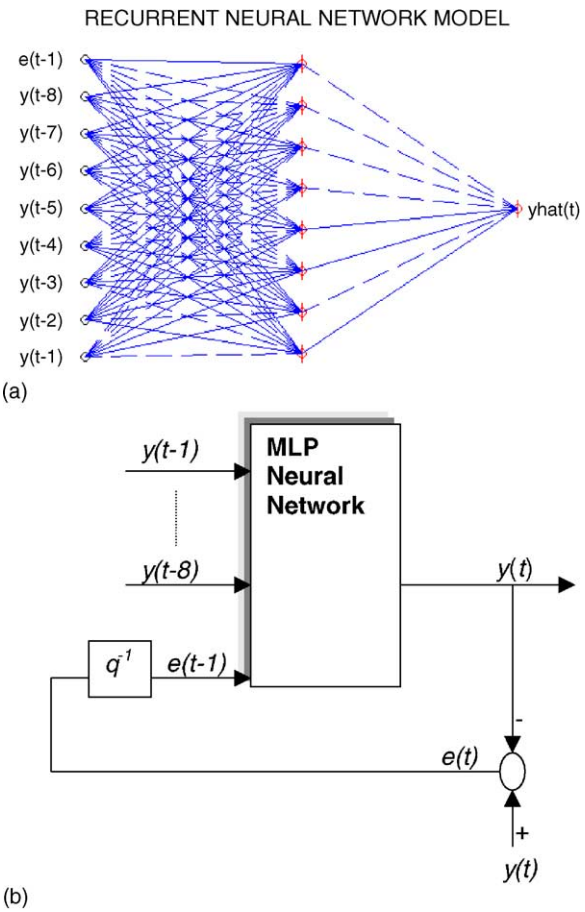

Fig. 10. (a) NNARMA(8,1) model 10; (b) NNARMA(8,1) model structure.

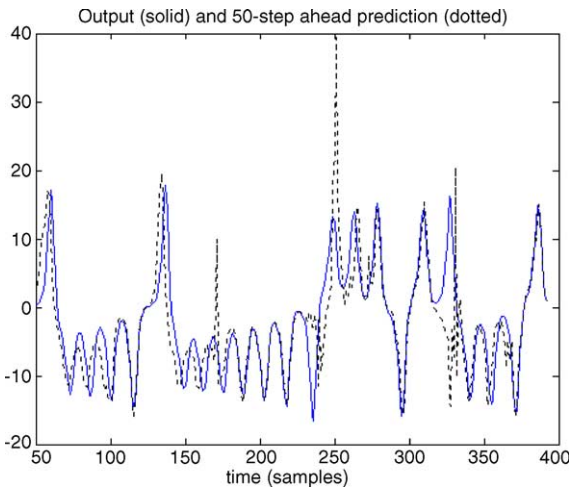

Fig. 11. Unregularized NNARMA (8,1) model; Fit for the first 100 samples = 31.78%.

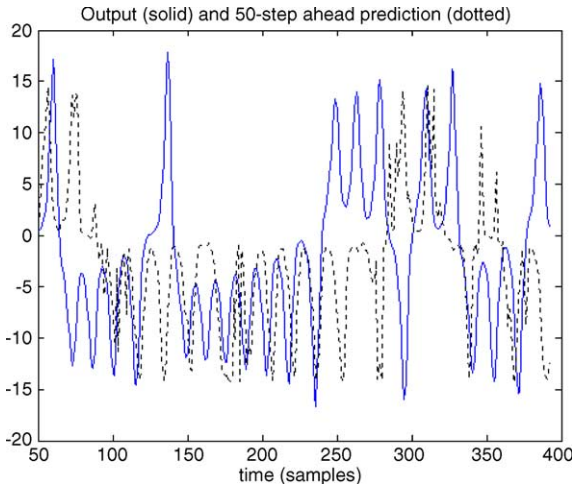

Fig. 12. Regularized NNARMA (8,1) model; Fit for the first 100 samples = 41.44%.

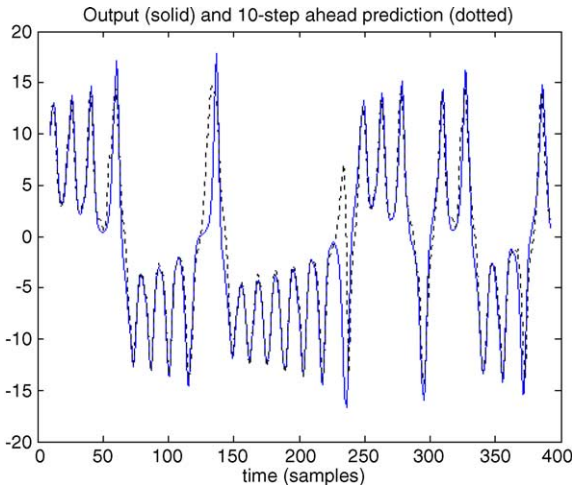

Fig. 14. Regularized NNARMA (8,1) model; Fit for the first 100 samples = 47.46%.

distant predictions are concerned. The results indicated that the performance of the proposed recurrent NN-based ARMA model is much better when predicting the near future (k = 10, 20), but it decreased dramatically for k = 50. From the visual inspection of Fig. 12, it is noticed that the output of the NN under 50-step-ahead prediction deviates from the actual trajectory of the Lorenz attractor. This could be the indication of the instability of the trained recurrent NN. Still, this performance is definitely better than the

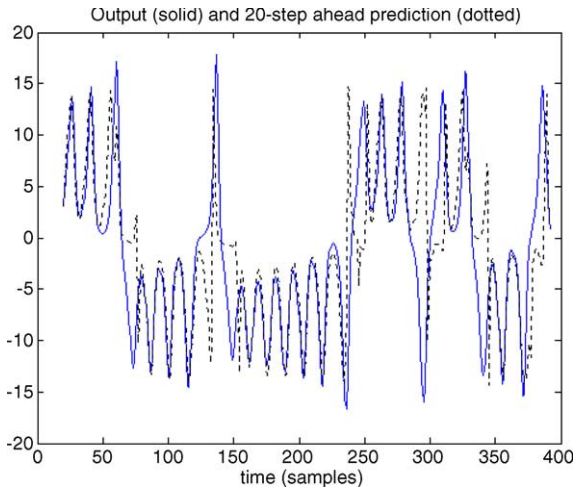

Fig. 13. Regularized NNARMA (8,1) model; Fit for the first 100 samples = 49.66%.

feedforward NN-based AR(8) model. Following Table 3 depicts important parameters of NN-based ARMA model.

It is true that the fit values for the 1-step-ahead predictions are not as high as those obtained for linear models shown in Table 1. However, this does not mean that the linear models are very accurate. If the system is sampled fast relative to its dynamical properties, the 1-step-ahead predictions will appear much accurate. Hence, it is always a good idea to consider more distant predictions. All these linear models are seen to fail not only in multi-step-ahead predictions, but also in learning underlying dynamics of the system (Lorenz chaotic attractor). In the case of more distant predictions, the proposed recurrent NN-based ARMA model with regularization by simple weight decay has certainly performed elegantly. This has been confirmed from the fit values as well as visual inspection of the multi-step-ahead prediction plots.

The test error, the estimates of average generalization error and the fit values for various iterated predictions measure the accuracy of the predictions in terms of a scalar quantity. This quantity gives an overall estimate of the accuracy of the prediction and will not generally give information about variation in accuracy among different regimes of the operating range of the system. In view of these facts, a careful visual inspection of different plots remains as an additional indispensable tool for model validation.

Table 3
Neural network based ARMA (8,1) model, NN-based ARMA (8,1) model; architecture 9–8–1; N (number of training patterns = 1000); p (number of connection weights = 89); N/p = 11.23; number of iterations = 400; algorithm used: Levenberg–Marquardt

α	Train error	Test error	Fit for predictions for the first 100 samples in %				FPE
			1-Step-ahead	10-Step-ahead	20-Step-ahead	50-Step-ahead	
0.0	0.0	0.000001	46.06	44.613	43.147	31.78	0.000003
0.9	0.0025	0.0148	46.35	47.46	49.66	41.44	0.017922

Here, one can visually detect whether the model has really learned the underlying dynamics of the system (process) or not. Thus, a very important part of the validation is to simply inspect plots comparing observed outputs to predictions.

For unregularized NN-based ARMA model, when the first row of the Table 3 for $\alpha = 0.0$ is examined, we see an obvious steady decrease in the Fit for predictions ranging from 1-step, 10-step, 20-step, and 50-step-ahead. However, when the second row for $\alpha = 0.9$ is perused, the expected trend in the fits for predictions is seen to be violated. This might be because; basically we are dealing with recurrent NN-based ARMA model armed with regularization. In the case of recurrent networks, the transient effects because of the feedback may corrupt the training of NN. These effects are due to the generally unknown initial conditions. To reduce these transient effects, although we have tried to skip the first few samples of training data set (10, 25, 50, and 100 samples) for updating weights, we could not completely eliminate the transient effects. The combination of the recurrent NN added with regularization might have led to the more pronounced transients. Perhaps, this might be the principal reason for the inconsistency of the results seen in the second row of the table. When we compare the multi-step-ahead predictions for $k = 10$, 20, and 50, it is seen that the Fit for $k = 10$ is lower than that for $k = 20$. There is hardly a difference of 2.2% in the fits for 10-step- and 20-step-ahead predictions. This is not surprising because first these fit values are just the quantitative measure of the accuracy of the estimated models and secondly a regularized recurrent NN-based ARMA model is under consideration that is not as stable as a regularized feedforward NN-based AR model. The Fit for 1-step-ahead predictions is found as 46.35%, which is smaller than that for 10-step and 20-step-ahead predictions. Here, as we have already explained, one should not heavily rely on the results of 1-step-ahead predictions. We should rather give more importance to the model's ability in multi-step-ahead predictions. As an additional check, more importantly, the meticulous visual examination of the plots comparing observed outputs to predictions is absolutely essential.

4. Conclusions

In the present paper, the problem of modeling a Lorenz chaotic attractor has been considered. The resultant time series is very rich in chaotic behavior. First, linear models like AR, ARMA, and state-space models are estimated. From the prediction performance, it is seen that none of the linear models has been able to learn the dynamics of this time series and thus, these linear models are not at all perfect. Later the proposed NN-based AR and ARMA models have been estimated with and without regularization. From the comparison of NN-based ARMA(8,1) and AR(8) models, it is observed that the recurrent NN-based model has learned the dynamics of the system with reasonable accuracy. Without even regularization, this recurrent model has performed reasonably. However, in general, without regularization, the solution to the dynamic reconstruction problem is unacceptable as it fails to approximate the true trajectory of the Lorenz attractor. It is noticed that the unregularized system is just a predictor. On the other hand, the solution to the dynamic reconstruction problem using regularized NNAR and NNARMA models have learned the dynamics, in the sense, that the output of the network closely approximates the actual trajectory of the Lorenz attractor in the short-term. It is observed that the proposed recurrent NN-based ARMA model with regularization has cleverly learned the dynamics of the chaotic attractor. However, because of the inherent complexity of the NN, the design of an appropriate NN

predictor is often a time-consuming trial-and-error procedure.

The short-term predictability of the reconstructed time series for 10-step-ahead prediction is observed as about 100 samples (visual inspection of Fig. 14). Once the period of short-term predictability is over, the reconstructed time series begins to deviate from the actual Lorenz attractor. This is basically a symptom of chaotic dynamics, namely sensitivity to initial conditions. The performance of the proposed recurrent NN-based ARMA model with regularization is much better than any other linear model or feedforward NN-based AR model (with or without regularization) when predicting the near future (horizon, $k = 10, 20$), but it is seen to decrease for larger prediction horizon like 50-step-ahead predictions ($k = 50$). This is an indication of the instability of the trained NN (an undesirable error accumulation). Yet, the proposed approach is able to predict and reconstruct the dynamics of Lorenz attractor exhibiting the rich chaotic behavior for reasonable prediction horizons.

References

[1] A.N. Tikhonov, V.Y. Arsenin, Solutions of Ill-Posed Problems., Washington, DC, W. H. Winston, 1977.

[2] O.A.S. Carpinteiro, A.J.R. Reis, A.P.A. Silva, A hierarchical neural model in short-term load forecasting, Appl. Soft Comput. 4 (4) (2004) 405–412.

[3] K. Xu, M. Xie, L.C. Tang, S.L. Ho, Applications of neural networks in forecasting engine systems reliability, Appl. Soft Comput. 2 (4) (2003) 255–268.

[4] E. Hairer, S.P. Norsett, G. Wanner, Solving ordinary differential equations I: nonstiff problems, vol. 8, Springer Series in Computational Mathematics, second ed., Springer-Verlag, 1991

[5] R.A. Cozzio, Approximation of differential equations using neural networks, in: Knowledge-Based Neurocomputing, MIT, 2002.

[6] A.R. Barron, Universal approximation bounds for superpositions of a sigmoid function, IEEE Transac. Inform. Theory 39 (1993) 930–945.

[7] A. Juditsky, H. Hjalmarson, A. Benveniste, B. Delyon, L. Ljung, J. Sjoberg, Q. Zang, Non-linear black-box models in system identification: mathematical foundations, Automatica 31 (12) (1995) 1725–1750.

[8] J. Park, I.W. Sandberg, Universal approximation using radial-basis-function networks, Neural Comput. 3 (1991) 246–257.

[9] J. Park, I.W. Sandberg, Approximation and radial-basis-function networks, Neural Comput. 5 (1993) 305–316.

[10] M.J.D. Powell, in: J.C. Mason, M.G. Cox (Eds.), Radial Basis Functions for Multivariable Interpolation: a Review in Algorithms for Approximation, Clarendon Press, Oxford, 1987.

[11] P. Niyogi, F. Girosi, On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions, Neural Comput. 8 (1996) 819–842.

[12] P.V. Yee, Regularized Radial Basis Function Networks: Theory and Applications to Probability Estimation, Classification, and Time Series Prediction, Ph. D. Thesis, McMaster University, Hamilton, Ontario, 1998.

[13] T. Poggio, F. Girosi, Networks for approximation and learning, Proceedings of the IEEE, 1990, pp. 1481–1497.

[14] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.

[15] D. Nguyen, B. Widrow, Improving the learning speed of the 2-layer neural networks by choosing initial values of adaptive weights, in: in Proceedings of the International Joint Conference on Neural Networks, San Diego, CA, 1990, pp. 21–26.

[16] H.G. Schuster, Deterministic Chaos, VCH Verlagsgesellschaft mbH, second ed., 1988.

[17] E.N. Lorenz, Deterministic non-periodic flows, J. Atm. Sci. 20 (1963) 130–141.

[18] A.M. Fraser, H.L. Swinney, Independent coordinates for strange attractors from mutual information, Phys. Rev. A 33 (1986) 1134–1140.

[19] M.B. Kennel, R. Brown, H.D.I. Abarbanel, Determining minimum embedding dimension using a geometrical construction, Phys. Rev. A 45 (1992) 3403–3411.

[20] F. Takens, On the numerical determination of the dimension of an attractor, in: D. Rand, L.S. Yyoung (Eds.), Dynamical Systems and Turbulence, Annual Notes in Mathematics, vol. 898, Springer-Verlag, Berlin, 1981, pp. 366–381.

[21] S. Haykin, Neural networks: a Comprehensive Foundation, second edition, Prentice Hall, 1998.

[22] L. Ljung, System Identification: Theory for the User. Englewood Cliffs, Prentice Hall, NJ, 1987.

[23] L. Ljung, System Identification-Theory for the User, second edition, Prentice Hall, Upper Saddle River, NJ, 1999.

[24] H. Akaike, Statistical predictor identification, Ann. Inst. Stat. Math. 22 (1970) 202–217.

[25] H. Akaike, A new look at the statistical model identification, IEEE Transac. Automatic Control, AC-19 (6) (1974) 716–723.

[26] X. He, H. Asada, A new method for identifying orders of input–output models for non-linear dynamical systems, in: Proceedings of the American Control Conference, San Francisco, California, 1993, pp. 2520–2523.

[27] F.M. Ham, Detection and classification of biological substances using infrared absorption spectroscopy and hybrid artificial network, J. Artif. Neural Networks 1 (1) (1994) 101–104.

[28] K.S. Narendra, Neural Networks for Identification and Control. Series of Lectures Presented at Yale University, December 1994.

[29] T.P. Vogl, J.K. Mangis, A.K. Zigler, W.T. Zink, D.L. Alkon, Accelerating the convergence of the backpropagation method, Biol. Cyber. 59 (1988) 257–263.

[30] K.S. Narendra, Parthasarathy K, Gradient methods for optimization of dynamic systems containing neural networks, IEEE Transac. Neural Networks 2 (1991) 252–262.
[31] K.S. Narendra, K. Parthasarathy, Identification and control of dynamic systems using neural networks, IEEE Trans. on Neural Networks. 1 (1) (1992) 4–27.
[32] S.Z. Quin, H.T. Su, T.J. McAvoy, Comparison of four neural net learning methods for dynamic system identification, IEEE Transac. Neural Networks 3 (1992) 122–130.
[33] H. Demuth, M. Beale, Neural Network Toolbox: User's Guide, version 3.0, The MathWorks, Inc., Natick, MA, 1998.
[34] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Sys. 2 (4) (1989) 303–314.
[35] D. Marquardt, An algorithm for least-squares estimation of non-linear parameters, SIAM J. Appl. Math. 11 (2) (1963) 164–168.
[36] J. Sjoberg, L. Ljung, Overtraining, regularization, and searching for minimum in neural networks, Int. J. Control 62 (6) (1995) 1391–1408.
[37] J. Moody, Note on generalization, regularization and architecture selection in nonlinear learning systems. in: B.H. Juang, S.Y. Kung, C.K., (Eds.), Proceedings of the First IEEE Workshop on Neural Networks for Signal Processing, IEEE, Piscataway, NJ, 1991, pp. 1–10
[38] J. Larsen, L. Hansen, Generalization performance of regularized neural network models. in: J. Vlontzos, J.-N. Whang, E. Wilson, (Eds.), Proceedings of the 4th IEEE Workshop on Neural Networks for Signal Processing, Piscataway, NJ, 1994, pp. 42–51.
[39] S. Gemen, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma, Neural Comput. 4 (1) (1992) 1–58.
[40] T. Heskes, Bias/variance decompositions for likelihood-based estimators, Neural Comput. 10 (6) (1998) 1425–1433.
[41] J. Hertz, A. Krogh, R.G. Palmer, An introduction to the Theory of Neural Computation, Lecture Notes, vol. I, Addison-Wesley, Redwood City, CA, 1991.