LETTER Communicated by José Principe

Learning Chaotic Attractors by Neural Networks

Rembrandt Bakker
DelftChemTech, Delft University of Technology, 2628 BL Delft, The Netherlands

Jaap C. Schouten
Chemical Reactor Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands

C. Lee Giles
NEC Research Institute, Princeton, NJ 08540, U.S.A.

Floris Takens
Department of Mathematics, University of Groningen, 9700 AV Groningen, The Netherlands

Cor M. van den Bleek
DelftChemTech, Delft University of Technology, 5600 MB Eindhoven, The Netherlands

An algorithm is introduced that trains a neural network to identify chaotic dynamics from a single measured time series. During training, the algorithm learns to short-term predict the time series. At the same time a criterion, developed by Diks, van Zwet, Takens, and de Goede (1996) is monitored that tests the hypothesis that the reconstructed attractors of model-generated and measured data are the same. Training is stopped when the prediction error is low *and* the model passes this test. Two other features of the algorithm are (1) the way the state of the system, consisting of delays from the time series, has its dimension reduced by weighted principal component analysis data reduction, and (2) the user-adjustable prediction horizon obtained by "error propagation"—partially propagating prediction errors to the next time step.

The algorithm is first applied to data from an experimental-driven chaotic pendulum, of which two of the three state variables are known. This is a comprehensive example that shows how well the Diks test can distinguish between slightly different attractors. Second, the algorithm is applied to the same problem, but now one of the two known state variables is ignored. Finally, we present a model for the laser data from the Santa Fe time-series competition (set A). It is the first model for these data that is not only useful for short-term predictions but 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 (Weigend & Gershenfeld, 1994). The methods focused on either characterization or prediction of the time series. No attention was given to a third, and much more appealing, objective: given the data and the assumption that it was produced by a deterministic chaotic 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. The choice for neural networks is inspired by the inherent stability of their iterated predictions, as opposed to for example, polynomial models (Aguirre & Billings, 1994) and local linear models (Farmer & Sidorowich, 1987). Lapedes and Farber (1987) were among the first who tried the neural network approach. In concise neural network jargon, we formulate our goal: train a network to learn the chaotic attractor. A number of authors have addressed this issue (Aguirre & Billings, 1994; Principe, Rathie, & Kuo, 1992; Kuo & Principe, 1994; Deco & Schürmann, 1994; Rico-Martínez, Krischer, Kevrekidis, Kube, & Hudson, 1992; Krischer et al., 1993; Albano, Passamente, Hediger, & Farrell, 1992). The common approach consists of two steps:

1. Identify a model that makes accurate short-term predictions.
2. Generate a long time series with the model by iterated prediction, and compare the nonlinear-dynamic characteristics of the generated time series with the original, measured time series.

Principe et al. (1992) found that in many cases, this approach fails; the model can make good short-term predictions but has not learned the chaotic attractor. The method would be greatly improved if we could minimize directly the difference between the reconstructed attractors of the model-generated and measured data rather than minimizing prediction errors. However, we cannot reconstruct the attractor without first having a prediction model. Therefore, research is focused on how to optimize both steps.

We highly reduce the chance of failure by integrating step 2 into step 1, the model identification. Rather than evaluating the model attractor *after* training, we monitor the attractor *during* training, and introduce a new test developed by Diks, van Zwet, Takens, and de Goede (1996) as a stopping criterion. It tests the null hypothesis that the reconstructed attractors of model-generated and measured data are the same. The criterion directly measures

the distance between two attractors, and a good estimate of its variance is available. We combined the stopping criterion with two special features that we found very useful: (1) an efficient state representation by weighted principal component analysis (PCA) and (2) a parameter estimation scheme based on a mixture of the output-error and equation-error method, previously introduced as the compromise method (Werbos, McAvoy, & Su, 1992). While Werbos et al. promoted the method to be used where equation error fails, we here use it to make the prediction horizon user adjustable. The method *partially* propagates errors to the next time step, controlled by a user-specified error propagation parameter.

In this article we present three successful applications of the algorithm. First, a neural network is trained on data from an experimental driven and damped pendulum. This system is known to have three state variables, of which one is measured and a second, the phase of the sinusoidal driving force, is known beforehand. This experiment is used as a comprehensive visualisation of how well the Diks test can distinguish between slightly different attractors and how its performance depends on the number of data points. Second, a model is trained on the same pendulum data, but this time the information about the phase of the driving force is completely ignored. Instead, embedding with delays and PCA is used. This experiment is a practical verification of the Takens theorem (Takens, 1981), as explained in section 3. Finally, the error propagation feature of the algorithm becomes important in the modeling of the laser data (set A) from the 1991 Santa Fe time-series prediction competition. The resulting neural network model opens new possibilities for analyzing these data because it can generate time series up to any desired length and the Jacobian of the model can be computed analytically at any time, which makes it possible to compute the Lyapunov spectrum and find periodic solutions of the model.

Section 2 gives a brief description of the two data sets that are used to explain concepts of the algorithm throughout the article. In section 3, the input-output structure of the model is defined: the state is extracted from the single measured time series using the method of delays followed by weighted PCA, and predictions are made by a combined linear and neural network model. The error propagation training is outlined in section 4, and the Diks test, used to detect whether the attractor of the model has converged to the measured one, in section 5. Section 6 contains the results of the two different pendulum models, and section 7 covers the laser data model. Section 8 concludes with remarks on attractor learning and future directions.

2 Data Sets

Two data sets are used to illustrate features of the algorithm: data from an experimental driven pendulum and far-infrared laser data from the 1991 Santa Fe time-series competition (set A). The pendulum data are described

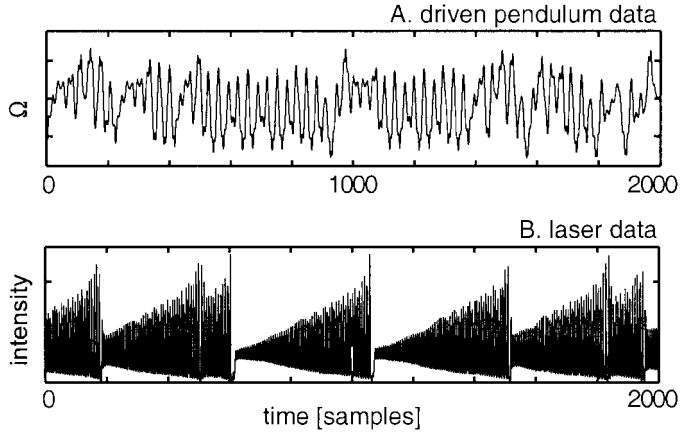

Figure 1: Plots of the first 2000 samples of (a) driven pendulum and (b) Santa Fe laser time series.

in section 2.1. For the laser data we refer to the extensive description of Hübner, Weiss, Abraham, & Tang (1994). The first 2048 points of the each set are plotted in Figures 1a and 1b.

2.1 Pendulum Data. The pendulum we use is a type EM-50 pendulum produced by Daedalon Corporation (see Blackburn, Vik, & Binruo, 1989, for details and Figure 2 for a schematic drawing). Ideally, the pendulum would

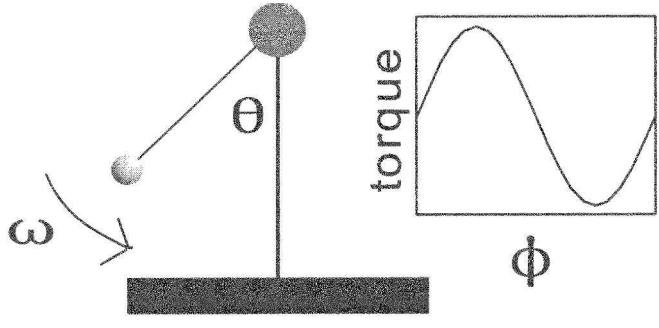

Figure 2: Schematic drawing of the experimental pendulum and its driving force. The pendulum arm can rotate around its axis; the angle θ is measured. During the measurements, the frequency, of the driving torque was 0.85 Hz.

obey the following equations of motion, in dimensionless form,

$$\frac{d}{dt}\begin{pmatrix}\theta\\ \omega\\ \phi\end{pmatrix} = \begin{pmatrix}\omega\\ -\gamma\,\omega - \sin\theta + \alpha\sin\phi\\ \omega_D\end{pmatrix}, \tag{2.1}$$

where θ is the angle of the pendulum, ω its angular velocity, γ is a damping constant, and (α, ω_D, ϕ) are the amplitude, frequency, and phase, respectively, of the harmonic driving torque. As observed by De Korte, Schouten, & van den Bleek (1995), the real pendulum deviates from the ideal behavior of equation 2.1 because of its four electromagnetic driving coils that repulse the pendulum at certain positions (top, bottom, left, right). However, from a comparison of Poincaré plots of the ideal and real pendulum by Bakker, de Korte, Schouten, Takens, & van den Bleek (1996), it appears that the real pendulum can be described by the same three state variables (θ, ω, ϕ) as the equations 2.1. In our setup the angle θ of the pendulum is measured at an accuracy of about 0.1 degree. It is not a well-behaved variable, for it is an angle defined only in the interval $[0, 2\pi]$ and it can jump from 2π to 0 in a single time step. Therefore, we construct an estimate of the angular velocity ω by differencing the angle time series and removing the discontinuities. We call this estimate Ω.

3 Model Structure

The experimental pendulum and FIR-laser are examples of autonomous, deterministic, and stationary systems. In discrete time, the evolution equation for such systems is

$$\vec{x}_{n+1} = F(\vec{x}_n), \tag{3.1}$$

where F is a nonlinear function, and $\vec{x}_n$ is the state of the system at time $t_n = t_0 + n\tau$, and 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. As Grassberger, Schreiber, and Schaffrath (1991) pointed out, it is possible to replace the vector of "true" state variables by a vector that contains delays of the single measured variable. The state vector becomes a delay vector $\vec{x}_n = (y_{n-m+1}, \ldots, y_n)$, where y is the single measured variable and m is the number of delays, called embedding dimension. Takens (1981) showed that this method of delays will lead to evolutions of the type of equation 3.1 if we choose $m \geq 2D + 1$, where D is the dimension of the system's attractor. We can now use this result in two ways:

1. Choose an embedding a priori by fixing the delay time τ and the number of delays. This results in a nonlinear auto regressive (NAR) model structure—in our context, referred to as a time-delay neural network.

2. Do not fix the embedding, but create a (neural network) model structure with recurrent connections that the network can use to learn an appropriate state representation during training.

Although the second option employs a more flexible model representation (see, for example, the FIR network model used by Wan, 1994, or the partially recurrent network of Bulsari & Saxén, 1995), we prefer to fix the state beforehand since we do not want to mix learning of two very different things: an appropriate state representation; and the approximation of F in equation 3.1. Bengio, Simard, and Frasconi (1994) showed that this mix makes it very difficult to learn long-term dependencies, while Lin, Horne, Tins, and Giles (1996) showed that this can be solved by using the NAR(X) model stucture. Finally, Siegelmann, Horne, and Giles (1997) showed that the NAR(X) model structure may be not as compact but is computationally as strong as a fully connected recurrent neural network.

3.1 Choice of Embedding. The choice of the delay time τ and the embedding m is highly empirical. First we fix the time window, with length $T = \tau(m-1)+1$. For a proper choice of T, we consider that it is too short if the variation within the delay vector is dominated by noise, and it is longer than necessary if it exceeds the maximum prediction horizon of the system—that is, if the first and last part of the delay vector have no correlation at all. Next we choose the delay time τ. Figure 3 depicts two possible choices of τ for a fixed window length. It is important to realize that τ equals the one-step-ahead prediction horizon of our model. The results of Rico-Martinez et al. (1992) show that if τ is chosen large (such that successive delays show little linear correlation), the long-term solutions of the model may be apparently different from the real system due to the very discrete nature of the model. A second argument to choose τ small is that the further we predict ahead in the future, the more nonlinear will be the mapping F in equation 3.1. A small delay time will require the lowest functional complexity.

Choosing τ (very) small introduces three potential problems:

1. Since we fixed the time window, the delay vector becomes very long, and subsequent delays will be highly correlated. This we solve by reducing its dimension with PCA (see section 3.2).

2. The one-step prediction will be highly correlated with the most recent delays, and a linear model will already make good one-step predictions (see section 3.4). We adopt the error-propagation learning algorithm (see section 4) to ensure learning of nonlinear correlations.

3. The time needed to compute predictions over a certain time interval is inversely proportional to the delay time. For real-time applications, too small a delay time will make the computation of predictions prohibitively time-consuming.

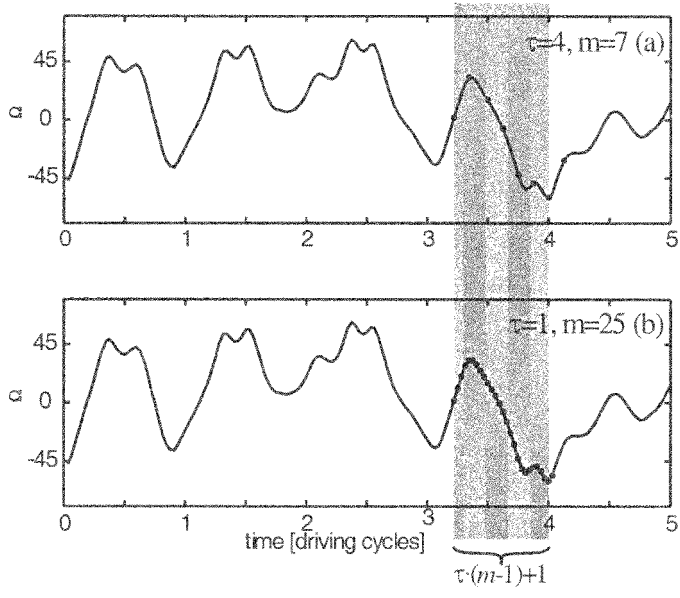

Figure 3: Two examples of an embedding with delays, applied to the pendulum time series of Figure 1a. Compared to (a), in (b) the number of delays within the fixed time window is large, there is much (linear) correlation between the delays, and the one-step prediction horizon is small.

The above considerations guided us for the pendulum to the embedding of Fig. 3b, where $\tau = 1$, and $T = m = 25$.

3.2 Principal Component Embedding. The consequence of a small delay time in a fixed time window is a large embedding dimension, $m \approx T/\tau$, causing the model to have an excess number of inputs that are highly correlated. Broomhead and King (1986) proposed a very convenient way to reduce the dimension of the delay vector. They called it singular system analysis, but it is better known as PCA. PCA transforms the set of delay vectors to a new set of linearly uncorrelated variables of reduced dimension (see Jackson, 1991, for details). The principal component vectors $\vec{z}_n$ are computed from the delay vectors $\vec{x}_n$

$$\vec{z}_n \approx \mathbf{V}^T \vec{x}_n, \tag{3.1}$$

where $\mathbf{V}$ is obtained from $\mathbf{U}\Sigma\mathbf{V}^T = X$, the singular value decomposition of the matrix $\mathbf{X}$ that contains the original delay vectors $\vec{x}_n$. An essential ingredient of PCA is to examine how much energy is lost for each element of $\vec{z}$ that is discarded. For the pendulum, it is found that the first 8 components

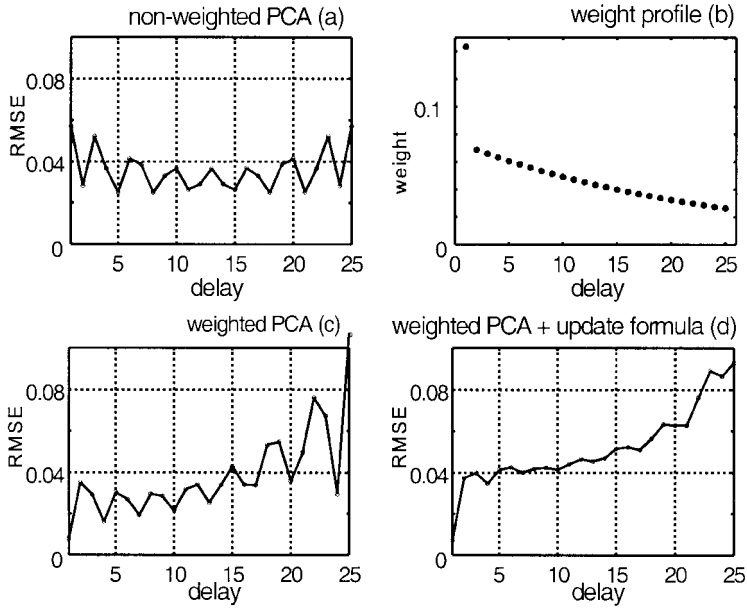

Figure 4: Accuracy (RMSE—root mean square error) of the reconstruction of the original delay vectors (25 delays) from the principal component vectors (8 principal components) for the pendulum time series: (a) nonweighted PCA; (c) weighted PCA; (d) weighted PCA and use of update formula equation 4.1. The standard profile we use for weighted PCA is shown in (b).

(out of 25) explain 99.9% of the variance of the original data. To reconstruct the original data vectors, PCA takes a weighted sum of the eigenvectors of $\mathbf{X}$ (the columns of $\mathbf{V}$). Note that King, Jones, and Broomhead (1987) found that for large embedding dimensions, principal components, are essentially Fourier coefficients and the eigenvectors are sines and cosines.

In particular for the case of time-series prediction, where recent delays are more important than older delays, it is important to know how well each individual component of the delay vector $\vec{x}_n$ can be reconstructed from $\vec{z}_n$. Figure 4a shows the average reconstruction error (RMSE, root mean squared error) for each individual delay, for the pendulum example where we use 8 principal components. It appears that the error varies with the delay, and, in particular, the most recent and the oldest measurement are less accurately represented than the others. This is not a surprise since the first and last delay have correlated neighbors only on one side. Ideally, one would like to put some more weight on recent measurements than older ones. Grassberger et al. (1991) remarked that this seems not easy when using PCA. Fortunately the contrary is true, since we can do a weighted PCA and set the desired

accuracy of each individual delay: each delay is multiplied by a weight constant. The larger this constant is, the more the delay will contribute to the total variance of the data, and the more accurately this delay will need to be represented. For the pendulum we tried the weight profile of Figure 4b. It puts a double weight on the most recent value, to compensate for the neighbor effect. The reconstruction result is shown in Figure 4c. As expected, the reconstruction of recent delays has become more accurate.

3.3 Updating Principal Components. In equation 3.1, 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,

$$y_{n+1} = F'(\vec{x}_n), \tag{3.2}$$

and then update the old state $\vec{x}_n$ with the prediction y_{n+1} to give $\vec{x}_{n+1}$. For delay vectors $\vec{x}_n = (y_{n-m+1}, \ldots, y_n)$, 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: (1) reconstruct the delay vector from the principal components, (2) shift the time window, and (3) 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, de Korte, Schouten, Takens, & van den Bleek, 1997, for a derivation):

$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_n + \vec{b}y_{n+1} \tag{3.3}$$

with

$$\mathbf{A}_{ij} = \sum_{k=2}^{m} \mathbf{V}^{-1}_{i,k}\mathbf{V}_{k-1,j}; \quad \vec{b}_i = \mathbf{V}^{-1}_{i,1}. \tag{3.4}$$

Since $\mathbf{V}$ is orthogonal, the inverse of $\mathbf{V}$ simply equals its transpose, $\mathbf{V}^{-1} = \mathbf{V}^T$. The use of the update formula, equation 3.3, introduces some extra loss of accuracy, for it computes the new state not from the original but from reconstructed variables. This has a consequence, usually small, for the average reconstruction error; compare for the pendulum example Figures 4c to 4d.

A final important issue when using principal components as neural network inputs is that the principal components are usually scaled according to the value of their eigenvalue. But the small random weights initialization for the neural networks that we will use accounts for inputs that are scaled approximately to zero mean and unit variance. This is achieved by replacing $\mathbf{V}$ by its scaled version $\mathbf{W} = \mathbf{V}\Sigma$ and its inverse by $\mathbf{W}^{-1} = (\mathbf{X}^T\mathbf{X}\mathbf{W})^T$. Note that this does not affect reconstruction errors of the PCA compression.

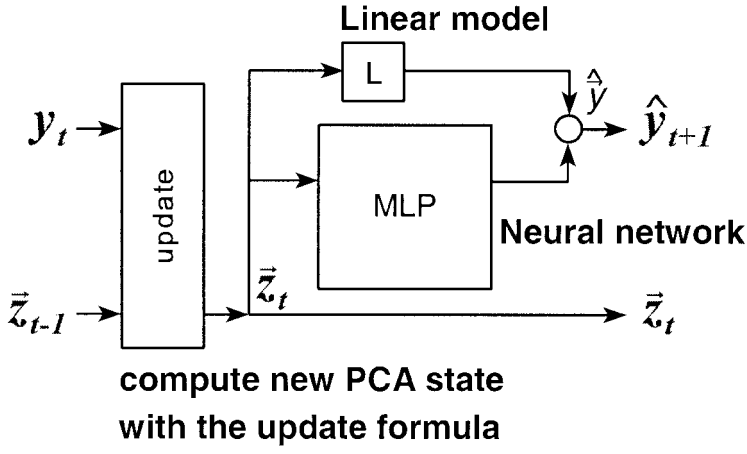

Figure 5: Structure of the prediction model. The state is updated with a new value of the measured variable (update unit), then fed into the linear model and an MLP neural network to predict the next value of the measured variable.

3.4 Prediction Model. After defining the state of our experimental system, we need to choose a representation of the nonlinear function F in equation 2.1. 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. A chaotic system is, however, inherently nonlinear and therefore we add a standard multilayer perceptron (MLP) to learn the nonlinearities. There are two reasons for using both a linear and an MLP model, even though the MLP itself can represent a linear model: (1) the linear model parameters can be estimated quickly beforehand with linear least squares, and (2) it is instructive to see at an early stage of training if the neural network is learning more than linear correlations only. Figure 5 shows a schematic representation of the prediction model, with a neural network unit, a linear model unit, and a principal component update unit.

4 Training Algorithm

After setting up the structure of the model, we need to find a set of parameters for the model such that it will have an attractor as close as possible to that of the real system. In neural network terms, train the model to learn the attractor. Recall that we would like to minimize directly the discrepancy between model and system attractor, but stick to the practical training objective: minimize short-term prediction errors. The question arises whether this is sufficient: will the prediction model also have approximately the

same attractor as the real system? Principe et al. (1992) showed in an experimental study that this is not always the case, and some years later Kuo and Principe (1994) and also Deco and Schürmann (1994) proposed, as a remedy, to minimize multistep rather than just one-step-ahead prediction errors. (See Haykin & Li, 1995, and Haykin & Principe, 1998, for an overview of these methods.) 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. In this study we introduce, in section 4.1, the compromise method (Werbos et al., 1992) as a computationally very efficient alternative for extending the prediction horizon. In section 5, we assess how best to detect differences between chaotic attractors. We will stop training when the prediction error is small *and* no differences between attractors can be detected.

4.1 Error Propagation. To train a prediction model, we minimize the squared difference between the predicted and measured future values. The prediction is computed from the principal component state $\vec{z}$ by

$$\hat{y}_{n+1} = F'(\vec{z}_n). \tag{4.1}$$

In section 3.1 we recommended choosing the delay time τ small. As a consequence, the one-step-ahead prediction will be highly correlated with the most recent measurements and eventually may be dominated by noise. This will result in bad multistep-ahead predictions. As a solution, Su, McAvoy, and Werbos (1992) proposed to train the network with an output error method, where the state is updated with previously predicted outputs instead of measured outputs. They call this a pure robust method and applied it successfully to nonautonomous, nonchaotic systems. For autonomous systems, applying an output error method would imply that one trains the model to predict the entire time series from only a single initial condition. But chaotic systems have only a limited predictability (Bockmann, 1991), and this approach must fail unless the prediction horizon is limited to just a few steps. This is the multistep-ahead learning approach of Kuo and Principe (1994), Deco and Schürmann (1994), and Jaeger and Kantz (1996): a number of initial states are selected from the measured time series, and the prediction errors over a small prediction sequence are minimized. Kuo and Principe (1994) related the length of the sequences to the largest Lyapunov exponent of the system, and Deco and Schürmann (1994) also put less weight on longer-term predictions to account for the decreased predictability. Jaeger and Kantz (1996) address the so-called error-in-variables problem: a standard least-squares fit applied to equation 4.1 introduces a bias in the estimated parameters because it does not account for the noise in the independent variables $\vec{z}_n$.

We investigate a second possibility to make the training of models for chaotic systems more robust. It recognizes that the decreasing predictability

of chaotic systems can alternatively be explained as a loss of information about the state of the system when time evolves. From this point of view, the output error method will fail to predict the entire time series from only an initial state because this single piece of information will be lost after a short prediction time—unless additional information is supplied while the prediction proceeds. Let us compare the mathematical formulation of the three methods: one-step ahead, multistep ahead, and output error with addition of information. All cases use the same prediction equation, 4.1. They differ in how they use the update formula, equation 3.3. The one-step-ahead approach updates each time with a new, measured value,

$$\vec{z}^{\,r}_{n+1} = \mathbf{A}\vec{z}^{\,r}_{n} + \vec{b}y_{n+1}. \tag{4.2}$$

The state is labeled $\vec{z}^{\,r}$ because it will be used as a reference state later in equation 4.9. Trajectory learning, or output error learning with limited prediction horizon, uses the predicted value

$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_{n} + \vec{b}\hat{y}_{n+1}. \tag{4.3}$$

For the third method, the addition of information must somehow find its source in the measured time series. We propose two ways to update the state with a mixture of the predicted and measured values. The first one is

$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_{n} + \vec{b}[(1-\eta)y_{n+1} + \eta\hat{y}_{n+1}], \tag{4.4}$$

or, equivalently,

$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_{n} + \vec{b}[y_{n+1} + \eta e_n], \tag{4.5}$$

where the prediction error e_n is defined $e_n \equiv \hat{y}_{n+1} - y_{n+1}$. Equation 4.5 and the illustrations in Figure 6 make clear why we call η the error propagation parameter: previous prediction errors are propagated to the next time step, depending on η that we must choose between zero and one. Alternatively, we can adopt the view of Werbos et al. (1992), who presented the method as a compromise between the output-error (pure-robust) and equation-error method. Prediction errors are neither completely propagated to the next time step (output-error method) nor suppressed (equation-error method) but they are partially propagated to the next time step. For $\eta = 0$, error propagation is switched off, and we get equation 4.2. For $\eta = 1$, we have full error propagation, and we get equation 4.3, which will work only if the prediction time is small. In practice, we use η as a user-adjustable parameter intended to optimize model performance by varying the effective prediction horizon. For example, for the laser data model in section 7, we did five neural network training sessions, each with a different value of η,

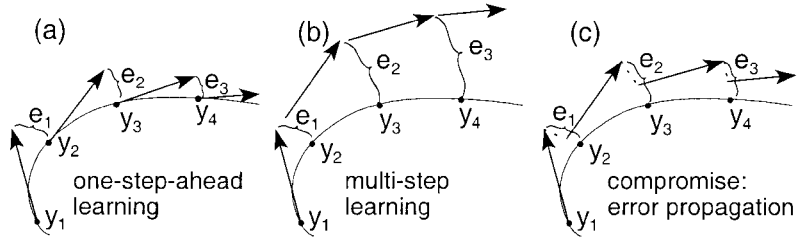

Figure 6: Three ways to make predictions: (a) one-step-ahead, (b) multistep-ahead, and (c) compromise (partially keeping the previous error in the state of the model).

ranging from 0 to 0.9. In the appendix, we analyze error propagation training applied to a simple problem: learning the chaotic tent map. There it is found that measurement errors will be suppressed if a good value for η is chosen, but that too high a value will result in a severe bias in the estimated parameter. Multistep-ahead learning has a similar disadvantage; the length of the prediction sequence should not be chosen too high.

The error propagation equation, 4.5, has a problem when applied to the linear model, which we often add before even starting to train the MLP (see section 3.4). If we replace the prediction model, 4.4, by a linear model,

$$y_{n+1} = \mathbf{C}\vec{x}_n, \tag{4.6}$$

and substitute this in equation 4.4, we get,

$$\vec{z}_{n+1} = \mathbf{A}\vec{z}_n + \vec{b}[(1-\eta)y_{n+1} + \eta\mathbf{C}\vec{z}_n] \tag{4.7}$$

or

$$\vec{z}_{n+1} = (\mathbf{A} + \eta\vec{b}\mathbf{C})\vec{z}_n + \vec{b}(1-\eta)y_{n+1}. \tag{4.8}$$

This is a linear evolution of $\vec{z}$ with y as an exogeneous input. It is BIBO (bounded input, bounded output) stable if the largest eigenvalue of the matrix $\mathbf{A} + \eta\vec{b}\mathbf{C}$ is smaller than 1. This eigenvalue is nonlinearly dependent on η, and we found many cases in practice where the evolution is stable when η is zero or one, but not for a range of in-between values, thus making error propagation training impossible for these values of η. A good remedy is to change the method into the following, improved error propagation formula,

$$\vec{z}_{n+1} = \mathbf{A}[\vec{z}^r_n + \eta(\vec{z}_n - \vec{z}^r_n)] + \vec{b}[y_{n+1} + \eta e_n], \tag{4.9}$$

where the reference state $\vec{z}_n^r$ is the state obtained without error propagation, as defined in equation 4.2. Now stability is assessed by taking the eigenvalues of $\eta(\mathbf{A} + \vec{b}\mathbf{C})$, which are η times smaller than the eigenvalues of $(\mathbf{A} + \vec{b}\mathbf{C})$, and therefore error propagation cannot disturb the stability of the system in the case of a linear model. We prefer equation 4.9 over equation 4.5, even if no linear model is added to the MLP.

4.2 Optimization and Training. For network training we use backpropagation through time (BPTT) and conjugate gradient minimization. BPTT is needed because a nonzero error propagation parameter introduces recurrency into the network. If a linear model is used parallel to the neural network, we always optimize the linear model parameters first. This makes convergence of the MLP initially slow (the easy part has already been done) and normal later on. The size of the MLPs is determined by trial and error. Instead of using "early stopping" to prevent overfitting, we use the Diks monitoring curves (see section 5) to decide when to stop: select exactly the number of iterations that yields an acceptable Diks test value *and* a low prediction error on independent test data. The monitoring curves are also used to find a suitable value for the error propagation parameter; we pick the value that yields a curve with the least irregular behavior (see the examples in sections 6 and 7). Fortunately, the extra work to optimize η replaces the search for an appropriate delay time. We choose that as small as the sampling time of the measurements.

In a preliminary version of the algorithm (Bakker et al., 1997), 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 this gives the minimization algorithm too many degrees of freedom, and we prefer simply to optimize η by hand.

5 Diks Test Monitoring

Because the error propagation training does not guarantee learning of the system attractor, we need to monitor a comparison between the model and system attractor. Several authors have used dynamic invariants such as Lyapunov exponents, entropies, and correlation dimension to compare attractors (Aguirre & Billings, 1994; Principe et al., 1992; Bakker et al., 1997; Haykin & Puthusserypady, 1997). These invariants are certainly valuable, but they do not uniquely identify the chaotic system, and their confidence bounds are hard to estimate. Diks et al. (1996) has developed a method that is unique: if the two attractors are the same, the test value will be zero for a sufficient number of samples. The method can be seen as an objective substitute for visual comparison of attractors plotted in the embedding

space. The test works for low- and high-dimensional systems, and, unlike the previously mentioned invariants, its variance can be well estimated from the measured data. The following null hypothesis is tested: *the two sets of delay vectors (model generated, measured) are drawn from the same multidimensional probability distribution*. The two distributions, $\rho_1'(\vec{r})$ and $\rho_2'(\vec{r})$, are first smoothed using gaussian kernels, resulting in $\rho_1(\vec{r})$ and $\rho_2(\vec{r})$, and then the distance between the smoothed distributions is defined to be the volume integral over the squared difference between the two distributions:

$$Q = \int d\vec{r}(\rho_1'(\vec{r}) - \rho_2'(\vec{r}))^2. \tag{5.1}$$

The smoothing is needed to make Q applicable to the fractal probability distributions encountered in the strange attractors of chaotic systems. Diks et al. (1996) provide an unbiased estimate for Q and for its variance, V_c, where subscript c means "under the condition that the null hypothesis holds." In that case, the ratio $S = \hat{Q}/\sqrt{V_c}$ is a random variable with zero mean and unit variance. Two issues need special attention: the test assumes independence among all vectors, and the gaussian kernels use a bandwidth parameter d that determines the length scale of the smoothing. In our case, the successive states of the system will be very dependent due to the short prediction time. The simplest solution is to choose a large time interval between the delay vectors—at least several embedding times—and ignore what happens in between. Diks et al. (1996) propose a method that makes more efficient use of the available data. It divides the time series in segments of length l and uses an averaging procedure such that the assumption of independent vectors can be replaced by the assumption of independent segments. The choice of bandwidth d determines the sensitivity of the test. Diks et al. (1996) suggest using a small part of the available data to find out which value of d gives the highest test value and using this value for the final test with the complete sets of data.

In our practical implementation, we compress the delay vectors to principal component vectors, for computational efficiency, knowing that

$$\|\vec{x}_1 - \vec{x}_2\|^2 \approx \|\mathbf{U}(\vec{z}_1 - \vec{z}_2)\|^2 = \|\vec{z}_1 - \vec{z}_2\|^2, \tag{5.2}$$

since $\mathbf{U}$ is orthogonal. We vary the sampling interval randomly between one and two embedding times and use $l = 4$. We generate 11 independent sets with the model, each with a length of two times the measured time-series length. The first set is used to find the optimum value for d, and the other 10 sets to get an estimate of S and to verify experimentally its standard deviation that is supposed to be unity. Diks et al. (1996) suggest rejecting the null hypothesis with more than 95% confidence for $S > 3$. Our estimate of S is the average of 10 evaluations. Each evaluation uses new and independent model-generated data, but the same measured data (only the

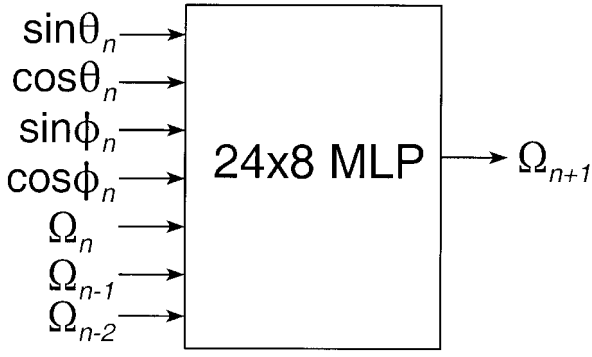

Figure 7: Structure of pendulum model I that uses all available information about the pendulum: the measured angle θ, the known phase of the driving force ϕ, the estimated angular velocity Ω, and nonlinear terms (sines and cosines) that appear in the equations of motion, equation 2.1.

sampling is done again). To account for the 10 half-independent evaluations, we lower the 95% confidence threshold to $S = 3/\sqrt{(1/2 \cdot 10)} \approx 1.4$. Note that the computational complexity of the Diks test is the same as that of computing correlation entropies or dimensions—roughly $O(N^2)$, where N is the number of delay vectors in the largest of the two sets to be compared.

6 Modeling the Experimental Pendulum

In this chapter, we present two models for the chaotic, driven, and damped pendulum that was introduced in section 2.1. The first model, pendulum model I, uses all possible relevant inputs, that is, the measured angle, the known phase of the driving force, and some nonlinear terms suggested by the equations of motion of the pendulum. This model does not use error propagation or principal component analysis; it is used just to demonstrate the performance of the Diks test. The second model, pendulum model II, uses a minimum of different inputs—just a single measured series of the estimated angular velocity Ω (defined in section 2.1). This is a practical verification of the Takens theorem (see section 3), since we disregard two of the three "true" state variables and use delays instead. We will reduce the dimension of the delay vector by PCA. There is no need to use error propagation since the data have a very low noise level.

6.1 Pendulum Model I. This model is the same as that of Bakker et al. (1996), except for the external input that is not present here. The input-output structure is shown in Figure 7. The network has 24 nodes in the first and 8 in the second hidden layer. The network is trained by backpropagation

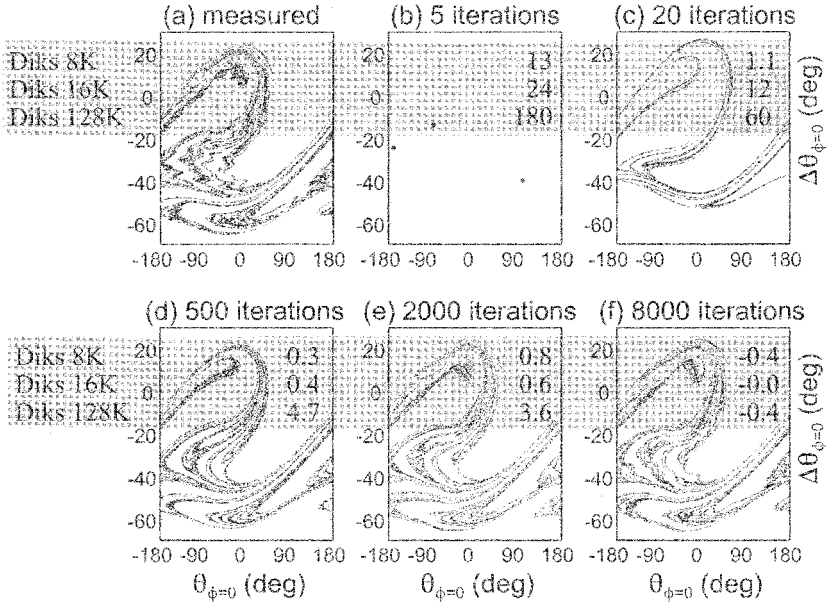

Figure 8: Development of the Poincaré plot of pendulum model *I* during training, to be compared with the plot of measured data (a). The actual development is not as smooth as suggested by (b–f), as can be seen from the Diks test monitoring curves in Figure 9.

with conjugate gradient minimization on a time series of 16,000 samples and reaches an error level for the one-step-ahead prediction of the angle θ as low as 0.27 degree on independent data. The attractor of this system is the plot of the evolutions of the state vectors $(\Omega, \theta, \varphi)_n$ in a three-dimensional space. Two-dimensional projections, called Poincaré sections, are shown in Figure 8: each time a new cycle of the sinusoidal driving force starts, when the phase φ is zero, a point $(\Omega, \theta)_n$ is plotted. Figure 8a shows measured data, while Figures 8b through 8f show how the attractor of the neural network model evolves as training proceeds. Visual inspection reveals that the Poincaré section after 8000 conjugate gradient iterations is still slightly different from the measured attractor, and it is interesting to see if we would draw the same conclusion if we use the Diks test. To enable a fair comparison with the pendulum model II results in the next section, we compute the Diks test not directly from the states $(\Omega, \theta, \varphi)_n$ (that would probably be most powerful), but instead reconstruct the attractor in a delay space of the variable Ω, the only variable that will be available in pendulum model II, and use these delay vectors to compute the Diks test. To see how the sensitivity of the test depends on the length of the time series, we compute it three times,

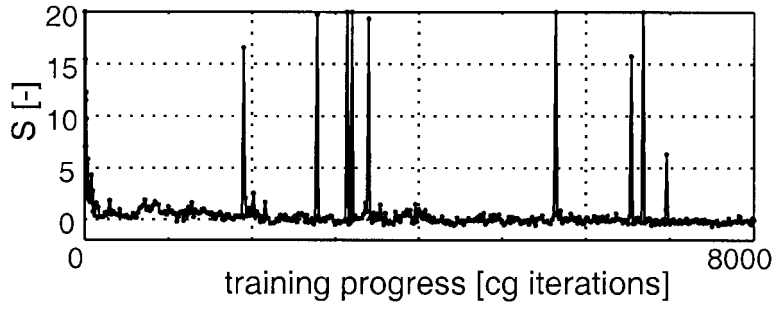

Figure 9: Diks test monitoring curve for pendulum model *I*. The Diks test converges to zero (no difference between model and measured attractor can be found), but the spikes indicate the extreme sensitivity of the model attractor for the model parameters.

using 1, 2, and 16 data sets of 8000 points each. These numbers should be compared to the length of the time series used to create the Poincaré sections: 1.5 million points, of which each 32nd point (when the phase φ is 0) is plotted. The Diks test results are printed in the top-right corner of each of the graphs in Figure 8. The test based on only one set of 8000 (Diks 8K) points cannot see the difference between Figures 8a and 8c, whereas the test based on all 16 sets (Diks 128K) can still distinguish between Figures 8a and 8e.

It may seem from the Poincaré sections in Figures 8b through 8f that the model attractor gradually converges toward the measured attractor when training proceeds. However, from the Diks test monitoring curve in Figure 9, we can conclude that this is not at all the case. The spikes in the curve indicate that even after the model attractor gets very close to the measured attractor, a complete mismatch may occur from one iteration to the other. We will see a similar behavior for the pendulum model II (see section 6.2) and the laser data model (see section 7), and address the issue in the discussion (see section 8).

6.2 Pendulum Model II. This model uses information from just a single variable, the estimated angular velocity Ω. From the equations of motion, equation 2.1, it follows that the pendulum has three state variables, and the Takens theorem tells us that in the case of noise-free data, an embedding of $7(2*3+1)$ delays will be sufficient. In our practical application, we take the embedding of Figure 3b with 25 delays, which are reduced to 8 principal components by weighted PCA. The network has 24 nodes in the first and 8 in the second hidden layer, and a linear model is added. First, the network is trained for 4000 iterations on a train set of 8000 points; then a second set of 8000 points is added, and training proceeds for another 2000 iterations,

reaching an error level of 0.74 degree, two and a half times larger than model I. The Diks test monitoring curve (not shown) has the same irregular behavior as for pendulum model I. In between the spikes, it eventually varies between one and two. Compare this to model I after 2000 iterations, as shown in Figure 8e, which has a Diks test of 0.6: model II does not learn the attractor as well as model I. That is the price we pay for omitting a priori information about the phase of the driving force and the nonlinear terms from the equations of motion.

We also computed Lyapunov exponents, λ. For model I, we recall the estimated largest exponent of 2.5 bits per driving cycle, as found in Bakker et al. (1996). For model II, we computed a largest exponent of 2.4 bits per cycle, using the method of von Bremen et al. (1997) that uses QR-factorizations of tangent maps along model-generated trajectories. The method actually computes the entire Lyapunov spectrum, and from that we can estimate the information dimension D_1 of the system, using the Kaplan-Yorke conjecture (Kaplan & Yorke, 1979): take that (real) value of i where the interpolated curve $\sum_i \lambda_i$ versus i crosses zero. This happens at $D_1 = 3.5$, with an estimated standard deviation of 0.1. For comparison, the ideal chaotic pendulum, as defined by equation 2.1, has three exponents, and as mentioned in Bakker et al. (1996), the largest is positive, the second zero, and the sum of all three is negative. This tells us that the dimension must lie between 2 and 3. This means that pendulum model II has a higher dimension than the real system. We think this is because of the high embedding space (eight principal components) that is used. Apparently the model has not learned to make the five extra, spurious, Lyapunov exponents more negative than the smallest exponent in the real system. Eckmann, Kamphorst, Ruelle, and Ciliberto (1986) report the same problem when they model chaotic data with local linear models. We also computed an estimate for the correlation dimension and entropy of the time series with the computer package RRCHAOS (Schouten & van den Bleek, 1993), which uses the maximum likeliehood estimation procedures of Schouten, Takens, and van den Bleek (1994a, 1994b). For the measured data, we found a correlation dimension of $3.1+/-0.1$, and for the model-generated data $2.9+/-0.1$: there was no significant difference between the two numbers, but the numbers are very close to the theoretical maximum of three. For the correlation entropy, we found $2.4+/-0.1$ bits per cycle for measured data, and $2.2+/-0.2$ bits per cycle for the model-generated data. Again there was no significant difference and, moreover, the entropy is almost the same as the estimated largest Lyapunov exponent.

For pendulum model II we relied on the Diks test to compare attractors. However, even if we do not have the phase of the driving force available as we did in model I, we can still make plots of cross-sections of the attractor by projecting them in a two-dimensional space. This is what we do in Figure 10: every time when the first state variable upwardly crosses zero, we plot the remaining seven state variables in a two-dimensional projection.

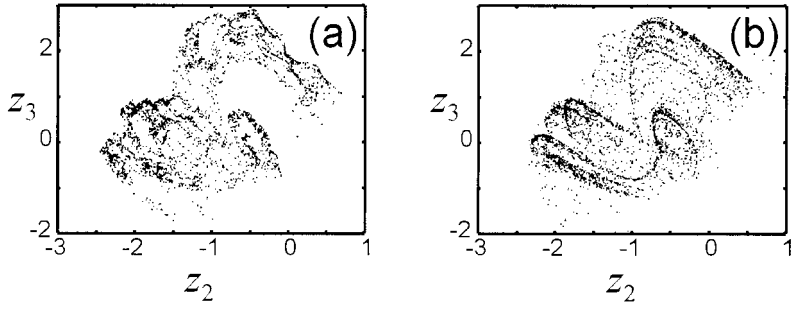

Figure 10: Poincaré section based on measured (a) and model-generated (b) data. A series of truncated states $(z_1, z_2, z_3)^T$ is reconstructed from the time series; a point $(z_2, z_3)^T$ is plotted every time z_1 crosses zero (upward).

7 Laser Data Model

The second application of the training algorithm is the well-known laser data set (set A) from the Santa Fe time series competition (Weigend & Gershenfeld, 1994). The model is similar to that reported in Bakker, Schouten, Giles, & van den Bleek (1998), but instead of using the first 3000 points for training, we use only 1000 points—exactly the amount of data that was available *before* the results of the time-series competition were published. 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. This already explained 80% of the output data variance. 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 11 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. We saw the same irregular behavior for the two pendulum models.

2. The case of 70% error propagation converges best to a model that has learned the attractor and does not "forget the attractor" when training proceeds. The error propagation helps to suppress the noise that is present in the time series.

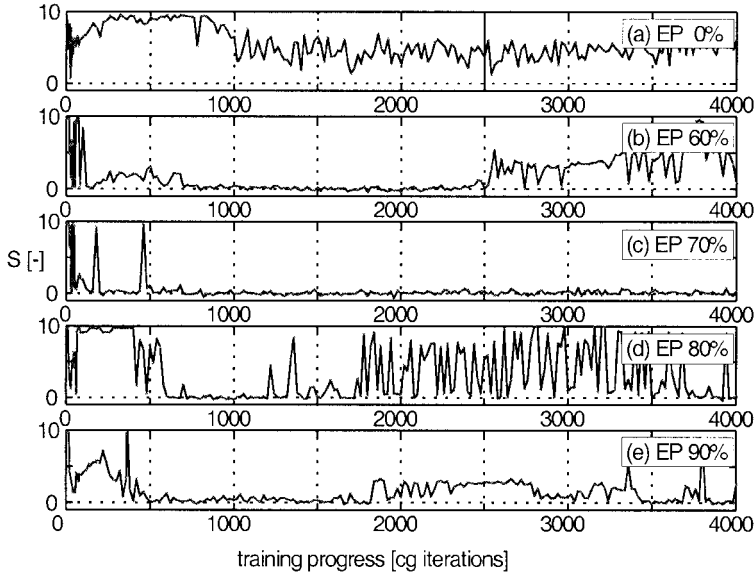

Figure 11: Diks test monitoring curves for the laser data model with (a) 0%, (b) 60%, (c) 70%, (d) 80%, (e) 90% error propagation. On the basis of these curves, it is decided to select the model trained with 70% error propagation and after 1300 iterations for further analysis.

3. The common stopping criterion, "stop when error on test set is at its minimum," may yield a model with an unacceptable Diks test result. Instead, we have to monitor the Diks test carefully and select exactly that model that has both a low prediction error and a good attractor—a low Diks test value that does not change from one iteration to the other.

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 example, in Bakker et al. (1998) where we used more training data to model the laser, we selected 90% error propagation (EP) to be the best choice by looking at the Diks test monitoring curves.

For further analysis, we select the model trained with 70% EP after 1300 iterations, having a Diks test value averaged over 100 iterations as low as -0.2. Figure 12b shows a time series generated by this model along with measured data. For the first 1000 iterations, the model runs in 90% EP mode and follows the measured time series. Then the mode is switched to free run (closed-loop prediction), or 100% EP, and the model gener-

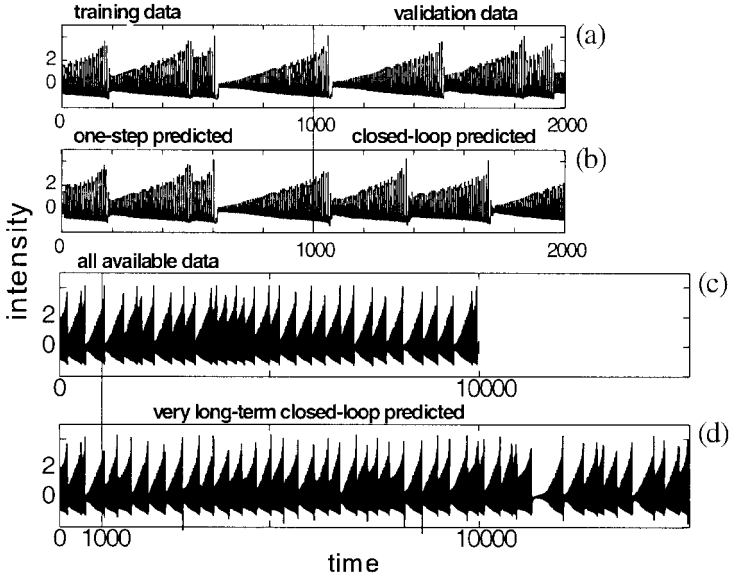

Figure 12: Comparison of time series (c) measured from the laser and (d) closed-loop (free run) predicted by the model. Plots (a) and (b) show the first 2000 points of (c) and (d), respectively.

ates a time-series on its own. In Figure 12d the closed-loop prediction is extended over a much longer time horizon. Visual inspection of the time series and the Poincaré plots of Figures 13a and 13b confirms the results of the Diks test: the measured and model-generated series look the same, and it is very surprising that the neural network learned the

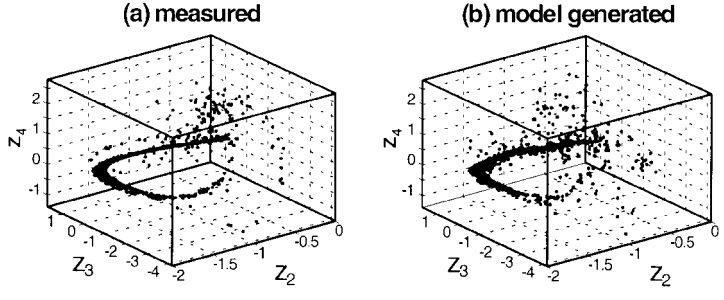

Figure 13: Poincaré plots of (a) measured and (b) model-generated laser data. A series of truncated states $(z_1, z_2, z_3, z_4)^T$ is reconstructed from the time series; a point $(z_2, z_3, z_4)^T$ is plotted every time z_1 crosses zero (upward).

chaotic dynamics from 1000 points that contain only three "rise-and-collapse" cycles.

We computed the Lyapunov spectrum of the model, using series of tangent maps along 10 different model generated trajectories of length 6000. The three largest exponents are 0.92, 0.09, and −1.39 bits per 40 samples. The largest exponent is larger than the rough estimate by Hübner et al. (1994) of 0.7. The second exponent must be zero according to Haken (1983). To verify this, we estimate, from 10 evaluations, a standard deviation of the second exponent of 0.1, which is of the same order as its magnitude. From the Kaplan-Yorke conjecture, we estimate an information dimension D_1 of 2.7, not significantly different from the value of 2.6 that we found in another model for the same data (Bakker et al., 1998).

Finally, we compare the short-term prediction error of the model to that of the winner of the Santa Fe time-series competition (see Wan, 1994). Over the first 50 points, our normalized mean squared error (NMSE) is 0.0068 (winner: 0.0061); over the first 100 it is 0.2159 (winner: 0.0273). Given that the long-term behavior of our model closely resembles the measured behavior, we want to stress that the larger short-term prediction error does not imply that the model is "worse" than Wan's model. The system is known to be very unpredictable at the signal collapses, having a number of possible continuations. The model may follow one continuation, while a very slight deviation may cause the "true" continuation to be another.

8 Summary and Discussion

We have presented an algorithm to train a neural network to learn chaotic dynamics, based on a single measured time series. The task of the algorithm is to create a global nonlinear model that has the same behavior as the system that produced the measured time series. The common approach consists of two steps: (1) train a model to short-term predict the time series, and (2) analyze the dynamics of the long-term (closed-loop) predictions of this model and decide to accept or reject it. We demonstrate that this approach is likely to fail, and our objective is to reduce the chance of failure greatly. We recognize the need for an efficient state representation, by using weighted PCA; an extended prediction horizon, by adopting the compromise method of Werbos, McAvoy, & Su, (1992); and (3) a stopping criterion that selects exactly that model that makes accurate short-term predictions *and* matches the measured attractor. For this criterion we propose to use the Diks test (Diks et al., 1996) that compares two attractors by looking at the distribution of delay vectors in the reconstructed embedding space. We use an experimental chaotic pendulum to demonstrate features of the algorithm and then benchmark it by modeling the well-known laser data from the Santa Fe time-series competition. We succeeded in creating a model that produces a time series with the same typical irregular rise-and-collapse behavior as the measured data, learned

from a 1000-sample data set that contains only three rise-and-collapse sequences.

A major difficulty with training a global nonlinear model to reproduce chaotic dynamics is that the long-term generated (closed-loop) prediction of the model is extremely sensitive for the model parameters. We show that the model attractor *does not* gradually converge toward the measured attractor when training proceeds. Even after the model attractor is getting close to the measured attractor, a complete mismatch may occur from one training iteration to the other, as can be seen from the spikes in the Diks test monitoring curves of all three models (see Figures 9 and 11). We think of two possible causes. First, the sensitivity of the attractor for the model parameters is a phenomenon inherent in chaotic systems (see, for example, the bifurcation plots in Aguirre & Billings, 1994, Fig. 5). Also the presence of noise in the dynamics of the experimental system may have a significant influence on the appearance of its attractor. A second explanation is that the global nonlinear model does not know that the data that it is trained on are actually on an attractor. Imagine, for example, that we have a two-dimensional state-space and the training data set is moving around in this space following a circular orbit. This orbit may be the attractor of the underlying system, but it might as well be an unstable periodic orbit of a completely different underlying system. For the short-term prediction error, it does not matter if the neural network model converges to the first or the second system, but in the latter case, its attractor will be wrong! An interesting development is the work of Principe, Wang, and Motter (1998), who use self-organizing maps to make a partitioning of the input space. It is a first step toward a model that makes accurate short-term predictions and learns the outlines of the (chaotic) attractor. During the review process of this manuscript, such an approach was developed by Bakker et al. (2000).

Appendix: On the Optimum Amount of Error Propagation: Tent Map Example

When using the compromise method, we need to choose a value for the error propagation parameter such that the influence of noise on the resulting model is minimized. We analyze the case of a simple chaotic system, the tent map, with noise added to both the dynamics and the available measurements. The evolution of the tent map is

$$x_t = a|x_{t-1} + u_t| + c, \tag{A.1}$$

with $a = 2$ and $c = 1/2$, and where u_t is the added dynamical noise, assumed to be a purely random process with zero mean and variance σ_u^2. It is convenient for the analysis to convert equation A.1 to the following

notation,

$$x_t = a_t(x_{t-1} + u_t) + c, \begin{cases} a_t = a, & (x_{t-1} + u_t) < 0 \\ a_t = -a, & (x_{t-1} + u_t) \geq 0 \end{cases} . \tag{A.2}$$

We have a set of noisy measurements available,

$$y_t = x_t + m_t, \tag{A.3}$$

where m_t is a purely random process, with zero mean and variance σ_m^2.

The model we will try to fit is the following:

$$z_t = b|z_{t-1}| + c, \tag{A.4}$$

where z is the (one-dimensional) state of the model and b is the only parameter to be fitted. In the alternate notation,

$$z_t = b_t z_{t-1} + c, \begin{cases} b_t = b, & z_{t-1} < 0 \\ B_t = -b, & z_{t-1} \geq 0 \end{cases} . \tag{A.5}$$

The prediction error is defined as

$$e_t = z_t - y_t, \tag{A.6}$$

and during training we minimize SSE, the sum of squares of this error. If, during training, the model is run in error propagation (EP) mode, each new prediction partially depends on previous prediction errors according to

$$z_t = b_t(y_{t-1} + \eta e_{t-1}) + c, \tag{A.7}$$

where η is the error propagation parameter. We will now derive how the expected SSE depends on η, σ_u^2 and σ_m^2. First eliminate z from the prediction error by substituting equation A.7 in A.6,

$$e_t = b_t(y_{t-1} + \eta e_{t-1}) + c - y_t, \tag{A.8}$$

and use equation A.3 to eliminate y_t and A.2 to eliminate x_t:

$$e_t = \{(b_t - a_t)x_{t-1} - a_t u_t + b_t m_{t-1} - m_t\} + b_t \eta e_{t-1}. \tag{A.9}$$

Introduce $\epsilon_t = \{(b_t - a_t)x_{t-1} - a_t u_t + b_t(1-\eta)m_{t-1}\}$, and assume $e_0 = 0$. Then the variance σ_e^2 of e_t, defined as the expectation $E[e_t^2]$, can be expanded:

$$\sigma_e^2 = E[-m_t + \epsilon_t + (b_t\eta)\epsilon_{t-1} + \cdots + (b_2\eta)^{t-1}\epsilon_1]^2. \tag{A.10}$$

For the tent map, it can be verified that there is no autocorrelation in x (for typical orbits that have a unit density function on the interval $[-0.5,0.5]$ (see Ott, 1993), under the assumption that u is small enough not to change this property. Because m_t and u_t are purely random processes, there is no autocorrelation in m or u, and no correlation between m and u, x, and m. Equation A.2 suggests a correlation between x and u, but this cancels out because the multiplier a_t jumps between -1 and 1, independent from u under the previous assumption that u is small compared to x. Therefore, the expansion A.10 becomes a sum of the squares of all its individual elements without cross products:

$$\sigma_e^2 = E[m_t^2] + E[\epsilon_t^2 + (b_t\eta)^2\epsilon_{t-1}^2 + \cdots + (b_2\eta)^{2(t-1)}\epsilon_1^2]. \tag{A.11}$$

From the definition of a_t in equation A.2, it follows that $a_t^2 = a^2$. A similar argument holds for b_t, and if we assume that at each time t, both the arguments $x_{t-1} + u_t$ in equation A.2 and $y_{t-1} + \eta e_{t-1}$ in A.7 have the same sign (are in the same leg of the tent map), we can also write

$$(b_t - a_t)^2 = (b - a)^2. \tag{A.12}$$

Since $(b_t\eta)^2 = (b\eta)^2$, the following solution applies for the series A.11:

$$\sigma_e^2 = \sigma_m^2 + \sigma_\epsilon^2 \left(\frac{1 - (b\eta)^{2t}}{1 - (b\eta)^2} \right), \quad |b\eta| \neq 1, \tag{A.13}$$

where σ_ϵ^2 is the variance of ϵ_t. Written in full,

$$\sigma_e^2 = \sigma_m^2 + ((b-a)^2\sigma_x^2 + \sigma_u^2 + b^2(1-\eta)^2\sigma_m^2) \left(\frac{1-(b\eta)^{2t}}{1-(b\eta)^2} \right), \ |b\eta| \neq 1, \tag{A.14}$$

and for large t,

$$\sigma_e^2 = \sigma_m^2 + ((b-a)^2\sigma_x^2 + \sigma_u^2 + b^2(1-\eta)^2\sigma_m^2) \left(\frac{1}{1-(\eta b)^2} \right), \ |\eta b| \neq 1. \tag{A.15}$$

Ideally we would like the difference between the model and real-system parameter, $(b-a)^2\sigma_x^2$, to dominate the cost function that is minimized. We can draw three important conclusions from equation A.15:

1. The contribution of measurement noise m_t is reduced when ηb approaches 1. This justifies the use of error propagation for time series contaminated by measurement noise.
2. The contribution of dynamical noise u_t is not lowered by error propagation.

3. During training, the minimization routine will keep b well below $1/\eta$, because otherwise the multiplier, $\frac{1}{1-(\eta b)^2}$, would become large and so would the cost function. Therefore, b will be biased toward zero if ηa is close to unity.

The above analysis is almost the same for the trajectory learning algorithm (see section 4.1). In that case, $\eta = 1$ and equation A.14 becomes

$$\sigma_e^2 = \sigma_m^2 + ((b-a)^2\sigma_x^2 + \sigma_u^2)\left(\frac{1-b^{2t}}{1-b^2}\right), \quad |b| \neq 1. \tag{A.16}$$

Again, measurement noise can be suppressed by choosing t large. This leads to a bias toward zero of b because for a given $t > 0$, the multiplier, $\frac{1-b^{2t}}{1-b^2}$, is reduced by lowering b (assuming $b > 1$, a necessary condition to have a chaotic system).

Acknowledgments

This work is supported by the Netherlands Foundation for Chemical Research (SON) with financial aid from the Netherlands Organization for Scientific Research (NWO). We thank Cees Diks for his help with the implementation of the test for attractor comparison. We also thank the two anonymous referees who helped to improve the article greatly.

References

Aguirre, L. A., & Billings, S. A. (1994). Validating identified nonlinear models with chaotic dynamics. *Int. J. Bifurcation and Chaos, 4*, 109–125.

Albano, A. M., Passamante, A., Hediger, T., & Farrell, M. E. (1992). Using neural nets to look for chaos. *Physica D, 58*, 1–9.

Bakker, R., de Korte, R. J., Schouten, J. C., Takens, F., & van den Bleek, C. M. (1997). Neural networks for prediction and control of chaotic fluidized bed hydrodynamics: A first step. *Fractals, 5*, 523–530.

Bakker, R., Schouten, J. C., Takens, F., & van den Bleek, C. M. (1996). Neural network model to control an experimental chaotic pendulum. *Physical Review E, 54A*, 3545–3552.

Bakker, R., Schouten, J. C., Coppens, M.-O., Takens, F., Giles, C. L., & van den Bleek, C. M. (2000). Robust learning of chaotic attractors. In S. A. Solla, T. K. Leen, & K.-R. Müller (Eds.), *Advances in neural information processing systems, 12*. Cambridge, MA: MIT Press (pp. 879–885).

Bakker, R., Schouten, J. C., Giles, C. L., & van den Bleek, C. M. (1998). Neural learning of chaotic dynamics—The error propagation algorithm. In *proceedings of the 1998 IEEE World Congress on Computational Intelligence* (pp. 2483–2488).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks, 5*, 157–166.

Blackburn, J. A., Vik, S., & Binruo, W. (1989). Driven pendulum for studying chaos. *Rev. Sci. Instrum., 60,* 422–426.

Bockman, S. F. (1991). Parameter estimation for chaotic systems. In *Proceedings of the 1991 American Control Conference* (pp. 1770–1775).

Broomhead, D. S., & King, G. P. (1986). Extracting qualitative dynamics from experimental data. *Physica D, 20,* 217–236.

Bulsari, A. B., & Saxén, H. (1995). A recurrent network for modeling noisy temporal sequences. *Neurocomputing, 7,* 29–40.

Deco, G., & Schürmann, B. (1994). Neural learning of chaotic system behavior. *IEICE Trans. Fundamentals, E77A*, 1840–1845.

DeKorte, R. J., Schouten, J. C., & van den Bleek, C. M. (1995). Experimental control of a chaotic pendulum with unknown dynamics using delay coordinates. *Physical Review E, 52A*, 3358–3365.

Diks, C., van Zwet, W. R., Takens, F., & de Goede, J. (1996). Detecting differences between delay vector distributions. *Physical Review E, 53,* 2169–2176.

Eckmann, J.-P., Kamphorst, S. O., Ruelle, D., & Ciliberto, S. (1986). Liapunov exponents from time series. *Physical Review A, 34,* 4971–4979.

Farmer, J. D., & Sidorowich, J. J. (1987). Predicting chaotic time series. *Phys. Rev. Letters, 59*, 62–65.

Grassberger, P., Schreiber, T., & Schaffrath, C. (1991). Nonlinear time sequence analysis. *Int. J. Bifurcation Chaos, 1*, 521–547.

Haken, H. (1983). At least one Lyapunov exponent vanishes if the trajectory of an attractor does not contain a fixed point. *Physics Letters, 94A*, 71.

Haykin, S., & Li, B. (1995). Detection of signals in chaos. *Proc. IEEE, 83,* 95–122.

Haykin, S. & Principe, J. (1998). Making sense of a complex world. *IEEE Signal Processing Magazine*, pp. 66–81.

Haykin, S., & Puthusserypady, S. (1997). Chaotic dynamics of sea clutter. *Chaos, 7,* 777–802.

Hübner, U., Weiss, C.-O., Abraham, N. B., & Tang, D. (1994). Lorenz-like chaos in NH_3-FIR lasers (data set A). In Weigend & Gershenfeld (Eds.), *Time series prediction: Forecasting the future and understanding the past.* Reading, MA: Addison-Wesley.

Jackson, J. E. (1991). *A user's guide to principal components*. New York: Wiley.

Jaeger, L., & Kantz, H. (1996). Unbiased reconstruction of the dynamics underlying a noisy chaotic time series. *Chaos, 6,* 440–450.

Kaplan, J., & Yorke, J. (1979). Chaotic behavior of multidimensional difference equations. *Lecture Notes in Mathematics*, 730.

King, G. P., Jones, R., & Broomhead, D. S. (1987). *Nuclear Physics B* (Proc. Suppl.), *2,* 379.

Krischer, K., Rico-Martinez, R., Kevrekidis, I. G., Rotermund, H. H., Ertl, G., & Hudson, J. L. (1993). Model identification of a spatiotemporally varying catalytic reaction. *AIChe Journal, 39,* 89–98.

Kuo, J. M., & Principe, J. C. (1994). Reconstructed dynamics and chaotic signal modeling. In *Proc. IEEE Int'l Conf. Neural Networks, 5*, 3131–3136.

Lapedes, A., & Farber, R. (1987). *Nonlinear signal processing using neural networks: Prediction and system modelling* (Tech. Rep. No. LA-UR-87-2662). Los Alamos, NM: Los Alamos National Laboratory.

Lin, T., Horne, B. G., Tino, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Trans. on Neural Networks, 7*, 1329.

Ott, E. (1993). *Chaos in dynamical systems*. New York: Cambridge University Press.

Principe, J. C., Rathie, A., & Kuo, J. M. (1992). Prediction of chaotic time series with neural networks and the issue of dynamic modeling. *Int. J. Bifurcation and Chaos, 2*, 989–996.

Principe, J. C., Wang, L., & Motter, M. A. (1998). Local dynamic modeling with self-organizing maps and applications to nonlinear system identification and control. *Proc. of the IEEE, 6*, 2240–2257.

Rico-Martínez, R., Krischer, K., Kevrekidis, I. G., Kube, M. C., & Hudson, J. L. (1992). Discrete- vs. continuous-time nonlinear signal processing of Cu Electrodissolution Data. *Chem. Eng. Comm., 118*, 25–48.

Schouten, J. C., & van den Bleek, C. M. (1993). *RRCHAOS: A menu-driven software package for chaotic time series analysis*. Delft, The Netherlands: Reactor Research Foundation.

Schouten, J. C., Takens, F., & van den Bleek, C. M. (1994a). Maximum-likelihood estimation of the entropy of an attractor. *Physical Review E, 49*, 126–129.

Schouten, J. C., Takens, F., & van den Bleek, C. M. (1994b). Estimation of the dimension of a noisy attractor. *Physical Review E, 50*, 1851–1861.

Siegelmann, H. T., Horne, B. C., & Giles, C. L. (1997). Computational capabilities of recurrent NARX neural networks. *IEEE Trans. on Systems, Man and Cybernetics—Part B: Cybernetics, 27*, 208.

Su, H-T., McAvoy, T., & Werbos, P. (1992). Long-term predictions of chemical processes using recurrent neural networks: A parallel training approach. *Ind. Eng. Chem. Res., 31*, 1338–1352.

Takens, F. (1981). Detecting strange attractors in turbulence. *Lecture Notes in Mathematics, 898*, 365–381.

von Bremen, H. F., Udwadia, F E., & Proskurowski, W. (1997). An efficient QR based method for the computation of Lyapunov exponents. *Physica D, 101*, 1–16.

Wan, E. A. (1994). Time series prediction by using a connectionist network with internal delay lines. In A. S. Weigend & N. A. Gershenfeld (Eds.), *Time series prediction: Forecasting the future and understanding the past*. Reading, MA: Addison-Wesley.

Weigend, A. S., & Gershenfeld, N. A. (1994). *Time series prediction: Forecasting the future and understanding the past*. Reading, MA: Addison-Wesley.

Werbos, P. J., McAvoy, T., & Su, T. (1992). Neural networks, system identification, and control in the chemical process industries. In D. A. White & D. A. Sofge (Eds.), *Handbook of intelligent control*. New York: Van Nostrand Reinhold.

Received December 7, 1998; accepted October 13, 1999.