Published in: IEEE World Conference on Computational Intelligence, p. 2483, 1998. Copyright IEEE.

Neural Learning of Chaotic Dynamics: The Error Propagation Algorithm

Rembrandt Bakker, Jaap C. Schouten, Cor M. van den Bleek
Delft University of Technology,
Department of Chemical Process Technology
Julianalaan 136, 2628 BL Delft, The Netherlands
r.bakker@stm.tudelft.nl

C. Lee Giles
NEC Research Institute
4 Independence Way
Princeton, NJ 08540
giles@research.nj.nec.com

Abstract

An algorithm is introduced that trains a neural network to identify chaotic dynamics from a single measured time-series. The algorithm has four special features:

1. *The state of the system is extracted from the time-series using delays, followed by weighted Principal Component Analysis (PCA) data reduction.*
2. *The prediction model consists of both a linear model and a Multi-Layer-Perceptron (MLP).*
3. *The effective prediction horizon during training is user-adjustable, due to 'error propagation': prediction errors are partially propagated to the next time step.*
4. *To decide when to stop training, a criterion is monitored during training to select the model that has a chaotic attractor most similar to the real system's attractor.*

The algorithm is applied to laser data from the Santa Fe time-series competition (set A). The resulting model is not only useful for short-term predictions but it also generates time-series with similar chaotic characteristics as the measured data.

1. Introduction

A time-series measured from a deterministic chaotic system has the appealing characteristic that its evolution is fully determined and yet its predictability is limited, due to exponential growth of errors in model or measurements. A variety of data-driven analysis methods for this type of time-series was collected in 1991 during the Santa Fe time-series competition [1]. The methods focused either on characterization or prediction of the time-series. No attention was given to a third, much more powerful objective: given the data and the assumption that it was produced by a deterministic system, find a set of model equations that will produce a time-series with identical chaotic characteristics, having the same chaotic attractor. The model could be based on first principles if the system is well understood, but here we assume knowledge of just the time-series and use a neural network based, black-box, model. In concise neural network jargon we formulate our goal: train a network to learn the chaotic attractor. A number of authors addressed this issue [2-6]. The common approach is to identify a prediction model, generate a time-series with it and investigate its characteristics. Principe et al. [2] found that in many cases this approach fails; learning of the attractor is not guaranteed and remains a matter of trial and error. To improve the common approach, Kuo and Principe [3] proposed to minimize multi-step instead of just one-step-ahead prediction errors. We proceed in this line of thought by introducing an algorithm that effectively increases the prediction horizon of the model during training; however, it differs in that it does not compute multi-step-ahead predictions but, more efficiently, propagates previous prediction errors. The algorithm is explained in section 4.1. It is implemented along with three other features that make the modeling more effective: the reduction of the number of model inputs by weighted PCA (section 2); use of a combined linear model and MLP (section 3); and monitoring during training of a criterion to compare model and measured attractor (section 4.2). This criterion is used to decide when to stop training. Section 5 describes how the algorithm was successfully applied to laser data (set A) from the Santa Fe competition. The paper ends with concluding remarks and directions for future research (section 6).

2. Method of delays and PCA

The discrete time evolution equation for a nonlinear, autonomous, deterministic and stationary system is

$$\vec{x}_{n+1} = F(\vec{x}_n), \qquad \textbf{(1)}$$

where F is a nonlinear function, and $\vec{x}_n$ is the state of the system at time $t_n = t_0 + n\tau$, τ is the sampling time. In most real applications only a single variable is measured, even if the evolution of the system depends on several variables. In this case, we extract the state from the time-series of the single measured variable y by taking a delay vector $\vec{x}_n = (y_{n-m+1}, ..., y_n)$, where m is the number of delays, called embedding dimension. Takens [7] showed that this *method of delays* will lead to evolutions of the type of Eq. (1) if we choose $m \geq 2D + 1$, where D is the dimension of the system's attractor.

The choice of the delay time τ and the embedding m is highly empirical. First we fix their product $T=\tau m$, the time window, and consider that: (i) it is too short if the variation within the window is dominated by noise; and (ii) it is longer than necessary if the first and last part of the delay vector have no correlation at all. Next we choose the delay time τ. Here we must realize that τ is not only the time interval between successive delays but it will also be the *one-step-ahead prediction time* of our model. That time must be small to enable generation of smooth time series and to make sure that the mapping F in Eq. (1) will be smooth.

2.1. Principal Component embedding

The consequence of choosing a small delay time in a fixed time window is a large embedding, $m=T/\tau$, causing the model to have an excess number of inputs that are highly correlated. A convenient way to reduce the dimension of a set of linearly correlated variables is Principal Component Analysis (PCA), as introduced in this context by Broomhead and King [8]. PCA transforms the set of delay vectors $\vec{y}_n$ to a new set of uncorrelated variables of reduced dimension. The transformation matrix $\boldsymbol{U}$ is computed by singular value decomposition (SVD) of the covariance matrix $\boldsymbol{S}$ of the input data. SVD decomposes $\boldsymbol{S}$ into the product of three matrices $\boldsymbol{U\Sigma V}^{\mathrm{T}}$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are both orthogonal and $\boldsymbol{\Sigma}$ is diagonal. The principal component vectors $\vec{z}_n$ are computed by

$$\vec{z}_n \approx \boldsymbol{U}^T \vec{y}_n, \tag{2}$$

and the original delay vectors $\vec{y}_n$ are reconstructed by

$$\vec{y}_n \approx \boldsymbol{U} \vec{z}_n. \tag{3}$$

The merit of PCA is that not all the elements of $\vec{z}_n$ need to be used. They can be sorted in order of explained variance and often the number of elements can be much reduced while retaining an accurate reconstruction in Eq. (3). Intuitively, one would like the reconstruction of recent measurements to be more accurate than that of older ones. This can be done by using *weighted PCA* [9]. In our experience, the standard weighting profile shown in Fig. 1 works well. The curve is computed $w(d) = \exp\left(-\frac{d-1}{m-1}\right)$, where $w(d)$ is the weight for the d^{th} delay. The weight of the most recent delay is doubled; without this correction it tends to be less well reconstructed because it has only one neighbor in the delay vector.

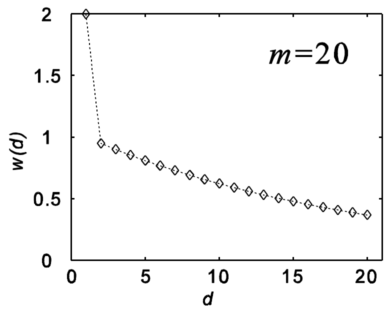

Figure 1: standard weighting profile used in the weighted PCA.

3. Prediction model

After defining the state of our experimental system, we need to choose a representation of the nonlinear function F in Eq. (1). Because we prefer to use a short delay time τ, the one-step-ahead prediction will be highly correlated with the most recent measurement, and a linear model may already make accurate one-step-ahead predictions. To learn the nonlinearities in F we *add* a standard Multi Layer Perceptron (MLP).

3.1. PCA State Update

In Eq. (1), the new state of the system is predicted from the previous one. But the method of delays has introduced information from the past in the state of the system. Since it is not sound to predict the past, we first predict only the new measured variable

$$y_{n+1} = F'(\vec{x}_n), \tag{4}$$

and then 'update' the old state $\vec{x}_n$ with the prediction y_{n+1} to give $\vec{x}_{n+1}$. For delay vectors $\vec{x}_n = (y_{n-m+1}, ..., y_n)$, this update simply involves a shift of the time window: the oldest value of y is removed and the new, predicted value comes in. In our case of principal components $\vec{x}_n = \vec{z}_n$, the update requires three steps: (i) reconstruct the delay vector from the principal components; (ii) shift the time window; and (iii) construct the principal components from the updated delay vector. Fortunately, these three operations can be done in a single matrix computation that we call 'update formula', see Bakker et al. [10] for a derivation,

$$\vec{z}_{n+1} = \boldsymbol{A} \vec{z}_n + \vec{b} y_{n+1} \tag{5}$$

with

$$A_{ij} = \sum_{k=2}^{m} U_{i,k}^{T} U_{k-1,j} \tag{6}$$

and

$$\vec{b}_i = U_{i,1}^{T}. \tag{7}$$

4. Training algorithm

The objective of our training procedure is to find a set of parameters for the model such that it will approximate the attractor of the real system. This differs from the usual goal of making good short-term predictions. The question arises what happens if we simply minimize short-term prediction errors; will the prediction model approximate the attractor of the real system? Principe et al. [2] showed in an experimental study that this is not necessarily the case, and later Kuo & Principe [3] and also Deco and Schürmann [4] proposed, as a remedy, to use *trajectory learning*, *i.e.*, minimize multi-step instead of one-step-ahead prediction errors. This approach still minimizes prediction errors (and can still fail) but it leaves more room for optimization as it makes the prediction horizon user-adjustable. The alternative for trajectory learning that we present in section 4.1 does not have the computational burden of computing multi-step-ahead predictions. The idea is to make successive predictions dependent on each other by propagating previous prediction errors. To achieve our goal of attractor learning, we extend, in section 4.2, the training procedure with a test developed by Diks et al. [11], that measures the distance between the model and real system attractor. Training can be stopped after the distance has reached its minimum.

4.1. Error Propagation

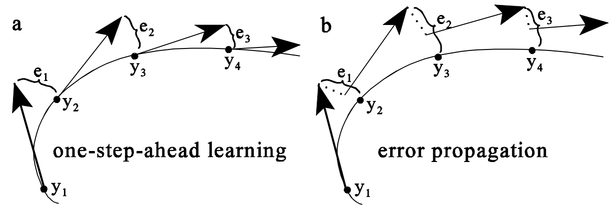

Figure 2. (a) one-step-ahead learning and (b) *error propagation*: previous errors are partially propagated to the next time step.

The common way to train a prediction model is to minimize the squared difference between the predicted and measured future value. The prediction is computed

$$\hat{y}_{n+1} = F(\vec{z}_n). \tag{8}$$

In section 3 we recommended to choose the delay time τ small. As a consequence, the one-step-ahead prediction will be highly correlated with the recent measurements and may eventually be dominated by noise, resulting in bad multi-step-ahead predictions. This motivated the aforementioned trajectory learning approach: from a large number of initial states compute predictions up to a fixed number of steps, and minimize the error of these predictions.

We envisage a second possibility to improve multi-step-ahead predictions without actually computing them, thus making the algorithm more efficient. The idea is to *partially propagate prediction errors* to the next time step, such that the next prediction is affected by the previous one. To be precise, we propose to update the next state with a mixture of the predicted and measured value

$$\vec{z}_{n+1} = A\vec{z}_n + \vec{b}[(1-\eta)y_{n+1} + \eta\hat{y}_{n+1}], \tag{9}$$

where η is the error propagation parameter that must be chosen between zero and one. Equivalently, defining the prediction error $e_n \equiv \hat{y}_{n+1} - y_{n+1}$,

$$\vec{z}_{n+1} = A\vec{z}_n + \vec{b}[y_{n+1} + \eta e_n]. \tag{10}$$

Equation (10) and the illustration in Fig. 2 make clear why we call this algorithm *error propagation* (EP). For minimization of the error we use conjugate gradients; gradients are computed with backpropagation through time.

In a preliminary version of the algorithm [10], the parameter η was taken as an additional output of the prediction model, to make it state-dependent. The idea was that the training procedure would automatically choose η large in regions of the embedding space where the predictability is high, and small where predictability is low. We now think that the training procedure converges better if we choose η fixed, not to let adaptations of η, having a big influence on the average prediction error, dominate the learning process.

4.2. Diks Test Model Selection

Error propagation training does not guarantee that we reach our goal: learning of the system attractor. We minimize prediction errors while actually we wish to find a model that has an attractor close to the attractor of the real system. To investigate the relationship between these two criteria, we monitor, during training, a test that computes the difference between reconstructed attractors. We will select the model that has low prediction error *and* a good result on this test. The test was recently developed by Diks et al. [11]. It is similar to visual comparison of attractors plotted in the embedding-space. The following null hypothesis is tested: *the two sets of delay vectors (model generated, measured) are drawn from the same multi-dimensional probability*

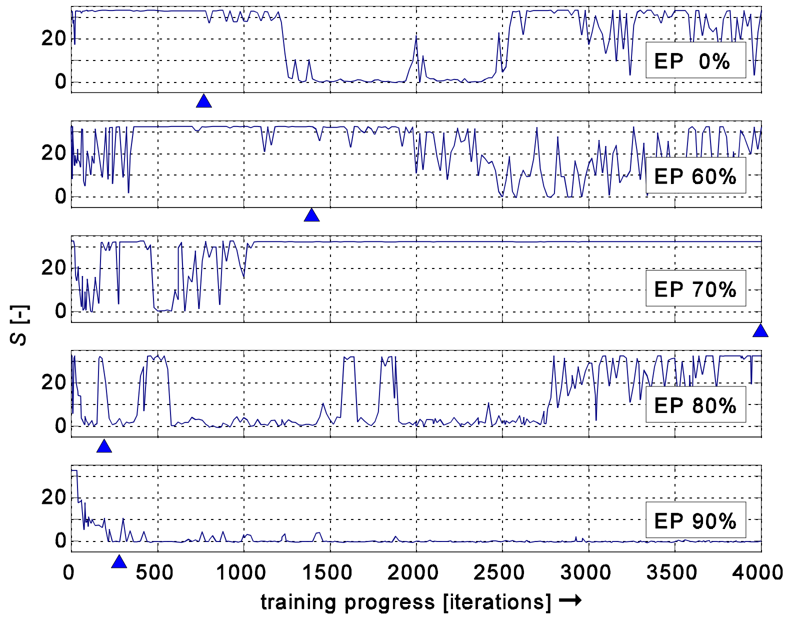

Figure 3. Diks test result (*S*) monitored during each training session. The triangles mark the point where the prediction error on the test set is at its minimum.

distribution. The test defines smoothed distributions of the two sets using Gaussian kernels and then provides an unbiased estimate for the squared difference between the distributions, $\hat{Q}$, and its variance $V_c(\hat{Q})$. Under the null hypothesis, the ratio $S = \hat{Q}/\sqrt{V_c(\hat{Q})}$ is a random variable with zero mean and unit variance. For details we refer to [11]. Special attention is needed for the choice of the bandwidth parameter that is used by the Gaussian kernels. It determines the length scale at which the two sets are compared. Diks suggests to use a small part of the available data to find out which bandwidth gives the highest test result, and to use this value for the final test with the complete sets of data. In our practical implementation we generate 11 independent datasets with the model, each with a length of two times the measured time series length. The first set is used to find the optimum bandwidth, the other ten sets to get an estimate of *S* and to roughly estimate its standard deviation (expected to be unity). Diks suggests to reject the null hypothesis with more than 95% confidence for *S*>3. Our estimate of *S* is the average of ten evaluations, and we choose a threshold of $S = 3/\sqrt{(10)} \approx 0.9$.

5. Laser data results

We applied the training algorithm to the benchmark laser data (set A) from the Santa Fe time series competition [1]. The first 3000 points were used for training, the next 3000 for testing. We took an embedding of 40 and used PCA to reduce the 40 delays to 16 principal components. First a linear model was fitted to make one-step-ahead predictions. Then the model was extended with an MLP, having 16 inputs (the 16 principal components), two hidden layers with 32 and 24 sigmoidal nodes, and a single output (the time series value to be predicted). Five separate training sessions were carried out, using 0%, 60%, 70%, 80% and 90% error propagation (the percentage corresponds to $\eta \cdot 100$). Each session lasted 4000 conjugate gradient iterations, and the Diks test was evaluated every 20 iterations. Figure 3 shows the monitoring results. We draw three conclusions from it:

1. While the prediction error on the training set (not shown) monotonically decreases during training, the Diks test shows very irregular behavior.
2. The case of 90% error propagation converges best to a model that has learnt the attractor, and does not 'forget the attractor' when training proceeds.
3. In all cases, the common stopping criterion 'stop when error on test set is at its minimum' does not yield a model with an acceptable Diks test result.

Experience has shown that the Diks test result of each training session is sensitive for the initial weights of the MLP, the curves are not well reproducible. Trial and error is required.

For further analyis we select the model trained with 90% EP after 980 iterations, having a Diks test value averaged over 100 iterations as low as -0.3. Figure 4 shows a time-series generated by this model along with measured data. For the first 3000 iterations, the model runs in 90% EP mode and follows the measured time-series. Then the mode is switched to freerun, or 100% EP, and the model generates a time-series on its own. Visual inspection confirms the results of the Diks test: at first sight, the measured and model generated series 'look the same'. Further inspection reveals that the model generated data shows small negative peaks that are not in the measured data and also the positive peaks are lower. This observation *includes* the first 3000 points of the generated series, and is therefore a direct consequence of the one-step-ahead prediction error during training in 90% EP mode. Not surprisingly, the root mean squared error on the test set is more than 5 times higher than

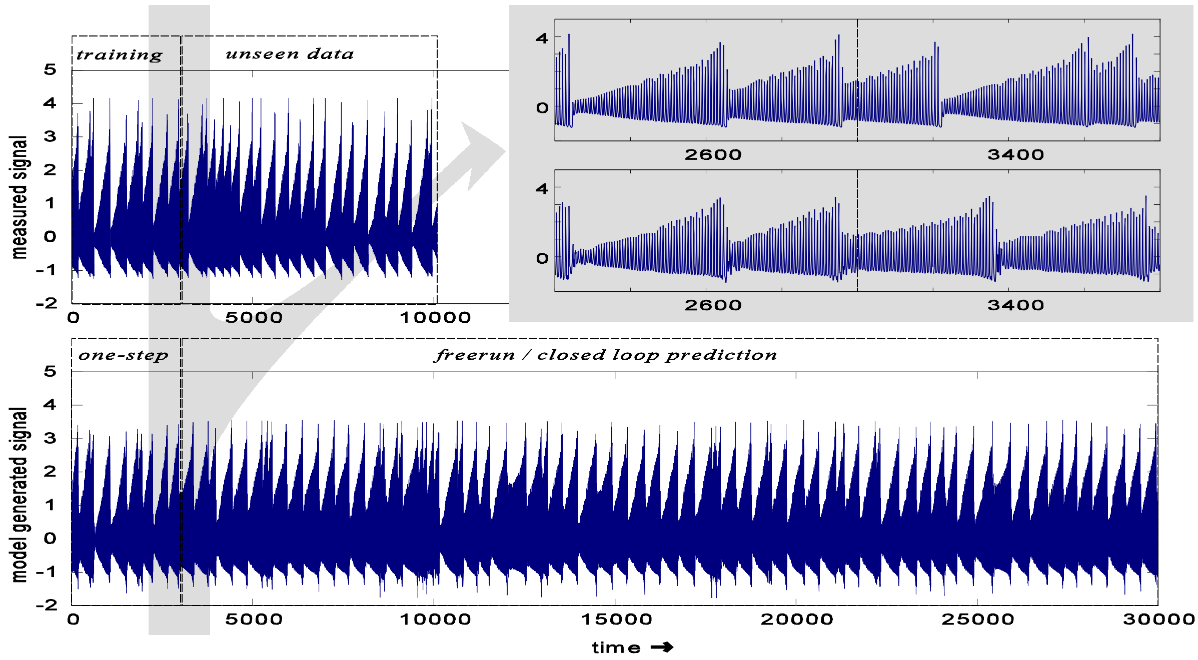

Figure 4. Comparison of measured (top) and model generated time-series (bottom). The laser data is taken from the Santa Fe time-series competition [1]. The network is trained with 90% error propagation.

that of the model without error propagation (see diagonal of Table 1); each prediction is based on information that contains accumulated previous prediction errors. It is also interesting to see what happens if we take the network *trained* with 0% EP (after 1920 iterations, where it has the lowest Diks test result) and *run* it in 90% EP mode. Table 1 shows that the Normalized Root Mean Squared (one-step-ahead prediction-) Error (NRMSE) then rises to 1.05, which is worse than predicting the mean. In other words, the 0% EP network *cannot follow the time-series* if as much as 90% of its previous prediction errors is propagated.

Table 1. NRMSE of 0% and 90% EP networks, run in 0% and 90% EP mode.

run with: \ trained with:	0% EP	90% EP
0% EP	0.07	0.27
90% EP	1.05	0.40

We computed the Lyapunov spectrum of our model using the method by Von Bremen et al. [13], using series of tangent maps along ten different model generated trajectories of length 6,000. The three largest exponents are, averaged over the ten evaluations and expressed in nats per 40 samples: 0.57, -0.06 and -0.88. The largest exponent is 20% higher than a rough estimate by Hübner et al. in [1].

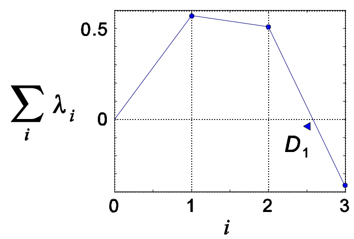

Figure 5. Sum of first *i* Lyapunov exponents vs. *i*. The dimension D_1 is estimated by the Kaplan-Yorke conjecture [12] as indicated.

The second exponent must be zero according to Haken [14]. To verify this, we estimate, from the ten evaluations, a standard deviation of the second exponent of 0.04, which is of the same order as its magnitude. From the Lyapunov spectrum, we estimate the *information dimension* D_1 of the system to be 2.6, using the Kaplan-Yorke conjecture [12]: take that (real) value of i where the interpolated curve $\sum_i \lambda_i$ vs. i crosses zero, see Fig 5. For comparison, Hübner et. al. reported an estimate of 2.06 for the correlation dimension D_2, which is a lower bound for the information dimension D_1.

Conclusion and discussion

The algorithm presented in this study has successfully learned the attractor of the system that produced the laser data from the Santa Fe time series competition. Each feature of the algorithm contributed to the result: PCA reduced the number of inputs by 40 to 16; the linear model explained 80% of the variance in the time-series before training of the MLP started; the selected model was trained with 90% error propagation; and the Diks test enabled selection of the optimum model.

The modeling procedure requires much trial and error, due to the irregular behavior of the Diks test monitoring curves that makes it hard to predict the effect of changes of model parameters (embedding dimension, number of principal components, size of the network and percentage of error propagation). Error propagation provides the experimenter with a knob to effectively control the prediction horizon during training. Unfortunately, we have no recipe yet to choose a good starting value.

The computation of the Lyapunov spectrum is only a first minor application of the model. The model can play an essential role in chaos control and synchronization, as demonstrated in the control of an experimental chaotic pendulum [15]. The selected model for the laser data has 56 sigmoidal nodes and over 1400 weights. In our experience, this kind of large size network helps training to quickly converge to an acceptable local optimum. However, in the context of chaos control, the speed of computing predictions is crucial, and the network should be as concise as possible. Therefore, a pruning technique to reduce the size of the model will be the next feature added to the algorithm. Also, pruning will improve the generalization performance of the model [16] and may enhance learning of the system attractor.

Acknowledgements

This work is supported by the Netherlands Foundation for Chemical Research (SON) with financial aid from the Netherlands Organization for Scientific Research (NWO). The authors like to thank Floris Takens for stimulating discussions and the idea to use the Diks test, Cees Diks for his support with the implementation, and Chuck Anderson for valuable comments.

References

[1] A.S. Weigend, and N.A. Gershenfeld, "Time Series Prediction: Forecasting the Future and Understanding the Past", Addison-Wesley, 1994.

[2] J.C. Principe, A. Rathie, and J.M. Kuo, "Prediction of Chaotic Time Series with Neural Networks and the Issue of Dynamic Modeling", *Int. J. Bifurcation and Chaos*, Vol. 2, 1992, pp. 989-996.

[3] J.M. Kuo, and J.C. Principe, "Reconstructed Dynamics and Chaotic Signal Modeling", In *Proc. IEEE Int'l Conf. Neural Networks*, Vol. 5, 1994, pp. 3131-3136.

[4] G. Deco, B. Schürmann, "Neural Learning of Chaotic System Behavior", *IEICE Trans. Fundamentals*, Vol. E77-A, 1994, pp. 1840-1845.

[5] R. Rico-Martínez, K. Krischer, I.G. Kevrekidis, M.C. Kube, and J.L. Hudson, "Discrete- vs. Continuous-Time Nonlinear Signal Processing Of Cu Electrodissolution Data", *Chem. Eng. Comm.*, Vol. 118, 1992, pp. 25-48.

[6] L.A. Aguirre, S.A. Billings, "Validating Identified Nonlinear Models with Chaotic Dynamics", *Int. J. Bifurcation and Chaos*, Vol. 4, 1994, pp. 109-125.

[7] F. Takens, "Detecting strange attractors in turbulence", *Lecture notes in Mathematics*, Vol. 898, 1981, pp. 365-381.

[8] D.S. Broomhead, G.P. King, "Extracting qualitative dynamics from experimental data", *Physica D*, Vol. 20, 1986, pp. 217-236.

[9] J.E. Jackson, "A User's Guide to Principal Components", Wiley, 1991.

[10] R. Bakker, R.J. de Korte, J.C. Schouten, F. Takens , and C.M. van den Bleek, "Neural Networks for Prediction and Control of Chaotic Fluidized Bed Hydrodynamics: A First Step", *Fractals*, Vol. 5, No. 3, 1997, pp. 523-530.

[11] C. Diks, W.R. van Zwet, F. Takens, and J. de Goede, "Detecting differences between delay vector distributions", *Physical Review E*, Vol. 53, 1996, pp. 2169-2176.

[12] J. Kaplan, and J. Yorke, "Chaotic Behavior of Multidimensional Difference Equations", *Lecture notes in mathematics*, Vol. 730, 1979.

[13] H.F. von Bremen, F.E. Udwadia, W. Proskurowski, "An efficient QR Based Method for the Computation of Lyapunov Exponents", *Physica D*, Vol. 101, 1997, pp. 1-16.

[14] H. Haken, "At Least One Lyapunov Exponent vanishes if the Trajectory of an Attractor does not contain a Fixed Point", *Physics Letters*, Vol. 94A, 1983, p. 71.

[15] R. Bakker, J.C. Schouten, F. Takens, and C.M. van den Bleek, "Neural network model to control an experimental chaotic pendulum", *Physical Review E*, Vol. 4-A, 1996, pp. 3545-3552.

[16] T. Lin, C.L. Giles, B.G. Horne, S.Y. Kung, "A Delay Damage Model Selection Algorithm for NARX Neural Networks", *IEEE Trans. Signal Proc., Special issue on Neural Networks*, Vol. 45, No. 11, 1997, p. 2719.