Neural Network Based Lag Selection, for Multivariate Time Series

A. Sarishvili[1]

26.02.2002

.

Vom Fachbereich Mathematik der
Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte Dissertation

.

1. Gutachter Prof. Dr. J. Franke
2. Gutachterin Prof. Dr. U. Gather

.

D 386

[1]e-mail alex.sarishvili@itwm.fhg.de

I thank all the people who helped me to write this thesis. Especially the thank Prof. J. Franke for decisive tips in the theoretical part of the thesis, Prof. U. Gather for giving me the possibility of working with biometric data in team of the university of Dortmund, Dr. G. Kroisandt for interesting discussions and his useful comments.

Contents

Introduction

Since nonlinear artificial neural networks are known to be capable of identifying relationships between input data that are not apparent to human analysis(Weigend et al. 1990) one hope has been that the network could be utilized to identify relationships in clinical and other similar data that have not been revealed by previous studies.

Time series analysis has three goals:

- Predicting,
- Modeling,
- Characterization.

The aim of predicting is the prediction of the short term evolution of the system. The goal of modeling is to find a description that accurately captures features of the long term behaviour of the system. The third goal, system or process characterization, attempts with little or no a-priori knowledge to determine fundamental properties, such as the number of degrees of freedom of a process or the amount of randomness.

Many examples of time series are important in engineering and science. The ability to predict the activity of stocks and other financial instruments carries with it major implications for how investing and securities trading are carried out. Chemical engineers have studied the chaotic behaviour of some chemical reactions as a time series problem in order to improve control over the rates at which these processes proceed. There are many important time series in medicine. For example, the white blood cell count of a cancer patient must be monitored and controlled. Decisions regarding drug dosages for such a patient can be greatly aided by predictions of the white blood cell count time series.

Many other chemical relationships in the body, such as the blood glucose and insulin concentrations, can also be studied as time series. In addition, the EEG and ECG time series are of great interest.

Time series exhibit reasonably well-understood behaviours. Often a time series is composed of a long-term trend plus various periodic and random components. Linear and periodic components are usually easy to model and remove from the time series. The prediction of this random component is often the focus of the time series analysis.

The random component of a time series usually falls into one of two categories. In the first case, the random component is truly random, that is the measurements are drawn from some underlying probability distribution. In this case, the random component can be described by a statistical distribution function or by the statistical moments of the data: mean, variance, skew, kurtosis, and so on.

The second class of apparently random behaviour in time series is not random at all, but rather, chaotic. A chaotic time series is characterized by values that appear to be randomly distributed and non periodic but are actually the result of a completely deterministic process. The deterministic behaviour in chaotic time series is usually due to underlying nonlinear dynamics.

It is desirable to model a time series but, one might ask, how do we go about modeling time series? Ideally, we would like to use past data to construct a set of basic rules, that can be used to model a component of a multivariate time series. Unfortunately, this approach cannot always be carried out in practice. In many cases, the underlying principles are not known, or are poorly understood because the system of interest(multivariate time series) is very complicated. Another problem with this approach is that often, even when the basic laws are known a direct solution of the equations is not possible without detailed information about initial values and boundary conditions. In practice, even barring the possibility of turbulence in the system, it is

often important to make enough measurements to specify the system sufficiently.

In a second approach to time series analysis one avoids these problems by making the assumption that a well-defined relationship exists between the past and future values of a single observable. In this phenomenological approach one seeks an approximate functional relationship between past values and the future values one wants to calculate. There are various ways to model this relationships. Here we try to find a "single" function that gives a future value of the observable as its output when some set of past observations is supplied as its input. This form of modeling is implemented by artificial neural networks(ANNs).

One of the first applications of artificial neural networks to clinical medicine and financial market analysis, was pattern recognition. One example of such real neural network applied in clinical medicine is given by W.G. Baxt 95. Many processes from clinical medicine like biological, pathopsychological, or from financial market analysis manifest a "long range" chaotic behaviour. Many available techniques for time series analysis assume linear relationships among variables(see Box and Jankins, 1970). But in the real world, temporal variations in data do not exhibit simple regularities and are difficult to analyze and predict accurately. Linear recurrence relations and their combinations for describing the behaviour of such data are often found to be inadequate. It seems necessary, therefore, that nonlinear models must be used for the analysis of real world temporal data. Tong(1983) describes some of the drawbacks of linear modeling for time series analysis. These include among other things, their inability to explain sudden bursts of very large amplitudes at irregular time intervals. Tiao and Tsay (1989) addresses some of the problems with linear models for multivariate time series. In order to accommodate for such inabilities, nonlinear statistical methods, such as the threshold model and

the bilinear model, have been suggested and discussed by Tong(1990),

whereas Granger and Newbold (1986) suggest the use of nonlinear transformation of the original data before performing the usual linear modeling. Farmer and Sidorowich(1987) report a significantly better prediction for chaotic time series by using a local approximation approach involving use of nearest k-neighbors with respect to the magnitudes of the values rather than with respect to time points to which the values belong.

Despite considerable progress over the last decades, formulation of reasonable nonlinear models is an extremely difficult task because of simplifications made in the modeling stage, e.g. omitting parameters which are unknown or which do not seem to affect the observed data directly. Also the relationships between known parameters and observed values can only be hypothesized with no simple laws governing their mutual behaviour. The first article from (Cross, et.al.1995) explained how non linear processing, as afforded by an artificial neural network, might improve upon the predictive power of the other approaches. The hope has been that the network's ability to identify multidimensional relationships in clinical data not apparent to other forms of analysis would allow the network to improve diagnostic accuracy.

Nonlinear statistical methods have been tried before, but they were complex and computationally intensive. The advantage of the neural network and the much greater speed of computers have changed this.

We distinguish between two kinds of applications of artificial neural networks in this context:

- Neural network pattern recognition,
- Time series modeling with neural networks.

One possibility to model multivariate time series with neural

networks is presented in this thesis.

Thesis overview

We consider the case that the value taken by a component of a multivariate time series is at least partially determined by the values taken by other components of the same time series, either at the same time or in the past. Models based on such ideas are often called causal models, although they are usually just observed, empirical relationships.

An explanatory model is then one that presents an equation explaining the value of the observed component of interest in terms of present values of one or more other components of the same process, and possibly also the past values of these components.

In the first chapter we present and estimate an explanatory model with a predefined system of explanatory equations, a so called lag dependent model. We present a lag estimator and theorems about consistency.

We consider a dynamic nonlinear equation system of the form

$$y_t = m_t(y_{t-L_t}) + \epsilon_t, \quad t \in \mathcal{N}, \tag{1}$$

where ϵ_t is i.i. $N(0, \sigma^2)$ distributed and the functions $m_t : R \to R$ are Borel measurable and twice differentiable for each $t \in \mathcal{N}$. Let y_0 denote some starting value. Then we define change points as follows:

Definition 1. We define the change point(CP) for the dynamical process (1) as follows

- *change point* in t, with respect to the time delay variables, if $L_{t-1} \neq L_t$,
- *change point* in t, with respect to the regression function class, if $m_{t-1}(x) \neq m_t(x)$,

Most dynamical processes are so complex that their universal theory, capturing all the details during all time periods, is unimaginable. That is the reason why the purpose of mathematical modeling is to model only fundamental aspects of the process, and to neglect insignificant features without losing relevant information. But, in many cases even vanishingly small forces can cause large changes in dynamical system parameters. Such situations are intuitively associated with the concept of instability. In other words we cannot consider any little deviation from the model as an insignificant feature. However, since they may be indistinguishable by human eye in the beginning, there is no way to incorporate them into a modeling procedure. This simply means that if the model is not able to describe the underlying dynamical process, it must be changed.

But what is the reason, for the generation of the instability? We can distinguish between two main reasons,

- First kind
  Instability by the inconsistency between models and reality.
- Second kind
  Instability as a consequence of biological, physical, etc. phenomena.

We note that in the second case the terms "instability" and "change point" are connected. Both kinds of instabilities are illustrated in figure 1.

In many cases the instability is an attribute of the mathematical model rather than of a natural phenomenon. Therefore the first step before we estimate any parameters of any model, is to be sure that the selected model is able to describe the true underlying structure of the process. If that is guaranteed, we can detect the deviations from this

model, i.e. recognize the change points, and make statements about any relevancy of detected change points for the observer.

If we select the correct model we can distinguish between different instabilities of the second kind (change points), and store them for further analysis.

We discriminate in this way three types of instabilities,

- known instability from its own past;
- known instability from experience;
- unknown instability.

The case of a known instability from its own past occurs if the current "state" (pattern) (will be defined in first chapter) of a component can be classified in a CSL (Component State Library) estimated in the past of the same component. An example for this kind of instability is the "similar" reaction of Dow Jones to oil price fluctuation in different periods.

Known instability from experience is the process state that cannot be matched to any state in CSL. An example for this kind of instability, is the picture of several blood pressure values during first cardiac infraction by many patients at the same age.

Unknown instability is detected if the current state is "new" in the sense described in section 3.2.

The figure 1 shows different modes of origins for a detected instability in a mathematical model.

In the third chapter the algorithms for change point detection and classification are presented. In the section 3.5 the bootstrap method for approximating the distribution of statistics of interest in dependent processes is presented.

The fourth chapter stands for application of lag dependent modeling on real and simulated data. We choose, biometric and financial

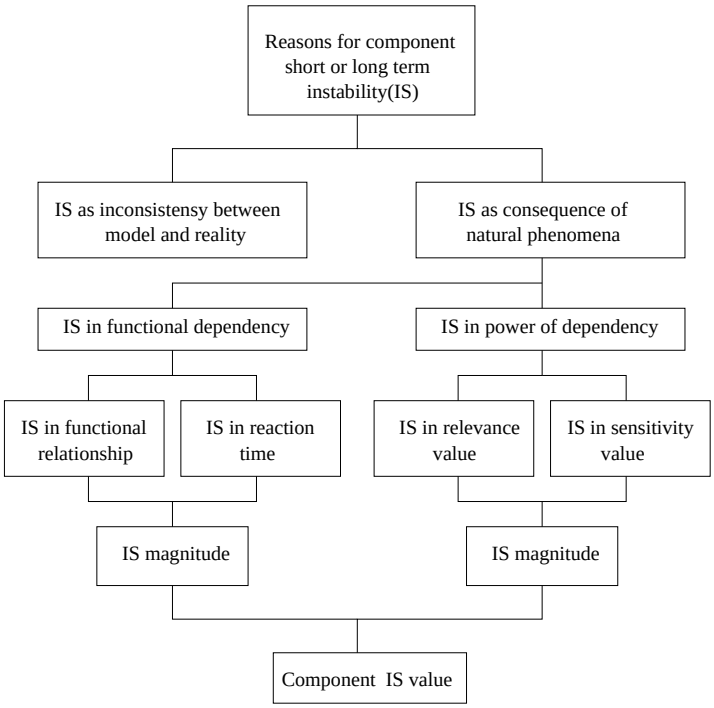

Figure 1: Instability in a mathematical model

market data for this aim.

Data from simulated time series, biometric time series and financial time series was estimated with lag dependent models. One step predictions were made in the fourth chapter.

The appendix includes known results from time series analysis and neural networks that we have been using.

Chapter 1

Lag Dependent Models(LDM)

A simple example

Before we present the multivariate lagged nonlinear time series models, to be considered in this thesis, in their full complexity we look at a simple example which does not suffer from the necessarily involved notation from the general case.

We define bivariate lagged additive NLAR(1)-process $(y_t^{(1)}, y_t^{(2)})^T$

$$\begin{aligned} y_t^{(1)} &= m_{11}(y_{t-l_{11}}^{(1)}) + m_{12}(y_{t-l_{12}}^{(2)}) + \epsilon_t^{(1)} \\ y_t^{(2)} &= m_{21}(y_{t-l_{21}}^{(1)}) + m_{22}(y_{t-l_{22}}^{(2)}) + \epsilon_t^{(2)}, \end{aligned}$$

where $\epsilon_t^{(1)}, \epsilon_t^{(2)}$ i.i.d zero-mean innovations. Furthermore we allow the model to change simultaneously its regression functions $m_{ij}, \quad i,j \in \{1,2\}$ and time lags $l_{ij}, \quad i,j \in \{1,2\}$ in time.

In the course of this thesis we estimate m_{ij} nonparametrically by fitting feedforward neural networks with block structure to the data for

each given lag vector $L = (l_{11}, l_{12}, l_{21}, l_{22})$. We minimize with respect to the lag vector L the average one-step forecasting error of the for given L optimal network and compute measure of relevancy R_{ij} of component j for component i which is average derivative or related statistic. Finally we use the estimated lag vector L and the relevance measure R to detect the change points in this process.

1.1 Model definition

Let $Y_t, \quad t = 1, ..., N$, $Y_t \in \mathcal{R}^d$ be a d-dimensional time series and $y_t^i \in \mathcal{R} \quad i = 1, ..., d, \quad t = 1, ..., N$, the i-th component. We describe a nonlinear regression problem for time series and assume that a true function m that describes the relationship between two components y^i and y^j exists, where, for the moment, i and j are fixed. Then we can write

$$m^{ij}(X) = E\{y^i | y^j = X\}.$$

We assume furthermore that the function m^{ij} is not the same over a large interval; it is different in each unique partition of the interval $[1, N]$, i.e. we have a partition of the time interval $[1, N]$ in $q^{ij} + 1$ parts with the set of change points in regression function class, $\Xi^{ij} = \{\xi_1^{ij}, ..., \xi_{q^{ij}}^{ij}\}, \quad q^{ij} \in \mathcal{N}$. First, we assume that we have an autoregressive structure of order 1. Then we can write that:

$$\begin{aligned} y_t^i &= m_1^{ij}(y_{t-1}^j) + \epsilon_t, \quad t \in \{1, ..., \xi_1^{ij}\}, \\ &... \\ y_t^i &= m_{q^{ij}+1}^{ij}(y_{t-1}^j) + \epsilon_t, \quad t \in \{\xi_{q^{ij}}^{ij}, ..., N\}, \end{aligned} \tag{1.1}$$

with $\epsilon_1, ..., \epsilon_N$ independent and identically $N(0, \sigma^2)$ distributed, $\sigma^2 = E(\epsilon_t^2) < \infty$, and $m_k^{ij} \neq m_{k+1}^{ij}, \quad k = 1, ..., q^{ij} + 1$. In the next step of model construction we allow the model to vary the time delay values

of explanatory variables, i.e. we can modify the equations from 1.1 in this manner:

$$\begin{aligned} y_t^i &= m_1^{ij}(y^j_{t-l_1^{ij}}) + \epsilon_t, \quad t \in \{1, ..., \xi_1^{ij}\}, \\ &... \\ y_t^i &= m^{ij}_{q^{ij}+1}(y^j_{t-l^{ij}_{q^{ij}+1}}) + \epsilon_t, \quad t \in \{\xi^{ij}_{q^{ij}} + 1, ..., N\}, \end{aligned} \tag{1.2}$$

with time delay vectors for component pair $y^j \to y^i, \quad i, j \in \{1, ..., d\}$

$$\begin{aligned} L_\Xi^{ij} &= \left(l_1^{ij}, ..., l^{ij}_{q^{ij}+1}\right)', \text{ where} \\ l_k^{ij} &\in \mathcal{N} \ , \forall\, k = 1, ..., q^{ij} + 1, \quad i, j \in \{1, ..., d\}, \end{aligned}$$

and vector of corresponding functions

$$M_\Xi^{ij} = \left(m_1^{ij}, ..., m^{ij}_{q^{ij}+1}\right).$$

For every component pair $y^j \to y^i, \quad i, j = 1, ..., d$ we must realize that we have vector Ξ of partitions, i.e.

$$\Xi^{ij} = \{\xi_1^{ij}, ..., \xi^{ij}_{q^{ij}}\},$$

and for one component i we have d possibly different partitions of given interval, i.e.

$$\Xi^{ij} = \{\xi_1^{ij}, ..., \xi^{ij}_{q^{ij}}\}, \quad j = 1, ..., d,$$

if we consider its dependence on the other component separately. Furthermore we have the following time delay and corresponding function vectors:
for first component

$$\begin{aligned} L^{11}_{\Xi^{11}} &= \left(l_1^{11}, ..., l^{11}_{q^{11}+1}\right), \quad M^{11}_{\Xi^{11}} = \left(m_1^{11}, ..., m^{11}_{q^{11}+1}\right), \\ L^{12}_{\Xi^{12}} &= \left(l_1^{12}, ..., l^{12}_{q^{12}+1}\right), \quad M^{12}_{\Xi^{12}} = \left(m_1^{12}, ..., m^{12}_{q^{12}+1}\right), \\ &... \\ L^{1d}_{\Xi^{1d}} &= \left(l_1^{1d}, ..., l^{1d}_{q^{1d}+1}\right), \quad M^{1d}_{\Xi^{1d}} = \left(m_1^{1d}, ..., m^{1d}_{q^{1d}+1}\right), \end{aligned}$$

for second component

$$\begin{aligned}
L^{21}_{\Xi^{21}} &= \left(l^{21}_1,...,l^{21}_{q^{21}+1}\right), \quad M^{21}_{\Xi^{21}} = \left(m^{21}_1,...,m^{21}_{q^{21}+1}\right),\\
L^{22}_{\Xi^{22}} &= \left(l^{22}_1,...,l^{22}_{q^{22}+1}\right), \quad M^{22}_{\Xi^{22}} = \left(m^{22}_1,...,m^{22}_{q^{22}+1}\right),\\
&...\\
L^{2d}_{\Xi^{2d}} &= \left(l^{2d}_1,...,l^{2d}_{q^{2d}+1}\right), \quad M^{2d}_{\Xi^{2d}} = \left(m^{2d}_1,...,m^{2d}_{q^{2d}+1}\right),
\end{aligned}$$

for d-th component

$$\begin{aligned}
L^{d1}_{\Xi^{d1}} &= \left(l^{d1}_1,...,l^{d1}_{q^{d1}+1}\right), \quad M^{d1}_{\Xi^{d1}} = \left(m^{d1}_1,...,m^{d1}_{q^{d1}+1}\right),\\
L^{d2}_{\Xi^{d2}} &= \left(l^{d2}_1,...,l^{d2}_{q^{d2}+1}\right), \quad M^{d2}_{\Xi^{d2}} = \left(m^{d2}_1,...,m^{d2}_{q^{d2}+1}\right),\\
&...\\
L^{dd}_{\Xi^{dd}} &= \left(l^{dd}_1,...,l^{dd}_{q^{dd}+1}\right), \quad M^{dd}_{\Xi^{dd}} = \left(m^{dd}_1,...,m^{dd}_{q^{dd}+1}\right),
\end{aligned}$$

with $l^{ij}_k \in \mathcal{N} \quad i,j = 1,...,d, \quad k = 1,...,q^{ij}+1$. The value of l^{ij}_k is the time delay in modeling the dependence of component i on component j in partition interval with index k.

We want to concentrate on changes in the lag structure of the time series. Therefore, we exclude the case where a change is happening only in the regression functions but the delays stay unchanged: We note an assumption,

Assumption 1.1. *If some of regression functions m^{ij}_k are changing from time ξ^{ij}_k to time ξ^{ij}_k+1, then at least one of the lags l^{ij}_k is changing too.*

Processes which contains other types of change points are not considered here. We summarize the model construction steps as follows: we augment with 0 the columns with a small length, then we can write the parameters in matrix form as follows,

- Interval partition coefficients,

$$\Xi^i = \left((\Xi^{i1})',...,(\Xi^{id})'\right)', \quad \forall i \in \{1,...,d\}.$$

and we combine the Ξ^i to a large matrix

$$\Xi = \left(\Xi^1, ..., \Xi^d\right),$$

where, if necessary we augment the Ξ^i by zero rows such that they all have the same number of rows. As we use Ξ not for calculations but only as a compact notation for the tuple of matrices $\Xi^1, ..., \Xi^d$, it is not necessary to distinguish between the original Ξ^i and their augmented counterparts.

- Time delay matrices,

$$L^i = \left(L^{i1}_{\Xi^{i1}}, ..., L^{id}_{\Xi^{id}}\right), \quad \forall i \in \{1, ..., d\},$$

and, as the Ξ^i, we write

$$L = \left(L^1, ..., L^d\right).$$

- True function matrices,

$$M^i = \left(M^{i1}_{\Xi^{i1}}, ..., M^{id}_{\Xi^{id}}\right), \quad \forall i \in \{1, ..., d\},$$

and, as the Ξ^i, we write

$$M = \left(M^1, ..., M^d\right).$$

Definition 1.1. The triple (Ξ, L, M) is called the **state flow** of a multivariate time series in the time interval [1,...,N].

We define a lag dependent model for one relevant component i.

Definition 1.2. We write the model from (1.2) in the form of a threshold models and call this a LD(1) (Lag Dependent) model in the time

interval $[1, N]$ with state flow (Ξ, L, M), we assume that all time lags are from the interval $[1, A]$ where $A \in \mathcal{N}$.

$$y_t^i = \begin{cases} m_1^{ij}(y^j_{t-l_1^{ij}}) + \epsilon_t & : \quad A < t \leq \xi_1^{ij} \\ m_2^{ij}\left(\left(1 - I_{[\xi_1^{ij}, \xi_1^{ij}+b_1]}(t)\right)(y^j_{t-l_2^{ij}})\right) + \epsilon_t & : \quad \xi_1^{ij} < t \leq \xi_2^{ij} \\ m_3^{ij}\left(\left(1 - I_{[\xi_2^{ij}, \xi_2^{ij}+b_2]}(t)\right)(y^j_{t-l_3^{ij}})\right) + \epsilon_t & : \quad \xi_2^{ij} < t \leq \xi_3^{ij} \\ \ldots \quad : \quad \ldots \\ m^{ij}_{q^{ij}+1}\left(\left(1 - I_{[\xi^{ij}_{q^{ij}}, \xi^{ij}_{q^{ij}}+b_{q^{ij}}]}(t)\right)(y^j_{t-l^{ij}_{q^{ij}+1}})\right) + \epsilon_t & : \quad \xi^{ij}_{q^{ij}} < t \leq N \end{cases}$$

For any $j \in \{1, ..., d\}$, where d is the dimension of the time series, ϵ_t i.i. $N(0, \sigma^2)$ distributed with finite variance and $b_c = l^{ij}_{c+1} - l^{ij}_c$.

We can see the similarity between LD and TNAR models (see section A.1.3); in TNAR models the threshold is chosen with respect to observation value, whereas in LD models the threshold is chosen with respect to the position of the observation on the time axis. Before we can define a lag dependent model we present an important assumption;

Assumption 1.2. *The conditional expectation function* $M^i : R^d \longrightarrow R$,

$$M^i(X) = E\{y_t^i | (y^1_{t-l_{q^{i1}}}, ..., y^d_{t-l_{q^{id}}}) = X\}.$$

has an additive structure

$$M^i(X) = m^{i1}(y^1_{t-l_{q^{i1}}}) + ... + m^{id}(y^d_{t-l_{q^{id}}}), \;\; i \in \{1, ..., d\},$$

where $m^{ij} : R \longrightarrow R, \;\; j = 1, ..., d.$

Definition 1.3. For a given state flow (Ξ, L, M) we can define LD(d)(Lag Dependent with dimension d), for any component $y^i, \;\; i \in \{1, ..., d\}$ in interval $[1, N]$,

$$y_t^i \;\; = \;\; m^{i1}_{k(1)}\left(y^1_{t-l^{i1}_{k(1)}}\right) + m^{i2}_{k(2)}\left(y^2_{t-l^{i2}_{k(2)}}\right) + ... + m^{id}_{k(d)}\left(y^d_{t-l^{id}_{k(d)}}\right) + \epsilon_t$$

where $\xi^{ij}_{k(j)} < t \leq \xi^{ij}_{k(j)+1} \quad \forall i,j = 1,...,d, \quad k(j) \in \{0,...,q^{ij}\}$, ϵ_t i.i. $N(0,\sigma^2)$ distributed with finite variance, $\xi^{ij}_0 = A, \quad \forall j = 1,...,d.$

Following example shows a main problem which can be arised during the process generation with definition 1.3.

Example

Consider simple linear LD(1) process $y_t = ay_{t-L} + \epsilon_t$, with $a \in \mathcal{R}$, $L = l_1$ if $t \in \{1,...,\xi\}$ and $L = l_2$ if $t \in \{\xi+1,...,N\}$. Assume that $l_2 - l_1 = 2$ and that we have a change point at $t = \xi$. Then the generation mechanism of the variable y at time $t_1 = \xi - 1$ and at time $t_2 = \xi + 1$ is respectively $y_{t_1} = ay_{t_1-l_1} + \epsilon_t$ and $y_{t_2} = ay_{t_2-l_2} + \epsilon_t$ but $t_2 - l_2 = \xi + 1 - l_2 = \xi + 1 - 2 - l_1 = \xi - 1 - l_1 = t_1 - l_1$. We see that two different observations in two different process states in the sense of LD modelling, are generated with the same explanatory variable $y_{t_1-l_1} = y_{t_2-l_2}$. To avoid the problems like this one we modify our definition 1.3 in the following manner:

Definition 1.4. For a given state flow (Ξ, L, M) we can define LD(d)(Lag Dependent with dimension d), for any component $y^i, \quad i \in \{1,...,d\}$ in interval $[1,N]$,

$$\begin{aligned} y^i_t \quad = \quad & m^{i1}_{k(1)}\left(\left(1 - I_{[\xi^{i1}_{k(1)},\xi^{i1}_{k(1)}+b^{i1}_{k(1)}]}(t)\right)(y^1_{t-l^{i1}_{k(1)}})\right) + \\ & + m^{i2}_{k(2)}\left(\left(1 - I_{[\xi^{i2}_{k(2)},\xi^{i2}_{k(2)}+b^{i2}_{k(2)}]}(t)\right)(y^2_{t-l^{i2}_{k(2)}})\right) + \\ & + ... + \\ & + m^{id}_{k(d)}\left(\left(1 - I_{[\xi^{id}_{k(d)},\xi^{id}_{k(d)}+b^{id}_{k(d)}]}(t)\right)(y^d_{t-l^{id}_{k(d)}})\right) + \epsilon_t \end{aligned}$$

where $\xi^{ij}_{k(j)} < t \leq \xi^{ij}_{k(j)+1} \quad \forall i,j = 1,...,d, \quad k(j) \in \{0,...,q^{ij}\}$, ϵ_t i.i. $N(0,\sigma^2)$ distributed with finite variance, $\xi^{ij}_0 = A, \quad \forall j = 1,...,d$ and $b^{ij}_{k(j)} = l^{ij}_{k(j)+1} - l^{ij}_{k(j)}$.

The indicator function avoids the problems by the beginning of the new state. If the new time lag is larger then the previous time lag, the indicator function switches off the corresponding component from the modeling procedure for b time steps.

Example

Figure 1.1 describes a component in a bivariate time series and shows that the first component is partitioned in the given interval in three parts with respect to itself, the time axis $1(1)$, and in three parts with respect to the second component , the time axis $1(2)$. In this case, the partition coefficients for each component have different values, and we have the effect of state overlapping.

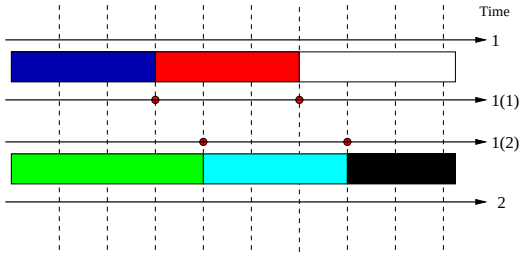

Figure 1.1: Overlap states of the first component.

1.2 Example for an LD(2) model

To simulate a time series from model LD(2) we need initial values. For simplicity we take for the first 30 observations two white noise processes, with different means, i.e.

$$\begin{aligned} y_t^1 &= c_1 + \epsilon_t^1, \quad t = 1, ..., 30 \\ y_t^2 &= c_2 + \epsilon_t^2, \quad t = 1, ..., 30. \end{aligned}$$

with ϵ_t^i i.i. $N(0,\sigma_i^2)$ distributed with finite variances for $i = 1, 2$.

At this stage we specify the functional dependence and time delay values. We assume that the change points occur for each component in the same time, then we have,

$$M^1_{\Xi^1} = \begin{bmatrix} \frac{c_1}{2} & x & \frac{1}{\exp(x)} & \frac{1}{30}(x)+5 & \frac{2}{\exp(x)} \\ \frac{c_2}{2} & 2x & \frac{1}{10}x & \frac{1}{231\exp(x)} & \frac{1}{6}x \end{bmatrix},$$

and for the second component we assume the following function combinations:

$$M^2_{\Xi^2} = \begin{bmatrix} \frac{c_1}{2} & \frac{1}{2}x & \frac{1}{6}x & x & \frac{1}{21}x \\ \frac{c_2}{2} & \frac{1}{6}x^2 & \frac{1}{\exp(x)} & 0.31x^2 & \frac{2}{5}x \end{bmatrix}.$$

We choose for the first component, as time delay values

$$L^1_{\Xi^1} = \begin{bmatrix} 0 & l_2^{11}=3 & l_3^{11}=10 & l_4^{11}=1 & l_5^{11}=10 \\ 0 & l_2^{12}=6 & l_3^{12}=6 & l_4^{12}=5 & l_5^{12}=6 \end{bmatrix}, \tag{1.3}$$

and for the second component,

$$L^2_{\Xi^2} = \begin{bmatrix} 0 & l_2^{21}=5 & l_3^{21}=7 & l_4^{21}=2 & l_5^{21}=7 \\ 0 & l_2^{22}=1 & l_3^{22}=1 & l_4^{22}=4 & l_5^{22}=1 \end{bmatrix}, \tag{1.4}$$

Zeros in (1.3) and (1.4) are standing for the constant part of the model at the beginning. Furthermore we specify corresponding interval partition coefficients: $\xi_1^i = 30$, $\xi_2^i = 60$, $\xi_3^i = 120$, $\xi_4^i = 150$ for $i = 1, 2$. Now we can simulate a bivariate time series on the interval $1 \le t \le 210$. Figure 1.2 shows the generation mechanism of this time series, particularly in the interval number three, (see third column of (1.3) and (1.4)).

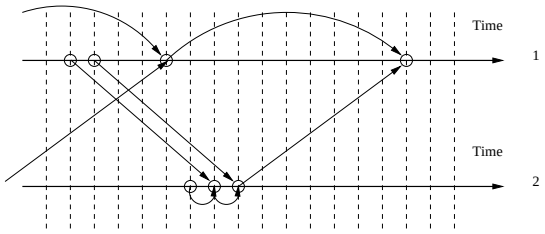

Figure 1.2: Generation of time series values in the LD(2) model.

The LD(2) model shows the following: For first component

$$y_t^1 = \begin{cases} c_1 + \epsilon_t & : \quad 1 \leq t \leq 30 \\ m_2^{11}(y^1_{t-l_2^{11}}) + m_2^{21}(y^2_{t-l_2^{21}}) + \epsilon_t & : \quad 31 \leq t \leq 60 \\ m_3^{11}(y^1_{t-l_3^{11}}) + m_3^{21}(y^2_{t-l_3^{21}}) + \epsilon_t & : \quad 61 \leq t \leq 120 \\ m_4^{11}(y^1_{t-l_4^{11}}) + m_4^{21}(y^2_{t-l_4^{21}}) + \epsilon_t & : \quad 121 \leq t \leq 150 \\ m_5^{11}(y^1_{t-l_5^{11}}) + m_5^{21}(y^2_{t-l_5^{21}}) + \epsilon_t & : \quad 151 \leq t \leq 210, \end{cases}$$

for second component,

$$y_t^2 = \begin{cases} c_2 + \epsilon_t & : \quad 1 \leq t \leq 30 \\ m_2^{12}(y^1_{t-l_2^{12}}) + m_2^{22}(y^2_{t-l_2^{22}}) + \epsilon_t & : \quad 31 \leq t \leq 60 \\ m_3^{12}(y^1_{t-l_3^{12}}) + m_3^{22}(y^2_{t-l_3^{22}}) + \epsilon_t & : \quad 61 \leq t \leq 120 \\ m_4^{12}(y^1_{t-l_4^{12}}) + m_4^{22}(y^2_{t-l_4^{22}}) + \epsilon_t & : \quad 121 \leq t \leq 150 \\ m_5^{12}(y^1_{t-l_5^{12}}) + m_5^{22}(y^2_{t-l_5^{22}}) + \epsilon_t & : \quad 151 \leq t \leq 210. \end{cases}$$

Figure 1.3 shows the plot of this bivariate time series. To show that good approximation results also can be achieved with a false model, we model the first component from the simulated time series in figure 1.3 and estimate an AR(p) model. The AIC reach its minimum at $p = 5$. The solid line time series in 1.4 is the same as in 1.3 but in the time interval $[50, 200]$, and the dashed time series in 1.4 is the fitted AR(5) model.

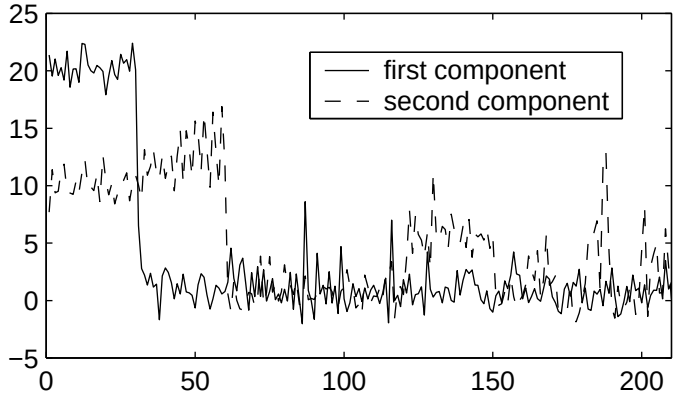

Figure 1.3: Simulated time series from model LD(2).

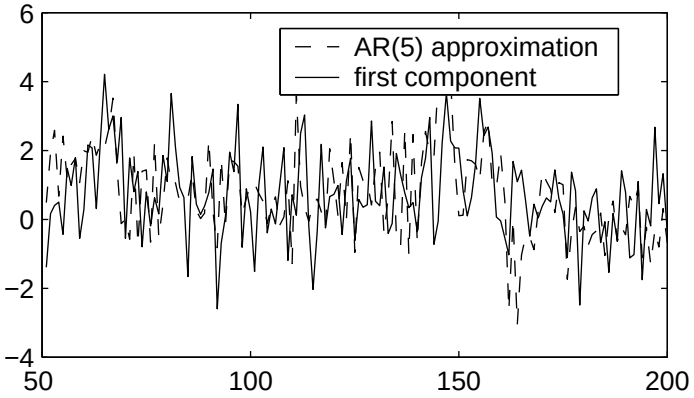

Figure 1.4: AR(5) approximation of LD(2) model.

In the next section we develop an algorithm for estimation of the unknown parameters from the LD model, with blocked feed-forward

neural networks.

1.3 NN LD model estimation

It is easy to show that a feed-forward neural network with one hidden layer and H neurons ($H \geq d$) in the hidden layer is able to estimate the composite function M.

If we take into account that the composite function M in particular is a result of addition of maximal d functions of one real variable, then the neural network without "nonparallel" connections seems to have the right architecture to estimate parameters from LD models. Figure 1.5 shows a neural network with d inputs, one output and without nonparallel connections(blocked neural network). Each neuron in the hidden layer accepts only one variable as input apart from the constant $x(0) = 1$.

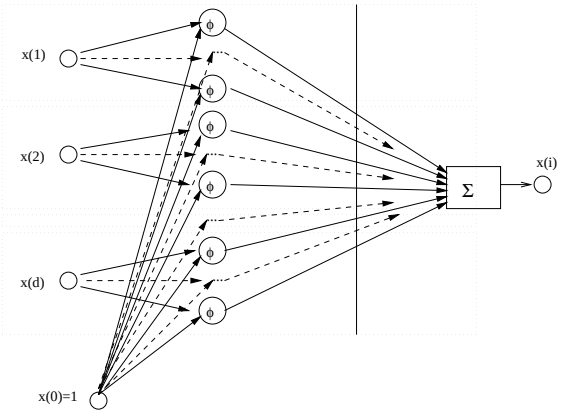

Figure 1.5: Blocked neural network.

The following theorem guarantees, that such a neural network

is able to estimate the composite function M. This theorem follows directly from theorem (B.5).

Theorem 1.3. *Let $\phi(y)$ be a non constant, bounded and monotonic increasing continuous function.*

Let K be a compact subset of R^d and fix an integer $k \geq 1$. Then any continuous mapping $F : K \longrightarrow R$ with $F(y^1, ..., y^d) = f_1(y^1) + ... + f_d(y^d)$ which is a composition of functions $f_1, ..., f_d$, where $f_i : R \longrightarrow R$, $i = 1, ..., d$ continuous and $K \subset D(f_i)$ can be approximated in the sense of uniform topology on K by input-output mappings of k hidden layer networks without non parallel connections whose output functions for the hidden layers are $\phi(y)$ and for the input and output layers are linear.

This theorem is identical to the theorem B.5 except for two assumptions: We assume in the theorem 1.2 that $a = 1$, and that the function f is a sum of d continuous functions and therefore continuous itself. The proof is analogous to the proof of theorem B.5.

1.3.1 Levenberg-Marquardt method

In this section we show a so called Levenberg-Marquardt method for parameter estimation of a neural network without non-parallel weights (see figure 1.5). For a given input sequence x we can compute the network output as follows:

$$f_{nn}(x, \theta) = v_0 + \sum_{h=1}^{H} v_h \phi\big(\tilde{x}' \cdot w_h\big)$$

where $\theta = (w_0', ..., w_H', v_0, ..., v_H)'$ is the vector of network weights, and ϕ is the hidden unit activation function. For the special case of blocked neural networks we choose for ϕ, the hyperbolic tangens function, that

means,

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

For the part of θ with weights from input to hidden layer θ_{IH} we have,

$$\theta_{IH} = \begin{bmatrix} w_{11} & 0 & ... & ... & ... & 0 & w_{b1} \\ 0 & w_{22} & 0 & ... & ... & 0 & w_{b2} \\ 0 & 0 & w_{33} & 0 & ... & 0 & w_{b3} \\ ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & 0 & ... \\ 0 & ... & ... & ... & 0 & w_{dd} & w_{bd} \end{bmatrix}.$$

If we assume that $H = d$, it means that one neuron is enough for one component. For two neurons for each component we have the following matrix of weights:

$$\theta_{IH} = \begin{bmatrix} w_{11} & 0 & ... & ... & ... & 0 & w_{b1} \\ w_{21} & 0 & ... & ... & ... & 0 & w_{b2} \\ 0 & w_{32} & 0 & ... & ... & 0 & w_{b3} \\ 0 & w_{42} & 0 & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & 0 & ... \\ 0 & ... & ... & ... & 0 & w_{2d-1,d} & w_{b,2d-1} \\ 0 & ... & ... & ... & 0 & w_{2d,d} & w_{b,2d} \end{bmatrix},$$

ψ is the activation function from each hidden unit, or hidden neurons.

We want to minimize the sum of squared errors, i.e.

$$D(\theta) = \sum_{t=1}^{N} (y_t - f_{nn}(Y_\cdot, \theta)^2,$$

where $Y_\cdot$ is for the given process state the optimal delayed input sequence. We take a vector of weights $\theta^0 \in \Theta$ and move iteratively, with

linear steps of length h^i in direction v^i, where Θ is the compact finite set.

$$\bar{\theta}^i = \bar{\theta}^{i-1} + h^i v^i, \quad i = 1, 2, \ldots \ .$$

To provide certainty for convergence of $D(\theta^i)$ against at least a local minimum we must demand that the sum of squared error at $\bar{\theta}^{i+j}$ for any $j \in \mathcal{N}$ is smaller than at $\bar{\theta}^i$. That means,

$$D(\bar{\theta}^{i+j}) < D(\bar{\theta}^i), \ \ j = 1, \ldots \ . \tag{1.5}$$

One method to solve this problem is represented by the so called family of gradient descent algorithms. In these methods the direction vector $v^i = -R^i g^{i-1}$ where the negative gradient of function $D(\bar{\theta})$ at $\bar{\theta}^{i-1}$, $-g^{i-1}$ is multiplied with a matrix R^i to guarantee the convergence criterion (1.5). The vector v^i will be multiplied with a scalar h^i. Then we get in i-th iteration step computed vector of weights,

$$\bar{\theta}^i = \bar{\theta}^{i-1} - h^i R^i g(\bar{\theta}^{i-1}), \ \ i = 1, 2, \ldots \ .$$

The several gradient descent methods are different in the choice of the matrix R^i and in the choice of the scalar h^i.

Definition 1.5. We call an iteration step hv acceptable if

$$D(\bar{\theta} + hv) < D(\bar{\theta}).$$

Lemma 1.4. *A direction vector v is acceptable iff there exists a positive definite matrix R such that $v = -Rg$ with $g = \partial D(\bar{\theta})/\partial\bar{\theta}$.*

The Levenberg-Marquardt method is based on the Gauss-Newton method, in this method, the matrix $R^{-1} = 2J'J$ is chosen, where $J = (\partial f_{nn}(Y_\cdot, \bar{\theta})/\partial\bar{\theta})'$. The matrix R in the Gauss-Newton method could be singular, in this case the inverse of the matrix $(J'J)^{-1}$ is

nonexistent and the stability of the method is not guaranteed; therefore the direction vector is not acceptable.

Levenberg(1944) suggests the addition of a positive definite matrix $\lambda A'A$ to the matrix $J'J$. With this addition, the positive definity of the matrix $R^{-1} = 2(J'J + \lambda A'A)$ is guaranteed. The lemma 1.3 shows that the direction vector

$$v(\lambda, A) = (J'J + \lambda A'A)^{-1}J'u,$$

with $u(\bar{\theta}) = y_t - f_{nn}(Y_\cdot, \bar{\theta})$ is acceptable. In many applications the matrix A is chosen as a diagonal matrix, with entries $d_{ii} > 0$. Therefore the matrix $A'A$ is positive definite.

We must realize that the demand that the matrix $R = [2(J'J + \lambda A'A)]^{-1}$ always must be positive definite restricts the allowed area for the scalar λ.

For the matrix $\tilde{J} = JA^{-1}$ there exists an eigenvalue decomposition $\tilde{J} = P\Pi^{1/2}Q$, where $\Pi^{1/2}$ is a diagonal matrix with elements on the diagonal $\pi_j^{1/2}$, Q is an orthogonal $(p \times p)$ matrix and P an $(n \times p)$ matrix with the property that $P'P = I$. Then we can write an eigenvalue decomposition for the matrix $\tilde{J}'\tilde{J}$:

$$\tilde{J}'\tilde{J} = Q'\Pi^{1/2}P'P\Pi^{1/2}Q = Q'\Pi Q.$$

We know that Q is an orthogonal matrix. That means that $Q'Q = I$. Using this we get a scaled eigenvalue decomposition for the matrix $(J'J + \lambda A^2)$:

$$\begin{aligned}(J'J + \lambda A^2) &= A(\tilde{J}'\tilde{J} + \lambda I)A = AQ'(\Pi + \lambda I)QA \\ &= \sum_{j=1}^{p}(\pi_j + \lambda)z_j z_j',\end{aligned}$$

where z_j are the column vectors of the matrix $Z = QA$. The matrix $(J'J + \lambda A^2)$ and its inverse $(J'J + \lambda A^2)^{-1}$ are positive definite iff

$(J'J + \lambda A^2)$ has only positive eigenvalues $(\pi_j + \lambda) \quad j = 1, ..., p$, but this is the case iff

$$-\min\{\pi_1, ..., \pi_p\} < \lambda < \infty,$$

and we proved following theorem:

Theorem 1.5. *Let π_i be an eigenvalue of the matrix $A^{-1}J'JA^{-1}$. Then the direction vector $v(\lambda) = (J'J + \lambda A^2)^{-1}J'u$ of the Levenberg-Marquardt method is acceptable iff*

$$\lambda > -\min\{\pi_1, ..., \pi_p\}.$$

Lemma 1.6. *For a blocked neural network with H neurons in each block, the matrix $(J'J + \lambda A^2)^{-1}$ is always positive definite for $\lambda > 0$, where A is a diagonal matrix with $a_{jj} = a > 0, \quad \forall j = 1, ..., D$.*

Proof

For a d-dimensional time series we have the $(D \times D)$ matrix $J'J$ where $D = H(d^2 + 2 \cdot d) + 1$, with $d - 1 \times d$ zero columns and rows. For $d = 3$ and one neuron in each block we have following matrix,

$$J'J = \begin{bmatrix} w_{11} & 0...0 & w_{1,5} & 0...0 & w_{1,8} & w_{1,9} & ... & w_{1,16} \\ 0 & ... & ... & ... & ... & ... & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & ... & ... & ... & ... & 0 \\ w_{5,1} & 0...0 & w_{5,5} & 0...0 & w_{5,8} & w_{5,9} & ... & w_{5,16} \\ 0 & ... & ... & ... & ... & ... & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & ... & ... & ... & ... & 0 \\ w_{8,1} & 0...0 & w_{8,5} & 0...0 & w_{8,8} & w_{8,9} & ... & w_{8,16} \\ w_{9,1} & 0...0 & w_{9,5} & 0...0 & w_{9,8} & w_{9,9} & ... & w_{9,16} \\ ... & ... & ... & ... & ... & ... & ... & ... \\ w_{16,1} & 0...0 & w_{16,5} & 0...0 & w_{16,8} & w_{16,9} & ... & w_{16,16} \end{bmatrix}.$$

We know that any $D \times D$ matrix is nonsingular if and only if all its eigenvalues are nonzero. We note that the matrix $J'J$ is singular for any $d \in \mathcal{N}$. Therefore the matrix $A^{-1}J'JA^{-1}$ is singular too, where A is a diagonal matrix with positive elements and has the same form as $J'J$. If we choose a diagonal matrix A with $a_{ii} = a_{jj} = a, \ \forall i, j = 1, ..., D$ then we have $A^{-1}J'JA^{-1} = a \cdot J'J$ and for the eigenvalues

$$a \cdot eig(J'J) = eig(A^{-1}J'JA^{-1}).$$

We see that the rank of the matrix $rank(J'J) = 1$ then we know that there exists only one eigenvalue of the matrix $J'J$ with $eig(J'J) = \pi > 0$. All others are equal to zero. If we want that the matrix $J'J + \lambda A^2$ is positive definite, then we need that the eigenvalue

$$\begin{aligned} eig(J'J + \lambda A^2) &= eig(J'J + \lambda aI) \\ &= eig(J'J) + \lambda a = \frac{\pi}{a} + \lambda a, \end{aligned} \tag{1.6}$$

is positive. Furthermore we know that $\pi > 0$ then

$$-\min(0, ..., 0, \pi, 0, ..., 0) = 0,$$

and for $\lambda > 0$ we have proved this theorem. □

A further property of the Levenberg-Marquardt method lies in an invariance of the method to the scaling of the parameter space.

The invariance property says that we get the same results if we begin with the weight vector θ^0 and use the function $u = u(\theta) = y - f_{nn}(Y_\cdot, \theta)$ and if we start with $\tilde{\theta}^0 = A\theta$ by using the function $\tilde{u} = \tilde{u}(\theta) = y - f_{nn}(Y_\cdot, A^{-1}\theta)$.

Let $\tilde{\Theta} = A\Theta$ with the matrix A transformed parameter space and $\tilde{\theta} = A\theta$ the points of $\tilde{\Theta}$. Then for the negative gradient $-\tilde{g}$ of the space $\tilde{\Theta}$ we get

$$-\tilde{g} = -2\tilde{J}'\tilde{u} = -A^{-1}(2J'u) = A^{-1}(-g).$$

We have use following equations:

$$J(\tilde{\theta}) = \frac{\partial f_{nn}(Y_\cdot, A^{-1}\theta)}{\partial \theta} = \frac{\partial f_{nn}(Y_\cdot, \theta)}{\partial \theta} A^{-1} = JA^{-1} = \tilde{J},$$

and

$$\tilde{u} = y - f_{nn}(Y_\cdot, A^{-1}\theta) = y - f_{nn}(Y_\cdot, \theta) = u.$$

We get the direction vector of the not transformed parameter space $v(\lambda)$ when we choose in the transformed parameter space instead of A a unit matrix I and build an inverse projection:

$$\begin{aligned} v(\lambda) &= (J'J + \lambda A'A)^{-1} J'u \\ &= A^{-1}[2(\tilde{J}'\tilde{J} + I)]^{-1}\tilde{g} = A^{-1}\tilde{v}(\lambda). \end{aligned}$$

The last equation shows the invariance of the Levenberg-Marquardt method relating to parameter space scaling.

Chapter 2

Estimator for LDM

In this chapter we use the framework of non-linear stochastic regression without assuming any specific form for the regression function in its composite parts. Our objective is to introduce a neural network based method to determine an optimal subset of the stochastic regressors to fit the underlying regression model. In this chapter, we do not consider change points but only deal with lag structure and autoregressive functions not depending on time.

To justify this approach without too much technical details, we deal only with the case in which it is assumed that the regressors in each state to the corresponding component of multivariate time series have a compact support.

Suppose that $\{(y_t^i, Y_{t-L_t}); \quad t = 1, ..., N\}$ is a sequence from a d-dimensional time series with $y_t^i \in \mathcal{R}$, $L_t = (l^{i1}, ..., l^{id})'$ and $Y_{t-L_t} = (y^1_{t-l^{i1}}, ..., y^d_{t-l^{id}}) \in \mathcal{R}^d \quad (d \geq 1)$, where $l^{ij} \in \{1, ..., A\}, \quad A, l^{ij} \in \mathcal{N}, \quad \forall j = 1, ..., d$. Consider the regression model

$$y_t^i = E(y_t^i | Y_{t-L_t}) + \epsilon_t, \quad 1 \leq t \leq N, \tag{2.1}$$

where $\epsilon_t = y_t^i - E(y_t^i | Y_{t-L_t})$ and $L_t = (l^{i1}, ..., l^{id})'$. We define the true

regression function as

$$g(x) = E(y_t^i | Y_{t-L_t} = x).$$

Obviously $E(\epsilon_t | Y_{t-L_t}) = 0$. The goal of this chapter is to estimate a proper lag vector L_t which determines the regressors providing (almost) the whole information which is needed to model component y_t^i with the neural network based LD(d) model.

Assumption 2.1. *The hidden unit activation function ϕ is twice continuously partially differentiable at every point θ, $\theta \in \Theta_H$, where Θ_H is a compact subset of $\mathcal{R}$.*

Assumption 2.2. *The noise variables are independent normally distributed i.e. $\epsilon_t \sim N(0, \sigma^2)$.*

Assumption 2.3. *Y_{t-L_t} and ϵ_t are independent of each other, $(Y_{t-L_t}, \epsilon_t)'$ is strictly stationary and ergodic.*

2.1 LDM NN-based estimator

In this section we determine an estimator $\bar{L}_0^i = \left(\bar{l}^{i1}, \bar{l}^{i2}, ..., \bar{l}^{id}\right)'$ for the vector of time delay variables in a particular process state with length N. Assume that the state begins at time $t = 1$ and ends at time $t = N$, and in this interval there exist no change points with respect to any component.

First we choose a so called "Lag Horizon" $A \in \mathcal{N}$ that is a maximal time delay that is reasonable to take into consideration. This choice is based on the experience of the observer.

Finiteness and the fact that the lag horizon A and the dimension of time series d both are discrete variables allows us to find the global minimum of the error function by A^d times training of the neural

network as follows,

$$L_0 = arg \min_{L \in \Pi(A,d)} \left(\min_{\theta \in \Theta_H} \left(\frac{1}{N} \sum_{t=1}^{N} \left(y_t^i - f_{nn}(Y_{t-L}, \theta) \right)^2 \right) \right),$$

where Π is the set of all possible groupings of A element set on d places.

2.1.1 Algorithm for time lag estimation

This algorithm shows a fast way for finding locally optimal time lag variables in d-dimensional time series. We determine $\bar{L}_0^i$ by starting with $\bar{L}^i(0) = \left(\bar{l}^{i1}(0), ..., \bar{l}^{id}(0)\right)' = (a, ..., a)'$, for some $a \in \{1, ..., A\}$.

The basic idea is to keep all time lag variables of the d input variables fixed, except for one. We fit a neural network for each of the possible values $u = 1, ..., A$ for that time lag. The value of u, which gives the minimal mean squared error between the output of the corresponding neural network and the data, is taken as the new value for that time lag.

There are two additional loops. The loop with index j is responsible for going through all the dimensions of the d-dimensional input vector. The outer loop k is just for iterating the search.

1. $k = 1$. We estimate $L^i(k)$,

   - j=1,
     for $u = 1, ..., A$

$$\begin{aligned} \left(\bar{D}_k^{ij}(\cdot, \bar{\theta})\right)_u &= \min_{\theta \in \Theta_H} \left(\frac{1}{N} \sum_{t=1}^{N} (y_t^i - f_{nn}(\tilde{y}_{jk}, \theta))^2 \right), \\ \text{where } \tilde{y}_{jk} &= (y_{t-u}^1, y_{t-a}^2, ..., y_{t-a}^d)', \\ \bar{l}_{\cdot}^{ij}(k) &= arg \min_{u=1,...,A} \left(\left(\bar{D}_k^{ij}(\cdot, \bar{\theta})\right)_u \right) \end{aligned}$$

- j=2,
  for $u = 1, ..., A$

$$\begin{aligned}\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u &= \min_{\theta\in\Theta_H}\left(\frac{1}{N}\sum_{t=1}^{N}(y_t^i - f_{nn}(\tilde{y}_{jk},\theta))^2\right),\\ \text{where } \tilde{y}_{jk} &= (y^1_{t-\bar{l}^{i1}(k)}, y^2_{t-u}, y^3_{t-a}, ..., y^d_{t-a})',\\ \bar{l}^{ij}_{\cdot}(k) &= arg\min_{u=1,...,A}\left(\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u\right)\end{aligned}$$

- ...
- $j = d$,
  for $u = 1, ..., A$

$$\begin{aligned}\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u &= \min_{\theta\in\Theta}\left(\frac{1}{\Delta}\sum_{t=1}^{N}(y_t^i - f_{nn}(\tilde{y}_{jk},\theta))^2\right),\\ \text{where } \tilde{y}_{jk} &= (y^1_{t-\bar{l}^{i1}(k)}, ..., y^{d-1}_{t-\bar{l}^{i,d-1}(k)}, y^d_{t-u})'.\\ \bar{l}^{ij}_{\cdot}(k) &= arg\min_{u=1,...,A}\left(\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u\right)\end{aligned}$$

2. $k = 2$.

- $j = 1$,
  for $u = 1, ..., A$

$$\begin{aligned}\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u &= \min_{\theta\in\Theta_H}\left(\frac{1}{N}\sum_{t=1}^{N}(y_t^i - f_{nn}(\tilde{y}_{jk},\theta))^2\right),\\ \text{where } \tilde{y}_{jk} &= \left(y^1_{t-u}, y^2_{t-\bar{l}^{i2}(k-1)}, ..., y^d_{t-\bar{l}^{i,d}(k-1)}\right)'\\ \bar{l}^{ij}_{\cdot}(k) &= arg\min_{u=1,...,A}\left(\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u\right)\end{aligned}$$

- $j = 2$,

for $u = 1, ..., A$

$$\begin{aligned}
\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u &= \min_{\theta\in\Theta_H}\left(\frac{1}{N}\sum_{t=1}^{N}(y_t^i - f_{nn}(\tilde{y}_{jk},\theta))^2\right), \\
\text{where } \tilde{y}_{jk} &= \left(y^1_{t-\bar{l}^{i1}(k)}, y^2_{t-u}, y^3_{t-\bar{l}^{i3}(k-1)}, ..., y^d_{t-\bar{l}^{id}(k-1)}\right)', \\
\bar{l}^{ij}(k) &= arg\min_{u=1,...,A}\left(\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u\right)
\end{aligned}$$

- ...
- $j = d$,
  for $u = 1, ..., A$

$$\begin{aligned}
\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_u &= \min_{\theta\in\Theta_H}\left(\frac{1}{N}\sum_{t=1}^{N}(y_t^i - f_{nn}(\tilde{y}_{jk},\theta))^2\right), \\
\text{where } \tilde{y}_{jk} &= \left(y^1_{t-\bar{l}^{i1}(k)}, ..., y^{d-1}_{t-\bar{l}^{i,d-1}(k)}, y^d_{t-u}\right)', \\
\bar{l}^{ij}(k) &= arg\min_{u=1,...,A}\left(\left(\bar{D}_k^{ij}(m,\bar{\theta})\right)_u\right)
\end{aligned}$$

3. ...

If $\bar{l}^{ij}(k) = \bar{l}^{ij}(k-1)\ \ \forall j = 1, ..., d$, then the algorithm will be stopped and the optimal time delay variable vector will be set to $\bar{L}_0^i = \bar{L}^i(k)$. The diagram 2.1 shows the time lag estimator.

2.1.2 Convergence of the numerical algorithm

Theorem 2.4. *For any $i \in \{1, ..., d\}$, every $j = 1, ..., d$ and every $k = 1, ...$ we have,*

$$\liminf_{N\longrightarrow\infty}\left[\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\bar{l}^{ij}(k)} - \left(\bar{D}_k^{iJ}(\cdot,\bar{\theta})\right)_{\bar{l}^{iJ}(k)}\right] \leq 0,\ \ \forall J \leq j$$

and

$$\liminf_{\substack{N\longrightarrow\infty \\ k\longrightarrow\infty}}\left[\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\bar{l}^{ij}(k)} - \left(\bar{D}_K^{ij}(\cdot,\bar{\theta})\right)_{\bar{l}^{ij}(K)}\right] \leq 0\ \ \forall K \leq k,$$

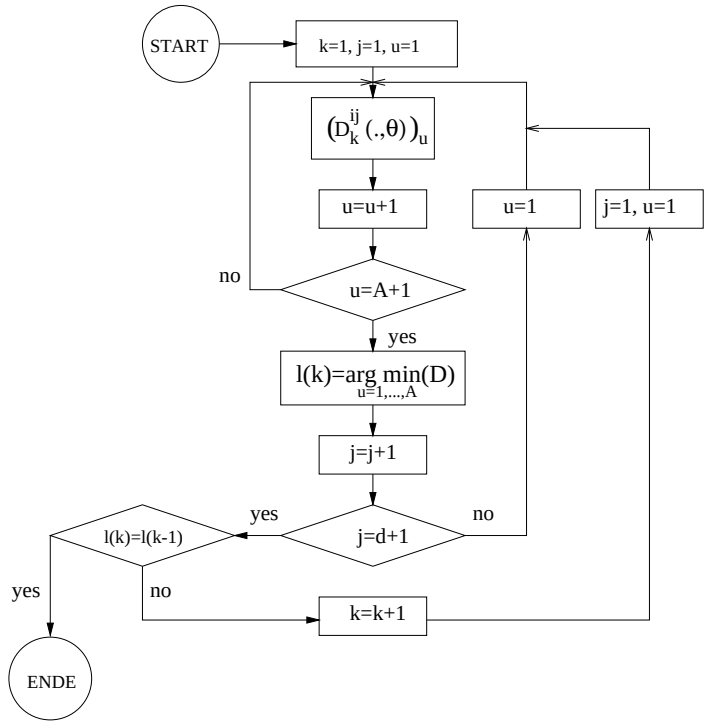

Figure 2.1: Time lag estimator.

define $\bar{L}^{ij}(0) = (a, ..., a)'$, $\bar{L}^{ij}(0) \in \mathcal{N}^d$, *and*

$$\tilde{L}_{\cdot}^{ij}(k) = \left(\bar{l}_{\cdot}^{i1}(k), \bar{l}_{\cdot}^{i2}(k), ..., \bar{l}_{\cdot}^{ij}(k), \bar{l}_{\cdot}^{i,j+1}(k-1), ..., \bar{l}_{\cdot}^{id}(k-1)\right)'$$

for any $k \in \mathcal{N}$, *iff* $\bar{D}$ *is estimated with blocked neural network.*

Proof

1. $k = 1$,
   - $j = 1$,

$$\begin{aligned}\left(\bar{D}_k^{i1}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i1}(k)} &= \rho\Big(y_t^i - f_{nn}\left(Y^{(1)},\bar{\theta}\right)\Big)\\ &= \rho\Big(y_t^i - f_{nn}\left(y^1_{t-\bar{l}^{i1}(k)},\bar{\theta}_1\right) -\\ &\quad -f_{nn}\left(y^2_{t-a},\bar{\theta}_2\right) - ... - f_{nn}\left(y^d_{t-a},\bar{\theta}_d\right)\Big),\end{aligned}$$

- $j = 2$,

$$\begin{aligned}\left(\bar{D}_k^{i2}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i2}(k)} &= \rho\Big(y_t^i - f_{nn}\left(Y^{(2)},\theta\right)\Big)\\ &= \rho\Big(y_t^i - f_{nn}\left(y^1_{t-\bar{l}^{i1}(k)},\bar{\theta}_1\right) -\\ &\quad -f_{nn}\left(y^2_{t-\bar{l}^{i2}(k)},\bar{\theta}_2\right) - f_{nn}\left(y^3_{t-a},\bar{\theta}_3\right) - ...\\ &\quad ... - f_{nn}\left(y^d_{t-a},\bar{\theta}_d\right)\Big)\end{aligned}$$

We know from the previous section that $\bar{l}^{i2}(k)$ is the optimal time delay value. Then if $a \neq \bar{l}^{i2}(k)$ we get,

$$\left(\bar{D}_k^{i,j+1}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i,j+1}(k)} < \left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k)} \tag{2.2}$$

and we get equality in (2.2) if $a = \bar{l}^{i2}(k)$. From 2.2 we obtain

$$\liminf_{N\longrightarrow\infty}\left[\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k)} - \left(\bar{D}_k^{iJ}(\cdot,\bar{\theta})\right)_{\tilde{L}^{iJ}(k)}\right] \leq 0,$$

analogously for every $j = 2, ..., d$ and $J < j$.

2. $k = 2$,

- $j = 1$,

$$\begin{aligned}\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i1}(k)} &= \rho\Big(y_t^i - f_{nn}\left(y^1_{t-\bar{l}^{i1}(k)},\cdot\right) -\\ &\quad - f_{nn}\left(y^2_{t-\bar{l}^{i2}(k-1)},\cdot\right) - ... -\\ &\quad - f_{nn}\left(y^d_{t-\bar{l}^{id}(k-1)},\cdot\right)\Big),\end{aligned}$$

- $j = 2$,

$$\begin{aligned}\left(\bar{D}_k^{i,j+1}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i2}(k)} &= \rho\Big(y_t^i - f_{nn}\left(y^1_{t-\bar{l}^{i1}(k)},\cdot\right) \\ &\quad - f_{nn}\left(y^2_{t-\bar{l}^{i2}(k)},\cdot\right) - f_{nn}\left(y^3_{t-\bar{l}^{i2}(k-1)},\cdot\right) - \\ &\quad - \ldots - f_{nn}\left(y^d_{t-\bar{l}^{id}(k-1)},\cdot\right)\Big),\end{aligned}$$

we know that

$$\begin{aligned}\bar{L}^{i,j+1}(k-1) &= arg \min_{u=1,\ldots,A}\left(\left(\bar{D}_{k-1}^{i,j+1}(\cdot,\bar{\theta})\right)_u\right) \\ \bar{L}^{i,j+1}(k) &= arg \min_{u=1,\ldots,A}\left(\left(\bar{D}_{k}^{i,j+1}(\cdot,\bar{\theta})\right)_u\right),\end{aligned}$$

therefore according to the definition of lag estimator,

$$\begin{aligned}&\left(\bar{D}_k^{i,j+1}(\cdot,\bar{\theta})\right)_{\tilde{L}^{i,j+1}(k)} \leq \left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k)} \\ \Longrightarrow \quad &\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k)} \leq \left(\bar{D}_{k-1}^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k-1)}. \qquad (2.3)\end{aligned}$$

Therefore from (2.3) we obtain

$$\liminf_{N\longrightarrow\infty}\left[\left(\bar{D}_k^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(k)} - \left(\bar{D}_K^{ij}(\cdot,\bar{\theta})\right)_{\tilde{L}^{ij}(K)}\right] \leq 0, \quad \forall K \leq k,$$

analogously for every $j = 2, ..., d$.

3. Analogously for $k = 3, ..., \infty$. □

2.1.3 Consistency of the lag estimator

In this section we will show consistency of the lag estimator described in section 2.1.1. Let $\bar{D}_N^i(L,\theta)$ denote the criterion function, and let $\hat{\theta}_{N,L}$ be a corresponding neural network estimator. Formally,

$$\hat{\theta}_{N,L} = arg \min_{\theta\in\Theta_H(c)}\left(\bar{D}_N^i(L,\theta)\right),$$

where

$$\bar{D}_N^i(L,\theta) = N^{-1}\sum_{t=1}^{N}\left(y_t^i - f_{nn}(Y_{t-L},\theta)\right)^2.$$

The classical consistency proof deduces the limiting behaviour of $\hat{\theta}_N$ from the limiting behavior of $\bar{D}_N$. Let

$$\bar{D}_0^i(L,\theta) = E\left(y_t^i - f_{nn}(Y_{t-L},\theta)\right)^2.$$

Following assumption is necessary to show the uniform convergence of the estimators.We assume that the error surface on that the neural network training algorithm moves, takes its unique global minimum on a compact subset of the parameter space for any vector of time delay variables $L \in \Pi(A,d)$.

Assumption 2.5. . *For $L \in \Pi(A,d)$, $\bar{D}_0^i(L,\theta)$ has a unique global minimum $\theta_{0,L}$, in the interior of a suitable compact subset $\Theta_H(c) = \{\theta; ||\theta - \theta_0|| \leq c\}$, where*

$$||\theta - \theta_0|| = \sum_{i=1}^{3Hd+1} |\theta_i - \theta_{0i}|,$$

of Θ_H:

$$\theta_{0,L} = arg\min_{\theta\in\Theta_H}\left(\bar{D}_0^i(L,\theta)\right) = arg\min_{\theta\in\Theta_H(c)}\left(\bar{D}_0^i(L,\theta)\right).$$

An alternative to the strategy of reducing the uniform convergence problem to a compact subset of the parameter space is to extend the neural network training procedure to a compactification of the parameter space. This way for showing the classical consistency proof of time lag and weight parameters in blocked neural network based LD models will not be considered here.

We showed in the previous section that the differences between the statistics $\bar{D}_N^i(L,\theta)$ and the nonstochastic functions $\bar{D}_0^i(L,\theta)$ converge to zero for all $i \in \{1,...,d\}$, as the sample size tends to infinity. Also the iteration number k of the algorithm for lag estimation tends to infinity.

Definition 2.1. For a given sequence of functions $\bar{D}_N : R^d \times \Theta_H \longrightarrow R$ the sequence of minimizers $\hat{\theta}_{N,L}$ of $\bar{D}_N(L,\theta)$ is called identifiably unique if for every $\epsilon > 0$,

$$\liminf_{N\longrightarrow\infty}\left[\inf_{\{\theta\in\Theta_H:||\theta-\hat{\theta}_{N,L}||\geq\epsilon\}}\left(\bar{D}_N(L,\theta)-\bar{D}_N(L,\hat{\theta}_{N,L})\right)\right]>0, \quad (2.4)$$

in probability for all $L \in \{1,...,A\}$, $A \in \mathcal{N}$.

For a special case of the lag estimators it is enough to prove that the sequence of neural network weight estimators is identifiably unique in the case that the input variables were optimally delayed. Then the lag estimator is identifiably unique too, as L can assume only finitely many values.

In the following, we fix the coordinate i, and we estimate the weights of the corresponding neural network by minimizing $\bar{D}_N^i(L,\theta)$. For sake of simplicity, we just write $\hat{\theta}_{N,L}$ for the estimated weights omitting the dependence on i.

First, we impose some conditions on the neural network used, which are common in neural network theory, compare, e.g., White(1989) or Franke and Neumann(2000).

Assumption 2.6. .

a) There is unique L_0 such that with $\theta_0 \equiv \theta_{0,L_0}$ the pair (L_0,θ_0) satisfies

$$(L_0,\theta_0) = arg \min_{\theta\in\Theta_H,\ L\in\Pi(A,d)}\left(\bar{D}_0^i(L,\theta)\right)$$

and for every $L \neq L_0$: $\bar{D}_0^i(L, \theta_{0,L}) > \bar{D}_0^i(L_0, \theta_0)$.

b) g is bounded and y_t *is a stationary time series which satisfies* β*-mixing condition with exponentially decaying mixing coefficients (compare, Douckan,1994 for the definition of* β*-mixing and its equivalence to geometric ergodicity).*

The mixing assumption is not unusual for nonlinear autoregressive processes. Franke et al.(2000) have given simple sufficient conditions on the autoregression function g such that the assumption is satisfied.

As the neural network function $f_{nn}(x, \theta)$ depends on θ in a very smooth and simple manner, assumption 2.5 is not a severe restriction but it excludes only a few degenerate constellations of the random process and the network function. $\Theta_H(c)$ can be chosen independent of L as we consider only finitely many values of L.

To use the Bernstain's inequality for a mixing time series in the uniform convergence proof, following inequality must be satisfied(Cramer's conditions): If there exists $c > 0$ then

$$E|\epsilon_t|^n \leq c^{n-1} n!\, E\epsilon_t^2 < +\infty,$$

for $n = 3, 4, ...$ We need a further important assumption.

Assumption 2.7. *The innovations* ϵ_t *are i.i.d. zero mean random variables, for which all moments* $E|\epsilon_t|^n$, $n = 1, 2, ..,$ *are finite.*

Theorem 2.8. *Under the assumptions 2.5, 2.6 and 2.7*

$$\sup_{\theta \in \Theta_H(c)} |\bar{D}_N^i(L, \theta) - \bar{D}_0^i(L, \theta)| \to 0$$

in probability for all $L = 1, ..., A$.

Proof

Using Bernstain's inequality for a mixing time series (Douckan,1994) in conjunction with a simple truncation argument (which is possible by assumption 2.7 we have, using assumptions 2.6 b) and 2.7

$$\sup_{\theta\in\Theta_H(c)} pr\left(|\bar{D}_N^i(L,\theta) - \bar{D}_0^i(L,\theta)| > \delta\right) \to 0, \tag{2.5}$$

for all $\delta > 0$, $L = 1, ..., A$(compare also Franke and Neumann(2000), for a similar argument).

Let $\gamma > 0$ be a small real number to be chosen later. Let θ_k, $k = 1, ..., m(\gamma)$, be a set of vectors in $\Theta_H(c)$ such that for all $\theta \in \Theta_H(c)$ there is $k \leq m(\gamma)$ such that $||\theta - \theta_k|| \leq \gamma$. Let

$$\Delta_N(\theta) = \bar{D}_N^i(L,\theta) - \bar{D}_0^i(L,\theta).$$

Then for arbitrary $\delta > 0$

$$pr\left(\sup_{\theta\in\Theta_H(c)} |\Delta_N(\theta)| > \delta\right) \leq pr\left(\sup_{\theta\in\Theta_H(c)} |\Delta_N(\theta)| > \delta,\right.$$
$$\left.\sup_{||\theta-\theta_k||\leq\gamma} |\Delta_N(\theta) - \Delta_N(\theta_k)| \leq \frac{\delta}{2} \;\forall\; k = 1, ..., m(\gamma)\right) +$$
$$+pr\left(\sup_{||\theta-\theta_k||\leq\gamma} |\Delta_N(\theta) - \Delta_N(\theta_k)| > \frac{\delta}{2},\;\; (*)k = 1, ..., m(\gamma)\right), \tag{2.6}$$

where (*) stands for "for at least one". The first term on the right hand side of (2.6) is bounded from above by

$$pr\left(\sup_{k\leq m(\gamma)} |\Delta_N(\theta_k)| > \frac{\delta}{2}\right) \leq \sum_{k=1}^{m(\gamma)} pr\left(|\Delta_N(\theta_k)| > \frac{\delta}{2}\right)$$
$$\leq m(\gamma) \sup_{\theta\in\Theta_H(c)} pr\left(|\Delta_N(\theta_k)| > \frac{\delta}{2}\right) \to 0 \;\text{ for }\; N \to \infty,$$

in probability by (2.5).

For the second term on the right hand side of (2.6), we have

$$|\Delta_N(\theta) - \Delta_N(\theta_k)| \leq$$
$$\left| \frac{1}{N} \sum_{t=1}^{N} \{(y_t - f_{nn}(Y_{t-L}, \theta))^2 - (y_t - f_{nn}(Y_{t-L}, \theta_k))^2\} \right| +$$
$$\left| E(y_t - f_{nn}(Y_{t-L}, \theta))^2 - E(y_t - f_{nn}(Y_{t-L}, \theta_k))^2 \right|.$$

Using the particular form of the network function f_{nn}, we have that the derivative of f_{nn} w.r.t. one of the parameters v_μ is bounded as the activation function $\phi(.)$ is bounded, and the derivative w.r.t. one of the parameters $w_{\nu\mu}$ is bounded by a constant terms $|v_\mu y^\nu_{t-l_{\nu\mu}}|$. $|v_\mu|$ is also bounded as we consider only θ in the compact set $\Theta_H(c)$. Using the mean-value theorem, we get

$$\left| E(y_t - f_{nn}(Y_{t-L}, \theta))^2 - E(y_t - f_{nn}(Y_{t-L}, \theta_k))^2 \right| \leq b||\theta - \theta_k||,$$

for a suitable constant b, and, analogously,

$$\left| \frac{1}{N} \sum_{t=1}^{N} \{(y_t - f_{nn}(Y_{t-L}, \theta))^2 - (y_t - f_{nn}(Y_{t-L}, \theta_k))^2\} \right| \leq B_N||\theta - \theta_k||,$$

where B_N is a nonnegative random variable with $B_N \to b$ in probability for $N \to \infty$ by a low of large numbers for mixing time series. Therefore, the second term on the right hand side of (2.6) is bounded by

$$pr\left(\sup_{||\theta - \theta_k|| \leq \gamma} \left((B_N + b)||\theta - \theta_k|| > \frac{\delta}{2} \;\; (*)k = 1, ..., m(\gamma) \right) \right) \leq$$
$$pr\left(\gamma(B_N + b) > \frac{\delta}{2} \right) = pr\left(B_N - b > \frac{\delta}{2\gamma} - 2b \right) \to 0,$$

(*) stands for "for at least one". If we choose γ small enough such that $\frac{\delta}{2\gamma} - 2b > 0$. □

From theorem 2.6 and assumption 2.5, it follows immediately that $\hat{\theta}_{N,L}$ is an identifiably unique sequence of minimizers of $\bar{D}_N^i(L,\theta)$ for given L.

Consistency of the least squared estimator $\hat{\theta}_{N,L}$ for $\theta_{0,L}$ can now be inferred from the following lemma applied to the compact subset $\Theta_H(c)$:

Lemma 2.9. *Let D_N, $\bar{D}_N : R^d \times \Theta_H(c) \longrightarrow R$ be two arbitrary sequences of functions such that a.s.*

$$\sup_{\Theta_H(c)} |D_N(L,\theta) - \bar{D}_N(L,\theta)| \longrightarrow 0 \quad as \quad N \longrightarrow \infty.$$

Let $\hat{\theta}_{N,L}$ be an identifiably unique sequence of minimizers of $\bar{D}_N(L,\theta)$. Then for any sequence $\bar{\theta}_{N,L}$ such that

$$D_N(L,\bar{\theta}_{N,L}) = \inf_{\Theta_H(c)} D_N(L,\theta),$$

holds, we have $||\hat{\theta}_{N,L} - \bar{\theta}_{N,L}|| \to 0$ a.s. in probability as $N \to \infty$.

The proof of this lemma is in Pötscher/Prucha[15].

We note that the identifiable uniqueness condition over the compact subset $\Theta_H(c)$ becomes in difference with definition 2.1

$$\inf_{\epsilon \le ||\hat{\theta}_{N,L} - \theta_{0,L}|| \le c} E\left[f_{nn}(Y_{t-L_0}, \hat{\theta}_{N,L}) - f_{nn}(Y_{t-L_0}, \theta_{0,L})\right]^2 > 0.$$

Next theorem shows the consistency of the time lag estimator for the case where the cyclicity in the time lag variables is excluded. This exclusion is showed in the next assumption,

Assumption 2.10.

$$pr\left(g(y^1_{t-l^{i1}}, ..., y^d_{t-l^{id}}) \neq g(y^1_{t-\lambda^{i1}}, ..., y^d_{t-\lambda^{id}})\right) > 0,$$

for all $(l^{i1}, ..., l^{id}) \neq (\lambda^{i1}, ..., \lambda^{id})$, where $g(x) = E(y^i_t | Y_{t-L} = x)$.

We remark that assumption 2.10 is automatically satisfied if the innovations ϵ_t have a density, which is everywhere positive.

Theorem 2.11. *Let us consider a component* y^i, $i \in \{1, ..., d\}$ *for any fixed* $L \in \mathcal{N}^d$. *Then we have, for*

$$\hat{\theta}_{N,L} = arg \min_{\theta \in \Theta_{H(c)}} \left(\bar{D}_N^i(L, \theta) \right),$$

where $\bar{D}_N^i(L, \theta) = N^{-1} \sum_{t=1}^{N} \left(y_t^i - f_{nn}(Y_{t-L}, \theta) \right)^2$ *and for*

$$\theta_{0,L} = arg \min_{\theta \in \Theta_{H(c)}} \left(\bar{D}_0^i(L, \theta) \right),$$

where $\bar{D}_0^i(L, \theta) = E \left(y_t^i - f_{nn}(Y_{t-L}, \theta) \right)^2$, *that*

$$\hat{L} \overset{N \to \infty}{\longrightarrow} L_0, \quad i.p.$$

where

$$\hat{L} = arg \min_{L} \left(\bar{D}_N^i(L, \hat{\theta}_{N,L}) \right), \quad and \quad L_0 = arg \min_{L} \left(\bar{D}_0^i(L, \theta_{0,L}) \right).$$

Proof

For a fixed L we know that during the training of the neural network the weight matrix converges to the true weight matrix under consideration of a chosen L. Using lemma 2.7 we write,

$$\hat{\theta}_{N,L} \overset{N \to \infty}{\longrightarrow} \text{i.p.} \; \theta_{0,L}$$

$$\begin{aligned} \theta_{0,L} &= arg \min_{\theta \in \Theta_{H(c)}} \left[E \left(y_t^i - f_{nn}(Y_{t-L}, \theta) \right)^2 \right] = \\ &= arg \min_{\theta \in \Theta_{H(c)}} \left[E \left(g(Y_{t-L_0}) - f_{nn}(Y_{t-L}, \theta) \right)^2 + \sigma_\epsilon^2 \right], \end{aligned}$$

and therefore we know that for continuous functions $\bar{D}^i$,

$$\bar{D}_N^i(L, \hat{\theta}_{N,L}) \overset{N \to \infty}{\longrightarrow} \bar{D}_0^i(L, \theta_{0,L}), \quad \text{i.p.}$$

For L finite,

$$\hat{L} = arg\min_{L}\left(\bar{D}_N^i(L,\hat{\theta}_{N,L})\right) \overset{N\to\infty}{\longrightarrow} arg\min_{L}\bar{D}_0^i(L,\theta_{0,L}) = L_0, \text{ i.p.} \quad \square$$

Example

We consider the following simple LD model. For y_t and ϵ_t independent, we write:

$$y_t = g(y_{t-l_0}) + \epsilon_t,$$

where $y_t \in \mathcal{R}$, g known, nonconstant. We estimate l by

$$\hat{l} = arg\min_{l\geq 1}\left(\frac{1}{N}\sum_{t=1}^{N}\left(y_t - g(y_{t-l})\right)^2\right).$$

We want to show that

$$pr\left(\hat{l} = l_0\right) \overset{N\to\infty}{\longrightarrow} 1.$$

The minimal condition is that there exist two points y_1 and y_2 such that $g(y) \neq g(z)$ and $p_y(y), p_y(z) > 0$ for all $y \in [y_1 - \delta, y_1 + \delta]$, $z \in [y_2 - \delta, y_2 + \delta]$ where p_y is a stationary density of y_t.

$$\begin{aligned}
D(l) &= \frac{1}{N}\sum_{t=1}^{N}\left(y_t - g(y_{t-l})\right)^2 = \\
(1) &= \frac{1}{N}\sum_{t=1}^{N}\left(y_t - g(y_{t-l_0})\right)^2 + \\
(2) &\quad + \frac{2}{N}\sum_{t=1}^{N}\left(y_t - g(y_{t-l_0})\right)\left(g(y_{t-l_0}) - g(y_{t-l})\right) \\
(3) &\quad + \frac{1}{N}\sum_{t=1}^{N}\left(g(y_{t-l_0}) - g(y_{t-l})\right)^2,
\end{aligned}$$

for $N \to \infty$

$$\begin{aligned}
(1) &\to E(\epsilon_t^2) = \sigma_\epsilon^2 \\
(2) &\to 2E(\epsilon_t)E\left(g(y_{t-l_0}) - g(y_{t-l})\right) = 0.
\end{aligned}$$

Therefore, for $N \to \infty$

$$D(l) \sim \sigma_\epsilon^2 + \frac{1}{N}\sum_{t=1}^{N}\big(g(y_{t-l_0}) - g(y_{t-l})\big)^2 = D_g(l) \geq 0,$$

and for $l = l_0$ we have the absolute minimum $D(l_0) \sim \sigma_\epsilon^2$, for $N \to \infty$.

$$D(l) = (1) + (2) + (3) \stackrel{LLN}{\longrightarrow} E(\epsilon_t^2) + E\big(g(y_{t-l_0}) - g(y_{t-l})\big)^2 \equiv D_0(l),$$

LLN stands for the law of large numbers for stationary processes. $D_0(l) > \sigma_\epsilon^2$ for $l \neq l_0$, if $g(y_1) \neq g(y_2)$ and $pr(y_{t-l_0} \approx y_1,\ y_{t-l} \approx y_2) > 0$ by assumption.

We want to show that, $D_0(l) > \sigma_\epsilon^2 + \Delta$ for $\Delta > 0$ for all $l \neq l_0$. we distinguish between two cases,

- $l = 1, ..., A$, then it follows automatically from $D_0(l) > \sigma_\epsilon^2$ for all $l \neq l_0,\ \ l \leq A$ that,

$$\Delta < \min_{l \neq l_0,\ l=1,...,A} D_0(l) - \sigma_\epsilon^2.$$

- $l \to \infty$ ($A \to \infty$ from the first case), we assume that y_t is ρ-mixing, that means that $cov(U, V) \leq \rho_n, \quad \rho_n \to 0, \quad n \to \infty$, for all U that is measurable with respect to the sigma algebra $\sigma(..., y_{t-1}, y_t)$ and V that is measurable with respect to the sigma algebra $\sigma(y_{t+n}, y_{t+n+1}, ...)$. From the ρ-mixing property it follows,

$$D_0(l) \quad = \quad \sigma_\epsilon^2 + E\big(g(y_{t-l_0}) - g(y_{t-l})\big)^2,$$

use that $E(g(y_{t-l_0})) = E(g(y_{t-l}))$ then,

$$\begin{aligned} D_0(l) &= \sigma_\epsilon^2 + var(g(y_{t-l_0}) - g(y_{t-l})) \\ &= \sigma_\epsilon^2 + 2var(g(y_t)) - 2cov(g(y_{t-l_0}), g(y_{t-l})) \\ &\geq \sigma_\epsilon^2 + 2var(g(y_t)) - 2\rho_{l-l_0}, \end{aligned}$$

we know that $\rho_{l-l_0} \to 0$ for $l \to \infty$, therefore,

$$D_0(l) \geq \sigma_\epsilon^2 + \Delta^*.$$

It holds for all $l = 1, 2, ...$:

$$\min_{l \neq l_0} D_0(l) > D_0(l_0) = \sigma_\epsilon^2.$$

We know that $D(l) \xrightarrow{P} D_0(l)$ because of finiteness of $l = 1, ..., A$ it follows the regularity:

$$pr\Big(|D(l) - D_0(l)| \leq \rho, \ \ \forall l = 1, ..., A\Big) \longrightarrow 1, \quad \forall \rho > 0.$$

Let be $\Delta = \min_{l \neq l_0} D_0(l) > \sigma_\epsilon^2$, $\hat{l} = \min_l D(l)$, then

$$\begin{aligned} pr(\hat{l} \neq l_0) &= pr(D(l) < D(l_0) \ \text{ for one } \ l \neq l_0) \\ &= 1 - pr(D(l) > D(l_0) \ \text{ for all } \ l \neq l_0) \\ &= 1 - pr\Big(|D(l) - D_0(l)| \leq \rho, \ \text{ for all } \ l\Big) + o_p(1), \end{aligned}$$

because

$$D(l) \geq D_0(l) - \rho \geq \Delta - \rho > D_0(l_0) + \rho \geq D(l_0),$$

for any ρ with $2\rho < \Delta - D_0(l_0)$. Then we have,

$$pr(\hat{l} \neq l_0) \longrightarrow 0, \ \ i.p.$$

A special case. We consider a neural network from figure 2.2. The network function has, then, the form:

$$f_{nn}(Y, \theta) = v_0 + \sum_{j=1}^{d} v_j \phi(w_{0j} + w_{jj} y^j),$$

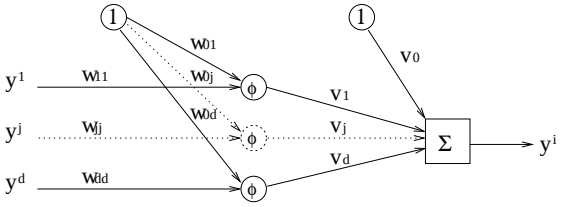

Figure 2.2: Neural network with one neuron in each block.

where $y^j$is the j-th coordinate of Y. If ϕ is differentiable, as we always assume, the derivatives w.r.t. y^j and w.r.t. the parameter w_{jj} are,

$$\begin{aligned}\frac{\partial f_{nn}(Y,\theta)}{\partial y^j} &= v_j\phi'(w_{0j}+w_{jj}y^j)w_{jj}\\ \frac{\partial f_{nn}(Y,\theta)}{\partial w_{jj}} &= v_j\phi'(w_{0j}+w_{jj}y^j)y^j.\end{aligned}$$

Using these identities we get,

Theorem 2.12. *For a blocked neural network with one neuron in each block, we consider*

$$\begin{aligned}C_j &= E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial w_{jj}}\right|,\\ R_{j,1} &= E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial y^j}\right|,\\ R^2_{j,2} &= E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial y^j}\right|^2.\end{aligned}$$

a) If the j-th coordinate of our random process y_t is bounded with essential supremum $||y_t^j||_\infty$, then

$$|w_{jj}|C_j \le ||y_t^j||_\infty R_{j1}$$

b) In general, we have

$$|w_{jj}|C_j \le \sqrt{E(y_t^j)^2}R_{j2}.$$

Proof

a)

$$\begin{aligned} C_j &= E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial w_{jj}}\right| \\ &= |v_j| E\big(|\phi'(w_{0j}+w_{jj}y^j_{t-L})|\cdot|y^j_{t-L}|\big) \\ &\leq |v_j|\cdot||y^j_{t-L}||_\infty E\big(|\phi'(w_{0j}+w_{jj}y^j_{t-L})|\big) \\ &= \frac{||y^j_{t-L}||_\infty}{|w_{jj}|} E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial y^j}\right| \\ &= \frac{||y^j_{t-L}||_\infty}{|w_{jj}|} R_{j,1} \end{aligned}$$

b)

$$\begin{aligned} C_j^2 &= \left(E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial w_{jj}}\right|\right)^2 \\ &= v_j^2\left(E\left(|\phi'(w_{0j}+w_{jj}y^j_{t-L})|\cdot|y^j_{t-L}|\right)\right)^2 \\ &\leq v_j^2 E|\phi'(w_{0j}+w_{jj}y^j_{t-L})|^2 E|y^j_{t-L}|^2 \\ &= \frac{E|y^j_{t-L}|^2}{w_{jj}^2} E\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial y^j}\right|^2 \\ &= \frac{E|y^j_{t-L}|^2}{w_{jj}^2} R_{j,2}^2, \end{aligned}$$

using the Bunjakowski-Schwarz inequality. □

The theorem tells us that $R_{j,1}$ and $R_{j,2}$ may be interpreted as relevance measures of the j-th coordinate of the random process related, of course, to the i-th coordinate which is described by the network under consideration. If e.g., $R_{j,1}$ is smaller than either $|w_{jj}|$ or the mean of $\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial w_{jj}}\right|$ or both are small, it means that the j-th neuron in the network describing the dependency of y_t^i on the past

observations is negligible. That part of the network function is scarcely reacting to changes in the j-th input coordinate y^j_{t-L}.(Compare also the formula for $\left|\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial w_{jj}}\right|$ given above).

For sake of reference, we state this qualitative property which is also applied in other parts of neural network theory, as a lemma.

Lemma 2.13. *Under the conditions of Theorem 2.10, the j-th lagged variable y^j_{t-L} has little influence on the i-th coordinate y^i_t if the derivative $\frac{\partial f_{nn}(Y_{t-L},\theta)}{\partial y^j}$ is small on the average which is measured by either $R_{j,1}$ or $R_{j,2}$.*

Remark 1. *In the case of ϕ being the identity, the neural network function f_{nn} reduces to*

$$f_{nn}(Y,\theta) = v_0 + \sum_{j=1}^{d} v_j w_{0j} + \sum_{j=1}^{d} v_j w_{jj} y^j.$$

In that case, $R_{j,1}$ becomes $|v_j \cdot w_{jj}|$. So $R_{j,1}$ is the coefficient of the input variable y^j. Obviously, the influence of y^j for the output is small, whenever $R_{j,1}$ is small and vice versa.

This explains the term "relevance measure" for $R_{j,1}$.

Chapter 3

Change Points in LDM

The goal of this chapter is to assess the applicability of a variety of nonlinear techniques for prediction, change point detection and state recognition, however the main conclusion is that the most important ingredient for good performance is knowing the data and a modeling technique that is able to represent the data. Furthermore we present the auxiliary methods needed to improve the state recognition performance in LD models, and in many cases to make state recognition possible at all.

3.1 Prediction inside of the current state

In this section we present an algorithm for single component prediction from a multivariate time series by its own past and the past of other components by LD models.

The following algorithm describes a procedure of a p-step prediction of component y^i at time t.

Assumption 3.1.

$\xi^{ij}_{k(j)+1} \neq t+l, \;\; l=1,...,p, \;\; \forall i,j \in \{1,...,d\}, \;\; k(j) \in \{1,...,q^{ij}\},$

where $\xi^{ij}_{k(j)+1}$ *is a change point.*

We consider one special case,

- $p \leq \min\{\bar{l}^{i1}_{k(1)},...,\bar{l}^{id}_{k(d)}\}$ then the p-step predictor of a component y^i is computed as follows:

$$\hat{y}^i_{t+p} = f_{nn}(Y_p, \bar{\theta}), \tag{3.1}$$

where

$$Y_p = (y^1_{t+p-\bar{l}^{i1}_{k(1)}},...,y^d_{t+p-\bar{l}^{id}_{k(d)}}),$$

and $\bar{\theta}$ is with the neural network estimated weight matrix.

In general we have the following universal p step predictor. Let be $p \in \mathcal{N}$ then the p-step predictor of a component y^i is computed as follows:

$$\hat{y}^i_{t+p} = f_{nn}(\tilde{Y}_p, \bar{\theta}), \tag{3.2}$$

where

$$\tilde{Y}_p = (\tilde{y}^1_{t+p-\bar{l}^{i1}_{k(1)}},...,\tilde{y}^d_{t+p-\bar{l}^{id}_{k(d)}}),$$

with

$$\tilde{y}^j_{t+p-\bar{l}^{ij}_{k(j)}} = \begin{cases} y^j_{t+p-\bar{l}^{ij}_{k(j)}} & : \;\; p \leq \bar{l}^{ij}_{k(j)} \\ \hat{y}^j_{t+p-\bar{l}^{ij}_{k(j)}} & : \;\; p > \bar{l}^{ij}_{k(j)} \end{cases}, \;\; j=1,...,d.$$

The prediction error (PE) is computed as follows:

$$PE^i_p = |y^i_{t+p} - \hat{y}^i_{t+p}|. \tag{3.3}$$

3.2 Component state recognition

In this section we present a useful algorithm for component state recognition in LD models.

We consider LD(d) model, for any component y^i, $i \in \{1, ..., d\}$ in interval $[1, N]$

$$\begin{aligned} y_t^i &= m_{k(1)}^{i1}\left(\left(1 - I_{[\xi_{k(1)}^{i1}, \xi_{k(1)}^{i1} + b_{k(1)}^{i1}]}(t)\right)(y_{t-l_{k(1)}^{i1}}^1)\right) + \\ &+ m_{k(2)}^{i2}\left(\left(1 - I_{[\xi_{k(2)}^{i2}, \xi_{k(2)}^{i2} + b_{k(2)}^{i2}]}(t)\right)(y_{t-l_{k(2)}^{i2}}^2)\right) + \\ &+ ... + \\ &+ m_{k(d)}^{id}\left(\left(1 - I_{[\xi_{k(d)}^{id}, \xi_{k(d)}^{id} + b_{k(d)}^{id}]}(t)\right)(y_{t-l_{k(d)}^{id}}^d)\right) + \epsilon_t, \end{aligned}$$

where $\xi_{k(j)}^{ij} < t \leq \xi_{k(j)+1}^{ij} \quad \forall i, j = 1, ..., d, \quad k(j) \in \{0, ..., q^{ij}\}$, ϵ_t i.i. $N(0, \sigma^2)$ distributed with finite variance, $\xi_0^{ij} = A, \quad \forall j = 1, ..., d$ and $b_{k(j)}^{ij} = l_{k(j)+1}^{ij} - l_{k(j)}^{ij}$. The integers $\xi_{k(j)}^{ij}$ for $k(j) \in \{1, ..., q^{ij}\}$ are change points of component y^i with respect to component y^j at time $k(j)$.

Assume that we observe a particular component at time t, in state c that begins at time $t = 0$. In each iteration step, the algorithm requests to see if the component y^i is still in the same state, i.e. if $t + 1 \neq \xi_q^i$? (see diagram 3.1). If $t + 1 = \xi_q^i$ that means that the new observation increases the estimated mean squared error dramatically, i.e. estimated neural network cannot represent the further evolution of the component. Formally let

$$\hat{D}_{t+1}^i(L, \theta) = \min_{\theta \in \Theta_H}\left(N^{-1}\sum_{k=0}^{t+1}(y_k^i - f_{nn}(Y_{k-L}, \theta))^2\right).$$

Let $\hat{L}_c$ be the with the LD model estimated time lag matrix for current state c. In the case where $\hat{D}_{t+1}^i(\hat{L}_c, \theta) > \hat{D}_{t+1}^i(L, \theta)$ for any L we have

a change point and the algorithm begins to detect the character of this change point. In step II of the algorithm we make an identification of a previous state for further analysis.

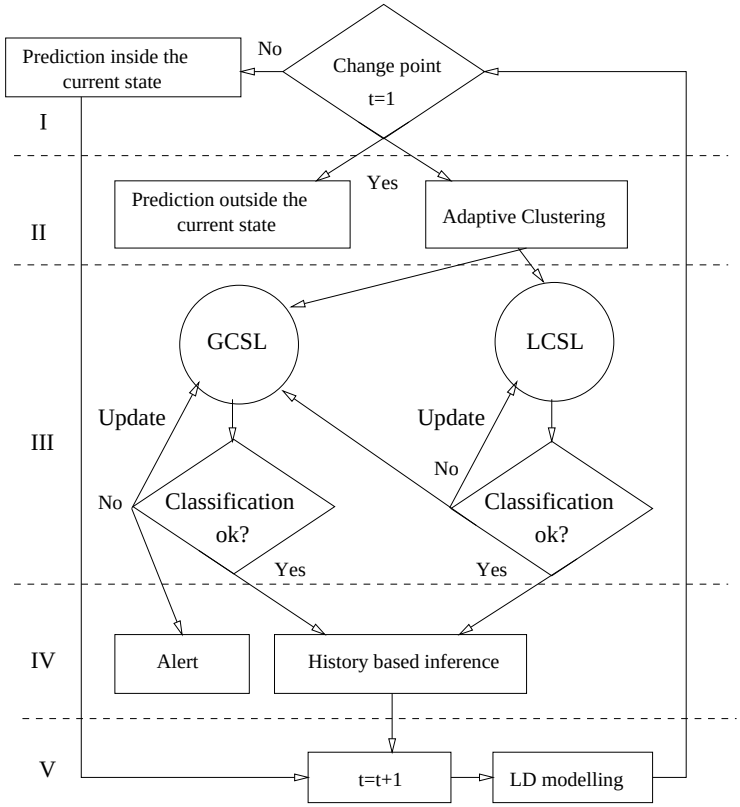

Figure 3.1: State recognition in LD models.

Each state or attractor can be envisioned as a k-dimensional vector representing a pattern, i. e. an ordered set of real numbers encoding some information about a certain object. If we make our

analysis with component states they have the form (L_c, R_c) where L_c is the $d \times d$ matrix of a time lags in particular state c, and R_c is the $d \times d$ matrix of relevance values in state c, then we have an attractor for any interested component in state c with dimension $2d$. The term of relevance will be discussed later, at this place we will mean under relevance of component j from a set of components to component i its contribution to the variance of component i.

In the next section we will see how to make a dimension reduction in attractor spaces. For example figure 3.2 shows a two dimensional attractor space. It is reasonable to assume that all the vectors within the basins of attraction (see attractor areas in figure 3.2) represent the same letters, respectively, but with possible distortions. Then any estimated component state with a state vector (L_c, R_c) located within the basin of attraction will be assigned to the corresponding attractor.

Three important elements of the information processing should be emphasized in this dynamic procedure. First, each attractor plays the role of a memory storage container for a prescribed pattern. Second, two different attractors perform a pattern classification, and each pattern is put at the corresponding place. Third, a process of attraction can be envisioned as a generalization procedure: all insignificant features of an initially distorted pattern are suppressed, and therefore an attractor becomes a typical representative of a class.

In the third stage of the algorithm in 3.1 we try to classify the new state in one of two component state librarys. We have two librarys, the local component state library(LCSL) and Global Component State Library(GCSL). In the LCSL we have patterns from the same component past states. In the GCSL we have patterns from analog components from other time series. For example, we observe diastolic blood pressure in a 70 year old patient with heart attack, Then in the diastolic blood pressure GCSL we have all entries from all

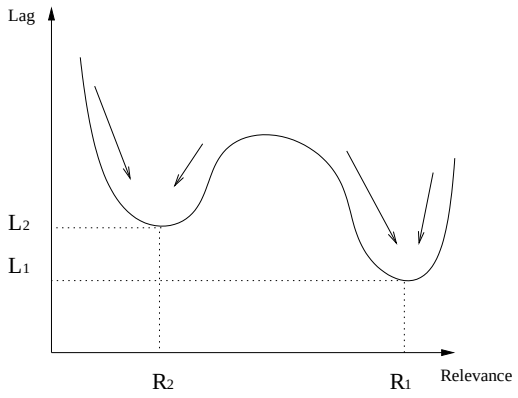

Figure 3.2: Two dimensional attractor space.

patients with analogous clinical picture and the same age.

The component state library could have, in general, entries from table 3.1, Here $\theta_c^i \in \mathcal{R}^{2m+1} \quad \forall i \in \{1, ..., d\}$, where m is the number of

State(Class)	1	. . .	n
Time delay	L_1	. . .	L_n
Relevance	R_1	. . .	R_n
Past	Previous State	. . .	Previous State
Future	Next State	. . .	Next State
Density	D_1	. . .	D_n
State Duration	t_1	. . .	t_n
Estimated parameters	$\{\theta_1^1, ..., \theta_1^d\}$	. . .	$\{\theta_n^1, ..., \theta_n^d\}$

Table 3.1: Entries in component state library.

neurons in each block of the blocked neural network, and c is the class

or state identification number.

In the fourth stage of the algorithm we have to make history based inference. Here exist two possibilities. In the first case, the actual state of the component is unknown for any component state library and we cannot make any statement about the process state. This situation is dangerous and must be reported to the observer in the form of, for example an alert. In the second case we can make several investigations to give insight into the future of this component and future of the process in general.

The fifth stage of the algorithm is the iteration and neural network learning, or in other words neural network estimation procedures.

To determine the component state librarys we must solve a well defined problem of classification. Of particular interest is to characterize the nearest neighbor regions bounded by hyperplanes for a finite set of random points in a metric space, to ensure that the regions are disjunct in each dimension. Such regions variously called Voronoy polytopes, or Dirichlet cells, can be computed by well established and highly optimized algorithms. For a set of randomly placed points in the $\mathcal{R}^{2d}$ dimensional space, the corresponding Voronoi polytopes will be normally irregular. The diameters of these polytopes will be, therefore, different along each direction. Voronoy diagrams and the method of "adaptive clustering" for computation of state cluster centers and boundaries are presented in the next section.

To illustrate the location of eleven state(class) centers in a space, we define component state as a typel $S_c = (R_c, L_c)\ \ c = 1, ...11$, where R_c is the relevance matrix with,

$$R_c = \begin{bmatrix} r_c^{11} & r_c^{12} & ... & r_c^{1d} \\ r_c^{21} & r_c^{22} & ... & r_c^{2d} \\ ... & ... & ... & ... \\ r_c^{d1} & r_c^{d2} & ... & r_c^{dd} \end{bmatrix}, \quad c = 1, ..., n, \tag{3.4}$$

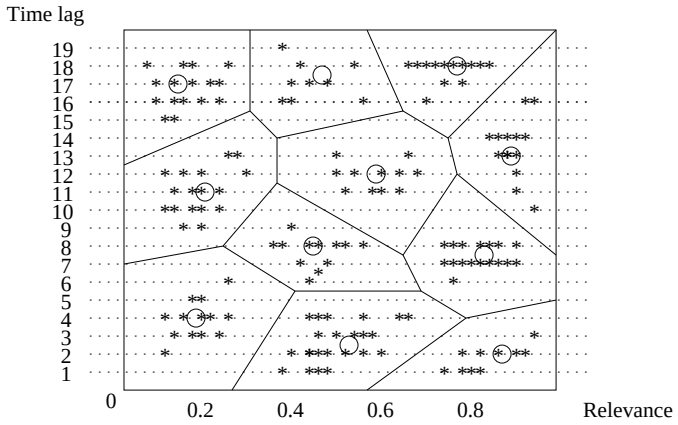

Figure 3.3: Voronoi diagram of 11 cluster centers between two components. ∘–cluster centers, ∗–relevance, time lag pair.

here r_c^{ij} is the estimated relevance of component y^j to component y^i in state c(how to compute the relevance values will be discussed in section 3.6), and L_c is the matrix of estimated time lag values with,

$$L_c = \begin{bmatrix} l_c^{11} & l_c^{12} & \dots & l_c^{1d} \\ l_c^{21} & l_c^{22} & \dots & l_c^{2d} \\ \dots & \dots & \dots & \dots \\ l_c^{d1} & l_c^{d2} & \dots & l_c^{dd} \end{bmatrix}, \quad c = 1, \dots, n, \tag{3.5}$$

where l_c^{ij} is the estimated time lag value between components y^j and y^i in state c.

To illustrate the Voronoi diagram clusters for any component pair (y^i, y^j) in any state c, we plot r_c^{ij} against l_c^{ij} and obtain one point on the Voronoi diagram.

The figure 3.3 shows the location of classes in a Voronoi diagram

between two components.

To make the classification procedure faster we prune the insignificant parameters of the time delay and relevance coefficient matrices. We distinguish two kinds of neural network pruning.

- First kind. For simple neural network reduction it is enough to choose a pruning coefficient η. That means that if the relevance value of the component $y^j, \quad j \in \{1, ..., d\}$ with respect to the component y^i in state c is less than η, the coefficients will be set to zero, i.e. $r_c^{ij} = 0$, and the time delay coefficients l_c^{ij} will not be defined. This kind of decision space dimension reduction is allowed by theorem 1.14 and lemma 1.15.

- Second kind. Usual neural network pruning algorithms.(OBS, OBD, a.o.) One universal method of neural network pruning is presented in section B.4.

Pruning procedures allows us to reduce the dimension of the decision space in each pruning step by the factor two. Note that the decision space dimension is an even natural number.

To make a quick classification (decision making stage) of a current state to the state from CSL, a Voronoi diagram clustering algorithm is presented.

3.3 Voronoi diagram clustering

We want to solve the following problem: given a set $\mathcal{P}$ of arbitrary distributed data points $p_1, ..., p_N \; p_i \in \mathcal{R}^2$, find a smaller set Q of cluster center points $q_1, ..., q_m$ $(q_j \in \mathcal{R}^2, \quad m \ll N)$ which optimizes a certain cost function. We minimize the least squares error.

The task of this section is to determine the cluster center points q_j which minimize the following error E:

$$E = \sum_{j=1}^{m} s_j \longrightarrow \min, \tag{3.6}$$

with

$$s_j = \sum_{i \in I_j} ||p_i - q_j||^2,$$

and the index sets

$$I_j = \{i : p_i \text{ is closer to } q_j \text{ than any other } q_k, \ \forall i \neq k\}.$$

Here the notation $||\cdot||$ denotes the Euclidean norm.

A multidimensional Voronoi diagram is a partition of the space E^2 in the regions Reg_j with the following properties: Each point q_j lies in exactly one region Reg_j, which consists of all points $x \in E^2$ that are closer to q_j than to any other point q_k, $j \neq k$, the points q_j are also called Voronoi points.

$$Reg_j = \left\{x \in E^2 : ||x - q_j|| < ||x - q_k||, \ \forall j \neq k\right\}.$$

The index set I_j is defined as,

$$I_j = \{i : p_i \text{ lies in Region } Reg_j\}.$$

3.3.1 Adaptive clustering

To find a local minimum in formula (3.6) a Voronoi diagram based clustering algorithm is described. The idea is based on an adaptive sequential insertion of a new cluster point in the region with the greatest error s_j of the Voronoi diagram of all up to now inserted points.

1. Initialize the first cluster point with the weighted mean of all data points. The corresponding region of the Voronoi diagram is the entire space.

2. Divide the set of data points of the region R_e with the greatest error into two sets and determine the new index sets and their center points.

   - First, compute the coordinate axis with the largest projection variance:

     $$k = arg \max_{l=1,2} \left\{ \sum_{i \in I_e} (p_i^l - q_e^l) \right\},$$

     with x^l is the l-th component of vector x.

   - Then divide all points p_i $(i \in I_e)$ by a hyperplane through q_e perpendicular to the k-th coordinate axis. The new index sets I_{e1}, I_{e2} and their cluster centers m_1, m_2 are determined as follows:

     $$I_{e1} = \{i : p_i^k \leq q_e^k,\ i \in I_e\} \text{ and } I_{e2} = \{i : p_i^k > q_e^k,\ i \in I_e\},$$

     with cluster centers

     $$m_1 = \frac{1}{|I_{e1}|} \sum_{i \in I_{e1}} p_i \text{ and } m_2 = \frac{1}{|I_{e2}|} \sum_{i \in I_{e2}} p_i,$$

     where $|I_e|$ is the number of elements in this set.

3. Move the cluster point q_e to the center m_1, insert a new cluster point m_2 and actualize the index sets of the affected regions.

4. Move the cluster center points in the weighted centroid of its related data points. Actualize the voronoi diagram, the index sets and the set of changed regions.

5. Repeat the steps 2-4 until clustering requirement is met.

The performance of this algorithm was investigated by Schreiber(1997), for many distributions he always achieved minimum of cost function or was very close to it. With this algorithm we have an instrument, that allows for quick state recognition of a multivariate time series.

The adaptive multidimensional clustering algorithm has following properties:

- allows weighted data.
- needs no initial cluster configuration.
- yields different degrees of accuracy.
- allows the optimization of a second cost function.
- produces a triangular based hierarchical surface representation.

In the next section we will generate for adaptive clustering algorithm required data. For this aim we will use bootstrap sampling methods.

3.4 Sampling variability estimation

As we have seen in the previous section, to make adaptive clustering of a detected state, the Voronoi cluster center points must be given. To compute the Voronoi cluster center points a sequence of relevance and time lag values from the same component state is needed. The winning of the data in such way could be realized in context of variable significance testing and is equivalent to the problem of sampling variability estimation.

Variable significance testing is one task of the problem "model identification". The diagram 3.4 shows the ingredients of this procedure.

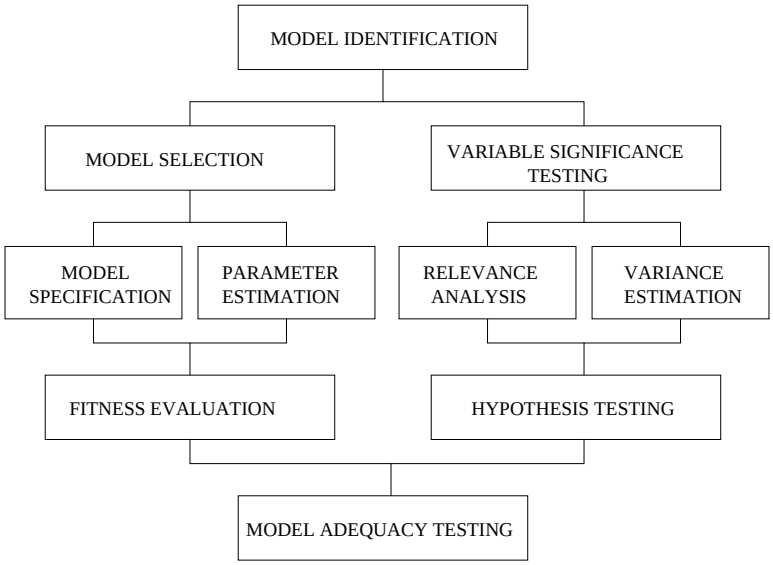

Figure 3.4: Model identification.

We want to estimate statistical significance of the explanatory variables. Irrelevant variables are removed in a way analogous to step by step variable selection in linear regression. It involves three stages:

- Define variables relevant to the model.
- Estimate the sampling variation of the relevance measure.
- Testing the variance relevancy

with linear models as

$$y = b_0 + \sum_{i=1}^{n} b_i x_i + \epsilon.$$

The relevance of x_i in this model is measured by the magnitude of the partial derivative of y with respect to x_i, and that is the coefficient b_i

of x_i. Testing the hypothesis that the variable is relevant consists of testing $H_0 : b_i = 0$ against the alternative $H_0 : b_i \neq 0$.

With non-linear models there are complications how to define the relevance. Unlike linear models , the partial derivative of the network output y^i with respect to input y^j is not a constant but varies through the range of y^j. As yet there does not exist a universally accepted relevance criterion. Zapranis (1996) demonstrated that testing whether the weights are significantly different than zero can be a misleading way of testing the overall significance of an explanatory variable. A number of criteria each measuring different aspects of relevance is proposed. Amongst others,

- Average magnitude of the partial derivative $y^i_{\cdot}$ with respect to $y^j_{\cdot}$
- Maximum sensitivity
- Average elasticity
- Classical neural network relevance analysis.

The second difficulty arises with estimating sampling variation. Since by specification, the true model has a fixed functional form it is often also assumed that the variance of the partial derivatives follows a particular distribution, thus, providing the basis for using asymptotic arguments to estimate standard error. It is more common to carry out stochastic simulations, such as bootstrapping, to provide empirical probability distributions for the relevance measures. Zapranis et. al. (1996) evaluated a number of common ways for estimating sampling variability relating to relevance measures.

- Bootstrapping with a fixed set of random initial values for the weight matrices.

- Bootstrapping in a local minimum
- Bootstrapping close to a local minimum.

It was shown in this paper that common bootstrap has a tendency to overestimate the variance. A parametric bootstrap was proposed which produces acceptable estimates and it is computationally attractive.

The third task involves testing the hypothesis of irrelevance. Once the empirical distribution of the relevance measure is obtained, hypothesis can be tested by forming confidence intervals either, of the crude form $\hat{\theta} \pm z_a \hat{\sigma}$, with $\hat{\theta}$ being the estimated relevance measure and z_a is the $100(1-\alpha)$ percentile point of the standard normal distribution and $\hat{\sigma}$ is the estimated relevance magnitude variance.

The procedure for variable selection deals with the statistical significance of the explanatory variables. It involves quantifying the relevance measure, estimating the sampling variability of this measure and testing the hypothesis of insignificance. This is shown diagramatically in Figure 3.5.

Of the tree legs by far the most important is that of estimating sampling variability. The two main choices are either using the distribution of the empirical p.d.f. of the parameters or the asymptotic p.d.f. Both have disadvantages. Empirical estimation is computationally intensive and the asymptotic estimates have the tendency to underestimate the true variance.(Zapranis, et al 1996). In the following we use bootstrapping procedure to estimate the empirical probability distribution function of the parameters. In context of LD modeling we mean under parameters the relevance values and the time lag values.

The diagram 3.5 shows that here exist two alternative computation ways of the bootstrap:

- bootstrapping blocks,

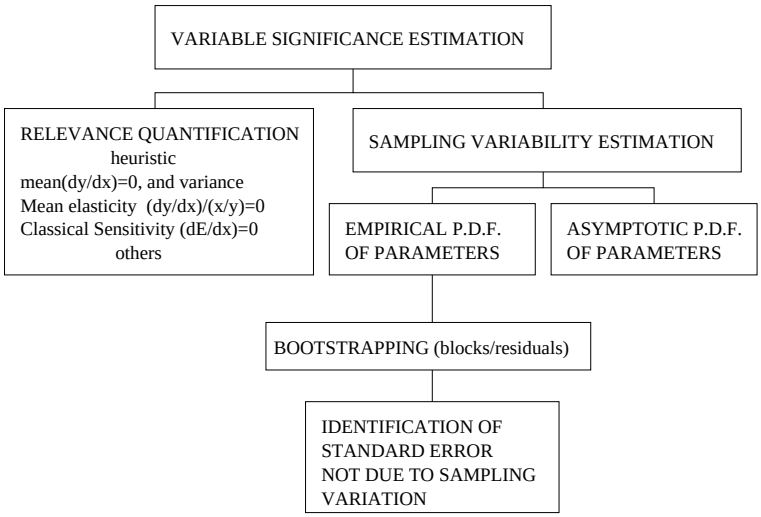

Figure 3.5: Variable significance estimation.

- bootstrapping residulals

In the next section we describe bootstrap based model estimation, i.e. we estimate the model(train the network) with residual bootstrap input, output variables.

3.5 Bootstrapping neural networks

The basic idea underlying the bootstrap is that one may investigate the sampling variability in a statistic of interest (e. g. effect of a given predictor variable, under the condition that the variables are known), by computing that statistic repeatedly from randomly drawn samples having the same probability distribution as the sample at hand one may then observe the distribution of the statistic of interest over the resampling interests. Under appropriate conditions, the distribution

of the resampled statistics corresponds to the sampling distribution of the statistic of interest(Giné and Zinn 1990). With other words let $\{X_1, ..., X_N\}$ be a random sample of size N from a population with distribution F. Let $T(X_1, ..., X_N; F)$ be the specified random variable of interest, possibly depending upon the unknown distribution F. Let F_N denote the empirical distribution function of $\{X_1, ..., X_N\}$ i.e the distribution that puts mass $1/N$ at each of the points $X_1, ..., X_N$. The bootstrap method is to approximate distribution of $T(X_1, ..., X_N; F)$ under F by that of $T(X_1^*, ..., X_N^*; F_N)$ under F_N where $\{X_1^*, ..., X_N^*\}$ denotes a random sample of size N from F_N.

3.5.1 Bootstrapping blocks

Independence of the observations is crucial for Efron's(1979) bootstrap. It is easily seen that they give incorrect answers if dependence is neglected, Singh(1981). For time series data a modification is needed. One such modification, which does not assume a special structure such as ARMA or Markov processes, is the blockwise bootstrap of Künsch(1989). It proceeds by resampling blocks of N consecutive observations from the original sample. Different blocks are conditionally independent.

Definition 3.1. For observations $X_1, ..., X_N$ from a stationary process the empirical m-dimensional marginal is

$$\rho_N^m = (N - m + 1)^{-1} \sum_{t=0}^{N-m} \delta_{(X_{t+1}, ..., X_{t+m})},$$

where δ_y is the point mass at $y \in R^m$.

We select blocks of length b at random. Assuming that $N = kb$

with $k \in N$, the bootstrap m-dimensional marginal is therefore

$$\rho_N^{m,*} = N^{-1} \sum_{j=1}^{k} \sum_{t=S_{j+1}}^{S_{j+b}} \delta_{Y_t},$$

where $S_1, ..., S_k$ are i.i.d. uniform on $\{0, ..., N-b\}$. We write this as follow:

$$\rho_N^{m,*} = N^{-1} \sum_{t=1}^{N} f_t \delta_{Y_t},$$

with $f_t = \#\{j; t-b \leq S_j \leq t-1\}$ or as

$$\rho_N^{m,*} = N^{-1} \sum_{t=1}^{N} \delta_{Y_t^*},$$

where the k blocks $(Y_1^*, ..., Y_b^*), (Y_{b+1}^*, ..., Y_{2b}^*), ..., (Y_{N-b+1}^*, ..., Y_N^*)$ are i.i.d. with distribution

$$(N-l+1)^{-1} \sum_{t=0}^{N-b} \delta_{(Y_{t+1}, ..., Y_{t+b})}.$$

We can now form the bootstrap statistics

$$T_N^* = T(\rho_N^{m,*}),$$

where $Y_1, ..., Y_n$ are fixed but $S_1, ..., S_k$ vary. After this method of generation the bootstrap world we can form the residual blocked bootstrap samples. Assume we have a vector of random variables $\{S_1, ..., S_k\}$ one k-time drawing than the blocked bootstrap samples looks as follows:

$$\Big((Y_1^*, y_1^{i*}), ..., (Y_b^*, y_b^{i*})\Big), ..., \Big((Y_{N-b+1}^*, y_{N-b+1}^{i*}), ..., (Y_N^*, y_N^{i*})\Big),$$

where $Y_\cdot^* \in \mathcal{R}^d$ and $y_\cdot^{i*}$ are real valued. With this choice of the block size we retain the correlation in observations they are less then b time units apart.

With B-times drawing of S values we can approximate the unknown distribution of $T_N - T(F^m)$ by the distribution of $T_N^* - T_N$. If we assume that the original data $(Y_1, ..., Y_N)$ comes from LD models, it is successfully to choose the length of the blocks b large enough so that observations more then b time units apart will be nearly independent. Therefore we choose for b,

$$b = max(L_0),$$

where $L_0 \in \mathcal{N}^d$ is the true vector of time lags in the current state.

3.5.2 Bootstrapping residuals

A further way to bootstrap time dependent data is the so called residuals bootstrap. The probability structure of the LD model in a particular state is as follows

$$y_t^i = f_{nn}(Y_{t-L_0}, \theta_0) + \epsilon_t \text{ for } t = 1, ..., N.$$

The error terms are assumed to be normally distributed with expectation 0 and finite variance. We calculate approximate errors

$$\hat{\epsilon}_t = y_t^i - f_{nn}(Y_{t-\hat{L}}, \hat{\theta}),$$

where $\hat{\beta}$ and $\hat{L}$ are the learned weight matrices after the neural network training with the original data set, and with in the 2.1.1 described algorithm estimated optimal time lag vector. First we select a random sample of bootstrap error terms $\hat{F} \to (\epsilon_1^*, ..., \epsilon_N^*)$, where $\hat{F}$ is the empirical distribution function of the $\hat{\epsilon}_t$. Then we generate the bootstrap replications from LD model probability structure

$$y_t^{i*} = f_{nn}(Y_{t-\hat{L}}, \hat{\theta}) + \epsilon_t^* \text{ for } t = 1, ..., N.$$

Finally we get a bootstrap training set

$$(Y_{1-\hat{L}}, y_1^{i*}), ..., (Y_{N-\hat{L}}, y_N^{i*}),$$

with B times drawing error terms $((\epsilon_1^*)_b, ..., (\epsilon_N^*)_b)$, $b = 1, ..., B$ from $\hat{F}$, we win B bootstrap training sets

$$\begin{aligned}
&\left\{\left(Y_{1-\hat{L}}, (y_1^{i*})_1\right), ..., \left(Y_{N-\hat{L}}, (y_N^{i*})_1\right)\right\} \\
&... \\
&\left\{\left(Y_{1-\hat{L}}, (y_1^{i*})_B\right), ..., \left(Y_{N-\hat{L}}, (y_N^{i*})_B\right)\right\},
\end{aligned}$$

where

$$(y_t^{i*})_b = f_{nn}(Y_{t-\hat{L}}, \hat{\theta}) + (\epsilon_t^*)_b \text{ for } b = 1, ..., B.$$

We take the residual bootstrap algorithm to estimate the sampling variability. We have, $g(x) = E\{y_t^i | Y_{t.} = x\}$, in the further discussion we assume for simplicity that $Y_{t.} = Y_{t-\hat{L}}$. To initialize the bootstrap procedure, we follow Franke/Neumann(1997) and take any uniformly consistent estimator $\hat{g}_N$ for g, i.e for which

$$\sup_{x \in supp(p)} \{|\hat{g}_N(x) - g(x)|\} \xrightarrow{\mathcal{L}} 0,$$

where we assume that the random vector Y_t has a density $p(y)$.

In the following we get a theorem that shows, that with residual bootstrap sampled data trained neural network weight matrices have still well known asymptotic properties, and therefore can be used for determination of Voronoi cluster center points.

To guarantee that the residual bootstrap works as neural network approximator, we must prove a central theorem about the asymptotic properties of the estimated weights. Before we give the theorem following assumptions must be satisfied:

$$\begin{aligned}
D_0(\theta) &= E(y_t^i - f_{nn}(Y_{t.}, \theta))^2 \\
&= \int \left[(g(Y) - f_{nn}(Y, \theta))^2 + \sigma_\epsilon^2\right] p(Y) dY = \min_{\theta \in \Theta}.
\end{aligned}$$

For the true vector of weights θ_0 we have,

$$\sqrt{N}(\hat{\theta}_N - \theta_0) \stackrel{N\to\infty}{\longrightarrow} N(0, \Sigma_2),$$

with covariance matrix

$$\Sigma_2 = A_0^{-1} B_0 A_0^{-1},$$

where

$$\begin{aligned} A_0 &= E\left[\nabla^2\left((y_t^i - f_{nn}(Y_{t.}, \theta_0))^2\right)\right], \\ B_0 &= E\left[\nabla\left((y_t^i - f_{nn}(Y_{t.}, \theta_0))^2\right) \cdot \nabla\left((y_t^i - f_{nn}(Y_{t.}, \theta_0))^2\right)'\right], \end{aligned}$$

and ∇ denotes the gradient and ∇^2 the Hessian with respect to θ_0. Under assumptions 2.1, 2.2 and 2.5 we assume for the true regression function g that,

Assumption 3.2. *$\hat{g}_N$ is uniformly consistent on the support of p, that is,*

$$\sup_{x \in supp(p)} \left\{ |\hat{g}_N(x) - g(x)| \right\} = o_P(1).$$

Let $\hat{\theta}_N^*$ be the weight vector after the training the network from the residual bootstrap training set, i. e. the solution of

$$\hat{D}_N^*(\theta) = \frac{1}{N} \sum_{t=1}^{N} \left(y_t^{i*} - f_{nn}(Y_{t.}, \theta) \right)^2 = \min_{\theta \in \Theta_H} ! \ ,$$

θ_N^* is the solution of:

$$D_N^*(\theta) = \frac{1}{N} \sum_{t=1}^{N} \left(\hat{g}_N^*(Y_{t.}) - f_{nn}(Y_{t.}, \theta) \right)^2 + (\sigma_\epsilon^*)^2 = \min_{\theta \in \Theta_H} ! \ ,$$

θ_0^* is the solution of:

$$D_0^*(\theta) = E\left(y_t^{i*} - f_{nn}(Y_{t.}, \theta)\right)^2 = \min_{\theta \in \Theta_H} ! \ .$$

Franke and Neumann have shown for regression models that the bootstrap works for neural network function estimates. From their results, it is rather obvious that the bootstrap holds also for similar autoregression models like $y_t^i = g(Y_t - L) + \epsilon_t$ if the time series satisfies appropriate mixing conditions. Such conditions on the regression function g have been given by Franke, Kreis, Mammen and Neumann. So we expect a result described in following lemma:

Lemma 3.3. *For residual bootstrap and under assumptions 3.2 til 3.5 it holds,*

$$L\left(\sqrt{N}\begin{pmatrix} \hat{\theta}_N^* - \theta_N^* \\ \theta_N^* - \theta_0^* \end{pmatrix} \middle| (Y_1, y_1^i), ..., (Y_N, y_N^i)\right) \to N\left(0, \begin{pmatrix} \Sigma_1^* & 0 \\ 0 & \Sigma_2^* \end{pmatrix}\right),$$

where

$$\begin{aligned} \Sigma_1^* &= A^*(\theta_0^*)^{-1} B_1^*(\theta_0^*) A(\theta_0^*)^{-1}, \\ \Sigma_2^* &= A^*(\theta_0^*)^{-1} B_2^*(\theta_0^*) A(\theta_0^*)^{-1}, \end{aligned}$$

with $A^*(\theta_0^*) = hess(D_0^*(\theta))$ *and*

$$\begin{aligned} B_1^*(\theta_0^*) &= 4 \cdot \int \nabla(\sigma_\epsilon^2(x) p(x) f_{nn}(Y, \theta) \cdot \nabla(f_{nn}(Y, \theta)') dx \\ B_2^*(\theta_0^*) &= 4 \cdot \int (y^{i*} - f_{nn}(Y, \theta))^2 \nabla(f_{nn}(Y, \theta)) \cdot \\ &\quad \cdot \nabla(f_{nn}(Y, \theta)') p(x) dx, \end{aligned}$$

where ∇ *denotes the gradient of the function with respect to* θ.

We do not give a proof here, as it would require the derivation of lots of auxiliary results which are available for the regression case but not for the time series case in the literature, but for which it is rather obvious that they hold under appropriate conditions.

3.6 Relevance measures

The data generated by the residual bootstrap procedure allow us to estimate probability distribution function of relevance measures of several components in a multivariate time series by lag dependent modeling. Figure 3.6 shows this algorithm. In the algorithm, l_t^{ij} is the time lag value estimated under consideration of the t-th observation. Whether a change point is significant or not, depends on the estimated relevance value and on, from observer chosen relevance threshold c^{ij}. This threshold is based on experience. If the relevance $r^{ij} < c^{ij}$, the change point will be considered as insignificant for the observed component.

We used the neural network bootstrapping procedure in LD models in the following way. We used bootstrapped data to generate a sequence of relevance values $R_b^i, \quad b = 1, ..., B$, where B is the number of bootstrap replications. With this sequence we make adaptive clustering and determine the Voronoi cluster center point for the current state.

The most commonly used measure of relevance is the average derivative (AD) since, the average change in $\hat{y}^i$ for a very small perturbation $\delta y^j \longrightarrow 0$ in the independent variables y^j, is simply:

$$AD(y^j) = \frac{1}{N} \sum_{t=1}^{N} \frac{\partial \hat{y}_t^i}{\partial y_{t-l^{ij}}^j} \equiv mean \left(\frac{\partial \hat{y}^i}{\partial y^j} \right).$$

However, because of cancellations between negative and positive values, the average derivative might not be representative for the effects of y^j on $\hat{y}$, and that is why we propose using their absolute values or the squares of derivatives. To quantify the AD a suitable measures

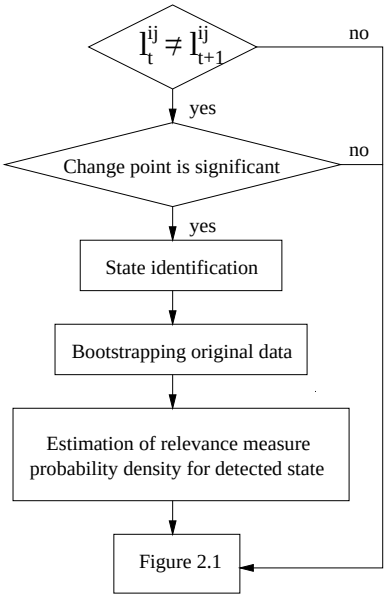

Figure 3.6: Bootstrapping LD models.

are,

$$AD(y^j) = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{\partial \hat{y}_t^i}{\partial y^j_{t-l^{ij}}}\right|$$

$$r = N^{-\frac{1}{2}}\sqrt{\sum_{t=1}^{N}\left(\frac{\partial \hat{y}_t^i}{\partial y^i_{t-l^{ij}}}\right)^2} \times \frac{\sigma_x}{\sigma_\epsilon}, \qquad (3.7)$$

σ_x is the standard deviation of the independent variable and σ_ϵ is the standard deviation of the residual error.

Sometimes although $\hat{y}^i$ is sensitive to y^j only in a small percentage of input vectors y^j, this can be quite important in the context of particular application. Such requirements give rise to the following

measures,

$$MaxD(y^j) = \max_{t=1,...,N} \left| \frac{\partial \hat{y}_t^i}{\partial y^j_{t-l^{ij}}} \right|.$$

Not so important is the measure $(\min(DM))$

$$MinD(y^j) = \min_{t=1,...,N} \left| \frac{\partial \hat{y}_t^i}{\partial y^j_{t-l^{ij}}} \right|.$$

Another quantification of $\hat{y}^i$-s sensitivity to y^j, which is relevant for financial and biometrical applications, is the average percentage change in $\hat{y}^i$ for a one percent change in y^j or more commonly known as the "Average elasticity" of $\hat{y}^i$ to y^j (ReZaUt96) which is computed as

$$AvE(y^j) = \frac{1}{N} \sum_{t=1}^{N} \left(\left| \frac{\partial \hat{y}_t^i}{\partial y^j_{t-l^{ij}}} \right| \right) \left(\left| \frac{y^j_{t-l^{ij}}}{\hat{y}_t^i} \right| \right), \ \hat{y}_t^i \neq 0 \ \ \forall \, t = 1, ..., N.$$

Another measure is computed as f.e. y^j's average contribution to the magnitude of the gradient vector. As average contribution to the gradient magnitude we can take a measure of sensitivity, a so called "standard deviation" of the derivatives across the sample, which measures the dispersion of the derivatives around their mean i.e.

$$SD(y^j) = \left[\frac{1}{N} \sum_{t=1}^{N} \left(\left| \frac{\partial \hat{y}_t^i}{\partial y^j_{t-l^{ij}}} \right| - AvE(y^j) \right)^2 \right]^{\frac{1}{2}},$$

dividing $SD(y^j)$ by the mean, provides us with the coefficient of variation that is the standard deviation per unit of sensitivity.

$$CV(y^j) = \frac{SD(y^j)}{AvE(y^j)}.$$

Chapter 4

Time Series Modeling

In the next three sections, we model time series from intensive care medicine, from the German stock market and time series that are simulated with LD models.

4.1 Simulated LD(3) time series

We simulate the following time series:

$$\begin{aligned} y_t^1 &= c_1 + \epsilon_t, \quad t = 1, ..., 30 \\ y_t^2 &= c_2 + \epsilon_t, \quad t = 1, ..., 30 \\ y_t^3 &= c_3 + \epsilon_t, \quad t = 1, ..., 30, \end{aligned}$$

where $\epsilon_t \sim N(0, 1)$. We take the following regression functions. For simplicity we consider only equal partitions, i.e. $\Xi^1 = \Xi^2 = \Xi^3$, where

$$\Xi^i = \Big(30, 160, 440\Big)', \quad \imath = 1, 2, 3.$$

We fix time lag and regression function matrices for each component,

$$L^1 = \begin{bmatrix} 0 & 3 & 10 & 1 \\ 0 & 5 & 7 & 2 \\ 0 & 7 & 8 & 7 \end{bmatrix}, M^1 = \begin{bmatrix} c_1 & x & \frac{1}{\exp(x)} & \frac{1}{30}x + 4 \\ c_1 & \frac{1}{21}x & \frac{1}{2}x & x \\ c_1 & \frac{1}{\exp(x)} & 3\cos(x) & \frac{1}{\exp(x)} \end{bmatrix}$$

$$L^2 = \begin{bmatrix} 0 & 6 & 1 & 6 \\ 0 & 1 & 1 & 1 \\ 0 & 9 & 7 & 3 \end{bmatrix}, M^2 = \begin{bmatrix} c_2 & \frac{1}{6}\sqrt{x} & \frac{1}{6x} + 0.2 & \frac{1}{3}x \\ c_2 & \frac{2}{5}\sin(x) & \frac{1}{\exp(x)} & \frac{2}{5}x \\ c_2 & \sin(x) & 0.2x & 0.4x \end{bmatrix} \quad (4.1)$$

$$L^3 = \begin{bmatrix} 0 & 6 & 8 & 3 \\ 0 & 8 & 6 & 2 \\ 0 & 1 & 2 & 10 \end{bmatrix}, M^3 = \begin{bmatrix} c_3 & \frac{1}{\exp(x)} & 0.7\sin(x) & 0.7\sin(x) \\ c_3 & \sin(x) & \frac{1}{\exp(x)} & \frac{2}{\exp(x)} \\ c_3 & 0.4x & 0.7\cos(x) & 0.2\cos(x) \end{bmatrix},$$

where $c_1 = 10$, $c_2 = 15$, $c_3 = 20$. The next figure shows a simulated three dimensional time series.

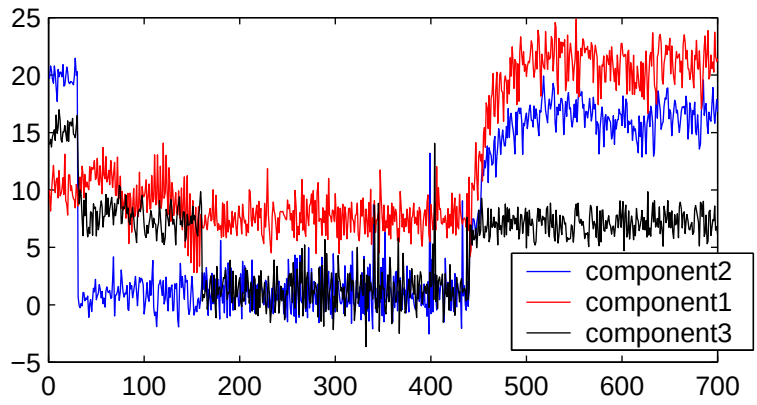

Figure 4.1: LD(3) time series.

We estimate the third component(see 4.1) from this time series with LD(3) models in interval [180, 330]. We take a blocked neural network with two sigmoid neurons in each block see figure 1.5, and let it train for several lags in range $[1, 10]$ for each component. We estimate the time lag matrix with the in section 2.1.1 described algorithm. With this time lags, we compute relevance values for each component, where for relevance value computation the average derivative(AD)(see(3.7)) was chosen. Figure 4.2 shows estimation results.

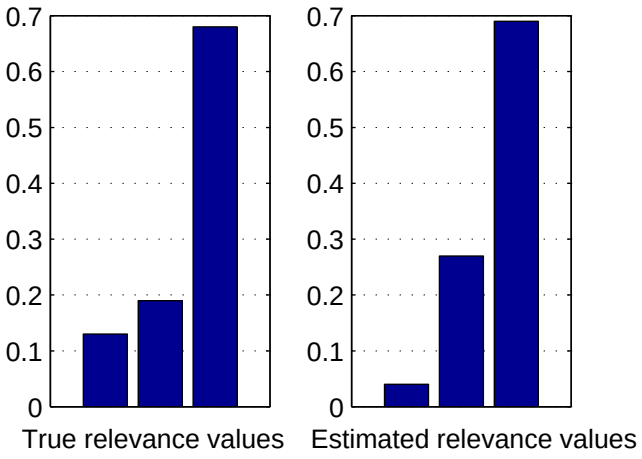

Figure 4.2: Relevance barplot for y_t^3, $t = 180, ..., 330$.

Note that the true relevance(AD) values are obtained if we take the partial derivatives of functions and time lag values from (4.1) and put them into the average derivative formula(see 3.7) for each component.

In the interval [180, 330] the true time delay values with respect to each component are listed in the third column of L^3(see 4.1), also $l_3^{31} = 8$, $l_3^{32} = 6$, $l_3^{33} = 2$. We estimate the following time lag values:

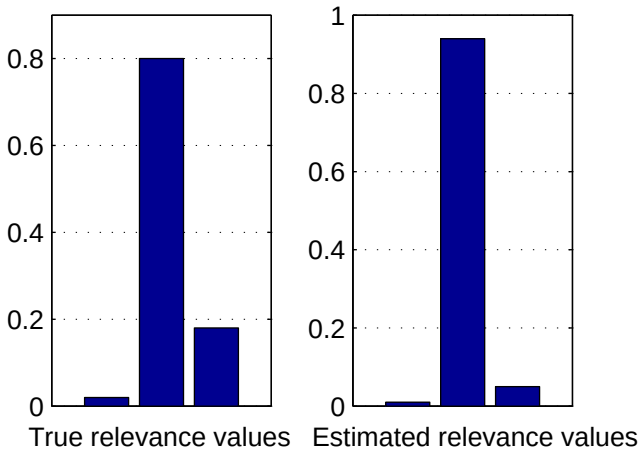

Figure 4.3: Relevance barplot for y_t^2, $t = 180, ..., 330$.

$\hat{l}_3^{31} = 6$, $\hat{l}_3^{32} = 6$, $\hat{l}_3^{33} = 2$. The wrong estimation of time lag $\hat{l}_3^{31} = 6 \neq l_3^{31}$ could be explained through lemma 2.11 and theorem 2.10 in the second chapter.

Finally we model the second component of the time series with the same conditions in the same interval. The relevance barplot is shown in figure 4.3.

The true time lags are listed in the third column of L^2(see (4.1)), also $l_3^{21} = 1$, $l_3^{22} = 1$, $l_3^{23} = 7$. We estimate the following time lag values: $\hat{l}_3^{21} = 4$, $\hat{l}_3^{22} = 1$, $\hat{l}_3^{23} = 7$ from the simulated data.

4.2 Biometric time series

This section is concerned with the problem of modeling and monitoring time series that arise in intensive care medicine. The use of clinical information systems (CIS) in intensive care medicine makes

it possible to report online, simultaneously, and automatically up to 2000 physiological variables, laboratory data, device parameters etc. The CIS is based on a network of autonomous UNIX workstations, one for each bed. Bedside devices (such as monitors, ventilators, etc.) are connected locally via serial interfaces. All patient data are stored on the local hard disc at the bedside and simultaneously mirrored onto a second workstation within the network. An administrative data server is used for administration of the network and the CIS. It may also serve as a communication hub with central data services like the hospital information system. For undisturbed data analysis the patient record is transferred into a secondary SQL (Sybase SQL server) and exported into standard statistical or mathematical software. Thus it is guaranteed that most of the data relevant for patient monitoring is recorded in a regular, reliable, and correct way. Hence the technical requirements for using statistical methods or tools for bedside decision support are fulfilled.

In this section we model time series that were stored in "Städtische Kliniken Dortmund" on a patient from the intensive care unit. We have the data from table 4.1. (These data are acquired in one minute intervals from a standard clinical information system.)

Plot 4.4 shows this nine dimensional time series with a length of 500 observations.

To model an interesting component from this time series we take a neural network with one neuron in each block and observe the "Artmn" variable in the interval of 300-350(quiet phase). Initial values from table 4.2 were used.

We assume that the observed component does not change its state during this quite interval, which seems to be plausible looking at the data. We use this part of the data to fit a lagged neural network and to get initial lag estimates $\bar{l}_k^{ij}$ and weights $\bar{\theta}$. To illustrate the change point detection procedure, we calculate one-step predictors for

Nr.	Variable	Description
1	Artsys	blood pressure(systolic)
2	Artdia	blood pressure(dyastolic)
3	Artmn	Mean pressure
4	Cvp	Central venous pressure
5	Hr	Heart rate
6	Papsys	Pulmonallerteria pressure(systolic)
7	Papdia	Pulmonallerteria pressure(dyastolic)
8	Papmn	Pulmonallerteria mean pressure
9	Puls	Puls

Table 4.1: Biometric data from CIS.

Variable	Description
$N = 50$	Sample size
$A = 15$	Expected time lag searching range
$R = 0.5$	Relevance threshold
$Neur = 1$	Number of neurons in each block
$AN = 300$	Beginning of estimation
$k = 2$	k in LD model estimator(see section 1.6)

Table 4.2: Neural network initialization.

the next 100 minutes, i.e. predictors of y_t^i, $t = 351, ..., 450$. First we use the initial fitted network, i.e. the predictor of y_{t+1}^i is given by

$$\hat{y}_{t+1}^i = f_{nn}(Y_{t-\bar{L}}, \bar{\theta}),$$

where

$$Y_{t-\bar{L}} = (y^1_{t+1-\bar{l}^{i1}_{k(1)}}, ..., y^d_{t+1-\bar{l}^{id}_{k(d)}}),$$

(for details about p-step prediction for $p \in \mathcal{N}$, $p \geq 1$ see section 3.1).

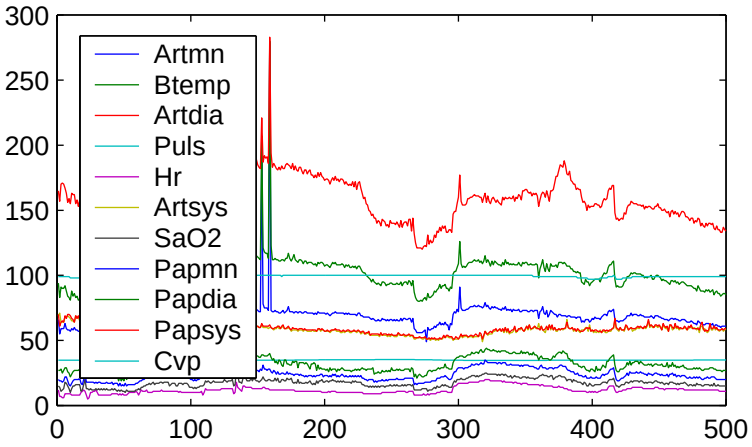

Figure 4.4: Biometric time series.

After observing each new data point, we again estimate the lag and the weights, and we use the updated network for prediction. The one-step forecasts $\hat{y}^i_{t+1}$ together with the true observations y^i_{t+1} are plotted in figure 4.5. Finally, we compute the prediction error. In the course of this computation, we detect change points in the structure of the time series if the estimated lags change in a significant manner. The details are explained in the next section.

Time effort of this calculation was 10 hours on a 433 MHZ processor.

4.3 Online change point detection

The following algorithm describes a simple method for change point detection in a component of a multivariate time series with respect to LD modeling.

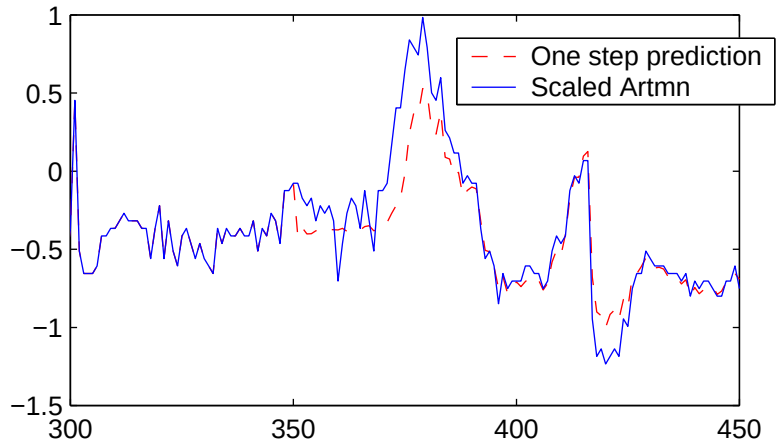

Figure 4.5: Artmn variable LD prediction.

The main variables in lag dependent models are the time lag and relevance between two components. Therefore it is meaningful to base the algorithm that detects a change point on these two variables. The flow diagram 4.6 shows a simple change point detector for the component y^i of interest and for $j = 1, ..., d$.

In this flow diagram $c^{ij} > 0$ is an experience based constant chosen by an observer and l_t^{ij} is the time lag value estimated under consideration of the t-th observation. The algorithm works as follows: At time $t + 1$ we approximate the time series with a LD model, that means we apply the time lag estimation algorithm from section 2.1.1 and estimate the vector of time lags. After the estimation procedure we compare this time lag vector with the time lag vector that was estimated at time t. In the next step we select all components for which the time lags have changed. The next step checks whether, one of selected components has "big" relevance or not. If the relevance

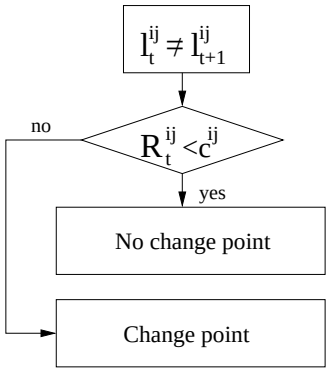

Figure 4.6: Algorithm for change point detection.

is bigger then a constant the algorithm detects the change point with respect to this component.

We use this algorithm and detect change points in the biometrical time series for variables "Artmn" and "Puls"(see plot 4.4) in the intervals 350-450 for "Artmn" and 250-400 for "Puls". The graphs 4.7 and 4.8 show detected change points for the threshold values $c^{3,j} = 0.5, \ \ j = 1, ..., d$, and $c^{9,j} = 0.75, \ \ j = 1, ..., 9$ respectively.

In figure 4.7, the variable y_t^i ="Artmn" starts with a slight upward trend, which is broken around time $t = 355$. There two close change points are detected in components "Cvp" and "Hr" which presumably correspond to the same structural change. Around time $t = 365$, a strong upward trend starts which is detected in component "Cvp". It ends around time 380, which is detected in "Cvp" and "Hr". The following change points around time 395 and later are not detected. The reason may be that the algorithm needs a long enough interval where the network providing the forecasts does not change too

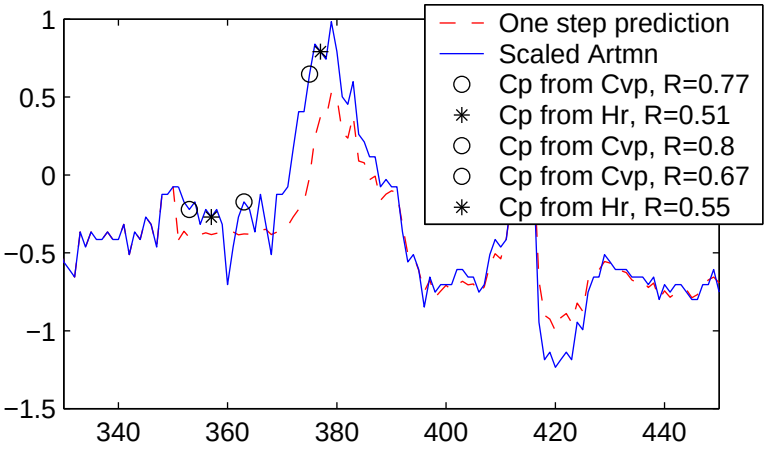

Figure 4.7: Detected CP in "Artmn" variable.

much. Therefore, after a dramatic change in the structure of the time series which seems to happen around time 380, the algorithm needs some time to again adapt the network to the current data. We see that it is able to detect change points after a "long" enough quiet phase, but it may have difficulties to detect further change points which follow too soon after a severe structural change.

In figure 4.8, change points in the variable y_t^i="Puls" are also quite well detected around time 270, 275, 300, 320 and 400. The end of the upward trend around time 360 is not detected. This may be due to the same reason discussed in the last section. However, here we observe another potential problem. The time series has obviously an increasing variance after about time 300, which makes changes in the trend harder to detect. As an extension, one would consider time series models which take into account a conditional heteroscedasticity explicitly like in Franke(2000).

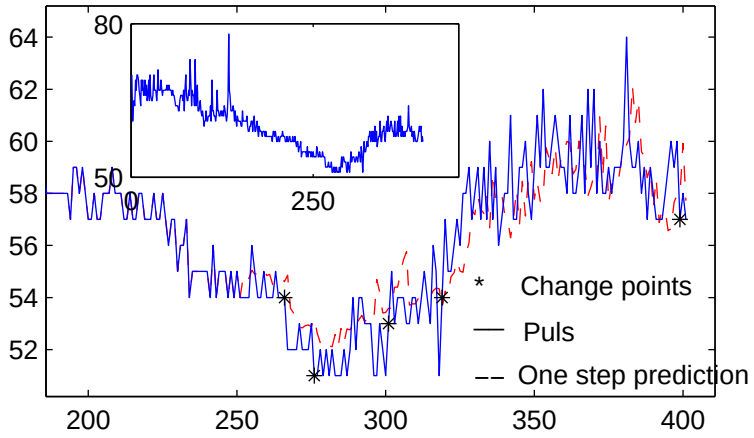

Figure 4.8: Detected CP in "Puls" variable.

Figure 4.9 shows the error values for each prediction step, for the variable y^i ="Puls"

$$error = \hat{y}_t^i - y_t^i, \quad t \in \text{"current state"}.$$

This example demonstrates the usefulness of the algorithm in biometric data analysis. The positive property of the LD modeling and especially of the change point detector is the possibility to adapt. With the choice of the constant c^{ij}(in example $c^{i\cdot}$ is a percentage value), we tune the sensitivity of the algorithm to the changes in time series. For "small" c^{ij} we consider almost every change point. In this situation we must pay attention to the fact that the lag estimator for small relevance values could not be uniquely identifiable and therefore misleading for the observer in the sense that the nonidentifiably unique estimated time lag values lead to the nonidentifiably unique detected change points.

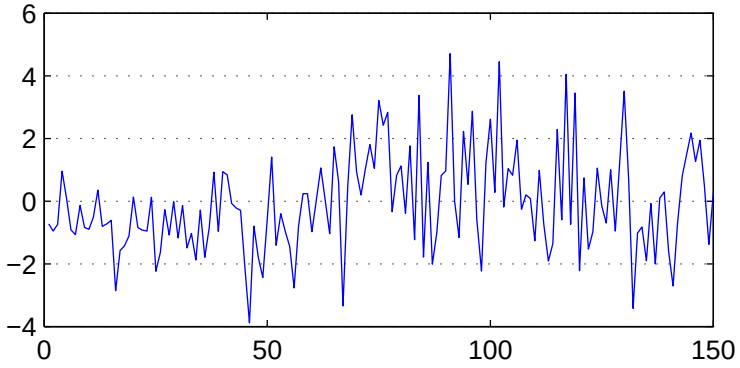

Figure 4.9: Prediction minus "Puls".

Figure 4.8 shows four detected states after the first change point. We bootstrap the data and detect Voronoi cluster center points for each state. For $B = 200$ we obtain Voronoi cluster center points from figures 4.10, 4.11, 4.12, 4.13. The figures show the 9 variables, where the variable y_t^i of interest "Puls" is variable no. $i = 5$. The height of the bars corresponds to the average relevancy of variable no. i compared to the variable no. $j = 1, ..., 9$ where the average is taken over the time interval corresponding to each state. The number above the bar is the estimated lag. "nd" stands for "not defined" as in these cases the relevances are close to 0 that the estimating lags makes no sense.

As a simple observation we note that in this case the "Puls" variable goes down, if the relevance to himself and to the variable "Heart rate" is low, and goes up if the relevance is high.(see figure 4.8)

A further approach of the lag dependent modeling to the biometric and other time series is the possibility to observe the development of the relevance values over time. This way of time series observation

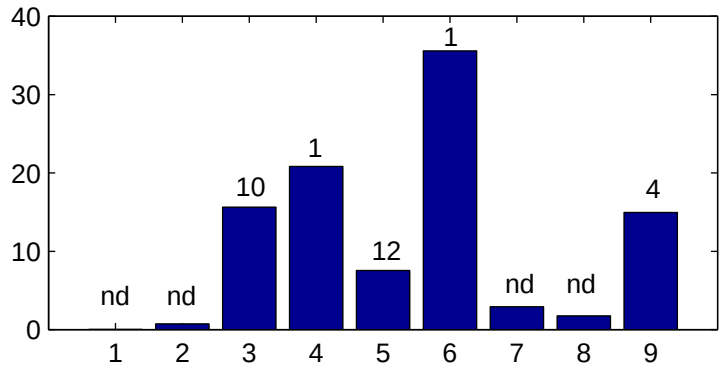

Figure 4.10: First state relevance and time lag.

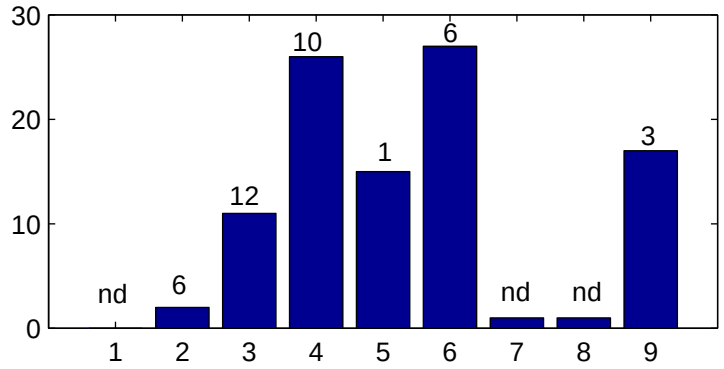

Figure 4.11: Second state relevance and time lag.

will not be considered anymore in this work.

Finally, we apply the change point detector to simulated data from section 4.1 and try to detect well known change point $\xi_\cdot^i = 440$ $i = 1, 2, 3$. For the second component we detect the change point at

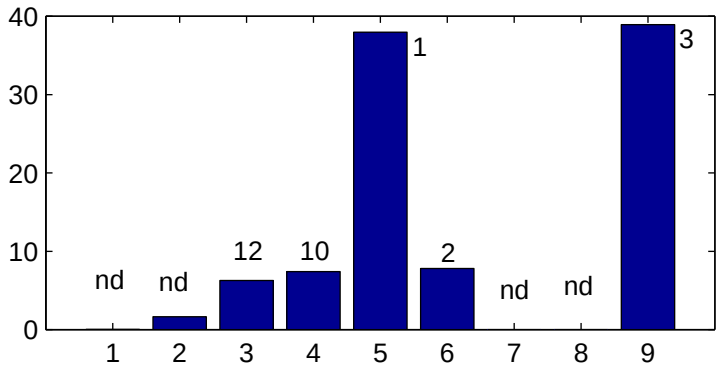

Figure 4.12: Third state relevance and time lag.

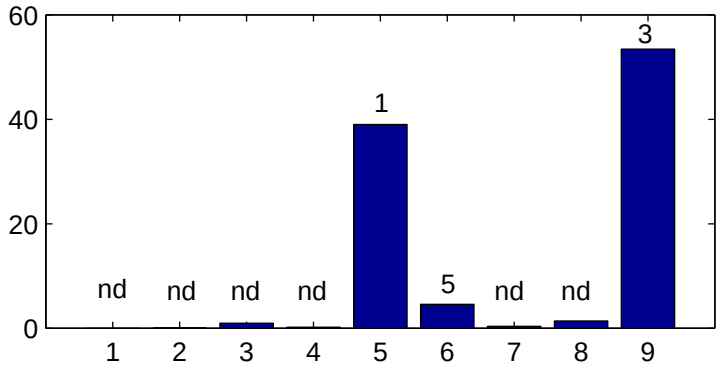

Figure 4.13: Fourth state relevance and time lag.

$t = 444$ by the relevance threshold choice of $c^{2\cdot} = 0.1$(see plot 4.14).

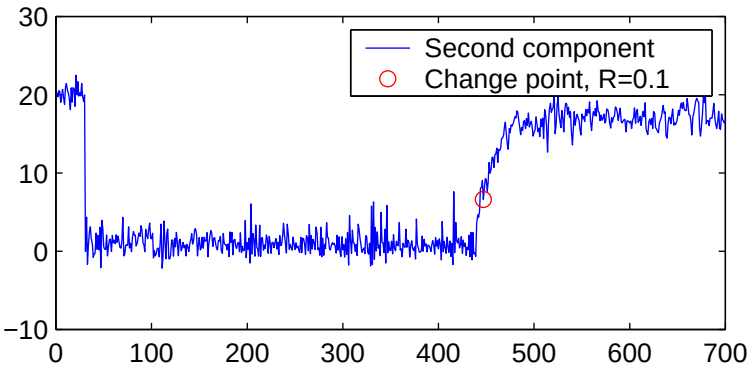

Figure 4.14: Detected CP.

4.4 Stock price prediction

Artificial neural networks, extended Kalman filters, nonlinear filters, hidden Markov models and threshold models have attached a lot of attention in recent years as a means of modeling chaotic and financial data. The main assumption is that, even if the data may look random at first glance, there must be a low dimensional deterministic structure which can be exploited to identify hidden characteristic patterns and to facilitate accurate forecasting. We apply lag dependent modeling techniques to the financial market data.

In this section, we explore the ability of lag dependent models to predict the future stock price given recent price movements. For this aim we take a blocked neural network with one neuron in each block, and one hidden layer. The graph 4.15 shows the German DAX, and its one step prediction with lag dependent models. For relevance threshold we choose $c = 0.3$. The plot 4.15 shows that no change points was detected. All procedures were programmed in LINUX MATLAB 5.3.

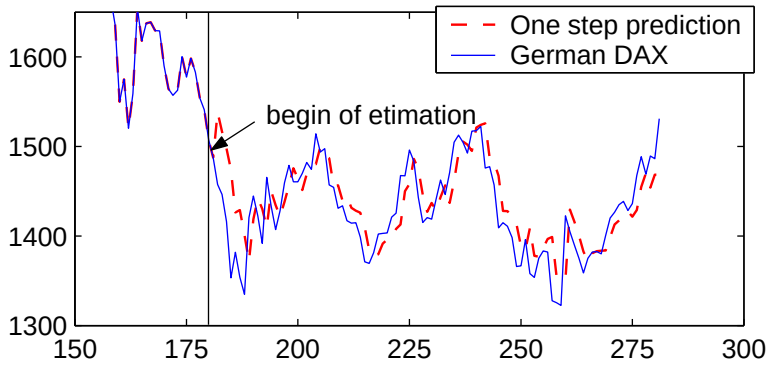

Figure 4.15: German DAX LD modeling.

By the LD modeling of stock prices we have the problem of shortage of data that we can present to the neural network. By the LD modeling of German DAX we used the fact that the index could be considered as a function of its ingredients. The problem was that all time delays were estimated to value one, and therefore the detection of change points was not possible.

Through these considerations we can make the statement about a functionality of LD models in the praxis.

LD models provide good prediction and change point detection results in the multivariate time series with the property of fluctuations in the time dependency between its components. The main problems by the LD modeling procedure are:

- The choice of the range for time lag values,
- The choice of the relevance threshold,
- Modeling after the change point detection.

In our simulation results the third problem was solved by the fact that the components they has detect a change point, were removed from the modeling procedure for time interval with length of the sample size. The problems of selection of the relevance threshold and the range of time lag values could be solved by experience.

Appendix A

Time Series Analysis

A.1 Multivariate time series modeling

In the study of multivariate time series analysis, a framework is needed for describing not only the properties of the individual series but also the possible cross relationships among the series. With these aims, the class of linear and nonlinear vector autoregressive time series models is developed.

A.1.1 Linear models

The main building block of theoretical linear models is the series that is independent and identically distributed (i.i.d.). A series $\epsilon_t, \quad t = 1, ..., N$ is i.i.d. if all terms are independent of the others and all have the same distribution function. If $p_t(x)$ is the probability density function of ϵ_t, then $p_t(x) = p(x)$ for all t and the joint distribution of any set of n components of ϵ_t is just the product of the marginal distributions. Let $Y_t = (y_t^1, ..., y_t^d)'$ denote a d-dimensional time series vector of random variables.

Definition A.1. The process $\{Y_t\}$ is called stationary if the probability distributions of the random vectors $(Y_{t_1}, ..., Y_{t_n})$ and $(Y_{t_1+l}, ..., Y_{t_n+l})$ are the same for arbitrary times $t_1, ..., t_n$, and all lags $l = 0, -1, -2, ...$

If we assume that the process has a first and second moments then the definition 1 tells us, that the process must have $E(Y_t) = \mu$ constant for all t, where $\mu = (\mu_1, ..., \mu_d)'$ is the mean vector of the time series. Also the process must have a constant covariance matrix,

$$\Sigma(0) = E[(Y_t - \mu)(Y_t - \mu)].$$

In addition, if a process $\{Y_t\}$ is stationary than the cross covariance between two components of this time series y_t^i and y_{t+l}^j must depend only on the lag l and not on time t. Essentially, this means that the generating function for the series does not itself change through time. If we assume for example that y_t was generated by equation,

$$y_t = \epsilon_t + \alpha(t)\epsilon_{t-3},$$

and the parameter $\alpha(t)$ is a function of the time, then the series would not be stationary. There are many time series from real world they are non-stationary, but if we have a time series with trend in mean, we can find a suitable trend curve , or we can remove this trend by differencing this time series. Then this particular form of non-stationarity can be removed. Another form of non-stationary time series are they with a trend in variance. Such series can often be made stationary, by considering the logarithms of the series values, which transfer the trend in variance to one in mean.

The assumption of stationarity, or of achieved stationarity, is very important in linear modeling of time series. The following theorem show this importance,

Theorem A.1 (Wold's Theorem). *Any stationary time series can be considered to be the sum of two parts, a self deterministic component and a moving average component of possibly infinite order.*

It can be shown that all moving average models of finite order are stationary. However, there is an alternative condition that has to apply to a moving average model to make it useful for modeling and forecasting processes, known as invertibility. If we consider an MA(1) model $y_t = \epsilon_t + b\epsilon_{t-1}$. Then for ϵ_t we can consider the difference equation,

$$\epsilon_t = y_t - by_{t-1} + b^2y_{t-2} - b^3y_{t-3} + ... + (-b)^k y_{t-k} + ... \ , \qquad \text{(A.1)}$$

It is clear that if b is larger than, or equal to unity in magnitude, the weights given to more distant y_t values will increase in size, meaning that the current ϵ_t cannot be estimated from past y_t. If a solution with nonexplosive feature is found that (A.1) solves, then the moving average is said to be invertible. In other words the condition is just that the estimated weight b must be less than one.

A further important class of linear models is a autoregressive process. An example of an AR(2) process is

$$y_t = a_1y_{t-1} + a_2y_{t-2} + \epsilon_t,$$

where again ϵ_t is i.i.d. and has zero mean. A wider class are autoregressive moving average models, an example being

$$y_t = a_1y_{t-1} + a_2y_{t-2} + \epsilon_t + b_1\epsilon_{t-1} + b_2\epsilon_{t-2},$$

which is denoted as ARMA(2,2).

In same cases we observe the long memory property of a linear models, so that $E[y_{t+h}|y_{t-j},\ j \geq 0]$ does not necessarily tend to zero as h becomes large. If this quantity does tend to zero, it is called short memory. A necessary condition to ensure that is $cov(y_{t+h}, y_t) \longrightarrow 0$ as $h \longrightarrow \infty$. An example of a long memory time series is one whose differences are stationary. Such a series is called ARIMA, I stands for integrated. If s differences are required to obtain a stationarity the series are called integrated of order s ARIMA(p,s,q).

The ARIMA models here are univariate. We can of course explain it in two ways. We can model a multivariate time series with for example AR representation

$$Y_t - \sum_{j=1}^{p} \Phi_j Y_{t-j} = \epsilon_t - \sum_{j=1}^{q} \Theta_j \epsilon_{t-j},$$

where Y_t has d components, each Θ_j and each Φ_j is an $(d \times d)$ matrix and ϵ_t is a vector i.i.d. input. Another possibility is to make a partial multivariate modeling, the so called ARMAX modeling. An example of a ARMAX representation is given

$$y_t^i = \sum_{j=1}^{p} a_j y_{t-j}^i + \sum_{j=1}^{r} B_j Y_{t-j}^{(j \in G)} + \sum_{j=1}^{q} c_j \epsilon_{t-j} + \epsilon_t,$$

where $G \subset \{1, ..., d\}$ with $i \notin G$. X is in ARMAX standing for "exogenous".

A.1.2 Order determination of an ARMA process with selection criteria

Here we consider a general model selection criteria for AR and ARMA processes. Various model selection criteria such as AIC, BIC, and FPE could be used to aid in the most appropriate choice of the model.

AIC model selection criterion

$$AIC_r = \log(|\tilde{\Sigma}_r|) + \frac{2r}{T} + C, \tag{A.2}$$

where T is a normalization term, r is the number of parameters estimated by the maximum likelihood in the vector ARMA model and $\tilde{\Sigma}_r$ is the corresponding ML residual covariance matrix estimate of Σ (Akaike, 1976).

The BIC model selection criterion is

$$BIC_r = \log(|\tilde{\Sigma}_r|) + \frac{r \log(T)}{T} + C, \tag{A.3}$$

and hence, BIC_r imposes a greater "penalty factor" for the number of estimated model parameters then does AIC_r.

The FPE ("Final Prediction Error") criterion, suggested by Akaike(1971) for selection of vector $AR(m)$ models, is

$$FPE_m = det\left\{\left[1 + \frac{mk}{T}\right]\hat{\Sigma}_m\right\},$$

where

$$\hat{\Sigma}_m = \left(\frac{T}{T - mk}\right)\tilde{\Sigma}_m$$

is the estimate of Σ adjusted for degrees of freedom. The FPE criterion is based on the result that an approximate covariance matrix of one-step ahead forecast errors when forecasting from AR(m) model with parameters that have been estimated by least squares or maximum likelihood methods is

$$\left[1 + \frac{mk}{T}\right]\Sigma.$$

The model selection criteria is used to compare various models fitted by maximum likelihood to the series such that the fitted model that yields a minimum value of a given criterion is chosen.

Quinn(1980) showed an asymptotic result under a true stationary AR(p) model. He proved that the order m that minimizes the order selection criterion of the form

$$IC_m = \log(|\tilde{\Sigma}_m|) + \frac{2mC_T \log(\log(T))}{T}, \quad C_T > 1,$$

where $\tilde{\Sigma}_m$ is the maximum likelihood residual covariance matrix estimate from fitting the m-th order AR model, is strongly consistent for the true AR order p as $T \longrightarrow \infty$ if and only if $\limsup C_T > 1$.

For the nonstationary vector AR models, Paulsen(1984) established weak consistency for the AR order m, selected by criteria of the

form

$$IC_m = \log(|\tilde{\Sigma}_m|) + \frac{mK_T}{T}, \tag{A.4}$$

if and only if $K_T \longrightarrow +\infty$ and $\frac{K_T}{T} \longrightarrow 0$ as $T \longrightarrow \infty$.

Koreisha and Pukkila(1993) suggested another way for order determination with information criteria. To determine the order of a vector AR model. This model was estimated for several orders $m = 1, 2, ...$ using the least squares estimation procedure, and the residuals $\tilde{\epsilon}_t(m)$ from the fitted AR model of order m are obtained. Then the procedure of order selection criterion was applied to the residual series, and if the order selection criterion leads to the selection of an AR model of order greater than zero for the residual series $\tilde{\epsilon}_t(m)$, then the residuals are viewed as not a white noise. Hence, an AR model of order m for the original series is rejected. The smallest AR order m for which the corresponding residual series $\tilde{\epsilon}_t(m)$ can be viewed as a white noise series is accepted, and this m is then selected as the appropriate AR order model for the original series.

As an alternative to the direct estimation by maximum likelihood of various models, in more complex modeling cases, initial model specification techniques that involve canonical correlations and other related techniques, such as those discussed by Akaike(1976), Cooper and Wood(1982), Tiao and Tsay(1989) and others may be considered.

A.1.3 Nonlinear models

In this subsection we introduce a few simple univariate nonlinear models. An important simple model is the nonlinear autoregressive of order one, where:

$$y_t = f(y_{t-1}) + \epsilon_t, \tag{A.5}$$

where ϵ_t is i.i $N(0,\sigma^2)$ distributed, is denoted as NLAR(1). We know that iff the conditional distribution of y_{t+1} given all $y_{t-j}, \quad j > 0$ is the same as the conditional distribution of y_{t+1} given just y_t we have a Markov process property. Clearly an NLAR(1) process is Markov. If we assume that the regression function f is known and a set of past time series data $y_1, ..., y_t$ are available we can give a one-step least squares predictor of y_{t+1}:

$$\begin{aligned} \hat{y}_{t+1} &= E(y_{t+1}|y_t, ..., y_1) \\ &= E\{f(y_t) + \epsilon_{t+1}|y_t\} = f(y_t). \end{aligned} \tag{A.6}$$

Under model (A.5) the calculation of $\hat{y}_{t+1}$ is easy and independent of the distribution of ϵ_{t+1}. This is an important property of an one step prediction for both linear and non-linear AR models. For multistep predictions that is true only for linear models. Indeed, if f is linear, say $f(y) = ay + b$, then from (A.6) we have

$$\begin{aligned} \hat{y}_{t+2} &= \int \Big[a(ay_t + b + \epsilon) + b\Big] dF(\epsilon) \\ &= (a^2 y_t + ab + b) + a \int \epsilon dF(\epsilon) \\ &= (a^2 y_t + ab + b) = a\hat{y}_{t+1} + b. \end{aligned}$$

For non-linear models the situation becomes more complicated. For example the two step least squares predictor for model (A.5) is

$$\begin{aligned} \hat{y}_{t+2} &= E(y_{t+2}|y_t, ..., y_1) \\ &= E\{f(y_{t+1}) + \epsilon_{t+2}|y_t\} \\ &= E\{f(f(y_t) + \epsilon_{t+1})|y_t\} \\ &= \int f(f(y_t) + \epsilon) dF(\epsilon), \end{aligned} \tag{A.7}$$

where $F(\cdot)$ is the distribution function of ϵ_t. We see that the right hand side of (A.7) depends on F, and can not be further reduced as

in case with a linear model. Indeed linear models are the only models with the property that the multistep prediction is independent of the innovation distribution. If we assume that the non-linear regression function f is known and a set of historical records are available, then we can compute a multistep predictor if we have an information about distribution function of innovations. Note that we can calculate innovations $\{\epsilon_t\}$ exactly from the data

$$\epsilon_k = y_k - f(y_{k-1}), \quad k = 2, ..., t.$$

Then, the empirical distribution F_n of the innovations $\epsilon_1, ..., \epsilon_t$ can be taken as estimate of the innovation distribution, where

$$F_n(y) = \frac{1}{t-1} \sum_{k=2}^{t} I(\epsilon_k < y),$$

and $I(\epsilon_k < y)$ denotes the indicator function of the set $(\epsilon_k < Y)$. We have

$$\hat{y}^*_{t+2} = \frac{1}{t-1} \sum_{k=2}^{t} f(f(y_t) + \epsilon_k), \tag{A.8}$$

as our two-step predictor. The extension of the prediction procedure (A.8) to exact l-step least squares predictor is,

$$\hat{y}_{t+l} = \int \quad ... \quad \int f(f(...(f(y_t) + \epsilon_1) + ...) + \\ +...\epsilon_{l-1}) dF(\epsilon_1)...dF(\epsilon_{l-1}), \tag{A.9}$$

and our proposed predictor is

$$\hat{y}^*_{t+l} = \frac{(t-l)!}{(t-l)!} \sum_{(l-1,t)} f(f(...(f(y_t) + \epsilon_{i_1}) + ...) + \epsilon_{i_{l-1}}), \tag{A.10}$$

where the summation runs over all possible $(l-1)$-tuples of distinct $i_1, ..., i_{l-1}$.

An obvious generalization of NLAR(1) to p lags is the NLAR(p) model

$$y_t = f(y_{t-j},\ \ j = 1, ..., p) + \epsilon_t,$$

which is not Markov if $p > 1$. Various forms of the NLAR(p) models were described by Tong(1990).

Historically, an important step beyond linear models for representation time series was taken in 1980 by Tong&Lim. After more than five decades of approximating a process with one globally linear function i.e. one hyperplane, they suggested the use of two hyperplanes. This model is globally nonlinear: it consists of choosing one of two local linear or nonlinear autoregressive models based on the value of process state. We can give an example of this, a so called threshold nonlinear autoregressive type 1 models TNAR1:

$$\left.\begin{array}{rcl} y_t = f(y_{t-1}, \theta_1) + \epsilon_t & : & y_{t-2} > 0 \\ = f(y_{t-1}, \theta_2) + \epsilon_t & : & y_{t-2} \leq 0 \end{array}\right\},$$

where θ is a vector of estimated parameters and $\theta_1 \neq \theta_2$. A further model is the so called threshold nonlinear autoregressive models from type 2 TNAR2:

$$\left.\begin{array}{rcl} y_t = f_1(y_{t-1}, \theta_1) + \epsilon_t & : & y_{t-2} > 0 \\ = f_2(y_{t-1}, \theta_2) + \epsilon_t & : & y_{t-2} \leq 0 \end{array}\right\},$$

with the property that the functions f_1 and f_2 are from different classes. There are clearly more possible generalizations, with several switching regimes, more lags, and the switching rule can also depend on other variables. A further specific NLAR model that have received some attention is the exponential AR model, such as

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \epsilon_t,$$

where

$$a_j = \phi_j + \pi_j e^{(-\alpha y_{t-1}^2)}, \quad j = 1, 2, ...,$$

see Ozaki(1985). It can be shown that if:

$$\frac{1 - \phi_1 - \phi_2}{\pi_1 + \pi_2},$$

is either negative or bigger then one, then the process will include limit cycles, so that the series includes a periodic component for t large.

Nonlinear moving average models can take as in the linear case many forms such as:

$$y_t = \epsilon_t + h_1(\epsilon_{t-1}) + h_2(\epsilon_{t-2}) + ... \ .$$

A simple version can be represented as follows:

$$y_t = \epsilon_t + b\epsilon_{t-1}^2,$$

which is called NLMA(1). A further class intermediate between NLAR and NLMA as the so called bilinear models:

$$y_t = ay_{t-1} + \epsilon_t + b\epsilon_{t-2}y_{t-2}.$$

The models considered so far except TNAR1 and TNAR2 models all had constant parameters, but it is easy to believe in time varying parameters. If the reasons for this change are unobserved, a class of models of the form:

$$\begin{aligned} y_t &= \beta_t x_{t-1} + \epsilon_t, \\ \beta_t &= m + \alpha\beta_{t-1} + \nu_t, \end{aligned}$$

was suggested, called time varying parameter autoregressive models, VPAR(1) in this case. β_t is unobserved but can be estimated iteratively

using the Kalman filter, see Granger/Newbold(1986), Harvey(1981). However, here is a problem: β_t is of course estimated from the data, and the model can equally well be thought of as being nonlinear rather then time varying parameter. It is very difficult to distinguish between these two kinds of models.

If the reason for the changing parameters is some observed variable z_t, one gets a class of models known as doubly stochastic, discussed by Tjostheim(1986). An example is

$$y_t = \beta x_{t-1} z_t + \epsilon_t,$$

where both x_t and z_t are observed series.

Appendix B

Neural Networks

B.1 Neural network modeling

In the competition, the majority of contributions, and also the best predictions for each set used neural networks, also called connectionist models. They provide a convenient language game for nonlinear modeling. They are typically used in pattern recognition, where a collection of features is presented to the network, and the task is to assign the input to one or more classes. Lippmann (1987) asserts intuitive that arbitrary complex decision regions, including concave regions, can be formed using four-layer neural networks. Huang and Lippmann (1987) demonstrated by simulations the ability of three-layer networks to form several complex decision regions in pattern recognition application.

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 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 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 performed weather forecasting. In the post-back-propagation era, Lapedes and Farber(1987) trained their neural network to emulate relationship between output(the next point in the series) and inputs (its predecessors) for computer generated time series. Weigend et.al.(1990,1992) addressed the issue of finding networks of appropriate complexity for predicting observed (real-world) time series.

A network that is to predict the future must know about the past. The simplest approach is to provide time delayed samples to its input layer. A network without nonlinear hidden units is equivalent to one linear filter (i.e. an autoregressive model). However, a network with nonlinear hidden units can be viewed as combining a number of squashed filters.

Example

Figure B.1 shows a neural network with one hidden layer, two hidden units and a one dimensional output For a given input sequence

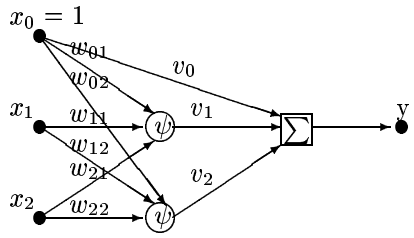

Figure B.1: Full connected neural network.

x we can compute the feed forward neural network output as follows:

$$f_{nn}(x,\theta) = v_0 + \sum_{h=1}^{H} v_h \psi\big(\tilde{x}' \cdot w_h\big), \tag{B.1}$$

where $\theta = (w_1', ..., w_h', v_0, ..., v_h)'$ is the vector of network weights, with $w_h' = (w_{0h}, ..., w_{dh})$, $h = 1, ..., H$, ψ is the activation function from each hidden unit and $\tilde{x}' = (1, y_{\cdot}^1, ..., y_{\cdot}^d)'$ is the vector of inputs.

More about the architecture of the neural networks will be discussed in the next sections.

Neural networks typically exhibit two types of behaviour. If no feedback loops connect neurons, the signal produced by an external input moves in only one direction and the output of the network is just the output of the last layer of neurons in the network. In this case neural network behaves mathematically like a nonlinear function of the inputs. This feedforward type of networks is most often used in time series forecasting, with past time series values as the input and desired future values as the output. The second kind of network behaviour is observed when there are feedback loops in the neuron connections. In this case the network behaves like a dynamical system, so the outputs of the neurons vary with time. The neuron outputs can than oscillate, or settle down into steady state.

In practice the available time series data will be subdivided into the training and test data sets. The data from the training data set are used for the training of the neural network. This network will be checked with test data set to determine whether the mapping performed by the network is a good representation of the time series and can therefore be expected to make reasonable predictions. To test it for short term prediction accuracy, the network is presented the actual time series values from the test data set as its input, and the resulting output will be compared with the next(corresponding) observation of a time series. This is done for every point in test data set and then

is the mean squared error calculated. This error represents the short term prediction ability of this neural network for this time series.

The ability of a neural network to make long term predictions can be tested too. For this reason an input vector from near the beginning the test data set will be presented to the network. The output of the network which is the predicted future time series value, is used as a part of next input vector. The output of second prediction is in the same way used as a part of third input vector. In this way the network recursively propagates the time series forward in time to make a prediction many time steps ahead.

B.2 Some main theorems

Let points of d-dimensional Euclidean space R^d be denoted by $y = (y^1, ..., y^d)$ Let $I = [0, 1]$ denote the closed unit interval, $I^d = [0, 1]^d$ $(d \geq 2)$ the Cartesian product of I. Kolmogorov(1957) proved following theorem.

Theorem B.1 (Kolmogorov). *Any continuous functions $f(y^1, ..., y^d)$ of several variables defined on $I^n = [0, 1]^n$ $(n \geq 2)$ can be represented in the form*

$$f(y) = \sum_{j=1}^{2d+1} \chi_j \left(\sum_{i=1}^{d} \psi_{ij}(y^i) \right),$$

where χ_j, ψ_{ij} are continuous functions of one variable and ψ_{ij} are monotone functions which are independent from f.

Sprecher (1965) after his refining of the theorem from Kolmogorov obtain following theorem,

Theorem B.2. *For each integer $d \geq 2$, there exists a real, monotone increasing function $\psi(y)$, $\psi([0, 1]) = [0, 1]$, dependent on d and having*

the property: For each preassigned number $\delta > 0$ there is a rational number ϵ, $0 < \epsilon < \delta$, such that every real continuous function of n variables, $f(y)$, defined on I^d, can be represented as

$$f(y) = \sum_{j=1}^{2d+1} \chi \left[\sum_{i=1}^{d} \left(\lambda^i \psi(y^i + \epsilon(j-1)) \right) + j - 1 \right],$$

where the function χ is real and continuous and λ is an independent constant of f.

A further theorem of this kind is the theorem presented in the paper from Funahashi(1989). Let the norm of y^i be defined by,

$$|y| = \left(\sum_{i=0}^{d} (y^i)^2 \right)^{1/2}.$$

Theorem B.3. *Funahashi(1989). Let $\phi(y)$ be a nonconstant, bounded and monotone increasing continuous function. Let K be a compact subset(bounded closed subset) of R^d and $f(y^1, ..., y^d)$ be a real valued continuous function on K. Then for an arbitrary $\epsilon > 0$, there exists an integer N and real constants $c_i, \theta_i (i = 1, ..., N)$, $w_{ij} (i = 1, ..., N,\ j = 1, ..., d)$ such that*

$$\tilde{f}(y^1, ..., y^d) = \sum_{i=1}^{N} c_i \phi \left(\sum_{j=1}^{d} w_{ij} y^j - \theta_i \right),$$

satisfies

$$\max_{y \in K} |f(y^1, ..., y^d) - \tilde{f}(y^1, ..., y^d)| < \epsilon.$$

In other words, there exists a three-layer network whose output functions for the hidden layer are $\phi(y)$, whose output functions for input and output layers are linear, and which has an input-output

function $\tilde{f}(y^1, ..., y^d)$ such that,

$$\max_{y \in K} |f(y^1, ..., y^d) - \tilde{f}(y^1, ..., y^d)| < \epsilon.$$

To prove this theorem Funahashi (1989) used following central lemma,

Lemma B.4. *Let $A_i > 0$, $(i = 1, ..., m)$, K be a compact subset of R^d and $g(y^1, ..., y^m, t_1, ..., t_d)$ be a continuous function on $[-A_1, A_1] \times ... \times [-A_m, A_m] \times K$. Then the function defined by the integral*

$$G(t) = \int_{-A_1}^{A_1} ... \int_{-A_m}^{A_m} g(y^1, ..., y^m, t_1, ..., t_d) dy^1 ... dy^m,$$

can be approximated uniformly on K by the Riemann sum

$$G_N(t) = \frac{2A_1 \cdots 2A_m}{N^m} \times \sum_{k_1 \cdots k_m}^{N-1} g\left(-A_1 + \frac{k_1 \cdot 2A_1}{N}, ..., -A_m + \frac{k_m \cdot 2A_m}{N}, t_1, ..., t_d\right).$$

In other words, for an arbitrary $\epsilon > 0$, there exists a natural number N_0 such that for $N \geq N_0$,

$$\max_{t \in K} |G(t) - G_N(t)| < \epsilon.$$

(without proof), The above theorem leads to a general theorem,

Theorem B.5. *Let $\phi(y)$ be a nonconstant, bounded and monotone increasing continuous function. Let K be a compact subset of R^d and fix an integer $k \geq 1$. Then any continuous mapping $f : K \longrightarrow R^a$ defined by $y = (y^1, .., y^d) \longrightarrow (f_1(y), ..., f_a(y))$ can be approximated in the sense of uniform topology on K by input-output mappings of k-hidden layer networks whose output functions for hidden layers are $\phi(y)$, for input and output layers are linear.*

Definition B.1. Let S be a function space. ρ is a mapping with the properties

1. For all $f, g \in S$, $\rho(f, g) \geq 0$,
2. For $f, g, h \in S$ $\rho(f, h) \leq \rho(f, g) + \rho(g, h)$.
3. $\rho(f, g) = 0$ if and only if $f = g$.

The mapping ρ is called a metric and the pair (S, ρ) is called a metric space.

In other words, for any continuous mapping $f : K \longrightarrow R^m$ and an arbitrary $\epsilon > 0$, there exists a k-layer network whose input-output mapping is given by $\tilde{f} : K \longrightarrow R^m$ such that,

$$\max_{y \in K} \rho(f(y), \tilde{f}(y)) < \epsilon.$$

In paper from Hornik et. al. (1990) to describe the ability of neural networks to approximate the space S, the concept of ρ-density was applied.

Definition B.2. Let U be a subset of R^d, let S be a collection of functions $f : U \longrightarrow R$ and let ρ be a metric on S. For any mapping $g : R^d \longrightarrow R$, with $g \in \Sigma(\phi^H)$ define the restriction of g to U, $g_{|U}$, as $g_{|U}(y) = g(y)$ for $y \in U$, and $g_{|U}$ unspecified for $y \notin U$. Suppose that for any $f \in S$ and $\epsilon > 0$ there exists $g \in \Sigma(\phi^H)$ such that $\rho(f, g_{|U}) < \epsilon$. Than we say that $\Sigma(\phi^H)$ contains a subset ρ-dense in S. If in addition $g_{|U} \in S$ for every $g \in \Sigma(\phi^H)$, we say that $\Sigma(\phi^H)$ is ρ-dense in S. Where $\Sigma(\phi^H)$ is the collection of network output functions that depends on the number of neurons in the hidden layer H.

The first part of this definition allows, that $g \in \Sigma(\phi^H)$ and $g_{|U} \notin S$, if $\Sigma(\phi^H)$ has this denseness property, it always contains a single hidden layer feedforward neural network output function with

ability of arbitrary accuracy approximation to any member of S in terms of the metric ρ.

The usual neural network function, as the sigmoid function $\frac{1}{1+e^{-x}}$ used in gradient descent algorithms, satisfies the condition of $\phi(x)$, that $\phi(x)$ is a nonconstant, bounded and monotone increasing continuous function, but the problem is in the chois of an integer H from for example theorem 1.6. If we choose a neural network with a very high number of neurons in the hidden layer, then the function space that can be approximated with this neural network will be large, in other words we have, with respect to theorem 1.6

$$Pr\left(f \in \Sigma(\phi^H)\right) \stackrel{H\to\infty}{\longrightarrow} 1.$$

That means that we can be "sure", that the true function f can be approximated by the neural network with H neurons when H is high.

Anther important feature of multilayer neural networks is the ability to approximate beside weight matrices, the corresponding derivatives. Hornik et. al. (1990) discuss that such a set of weights does indeed exist, and multilayer feedforward neural networks with at least one hidden layer and sigmoid activation functions are in fact capable of arbitrary accurate approximation to unknown mapping and simultaneously to its derivatives. Application areas and the importance of derivatives in modeling multivariate time series will be discussed in the next sections.

Now we have found an instrument to model any existent relationship between the past observations of the components in a multivariate time series and interesting component from the same time series with any required precision.

B.3 Pattern recognition with NN

We begin by supposing that we wish to classify a new identified state of a component, but as yet we have no entries in CSL like that. The goal is to classify the new state in the CSL in such a way that the probability of misclassification is minimal. If we have a long range experience, that means that the CSL is not "new". Than we can compute the prior probabilities $P(S_c^i)$. For all states they are stored in CSL. Now as assumed we have estimated a new state. This give us further information on which to base the classification decision, and we seek a formalism which us allows to combine this information with the knowledge of prior probabilities. The joint probability $P(C_k, S_c^i)$ is defined as the probability that new state vector has estimated entries S_c^i, and belongs to the class C_k. A further definition is the conditional probability: $P(S_c^i|C_k)$ is the probability that if the new state is from class C_k, then the state vector has the entries S_c^i. We now note that the fraction of the total number of patterns, which falls into the cell (C_k, S_c^i), is given by the fraction of the number of patterns in row C_k which fall in cell (C_k, S_c^i) times the fraction of the total number of patterns which fall in row C_k. This is equivalent to writing the joint probability in the following form,

$$P(C_k, S_c^i) = P(C_k)P(S_c^i|C_k), \tag{B.2}$$

by a similar argument, we can see that the joint probability $P(C_k, S_c^i)$ also can be written in the form

$$P(C_k, S_c^i) = P(S_c^i)P(C_k|S_c^i), \tag{B.3}$$

where $P(C_k|S_c^i)$ is the probability that the class is C_k if we have estimated the state vector S_c^i, and $P(S_c^i)$ is the probability of estimating this state vector, irrespective of class membership. Then we can write

the desired combination of given information, known as **Bayes' Theorem**, in a formal way, from (B.2) and (B.3) we have,

$$P(C_k|S_c^i) = \frac{P(S_c^i|C_k)P(C_k)}{P(S_c^i)}. \tag{B.4}$$

The quantity on the left hand side in (B.4) is called the posterior probability and the denominator $P(S_c^i)$ guarantees that the posterior probabilities sum to unity. Then if the new state was estimated, the probability of misclassification is minimized if we assign this state vector to the class C_k to which the posterior probability $P(C_k|S_c^i)$ is largest.

The typical use of neural networks is in pattern recognition. Outputs of a neural network can be interpreted as approximation to posterior probabilities. In such a task, a collection of features, like visual, semantic, numeric, and others is presented to the network and the network must classify the input feature pattern as belonging to one or more classes. For example, a network might be trained to classify a two dimensional pixel array as a letter in alphabet, or to classify the scorpions in male and female classes with length of species as a input sequence. These are examples of structural pattern recognition problems.

To show the ability of neural networks with sigmoid activation functions for pattern recognition, we consider the use of a non-linear function $\phi(\cdot)$ which acts to linear sum to give a discriminant function for the two class problem of the form

$$C = \phi(W^T S_c^i + b_0), \tag{B.5}$$

where $\phi(\cdot)$ is the neural network activation function, with Gauss distributed class conditional densities, with equal covariance matrices, $\Sigma_1 = \Sigma_2 = \Sigma$,

$$p(S_c^i|C_k) = \frac{1}{(2\pi)^d\sqrt{|\Sigma|}} \exp\left\{-\frac{1}{2}(S_c^i - \mu_k^i)^T\Sigma^{-1}(S_c^i - \mu_k^i)\right\}. \tag{B.6}$$

The denominator in the Bayes formula (B.4) can be expressed in form of prior probabilities and the class conditional probabilities. In the case of two classes we have,

$$P(C_1|S_c^i) + P(C_2|S_c^i) = 1. \tag{B.7}$$

Substituting (B.4) in (B.7) we obtain,

$$P(S_c^i) = P(S_c^i|C_1)P(C_1) + P(S_c^i|C_2)P(C_2). \tag{B.8}$$

Now we can write the Bayes formula in the following form, where we take in place of $P(S_c^i|C_k)$ the conditional densities $p(S_c^i|C_k)$ and write,

$$P(C_1|S_c^i) = \frac{p(S_c^i|C_1)P(C_1)}{p(S_c^i|C_1)P(C_1) + p(S_c^i|C_2)P(C_2)}. \tag{B.9}$$

We know that the sigmoid function ϕ has the form

$$\phi(x) = \frac{1}{1 + \exp(x)},$$

and we want that the Bayes formula has the same form, to achieve that we divide in (B.9) the right hand side with $p(S_c^i|C_1)P(C_1)$ and after simple arithmetical transformations we obtain that,

$$P(C_1|S_c^i) = \frac{1}{1 + \exp(A)}, \tag{B.10}$$

where

$$A = \ln \frac{p(S_c^i|C_2)P(C_2)}{p(S_c^i|C_1)P(C_1)}, \tag{B.11}$$

and finally substituting (B.6) in (B.11) we obtain,

$$A = W^T S_c^i + b_0, \tag{B.12}$$

where

$$\begin{aligned} W &= \Sigma^{-1}(\mu_2^i - \mu_1^i), \\ b_0 &= -\frac{1}{2}(\mu_2^i)^T \Sigma^{-1}(\mu_2^i) + \frac{1}{2}(\mu_1^i)^T \Sigma^{-1}(\mu_1^i) + \ln \frac{P(C_2)}{P(C_1)}. \end{aligned}$$

These computations show us that the use of sigmoid activation functions in neural networks allows the outputs of the network to be interpreted as posterior probabilities. This implies that such a network provide simply a classification decision.

We distinguish between two separate stages in the classification process.

- Inference,
- Decision making.

Inference is the stage in that we determine the values for the posterior probabilities (we train a network and compute the optimal weight matrices), and use them for making a decision in the second stage.

B.4 Neural network pruning techniques

In this section we discuss the case where the network has not a specific architecture, (we assume only that the number of units in hidden layer is equal to the number of components of a multivariate time series) i.e. is full connected and analyze the minimization procedure of mean squared error. Finally we present pruning algorithms and show the behaviour of "not horizontal" weights during the pruning procedure.

One of the difficulties with using feedforward neural networks is the need to determine the optimal number of hidden units before the training process can begin. Too many hidden units may lead to overfitting of the data and poor generalization in other words such large networks are able to fit the training data arbitrarily close but, in essence, they memorize the training patterns and typically do not generalize well to new inputs. Another motivation fact is that the amount of computation both for forward computation and for learning grows with increasing network size; too few hidden units may not give a

network that learns the data. The primary reason for the extensive interest in such network optimization algorithms is the fact that minimal nets tend to have the best generalization properties. However, the relation between the networks size and its ability to generalize is still plague to uncertainty. In a study of this relationship, Richards(1991) found examples in the literature in which larger networks showed better generalization then comparable smaller ones. Richards' own experimental results also show that networks with fewer hidden units do not necessarily generalize better. In the paper from Mozer and Smolensky reported results from several small experiments but they do not test the effect of their algorithm on generalization, Le Cun et. al. report only a slight increase in the recognition accuracy of their pruned network.

We have two different approaches to overcome the problem of determining the optimal number of hidden units required by an artificial neural network to solve a given problem. The first method begins with a minimal network and adds more hidden units only when they are needed to improve the learning capability of the network. The other begins with an oversized network and then prunes redundant hidden units(Skeletonization), or a connection(OBD-Optimal-Brain-Damage)(Le Cun et al.1990, Hassibi and Stork 1993, Thodberg 1991). This process is repeated until an optimal size is reached.

Pruning is based on a measure of importance. The least important node or connection is the one to be removed in each iteration. The basic method of determining this importance is very similar in almost all pruning algorithms. The total error at the output layer, summed over all training patterns, is calculated before and after removal of a particular hidden unit or connection. The hidden unit or connection that increases this error by the least amount is the least important. In this context speak Mozer and Smolensky of the relevance of a hidden unit. Le Cun et. al. call the some measure the saliency

of a connection. These pruning algorithms require the total change in error to be calculated for every hidden units or connection in the network. This calculation is expensive, especially in large networks. Many algorithms, therefore, base their decision on approximated values for hidden unit relevance or for connection saliency. For better visualization and better description of the neural network architecture from the previous section we will show a modificated(restricted) method for pruning neural network, so called "pruning orthogonally to learning". This Method without restricted properties was presented in paper from A. N. Burkitt and P. Ueberholz 1992.

B.4.1 Pruning orthogonally to learning

We begin with an oversized neural network and use the property of such a network: It exists a possibility to give a neural network with a smaller number of connections and some capacity of learning. With other words some value of the cost function can be obtained also with other networks of other connection(weight) architecture. Furthermore, for every successful value of the cost function, we have many weight combinations, i.e. neural nets, also we can define a contour of all weight combinations in a space of all weights for this value of the cost function. Since we also wish that our neural net to have good generalization properties, it is reasonable to seek a neural network configuration where most of the connection weights vanish. During the learning procedure the algorithm(usually back-propagation) moves from upper levels of contours of the cost function to the minima, i.e. with the property that in each contour the cost function is a constant. Each point at the contour is a different combination of the weights. They lead to some value of the cost function after the training of the neural network. The surface of the contour S_ζ^H for each value of cost function ζ and full connected neural network with H hidden units, has

following behaviour:

$$S_{\zeta}^{H} \overset{H\to\infty}{\longrightarrow} \infty.$$

Learning(training) proceeds with a minimization in the weight space of the cost function, usually mean squared error, and the network reduction proceeds by keeping this function E_{err} constant. For the learning with the back propagation algorithm is the gradient of the error function E_{err} in weight space i.e. with respect to each weight required, which is exactly orthogonal to the contour $E_{err} = \zeta$. The target of the pruning algorithms is to move around the contours as the cost function $E_{err} = \zeta$ is constant. In order to move around any contour $E_{err} = \zeta$ we must introduce the function E_{nr} for the network pruning(reduction) algorithm. We represent the network configuration in a weight vector $\vec{w}$, where $\vec{w} \in R^{H(d+1)}$ and where H is the number of hidden units and d is the dimension of the input vectors. At this point we restrict the network variability in the sense that all weights from hidden to output layer are important(significant) and cannot be pruned. Than we represent our neural network from the vectors $\vec{w} \in R^{Hd}$, which have lower dimension. We define the gradient of E_{err} by

$$\vec{\nabla}_{err} = -\frac{\partial E_{err}}{\partial \vec{w}},$$

and similarily the gradient of the network reduction term by

$$\vec{\nabla}_{nr}^{1} = -\frac{\partial E_{nr}}{\partial \vec{w}}.$$

In order to ensure that we remain on a contour of constant E_{err}, it is necessary to add a correction term to $\vec{\nabla}_{err}^{1}$ of the following form:

$$\vec{\nabla}_{nr} = \vec{\nabla}_{nr}^{1} - \beta \vec{\nabla}_{err}, \tag{B.13}$$

where β is given by

$$\beta = \frac{\vec{\nabla}_{nr}^{1'} \vec{\nabla}_{err}}{\vec{\nabla}_{err}' \vec{\nabla}_{err}}. \tag{B.14}$$

By moving through the weight space in the direction of $\vec{\nabla}_{nr}$ we generate a series of configurations with the same E_{err} value. The purpose of the function E_{nr} is to reduce a number of weights and hidden units in the network. To implement this we can define this function in several forms. They are discussed in the literature, for example:

$$\begin{aligned} E_{nr}^{(1)} &= \mu_w \sum_{weights} |w_{ij}|, \qquad E_{nr}^{(2)} = \frac{w_{ij}^2}{(1+w_{ij}^2)}, \\ E_{nr}^{(3)} &= 1 - \exp(-w_{ij}^2), \qquad E_{nr}^{(4)} = 1 - \exp(-|w_{ij}|). \end{aligned}$$

By an appropriate choice of the function E_{nr} it is possible to reduce redundant weights towards zero. The algorithm begins with an oversized random initial network architecture and implements a numerical algorithm like back propagation including contributions from both $\vec{\nabla}_{err}$ and $\vec{\nabla}_{nr}$. The correction term in (B.13) is only necessary if β in (B.14) is negative. If β is positive, then the contribution from E_{nr} is not in conflict with learning and we need no correction, the correction guarantee that the E_{nr} contribution stays on a contour of E_{err}, whereas if β is positive and no correction is carried out, the network configuration moves to a contour with smaller E_{err}, only if β is negative, which means that network reduction conflicts with learning, is correction carried out.

List of Figures

List of Tables

Bibliography

[1] U. Anders.
Statistische neuronale Netze.
Vahlen Verlag München 1997.

[2] Coryn A. L. Bailer-Jones, David J. C. MacKay.
A Recurrent Neural Network for Modelling Dynamical Systems.
Computation in Neural Systems No. 9, 531-547, 1998.

[3] Peter J. Bickel, David A. Freedman.
Some asymptotic theory for the bootstrap.
The Annals of Statistics Vol. 9 No. 6, 1196-1217, 1981.

[4] Christopher M. Bishop.
Neural Networks for Pattern Recognition.
Clarendon press, Oxford, 1995.

[5] D. Bosq.
Nonparametric Statistics for Stochastic Processes.
Springer, 1996, Lecture Notes in Statistics 110.

[6] A. N. Burkitt, P. Ueberholz.
Refined Pruning Techniques for Feed-forward Neural Net-

works.
Complex Systems, Vol. 6, 1992.

[7] Chakraborty K, Mehrotra K, Mohan C. K, Ranka S.
Forecasting the behaviour of multivariate time series using neural networks.
Neural Networks, Vol. 5, 1992.

[8] P. Doukhan.
Mixing properties and examples. Lecture Notes in Statistics, 1985, Springer-Verlag, Heidelberg.

[9] B. Efron, R. Tibshirani.
Bootstrap methods for standard errors, confidence intervalls, and other measures of statistical accuracy.
Statistical Science Vol. 1 No 1, 54-77 (1986).

[10] Bradley Efron, Robert J. Tibshirani.
An introduction to the Bootstrap.
Monographs on statistics and Applied Probability 57 (1993).

[11] J. Franke, J. P. Kreiss, E. Mammen, M. H. Neumann.
Properties of the nonparametric autoregressive bootstrap.
Manuscript(2000).

[12] J. Franke, M, H. Neumann.
Bootstrapping neural networks.
Neural Computation 12, 1929-1949(2000)

[13] J. Franke.
Portfolio management and market risk quantification using neural networks.
Statistics and Finance: An Interface, W.S. Chan, W. K. Li and H. Tong eds., Imperial College Press, London.(2000)

[14] D. A. Freedman.
Bootstrapping regression models.
Annals of Statistics, Vol.9(1981).

[15] Ken-ichi Funahashi.
On the Approximate Realization of Continuous Mappings by Neural Networks.
Neural Networks, 2(1989).

[16] C. W. J. Granger, T Teräsvirta. *Modeling Nonlinear Economic Relationships.*
Advanced texts in Econometrics, 1.3(1993).

[17] Meihui Guo, Zhidong Bai, Hong Zhi.
Multi-Step prediction for Nonlinear Autoregressive Models Based On Empirical Distributions.
Statistica Sinica 9(1999).

[18] W. Härdle A. Bowman.
Bootstrapping in nonparametric regression: local adaptive smoothing and confidence bands.
Journal of American Statistical Association 83(1988)

[19] K. Hornik, M. Stinchcombe, H. White.
Multilayer feedforward networks are universal approximators.
Neural Networks 2(1989).

[20] K. Hornik, M. Stinchcombe, H. White.
Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks.
Neural Networks 3(1990).

[21] M. Imhoff, M. Bauer, U. Gather, D. Löhlein.
Time series analysis in intensive care medicine.
Applied Cardiopulmonary Pathophysiology, 6(1997).

[22] M. Lehtokangas, J. Saarinen, K. Kaski, P. Huuhtanen.
A Network of Autoregressive Processing Units for Time Series Modeling.
Applied Mathematics and Computation, 75, 1996.

[23] B. M. Pötscher, I. R. Prucha.
Dynamic Nonlinear Econometric Models, Asymptotic Theory.
Springer 1997.

[24] P. N. Refenes, A. D. Zapranis, J. Utans.
Neural model identification, variable selection and model adequacy.
Neural Networks in Financial Engineering, Proc. NnCM-1996.

[25] Gregory C. Reinsel.
Elements of Multivariate Time Series Analysis.
Springer 1993.

[26] St. Rüger, A. Ossen.
The metric structure of weight space.
Manusktipt, Technische Universität Berlin (1997).

[27] Thomas Schreiber.
A Voronoi Diagram Based Adaptive K-means-Type Clustering Algorithm for Multidimensional Weighted Data.
Univerität Kaiserslautern(1997).

[28] Tschernig R, Yang LJ.
Nonparametric lag selection for time series.
Journal of Time Series Analysis 21:(4) Jul 2000.

[29] Howell Tong.
Threshold Models in Non-Linear Time series Analysis.
Lecture Notes in Statistics, Volume 21(1983).

[30] Andrew Webb.
Statistical Pattern Recognition.
Arnold press, Oxford, 1999.

[31] A. S. Weigend, N. A. Gershenfeld.
Time Series Prediction, Forecasting the future and understanding the past.
A Proceedings volume of the NATO Advanced Research Workshop on Comparative Time Series Analysis held in the Santa Fe institute studies in the sciences of complexity, New Mexico, May 14-17, 1992.

[32] H. White.
Learning in artificial neural networks: A statistical perspective. Neural Computation 1(1989).

[33] H. White.
Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings.
Neural Networks 3(1990).

[34] Qiwei Yao, Howell Tong
On subset selection in non-parametric stochastic regression
Statistica Sinca 4(1994).

Akademisches Lebenslauf

- Geboren am 1. März 1971 in Tbilisi/Georgien.
- 1977-1988 1. Exp. Schule von Tbilisi/Georgien.
- 1988-1992 Studium von Ingenieur Ökonomie. Staatsuniversität Georgiens, Tbilisi/Georgien.
- 1993-1998 Universität Kaiserslautern, Studienfach "Wirtschaftsmathematik", Abschluss Diplom Math.-eoc.
- 1995-1998 Stipendiat der Friedrich-Ebert-Stiftung.
- 1998-1999 Wissenschaftlicher Mitarbeiter an der Universität Dortmund, Fachbereich Statistik.
- 1999-2001 Stipendiat des Graduiertenkollegs Technomathematik.
- 2002 Promotion, Universität Kaiserslautern Fachbereich Mathematik.