Estimation and prediction for nonlinear time series
Borovkova, Svetlana Alfredovna

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
1998

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
7 Prediction

Prediction of future observations is an important problem in the analysis of time series. Given a time series \( \{Y_n\}_{n \in \mathbb{N}} \) the question is to find a predictor for \( Y_{n+s} \), \( s \geq 1 \), as a function of a certain number of previous observations, i.e. we are looking for

\[
\hat{Y}_{n+s} = F(y_n, y_{n-1}, \ldots, y_{n-k+1}) \tag{7.1}
\]

\[
:= E(Y_{n+s} | Y_n = y_n, \ldots, Y_{n-k+1} = y_{n-k+1}).
\]

In traditional time series analysis, \( F \) is restricted to a parametric, usually linear form and a linear autoregression with Gaussian innovations (AR of ARMA) is applied to estimate \( F \). However, as we argued in the introduction, for many time series the function \( F \) can be nonlinear for many time series. In that case linear predictions are not necessarily appropriate.

For example, if \( f : \mathbb{R} \rightarrow \mathbb{R} \) is a nonlinear chaotic map, then the time series obtained by \( y_{n+1} = f(y_n) \), for some \( y_0 \in \mathbb{R} \), satisfies the first order autoregression model with nonlinear autoregression function \( f \). Usually there is noise present in the time series. What we actually observe is

\[
\tilde{y}_{n+1} = y_{n+1} + \epsilon_{n+1} = f(y_n) + \epsilon_{n+1},
\]

where \( \epsilon_n \) are zero-mean errors with finite variance.

Theory of chaotic dynamical systems shows that even simple nonlinear dynamical systems can give rise to time series which exhibit highly erratic and seemingly random behaviour. Well-known examples, already mentioned above, are: the logistic map \( f(x) = 4x(1 - x) \), the Henon map, the Lorenz system of 3 coupled nonlinear differential equations.

Time series coming from a chaotic dynamical system in general satisfy a nonlinear autoregression model. This can be briefly explained by the following. Recall that the reconstruction vectors \( X_1, X_2, \ldots \in \mathbb{R}^k \) are obtained
from the observed time series \( \{ y_n \}_{n \in \mathbb{N}} \) by

\[
X_i = (y_i, y_{i+1}, \ldots, y_{i+k-1}).
\]

Takens reconstruction theorem implies that, in generic situations, if \( k \) is sufficiently high, the reconstruction vectors \( X_i \) are accumulating in \( \mathbb{R}^k \) in the neighbourhood of an object, diffeomorphic to the original attractor, and, moreover, that there is dynamical map induced on the space of the reconstruction vectors. This map is also diffeomorphic to the original chaotic map \( T \), i.e. in the absence of noise there exists a map \( G : \mathbb{R}^k \to \mathbb{R}^k \), diffeomorphic to \( T \), such that for all \( i \)

\[
X_{i+1} = G(X_i). \tag{7.2}
\]

In terms of the original time series, the relation (7.2) means that there also exists a function \( F : \mathbb{R}^k \to \mathbb{R} \) such that

\[
y_{i+k} = F(y_i, y_{i+1}, \ldots, y_{i+k-1}). \tag{7.3}
\]

The original transformation \( T \) was assumed to be nonlinear, so \( F \) is also a nonlinear function. The relationship (7.3) implies that the time series \( \{ y_i \}_{i \in \mathbb{N}} \) satisfies the \( k \)th order nonlinear autoregression model, where the autoregression function \( F \) is some unknown nonlinear function not restricted to any parametric form. If there is noise present in the system, we shall for simplicity again assume that it is additive measurement noise, i.e. that we observe

\[
y_{i+k} = y_{i+k} + \epsilon_{i+k} = F(y_i, y_{i+1}, \ldots, y_{i+k-1}) + \epsilon_{i+k}, \tag{7.4}
\]

where \( \epsilon_i \) are mean zero and finite variance errors. For clarity’s sake we shall write \( \{ \tilde{y}_i \}_{i \in \mathbb{N}} \) for the observed time series, even when an additive noise is present.

Relation (7.4) expresses the functional dependence of the next observation in the time series \( \{ \tilde{y}_i \}_{i \in \mathbb{N}} \) on the previous \( k \) observations. In many practical situations we are interested in the relation between the observation \( s > 1 \) steps ahead and the previous \( k \) values, i.e. in the nonlinear function \( F^s \):

\[
y_{i+s} = F^s(y_i, y_{i-1}, \ldots, y_{i-k+1}). \tag{7.5}
\]
If (7.3) holds, $F^s$ can be expressed as

$$F^s(\cdot) = F(F(...(F(\cdot)...))...).$$

$s$ times

In the case of chaotic time series, as well as other nonlinear time series (not necessarily chaotic), one expects nonlinear or locally linear methods of prediction to have an advantage over traditional linear methods.

A method of prediction frequently mentioned in the literature on chaotic time series is the so-called $(k, \epsilon)$-method (or locally linear predictors). It is based on the assumption that it is best to capture the underlying dynamics locally, to reduce the effects of nonlinearity. On the local scale the nonlinear function $F$, if it is smooth enough, can be successfully interpolated by a linear function. The method can be briefly described as follows: suppose we have observed $k$ past values of the time series $\tilde{y} := y_1, ..., y_k$, and want to predict $y_{k+s}$. We collect all vectors of length $k$ from the time series which are within distance $\epsilon$ from $\tilde{y}$, as well as the corresponding observations $s$ steps after, and then apply a linear regression to the collected data. The pair $(k, \epsilon)$ is selected by minimising some measure of prediction error. This method turns out to be quite successful if the time series comes from a low-dimensional chaotic dynamical system and enough data are available.

In the next two sections we shall concentrate on another method: the kernel regression smoothing. This method also has a local character, but is more flexible.

### 7.1 Kernel autoregression estimation for time series

Kernel smoothing is a method from nonparametric statistics for estimating an unknown regression function. First we describe it briefly in a more general setting.

Suppose that random pairs $(X_i, Y_i)$, $i = 1, ..., n$, are coming from the same distribution as a random vector $(X, Y)$, where $X, X_i \in \mathbb{R}^k$, $Y, Y_i \in \mathbb{R}$, and suppose that there exists a function $r : \mathbb{R}^k \rightarrow \mathbb{R}$ such that

$$Y_i = r(X_i) + \epsilon_i,$$  \hspace{1cm} (7.6)
where \( \epsilon_i \) are i.i.d. zero-mean observation errors with fixed finite variance. Here \( r \), called the \textit{regression function}, is an unspecified function and it is not restricted to any parametric form. It can also be defined as the conditional expectation of \( Y \) given that \( X = x \):

\[
r(x) = \mathbb{E}(Y|X = x)
\]

(it is well-defined if \( \mathbb{E}|Y| < \infty \)). The components of the random vector \( X \) are also called the \textit{explanatory variables}, and the random variable \( Y \) is regarded as the \textit{response}. One is interested in approximating the general relationship between \( X \) and \( Y \), the function \( r \), on the basis of the sample \( \{(X_i, Y_i)\}_{i=1,\ldots,n} \).

A \textit{kernel estimator} of the function \( r \) is

\[
\hat{r}_{n,h}(x) = \frac{\sum_{i=1}^{n} K_h(x - X_i) Y_i}{\sum_{i=1}^{n} K_h(x - X_i)},
\]

where \( K_h \) for \( k > 1 \) is defined via the so-called \textit{kernel function} \( K : \mathbb{R} \rightarrow \mathbb{R} \) by

\[
K(z) = \prod_{l=1}^{k} K(z_l/h),
\]

where \( z = (x_1, \ldots, x_k) \). \( K_h \) is called the \textit{product kernel}. For \( k = 1 \) one takes \( K_h(y) = K(y/h) \). Here \( h = h_n \) is a sequence of scaling parameters, called the \textit{bandwidth sequence}. The estimator (7.7) is usually referred to as the \textit{Nadaraya-Watson estimator}.

The kernel function \( K(y) \) is taken to be a continuous bounded and symmetric real-valued function which integrates to 1. Usually these functions are also taken to be unimodal with maximum at 0. The most commonly used kernel functions are:

- Triangle \((1 - |y|) \cdot 1(|y| \leq 1)\);
- Epaneshnikov \(\frac{3}{4}(1 - y^2) \cdot 1(|y| \leq 1)\);
- Gaussian \(\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}y^2)\).

In the context of time series, we can denote \( X_i = (y_{i-1}, \ldots, y_{i-k+1}) \) and \( Y_i = y_{i+s} \). Then the kernel estimator of the autoregression function \( F \) is

\[
\hat{F}_h(x) = \frac{\sum_{i=k}^{n} K_h(x - X_i) Y_i}{\sum_{i=k}^{n} K_h(x - X_i)}.
\]
The application of the kernel method to time series was studied by Collomb (1984), Delecroux (1987) (see [35] and the references therein), and it was shown (Delecroux (1987)) that the estimator (7.8) is consistent if the time series is a stationary ergodic process and the bandwidth sequence converges to 0 at some specified rate. (See also [13].)

In applications a very important question is how to select the bandwidth. Selecting a bandwidth that is too small leads to a higher variance of the kernel estimator (this is called undersmoothing), and choosing a bandwidth that is too large increases its bias (oversmoothing). In practice we have to balance these two factors. Consider the average squared error:

\[ \text{ASE}[\hat{F}_h] = \frac{1}{n} \sum_{i=k}^{n} [\hat{F}_h(X_i) - F(X_i)]^2. \]

which is the sum of the variance and the squared bias components. A value of the bandwidth which minimises the ASE is desirable.

The consistency results of Delecroux, Bosq, mentioned above, give the theoretical optimal rate of convergence of the bandwidth in terms of \( n \), which balances the variance and the bias of the kernel estimate. In most cases this rate is \( h = h_n \sim 0(n^{-1/5}) \). However, these results are of purely theoretical value and do not tell us how to choose the bandwidth in practice. A data-driven approach for bandwidth selection is therefore necessary. Such an approach is called cross-validation and amounts to the following. We estimate the ASE by the cross-validation function

\[ CV(h) = \frac{1}{n} \sum_{i=k}^{n} [Y_i - \hat{F}_{h,i}(X_i)]^2, \]

where \( \hat{F}_{h,i}(X_i) \) is the leave-i-out kernel estimate of \( F(X_i) \)

\[ \hat{F}_{h,i}(X_i) = \frac{\sum_{i \neq j} K_h(X_i - X_j) Y_j}{\sum_{i \neq j} K_h(X_i - X_j)}. \]

The cross-validation function \( CV(h) \) is an asymptotically unbiased estimator of the ASE and has the advantage that it can be computed directly from the data. Then we choose the value of the bandwidth \( \hat{h}_{opt} \) which minimises \( CV(h) \). The question then is whether \( \hat{h}_{opt} \) also (asymptotically) minimises the ASE? Haerdle and Vieu [38] have shown that, if the sequence \( (X_i, Y_i) \)
is strongly mixing and some additional conditions on $K$, $F$ and on the distribution of $(X_i, Y_i)$ are satisfied, then the cross-validation procedure is asymptotically optimal, i.e. that the bandwidth chosen by means of the cross-validation asymptotically minimises the ASE.

### 7.2 Variation of kernel smoothing method

One of the main disadvantages of the methods mentioned above (traditional kernel smoothing, $(k, \epsilon)$-method) is the fixed choice of the autoregression order $k$, which results into taking entirely into account the previous $k$ observations and completely disregarding the rest. Moreover, in applications $k$ is usually unknown and has to be estimated in some way (traditionally via linear models).

In this section we suggest the variation of kernel autoregression smoothing which overcomes this disadvantage and is in general more flexible.

One way to apply the kernel smoothing method for prediction in the case $k > 1$ involves product kernels. Another way is to define in some suitable way a distance between two $k$-dimensional vectors $\underline{x}, \underline{y}$: $d(\underline{x}, \underline{y})$, and define the kernel regression estimator via a one-dimensional kernel function as

$$\hat{F}_h(x) = \frac{\sum_{i=1}^n K[d(\underline{x}, X_i)/h] Y_i}{\sum_{i=1}^n K[d(\underline{x}, X_i)/h]}$$

(7.9)

The distance $d(\cdot, \cdot)$ can be taken as the Euclidean or the maximum distance. However, the choice of these distances would be purely formal and would not exploit the fact that the vectors $X_i$’s are parts of a time series, i.e. sequences of observations ordered in time. Here we suggest a variation of the estimator (7.9) and choose a distance between the vectors which takes the specifics of the time series setting into account.

We define the distance between two vectors $\underline{x} = (x_1, x_2, ..., x_k)$ and $\underline{y} = (y_1, y_2, ..., y_k)$ as

$$d(\underline{x}, \underline{y}) = \sum_{i=1}^k (x_i - y_i)^2 \gamma_i,$$  

(7.10)

where $\{\gamma_i\}_{i=1}^k$, $\gamma_i \in [0,1]$, is a collection of weights which we put on each of the differences between the coordinates.

In general, we will chose decreasing weights $\gamma_1 \geq \gamma_2 \geq ... \geq \gamma_k$, expressing the idea that the influence of past observations on the prediction should
be discounted as the time lag grows. More specifically, we suggest to take

$$\gamma_i = \gamma^i,$$  \hspace{1cm} (7.11)

for some $\gamma \in (0, 1]$. One motivation for this is the exponential divergence of trajectories (whose starting points are close) in chaotic dynamical systems. However, this technique is appropriate for many time series, including those with stochastic properties. Thus, the more recent observations give more precise information about the present state of the system, and the information decreases exponentially as time passes.

Observations further in the past are less relevant for future predictions since they are most affected by noise. In chaotic dynamical systems this is expressed via the notion of the sensitive dependence on initial conditions. The exponential growth of errors, which are always present in a real-life systems, makes the chaotic evolution self-independent of its own past. Quantitatively, this is described by the Lyapunov exponents of a chaotic map. The mean exponential rate with which nearby orbits diverge with time is measured by the highest Lyapunov exponent, and the existence of at least one positive Lyapunov exponent is evidence for the sensitive dependence on initial conditions. In the context of chaotic time series we are more interested in backward divergence of orbits, which is responsible for decreasing influence of the past observations on the future ones. In that case the mean rate of divergence of backwards orbits is measured by the highest Lyapunov exponent of the inverse map $T^{-1}$ (provided $T$ is invertible). It is given by the inverse of the lowest Lyapunov exponent of the original map $T$.

For our applications this situation can be illustrated by the following example. If $\underline{x} = (x_1, ..., x_k)$ and $\underline{y} = (y_1, ..., y_k)$ are segments of two orbits of an invertible chaotic dynamical system, or two parts of the same orbit, and if $||x_k - y_k|| = \delta$, then, due to the exponential divergence of backward trajectories, $||x_{k-1} - y_{k-1}|| = \delta e^\lambda$, $||x_{k-2} - y_{k-2}|| = \delta e^{2\lambda}$, etc., where $\lambda$ is the highest Lyapunov exponent of the inverse of the underlying chaotic map. The distance between $\underline{x}$ and $\underline{y}$ is:

$$d(\underline{x}, \underline{y}) = \sum_{j=1}^{k} ||x_{k-j} - y_{k-j}||^2 \gamma^j = \sum_{j=1}^{k} \delta e^{2j\lambda} \gamma^j$$  \hspace{1cm} (7.12)
and it is completely determined by \( \delta \), while the factors \( e^{i\delta} \) indicate the natural divergence of trajectories with the rate \( \lambda \). This reasoning suggests taking \( \gamma = e^{-2\lambda} \) to “discount” for this exponential divergence. Due to the state space reconstruction, the above reasoning also applies when \( \mathbf{x} = (x_1, \ldots, x_k) \) and \( \mathbf{y} = (y_1, \ldots, y_k) \) are not parts of an orbit, but segments of a chaotic time series.

In practice we will choose \( \gamma \) in a different way, based on optimalisation of the value of \( \gamma \) in an experimental way. This method is explained in more detail below.

We believe our method is more flexible than that of locally linear predictors, as well as the other methods mentioned above. Rather than having sharp cut-off points \((k, \varepsilon)\) in time we have a smoother way to assign the significance of past observations. The choice of the parameter \( \gamma \) in a way replaces the choice of the order of autoregression \( k \). By taking the decreasing weights (7.11), the choice of \( k \) becomes less important, because the dependence on the previous observations is not cut off at the number \( k \), but decreases smoothly with the decrease of \( \gamma \).

The influence of the parameter \( \gamma \) on our kernel estimate is comparable to the bandwidth influence. Thus, the choice of \( \gamma \) as well as of \( h \) determines the quality of our predictions. At the same time the choice of \( \gamma \) and of \( h \) is not independent: the bigger the bandwidth \( h \), the higher the value of \( \gamma \) we should choose. Again we use a cross-validation algorithm. But since the parameters \( h \) and \( \gamma \) are bound together, the selection of their optimal values should be carried out simultaneously, i.e. we choose the optimal pair \((h, \gamma)_{opt}\) as

\[
(h, \gamma)_{opt} = \text{argmin}\{CV(h, \gamma)\},
\]

where \( CV(h, \gamma) \) is a cross-validation function of the estimator (7.9).

The consistency of our estimator (7.9) follows from standard results for the traditional kernel estimator (see [35], [13]). The asymptotic optimality of the double cross-validation procedure can possibly be seen using similar arguments to those found in [38].

Taking decreasing weights \( \gamma \) is not the only possible choice, and sometimes not the most efficient one. For some time series not the most recent observation(s), but those further in the past have more influence on future observations. Consider the following example.
Suppose that the time series \( \{x_n\}_{n \in \mathbb{N}} \) is obtained by
\[
x_{n+1} = 0.01x_n + f(x_{n-1}) + \epsilon_n,
\]
where \( f : \mathbb{R} \rightarrow \mathbb{R} \) is a nonlinear chaotic map (e.g., a logistic map) and \( \epsilon_n \) are mean zero and finite variance errors. Here the value of the next observation depends much more on the value of the observation \( t \) time units in the past than on the previous one. Consequently, \( \gamma_t \) (and, possibly, also \( \gamma_{2t}, \gamma_{3t}, \text{etc.} \)) should be taken significantly bigger than \( \gamma_1 \) and other weights.

How can we recognize such a time series? It is usually not possible by just observing its plot, because the chaotic evolution \( f \) produces the time series which appears random. We generated the following time series:
\[
x_n = 0.01x_{n-1} + f(x_{n-4}) + \epsilon_n, \quad n = 1, \ldots, 100,
\]
with \( f(x) = 4xe^{-x^2} \), and the i.i.d. errors \( \epsilon_i \sim \mathcal{N}(0,0.05) \) (Fig. 7.1).

![Figure 7.1: Time series \( x_n \)](image-url)
Brockwell and Davis [15], pp. 98-102. The partial autocorrelation has the following meaning: its value at lag $k$ indicates the amount of the additional information obtained from considering the linear autoregression model of order $k$ instead of order $k - 1$. Although the partial autocorrelations are defined in terms of a linear model, they certainly do show at which lags the dependence is most significant. The plot of partial autocorrelations for our working example is given in Fig. 7.5.

The plot shows that the largest partial autocorrelations are at lags 4 and 12, with less significant ones at 1 and 3. This information can be used for determining the weights $\gamma_i$. For instance, $\gamma_i$'s can be taken proportional
to the partial autocorrelations. In terms of dynamical systems, the quantity analogous to the partial autocorrelation can be considered. This quantity is called the *mutual information*. Here we shall not go into detail as to how to define it, we shall only mention that this quantity with respect to an orbit of a dynamical system carries essentially the same meaning as the partial autocorrelation with respect to a time series. It can be also consistently estimated from a time series and, consecutively, used for determining the weights $\gamma_i$.

The example we just considered is more an exception than the rule. For most nonlinear time series geometrically decreasing weights $\gamma_i = \gamma^i$ is the most natural choice, since it reflects the idea of a decreasing influence of past observations on future ones. Moreover, this has the advantage that the choice of the autoregression parameter $k$ becomes less essential, and also that the value of the parameter $\gamma$ can in practice be selected together with the value of $h$ by the cross-validation procedure.

Taking a distance of the form (7.12) with weights $\gamma_i = \gamma^i$ for application of the kernel smoothing method for prediction is also possible when high values of autocorrelations are observed at lags $\tau, 2\tau, \ldots$ for some $\tau > 1$. Then we consider the model

$$y_i = F(y_{i-\tau}, y_{i-2\tau}, \ldots, y_{i-k\tau}) + \epsilon_i$$

and estimate $F$ by (7.9) with a distance as in (7.12) and weights $\gamma_i = \gamma^i$, $i = 1, \ldots, k$ for some $\gamma \in (0, 1)$. This is also useful when a time series is oversampled. In that case considering the autoregression on a number of all consecutive previous observations is not necessary and only leads to an oversized model. Then, by choosing a time delay $\tau$ and proceeding in the same way as above, we significantly reduce the model and, so, the computation time.

In general, the kernel autoregression estimate (7.9) with a distance as in (7.10) allows for broader flexibility of an assumed autoregression model and of assigning the influence of past observations on future ones.

### 7.3 Prediction by Neural Networks

Here we shall address briefly another nonlinear method of prediction which has become popular in the past few years: the application of neural net-
works.

7.3.1 Feedforward Neural Networks

Artificial neural networks (NN) originated as a mathematical model of the functioning of the human brain. Mathematically a neural network represents a directed graph, where the vertices, arranged in layers, are called the neurons and the directed edges the synaptic connections. Feedforward neural nets are those with connections leading only from neurons on the \( l \)th layer to the \( l + 1 \)st layer. Each edge feeds the output of a neuron on the previous layer to the input of a neuron on the next layer, and has a synaptic weight assigned to it. The neurons which receive an input from outside a network, i.e. those without input connections from other neurons, are called input neurons; those without output connections to other neurons are called output neurons. The layers in between are called hidden layers.

An output of a neuron \( j \) is multiplied by the synaptic weight \( w_{ij} \), which corresponds to the connection from the \( j \)th to the \( i \)th neuron, and the result is received as an input of the \( i \)th neuron. Then, all inputs of the \( i \)th neuron and some threshold \( \theta_i \) are summed up, and a function called the activation or transfer function \( \sigma \) is applied to the sum. This procedure is repeated till the output layer of the network is reached. If the output layer consists of more than 1 neuron, such a network can be considered as a superposition of several networks, so it makes sense to consider only neural networks with one output neuron.

The most commonly used neural networks are those with one hidden layer. A feedforward neural network with \( k \) input neurons, 1 hidden layer of \( n \) neurons and one output neuron can be viewed as a function \( f = f_n : \mathbb{R}^k \rightarrow \mathbb{R} \), and the total output of such a neural net is given by

\[
f_n(x) = \sum_{j=1}^{n} v_j \sigma \left( \sum_{i=1}^{k} (w_{ij} x_i - \theta_j) \right) = \sum_{j=1}^{n} v_j \sigma (w_j \cdot x - \theta_j),
\]

where \( x = (x_1, ..., x_k) \) is the input of a network, \( w_{ij} \) are the weights assigned to the edges leading from the input layer to the hidden layer, \( \sigma \) is a transfer function, \( \theta_i \) are the thresholds, and \( v_j \) are the weights from the hidden layer to the output neuron. Usually the sigmoidal function \( \sigma(x) = 1/(1 + e^{-x}) \) is taken.
It is assumed that the configuration of a neural net (a graph and a number of hidden neurons) and the transfer function are given, and weights and thresholds are adjusted so that the network can perform a given task. Adjustment of weights is called training, and it is done by some learning algorithm designed to minimise the mean square error between the desired and the actual output of the network. The most commonly used learning algorithm is the error backpropagation (BPL), based on the method of gradient descent.

Due to its rich connection structure, a neural network is supposed to learn how to perform a complex task from the examples, just like the human brain does, instead of being given a large set of rules in advance. The training process involves presenting to a network a set of known examples of inputs and corresponding outputs, and continuously adjusting weights until the network output maximally matches the desired output.

The major areas of application of artificial neural networks include a wide variety of image and pattern classification problems, function estimation and regression problems. In recent years, neural networks have been applied extensively to the prediction of future observations of a time series. Having observed a sufficiently long part of the time series, we can try to find the unknown functional relationship between the past and the future observations (7.1) by training a neural network to approximate the unknown autoregression function $F$.

The universal approximation capability of a one-hidden-layer feedforward neural network with sigmoidal activation function follows from the results of Cybenko [21]. He showed that the functions

$$
\sum_{j=1}^{n} v_j \sigma(w_j : z - \theta_j)
$$

are dense in $L_2(\mu)$ (where $\mu$ is a probability measure concentrated on a bounded subset of $R^k$), which makes them candidates to approximate any function in $L_2(\mu)$.

While the one-hidden-layer network with backpropagation learning algorithm provides a very powerful approximation tool, the training may be very slow and inefficient, especially for a large network. An important question is how to choose the optimal size and configuration of the network for a specific problem. The speed of approximation depends on the number
of hidden neurons. Barron [3] proved that a sufficiently smooth function can be approximated by $(7.14)$ in $L_2(\mu)$ at a rate $O(\frac{1}{\sqrt{n}})$. Hence, a network with a number of hidden neurons that is too small will not be able to reach a given error level, while a network that is too large requires too much training time. Training of a neural network involves minimisation of a function of $n(k + 2) + 1$ parameters, and so, a large network cannot be trained properly in reality.

The next section will deal with a constructive learning algorithm of Projection Pursuit Learning (PPL), which is inspired by the Projection Pursuit Regression (PPR). This learning algorithm builds a neural network by dynamically adding hidden neurons, in this way optimising the network size and decreasing learning time.

### 7.3.2 Projection Pursuit Learning

Projection pursuit is a nonparametric regression technique, known from nonparametric statistics, that allows the interpretation of high-dimensional data by considering well-chosen one-dimensional projections.

Let again $X, X_i$ be the $k$-dimensional vectors of explanatory variables, $Y, Y_i$ be the responses, $i = 1, \ldots, N$, and the (unknown) functional relation between $X$ and $Y$ is the regression surface $r$ $(7.6)$. In projection pursuit, $r$ is approximated by the sum of empirically determined univariate functions $g_j$ of linear combinations of explanatory variables, i.e.

$$
\hat{r}_n(z) = \sum_{j=1}^{n} g_j(a_j^T \cdot z),
$$

(7.15)

where $z$ is a vector of observed explanatory variables and $a_j$ is a unit projection vector. The word “pursuit” refers to finding good projection directions by optimisation. The functions $g_j$ (also called the ridge functions) are then estimated nonparametrically, e.g., by kernel or spline smoothing, or using a supersmoother. For a good review on projection pursuit regression see Huber [44].

Note that the structure of PPR $(7.15)$ is similar to that of a neural network $(7.13)$, where the activation functions $g_j$ in each neuron have to be estimated, instead of being fixed in advance (e.g., the sigmoidal function). The algorithm of projection pursuit learning implements PPR into a one-
7.3. Prediction by Neural Networks

hidden-layer neural network

\[ f_n(x) = \sum_{j=1}^{n} v_j g_j(a_j^T x - \theta_j), \]  

(7.16)

where the weights \( v_j, a_j, \theta_j \) and the functions \( g_j \) are adjusted to minimise the mean-squared error between the network output and the desired output.

The following training scheme is considered: the weights and the ridge function corresponding to one hidden neuron are adjusted until there is no further improvement in error level. Then a new neuron is added and procedure is repeated.

Studies show that in general PPL performs better than BPL for model-free regression problems: PPL requires fewer neurons to achieve comparable accuracy and it is less sensitive to outliers. However, due to additional problem of ridge functions estimation, the total learning time is comparable to that of BPL, or even higher.

The learning time will decrease if the activation functions do not have to be estimated. For instance, we again can take the sigmoidal activation function and for the rest carry out PPL in the same way, i.e. adding the hidden neurons one by one. The following “relaxed” variant of projection pursuit regression does just that and, moreover, achieves the desirable approximation accuracy.

Suppose that \( f_n \) has been selected, then \( f_{n+1} \) is selected from the restricted class of functions of the form

\[ (1 - \alpha)f_n(x) + \alpha v \sigma(w_{n+1} \cdot x - \theta_{n+1}), \]

where \( \alpha, v, w_{n+1}, \theta_{n+1} \) are the required parameters. Here the parameter \( \alpha \) plays the role of a “relaxing” parameter. It allows more “space” for improvements of the approximation when \( f_{n+1} \) is being selected. In terms of a neural network this is the following iterative procedure: after having trained a network with \( n \) neurons, until some stopping criteria is reached, all the weights are kept fixed, one more neuron is added and the weights corresponding to this neuron are adjusted together with \( \alpha \). When there is no more improvement in terms of approximation error, one more neuron is added, etc., until the desirable error level is achieved.

The so-called “greedy approximation lemma” of Jones [46] establishes convergence of this procedure in \( L_2(\mu) \) with the approximation rate of
Adding hidden neurons with the sigmoidal activation function one by one substantially decreases learning time in comparison to BPL, where all weights must be adjusted simultaneously. This is confirmed by the numerical examples of the next section. However, replacing the unknown ridge functions by a single sigmoidal function is too restrictive. A more flexible approach would be to add more than one neuron at a time, in this way increasing the richness of the class of admissible ridge functions. We suggest adding neurons in pairs - two sigmoidal functions of the opposite signs form a unimodal function and can successfully approximate peaks and valleys of an unknown regression function. Comparison to a kernel smoothing methods provides an extra motivation for this suggestion.

In kernel smoothing the multivariate case is treated either by product kernels or by considering a distance between vectors. Instead, we may consider a projection pursuit kernel estimator:

\[
\hat{r}_{N,n} = \sum_{j=1}^{n} \sum_{i=1}^{N} \frac{1}{h_j} K \left( \frac{a_j^T \cdot x - a_j^T \cdot X_i}{h_j} \right) Y_i,
\]

where \(a_j\) is a unit projection vector, \(h_j\) is a bandwidth in direction \(a_j\), and we suppose that the sum has been properly normalised. Here \(n\) is the number of hidden neurons and \(N\) is the sample size.

Note that we can rewrite an output of a one-hidden-layer neural network (7.13) as

\[
f_n(x) = \sum_{j=1}^{n} v_j \sigma \left( \frac{a_j^T \cdot x - t_j}{s_j} \right),
\]

where \(|a_j| = 1\). The similarity between (7.17) and (7.18) reveals some connections between these two approaches.

If we set \(\sigma = K\), then both (7.17) and (7.18) perform data smoothing in each direction \(a_j\). The parameter \(s_j\), which is chosen by the backpropagation algorithm, plays the role of the smoothing parameter \(h_j\), which is chosen in kernel methods by the cross-validation method. The main difference between the two methods is that, while the kernel method performs explicit smoothing in all directions \(a_j\) using all the data, a neural network performs implicit smoothing, determining \(v_j\) (instead of \(Y_i/h_j\)) and \(t_j\) (instead of \(a_j^T \cdot X_i\)) by some nonlinear optimisation procedure.
In a neural network the activation function $\sigma$ is the sigmoidal function and does not have the form of a kernel. However, two sigmoidal function of the opposite signs approximate the typical unimodal bell-shape of a kernel function. Moreover, different bandwidths $s_j$ are selected for each half of such an asymmetric kernel, in this way possibly improving approximation capabilities in comparison with symmetric kernel $K$ with a single value of a bandwidth. These considerations support the suggestion of adding hidden neurons in PPL in pairs.

In the next section we shall apply PPL to predict a real-life time series and compare it with BPL in terms of the prediction error and the learning time.

### 7.4 Application to real-life time series

In this section we shall apply the methods described above to the forecasting of two real-life time series, one of which comes from a laboratory experiment and the other one from industry.

#### 7.4.1 Fluidized bed time series

Here we will consider a time series of pressure measurements in a fluidized bed. We used this time series in the introduction as one of our examples of nonlinear time series. Part of the data set together with the delay map at lag 35 is shown again in Fig.7.6 and 7.7. The prediction step for this time series was chosen 35 measurements ahead, and, since the time series is oversampled, we introduce a time delay $\tau = 35$ and look for a predictor of the form:

$$
\hat{y}_{n+35} = F(y_n, y_{n-\tau}, \ldots, y_{n-5\tau}).
$$

We used a larger part of the time series as the basis for predictions, and disjoint smaller parts as test sets. For this time series, as a measure of quality of our predictions, we take the *Average Absolute Error*:

$$
AAE_N = \frac{1}{N} \sum_{j=1}^{N} \frac{|y_{j}^{(t)} - \hat{y}_{j}^{(t)}|}{R}, \quad 100\%,
$$

where $R$ is the range of values of the time series, $y_{1}^{(t)}, \ldots, y_{N}^{(t)}$ is a test set and $\hat{y}_{1}^{(t)}, \ldots, \hat{y}_{N}^{(t)}$ are the obtained predictions.
For application of our kernel method we took the Gaussian kernel function and geometrically decreasing weights $\gamma_i = \gamma^i (i = 1, \ldots, 5)$ for computation of the distance (7.10). Selection of the bandwidth $h$ and the parameter $\gamma$ was done by simultaneous cross-validation. The cross-validation surface is shown in Fig. 7.8, the optimal pair of parameters being $(h, \gamma) = (0.03, 0.6)$.

Fig. 7.9 shows two test sets each consisting of 600 observations (solid line) together with predictions (dashed line). The value of the $AAE$ is
In general, the quality of prediction is quite good, and the dashed curve is rather smooth, but, as we could expect, on highs and lows, where the fluctuations are most noticeable, we get slightly worse predictions than on intermediate parts of the time series.

Next we compare the results with the performance of a neural net trained by the method of backpropagation. Looking again for a predictor of the form (7.19), we used the one-hidden-layer feedforward neural net with 5 inputs, 3 hidden neurons and 1 output, which was trained to be the prediction for a value 35 measurements ahead. We used $10^5$ training iterations performed on the training set of length 3000 observations. Since we used a small neural net (5:3:1), this number of training iterations should be sufficient to train it properly.

The example of a test set together with neural net predictions is shown in Fig.7.10. The obtained $AAE$ is 8.7%. Comparison of Fig.7.9 with Fig.7.10 shows that the kernel method works better not only in terms of average error, but also has a definite advantage in predicting high and low values where the neural net failed.

We also apply local and global linear predictors to this data set. For the $(k, \epsilon)$-method we take into account $\epsilon$-close vectors of $k$ past values, sampled with time delay 35, and we base the choice of the pair $(k, \epsilon)$ on the minimisation of the $AAE$. The optimal pair is $k = 3, \epsilon = 0.2$. A test set together with local linear predictions is shown in Fig.7.11. The $AAE$ is in this case 8.4%, but note that again the predictions of high and low values
are of lower quality.

Note, however, that the predictions obtained by the \((k, \epsilon)\)-method are rather undersmoothed (Fig. 7.11), and the method has a very local character. This is different from the smooth prediction curves obtained by the kernel method, where the parameters were tuned so that no under and oversmoothing occurs, and those obtained by the neural net, which is a more global approximation procedure.

For comparison we also fitted global linear autoregression of the past 5 values sampled with time delay \(\tau = 35\) (as in (7.19), with \(F\) linear). This delivered the \(AAE\) of 13\%, much higher than in applications of nonlinear and locally linear methods. This shows that strong nonlinear dependence in this data set is indeed best captured by applying nonlinear methods of predictions, and that linear methods perform quite poorly in this case.

### 7.4.2 Gas export time series

The time series considered in this section was provided by an industrial company and represents the export of gas by this company to a foreign country per hour in a period of 4 years (1990-1993) (Fig. 7.12). The data contains a clear yearly periodic component, and, if a shorter part of the time series is observed, such as 2-week period (Fig. 7.13), we can also notice the presence of a weekly and a daily periodic components. There is no global trend in the data set.

Standard methods of analysis and prediction of such a time series in-
volve first extracting all the trends and seasonalities from the data to make it stationary, and then fitting a linear (mostly Gaussian) AR or ARMA model to the residuals. Since here we were mostly interested in the non-linear methods of prediction, extraction of all the trends and seasonalities was not necessary. In pre-processing we extracted only the yearly seasonal component, and the weekly and daily seasonal components were kept untouched.

The variation in this time series is clearly multiplicative: fluctuations are larger in a winter, when the gas consumption is higher; the summer parts behave more regularly. This can be eliminated applying logarithmic transformation. However, simulations show that this does not improve predictions. Another way of dealing with this problem the one we used here - is to separate summer and winter months and build separate predictors for them. We used the data from the first 3 years as the basis for predictions and parts of the time series of the last year as test sets.

The prediction step for this first time series follows from the needs of the company and it is 24 hours. As a measure of accuracy of our prediction we took the Average Relative Error (ARE):

\[ ARE_N = \frac{1}{N} \sum_{j=1}^{N} \frac{|y_j^{(t)} - \tilde{y}_j^{(t)}|}{y_j^{(t)}} \times 100\% . \]

(Here again \( y_1^{(t)}, \ldots, y_N^{(t)} \) is a test set and \( \tilde{y}_1^{(t)}, \ldots, \tilde{y}_N^{(t)} \) are the obtained pre-
predictions).

As this time series is most likely weather-dependent, and the saying goes that “the best prediction for the weather of tomorrow is the weather of today”, the fair comparison for all our methods would be the prediction error obtained by taking the present value as the prediction for 24 hours ahead. The \( ARE \) in this case is 8\% for summer months and 16\% for winter months, with the average for the whole test set being 12.5\%.

We choose here for the autoregression model without time delay, i.e. we take into account \( k \) consecutive observations from the past for some \( k \):

\[
y_{n+24} = F(y_n, y_{n-1}, \ldots, y_{n-k+1}) + \epsilon_n. \tag{7.20}
\]

We have applied the methods below for different values of \( k \), such as 12, 24, 48 and 72 (half, 1, 2 or 3 days in the past respectively), and computations show that taking \( k \) too high (48 or 72) does not improve predictions and leads to higher computation time, while both \( k = 12 \) or \( k = 24 \) provides comparable error level and a reasonable computation time for all methods below (taking \( k = 12 \), however, significantly decreases the computation time while only slightly increasing the prediction error).

First we apply the kernel method of prediction. The parameters \( h \) and \( \gamma \) were chosen by the cross-validation procedure on the set of 3000 observations. The following combination of the parameters minimise the cross-validation function:

\[
\begin{align*}
\text{for } k = 12 &: \ h = 0.02, \ \gamma = 0.9, \\
\text{for } k = 24 &: \ h = 0.02, \ \gamma = 0.8.
\end{align*}
\]

We test the performance of the kernel method on the parts of the time series taken from winter and summer months of the last year, and the basis for predictions (“comparison” set) is the data of the second year. While the two above combinations of parameters provided approximately the same error level, note that they are essentially different. This can also be seen observing the average number of vectors (from the comparison set) that fall into the local neighbourhood of vector for which we have to make a prediction. For the first parameter combination this number is 31, and for the second one it is 119. This shows that the error curve has more than one local minimum.

As expected, for data from summer months the predictions are more accurate than for those from winter months. The obtained \( ARE \) is approx-
imately 6% for summer parts of the time series, up to 14% for winter parts, with the average of 11% for the whole test set. On Fig. 7.14 and 7.15 two examples of test sets are shown (solid line) together with predictions 24 hours ahead (dashed line).

![Figure 7.14: Kernel method predictions](image)

Next, we apply a one-hidden-layer feedforward neural network for prediction. Again we assume the autoregression model of the form \( (7.20) \), with \( k = 24 \).

The 24:10:1-neural net was trained, i.e. we took as the input all 24 observations of the previous day, the number of hidden neurons was 10, and 1 output was trained to be the value 24 hours ahead. This neural net was trained on the data from the first year (1990), picking training patterns (i.e. days) from the test set at random. In total \( 4 \cdot 10^5 \) training iterations were performed. Then the neural net was tested on the parts of time series from the year 1993. The obtained \( ARE \) fluctuates from 7% for the summer months to 14% for the winter months, with the average for the whole testing set being 11.5%. Fig. 7.16 shows two training sets with predictions. Note that the dashed line (predictions) seem to be “shifted” to the right with respect to the solid line (real data) by approximately 24 hours. It shows that the neural net has “learned” to approximate the last observation fed into it (i.e. the present value) and take it as a prediction for the value 24 hours ahead. However, this is not entirely true, since the prediction error is smaller than when just taking the present value for prediction.

Training a neural network for this application (24:10:1-network) is a
rather time-consuming process: performing $4 \cdot 10^5$ training iterations took a computation time of a few hours on a SUN workstation. So, in order to speed up the training process, we applied the projection pursuit learning (the relaxed procedure described above). We started with a network with 2 hidden neurons (24:2:1-network) and kept adding hidden neurons in pairs until the same level of the learning error as in the regular BPL was reached. This required much less learning time (less than 1 hour) and 8 hidden neurons instead of the original 10. For comparison we then tested the neural networks trained by PPL and by BPL on the same parts of the time series, but different from the training set. PPL and BPL gave the same prediction error for summer months ($ARE = 7\%$) and PPL gave slightly higher prediction error for winter months ($ARE = 14\%$ for BPL and $ARE = 14.5\%$ for PPL). Fig. 7.18 shows the decrease of the training error in the process of PPL as the hidden neurons are added.

We also wanted to explore the fact that the behaviour of the time series is qualitatively different for different days of the week and that the data of the same day of the week have a lot of features in common. So we trained 7 different neural nets, according to the day of the week in which the predicted value falls. Again we opted for 24:10:1-net, assuming the autoregression model with $k = 24$. Since there are only 52 weeks in a year, the training set for each network can not be as large as when the days of the week are not separated. For our training set we therefore took the data from the first 3 years, and for test set the data from the last year. This
7.4. Application to real-life time series

Figure 7.16: Learning error vs. # of hidden neurons

The approach gave reasonably good predictions for some days of a week and worse for other days: the ARE fluctuates from 7% for Sundays to 17% for Fridays. An example of the data of 2 days from the test set together with predictions is shown in Fig. 7.17.

Figure 7.17: 24:10:1-NN predictions for Sunday and Wednesday

Next, we compared the above methods with linear autoregression. We fit linear autoregressions of order $k = 24$ and $k = 12$:

$$X_{n+24} = a_0 + a_1 X_n + a_2 X_{n-1} + ... + a_k X_{n-k+1} + \epsilon_n,$$

estimating $a_i$'s by the method of least-squares.
Predictions by the linear autoregression of order 24 or of order 12 are of approximately the same precision, the \textit{ARE} is 7\% for predictions in summer months and 15\% in winter months, with the average for the whole test set being 12\%. A familiar effect is also observed here: “approximating” the last known value $X_n$ for prediction of $X_{n+24}$, which can be seen on Fig. 7.18. This effect is also confirmed by observing the estimated regression coefficients. In both cases $k = 24$ and $k = 12$ the regression coefficient $a_1$ is the highest of all, i.e. the latest value gets the biggest “weight” in the model.

![Global linear predictions, k=24, rel.err.=10.91%](image1)

![Global linear predictions, k=12, rel.err.=11.11%](image2)

Figure 7.18: 24-hours predictions by linear autoregression
7.4.3 Comparisons

In this chapter we have applied some nonlinear and locally linear methods to prediction of two real-life time series - pressure fluctuation in the fluidized bed and gas export time series - and compared their performance with each other and with linear methods. The predictions errors corresponding to each method and time series are incorporated in the table below.

<table>
<thead>
<tr>
<th>Method</th>
<th>fl. bed</th>
<th>gas summer</th>
<th>gas winter</th>
<th>gas aver.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>5.8%</td>
<td>6%</td>
<td>14%</td>
<td>11%</td>
</tr>
<tr>
<td>NN: BPL</td>
<td>8.7%</td>
<td>7%</td>
<td>14%</td>
<td>11.5%</td>
</tr>
<tr>
<td>NN: PPL</td>
<td>*</td>
<td>7%</td>
<td>14.5%</td>
<td>11.5%</td>
</tr>
<tr>
<td>Loc. Lin</td>
<td>8.4%</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>AR($k$)</td>
<td>13%</td>
<td>7%</td>
<td>15%</td>
<td>12%</td>
</tr>
</tbody>
</table>

(Here * means that the method was not applied for that time series.)

We can see that the kernel method has a considerable advantage over all other methods for the fluidized bed time series. This is mostly because the fluidized bed time series is a good example of a chaotic time series, and the local nonlinear character of the kernel method could best capture its local dynamics. The locally linear predictor ($k, \epsilon$)-method and the neural net showed comparable performance, while the linear method performed much worse. This confirms the suggestion that for chaotic time series nonlinear and locally linear prediction methods are preferred to traditional linear ones.

For the gas export time series it was difficult to conclude which method should be preferred. The average prediction error level was slightly smaller for the kernel method, which again could be attributed to its local character, but it was still comparable to the performance of the neural net. On the other hand, the neural network, having more global approximation features, was able to capture better the global structure of the gas data, such as daily and weekly fluctuations. The performance of linear predictions was slightly worse than that of the neural network, but it was still hard to say whether the kernel method or the neural net definitely outperformed the linear autoregression. The greatest advantage of the linear method over both nonlinear ones was its low computational time.
In general, the comparison in terms of the computation time shows that nonlinear methods of prediction are more time-consuming than the linear ones. For the neural network the learning process (i.e., training of a neural net by BPL) takes most of the computation time, while making predictions with the help of a trained network is very fast. As the application to the gas export time series shows, the learning time can be decreased considerably with the help of a relaxed PPL algorithm, without significant loss of prediction accuracy.

For the kernel method, the cross-validation procedure (which is analogous to "training") is not as time-consuming as training of a neural net by BPL. Computation of predictions, however, is rather slow, since it involves comparing the vector of most recent observations (i.e., the one for which we want to make prediction) with all vectors of previous observations, i.e., with a larger part or the whole time series.

For the practical point of view the research worker will look for additional information to improve the forecasts. In many cases, such as fluidized bed, only one univariate time series is available. However, in many practical situations there are ways to obtain more information about the underlying system, e.g., by measuring more than just one quantity or by considering other time series that may be relevant for describing the underlying phenomena. For example, in the case of gas export time series one may use one or more characteristics of the weather prediction for tomorrow as explanatory variables, as well as some other factors, such as the day of the week, or whether tomorrow is a workday or not. These variables are known the day before and may lead to a dramatical decrease of prediction error. In such cases multivariate time series become the subject of analysis. Multivariate time series are particularly important in economics and econometrics studies, where many related factors should be investigated. Analysis of multivariate time series and, especially, extensions of nonlinear techniques to multivariate case remained outside the scope of this manuscript, mostly because univariate case is already complex enough and provided us with many interesting problems. However, we are in no way trying to diminish the great importance of the case of multivariate time series and consider this as a subject of future research.