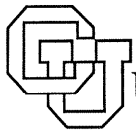
# Time Series Analysis and Prediction

Andreas S. Weigend

CU-CS-744-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE ACKNOWLEDGMENTS SECTION.

Weigend, A. S. (1996) "Time Series Analysis and Prediction" In *Mathematical Perspectives on Neural Networks*, edited by P. Smolensky, M. C. Mozer, and D. E. Rumelhart. Hillsdale, NJ: Erlbaum.

# 12 Time Series Analysis and Prediction

Andreas S. Weigend
*University of Colorado*

## 1. INTRODUCTION

The desire to predict the future and understand the past drives the search for laws that explain the behavior of observed phenomena; examples range from the irregularity in a heartbeat to the volatility of a currency exchange rate. If there are known underlying deterministic equations, in principle they can be solved to forecast the outcome of an experiment based on knowledge of the initial conditions. To make a forecast if the equations are not known, one must find both the rules and the actual state of the system. In this chapter we will focus on phenomena for which underlying equations are not given; the rules that govern the evolution must be inferred from regularities in the past. For example, the motion of a pendulum or the rhythm of the seasons carry within them the potential for predicting their future behavior from knowledge of their oscillations without requiring insight into the underlying mechanism. We will use the terms "understanding" and "learning" to refer to two complementary approaches taken to analyze an unfamiliar time series. Understanding is based on explicit mathematical insight into how systems behave, and learning is based on algorithms that can emulate the structure in a time series. More specifically, we use information theory to obtain some insights into the system, and we use neural networks to build models that emulate the system's behavior. In both cases, the goal is to explain observations; the important related problem of using knowledge about a system for controlling it in order to produce some desired behavior is discussed in Chapter 11 in this volume by Narendra and Li.

Time series analysis has three goals: forecasting, modeling, and characterization. The aim of *forecasting* (also called predicting) is to accurately predict the

short-term evolution of the system; the goal of *modeling* is to find a description that accurately captures features of the long-term behavior of the system. These are not necessarily identical: finding governing equations with proper long-term properties may not be the most reliable way to determine parameters for good short-term forecasts, and a model that is useful for short-term forecasts may have incorrect long-term properties. The third goal, system *characterization*, attempts with little or no a priori knowledge to determine fundamental properties, such as the number of degrees of freedom of a system or the amount of randomness. This overlaps with forecasting but can differ; the complexity of a model useful for forecasting may not be related to the actual complexity of the system.

Before the 1920s, forecasting was done by simply extrapolating the series through a global fit in the time domain. The beginning of modern time series prediction might be set at 1927 when Yule invented the autoregressive technique in order to predict the annual number of sunspots. His model predicted the next value as a weighted sum of previous observations of the series. In order to obtain "interesting" behavior from such a linear system, outside intervention in the form of external shocks must be assumed. For the half-century following Yule, the reigning paradigm remained that of linear models driven by noise.

There are simple cases, however, for which this paradigm is inadequate. For example, a simple iterated map, such as the logistic equation [Eq. (12.13)], can generate a broadband power spectrum that cannot be obtained by a linear approximation. The realization that apparently complicated time series can be generated by very simple equations pointed to the need for a more general theoretical framework for time series analysis and prediction.

Two crucial developments occurred around 1980; both were enabled by the general availability of powerful computers that permitted much longer time series to be recorded, more complex algorithms to be applied to them, and the data and the results of these algorithms to be interactively visualized. The first development, state-space reconstruction by time-delay embedding, drew on ideas from differential topology and dynamical systems to provide a technique for recognizing when a time series has been generated by deterministic governing equations and, if so, for understanding the geometrical structure underlying the observed behavior. The second development was the emergence of the field of machine learning, typified by neural networks, that can adaptively explore a large space of potential models.

Since both approaches are data driven (rather than theory driven), we illustrate the ideas presented in this chapter with a "real-world" data set, recorded from a far-infrared laser in chaotic state. It was used in the *Santa Fe Times Series Prediction and Analysis Competition* as data set A (Hübner Weiss, Abraham, & Tang 1994). It is plotted in Fig. 12.1, along with other 5 data sets used in the competition: (B) physiological date from a patient with sleep apnea, (C) high-frequency currency exchange rate data, (D) a numerically generated series designed for the competition, (E) astrophysical data from a variable star, and (F) J. S. Bach's final

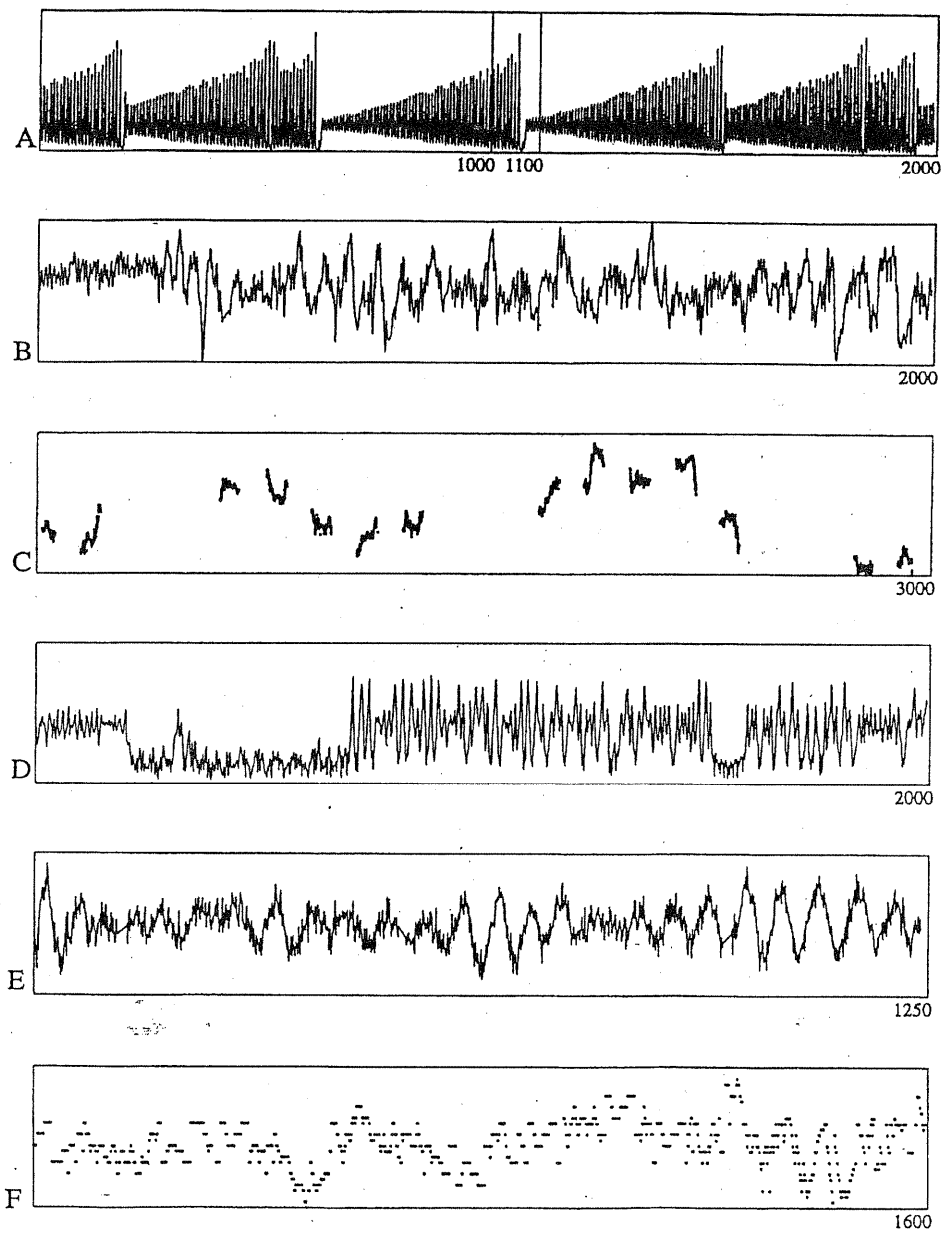



FIG. 12.1. Examples of some time series. The first series, the laser data, is used as example for all models discussed in this chapter.

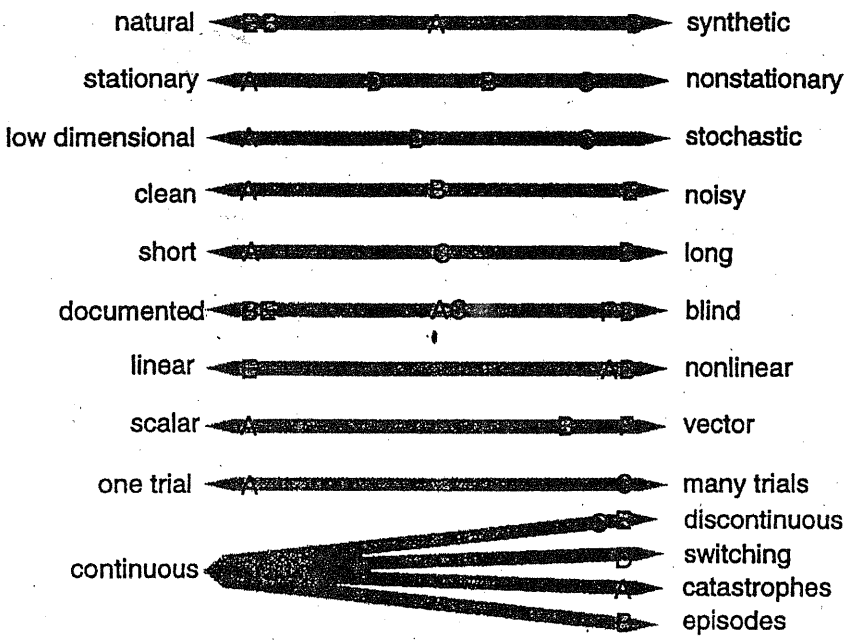



FIG. 12.2. Some attributes spanned by the data sets.

(unfinished) fugue from *The Art of the Fugue*.[1] Some characteristics or these time series are summarized in Fig. 12.2. The appropriate level of description for these data ranges from low-dimensional stationary dynamics to stochastic process. The competition tasks included predicting the (withheld) continuations of the data sets with respect to given error measures, as well as characterizing the systems, for example, through the number of degrees of freedom, predictability, noise characteristics, and the nonlinearity of the system. The results of the competition and of the follow-up NATO Advanced Research Workshop are collected in Weigend and Gershenfeld eds. 1994.

Before turning to the modern analysis and prediction of time series, we review linear models for time series and show where they break down.

[1]The data sets are available in the directory `ftp://ftp.cs.colorado.edu/pub/Time-Series/SantaFe`, or via `http://www.cs.colorado.edu/~andreas/TSWelcome.html`.

## 2. LINEAR TIME SERIES MODELS

Linear time series models have two particularly desirable features: they can be understood in great detail and they are straightforward to implement. The penalty for this convenience is that they may be entirely inappropriate for even moderately complicated systems. In this section we will review their basic features and then consider why and how such models fail. The literature on linear time series analysis is vast; a good introduction is the very readable book by Chatfield (1989), many derivations can be found in the comprehensive text by Priestley (1981), and a classic reference is Box and Jenkins' book (1976). Historically, the general theory of linear predictors can be traced back to Kolmogorov (1941) and to Wiener (1949).

### 2.1. Autoregressive Moving Average, Finite-Impulse Response, and Other Definitions

There are two complementary tasks that need to be discussed: understanding how a given model behaves and finding a particular model that is appropriate for a given time series. We start with the former task. It is simplest to discuss separately the role of external inputs (moving average models) and internal memory (autoregressive models).

#### *2.1.1. Properties of a Given Linear Model*

*Moving Average (MA) Models.* Assume we are given an external input series $\{e_t\}$ and want to modify it to produce another (the observed) series $\{x_t\}$. Assuming linearity of the system and causality (the present value of $x$ is influenced by the present and $N$ past values of the input series $e$), the relationship between the input the output is

$$x_t = \sum_{n=0}^{N} b_n e_{t-n} = b_0 e_t + b_1 e_{t-1} + \cdots + b_N e_{t-N} \tag{1}$$

This equation describes a convolution filter: the new series $x$ is generated by a linear filter with coefficients $b_0, \ldots, b_N$ from the series $e$. Statisticians and econometricians call this an $N$th-order moving average model MA($N$). The origin of this (sometimes confusing) terminology can be seen if one pictures a simple smoothing filter that averages the last few values of series $e$. Engineers call this a *finite-impulse response* (FIR) filter, because the output is guaranteed to go to zero at $N$ time steps after the input becomes zero.

Properties of the output series $x$ clearly depends on the input series $e$. The task is to describe the system independent of a specific input sequence. For a linear system, the response of the filter is independent of the input.

We will give three equivalent characterizations of an MA model: in the time domain (the impulse response of the filter), in the frequency domain (its spectrum),

and in terms of its autocorrelation coefficients. In the first case, we assume that the input is nonzero only at a single time step $t_0$ and that it vanishes for all other times. The response (in the time domain) to this impulse is simply given by the $b$s in Eq. (12.1): at each time step the impulse moves up to the next coefficient until, after $N$ steps, the output disappears. The series $b_N, b_{N-1}, \ldots, b_0$ is thus the impulse response of the system. The response to an arbitrary input can be computed by superimposing the response at appropriate delays, weighted by the respective input values (convolution). The transfer function thus completely describes a linear system, that is, a system where the superposition principle holds: the output is determined by impulse response and input.

Sometimes it is more convenient to describe the filter in the frequency domain. This is useful (and simple) because a convolution in the time domain becomes a product in the frequency domain. If the input to a MA model is an impulse (which has a flat power spectrum), the discrete Fourier transform of the output is given by

$$\sum_{n=0}^{N} b_n \exp(-i2\pi nf) \tag{2}$$

(see, for example, Box & Jenkins 1976, p. 69). The power spectrum is given by the squared magnitude of this:

$$|b_0 + b_1 e^{-i2\pi 1 f} + b_2 e^{-i2\pi 2 f} + \cdots + b_N e^{-i2\pi N f}|^2 \tag{3}$$

The third way of representing yet again the same information is, in terms of the autocorrelation coefficients, defined in terms of the mean $\mu = \langle x_t \rangle$ and the variance $\sigma^2 = \langle (x_t - \mu)^2 \rangle$ by

$$\rho_\tau \equiv 1/\sigma^2 \langle (x_t - \mu)(x_{t-\tau} - \mu) \rangle \tag{4}$$

The angular brackets $\langle \cdot \rangle$ denote expectation values, in the statistics literature often indicated by $E\{\cdot\}$. The autocorrelation coefficients describe how much, on average, two values of a series that are $\tau$ time steps apart co-vary with each other. (We will later replace this linear measure with mutual information, suited also to describe nonlinear relations). If the input to the system is a stochastic process with input values at different times uncorrelated, $\langle e_i e_j \rangle = 0$ for $i \neq j$, then all of the cross terms will disappear from the expectation value in Eq. (12.3), and the resulting autocorrelation coefficients are

$$\rho_\tau = \begin{cases} \dfrac{1}{\sum_{n=0}^{N} b_n^2} \displaystyle\sum_{n=r}^{N} b_n b_{n-|\tau|} & |\tau| \leq N \\ 0 & |\tau| > N \end{cases} \tag{5}$$

*Autoregressive (AR) Models.* MA (or FIR) filters operate in an open loop without feedback; they can only transform an input that is applied to them. If we

do not want to drive the series externally, we need to provide some feedback (or memory) in order to generate internal dynamics,

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + e_t \tag{6}$$

This is called an $M$th oeder *autoregressive model* [AR($M$)] or an *infinite-impulse response* (IIR) filter (because the output can continue after the input ceases). Depending on the application, $e_t$ can represent either a controlled input to the system or noise. As before, if $e$ is white noise, the autocorrelations of the output series $x$ can be expressed in terms of the model coefficients. Here, however, due to the feedback coupling of previous steps, we obtain a set of linear equations rather than just a single equation for each autocorrelation coefficient. By multiplying Eq. (12.6) by $x_{t-\tau}$, taking expectation values, and normalizing (see Box & Jenkins 1976, p. 54), the autocorrelation coefficients of an AR model are found by solving this set of linear equations, traditionally called the *Yule–Walker equations*,

$$\rho_\tau = \sum_{m=1}^{M} a_m \rho_{\tau-m}, \quad \tau > 0 \tag{7}$$

Unlike the MA case, the autocorrelation coefficient need not vanish after $M$ steps. Taking the Fourier transform of both sides of Eq. (12.6) and rearranging terms shows that

$$\text{output} = \text{input}/1 - \sum_{m=1}^{m} a_m \exp(-i2\pi mf) \tag{8}$$

The power spectrum of output is thus that of the input times

$$\frac{1}{|1 - a_1 e^{-i2\pi 1 f} - a_2 e^{-i2\pi 2 f} - \cdots - a_M e^{-i2\pi Mf}|^2} \tag{9}$$

To generate a specific realization of the series, we must specify the initial conditions, usually by the first $M$ values of series $x$. Beyond that, the input term $e_t$ is crucial for the life of an AR model. If there was no input, we might be disappointed by the series we get: depending on the amount of feedback, after iterating it for a while, the output produced can only decay to zero, diverge, or oscillate periodically. (In the case of a first-order AR model, this can easily be seen: if the absolute value of the coefficient is less than unity, the value of $x$ exponentially decays to zero; if it is larger than unity, it exponentially explodes. For higher order AR models, the long-term behavior is determined by the locations of the zeros of the polynomial with coefficients $a_i$.)

*Autoregressive Moving-Average (ARMA) Models.* The next step in complexity is to allow both AR and MA parts in the model; this is called an

ARMA($M$, $N$) model:

$$x_t = \sum_{m=1}^{M} a_m x_{t-m} + \sum_{n=0}^{N} b_n e_{t-n} \tag{10}$$

Its output is most easily understood in terms of the $z$ transform (Oppenheim & Schafer 1989), which generalizes the discrete Fourier transform to the complex plane

$$X(z) \equiv \sum_{t=-\infty}^{\infty} x_t z^t \tag{11}$$

On the unit circle, $z = \exp(-2i\pi f)$, the $z$ transform reduces to the discrete Fourier transform. Off the unit circle, the $z$ transform measures the rate of divergence or convergence of a series. Since the convolution of two series in the time domain corresponds to the multiplication of their $z$ transforms, the $z$ transform of the output of an ARMA model is

$$\begin{aligned} X(z) &= A(z)X(z) + B(z)E(z) \\ &= \frac{B(z)}{1 - A(z)} E(z) \end{aligned} \tag{12}$$

(ignoring a term that depends on the initial conditions). The input $z$ transform $E(z)$ is multiplied by a transfer function that is unrelated to it; the transfer function will vanish at zeros of the MA term ($B(z) = 0$) and diverge at poles ($A(z) = 1$) due to the AR term (unless cancelled by a zero in the numerator). As $A(z)$ is an $M$th-order complex polynomial, and $B(z)$ is $N$th order, there will be $M$ poles and $N$ zeros. Therefore, the $z$ transform of a time series produced by Eq. (12.10) can be decomposed into a rational function and a remaining (possibly continuous) part due to the input. The number of poles and zeros determines the number of degrees of freedom of the system (the number of previous states that the dynamics retains). Note that since only the ratio enters, there is no unique ARMA model. In the extreme cases, a finite-order AR model can always be expressed by an infinite-order MA model, and vice versa.

ARMA models have dominated all areas of time series analysis and discrete-time signal processing for more than half a century. For example, in speech recognition and synthesis, linear predictive coding (Press, Flannery, Teukolsky, & Vetterling 1992, p. 571) compreses speech by transmitting the slowly varying coefficients for a linear model (and possibly the remaining error between the linear forecast and the desired signal) rather than the original signal. If the model is good, it transforms the signal into a small number of coefficients plus residual white noise (of one kind or another).

### 2.1.2. *Fitting a Linear Model to a Given Time Series*

*Fitting the Coefficients.* The Yule–Walker set of linear equations [Eq. (12.7)] allowed us to express the autocorrelation coefficients of a time series in terms of

the AR coefficients that generated it. But there is a second reading of the same equations: they also allow us to estimate the coefficients of an AR($M$) model from the observed correlational structure of an observed signal. (In statistics, it is common to emphasize the difference between a given model and an estimated model by using different symbols, such as $\hat{a}$ for the estimated coefficients of an AR model. In this chapter, we avoid introducing another set of symbols; we hope that it is clear from the context whether values are theoretical estimated.) An alternative approach views the estimation of the coefficients as a regression problem: expressing the next value as a function of $M$ previous values, that is, linearly regress $x_t$ onto $\{x_{t-1}, \ldots, x_{t-M}\}$. This can be done by minimizing squared errors: the parameters are determined such that the squared difference between the model output and the observed value, summed over all time steps in the fitting region, is as small as possible. There is no comparable conceptually simple expression for finding MA and full ARMA coefficients from observed data. For all cases, however, standard techniques exist, often expressed as efficient recursive procedures (Box & Jenkins 1976; Press, Flannery, Tenkolsky, & Vetterling 1992).

Although there is no reason to expect that an arbitrary signal was produced by a system that can be written in the form of Eq. (12.10), it is reasonable to attempt to approximate a linear system's true transfer function ($z$ transform) by a ratio of polynomials, that is, an ARMA model.

*Selecting the (Order of the) Model.* So far we have dealt with the question of how to estimate the coefficients from data for an ARMA model of order ($M$, $N$), but have not addressed the choice of the order of the model. There is not a unique best choice for the values or even for the number of coefficients to model a data set—as the order of the model is increased, the fitting error decreases, but the test error of the forecasts beyond the training set will usually start to increase at some point because the model will be fitting extraneous noise in the system. There are several prescriptions to find the "right" order [such as the Akaike Information Criterion (AIC), Akaike 1974), but they rely on assumptions about the linearity of the model and the distribution from which the noise is drawn. When it is not clear whether these assumptions hold, a simple approach (but possibly wasteful in terms of the data) is to hold back some of the training data and use these to evaluate the performance of competing models. Selecting the order of a linear model is a specific example of model selection in general. The problem of model selection will reappear even more forcefully in the context of nonlinear models, because they are more flexible and, hence, more capable of modeling irrelevant noise. We will return to this problem of overfitting in Sec. 5.4 in the context of prediction with neural networks.

## 2.2. The Breakdown of Linear Models

We have seen that ARMA coefficients, power spectra, and autocorrelation coefficients contain the same information about a linear system driven by uncorrelated

white noise. Thus, if and only if the power spectrum is a useful characterization of the relevant features of a time series, an ARMA model will be a good choice for describing it. This appealing simplicity can fail entirely for even simple nonlinearities if they lead to complicated power spectra—as they can. Two time series can have very similar broadband spectra but can be generated from systems with very different properties, such as a linear system that is driven stochastically by external noise, and a deterministic (noise-free) nonlinear system with a small number of degrees of freedom. One of the key problems addressed in this chapter is how these cases can be distinguished; linear operators definitely will not be able to do the job.

Let us consider two nonlinear examples of discrete-time maps (like an AR model, but now nonlinear):

The first example can be traced back to Ulam and von Neumann (1947): the next value of a series is derived from the present one by a simple parabola

$$x_t = \lambda x_{t-1}(1 - x_{t-1}) \tag{13}$$

Popularized in the context of population dynamics as an example of a "simple mathematical model with very complicated dynamics" (May 1976), it has been found to describe a number of controlled laboratory systems, such as hydrodynamic flows and chemical reactions, because of the universality of smooth unimodal maps. In this context, this parabola, is called the logistic map or quadratic map. The value $x_t$ deterministically depends on the previous value $x_{t-1}$; $\lambda$ is a parameter that controls the qualitative behavior, ranging from a fixed point (for small values of $\lambda$) to deterministic chaos. For example, for $\lambda = 4$, each iteration destroys (or creates, depending on the perspective) one bit of information. Consider that by plotting $x_t$ against $x_{t-1}$ each value of $x_t$ has two equally likely predecessors or, equally well, the average slope (its absolute value) is two: if we know the location within $\epsilon$ before the iteration, we will on average know it within $2\epsilon$ afterwards. This exponential increase in uncertainty is the hallmark of deterministic chaos (divergence of nearby trajectories).

The second example is equally simple: consider the time series generated by the map

$$x_t = 2x_{t-1} \pmod 1 \tag{14}$$

The action of this map is easily understood by considering the position $x_t$ written in a binary fractional expansion (i.e., $x_t = 0.d_1d_2\ldots = (d_1 \times 2^{-1}) + (d_2 \times 2^{-2}) + \cdots$): each iteration shifts every digit one place to the left ($d_i \leftarrow d_{i+1}$). This means that the most significant digit $d_1$ is discarded and one more digit of the binary expansion of the initial condition is revealed. This map can be implemented in a simple physical system consisting of a classical billiard ball and reflecting surfaces, where the $x_t$ are the successive positions at which the ball crosses a given line (Moore 1991).

Both systems are completely deterministic (their evolutions are entirely determined by the initial condition $x_0$), yet they can easily generate time series with

broadband power spectra. In the context of an ARMA model, a broadband component in a power spectrum of the output must come from external noise input to the system, but here it arises in two one-dimensional system as simple as a parabola and two straight lines. Nonlinearities are essential for the production of "interesting" behavior in a deterministic system; the point here is that even simple nonlinearities suffice. In order to capture nonlinearities, we must relax the strong assumption of linear models and consider a wider class of broader (or weaker) models.

Strong models have strong assumptions. They are usually expressed in a few equations with a few parameters and can often explain a plethora of phenomena. In weak models, on the other hand, there are only a few domain-specific assumptions. To compensate for the lack of explicit knowledge, weak models usually contain many more parameters (which can make a clear interpretation difficult). It can be helpful to conceptualize models in the two-dimensional space spanned by the axes data poor $\leftrightarrow$ data rich and theory poor $\leftrightarrow$ theory rich. Because of the dramatic expansion of the capability for automatic data acquisition and processing, it is increasingly feasible to venture into the theory-poor and data-rich domain. The premise of the approaches described in this chapter is that we do not have "first principles" about the observed time series that we can start from. This is both true in the next section where we describe characterization methods that are essentially model free, and in the subsequent section on prediction.

## 3. TIME-DELAY EMBEDDING AND INFORMATION THEORY

In this section, we first discuss the meaning of embedding from the perspective of physics: time-delay embedding, although appearing similar to traditional models with lagged vectors, makes a crucial link between behavior in the reconstructed state space and the internal degrees of freedom. We then present information-theoretic measures that can be used to characterize the series.

Yule's original idea for forecasting was that future predictions can be obtained by using immediately preceding values. An ARMA model, Eq. (12.10), can be rewritten as a dot product between vectors of the time-lagged variables and coefficients,

$$x_t = \boldsymbol{a} \cdot \boldsymbol{x}_{t-1} + \boldsymbol{b} \cdot \boldsymbol{e}_t \tag{15}$$

where $\boldsymbol{x}_t = (x_t, x_{t-1}, \ldots, x_{t-(d-1)})$ and $\boldsymbol{a} = (a_1, a_2, \ldots, a_d$. [We slightly change notation here: what was $M$ (the order of the AR model) is now called $d$ (for dimension).] Such lag vectors, also called tapped delay lines, are used routinely in the context of signal processing and time series analysis, suggesting that they are more than just a typographical convenience.

In fact, there is a deep connection between time-lagged vectors and underlying dynamics. This connection was proposed in 1980 by Ruelle, first put in writing

by Packard, Crutchfield, Farmer, and Shaw (1980), first proven by Takens (1981), and later strengthened by Sauer, Yorke, and Casdagli (1991). Delay vectors of sufficient length are not just a way of trying to capture the state of a linear system—it turns out that delay vectors can recover the full geometrical structure of a nonlinear system! These results address the general problem of inferring the behavior of the intrinsic degrees of freedom of a system when only some function of the underlying state of the system is measured. We here address the question: What properties can be inferred from the time series if we do not assume knowledge of the specific governing equations, but only that the series was generated from a dynamical system, given by some set of unknown differential equations? (If the governing equations and the functional form of the observable are known in advance, then a Kalman filter is the optimal linear estimator of the state of the system. See also the discussion on observability in Chapter 11 this volume by Narendra and Li.)

## 3.1. State-Space Reconstruction

There are four relevant (and easily confused) spaces and dimensions for this discussion. [The first point (configuration space and potentially accessible degrees of freedom) will not be used again in this chapter. On the other hand, the dimension of the solution manifold (the actual degrees of freedom) will be important both for characterization and for prediction.] They are as follows:

1. The *configuration space* of a system is the speces "where the equations live." It specifies the values of all the potentially accessible physical degrees of freedom of the system. For example, for a fluid governed by the Navier–Stokes partial differential equations, these are the infinite-dimensional degrees of freedom associated with the continuous velocity, pressure, and temperature fields.
2. The *solution manifold* is where "the solution lives," that is, the part of the configuration space that the system actually explores as its dynamics unfolds (such as the support of an attractor or an integral surface). Because of unexcited or correlated degrees of freedom, this can be much smaller than the configuration space; the dimension of the solution manifold is the number of parameters that are needed to uniquely specify a distinguishable state of the overall system. For example, in some regimes the infinite physical degrees of freedom of a convecting fluid reduce to a small set of coupled ordinary differential equations for a mode expansion (Lorenz 1963). Dimensionality reduction from the configuration space to the solution manifold is a common feature of dissipative systems: dissipation in a system will reduce its dynamics onto a lower dimensional subspace (Temam 1988).
3. The *observable* is a (usually) one-dimensional function of the variables of the configuration space. In an experiment, this might be the temperature or a velocity component at a point in the fluid.

4. The *reconstructed state space* is obtained from that (scalar) observable by combining past values of it to form a lag vector (which for the convection case would aim to recover the evolution of the components of the mode expansion).

Given a time series measured from such a system—and no other information about the origin of the time series—the question is: What can be deduced about the underlying dynamics?

Let $\boldsymbol{y}$ be the state vector on the solution manifold (in the convection example the components of $\boldsymbol{y}$ are the magnitude of each of the relevant modes), let $d\boldsymbol{y}/dt = f(\boldsymbol{y})$ be the governing equations, and let the measured quantity be $x_t = x(y(t))$ (e.g., the temperature at a point). (The results that we will cite here also apply to systems that are described by iterated maps.)

Given a delay time $\tau$ and a dimension $d$, a lag vector $\boldsymbol{x}$ can be defined

$$\text{lag vector: } \boldsymbol{x}_t = (x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) \tag{16}$$

The central result is that the behavior of $\boldsymbol{x}$ and $\boldsymbol{y}$ will differ only by a smooth local invertible change of coordinates (i.e., the mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$ is an embedding[2] which requires that it be diffeomorphic) for almost every possible choice of $f(\boldsymbol{y})$, $x(\boldsymbol{y})$, and $\tau$, as long as $d$ is large enough (in a way that we will make precise), $x$ depends on at least some of the components of $\boldsymbol{y}$, and the remaining components of $\boldsymbol{y}$ are coupled by the governing equations to the ones that influence $x$. The proof of this result have two parts: a local piece, showing that the linearization of the embedding map is almost always nondegenerate, and a global part, showing that this holds everywhere (Gershenfeld 1989). If $\tau$ tends to zero, the embedding will tend to lie on the diagonal of the embedding space and, as $\tau$ is increased, it sets a length scale for the reconstructed dynamics. There can be degenerate choices for $\tau$ for which the embedding fails (such as choosing it to be exactly equal to the period of a periodic system), but these degeneracies almost always will be removed by an arbitrary perturbation of $\tau$. The intrinsic noise in physical systems guarantees that these results hold in all known nontrivial examples, although in practice, if the coupling between degrees of freedom is sufficiently weak, then the available experimental resolution will not be large enough to detect them (see Casdagli, Eubank, Farmer, & Gibson 1991, for further discussion of how noise constrains embedding). [The Whitney embedding theorem from the 1930s (see Guillemin & Pollack 1974, p. 48) guarantees that the number of independent observations $d$ required to embed an arbitrary manifold (in the absence of noise) into a Euclidean embedding space will be no more than twice the dimension of the manifold. For

[2]The term *embedding* is used in the literature in two senses. In its wider sense, the term denotes any lag-space representation, whether there is a unique surface or not. In its narrower (mathematical) sense used here, the term applies if and only if the resulting surface is unique, that is, if a diffeomorphism exists between the solution manifold in configuration space and the manifold in lag space.

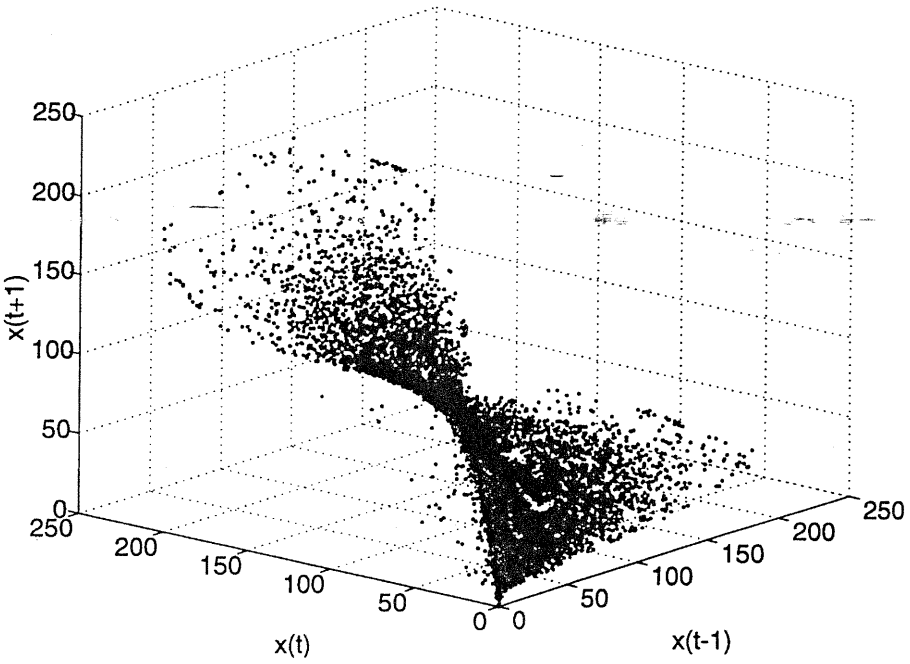



FIG. 12.3. State-space embedding of the laser data. They grey scale indicates the degree of certainty ($v(x)$ from lightest to darkest, $<5$, $5-8$, $8-16$, $>16$). Different regions have clearly different degrees of predictability.

example, a two-dimensional Möbius strip can be embedded in a three-dimensional Euclidean space, but a two-dimensional Klein bottle requires a four-dimensional space.]

The laser data that we will use as illustration appear complicated when plotted as a time series (Fig. 12.1). The simple structure of the system becomes visible in a figure of its three-dimensional embedding. Figure 12.3 allows one to guess a value of the dimension of the manifold of around two. In contrast, high-dimensional dynamics would show up as a structureless cloud in such a three-dimensional plot. (The different grey values used in Fig. 12.3 will be discussed in the context of local error bars in Sec. 5.1).

Time-delay embedding differs from traditional experimental measurements in three fundamental respects:

1. It provides detailed information about the behavior of degrees of freedom other than the one that is directly measured.
2. It rests on probabilistic assumptions, and—although it has been routinely and reliably used in practice—it is not guaranteed to be valid for any system.

3. It allows precise questions only about quantities that are invariant under such a transformation, since the reconstructed dynamics have been modified by an unknown smooth change of coordinates.

This last restriction may be unfamiliar, but it is surprisingly unimportant: the embedded data suffice as a representation for characterizing the essential features of the dynamics that produced the time series, as well as for predicting its future behavior.

## 3.2. Characterization

Having introduced an embedding space in the previous section, we now characterize a system through an explicit analysis of the structure in that space. This characterization is in terms of the information-theoretic measures of redundancy and source entropy, essentially describing the numbers of bits a new measurement reveals about the system [the average (of the negative logarithm) of the number of the states the system can go to at each time step]. No specific forecasting model is assumed here; in this sense this approach to characterization can be called model free. We will contrast it later with characterization based on an analysis of a forecasting model and its prediction errors.

Simple systems can produce time series that appear to be complicated; complex systems can produce time series that are complicated. These two extremes have different goals and require different techniques; what constitutes a successful forecast depends on where the underlying system falls on this continuum. In this section we will look at characterization methods that can be used to extract some of the essential properties that lie behind an observed time series, both as an end in itself and as a guide to further analysis and modeling. We use some concepts of information theory to introduce a basic measure of predictability of a given series, its source entropy. Characterizing time series through their frequency content goes back to Schuster's "periodogram" (1898). For a simple linear system the traditional spectral analysis is very useful (peaks = modes = degrees of freedom), but different nonlinear systems can have similar featureless broadband power spectra. Therefore, a broadly useful characterization of a nonlinear system cannot be based on its frequency content.

It is always possible to define a time-delayed vector from a time series, but this certainly does not mean that it is always possible to identify meaningful structure in the embedded data. Because the mapping between a delay vector and the system's underlying state is not known, the precise value of an embedded data point is not significant. Because an embedding is diffeomorphic (smooth and invertible), however, a number of important properties of the system will be preserved by the mapping. These include local features, such as the number of degrees of freedom, and global topological features, such as the linking of trajectories. The literature on characterizing embedded data in terms of such invariants is vast, motivated

by the promise of obtaining deep insight into observations, but plagued by the problem that plausible algorithms will always produce a result—whether or not the result is significant. General reviews of this area may be found in Ruelle and Eckmann (1985), Gershenfeld (1989), and Theiler (1990).

We summarize here an information-based approach due to Fraser (1989) that applied to the Santa Fe data sets by Paluš (1994), and by Theiler, Linsay, and Rubin (1994); see also Prichard and Theiler (in press).

Although the connection between information theory and ergodic theory has long been appreciated, Shaw (1981) helped point out the connection between dissipative dynamics and information theory, and Fraser and Swinney (1986) first used information-theoretic measures to find optimal embedding lags. This example of the physical meaning of information (Landauer 1991) can be viewed as an application of information theory back to its roots in dynamics: Shannon (1948) built his theory of information on the analysis of the single-molecule Maxwell Demon by Szilard in 1929, which in turn was motivated by Maxwell and Boltzmann's effort to understand the microscopic dynamics of the origin of thermodynamic irreversibility (circa 1870).

Assume that a time series $x(t)$ has been digitized to integer values lying between 1 and $N$. If a total of $n_T$ points have been observed and a particular value of $x$ is recorded $n_x$ times, then the probability of seeing this value is estimated to be $p_1(x) = n_x/n_T$.[3] [The subscript of the probability indicates that we are at present considering one-dimensional distributions (histograms). It will soon be generalized to $d$-dimensional distributions.] In terms of this probability, the *entropy* of this distribution is given by

$$H_1(N) = -\sum_{x-1}^{N} p_1(x) \log_2 p_1(x) \tag{17}$$

Let us consider two extremes. On the one hand, if there is only one possible value for $x$, the probability $p$ of that value is unity, and $\log_2 p(x) = 0$. All other values in the sum are suppressed by the weighting with zero probability. This case can be viewed as maximum order because we know the next state with certainty. On the other hand, let us assume that there are $N$ bins (or cells or symbols) that are equally likely. (This means that the full resolution of $x$ is required). In this case, the entropy reaches is maximal possible value of $H_1(N) = \log_2 N$. This case can be viewed as maximum disorder because all states are equally likely. All other cases lie between these two extremes, that is, the average number of bits required to describe an isolated observation can range from 0 to $\log_2 N$.

Consider now the case of $M$ equally probable states in the series. As $N$ is increased, the entropy first grows as $\log N$ (since all values are equally probable); it will then reach as asymptotic value of $\log M$ independent of $N$ (once there are

[3]Note that there can be corrections to such estimates if one is interested in the expectation value of functions of the probability (Grassberger 1988).

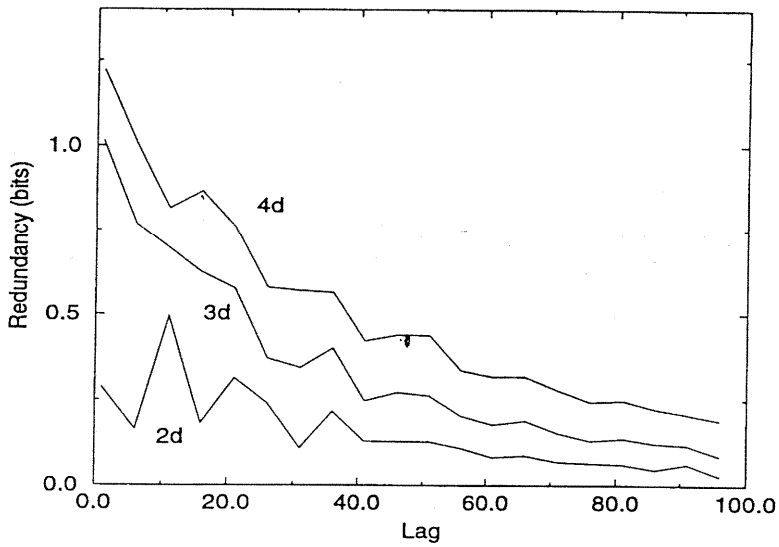



FIG. 12.4. The redundancy (incremental mutual information) of the laser data as a function of the number of time steps. The figure indicates that three past values are sufficient to retrieve most of the predictable structure and that the system has lost the memory of its initial conditions after roughly 100 steps.

given unlimited resolution:

$$h(\tau, N) = \lim_{N\to\infty} \lim_{d\to\infty} H_d(\tau, N) - H_{d-1}(\tau, N) \tag{24}$$

The limit of infinite resolution is usually not needed in practice: the source entropy reaches its maximum asymptotic value once the resolution is sufficiently fine to produce a generating partition (Petersen 1989, p. 243). The source entropy is important because the *Pesin identity* relates it to the sum of the positive Lyapunov exponents (Ruelle & Eckmann 1985):

$$h(\tau) = \tau h(1) = \tau \sum_i \lambda_i^+ \tag{25}$$

The *Lyapunov exponents* $\lambda_i$ are the eigenvalues of the local linearization of the dynamics, measuring the average rate of divergence of the principal axes of an ensemble of nearby trajectories. They can be found from the Jacobian (either directly, if the Jacobian is known, or indirectly from an emulation of the system by a neural network, for example), or by following trajectories (Brown, Bryant, & Abarbanel 1991). Diverging trajectories reveal information about the system that is initially hidden by the measurement quantization. The amount of this information is proportional to the expansion rate of the volume that is given by the sum of the positive exponents.

If the embedding dimension $d$ is large enough, the redundancy is just the difference between the scalar entropy and an estimate of the source entropy,

$$R_d(\tau, N) \approx H_1(\tau, N) - h(\tau, N) \tag{26}$$

In the limit of small lags,

$$H_{d-1}(0, N) = H_d(0, N) \Rightarrow R_d(0, N) = H_1(N) \tag{27}$$

and for long lags

$$\lim_{\tau \to \infty} H_d(\tau, N) = dH_1(\tau, N) \Rightarrow R_d(\infty, N) = 0 \tag{28}$$

The value of $\tau$ where the redundancy vanishes provides an estimate of the limit of predictability of the system at that resolution (prediction horizon) and will be very short if $d$ is less than the minimum embedding dimension. Once $d$ is larger than the embedding dimension (if there is one), then the redundancy will decay much more slowly, and the slope for small $\tau$ will be the source entropy,

$$R_d(\tau, N) = H_1(N) - \tau h(1) \tag{29}$$

We have seen that it is possible from the block entropy [Eq. (12.18)] and the redundancy [Eq. (12.21)] to estimate the resolution, the minimum embedding dimension, the source entropy, and the prediction horizon of the data, and hence learn about the number of degrees of freedom underlying the time series and the rate at which it loses memory of its initial conditions.

Figure 12.4 shows the redundancy for the laser data. The redundancy can be efficiently computed with an $O(N)$ algorithm by sorting the measured values on a simple fixed-resolution binary tree (Gershenfeld 1993). Sorted on the three most significant bits, 10,000 data points were used. Note that three lags suffice to retrieve most of the predictable structure. This is in agreement with Fig. 12.3, where the $2 + 1$ dimensions plotted appear to be sufficient for an embedding. Furthermore, we can read off from Fig. 12.4 that the system has lost memory of its initial condition after about 100 steps.

The tree sort algorithm used to generate Fig. 12.4 is related to the frequently rediscovered fact that box-counting algorithms (such as are needed for estimating dimensions and entropies) can be implemented in high-dimensional space with an $O(N \log N)$ algorithm requiring no auxiliary storage by sorting the appended indices of lagged vectors (Pineda & Sommerer 1994). Equal-probability data structures can be used (at the expense of computational complexity) to generate more reliable unbiased entropy estimates (Fraser & Swinney 1986). Computationally more expensive but more data efficient are methods that use kernels to estimate information theoretic quantities; we have used them in the context of time series prediction and information theory for the selection of the most-relevant subset of inputs (Bonnlander & Weigend 1994). There are many other approaches to characterizing embedded data (and choosing embedding parameters); some of them

are described in Gershenfeld and Weigend (1994). [We here point out a paper by Pi and Peterson (1994) that exploits the notion of continuity of a function (of the conditional probabilities in the lag space as the number of lags increases) and applies it to the laser data.] Regardless of the algorithm employed, it is crucial to understand the nature of the errors in the results (both statistical uncertainty and possible artifacts) and to remember that there is no universally "right" answer for the time delay $\tau$ and the number of lags $d$; the choices will always depend on the goal.

## 4. FORECASTING

If an experimentally observed quantity arises from deterministic governing equations, it is possible to use time-delay embedding to recover a representation of the relevant internal degrees of freedom of the system from the observable. Although the precise values of these reconstructed variables are not meaningful (because of the unknown change of coordinates), they can be used to make precise forecasts because the embedding map preserves their geometrical structure. In this section we explain how this is done for a time series that has been generated by a deterministic system. This is done in the light of the previous section, where we showed how to use information theoretic measures to determine whether or not this is the case.

Figure 12.3 is an example of the structure that an embedding can reveal. Notice that the points form a surface that appears to be single valued; this, in fact, must be the case if the system is deterministic and if $d$, the number of time lags used, is sufficient for an embedding. Differential equations and maps have unique solutions forward in time; this property is preserved under a diffeomorphic transformation and so the first component of an embedded vector must be a unique function of the preceding values once $d$ is large enough. Therefore, the points must lie on a single-valued surface.

Given an embedding, the task of predicting the next point in a series is thus reframed as finding a value $\hat{x}_t$ above the input plane $\{x_{t-1}, \ldots, x_{t-d}\}$. We now describe a variety of approaches, progressing from looking up of the nearest point through local models to global models.

### 4.1. Local Models

Perhaps the simplest solution to the prediction problem is to first create a library of all the past $d$-tupels, $\{(x_{t-1}, \ldots, x_{t-d})\}$ (input patterns). Then, when the pattern comes in whose continuation we want to give, we search through the library for that pattern that is closest to the present one, and simply use as our prediction the continuation that happened at that time. This *nearest neighbor lookup* requires that all points of the past remain stores (as opposed to a neural network, discussed later, where the knowledge is transferred to the weights and the individual patterns can be discarded after learning).

Let us consider noisy observations. It is likely that we are able to improve the prediction quality by averaging over the continuation of a few nearest neighbors (rather than only taking the closest one). Here a tradeoff important in all statistical modeling appears: the *bias-variance dilemma* (e.g., German, Bienenstock, & Doursat 1992). On the one hand, if we take too few neighbors into account, our predictions will still be very noisy (i.e., have a large variance). At the same time, they are very flexible (i.e., have a small bias). On the other hand, if we average over too large a neighborhood, we will have very stable predictions (in the extreme case of always taking all points, a zero-variance predictor), but a quite inflexible model (high bias).

The next model still uses the local information from a few neighbors. Rather than just computing their mean, however we now but build a local linear model, that is, fit a hyperplane through them. (We would like to point out that there are many ways of weighting the errors of the individual points. For example, points farther away from the input can be down weighted. It is also possible to relax the assumption of stationarity made throughout this chapter by keeping track of the time index and weighting more recent points more strongly than points from the distant past). We obtain the prediction by reading off the value of the hyperplane at the input pattern whose neighbors we used to construct the local linear model. Note that we have to fit a new hyperplane for each prediction, requiring us to still keep all points.

Such local linear models were first used for time series prediction by Farmer and Sidorowich (1987). The variation of performance with two crucial parameters—the number of neighbors $k$ and the number of lags of the embedding $d$—has been explored by Casdagli (1991), and applied to the Santa Fe data by Casdagli and Weigend (1994).

Figure 12.5 shows the out-of-sample performance for a local linear model on three of the Santa Fe data sets (laser data, computer-generated data, and heart data) as a function of the number of neighbors $k$ used for the linear fit. The left side of each plot corresponds to a simple lookup of the neighbor closest in lag space; the right corresponds to a global linear model that fits a hyperplane through all points. In all three panels the scale of the $y$ axis (absolute errors) is the same. (Before applying the algorithm, all series were normalized to unit variance).

The first observation is the overall size of the out-of-sample errors. The laser data are much more predictable than the computer-generated data, which in turn are more predictable than the heart data. The second observation concerns the shape of the three curves. The location of the minimum shifts from the left extreme for the laser (next-neighbor lookup), through a clear minimum for the computer generated data (for about 100 neighbors), to a rather flat behavior beyond a sharp drop for the heart data. The third observation concerns the order of the linear model, that is, the number of time delays $d$. For the laser data $d = 6$ is sufficient, and for the computer generated data $d = 12$. For the heart data, because the plots give no indications of low-dimensional chaos, it does not make sense to give an embedding dimension into which the geometry can be disambiguated.

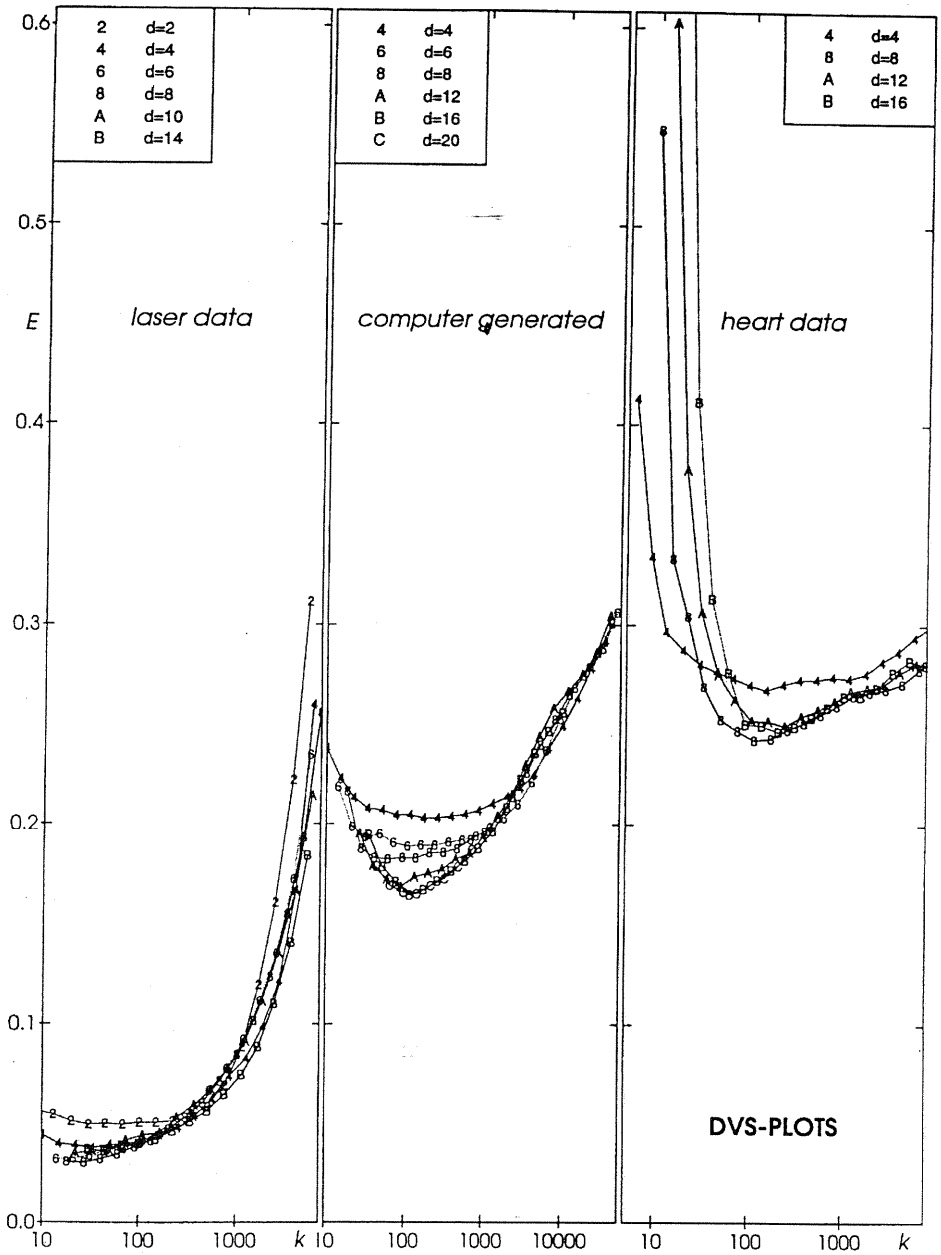



FIG. 12.5. Deterministic vs. stochastic (DVS) plots. The normalized out-of-sample error is shown as function of the number of neighbors $k$ used to construct a local linear model of order $d$.

Local linear models have proven quite successful for time series prediction. The reason is that any manifold is locally linear (i.e., locally a hyperplane). We close this section on local models by summarizing the method used by Sauer (1994) to very successfully predict the laser data. The continuation he obtained is shown in Figs. 12.7 and 12.8. His implementation consisted of five steps:

1. Low pass embed the data to help remove measurement and quantization noise. This low-pass filtering produces a smoothed version of the original series.
2. Generate more points in embedding space by (Fourier-) interpolating between the points obtained from step 1. This is to increase the coverage in embedding space.
3. Find the $k$ nearest neighbors to the point of prediction (the choice of $k$ tries to balance the increasing bias and decreasing variance that come from using a larger neighborhood).
4. Use a local singular-value decomposition (SVD) to project (possibly very noisy) points onto the local surface. (Even if a point is very far away from the surface, this step forces the dynamics back on the reconstructed solution manifold.)
5. Regress a linear model for the neighborhood and use it to generate the forecast.

Because the laser data was generated by low-dimensional smooth dynamics, such a local linear model is able to capture the geometry remarkably well based on the relatively small sample size. The great advantage of local models is their ability to adhere to the local shape of an arbitrary surface; the corresponding disadvantage is that they do not lead to a compact description of the system.

## 4.2. Global Models

For the local models discussed in the previous section, all measurements of the observable from the past had to be stored because any of them may be needed for the next prediction. Furthermore, a new model had to be constructed for every prediction. We now turn to global models where we once build a model of the surface in lag space and then just read off the predictions from this surface.

For more than five decades following Yule (1927), that surface was chosen to be a simple hyperplane. The late 1970s saw the first efforts to weaken global linear AR models. Granger and Anderson (1978) introduced second-order interactions between the inputs, that is, $x_i x_j$. Since two inputs enter linearly into such products, they call it a bilinear model. Tong and Lim (1980) split the input across one variable and allow for two AR models. Because of the presence of this threshold, they call it a *threshold autoregressive* (TAR) model. A recent example of a global model are *multivariate adaptive regression splines* (MARS), introduced by Friedman (1991)

and used on the Santa Fe data by Lewis, Ray, and Stevens (1994). In this chapter, we focus on connectionist modeling.

Neural networks are typically used in pattern recognition, where a collection of features (such as an image) is presented to the network, and the task is to assign the input feature to one or more classes. Another typical use for neural networks is (nonlinear) regression, where the task is to find a smooth interpolation between points. In both these cases, all of the relevant information is presented simultaneously. In contrast, time series prediction involves processing of patterns that evolve over time; the appropriate response at a particular point in time depends not only on the current value of the observable but also on the past. Time series prediction has had an appeal for neural networkers from the very beginning of the field. In 1964, Hu applied Widrow's adaptive linear network to weather forecasting. In the post-backpropagation era, Lapedes and Farber (1987) trained their (nonlinear) network to emulate the relationship between output (the next point in the series) and inputs (its predecessors) for computer-generated time series, and Weigend, Huberman, and Rumelhart (1990) addressed the issue of finding networks of appropriate complexity for predicting observed (real-world) time series. In all of these cases, temporal information is presented spatially to the network by a tapped delay line.

Figure 12.6 shows a typical network; activations flow from the bottom up. In addition to a second layer of (nonlinear) hidden units, we also include direct (linear) connections between each input and the output. This architecture can extract the linearly predictable part early in the learning process and free up the nonlinear resources to be employed where they are really needed. It can be advantageous to choose different learning rates for different parts of the architecture, and thus not follow the gradient exactly (Weigend 1991).

A network that is to predict the future must know about the past. The simplest approach is to provide time-delayed samples as its inputs. A network without (nonlinear) hidden units is equivalent to an AR model (one linear filter). With nonlinear hidden units, however, the network can be viewed as combining a number of "squashed" filters.

A modification to the simple architecture shown in Fig. 12.6 is to replace each connection by an AR filter. Rather than displaying the explicit buffer of the input units, it suffices then to draw only a single input unit an conceptually move the weight vectors into the tapped delay lines from the input to each hidden unit. This is called a *time-delay neural network* by Lang, Waibel, and Hinton (1990) or (in the spatial domain) a network with *linked weights*, as suggested by le Cun (1989). Wan (1994) successfully used this architecture to predict the laser data and calls it an *FIR-network*. The sole difference to the network with simple weights is that in an FIR network the hidden units in the second layer have access to a stack of past values of activations of the units of the first hidden layer, enabling the network to find richer internal representations of time.

In some detail, Wan's network had one input unit, two layers of 12 hidden units

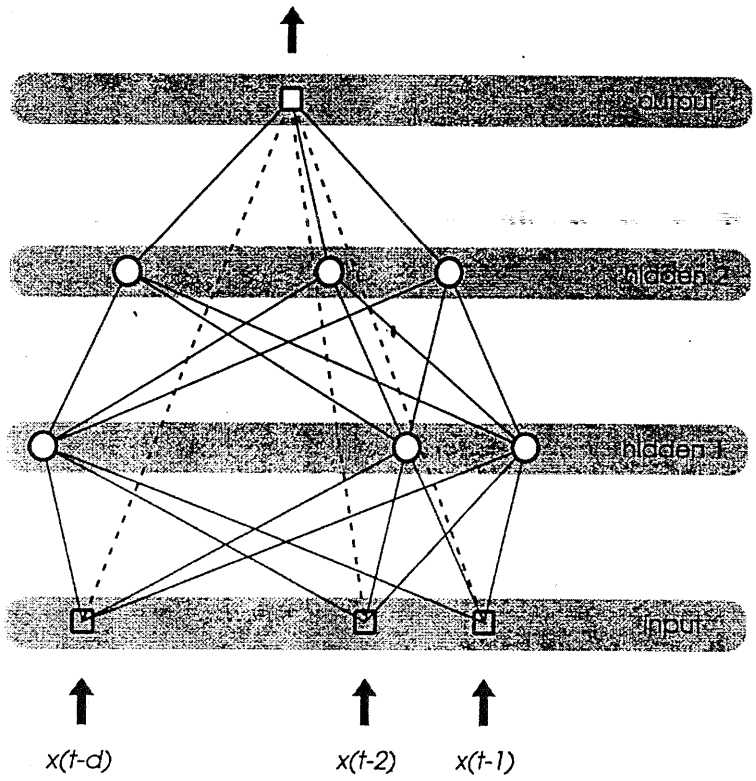



FIG. 12.6. Architecture of a feedforward network with two hidden layers and direct connections from the input to the output. The lines correspond to weight values. The dashed lines represent direct connections from the inputs to the (linear) output unit. Biases are not shown.

each, and one output unit. The generalized weights of the first layer were tapped delay lines with 25 taps; the second and third layers had 5 taps each. These values are not the result of a simple quantitative analysis, but rather the result of evaluating the performance of a variety of architectures on some part of the available data that Wan had set aside for this purpose. Such careful exploration is important for the successful use of neural networks.

At first sight, selecting an architecture with 1105 parameters to fit 1000 data points seems absurd. How is it possible not to overfit if there are more parameters than data points? The key is knowing when to stop. At the outset of training, the parameters have random values, and so changing any one coefficient has little impact on the quality of the predictions. As training progresses and the fit improves, the effective number of parameters grows (Weigend & Rumelhart 1991, Moody 1994, Weigend 1994). The overall error in predicting points out of the training set (cross-validation error) will initially decrease as the network learns to

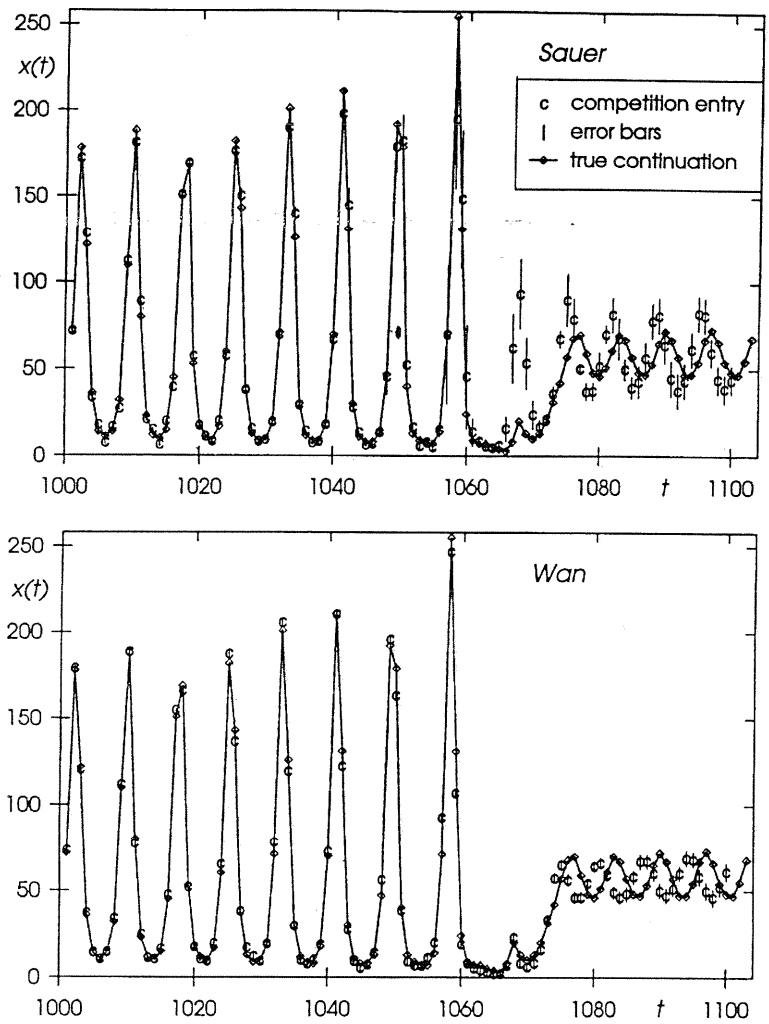



FIG. 12.7. The two best predicted continuations for the laser data of the Santa Fe competition, by Sauer (1994) and by Wan (1994). Predicted values are indicated by c, predicted error bars by vertical lines. The true continuation (not available at the time when the predictions were received) is shown in grey (the points are connected to guide the eye).

do something, but then will begin to increase once the network learns to do too much; the location of the minimum of the cross-validation error determines when the effective network complexity is appropriate. We will return to this issue in Secs. 5.4 and 5.5.

In Figs. 12.7 and 12.8, we compare the predictions for the laser data obtained with a local linear model (Sauer 1994) and with a feedforward network (Wan 1994).

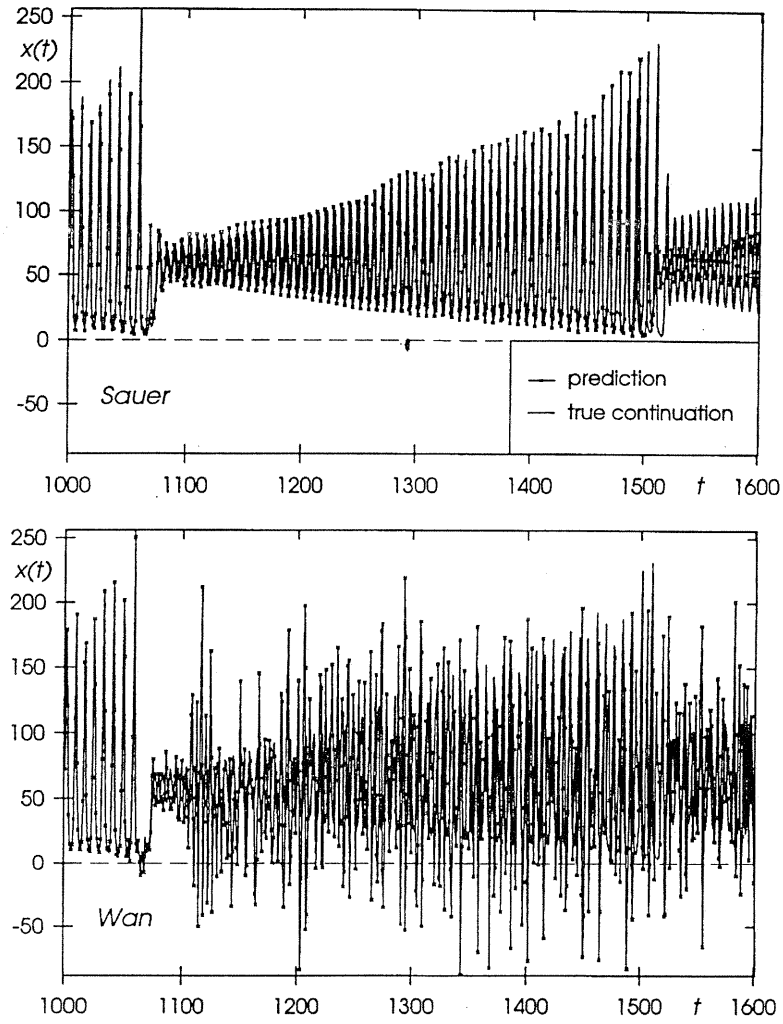



FIG. 12.8. Predictions obtained by the same two models as in the previous figure, but continued 500 points further into the future. The solid line connects the predicted points; the grey line indicates the true continuation.

The network did very well over the prediction interval of the first 100 points after the end of the training set (Fig. 12.7), but notice how it fails dramatically thereafter (Fig. 12.8) while the local linear forecast continues on. The reason for this difference is that the local linear model with local SVD (described at the end of the last section) is constrained to stay on the reconstructed surface in the embedding space (any point gets projected onto the reconstructed manifold before the prediction is made), whereas the network does not have a comparable constraint.

# 5. ISSUES IN CONNECTIONIST TIME SERIES MODELING

The first part of this section shows an explicit way of obtaining error bars of the predictions by a second output unit. We then introduce a representation that allows us to predict the evolution of the probability density. Next, we address the question of iterated versus direct predictions. We then briefly discuss some approaches to preventing overfitting, and close with some example of how an analysis of the trained network can help characterize the time series.

## 5.1. Predicting the Error Bars

So far we have discussed how to predict the continuation of a time series. It is often desirable and important to also know how sure we can be about a prediction. At the Santa Fe competition, none of the entries estimated the error bars (required for the laser data) in a principled way. Before presenting our solution to the problem (Nix & Weigend 1995), we make explicit the three assumptions that minimizing sum squared errors implies in a maximum likelihood framework (which we need not accept):

**Assumption 12.1.** The errors of different data points are independent of each other. Statistical independence is present if the joint probability between two events is precisely the product between the two individual probabilities. After taking the logarithm, the product becomes a sum. Summing errors thus assumes statistical independence of the measurement errors.

**Assumption 12.2.** The errors are Gaussian distributed. The error is the difference between the desired value (target) and the predicted value. The predicted value is a deterministic function of the input. (The function is given by the network parameters.) Given an input [and with the weights determined, this also means: given a (predicted) output], we now assume that some Gaussian noise was added to that (ideal) output to generate the (observed) target value. This probability distribution of the observation given the prediction, sometimes called the error model or the conditional target distribution (conditional because it depends directly on the prediction and indirectly on the input) is here assumed to be a Gaussian. Taking the logarithm of a Gaussian transforms it into a squared difference. This squared difference can be interpreted as a squared error.

**Assumption 12.3.** The errors are identically distributed; that is, the size of the error bar is assumed to not vary with the location in state space. (The errors are summed with the same weight for each data point.)

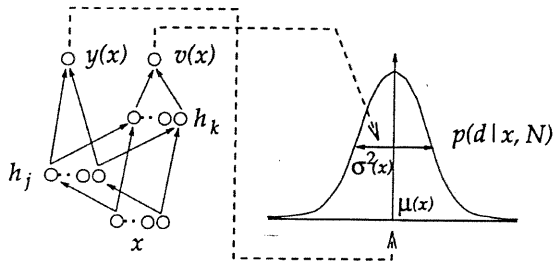



FIG. 12.9. Architecture of the network with two output units. The $y$ unit predictes the conditional mean (i.e., given the input) of the output distribution. The $v$ unit predicts the conditional variance of that distribution. All weight layers have full connectivity. This architecture allows the $v$ unit to access both information in the input pattern itself and in the hidden unit representation formed while learning $y(\boldsymbol{x})$.

The third assumption is clearly violated in the laser data: the error bars depend strongly on the location in state space (Fig. 12.3). This is not surprising, because the local properties of the attractor (such as the rate of divergence of nearby trajectories) vary with the location. Furthermore, different regions are sampled unequally since the training set is finite.

We now introduce our method for estimating error bars. We need to assumes a specific parameterized form of the conditional target distribution (this will be relaxed in the next section). We then estimate the local error bars by finding those parameters that maximize the likelihood of the data given the model.

Specifically, when we use a network output $y$ to approximate a function $f(\boldsymbol{x})$, we assume that the desired output ($d$) (i.e., the observed values) can be modeled by $d(\boldsymbol{x}) = f(\boldsymbol{x}) + n(\boldsymbol{x})$ where $n(\boldsymbol{x})$ is noise drawn from the assumed error model distribution. Just as the estimate of the mean of this distribution $y(\boldsymbol{x})$ is a function of the input, the variance $\sigma^2$ may also vary as a function of the location in input space $\boldsymbol{x}$. When the noise level varies over the input space (i.e., $\sigma^2(\boldsymbol{x})$ depends on $\boldsymbol{x}$; it is not a constant), not only do we want the network to learn an output function $y(\boldsymbol{x})$ that estimates the expectation value $\mu(\boldsymbol{x})$ of the conditional target distribution, but we also want to learn a function $v(\boldsymbol{x})$ that estimates the variance $\sigma^2(\boldsymbol{x})$ of that distribution.

Therefore, we simply add an auxiliary output unit, the $v$ unit, that computes $v(\boldsymbol{x})$, our estimate of $\sigma^2(\boldsymbol{x})$. Because $\sigma^2(\boldsymbol{x})$ must be positive, we choose an exponential activation function for $v(\boldsymbol{x})$ naturally impose this bound,

$$v(\boldsymbol{x}) = \exp\left[\sum_k w_{vk} h_k(\boldsymbol{x}) + \beta\right] \tag{30}$$

where $\beta$ is the offset (or bias), and $w_{vk}$ is the weight between hidden unit $k$ and the $v$ unit. This architecture is sketched in Fig. 12.9.

The network thus has two output units. For one of them, the $y$ unit, the target is easily available; it is simply given by $d$. But what is the target for the $v$ unit? We effectively invent a target by maximizing the likelihood of our network $\mathcal{N}$ given the data. Assuming statistical independence of the errors (this was Assumption 12.1), we equivalently minimize the negative log likelihood or cost

$$C = -\sum_i \log p(d_i \mid \boldsymbol{x}_i, \mathcal{N}) \tag{31}$$

Traditionally, the resulting form of the cost $C$ involves only the estimate $y(\boldsymbol{x}_i)$ of the mean of the assumed error model as a function of the input; the variance is assumed to be constant, and constant terms drop out after differentiation (see Chapter 15 this volume by Rumelhart, Durbin, Golden, & Chauvin). In contrast, we here allow the variance to depend on the input and explicitly keep these terms in $C$. Given any network architecture and any error model, the appropriate weight-update equations for gradient descent learning can be derived straightforwardly.

Let us illustrate this for the case where we assume that the deviations of the observed value from the mean are Gaussian distributed and the variance of the Gaussian is given by $v(\boldsymbol{x})$, that is, keeping Assumption 12.2. Assuming normally distributed errors around $y = f(\boldsymbol{x})$ corresponds to a conditional target probability distribution of

$$p(d_i \mid \boldsymbol{x}_i, \mathcal{N}) = \frac{1}{\sqrt{2\pi\, v(\boldsymbol{x}_i)}} \exp\left\{ \frac{-[d_i - y(\boldsymbol{x}_i)]^2}{2v(\boldsymbol{x}_i)} \right\} \tag{32}$$

where the network output $y(\boldsymbol{x}_i) \approx \mu(\boldsymbol{x}_i)$ estimates the mean, and $v(\boldsymbol{x}_i) \approx \sigma^2(\boldsymbol{x}_i)$ estimates the variance. By taking the negative logarithm, we obtain for, monotonically related negative log likelihood,

$$-\log p(d_i \mid \boldsymbol{x}_i, \mathcal{N}) = \frac{\log 2\pi\, v(\boldsymbol{x}_i)}{2} + \frac{[d_i - y(\boldsymbol{x}_i)]^2}{2v(\boldsymbol{x}_i)} \tag{33}$$

Dropping constant terms, summation over all patterns $i$ yields the total cost:

$$C = \sum_i \frac{1}{2} \left( \frac{[d_i - y(\boldsymbol{x}_i)]^2}{v(\boldsymbol{x}_i)} + \log[v(\boldsymbol{x}_i)] \right) \tag{34}$$

In order to write the explicit weight-update equations, we have to make assumptions about the transfer function of the units in the network. We here choose linear activation function for the $y$ unit, tanh activation functions for the hidden units, and an exponential activation function for the $v$ unit. We can then take derivatives of the cost $C$ with respect to the network weights. To update weights connected to the $y$ and $v$ units we have

$$\Delta w_{yj} = \eta[1/v(\boldsymbol{x}_i)][d_i - y(\boldsymbol{x}_i)]h_j(\boldsymbol{x}_i) \tag{35}$$

$$\Delta w_{vk} = \eta[1/2v(\boldsymbol{x}_i)]\{[d_i - y(\boldsymbol{x}_i)]^2 - v(\boldsymbol{x}_i)\}h_k(\boldsymbol{x}_i) \tag{36}$$

where $\eta$ is the learning rate. For weights not connected to the output, the weight-update equations are derived by using the chain rule in the same way as in standard backpropagation. Note that Eq. (12.36) is equivalent to training a separate function-approximation network where the targets for $v(\boldsymbol{x})$ are the squared errors. Note also that if $v(\boldsymbol{x}_i)$ is constant, Eqs. (12.35) and (12.36) reduce to their familiar forms for standard backpropagation with a sum-squared error cost function.

A variation of $v(\boldsymbol{x}_i)$ with $i$ implies a different weighting of each pattern. The $1/v(\boldsymbol{x})$ term in the update equations can be interpreted as a form of weighted regression, lowering the effective learning rate in high-noise regions. The effect is that the network tries hard to obtain small errors on those patterns where it can; it tries less hard on the patterns for which the expected error is going to be large anyway.

If the weighted regression term is allowed a significant influence early in gradient descent, local minima frequently result: the network consumes all its resources by trying hard to fit the first statistical feature it happens to encounter low errors on, discounting other patterns as being high error. To avoid premature weighting of different patterns [which would be based on inaccurate values of $v(\boldsymbol{x}_i)$ before $f(\boldsymbol{x})$ is at least roughly approximated by $y(\boldsymbol{x})$], we separate training into three phases:

- **Phase I (Mean):** Randomly split the available data into equal halves, sets $\mathcal{A}$ and $\mathcal{B}$. Learn the conditional expectation value $y(\boldsymbol{x})$ by using set $\mathcal{A}$ as training set. In phase I we use simple gradient descent on a simple squared-error cost function, that is, Eqs. (12.35) and (12.36) without the $1/v(\boldsymbol{x})$ terms.[5] To guard against overfitting, training is considered complete at the minimum of the squared error on the cross-validation set $\mathcal{B}$, monitored at the end of each complete pass through the data.
- **Phase II (Variance):** Attach a layer of hidden units connected to both the inputs and the hidden units of the network from phase I (see Fig. 12.9). Freeze the weights trained in phase I, and train the $v$ unit to predict the squared errors, again using simple gradient descent as in phase I. The training set for this phase is set $\mathcal{B}$, with set $\mathcal{A}$ used for cross validation; if set $\mathcal{A}$ were used as the training set in this phase as well, possible overfitting in phase I could result in seriously underestimating $v(\boldsymbol{x})$. To avoid this risk, we interchange the data sets. The initial value for the offset $\beta$ of the $v$ unit is the natural logarithm of the mean squared error (from phase I) of set $\mathcal{B}$. Phase II stops when the squared error on set $\mathcal{A}$ levels out or starts to increase.

[5] Further details are: all inputs are scaled to zero mean and unit variance. All initial weights feeding into hidden units are drawn from a uniform distribution $\in [1/i, -1/i]$ where $i$ is the number of incoming connections. All initial weights feeding into $y$ or $v$ are drawn from a uniform distribution $\in [s/i, -s/i]$ where $s$ is the standard deviation of the (overall) target distribution. No momentum is used, and all weight updates are averaged over 20 patterns.

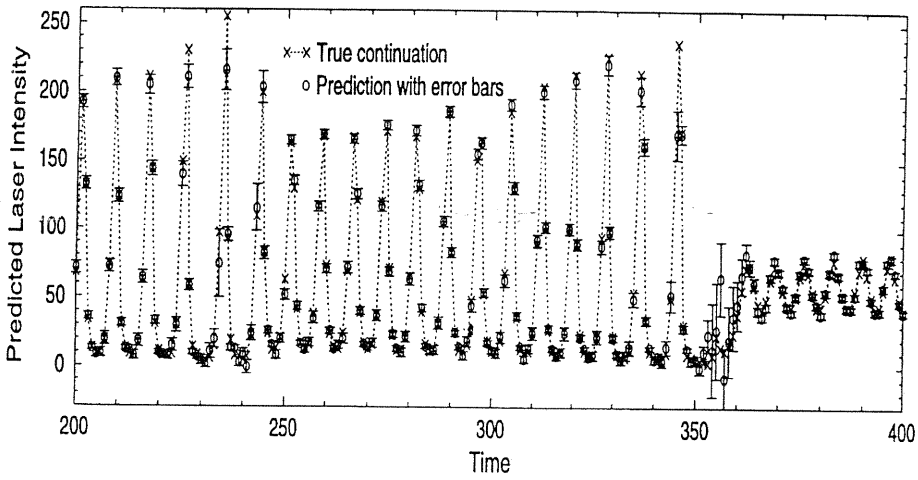



FIG. 12.10. Single-step predictions with error bars. The plotted error bars are ± the square root the value of the $v$ unit. Note the large errors and large error bars in the region of great uncertainty after the collapse.

- **Phase III (Weighted Regression):** Re-split the available data into two new halves, $\mathcal{A}'$ and $\mathcal{B}'$. Unfreeze all weights and train all network parameters to minimize the full cost function $C$ using Eqs. (12.35) and (12.36) and their chain-rule counterparts on set $\mathcal{A}'$. Training is considered complete when $C$ has reached its minimum on set $\mathcal{B}'$. Note that the $1/v(\boldsymbol{x})$ terms in Eqs. (12.35) and (12.36) implement a form of weighted regression, attenuating the learning in regions where the estimated $v(\boldsymbol{x})$ is high. The three-phase approach greatly reduces the probability of local minima by giving the composite network a head start on learning the function $f(\boldsymbol{x})$.

Figure 12.10 shows the application of this method to the laser data. The detailed choices of the parameters are described in Nix and Weigend (1995). We used the upsampling trick by Sauer (1994) (described at the end of the Sec. 4.1 on local models): the 1000 available data points were upsampled by a factor of 32. This does not change the effective sampling rate, but it fills in more points in the manifold. We then randomly split the resulting 31,200 data points into two equal-sized sets, $\mathcal{A}$ and $\mathcal{B}$, and proceed in the three phases. Figure 12.3 shows that it is a good idea to get local estimates of the uncertainty; different regions of state space can be predicted with different accuracies.

In summary, we started with the maximum likelihood principle and arrived at local error bars that depend on the location in input space. In any example, we have to choose a specific error model. The framework presented here encompasses any distribution with a location parameter (conditional mean) and a scale

parameter (local error bar). The framework also carries over from regression to classification where it allows to quantify the amount of uncertainty of a probability estimate (of a pattern belonging to a certain class) by giving the width of that distribution, again depending on the input pattern. This method encompasses most sources of uncertainty, such as uncertainty due to stochasticity (outside shocks, measurement noise), and uncertainty due to the divergence of nearby trajectories (particularly large after the collapses). It indirectly also handles model misspecification by appropriately overestimating the variance. It does not take into account the uncertainty of the predictor due to specific splits of the data into training, cross validation, and test sets. This last point will be discussed in Sec. 6.3 in the context of evaluating neural networks by using a bootstrap.

## 5.2. Predicting the Probability Density

In the previous section we presented our method for obtaining local error bars, that is, estimates of the confidence in the predicted value that depend on the input. We now show a connectionist method to estimate the entire conditional target distribution (Srivastava & Weigend 1994). Such nonparametric estimates of the shape of a conditional target distribution require large quantities of data.

We first introduce a representation appropriate for conditional probability distributions. The idea is to perform *fractional binning*, using the following procedure:

1. Partition the range space into $N$ approximately equal mass bins.
2. Compute the mean of the data points within each bin. In this manner, we can compute $N$ different bin centers: $\xi^{(1)} < \xi^{(2)} < \cdots < \xi^{(N)}$.
3. A given target point $d_i$ is now described by two adjacent nonzero activations (all of the other activations are zero). Between the two adjacent bins $j$ and $k = j + 1$, a weight of one ($c^{(j)} + c^{(k)} = 1$) is split such that

$$c_i^{(j)} \xi_i^{(j)} + c^{(k)} \xi_i^{(k)} = d_i \tag{37}$$

Following a suggestion by Tukey (personal communication, 1994), we call this representation fractional binning. We use this representation instead of ordinary hard binning because it avoids quantization errors.

In our experiments, we use a set of $N$ normalized exponentials (softmax). The architecture depends on whether the network is to only generate predictions one step ahead or whether it should be able to accommodate iterated predictions. For single-step predictions of the density, the network has $d$ inputs, each input corresponding to a component of the $d$-dimensional lag vector. If the network is required to perform iterated predictions (where the network output is fed back to the input), the input to the network is also a fractionally binned representation of the lag vector, so that the form of the output is identical to the form of the input. This is sketched in Fig. 12.11.

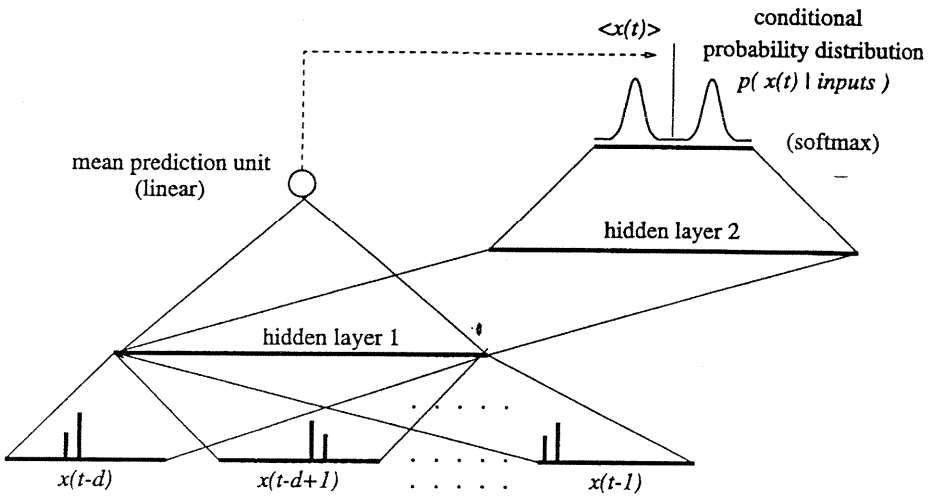



FIG. 12.11. Architecture of a network for iterated predictions of the density. The output unit on the left is to predict the mean of the distribution of the next time step, $\langle x_t \rangle = \hat{x}_t$. The fractional binning used for the set of output units on the right allows for multimodal predictions.

The network has two kinds of outputs. To get the learning started, it is useful to have a single output unit that predicts the mean. It is connected to the first hidden layer. The second set of output units consists of the $N$ units that describe the distribution in the fractionally binned representation, approximating the conditional probability distribution

$$p(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_{t-d}) \tag{38}$$

We added hidden layer 2 to allow for a nonlinear transformation between the internal representation of the mean in layer 1 to the fractional binned representation.

Figure 12.12 shows iterated predictions of the laser data for 1–12 time steps into the future. We used 13 bins both for the output and each of the three sets of inputs. Both hidden layers had 25 sigmoidal units. We divided the 25,000 data points available into three sets: a training set [12,000 points, even-numbered (the 1000 points given as training set in the Santa Fe competition were part of this set)], a cross-validation set (12,000 points, odd-numbered), and a test set (1000 points). The network was trained using backpropagation, minimizing sum squared errors.

We used stochastic teacher forcing to ease the network into iterated predictions. There are two extremes for what can be used at the inputs. One extreme is to always use the exact values at the input (giving the short-term errors at the output). The other extreme is to always use the predicted values at the input (long-term error). [There are a number of terms associated with this distinction. Engineers use the expression open loop when the inputs are set to true values and closed loop for the

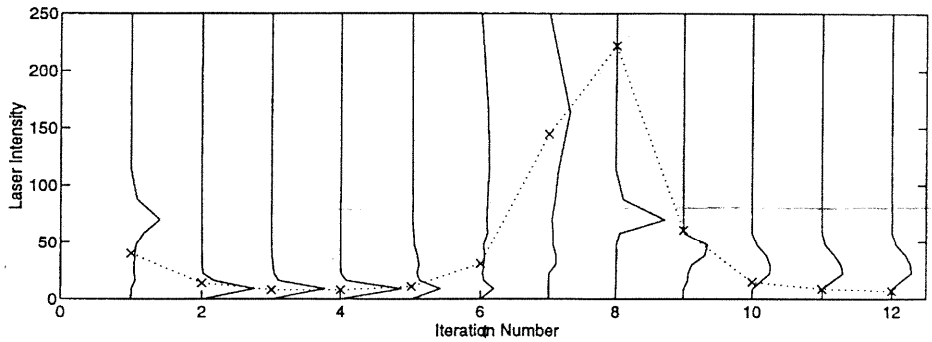



FIG. 12.12. Iterated predictions of the laser data. The vertical curves correspond to the output of network; the crosses indicate the target values. The dots connecting the targets are to guide the eye. Note the bimodal distribution at iterations 6 and 7; they could not have been obtained with a network that only predicts a mean and an error bar. Note also that at iteration 8, the network seems quite sure (although wrong) that the intensity will be around 60. At iteration 10 or 11, the network reaches a fixed point.

case when the input are given the predicted values, as well as equation error and output error for the two cases. In the connectionist community, the term teacher forcing is used when the inputs are set to the true values and the term trajectory learning (Principe, de Vries, & Oliveira 1993) when the predicted output is fed back to the inputs.] There is a continuum between these two extremes: in training, the errors from both sources can be combined in a weighted way, that is,

$$\lambda \times (\text{short-term error}) + (1 - \lambda) \times (\text{long-term error}) \tag{39}$$

This mixed error is then used in backpropagation. Since the network produces very poor predictions at the beginning of the training process, it can be advantageous to begin with $\lambda = 1$ (full teacher forcing) and then anneal to the desired value. This mixed error is revisited in the next section as $\lambda \times$ case $1 + (1 - \lambda) \times$ case 2 (defined in Table 12.1).

### 5.3. Iterated Versus Direct Predictions

Having discussed predicting error bars and the full distribution, we now return to the predictions of a single value. We approach the question "direct vs. iterated predictions?" first from a physics perspective and then from a connectionist perspective.

Physicists often view a time series as a sequence of measurements of some function of variables of a dynamical system. The key to the direct-vs.-iterated issue is to note that there are two time scales involved: the time scale of the dynamics of the system (e.g., the ups and downs of the laser intensity) and the time scale of the measurements.

more cells than values). If the probability distribution is more complicated, it can grow as $D_1 \log N$ where $D_1$ is a constant $\leq 1$. Therefore, the dependence of $H_1$ on $N$ provides information about the resolution of the observable. ($D_1$ is called the *information dimension*, an example of the generalized dimensions introduced by Hentschel & Procaccia 1983.)

The probability of seeing a specific lag vector $\boldsymbol{x}_t = (x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau})$ in $d$-dimensional lag space is similarly estimated by counting the relative population of the corresponding cell in the $d$-dimensional array: $p_d(x) = n_x/n_T$. The probability of seeing a particular sequence *of D* embedded vectors $(\boldsymbol{x}_t, \ldots, \boldsymbol{x}_{t-(D-1)\tau})$ is just $p_{d+D}(x_t, \ldots, x_{t-(d+D-1)\tau})$ because each successive vector is equal to the preceding one with the coordinates shifted over one place and a new observation added at the end. This means that the joint probability of $d$ delayed observations $p_d$ is equivalent to the probability of seeing a single point in the $d$-dimensional embedding space (or the probability of seeing a sequence of $1 + d - n$ points in a smaller $n$-dimensional space). In terms of $p_d$, the *joint entropy* or *block entropy* is

$$H_d(\tau, N) = -\sum_{x_t=1}^{N} \cdots \sum_{x_{t-(d-1)\tau}=1}^{N} p_d(x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) \times \log_2 p_d(x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) \tag{18}$$

This is the average number of bits needed to describe a sequence. The sums capture all the bins (or cells) of the $d$-dimensional space. (The ranges of the sums might seem strange at first sight, but keep in mind that we are assuming that $x_t$ is quantized to integers between 1 and $N$.)

The $d$ dependence of the entropy can be understood in terms of the concept of *mutual information*. The mutual information between two samples is the difference between their joint entropy and the sum of their scalar entropies,

$$\begin{aligned} I_2(\tau, N) &= -\sum_{x_t=1}^{N} p_1(x_t) \log_2 p_1(x_t) - \sum_{x_{t-\tau}=1}^{N} p_1(x_{t-\tau}) \log_2 p_1(x_{t-\tau}) \\ &\quad + \sum_{x_t=1}^{N} \sum_{x_{t-\tau}=1}^{N} p_2(x_t, x_{t-\tau}) \log_2 p_2(x_t, x_{t-\tau}) \\ &= 2H_1(\tau, N) - H_2(\tau, N) \end{aligned} \tag{19}$$

If the samples are statistically independent [this means by definition that the probability distribution factors, that is, $p_2(x_t, x_{t-\tau}) \equiv p_1(x_t)p_1(x_{t-\tau})$], then the mutual information will vanish: no knowledge can be gained for the second sample by knowing the first. On the other hand, if the first sample uniquely determines the second sample ($H_2 = H_1$), the mutual information will equal the scalar entropy $I_2 = H_1$. In between these two cases, the mutual information measures in bits the degree to which knowledge of one variable specifies the other. [Mutual information can be transformed onto a range from 0 to 1 [similar to that of the square of

the linear autocorrelation coefficient, Eq. (12.4)] by using $1 - \exp(-2I_2(\tau, N))$; see Granger and Teräsvirta (1993, p. 24).]

The mutual information can be generalized to higher dimensions either by the *joint mutual information*

$$I_d(\tau, N) = dH_1(\tau, N) - H_d(\tau, N) \tag{20}$$

or by the *redundancy* or *incremental mutual information*

$$R_d(\tau, N) = H_1(\tau, N) + H_{d-1}(\tau, N) - H_d(\tau, N) \tag{21}$$

The redundancy measures the average number of bits about an observation that can be determined by knowing $d-1$ preceding observations. Joint mutual information and redundancy are related by $R_d = I_d - I_{d-1}$.

For systems governed by differential equations or maps, given enough points and enough resolution, the past must uniquely determine the future[4] (to a certain horizon, which depends on the finite resolution). If $d$ is much less than the minimum embedding dimension, then the $d-1$ previous observations do not determine the next one, and so the value of the redundancy will approach zero,

$$\begin{aligned} p_d(x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) &= p_1(x_t)p_{d-1}(x_{t-\tau}), \ldots, x_{t-(d-1)\tau} \\ &\Rightarrow H_d = H_1 + H_{d-1} \Rightarrow R_d = 0 \end{aligned} \tag{22}$$

On the other hand, if $d$ is much larger than the required embedding dimension, then the new observation will be entirely redundant. [To be precise, $H_d = H_{d-1}$ is only valid for (1) the limit of short times $\tau$, (2) discrete measurements, and (3) the noise-free case.]

$$\begin{aligned} p_d(x_t, x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) &= p_{d-1}(x_{t-\tau}, \ldots, x_{t-(d-1)\tau}) \\ &\Rightarrow H_d = H_{d-1} \Rightarrow R_d = H_1 \end{aligned} \tag{23}$$

The minimum value of $d$ for which the redundancy converges (if there is one) is equal to the minimum embedding dimension at that resolution and delay time, that is, the size of the smallest Euclidean space that can contain the dynamics without trajectories crossing. Before giving the redundancy for the laser data of the competition (in Fig. 12.4), we provide relations to other quantities, such as the Lyapunov exponents and the prediction horizon.

The *source entropy* or *Kolmogorov–Sinai entropy* $h(\tau, N)$ is defined to be the asymptotic rate of increase of the information with each additional measurement

[4]Note that the converse remains true for differential equations but need not be true of maps; for example, neither of the two maps introduced in Sec. 2.2 has a unique preimage in time—they cannot be inverted. Given that most computer simulations approximate the continuous dynamics (1989) shows how discretization in time can create dynamical features that are impossible in the continuous-time systems; see also Grebogi, Hammel, Yorke, and Sauer (1990). Rico-Martinez, Kevrekidis, and Adomaitis (1993) discuss noninvertibility in the context of neural networks.

TABLE 12.1
Three Neural Networks for $h$-Step Ahead Predictions

| | *Case* | *Architecture* | *Objective* |
|---|---|---|---|
| 1 | Iterated | single step | minimize error after one iteration |
| 2 | Iterated | single step | minimize error after $h$ iterations |
| 3 | Direct | $h$-step | (minimize error for one $h$-step forecast) |

Consider the one extreme where the data are taken on a time scale much faster than the intrinsic dynamics of the system. An iterated predictor will try to capture variations that are primarily noise; the system does not change much from one time step to the next—only the noise does, because it typically has more high-frequency components than the signal. In this case, a direct predictor will be more suited, because its longer prediction time will be closer to the dynamics of the system. In the other extreme where the system is severely undersampled, the time scale of the iterated predictor is closer to the intrinsic time scale, and an iterated predictor might be more appropriate.

In order to emulate the system, however, the iterated predictor often requires higher functional complexity. This can be illustrated with the noise-free quadratic map, Eq. (12.13). On the one hand, for single-step predictions (subsequently to be iterated), all that is needed is a model that can represent a parabola as input–output behavior (an easy task for a neural network with a few hidden units). On the other hand, for direct $h$-step predictions, the function becomes a complicated polynomial of order $2h$. For example, $h = 10$-step predictions require a network with of the order of 1000 hidden units!

Table 12.1 classifies neural networks for $h$-step ahead predictions; the first and second network have the same number of hidden units. In all cases, the same lagged variables are used as inputs.

The first and second case both generate one-step ahead predictions at the output of the network. The output is used as an input in the next step; $h$ such iterations yield the desired $h$-step ahead forecast. The difference between the two cases is how they are trained:

- In the first case, the parameters of the network are optimized to minimize the error on one-step ahead forecasts. Subsequent iterating is an add-on, done after training.
- In the second case, the goal of network training is to minimize the error on the iterated $h$-step ahead forecasts. This can be viewed as putting $h$ copies of identical networks "on top of each other," that is, the second copy uses the output of the first as one of its inputs, and so forth. The parameters are then adjusted to have the smallest error after the $h$ iterations, that is, at the top of the unfolded network (Rumelhart, Hinton, & Williams 1986).

- The third network has a different task: it projects directly from the present to the desired point $h$ steps in the future; it does not involve any intermediate predictions or feed-back.

The heuristics just given address the case with noise on a timescale different from that of the dynamics. For deterministic chaotic systems (noise free), Farmer and Sidorowich (1988) argue that iterated forecasts lead to better predictions than direct forecasts. Fraser (personal communication 1993) points out that this argument can be traced back to Rissanen and Langdon (1981).

## 5.4. Preventing Overfitting

Overfitting is observed during training when the out-of-sample performance gets worse while the in-sample performance is still getting better, that is, the network extracts more features that do not generalize well (idiosyncrasies of the training set) than features that do generalize well. The severity of the effect depends on the noise level of the data and on the training set size. A notoriously hard case is financial time series where overfitting is a very serious problem. Figure 12.13 gives a typical example.

Weigend, Rumelhart, and Huberman (1991) discuss three approaches to prevent overfitting and apply them to the prediction on time series:

- **Early stopping.** The simplest algorithm is to use an oversize network with about as many weights as data points, to monitor the out-of-sample performance and to stop when the out-of-sample performance stops improving. The network is initialized with very small weights: large enough to break the symmetry but small enough to keep the hidden sigmoids in their linear range. The weights grow as the network learns. In this sense, training time can be viewed as a complexity term that penalizes weights according their size, strongly at first, and later relaxes. [Early stopping can be motivated by an analysis of the principal components of the hidden units as a function of training time (Weigend & Rumelhart 1991, Weigend 1994). It turns out that the principal components are extracted sequentially, so stopping early means that some eigenvalues are already fully developed whereas others are still dormant. The hope is that the large principal components correspond to the signal and the still-dormant ones to the noise. This sequential appearance of new principal components as training proceeds can also be interpreted as the network breaking more and more symmetries in the weights.]
- **Add noise to inputs**. A simple alternative is to add noise to each input at each iteration of backpropagation. That prevents the weights from getting too precise. Bishop (in press) shows that adding noise is equivalent to Tikhonov regularization.
- **Penalize complexity**. Inspired by the information-theoretic principle of minimum description length (Wallace & Boulton 1968, Rissanen 1986; see also

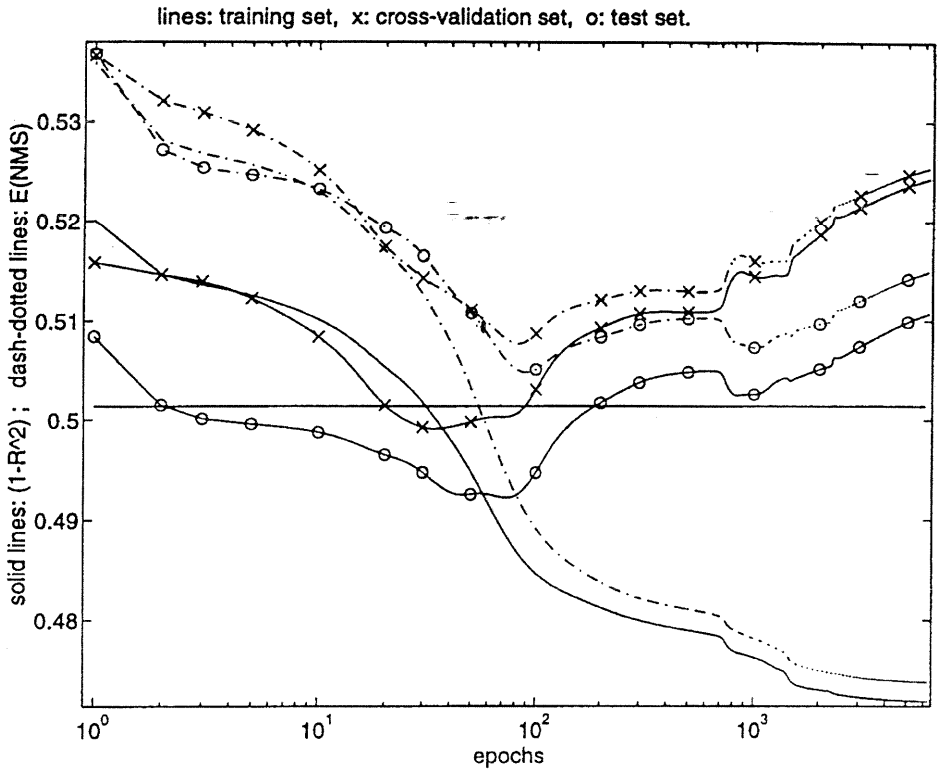



FIG. 12.13. Example of learning curves for predicting the trading volume of the New York Stock Exchange. There are three pairs of curves. The first pair (monotonically decreasing) gives the performance on the training set, the second pair (denoted by $\times$) on the cross-validation set, and the third pair ($\circ$) on the test set. The three solid lines plot the $(1 - R^2)$ measure; the three dash-dotted lines give the normalized mean squared error $E_{\mathrm{NMS}}$. The cost function was simple squared error (no momentum, no complexity term).

Chapter 16 in this volume, by Rissanen), we add to the error term in the cost function a second term penalizing network complexity. In a Bayesian framework, such a term is interpreted as describing a prior, that is, our belief of the distribution from which the weights are drawn. We suggested a weight-elimination term of the form

$$\sum_i \frac{(w_i/w_0)^2}{1 + (w_i/w_0)^2} \tag{40}$$

and applied it to sunspots predictions with good results; the final network had three hidden units and did not show overfitting. [The sum over $i$ extends over all the connections (excluding the biases). Here, $w_0$ is the scale of

the weights; it can be chosen differently for different parts of the network. Any such sum over individual weights does not take interactions between the weights into account. Weight elimination is discussed in more detail in Chapter 15 this volume, by Rumelhart, Durbin, Golden, & Chauvin.]

The sunspots time series has served as an example for a number of algorithms trying to produce small networks, for example, soft weight sharing (Nowlan & Hinton 1992) and optimal brain damage (developed by le Cun, Denker, & Solla 1990 and applied to the sunspots series by Svarer, Hansen, & Larsen 1993). The latter method is an example of deleting unimportant weights (pruning) based on the Hessian. Other pruning algorithms derive the significance of a weight from the ratio of the size of the weight to the standard deviation of its pattern-by-pattern changes, that is, the square root of the variance of the $\Delta ws$ in one pass through the data in backpropagation (Zimmermann 1994). In general, it seems useful to combine some of the methods, for example, to first use early stopping (i.e., to train to a minimum on a cross-validation set) and then crank up one of the complexity penalizers or pruning algorithms.

For very noisy problems, these broad techniques are not always powerful enough; specific (and hopefully correct!) assumptions about the data are needed to guide the search of the network through weight space. In some cases, these assumptions can be embodied in the architecture as hard constraints. FIR networks or time-delay neural networks (discussed in Sec. 4.2) are examples; linking weights embodies the notion of translational invariance.

A technique can also work when beliefs cannot be incorporated into the architecture is the use of pseudodata. The idea is best explained through an example. Let us assume that we believe that the response of the network to a certain pattern should be the same also when the pattern is slightly compressed or stretched in time. Backpropagation is an iterated procedure; at each presentation of a training pattern, we allow for some stretching or compressing of the pattern by repeating or dropping an observation of the time series with a certain probability. The network thus learns to also recognize stretched and compressed versions of the training patterns. It will subsequently generalize better on the test set if this belief about the data is actually correct.

The method of pseudodata can be compared to giving hints to the network. Abu-Mostafa (1994) gives the example of the symmetry hint for predicting foreign exchange data. This hint corresponds to viewing exchange rate returns first from one country, then from the other country, and the hint suggests that the dynamics should be the same. During training, the cost functions switches back and forth between gradient descent in the usual cost function (i.e., learning to predict the returns) and learning the hint (i.e., to minimize the difference in response to a pattern and to the flipped version of the pattern). The difference between hints and pseudodata is: any input vector can be used to descend on the hint, whereas pseudodata tend to stay close to the actual data.

Another method, used in Weigend, Huberman, and Rumelhart (1992) is to give additional tasks to the network that help constrain the hidden unit representation. For foreign exchange rates, examples are to add one or more output units, for example, for the sign and for the absolute value of the next-day returns.

## 5.5. Analyzing the Network

In Sec. 2.1 we showed that a linear time system is fully characterized by its Fourier spectrum (or equivalently by its ARMA coefficients or its autocorrelation function). We then showed how we have to go beyond that in the case of nonlinear systems and focused in Sec. 3.2 on model-free properties of the observed points directly in embedding space and in Sec. 4.1 on the indirect analysis of the time series by observing the behavior of out-of-sample errors as we varied parameters of local linear models. We now give examples of how a trained network can yield some insight into the process that generated the time series. We first show how it is possible to extract characteristic properties, such as the minimal embedding dimension and the manifold dimension from the network, then indicate how the network's emulation of the system can be used to estimate Lyapunov coefficients, and, finally, discuss how the network can shed some light on the amount of nonlinearity of the process, relating the performance back to an information theoretic concept, the entropy rate of a linear process.

*Dimensions.* In the early days of backpropagation, networks were trained with varying numbers of hidden units and the "final" test error (when the training had converged) was plotted as a function of the number of hidden units: it usually first drops and then reaches a minimum; the number of hidden units when the minimum is reached can be viewed as a kind of measure of the degrees of freedom of the system. A similar procedure can determine a kind of embedding dimension by systematically varying the number of input units.

Problems with this approach are that these numbers can be strongly influenced by the choice of the activation function and the search algorithm employed. First, if (for example) sigmoids are chosen as the activation function, we obtain the manifold dimension as expressed by sigmoids, which is an upper limit to the true manifold dimension.[6] Second, a small size of the bottleneck layer can make the search (via gradient descent in backpropagation) hard; overfitting even occurs for

[6]To reduce the dependence on the specific choice of the activation function, Saund (1989) suggested, in the context of nonlinear dimensionality reduction, to sandwich the hidden layer (let us now call it the central hidden layer) between two (large) additional hidden layers. An interpretation of this architecture is that the time-delay input representation is transformed nonlinearly by the first encoding layer of hidden units; if there are more hidden units than inputs, it is an expansion into a higher dimensional space. The goal is to find a representation that makes it easy for the network subsequently to parameterize the manifold with as few parameters as possible (done by the central hidden layer). The prediction is obtained by linearly combining the activations of the final decoding hidden layer that follows the central hidden layer.

small networks, before they have reached their full potential (Weigend 1994). There are two approaches to this problem: to penalize network complexity or to use an oversize network and analyze it. We have mentioned some complexity penalizing and/or pruning approaches in the last section; with luck, they give a minimal network that bears some resemblance to physical properties of the series. The alternative is to train a network without penalty terms and analyze it. The idea is to use a large network that easily reaches the training goal (and also easily overfits). The spectrum of the eigenvalues of the covariance matrix of the (central) hidden unit activations is computed as a function of training time. The covariance

$$C_{ij} = \langle (f_i - \overline{f}_i)(f_j - \overline{f}_j) \rangle \tag{41}$$

describes the two-point interaction between the activations of the two hidden units $i$ and $j$. Here, $\overline{f}_i = \langle f_i \rangle$ is the average activation of hidden unit $i$. The number of significantly sized eigenvalues of the covariance matrix (its effective rank) can serve as a measure of the effective dimension of the hidden unit space.

All of these approaches have to be used with caution as estimates of the true dimension of the manifold. We have already pointed out that the estimate can be too large (e.g., if the sigmoid basis functions are not suitable for the manifold or if the network is overfitting). But it can also be too small (e.g., if the network has essentially learned nothing), as often is the case for financial data (either because there is nothing to be emulated or because the training procedure or the architecture was not suited to the data).

*Lyapunov exponents.* It is notoriously difficult to estimate Lyapunov exponents (defined in Sec. 3.2) from short-time records of noisy systems. The hope is that if a network has reliably learned how to emulate such a system, the exponents can be found through the network. Weigend, Huberman, and Rumelhart (1990) give an example of how the divergence rate of a chaotic process can be estimated by analyzing the out-of-sample errors as a function of prediction time. Alternatively, the Jacobian can be computed from the network and averaged over the data points to obtain global Lyapunov exponents (Gencay & Dechert 1992; Nychka, Ellner, McCaffrey, & Dallart 1992).

Often, the Lyapunov coefficients strongly depend on the location in state space. In such cases, averaging over the attractor smears out potentially interesting information. The laser data is such an example; grey-scale coding the largest local Lyapunov coefficient gives a figure similar to Fig. 12.3.

*Nonlinearity.* DVS plots (Fig. 12.5) analyze the error as a function of the nonlinearity of the model (smaller neighborhoods $\Rightarrow$ more nonlinear). Also for neural networks, it is a good idea to compare the out-of-sample error of the network to the out-of-sample error of a linear model.

Furthermore, we can use a property of the network to characterize the amount of nonlinearity by analyzing the distribution of the activations $f$ of sigmoidal hidden

units. The ratio of the quadratic part of the Taylor expansion of a sigmoid with respect to the linear part, that is, the ratio of the second derivative over the first derivative, can be expressed in terms of the network quantities

$$|f''(\xi)|/|f'(\xi)| = a|1 - 2f| \tag{42}$$

Here $f'$ and $f''$ denote the first and second derivatives of the sigmoid activation function

$$f\left(\xi_h^{(t)}\right) = \frac{1}{1 + e^{-a\xi_h^{(t)}}} = \frac{1}{2}\left(1 + \tanh\frac{a}{2}\xi_h^{(t)}\right) \tag{43}$$

where $a$ denotes the slope of the sigmoid, and

$$\xi_h^{(t)} = \sum_{i=1}^{d} w_{hi} x_i^{(t)} + b_h$$

the net input. Weigend, Huberman, and Rumelhart (1990) show the distribution of this statistic (averaged over patterns $t$ and hidden units $h$) for a few time series, exhibiting different degrees of nonlinearity for the different time series.

Once the importance of nonlinearities has been established for a given problem, interesting questions are: Where do the nonlinearities appear? What are they used for? We have encountered time series problems where the sole use of nonlinearities of the network could be removed by preprocessing each input individually, that is, the network could be reduced to a linear superposition of individually transformed inputs:

$$\text{output } y = \sum_i w_i f_k(x_i) \tag{44}$$

We call these nonlinearities of the first kind that can be preprocessed away *preprocessing nonlinearities*. There are also cases, however, where interactions between the inputs are indeed crucial; the laser data set falls into that category. We call those nonlinearities of the second kind *interaction nonlinearities.*

The importance of interaction nonlinearities compared to preprocessing nonlinearities can be captured by the following measure. Compute the Hessian matrix of second derivatives of the network output with respect to the inputs $i$ and $j$,

$$\frac{\partial^2 y}{\partial x_i \, \partial x_j} \tag{45}$$

and evaluate it at the given data points (empirical density). If, on the one hand, the magnitude of the off-diagonal terms (e.g., the sum of their squares) is negligible compared to the main-diagonal terms, the network uses only preprocessing nonlinearities. If, on the other hand, the magnitude of the off-diagonal terms is large, the network indeed makes use of interactions between the input variables.

We close this section by bridging back to the section on information theory. In Sec. 3.2, we established the source entropy rate of a series as the lower bound

for any model. For a linear model, the entropy rate is given by (Cover & Thomas 1991, p. 274)

$$\frac{\log 2\pi e}{2} + \frac{1}{4\pi}\int_{-\pi}^{\pi} \log S(\lambda)\, d\lambda \tag{46}$$

where $S(\lambda)$ is the power spectral density; that is, the Fourier transform of the autocorrelation function (see Sec. 2.1 for a discussion why a linear model is completely described by its spectrum or, equivalently, by its autocorrelation function). In any given time series problem, it is interesting to see where a predictor falls between these two limits.

## 6. EVALUATING FORECASTS

In this section we discuss the importance of a proper evaluation of a predictor. We first list some standard performance measures and then discuss the uncertainty introduced by splitting the available data into a training, cross validation, and test set.

### 6.1. Evaluating Predictions Without Error Bars

The most basic measure of prediction accuracy uses only the predicted values $\hat{x}_k$ (in addition to the observed values $x_k$). We define the normalized mean squared error

$$E_{\mathrm{NMS}} = \frac{\sum_{t\in\mathcal{T}}(\mathrm{target}_t - \mathrm{prediction}_t)^2}{\sum_{t\in\mathcal{T}}(\mathrm{target}_t - \mathrm{mean}_{\mathcal{T}})^2} \approx \frac{1}{\hat{\sigma}_{\mathcal{T}}^2}\frac{1}{N}\sum_{t\in\mathcal{T}}(x_t - \hat{x}_t)^2 \tag{47}$$

where $t = 1, \ldots, N$ enumerates the points in the withheld test set $\mathcal{T}$, and $\mathrm{mean}_{\mathcal{T}}$ and $\hat{\sigma}_{\mathcal{T}}^2$ denote the sample mean and variance of the observed values (targets) in $\mathcal{T}$. It can be seen from the definition that simply predicting the overall mean (as done in the denominator) gives a value of $E_{\mathrm{NMS}} = 1$.

It is useful to relate $E_{\mathrm{NMS}}$ to the correlation coefficient $R$ between prediction and target. Let $x_t = \rho\hat{x}_t + \epsilon_t$; $\epsilon_t$ is the forecast error. For the variances we obtain $\sigma_x^2 = \rho^2\sigma_{\hat{x}}^2 + \sigma_\epsilon^2$. Assuming that the variances of $x_t$ and $\hat{x}_t$ are equal gives

$$1 - \rho^2 = \sigma_\epsilon^2/\sigma_x^2 \tag{48}$$

If the means of $x_t$ and $\hat{x}_t$ are also close, then $\rho$ can be approximated by the correlation coefficient between forecast and target, $R$.

When plotting both $(1 - R^2)$ and $E_{\mathrm{NMS}}$ on the same scale (e.g., as a function of training time to monitor overfitting), $E_{\mathrm{NMS}}$ is lower bounded by $(1 - R^2)$. As an example, Fig. 12.13 shows the learning curves for a training, a cross-validation set an a test set, both in terms of $(1 - R^2)$ and $E_{\mathrm{NMS}}$. [We wanted to use a data set that lies somewhere between simple noise-free function fitting and a sequence of true random numbers where no model has a chance. We picked the task of predicting

daily trading volume on the New York Stock Exchange, from 1962 until the 1987 stock market crash, from past volume, past returns and their absolute values, and past volatility (Weigend & LeBaron 1994).]

For data close to random walk, a good measure to assess the quality of the predictions is the ratio of squared errors,

$$\frac{\sum_{t\in\mathcal{T}}(\text{target}_t - \text{prediction}_t)^2}{\sum_{t\in\mathcal{T}}(\text{target}_t - \text{observation}_{t-1})^2} \tag{49}$$

The denominator uses the last observed value as prediction, which is the best that can be done for a random walk. A ratio above 1.0 thus corresponds to a prediction that is worse than random walk; a ratio below 1.0 is an improvement over a random walk. The random walk model used here for comparison is a weak null hypothesis. If there are significant autocorrelations (which also generalize out-of-sample), a low-order AR model can be tried as a stronger null hypothesis.

It is often useful to evaluate more information of the errors than just their mean. The distribution of the errors sorted by their size helps distinguish between forecasts that have the same average error but very different distributions (uniformly medium-sized errors versus mostly very small errors, along with a few large outliers). Other basic tests plot the prediction errors against the true (or against the predicted) value. This distribution should be flat if the errors are Gaussian distributed and proportional to the mean for a Poissonian error distribution. The time ordering of the errors can also contain information: a good model should turn the time series into structureless noise for the residual errors; any remaining structure indicates that the predictor missed some features or extracted features that do not generalize. It can be interesting to plot the residuals' autocorrelation coefficients against the number of training epochs and see how it relates to the out-of-sample error (overfitting).

## 6.2. Evaluating Predictions with Error Bars

If predicted error bars $\hat{\sigma}_k$ are available, we can use as performance measure the likelihood of the observed data, given the predicted values and the predicted error bars. In every case, an error model has to be assumed. Although the commonly made assumption of independent Gaussian errors assumption may not be correct, it provides a simple form that captures some desirable features of error weighting. In the example of the laser data, the original data is quantized to integer values. The probability of observing a given point $x_t$ is found by integrating the Gaussian distribution over a unit interval (corresponding to the rounding error of 1 bit of the analog-to-digital converter) centered on $x_t$,

$$p(x_t \mid \hat{x}_t, \hat{\sigma}_t) = \frac{1}{\sqrt{2\pi\hat{\sigma}_t^2}} \int_{x_t-0.5}^{x_t+0.5} \exp\left(-\frac{(\xi - \hat{x}_t)^2}{2\hat{\sigma}_t^2}\right) \tag{50}$$

If the predicted error is large, then the computed probability will be relatively small, almost independent of the value of the predicted point. If the predicted error is small and the predicted point is close to the observed value, then the probability will be large, but if the predicted error is small and the prediction is not close to the observed value, then the probability will be very small. The potential reward, as well as the risk, is greater for a confident prediction (small error bars). Under the assumption of independent errors, the likelihood of the whole test for the observed data given the predicted values and the predicted error bars is then

$$p(D \mid M) = \prod_{t=1}^{N} p(x_t \mid \hat{x}_t, \hat{\sigma}_t) \tag{51}$$

Finally, we take the logarithm of this probability of the data given the model (this turns products into sums and also avoids numerical problems), and then scale the result by the size of the data set $N$. This defines the *negative average log likelihood*,

$$-\frac{1}{N}\sum_{t=1}^{N} \log p(x_t \mid \hat{x}_t, \hat{\sigma}_t) \tag{52}$$

## 6.3. Boostrapping the Data

A standard approach of model selection is to split the data in a training, a cross validation, and a test set. In evaluating the performance of an architecture, one question is: how much does the performance vary with different splits? Because there is no unique way of splitting the data, we trained some 2500 networks each on different random splits of the available data (on the same task of predicting the volume of the New York Stock Exchange as in Fig. 12.13). We also drew randomly most network parameters (number of hidden units, initial weights, etc.; see Weigend & LeBaron 1994). For each network, we read off the test-set error at the minimum of the cross-validation error, and we histogram those errors (in terms of $1 - R^2$) in Fig. 12.14.

Figure 12.14 compares the performance histogram of the networks trained on different splits (solid line) with networks trained on a single split of the data where the only variation is due to the network parameter (dotted line). It turns out that the randomness in the splitting of the data generates significantly more variability than the randomness in network initialization and architecture does. Note that the network manages to explain about 50% of the variance of the data, at first sight a reasonable amount. It turns out, however, that the performance is very close to that of linear models, obtained on the same (test + cross validation) data by matrix inversion and tested on the same test data for each split (dashed line): the network does not manage to extract nonlinearities from the entire interval that generalize through the entire interval. This might be either because there are none, or because the nonlinearities are nonstationary, that is, vary over time, and thus get averaged out by this bootstrap procedure. Further steps here are to train on the residuals

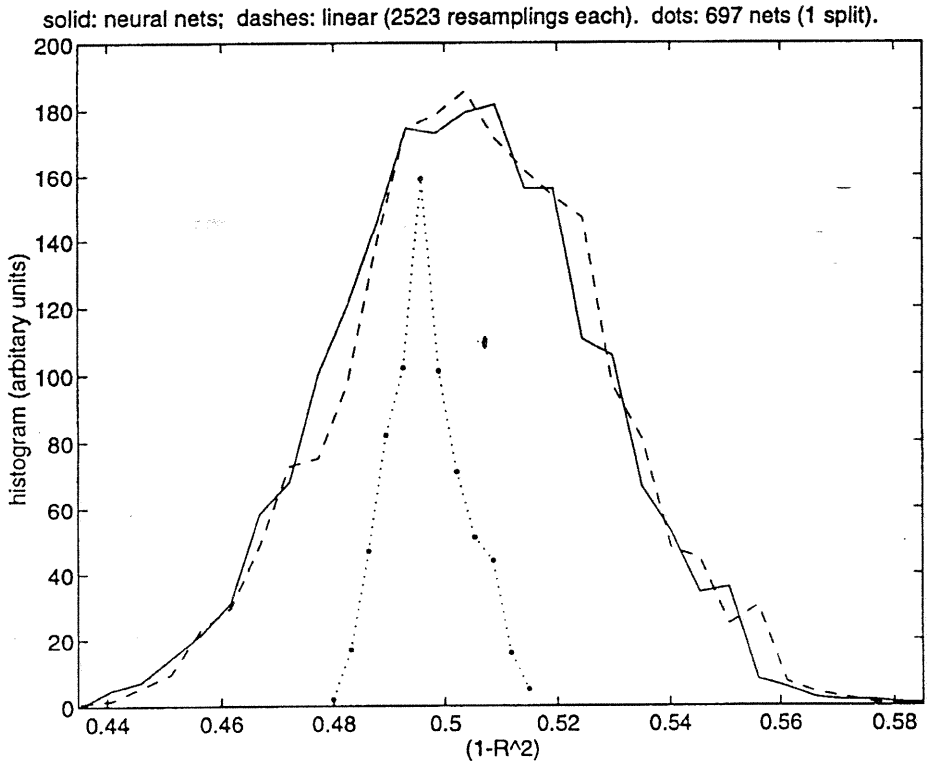



FIG. 12.14. Histograms of $(1 - R^2)$ forecast performance. The solid line shows the distribution of the networks, the dashed lines of linear model, both estimated on 2523 different resamplings of the available data. The dotted line takes just one split of the data and describes the distribution of 697 networks. The fact that the width of the dotted histogram is a lot smaller than the width of the other two indicates that the randomness in the splitting of the data generates more variability than the randomness in network initialization does.

from the linear fit (any improvement then is then due to nonlinear structure), to use pruning techniques, and to explore the effect of other output representations, such as fractional binning and error-correcting codes. Furthermore, we allow for non-stationarity and structure breaks by training (and cross validating) on only a subset contiguous in time, and evaluating the performance on the period immediately following the training/cross-validation period (roll-forward test).

## 7. THE FUTURE

We have surveyed the results of what appears to be a steady progress of insight over ignorance in analyzing and predicting time series. One of the underlying threads has been to find the best model for the data. A natural definition of best is the model

that requires the least amount of information to describe it. This is exactly the aim of Algorithmic Information Theory, independently developed by Chaitin (1966), Kolmogorov (1965), and Solomonoff (1964). Classical information theory, described in Sec. 3.2, measures information with respect to a probability distribution of an ensemble of observations of a string of symbols. In algorithmic information theory, information is measured within a single string of symbols by the number of bits needed to specify the shortest algorithm that can generate them. This has led to significant extensions of the results by Gödel (1931) and Turing (1936) (see Chaitin 1990), and through the minimum description length principle, it has been used as the basis for a general theory of statistical inference (Wallace & Boulton 1968, Rissanen 1987). There can be no universal algorithm to find the shortest program to generate an observed sequence, however, because we cannot determine whether an arbitrary candidate program will continue to produce symbols or will halt (e.g., see Cover & Thomas 1991, p. 162).

Although there are deep theoretical limitations on time series prediction, the constraints associated with specific domains of application can nevertheless permit strong useful results and can leave room for significant future development. In this chapter, we have ignored many of the most important time series problems that will need to be resolved before the theory can find widespread application, including the following.

*Building parameterized models*: For example, we have shown how the stationary dynamics of the laser can be correctly inferred from an observed time series, but how can a family of models be built as the laser pump energy is varied in order to gain insight into how this parameter enters into the governing equations? The problem is that for a linear system it is possible to identify an internal transfer function that is independent of the external inputs, but such a separation is not possible for nonlinear systems. A parametrized network allows us to characterize the system in terms of its qualitative behavior (Kevrekidis, Rico-Martinez, Ecke, Farber, & Lapedes 1993). Will it be possible to prove a theorem similar to Takens' theorem, but not for past values of the observable but for past values of the parameters?

*Controlling nonlinear systems*: How can observations of a nonlinear system and access to some of its inputs be used to build a model that can then be used to guide the manipulation of the system into a desired state? The control of nonlinear systems has been an area of active research; approaches to this problem include both explicit embedding models (Hübler 1989, Ott, Grebogi, & Yorke 1990, Bradley 1992) and implicit connectionist strategies (Miller, Sutton, & Werbos, Eds. 1990; White & Sofge, Eds. 1992).

*Analyzing systems that have spatial as well as temporal structure*: The transition to turbulence in a large-aspect-ratio convection cell is an experimental example of a spatio-temporal structure (Swinney 1994), and cellular automata and coupled map lattices have been extensively investigated to explore the theoretical relationship between temporal and spatial ordering (Gutowitz 1991). A promising

approach is to extend time series embedding (in which the task is to find a rule to map past observations to future values) to spatial embedding (which aims to find a map between one spatial, or spatio-temporal, region and another). Unfortunately, the mathematical framework underlying time-delay embedding (such as the uniqueness of state-space trajectories) does not simply carry over to spatial structures.

More modestly—narrowing the scope to neural networks for time series prediction—some examples of recent work are as follows.

*Temporal Difference learning (TD)*: The TD algorithm (Sutton 1988) has traditionally been used for Markovian and gamelike situations (Tesauro 1992). We applied TD learning to the prediction of real-value time series, such as the laser data (Kazlas & Weigend 1995). In some cases the network trained with TD (where the target is the prediction made at the next time step) outperforms the network trained with standard supervised learning (where the target is always the observed value).

*Cleaning the data*: Throughout this chapter, we have assumed that there is no noise in the inputs. In time series prediction, where lagged values are used at the input, this is clearly an inconsistent assumption. How can we exploit the fact that we are dealing with time series as opposed to variables that have nothing to do with each other? First, we can use a connectionist two-sided filter: a network learns to predict the central value of a window sliding across a time series from both a few preceding and following values. First, denote the original values by $x_t$ and the output values of that network by $o_t$. We now train the final prediction network on a new series $(1 - \lambda)x_t + \lambda o_t$. For $\lambda = 0$, it reduces to not performing any cleaning. The key is that by using $0 < \lambda < 1$, the points in the input plane (as well as the target) move to some hopefully cleaner values. Second, we use the prediction network itself by backpropagating the activations through the network and moving the (input) data toward those values that would have generated the target output (Weigend & Zimmermann 1994).

*Experts for prediction*: A Gaussian mixture model (Jacobs, Jordan, Nowlan, & Hinton 1991) can be used for the prediction of time series that exhibit regime switching and structure breaks: the gating network learns to partition the data between competing subnetworks according to similarities in the internal dynamics. After learning, the subnetworks identify the different dynamical regimes. This estimation of the hidden parameters allows us to uncover structure on different time scales, particularly on scales much slower than the sampling time. Inspired by Cacciatore and Nowlan (1994), we gave the gating network some recurrence, acting as capacitance to match the expected time scale of the gating output (Mangeas & Wiegend 1994).

*Multivariate and nonstationary time series*: Throughout this chapter, we have considered only univariate time series, that is, a single variable as a function of time. Takens' (1981) theorem (Sec. 3.1) assured us that past values of a single observable suffice to reconstruct the local properties of the solution manifold, if

we are dealing with a noise-free dynamical system. When dealing with real-world problems, this is less often the case as one might hope. Often, however, more than a single observable is measured. It is thus possible to build models with lagged values of several time series as inputs. First, taking additional information into account can make the problem appear more stationary. Second, in some cases, there are sets of variables that wander as a group. In the econometric literature, this property is called *cointegration* (Engle & Granger 1987). Using currency exchange rates as an example, this information can be exploited by first estimating a linear vector-autoregressive (VAR) model with one lag only on several foreign exchange price series. Because this is a very inflexible model, no overfitting occurs.) Then, a network is trained on the residuals from that fit. In the worst case (i.e., if the VAR matrix is an identity matrix) this reduces to training the network on the individual returns. Any improvement beyond this standard approach is due to the exploitation of the cointegration relation between the price series (Lütkepohl & Weigend 1994).

*Recurrent networks*: In this chapter, we did not cover recurrent networks. One important advantage they have over feedforward networks is that they can recover state in the present of noise, similar to a hidden Markov model (see Chapter 17 in this volume by Nádas & Mercer). The interested reader is referred to the review by Mozer (1994) that provides a unified treatment and classification scheme for recurrent networks.

There are many other recent advances that can be used for connectionist time series analysis and prediction—just to name a few examples: combination of forecasts (Perrone 1994), "boosting" of the training set (Drucker, Cortes, Jackel, LeCun, & Vapnik 1994), variable subset selection (Bonnlander & Weigend 1994), wavelets for filtered embedding, a new embedding theorem for interspike time series (Sauer, personal communication, 1994), but let us try to regain perspective: where once, less than a decade ago, time series analysis was shaped by linear systems theory, it is now possible to recognize when an apparently complicated time series has been produced by a low-dimensional nonlinear system and to characterize its essential properties. A rich framework has been developed for designing algorithms that can learn the regularities also of time series whether they have a simple origin or not. The future will tell where in the space of depth and relevance these techniques fall.

## ACKNOWLEDGMENTS

This chapter would not have been possible without Neil Gershenfeld. I also deeply thank Dave Rumelhart for having seeded many of the connectionist ideas expressed in this chapter, Dave Nix and Ashok Srivastava for their computer simulations and their research, and Blake LeBaron for having introduced me to world of financial data. I am also very grateful for the discussions at many places with many excellent researchers that helped me understand some of the problems in the fields touched

upon in this chapter a little better. Some of this material is based upon work supported by the National Science Foundation under Grant RIA ECS-9309786.

## REFERENCES

Abu-Mostafa, Y. (1994). Hints. Submitted for publication.

Akaike, H. (1974), A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, **19**, 716–723.

Beck, C. (1990), Upper and lower bounds on the Renyi dimensions and the uniformity of multifractals. *Physica D*, **41**, 67–78.

Bishop, C. M. (in press). Training with noise is equivalent to Tikhonov regularization. *Neural Computation*.

Bonnlander, B. V., & Weigend, A. S. (1994). Selecting input variables using kernel density estimation. In *Proceedings of the International Symposium on Artificial Neural Networks* (ISANN'94), Tainan, Taiwan, R.O.C.

Box, G. E. P., & Jenkins, F. M. (1976). *Time series analysis: Forecasting and control* (2nd ed.). Oakland, CA: Holden–Day.

Bradley, E. (1992). *Taming chaotic circuits*. Doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA.

Brown, R., Bryant, P., & Abarbanel, H. D. I. (1991). Computing the Lyapunov spectrum of a dynamical system from an observed time series. *Physical Review A*, **43**, 2787–2806.

Cacciatore, T. W., & Nowlan, S. J. (1994). Mixtures of controllers for jump linear and nonlinear plants. In J. D. Cowen, G. Tesauro, & J. Alspector (Eds.). *Advances in neural information processing systems 6* (NIPS*93) (pp. 719–726). San Francisco, CA: Morgan Kaufmann.

Casdagli, M. C. (1991). Chaos and deterministic versus stochastic nonlinear modeling. *Journal of the Royal Stat. Soc. B*, **54**, 303–328.

Casdagli, M., Eubank, S., Farmer, J. D., & Gibson, J. (1991). State space reconstruction in the presence of noise. *Physica D*, **51D**, 52–98.

Casdagli, M. C., & Weigend, A. S. (1994). Exploring the continuum between deterministic and stochastic modeling. In edited by A. S. Weigend and N. A. Gershenfeld, (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 347–366). Reading, MA: Addison–Wesley.

Chaitin, G. J. (1966). On the length of programs for computing finite binary sequences. *J. Assoc. Comp. Mach.*, **13**, 547–569.

Chaitin, G. J. (1990). *Information, randomness and incompleteness* (2nd ed.). Series in Computer Science, Vol. 8, Singapore: World-Scientific.

Chatfield, C. (1989). *The analysis of time series* (4th ed.). London: Chapman and Hall.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information Theory*, New York: Wiley.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y., & Vapnik, V. (1994). Boosting and other machine learning algorithms. In W. W. Cohen, & H. Hirsh (Eds.). *Machine Learning: Proceedings of the Eleventh International Conference* (ML'94) Rutgers, NJ (pp. 53–61) San Francisco: Morgan Kauffmann.

Engle, R. F., & Granger, C. W. J. (1987). Cointegration and error-correcting representation, estimation and testing. *Econometrica*, **55**, 251–276.

Farmer, J. D., & Sidorowich, J. J. (1987). Predicting chaotic time series. Physical Review Letters, **58(8)**, 845–848.

Farmer, J. D., & Sidorowich, J. J. (1988). Exploiting chaos to predict the future and reduce noise. In Y. C. Lee (Ed.). *Evolution, learning, and cognition*, Singapore: World Scientific.

Fraser, A. M. (1989). Information and entropy in strange attractors. *IEEE Transactions Info. Theory*, **IT-35**, 245–262.

Fraser, A. M. & Swinney, H. L. (1986). Independent coordinates for strange attractors from mutual information. *Physical Review A*, **33**, 1134–1140.

Friedman, J. H. (1991). Multivariate adaptive regression splines. [with discussion.] *Ann. Stat.*, **19**, 1–142.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation* **5**, 1–58.

Gencay, R., & Dechert, W. D. (1992). An algorithm for the $n$ Lyapunov exponents of an $n$-dimensional unknown dynamical system. *Physics D*, **59**, 142–157.

Gershenfeld, N. A. (1989). An experimentalist's introduction to the observation of dynamical systems. In B.-L. Hao (Ed.). *Directions in Chaos*, Vol. 2 (pp. 310–384). Singapore: World Scientific.

Gershenfeld, N. A. (1993), Information in dynamics. In D. Matzke (Ed.). *Proceedings of the Workshop on Physics of Computation*, (pp. 276–280). Los Alamitos, CA: IEEE Press.

Gershenfeld, N. A., & Weigend, A. S. (1994). The future of time series: Learning and understanding, In A. S. Weigend & N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 1–70). Reading, MA: Addison–Wesley.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, **38**, 173–198 (English translation of this paper in *On formally undecidable propositions* by K. Gödel, 1962, New York: Basic Books).

Granger, C. W. J., & Andersen, A. P. (1978). An introduction to bilinear time series models. Göttingen: Vandenhoek and Ruprecht.

Granger, C. W. J., & Teräsvirta, T. (1993). *Modelling nonlinear economic relationships.* Oxford, UK: Oxford University Press.

Grassberger, P. (1988). Finite sample corrections to entropy and dimension estimates. *Phys. Lett A*, **128**, 369–373.

Grebogi, C., Hammel, S. M., Yorke, J. A., & Sauer, T. (1990). Shadowing of physical trajectories in chaotic dynamics: Containment and refinement. *Physical Review Letters*, **65**, 1527.

Guillemin, V., & Pollack, A. (1974). *Differential topology*, Englewood Cliffs, NJ:Prentice–Hall.

Gutowitz, H. (ed.) (1991). *Cellular automata, theory and experiment.* Cambridge, MA: MIT Press.

Hentschel, H. G. E., & Procaccia, I. (1983). The infinite number of generalized dimensions of fractals and strange attractors. *Physica D*, **8**, 435–444.

Hu, M. J. C. (1964). *Application of the Adaline system to weather forecasting*. E. E. degree thesis. Tech. Rep. 6775-1, Stanford University, Stanford Electronic Laboratories, Stanford, CA.

Hübler, A. (1989). Adaptive control of chaotic systems. *Helv. Phys. Acta*, **62**, 343–346.

Hübner, U., Weiss, C. O., Abraham, N. B., & Tang. D. (1994). Lorenz-like chaos in $NH_3$-FIR lasers. In A. S. Weigend and N. A. Gershenfeld, (Eds.). *Time series prediction: forecasting the future and understanding the past* (pp. 73–104). Reading, MA: Addison–Wesley.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, **3**, 79–87.

Kazlas, P. T., & Weigend, A. S. (1995). Settling temporal differences: Time series prediction using TD($\lambda$). In *Advances in neural information processing systems 7* (NIPS*94). San Francisco, CA: Morgan Kaufmann.

Kevrekidis, I. G., Rico-Martinez, R., Ecke, R. E., Farber, R. M., & Lapedes, A. S. (1993, May). *Global bifurcations in Rayleigh–Benard convection* (Los Alamos preprint, LA-UR-93-2922). Submitted for publication.

Kolmogorov, A. (1941). Interpolation und extrapolation von stationären zufälligen Folgen. *Bull. Acad. Sci. (Nauk)* U.S.S.R., Ser. Math, **5**, 3–14.

Kolmogorov, A. N. (1965). Three approaches to the quantitative definition of information. *Prob. Infor. Trans.*, **1**, 4–7.

Landauer, R. (1991). Information is physical. *Physics Today*, **44**, 23.

Lang, K. J., Waibel, A. H., & Hinton, G. E. (1990). A time-delay neural network architecture for isolated word recognition. *Neural Networks* **3**, 23–43.

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

le Cun, Y. (1989). Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels (Eds.). *Connectionism in perspective*. Amsterdam: North Holland.

le Cun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In D. S. Touretzky (Ed.). *Advances in Neural Information Processing Systems 2* (NIPS*89), (pp. 598–605). San Mateo, CA: Morgan Kaufmann.

Lewis, P. A. W., Ray, B. K., & Stevens, J. G. (1994). Modeling time series using multivariate adaptive regression splines (MARS). In A. S. Weigend and N. A. Gershenfeld, (Eds.). *Time series prediction: Forecasting the future and understanding the past*, (pp. 296–318). Reading, MA: Addison–Wesley.

Lorenz, E. N. (1963). Deterministic non-periodic flow. *J. Atmos. Sci.*, **20**, 130–141.

Lorenz, E. N. (1989). Computational chaos—A prelude to computational instability. *Physica D*, **35**, 299–317.

Lütkepohl, H., & Weigend, A. S. (1994). Manuscript in preparation.

Mangeas, M., & Weigend, A. S. (1994). *Experts for prediction*. Manuscript in preparation, University of Colorado at Boulder. Computer Science Department.

May, R. M. (1976). Simple mathematical models with very complicated dynamics. *Nature*, **261**, 459.

Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.). (1990). *Neural networks for control*. Cambridge, MA: MIT Press.

Moody, J. (1994). Prediction risk and architecture selection for neural networks. In V. Cherkassky, J. H. Friedman & H. Wechsler (Eds.). *From statistics to neural networks: Theory and pattern recognition applications*. NATO ASI Series F, Berlin: Springer–Verlag.

Moore, C. (1991). Generalized shifts: Unpredictability and Undecidability in dynamical systems. *Nonlinearity*, **4**, 199–230.

Mozer, M. C. (1994). Neural net architecture for temporal sequence processing. In A. S. Weingard & N. A. Gersenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 243–264). Reading, MA: Addison–Wesley.

Nádas, A., & Mercer, R. L. (1996). Hidden Markov models and some connections with artificial neural nets. This volume.

Narendra, K. S., & Li, S.-M. (1996). Neural networks in control systems. This volume.

Nix, D. A., & Weigend, A. S. (1995). Local error bars for nonlinear regression and time series prediction. In *Advances in neural information processing systems 7* (NIPS*94). San Francisco, CA: Morgan Kaufmann.

Nowlan, S. J., & Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation*, **4**, 473–493.

Nychka, D., Ellner, S., McCaffrey, D., & Gallant, A. R. (1992). Finding chaos in noisy systems. *J. Roy. Stat. Soc. B*, **54**(2), 399–426.

Oppenheim, A. V., & Schafer, R. W. (1989). *Discrete-time signal processing*. Englewood Cliffs, NJ: Prentice–Hall.

Ott, E., Grebogi, C., & Yorke, J. A. (1990). Controlling chaos. *Physical Review Letters*, **64**, 1196.

Packard, N. H., Crutchfeld, J. P., Farmer, J. D., & Shaw, R. S. (1980). Geometry from a time series. *Physical Review Letters*, **45**(9), 712–716.

Paluš, M. (1994). Identifying and quantifying chaos using information-theoretic functionals. In A. S. Weigend and N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past*. (pp. 387–413). Reading, MA: Addison–Wesley.

Perrone, M. P. (1994). General averaging results for complex optimization. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend (Eds.). *Proceedings of the 1993 Connectionist Models Summer School* (pp. 364–371). Hillsidale, NJ: Lawrence Erlbaum Associates.

Petersen, K. (1989). *Ergodic theory* (2nd ed.). Cambridge Studies in Advanced Mathematics. Vol. 2, Cambridge, MA: Cambridge University Press.

Pi, H., & Peterson, C. (1994). Finding the embedding dimension and variable dependences in time series. *Neural Computation*, **6**, 509–520.

Pineda, F. J., & Sommerer, J. C. (1994). Estimating generalized dimensions and choosing time delays: A fast algorithm. In A. S. Weigend and N. A. Gershenfeld, (Eds.) *Time series prediction: Forecasting the future and understanding the past* (pp. 367–385). Reading, MA: Addison–Wesley.

Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T. (1992). *Numerical Recipes in C: The art of scientific computing* (2nd ed.). Cambridge, UK: Cambridge University Press.

Prichard, D., & Theiler, J. (in press). Generalized redundancies for time series analysis. *Physica D.*

Priestley, M. (1981). *Spectral analysis and time series*. London: Academic.

Principe, J. C., de Vries, B., & Oliveira, P. (1993). The gamma filter—A new class of adaptive IIR filters with restricted feedback. *IEEE Trans. Sig. Proc.*, **41**, 649–656.

Rico-Martinez, R., Kevrekidis, I. G., & Adomaitis, R. A. (1993). Noninvertibility in neural networks. In *IEEE International Conference on Neural Networks* (ICNN) San Francisco, CA (pp. 382–386). Piscataway, NJ: IEEE Press.

Rissanen, J. (1986). Stochastic complexity and modeling. *Ann. Stat.*, **14**, 1080–1100.

Rissanen, J. (1987). Stochastic complexity. *J. Roy. Stat. Soc. B*, **49**, 223–239, with discussion 252–265.

Rissanen, J. (1996). Information theory and neural nets. This volume.

Rissanen, J., & Langdon, G. G. (1981). Universal modeling and coding. *IEEE Trans. Info. Theory*, **IT-27**, 12–23.

Ruelle, D., & Eckmann, J. P. (1985). Ergodic theory of chaos and strange attractors. *Reviews of Modern Physics*, **57**, 617–656.

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (1996). Backpropagation: The basic theory. This volume; in Y. Chauvin, and D. E. Rumelhart (Eds.). *Backpropagation: Theory, Architectures, and Applications*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.). *Parallel distributed processing: Explorations in the microstructure of cognition. Volume I: Foundations* (pp. 318–362). Cambridge, MA: MIT Press/Bradford Books.

Sauer, T. (1994). Time Series Prediction Using Delay Coordinate Embedding. In A. S. Weigend and N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 175–193). Reading, MA: Addison–Wesley.

Sauer, T., Yorke, J. A., & Casdagli, M. (1991). Embedology,. *J. Stat. Phys.*, **65**(3/4), 579–616.

Saund, E. (1989). Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-11**, 304–314.

Schuster, A. (1898). On the investigation of hidden periodicities with applications to a supposed 26-day period of meteorological phenomena. *Terr. Mag.*, **3**, 13–41.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Tech. J*, **27**, 379–423, 623–656. (Reprinted in *Key Papers in the Development of Information Theory*, pp. 5–18 by D. Slepian, Ed., New York: IEEE Press.)

Shaw, R. S. (1981). Strange attractors, chaotic behavior and information flow. *Z. Naturforsch*, **36A**, 80–112.

Solomonoff, R. J. (1964). A formal theory of induction inference, Parts I and II. *Information Control*, **7**, 1–22, 221–254.

Srivastava, A. N., & Weigend, A. S. (1994). Computing the probability density in connectionist regression. In *Proceedings of the International Conference on Artificial Neural Networks* (ICANN), Sorrento, Italy, (pp. 685–688). Berlin: Springer–Verlag.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**, 9–44.

Svarer, C., Hansen, L. K., & Larsen, J. (1993), On design and evaluation of tapped-delay neural network architectures. In *IEEE International Conference on Neural Networks*, (ICNN), San Francisco, CA (pp. 46–51). Piscataway, NJ: IEEE Press.

Swinney, H. L. (1994). Spatio-temporal patterns: Observations and analysis. In A. S. Weigend & N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 557–567). Reading, MA: Addison–Wesley.

Takens, F. (1981). Detecting Strange attractors in turbulence. In D. A. Rand & L.-S. Young (Eds.). *Dynamical systems and turbulence*, Lecture Notes in Mathematics, Vol. 898 (pp. 336–381). Warwick, 1980. Berlin: Springer–Verlag.

Temam, R. (1988). *Infinite-dimensional dynamical systems in mechanics and physics.* Applied Mathematical Sciences, Vol. 68, Berlin: Springer–Verlag.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, **8**, 257–277.

Theiler, J. (1990). Estimating fractal dimension. *J. Opt. Soc. Am. A*, **7(6)**, 1055–1073.

Theiler, J., Linsay, P. S., & Rubin, D. M. 1994. Detecting nonlinearity in data with long coherence times. In A. S. Weigend & N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 439–455). Reading, MA: Addison–Wesley.

Tong, H., & Lim, K. S. (1980). Threshold autoregression, limit cycles and cyclical data. *J. Roy. Stat. Soc. B*, **42**, 245–292.

Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, **42**, 230–265.

Ulam, S. M, & von Neumann, J. (1947). *Bulletin Amer. Math. Soc.*, **53**, 1120.

Wallace, C. S., & Boulton, D. M. (1968). An information measure for classification. *Comp. J.*, **11**, 185–195.

Wan, E. A. (1994). Time series prediction using a connectionist network with internal delay lines. In A. S. Weigend and N. A. Gershenfeld (Eds.). *Time series prediction: Forecasting the future and understanding the past* (pp. 195–217). Reading, MA:Addison–Wesley.

Weigend, A. S. (1991). *Connectionist architectures for time series prediction of dynamical systems.* Doctoral dissertation, Stanford University, Stanford, CA.

Weigend, A. S. (1994). On overfitting and the effective number of hidden units. In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend (Eds.). *Proceedings of the 1993 Connectionist Models Summer School* (pp. 335–342). Hillsdale, NJ: Lawrence Erlbaum Associates.

Weigend, A. S., & Gershenfeld, N. A. (Eds.). (1994). *Time series prediction: Forecasting the future and understanding the past* (Santa Fe Institute Studies in the Sciences of Complexity, Proc.). Vol. XV. Reading, MA: Addison-Wesley.

Weigend, A. S., Huberman, B. A., & Rumelhart, D. E. (1990). Predicting the future: A connectionist approach. *International Journal of Neural Systems*, **1**, 193–209.

Weigend, A., Huberman, B. A., & Rumelhart, D. E. (1992). Predicting sunspots and exchange rates with connectionist networks. In M. Casdagli & S. Eubank (Eds.). *Nonlinear modeling and forecasting* (pp. 395–432). Redwood City, CA: Addison–Wesley.

Weigend, A. S., & LeBaron, B. (1994). Evaluating neural network predictors by bootstrapping. In *Proceedings of International Conference on Neural Information Processing* (ICONIP'94), Seoul, Korea; also (Tech. Rep. CU-CS-725-94, University of Colorado at Boulder, Computer Science Department.

Weigend, A. S., & Rumelhart, D. E. (1991). Generalization through minimal networks with application to forecasting. In E. M. Keramidas (Eds.). *23rd Symposium on the Interface: Computing Science and Statistics* (INTERFACE'91), Seattle, WA (pp. 362–370). Interface Foundation of North America.

Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. 1991. Generalization by Weight-Elimination with Application to Forecasting. In R. P. Lippmann, J. Moody, & D. S. Touretzky (Eds.). *Advances in Neural Information Processing Systems 3* (NIPS*90) (pp. 875–882). Redwood City, CA: Morgan Kaufmann.

Weigend, A. S., & Zimmermann, H. G. (1994). Manuscript in preparation, Computer Science Department, University of Colorado at Boulder.

White, D. A., & Sofge, D. A. (Eds.). (1992). *Handbook of Intelligent control.* Van Nostrand Reinhold.

Wiener, N. (1949). *The extrapolation, interpolation and smoothing of stationary time series with engineering applications.* New York: Wiley.

Yule, G. (1927). On a method of investigating periodicity in distributed series with special reference to Wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London*, **A 226**, 267–298.

Zimmermann, H. G. (1994), Neuronale Netze in der Ökonometrie. In H. Rehkugler, & H. G. Zimmermann (Eds.). *Neuronale Netze als Entscheidungskalkul* (pp. 1–87). Münichen: Vahlen Verlag (In German.)