ningen Faculteit der Wiskunde en Natuurwetenschappen

Wiskunde en
Informatica

Estimating Quality of Segmentation for Multistationary Time Series Modeling

H. Punt

supervisors:
Dr. ir. J.A.G. Nijhuis
Drs. M. van Veelen

Oktober 2000

Technische Informatica

RuG

Abstract 1

1. Introduction 2
1.1. Background 2
1.2. The Problem of Estimating Segmentation Quality 4
1.3. Thesis Goal 7
1.4. Thesis Structure 8

2. Time Series Modeling 9
2.1. Time Series 9
2.2. Time Series Modeling 10
2.3. Time Series Models 11
2.4. Stationarity 14
2.5. Monolitic Models and Nonstationarity 16

3. Modular Time Series Modeling 17
3.1. A framework for Modular Time Series Models 17
3.2. Segmentation in Modular Time Series Models 18
3.3. The problem of Estimating Segmentation Quality 22

4. Measures for Segmentation Quality 23
4.1. Measuring Segmentation Quality in Modular Models 23
4.2. Measuring Stationarity 24
4.3. Testing for Stationarity 24
4.4. The Varratio Method for Measuring Stationarity 26
4.5. Summary 27

5. Quality of Unsupervised Segmentation 28
5.1. Unsupervised Segmentation 28
5.2. The SOLEX Model 28
5.3. Unsupervised Segmentation of Multistationary Time Series 31

6. Quality of Segmentation Measures 39
6.1. Evaluating Quality of Segmentation Measures 39
6.2. Correlation of Varratio with Local Model Performance 40
6.3. Stability of Varratio 45

7. Conclusions & Future Research 47

References 49

A. Multistationary Time Series 51

B. Statistics 53

C. SOFM Implementation 56

Abstract

The main problem associated with time series modeling is that of modeling non–stationary time series. Monolitic, global models assume stationarity, e.g. the properties of the time series do not change with time. These models fail to capture the non–stationary dynamics of many real–world time series.

One class of non–stationary time series are those that switch between multiple stationary regions at certain points in time. These multistationary time series series can be modeled by modular models which combine a segmentation of time series into stationary regions with a number of local models specialized in modeling these regions. The outputs of the local models are combined to form a single output.

Assuming good local models can be found for stationary local data and that they can be combined successfully, the remaining problem is segmenting the stationary regions. During the construction of a modular model the quality of segmentation must therefore be evaluated, e.g. the stationarity of local data must be quantified. This is a problem because time series modeling does not make assumptions about the properties of the stationary regions, and no non–parametric estimate for stationarity exists.

One solution is to assume that stationarity of local data is expressed by the the local models, estimating the quality of segmentation indirectly from their performance. This is the approach taken by modular models found in literature such as Gated Experts Networks. A number of problems are associated with this approach, caused by the interdependence between segmentation and local modeling.

The goal of this thesis is was to find out if stationarity of local data can be expressed without constructing local models, with the aim of breaking the interdependence between segmentation and modeling. Based on a hypothesis that stationarity may be expressed by means of a number of traditional statistics, measures were developed to estimate segmentation quality directly from the statistical properties of local data. The viability of these measures of segmentation quality was subsequently tested in a number of experiments.

Based on the results of these experiments, the main conclusion must be that it is not possible to generally estimate segmentation quality by means of estimating stationarity of local data using traditional statistics. The newly developed measures of segmentation quality can therefore not be used to break the interdependence between segmentation and local modeling in modular time series models.

1. Introduction

This thesis deals with the subject of *time series modeling*, more specifically it deals with some of the problems associated with the *construction* of these models.

The field of time series modeling deals with the construction of models for dynamical processes for which an adequate analytical description is not known, and hence analytical models cannot be build. Instead, time series models are build by considering the dynamic relations found in consecutive outcomes of such a process, e.g. a time series. This way an analytical description of the process is not necessary in order to build a model.

Applications of time series models are found in many areas of science & industry; For instance in the field of feedback control, models of the controlled plant are used to provide estimates in the case where continues measurements on the real plant would be either too expensive or too dangerous. In the field of economics, time series models are used to simulate the processes that govern economies. In this way insight is gained into the effects proposed changes would have on these economies. Time series models may also be used to forecast future observations on a given process. Numerous applications of time series forecasting exist; Stock market analysis, where predictions are made on future stock values and for instance weather and climate forecasting.

In order to properly state the thesis goal, a number of concepts & problems relating to time series modeling need to be considered first. The following section will provide only a minimal overview of time series modeling as needed to state the thesis goal. These concepts will be discussed in more detail in chapter 2.

1.1. Background

Time series modeling starts of with some dynamic process for which a model needs to be build:

Definition 1.1: Dynamic Process

> A process P is called dynamic if the outputs of the process at some time t not only depend on the inputs at time t but also on the history of previous inputs and outputs of the process.

In time series modeling the nature of this dynamic process P is *unknown* (Otherwise an analytical model could have been used). This means that in the construction of a time series model M for P only consecutive observations on the outputs $Y(t)$ of P are considered. A sequence of such observations on P is referred to as a *time series* and the task of constructing a model for P from these observations is referred to as *time series modeling*.

If a time series consists of a sequence of *single* observations y_t, it is called a *univariate* time series. When *multiple* simultaneous observations y_t are made, the sequence $y_1,\ldots,y_T$ is referred to as a *multivariate* time series. For reasons of simplicity, this thesis only considers modeling of univariate time series.

Time series modeling considers the dynamical relations found in a given time series in order to express these in a model. The model works by providing a mapping from a limited number of past observations $y_{t-1},y_{t-2},\ldots,y_{t-N}$ to a forecast of a future observation $\hat{y}_{t+\tau}$. The mapping

is associated with a number of *model parameters*, the values of which are estimated from the given time series using algorithmic *parameter estimation* methods with the aim of minimizing the models forecasting error.

Various types of models exist and a choice for a particular model will be based on assumptions about the nature of the dynamical relations in the time series. Traditional models such as ARMA[1] assume linear relations and consist of a single linear mapping from past observations to the model forecast. In recent literature non–linear models have been introduced capable of expressing non–linear dynamical relations.

The relation between dynamic process, time series and time series model is shown in Figure 1.1:

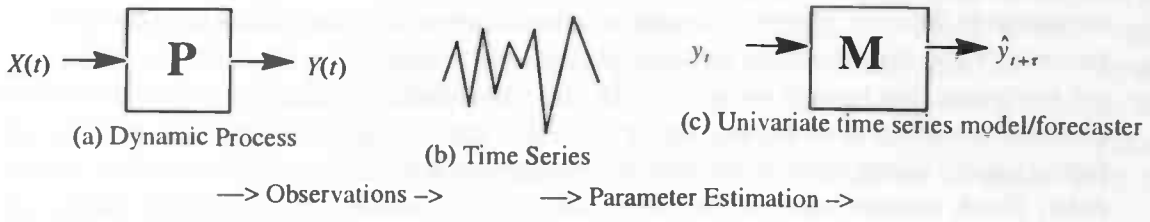

Figure 1.2 Relation between dynamic process, time series and time series model

An important property of time series that must be considered in time series modeling is that of stationarity. Multiple definitions of stationarity exist, but in the context of this thesis it will be defined as follows:

Definition 1.2: Stationarity

> A time series is said to be *stationary* if its properties do not change with time [1][9], and consequently is considered *non–stationary* if its properties *do* change with time.

In this thesis the focus will be on the problems associated with the modeling of *multistationary* time series. Multistationary time series are the class of non–stationary time series that switch between different *stationary regions* at certain points in time. Within one region, the properties do not change and the time series is stationary, but on a longer timescale the series is non–stationary as its properties change from one region to the other. Time series of this type are also referred to as being *piecewise stationary*. Examples of multistationary time series are given in Figure 1.3 and Figure 1.5, where the switching between two stationary regions is clearly visible.

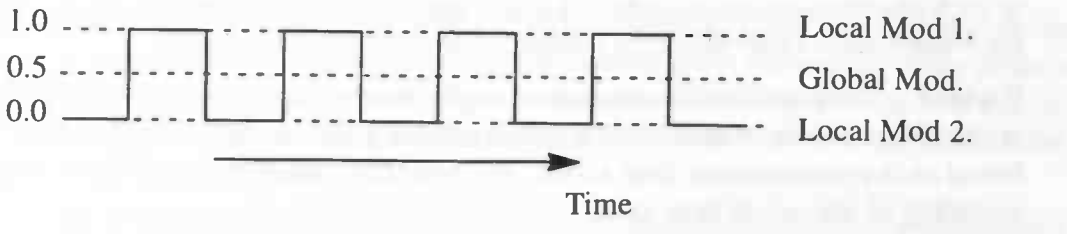

Figure 1.3 A Simple Multistationary Time Series

The main problem associated with multistationary time series is that they can not be effectively modeled by traditional monolithic global time series models. These models can only

be adapted to reproduce the average *global* dynamics of the multistationary time series. The monolithic model will therefore underperform in each of the *local* regions of the multistationary time series. The most simple example of this is shown in Figure 1.3. In this example a simple time series switches between two stationary regions with a different but constant amplitude. If a *mean* model (a model that predicts a constant value) is used to model this series, it can do no better than to predict the mean of the series (0.5 in this case).

1.2. The Problem of Estimating Segmentation Quality

The fact that a multistationary time series contains multiple stationary regions suggests the applicability of a divide–and–conquer strategy to the problem of modeling multistationary series (Figure 1.4). In recent literature this approach has been adopted in the form of *modular* models [10][11][12][13][14][15].

The idea behind this approach is to *split* or *segment* the multistationary series into its stationary regions (**divide**). Appropriate *local* models are then responsible for modeling the stationary dynamics in each individual region (**model**). Finally, the responses of the local models are combined to form a single *global* forecast (**conquer**). The local models only need to model stationary dynamics and it has already been shown that modular models perform better than a single global model [10][15]

This can also easily be seen in the example of Figure 1.3. If 2 mean models are used for this series, one local model could be responsible for predicting the regions with amplitude 1 and second local model for the regions with amplitude 0. Assuming that this series can be split perfectly, the combination of the 2 local models will be optimal.

Note that in modular modeling the following assumptions are made;

1. A good segmentation will lead to stationary local data.
2. A good local model can always be made for local stationary data.
3. The predictions of the local models can be combined in optimally.

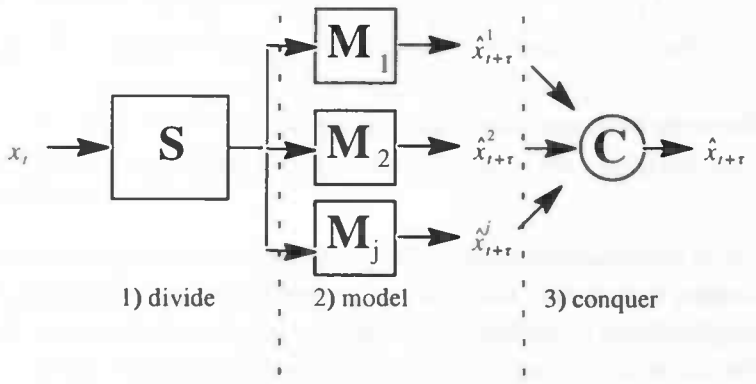

Figure 1.4 A modular approach to multistationary time series modeling

Because of these assumption the problem of modeling multistationary time series with a modular model essentially becomes a segmentation problem. Therefore attention shifts to the problems associated with the process of segmenting the multistationary time series:

1. *Source Identification*. How many different types *classes* of stationary regions are there and what are their statistical characteristics?.
2. *Time Series Classification* How to identify the stationary regions during the forecasting task. In other words, given a particular input of past observations which local model is appropriate at the given moment?.
3. How to evaluate the quality of a particular segmentation during construction of the model?.

All of these segmentation problems require that the (non)stationarity of the time series must at some point be quantified (e.g. is the local data stationary for a particular local model?). If assumptions about statistical properties of the time series can be made, e.g. the properties of the multistationary time series are well understood,. these problems can be handled by hand by the model designer. Just consider the example of Figure 1.3, here it is easily seen that it is the mean value of the series that is changing with time and that there are 2 different stationary regions. In this case a segmentation of the input can be made by setting a threshold. If the input is above 0.5, Local Model 1 is used to predict the next value, otherwise Local Model 2 is used. This is similar to the approach taken in [21] with the so called TAR or *Threshold Autoregressive* models.

But what happens if the statistical properties of the multistationary time series are not known?. Take for example the multistationary series of Figure 1.5. It is clear that there 2 stationary regions in this series. But it is not clear exactly which statistical properties are switching.

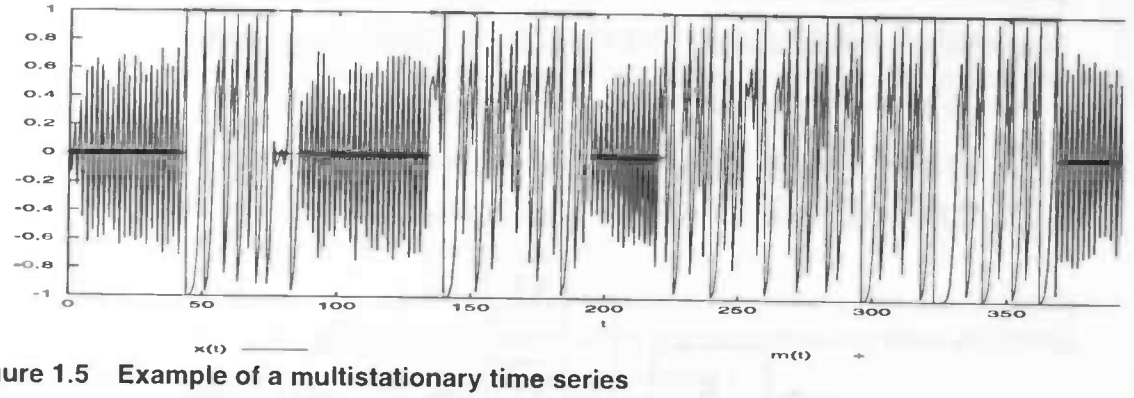

Figure 1.5 Example of a multistationary time series

When addressing the problems of segmenting multistationary, the following *chicken and egg* problem is encountered

In order to build a mulistationary time series model, stationarity of the time series must be quantified. In order to quantify stationarity, proper knowledge is needed about the process underlying the time series (which statistical properties switch in time?). But this was the reason why the time series model needed to be build in the first place. So in order to build a model, a model would be needed, etc., etc.

The discussion in thesis will be limited to the third problem of segmenting multistationary time series, e.g. the question how to evaluate the quality of the segmentation during the constructing of a multistationary time series model in the face of the chicken and egg problem.

One solutions to this problem, is to avoid quantifying stationarity of local data, and to estimate the quality of the segmentation indirectly. This is the solution that is used by all the modular networks that were investigated in the context of this thesis (See chapter 3.). In particular these modular models estimate the quality of the segmentation from the performance of the local models. This is motivated by the assumption that if the performance of the local models is good, then the local data must be stationary, and thus that segmentation quality is good.

At each iteration of their construction process the performance of the local models is evaluated, and the segmentation quality is estimated from these values. Using this estimate the segmentation is updated to provide a new segmentation which produces lower model errors. The construction process continues until the local models show no more significant reduction of their errors. At this moment, it is assumed that the local data is stationary and that the segmentation is optimal (See Figure 1.6).

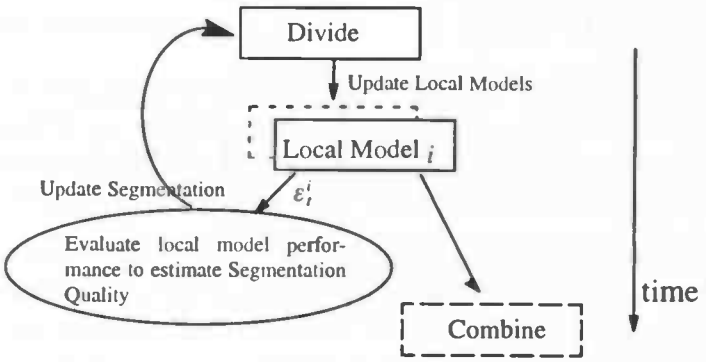

Figure 1.6 Estimating Segmentation Quality from Local Model Performance

There are a number of problems associated with estimating segmentation quality in this way, caused by the interdependence of segmentation and modeling during construction:

1. It only allows for one type of quality metric, e.g. quality of segmentation is defined as optimal local model performance.
2. Due to 1, The dynamics of the local models in practice do not necessarily correspond to the stationary regions of the multistationary time series. It is entirely feasible that a particular segmentation exists which produces an excellent model in the context of forecasting, but does not provide any insight in the modeled process (interpretation problem).
3. If an 'intelligent' segmentation device is used (such as MLP in the GEN models), it is possible that a clear separation of concerns is lost, e.g. part of the modeling task ends up being done by the segmentation device (e.g. the 'gate' in GEN models).
4. The local models need to be constructed simultaneously with the segmentation. In practice this can lead to long training times if a great number of local models are used. Ideally one would like to have a modular construction process for a modular model, e.g. first a satisfactory segmentation is made, secondly all the local models are constructed only once for this particular segmentation.

To address these problems is the goal of this thesis and this will be discussed further in the following section:

1.3. Thesis Goal

It is the aim of this thesis to break the interdependence between segmentation and local modeling in modular time series models. It is believed that a method of estimating segmentation quality that does not rely on local model performance will lead to a more desirable construction process for multistationary time series models (see Figure 1.7).

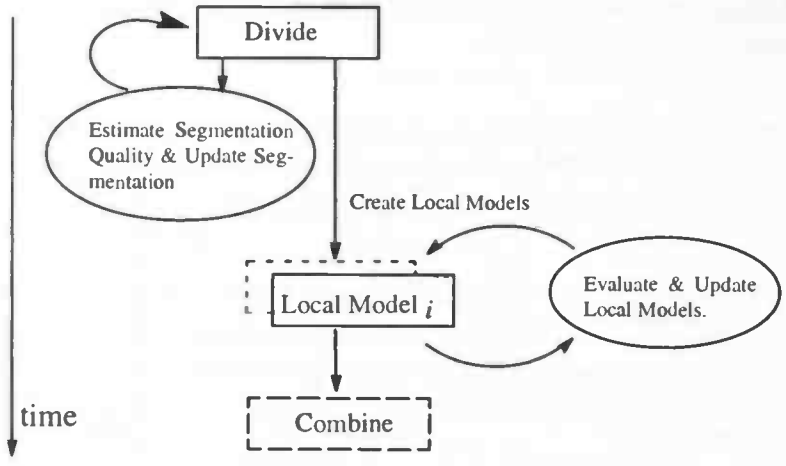

Figure 1.7 Estimating Segmentation Quality before constructing Local Models

With such a measure, the construction process for modular models will have more desirable properties:

1. The freedom to use different quality measures for the segmentation. This way requirements for a certain type of segmentation can be enforced by the model designer by choosing an appropriate measure.
2. The modular model will have a modular construction process, the local models need only be trained once, *after* a satisfactory segmentation has been achieved, reducing complexity and training effort (see Figure 1.7).
3. A clear separation of concerns between the modeling and the segmentation task.

Thus the thesis goal is summarized as follows:

To find a method to estimate quality of segmentation in modular time series models independent from local model performance.

And the following hypothesis is made:

The quality of segmentation in a modular time series model can be measured independent from local model performance.

Assuming that local model performance will be good if local data is stationary, the following hypothesis is motivated:

The quality of segmentation can be estimated by directly estimating stationarity of local data

Which leaves the question if measures of stationarity exist that can be used to quantify stationarity when the statistical properties of the time series are not known (e.g. the chicken and egg problem introduced earlier). To overcome this problem the following hypothesis is made:

Stationarity can be quantified by means of a limited set of traditional statistics

This hypothesis will be motivated further in chapter 4.

The approach taken in this thesis is to find a number of estimates for segmentation quality that do not depend on local model performance. The quality of these segmentation measures will be tested in a modular model and conclusions will be drawn on the validity of these hypothesis and assumptions and the usability of the segmentation measures.

1.4. Thesis Structure

First the traditional theory of time series modeling will be discussed in greater detail in chapter 2. The general theory of linear and non–linear time series modelling and analysis will be introduced. And more formal definitions of stationarity are given. Finally the problem of modeling multistationarity with monolitic models will be restated in more detail.

Chapter 3. continues with an overview of modular time series models for multistationary time series. A literature survey is presented that provides an overview of a number of modular architectures from current research with a focus on their construction process and the way in which segmentation quality is estimated in these models.

In chapter 4, new measures for estimating segmentation quality will be developed.

In chapter 5. an experimental setup is presented that provides unsupervised clustering of time series. A new modular architecture for multistationary time series modelling is introduced that employs Kohonens Self Organizing Feature Map for the segmentation task. The results of experiments on the quality of unsupervised segmentation of multistationary time series are also presented in this chapter.

In chapter 6. presents the results on the experiments on the quality of the new segmentation measures.

And finally in 7. Conclusions are drawn on the usability of the new segmentation measures.

2. Time Series Modeling

This chapter provides an introduction to the theory of univariate time series modeling. A number of common types of univariate time series models are presented. The main problem associated with these models e.g. their inability to model multistationary time series, is discussed at the end of the chapter.

2.1. Time Series

Definition 2.1:

A time series consists of a number of observations y_t, taken sequentially over time [1]:

$$[y_1, y_2, \dots, y_t], \quad t = 1, 2, \dots, T \tag{2.1}$$

If a time series consists of a sequence of *single* observations y_t, it is referred to as a *univariate* time series. When *multiple* simultaneous observations y_t are made, the sequence $y_1, \dots, y_T$ is referred to as a *multivariate* time series. For reasons of simplicity, only univariate time series are considered in this thesis.

As said in the introduction, a time series is obtained by making observations on some process P. As most real–world processes are *continues*, these observations could be made at any point in time. However, in the context of time series modeling only *discrete* time series are considered, where observations are made at equally spaced intervals over time, e.g.:

$$Y_n = Y(n \cdot \delta t + t_0) \tag{2.2}$$

Also, a time series is considered to be produced by a *stochastic* process. Each observation on a stochastic process is unique, that is, it represents only one of many possible results which might have occurred. As a consequence, if the output of the process is recorded as a function of time, a *specific* time series is obtained. If the output of a second, identical process is recorded at the *same* time, it would produce a *different* time series. A single time series (or *sample function*) is thus only one realization of a stochastic process. The stochastic process itself is defined by the *ensemble* of *all* possible realizations or time series it can produce .

Without further restrictions statistical properties of a stochastic process can only be calculated by considering the ensemble of all possible realizations. For example the ensemble average of a stochastic process:

$$\textit{ensemble average} = \mu_x(t) = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} x_k(t) \tag{2.3}$$

To estimate the ensemble average a large number of realizations are needed, while in practice usually only a single realization is available. The common solution is to assume that the process is *ergodic*. A full definition of *ergodicity* is beyond the scope of this thesis, but what it basically means is that a suffiently long single realization of an ergodic process gives consistent estimates of the ensemble statistical properties of the process. Attention may then shift to time averaging a single realization k:

$$\textit{time average} = \mu_x(k) = \lim_{T \to \infty} \frac{1}{T} \int_0^T x_k(t) dt \tag{2.4}$$

and for an ergodic process:

$$\mu_x(k) = E(\mu_x(t)) \tag{2.5}$$

The time series used in this thesis are assumed to be ergodic, and consequently only single realizations are considered.

2.2. Time Series Modeling

An important property of time series is that the value of an observation y_t depends on the values of past observations $y_{t-1}, y_{t-2}, \ldots, y_{t-N}$. It is the aim of time series modeling to find the relations between past and future observations from a given time series and express them in a model.

A univariate time series model explains the movements of the series y_t by predicting some future value $\hat{y}_{t+\tau}$ using only past the observations y_t.. The model defines a mapping M which relates these past observations to the forecast $\hat{y}_{t+\tau}$ (Figure 2.1).

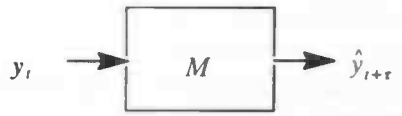

Figure 2.1 A univariate time series model

The architecture of a univariate model is characterized by:

1. The type of mapping M. This can be anything from a constant, a linear function, or various types of non–linear mappings.
2. Its order N, the number of past observations considered to produce the forecast.
3. The number of hidden layers/hiddens if the model uses some form of multilayer approach (as in a Multilayer Perceptron).

The mapping M is associated with a parameter vector w, also known as *model parameters*. These parameters are estimated from a given time series during construction of the model with the aim of expressing the dynamic relations of the time series.

Various statistical methods for estimating model parameters exist and depending on the type of model different methods will be used. Traditional analytical methods of estimating model parameters include Ordinary Least Squares regression and estimation from the correlogram of the time series using Yule–Walker equations [1].

More recently sample–wise or pattern–wise parameter estimation methods are used. These include Expectation Maximization (EM) [19][20], Maximum Likelihood (ML) and Least Mean Squares (LMS) estimation and also Error Back Propagation [17]. These methods are iterative algorithms where w is updated at each step with the aim of minimizing the error of the model with respect to forecasting. This process of iteratively adapting model parameters is referred to as *learning*, and these iterative algorithms for parameter estimation are known as *learning algorithms*.

Depending on the properties of the time series, models with different characteristics will be used. In the following section a number of common models are discussed (These models will also be used later in this thesis).

2.3. Time Series Models

2.3.1. Mean Model

The simplest time series model is the *mean* model. It simply predicts the mean or *bias* of a time series and past observations are not considered:

$$\hat{y}_t = w, \quad \forall_t \tag{2.6}$$

Its single model parameter w is estimated as the sample mean of a number of observations y_t from the series in the following way:

$$w = \frac{1}{N}\sum_{i=1}^{n} y_i \tag{2.7}$$

Although it may not be suitable for many types of time series, it has the advantage that it can easily be constructed.

2.3.2. Linear Models

Time series are traditionally modeled as linear stochastic processes. Here the assumption is made that the relations between observations are linear and that these relations can be expressed by a linear stochastic process.

An example of a linear stochastic process is the MA or Moving Average process. An MA(q) process of *order* q is defined by:

Definition 2.2 MA Process

$$y_t = \varepsilon_t + \theta_1\varepsilon_{t+1} + \ldots + \theta_q\varepsilon_{t-q}, \quad t = 1, 2, \ldots, T$$

Where ε_t is a sequence of uncorrelated random variables with mean zero and constant variance and θ are parameters. A particular set of values of $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_T$, results in a corresponding sequence of observations $y_1, y_2, \ldots, y_T$. By drawing a different set of values of $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_T$, a different set of observations is obtained, and the MA(q) process can be regarded as being capable of generating an infinite set of *realizations* over the period $t = 1, 2, \ldots, T$. Thus, the process effectively defines a *probability distribution* for the random variables $y_1, y_2, \ldots, y_T$.

Another example of a linear stochastic process commonly used in modeling time series is the AR or Auto Regressive process. The AR(p) process of *order* p is defined as:

Definition 2.3 AR Process

$$y_t = \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \varepsilon_t, \quad t = 1, \ldots, T$$

The MA(q) and AR(p) processes can be combined to form the ARMA(p,q) or Autoregressive–Moving Average process defined by:

Definition 2.4 ARMA Process

$$y_t = \theta_1 y_{t-1} + \cdots + \theta_p y_{t-p} + \varepsilon_t + \theta_1\varepsilon_{t-1} + \cdots + \theta_q\varepsilon_{t-q}, \quad t = 1, \ldots, T$$

An ARMA(p,q) model for a particular time series instance y_t can be found by first estimating the appropriate model order, that is finding the correct values for p and q. [1] suggests overestimating p,q followed by a cycle of construction/evaluation steps. At each step p and q are

decreased as long as the model performance remains satisfactory. For a pure MA(q) process, [1] shows that the order q can be directly estimated from the correlogram.

Various methods of parameter estimating can be used for parameter estimation in ARMA(p,q) models. Ordinary Least Squares, Yule–Walker equations and LMS are commonly used as well as Error Back Propagation.

Due to their linear mapping, ARMA models can only be used effectively in modeling linear time series.

2.3.3. Artificial Neural Networks

The assumption of linearity in ARMA models is rather restrictive as most real world time series are of a non–linear nature. Luckily the assumption of linearity can be dropped with the application of Artificial Neural Networks or ANAS in time series models. Due to their ability to express arbitrary complex non–linear relations between their inputs and outputs ANNs can effectively be used to create non–linear time series models.

Traditionally ANNs are inspired by the way the human brain performs its wide range of tasks. Both the brain and ANN perform computations by means of a massive interconnect of simple (non–linear) processing units. The processing units of the human brain as well as those of ANNs are referred to as *neurons*.

The *neuron* is the basic information processing unit used in artificial neural networks. It calculated the sum of its weighted input signals and transforms this sum to form the output by means of an *activation function*. Figure 2.2 shows the neuron model, and its most commonly used non–linear activation function (the sigmoid or logistic function).

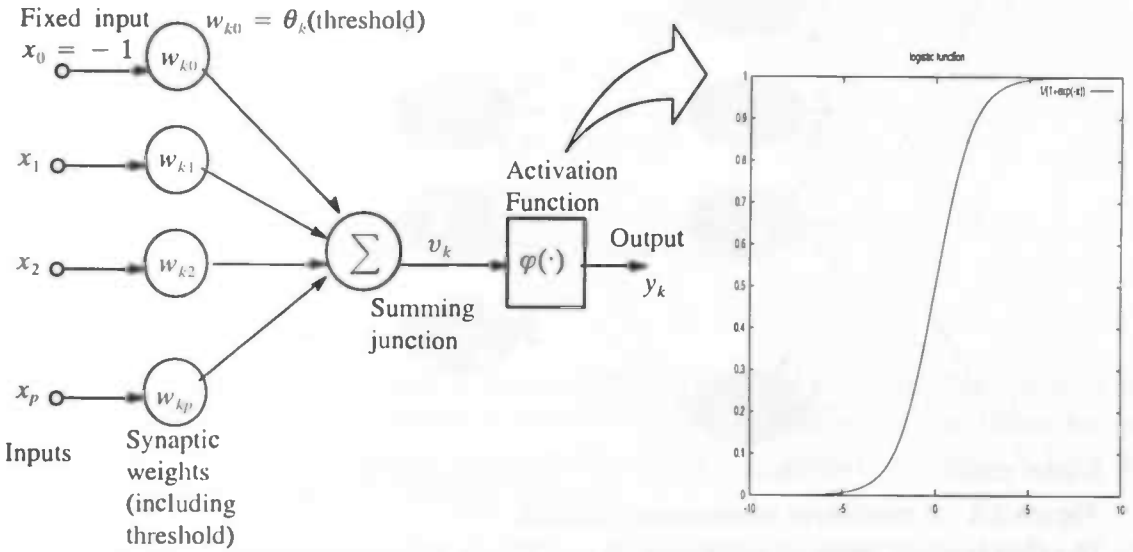

Figure 2.2 The Artificial Neuron

The neuron model contains four basic elements:

- A set of connections, characterized by their weights (w_{kp} in Figure 2.2). This weight is multiplied by the input signal. The connections are uni–directional, or *feedforward* connections.

- A summing unit for summing the weighted input signals.
- An activation function $\varphi(\cdot)$ for transforming the sum into the output of the neuron.
- A *bias* or *treshold* term characterized by a fixed input (w_{k0} in Figure 2.2).

Mathematically the artificial neuron can be described by the equations:

$$u_k = \sum_{j=0}^{p} w_{kj} x_j \tag{2.8}$$

$$y = \varphi(u_k) \tag{2.9}$$

Where $x_1, x_2, \ldots, x_p$ are input signals and $w_{k1}, w_{k2}, \ldots, w_{kp}$ are the connection weights of neuron k.

Two commonly used activation functions are;

- The *sigmoid* activation function (shown in Figure 2.2). This is the most common activation function used in ANNs. It is defined by:

$$\varphi(u) = \frac{1}{1 + \exp(-au)} \tag{2.10}$$

- The *hyperbolic tangent* activation function, defined by:

$$\varphi(u) = \tanh(\frac{u}{2}) \tag{2.11}$$

It is also possible to use a linear activation function.

The manner in which the artificial neurons are connected determines the architecture of the ANN. Figure 2.3 gives an illustration of a *multilayer feedforward network* architecture.

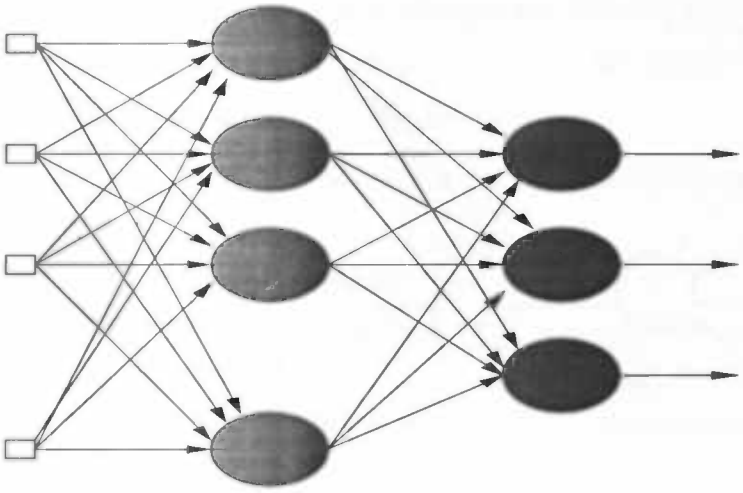

Figure 2.3 A multilayer feedforward network

The first layer of neurons is the *input layer*. This is where the network receives information from its environment. The second layer is called a *hidden layer*. A multilayer feedforward network contains one or more hidden layers. The last layer of the network is the *output layer*.

Information in the feedforward network flows from the input layer through the hidden layer(s) to the output layer. In this way the network provides a functional input–output mapping. Depending on the type of activation function used in the neurons this mapping can be either linear or non–linear.

The actual mapping of the network is determined by its free parameters, the connection weights w_{kp}. A mapping can be *learned* by the network by employing a *learning algorithm* in which the weights are adjusted iteratively according to *examples* from the desired mapping. The class of multilayer feedforward networks that employ the *error back–propagation* learning algorithm are called Multilayer Perceptrons (MLP). The MLP architecture and the back–propagation learning algorithm are treated extensively in [17].

Due to the fact that the MLP only provides a *static* non–linear mapping from its inputs to its outputs, the MLP cannot be used directly as a time series model. If the MLP is to be used to capture the dynamic releations of a time series, *memory* must be added to the network structure. The easiest way to do this is to provide *unit delay elements* to the inputs as shown in Figure 2.4. The input vector for the network then consists of a fixed number of previous observation on the series.

The set of unit delay elements can be considered as a separate component from the network structure and is commonly refered to as a *tapped delay line*. Other, more complex neural architectures with memory have also been developed such as the TDNN or *time delay neural network* and FIR networks. An overview of the capturing temporal information in neural networks can be found in [24] and [17].

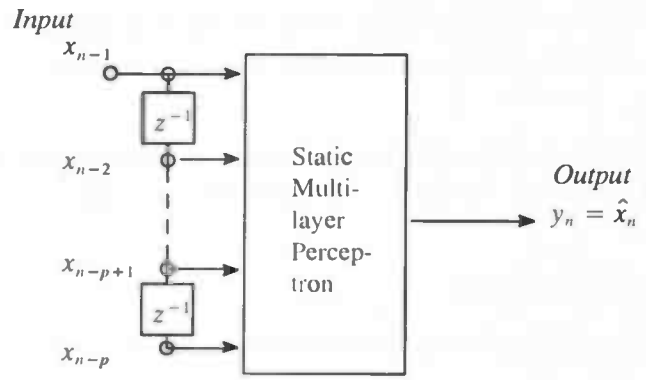

Figure 2.4 MLP used as a non–linear time series model

2.4. Stationarity

The models of the previous section all share the property that they assume stationarity of the modeled time series. The assumption of stationarity and the existence of time series that are *non–stationary* leads to a modeling problem that will be discussed in section 2.5. First the concept of stationarity must be properly defined.

The definition of stationarity in time series may be a problem by itself, and a number of different (statistical) definitions can be found in literature [1][9][17]:

Definition 2.5 Weak Stationarity:

A time series is considered weak(ly) stationary if the following conditions are satisfied for all values of t:

$$E[y_t] = \mu \tag{2.12}$$

$$E[(y_t - \mu)^2] = \gamma(0) \tag{2.13}$$

$$E[(y_t - \mu)(y_{t-\tau} - \mu)] = \gamma(\tau), \quad \tau = 1, 2, \ldots \tag{2.14}$$

In other words, a time series is considered weakly stationary if:

- It does not have a time–varying mean value.
- It does not have a time–varying mean square value (variance).
- Linear dependencies between observations at different times (autocorrelation) do not vary with time.

Note that this only requires linear properties to remain constant over time. This definition is usually found in literature on traditional time series models (ARMA), where the assumption of linearity is already made. As real world time series may be nonlinear, a more general definition of stationarity is also used:

Definition 2.6 Strict Stationarity:

> *A time series is said to be strictly stationary if the joint probability distribution* $X(t_1), \ldots, X(t_n)$ *is the same as the joint distribution of* $X(t_1 + \tau), \ldots, X(t_n + \tau)$ *for all* τ.

The problem with this definition is that in time series modeling the probability distribution of the series is unknown (This is the reason why a model is needed in the first place). So this definition does not provide a means to make statements about the (non)–stationarity of some arbitrary time series.

As estimates on different features of the probability distribution can be calculated from statistical properties of the series (mean, variance, skewness, kurtosis, etc. etc), the following more intuitive definition is motivated:

Definition 2.7 Thesis definition of Stationarity:

> *A timeseries is said to be stationary if its statistical properties do not vary with time.*

Which leaves the following questions unanswered:

1. Which statistical properties should not vary with time for the time series to be considered stationary.
2. At what timescale the statistical properties of the time series should not vary with time.

Altough literature research on the subject did not provide answers to these question, this will be the definition of stationarity used troughout the remainder of this thesis. Tthe question which statistics and timescale are appropriate will need to be addressed in later chapters.

Two important examples of non–stationary behaviour found in time series are *switching* and *trending*. Time series that switch between different *stationary regions* at certain points in time are said to be *multistationary*, or *piecewise stationary*. Within one region, the statistical properties do not change and the time series is stationary, but on a longer timescale the series is non–stationary as its statistical properties change from one region to the other. Time series of this type are usually generated by a process that switches between multiple modes of sta-

tionary dynamics at certain points in time. A synthetic example of a multistationary time series is given in Figure 2.5, where the switching between two stationary regions is shown:

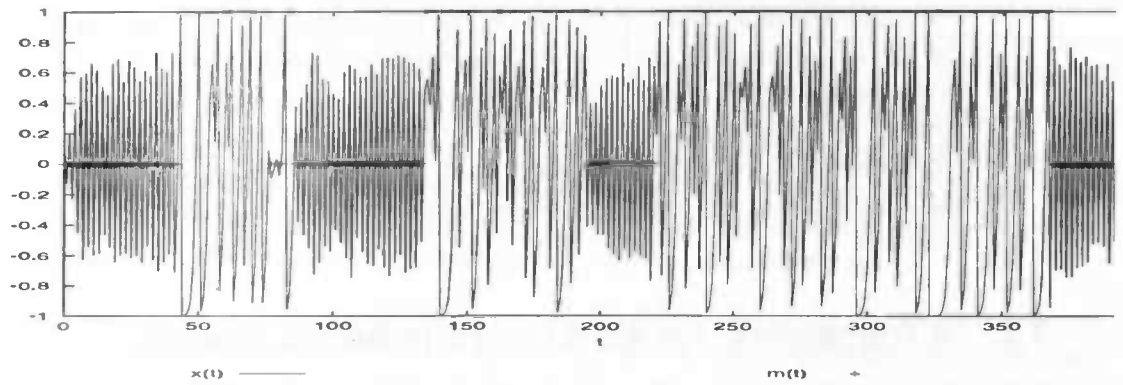

Figure 2.5 Multistationary time series

Trending non–stationarity is found when one or more statistical properties slowly change with time. An example of this kind of non–stationary behavior is shown in Figure 2.6.

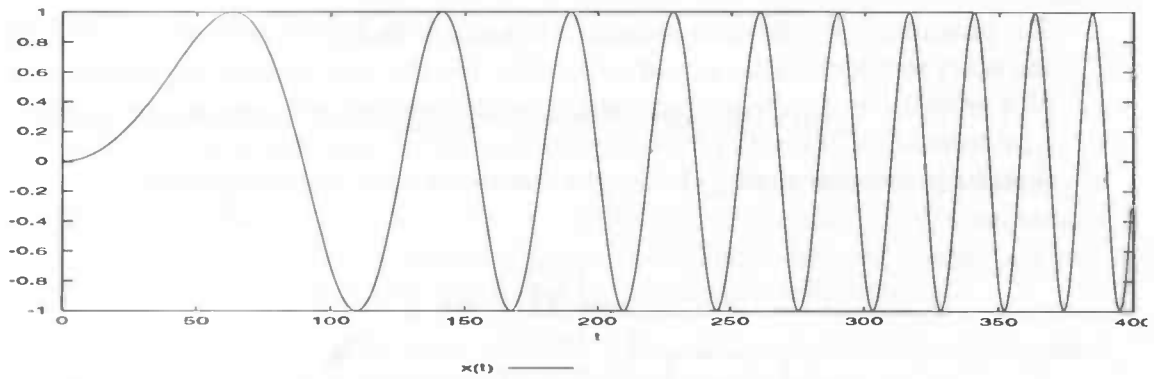

Figure 2.6 Trending non–stationary time series

This type of non–stationarity can be observed in industrial processes, where the dynamics of the underlying process slowly change in time due to wear, or aging of the mechanical components. Only *multi–stationary* time series will be considered in this thesis.

2.5. Monolitic Models and Nonstationarity

Now that the concept of stationarity in time series has been defined, the problem of modeling *multi–stationary* time series can be reiterated:

The models introduced in section 2.3. are all *monolithic* models, which means that they only contain a single model process. This single process can only be estimated to reproduce the average *global* dynamics of a multistationary time series. The monolithic models will therefor underperform in *local* regions of the multistationary time series. An example of this problem has already been given in section 1.1.

In the following chapter it is shown that this problem can be solved if these monolithic models are used in a modular approach.

3. Modular Time Series Modeling

In this chapter a number of solutions from current literature are presented that address the problem of modeling multistationary time series by employing a modular divide & conquer approach. These models segment a time series into its stationary regions and specialize a number of local models for modeling these regions. This chapter discusses the problem of estimating segmentation quality during their construction. It will become clear that these models use the error performance of the local models as an estimate for segmentation quality, and that this leads to some undesirable properties of current modular models.

3.1. A framework for Modular Time Series Models

In the previous chapter a number of monolithic time series models were presented. It was shown that these models fail when modeling multistationary time series. A number of multistationary time series models have been introduced in recent literature that overcome the problems associated with monolithic models. These models employ a divide and conquer strategy to multistationary time series modeling.

The principle of divide–and–conquer is a principle that has proven to be very useful in solving many problems in the area of computing. Divide–and–conquer algorithms attack a complex problem by dividing it into simpler problems whose solutions can be combined to yield a solution to the complex problem. In the context of multistationary time series models, the divide and conquer strategy is used in a *modular* framework (Figure 3.1):

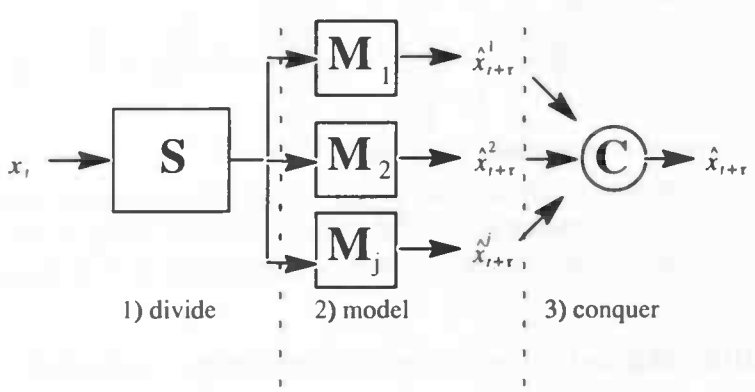

Figure 3.1 Modular divide–model–conquer framework

This framework, which will be referred to as the divide/model/conquer framework is motivated by the following definition of multistationary time series:

Definition 3.1 Multistationary Time Series:

An observation y_t in a multistationary timeseries can be considered to be generated by a source process $S(z_t)$, where z_t is a time varying *source parameter*, taking values in a finite set of parameter vectors $\Phi = \{\phi_1, \phi_2, \ldots, \phi_K\}$. For every value of z_t a source process $S(z_t)$ is activated, which produces y_t using $y_{t-1}, y_{t-2}, \ldots$, and parameter vector ϕ [13].

This means that a multistationary time series is considered to be generated by a finite number of *stationary* sources that are activated in time according to some unknown process.

The idea behind the modular approach is then to identify these different sources (**divide**) and to construct an appropriate *local* model for each distinct source (**model**). The local models are constructed using only stationary data generated by their associated source processes. During forecasting, at each time step t, the currently active source process $S(z_t)$ is classified. Depending on the actual implementation of the modular model on more local model(s) are then selected to provide the forecast. Finally these forecasts are combined to provide the response of the modular model (**conquer**). In this way the local models are only used on stationary data and the modular model is expected to outperform monolithic *global* time series models (chapter 2.) in the context of forecasting multistationary time series.

In this thesis the assumption is made that a good local model can always be made for stationary data, and that the forecasts of these models can be combined optimally, the problem of modeling multistationary time series can then be considered to be a segmentation problem. The focus of this thesis will therefore be limited to the problems associated with the segmentation of multistationary time series.

In the following section the problems associated with the segmentation of multistationary time series are discussed further and a number of modular time series models from literature are reviewed to see what kind of solutions exist to overcome them.

3.2. Segmentation in Modular Time Series Models

The aim of the segmentation in modular time series models is to increase the stationarity of the data that is presented to the local models, e.g. to enforce a specialization of local models on the stationary regions of the multistationary time series. The problem of segmenting multistationary time series can be broken down into the following 3 subproblems:

1. *Source Identification*. How many different types *classes* of stationary regions are there and what are their statistical characteristics. e.g. the source set $\Phi = \{\phi_1, \phi_2, \ldots, \phi_K\}$ must be determined.
2. *Time Series Classification* How to identify the stationary regions during the forecasting task. In other words, given a particular input of past observations which local model is appropriate at the given moment?. e.g. determining at every time step t the active source $S(\phi_k)$ that generated y_t.
3. *Evaluating Segmentation Quality* How to evaluate the quality of a particular segmentation during construction of the model? e.g. Does a particular segmentation enforce stationarity of local data?.

In the following section a number of implementations of modular models from recent literature are reviewed to see how these problems are handled.

3.2.1. PREMONN

The modeling of multistationary time series is discussed in a number of papers by Kehagias and Petridis [13], [14], [15].

In [13] they present a general framework (PREMONN) for the time series classification problem. The PREMONN architecture (Figure 3.2) consists of a number of predictor modules and a decision module. It is assumed that the source processes are known and that the appropriate local models have been trained. The article presents a decision module that employs a Bayesian approach to the time series classification problem. A every time t the decision module produces an estimate of the *conditional posterior probability* p_t^k, that the observation y_t was produced by process ϕ_k from the observations $y_{t-1}, y_{t-2}, \ldots$, . The active source at time t is credited to be the one with maximum probability p_t. An iterative method is presented to efficiently compute p_t^k from p_{t-1}^k, $y_{t-1}, y_{t-2}, \ldots$, and the prediction error e_t^k, where $e_t^k \doteq y_t - y_t^k$. The PREMONN architecture by itself provides no solution to the source identification problem. The outputs of the local models are combined (mixed) according to their posterior probability. Thus input vectors are not segmented exclusively among the different local models in the PREMONN architecture. Instead, all models are trained on the same data, and a 'soft' segmentation is performed afterwards by the decision module. The quality of the segmentation during construction is evaluated by means of the error of the local models, and the segmentation itself is updated by calculating new posterior probabilities from these errors at each step of the learning algorithm.

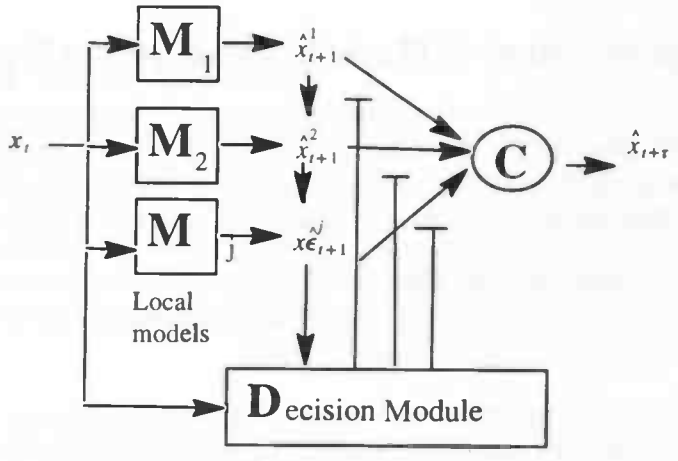

Figure 3.2 PREMONN

An application of the PREMONN architecture is given in [15] where Kehagias and Petridis *et al.* present the use of their bayesian classification algorithm in the context of short term electrical load forecasting. The article presents three global models that are in use to predict the hourly demand for electricity on the island of Crete, Greece. The models use the relations between several factors such as weather conditions, past demand patterns, day of week and time of day to produce the short–term demand forecast (one hour). Two of the models are of the linear ARMA type, the third a non–linear ANN model. Motivated by the fact that the errors made by these models depend on the time of day (some models performed better during certain hours than others), a PREMONN is presented with a bayesian decision module as in [13]. The decision module combines the forecasts made by the three existing 'local'

models based on their posterior probability p_t^k. It is shown that the PREMONN consistently outperforms the original global models.

A solution for the source identification problem is presented in [14]. In this paper, an unsupervised divide and conquer method is presented that can be used in the PREMONN framework. The presented algorithm starts of with a single non–linear ANN local model that is trained on the first fixed length data block of a series y_t. Then for each iteration of the algorithm a consecutive data block is considered. Time delayed windows or *segments* are taken from the data block. The segments are presented to the current local models to calculate the *Segment–Average Square Prediction Error, or SASPE* for each segment. Each segment is assigned to the local model with the the minimum SASPE for the segment. If the minimum SASPE falls below a certain *threshold* a new local model is created, and the segment is assigned to this model instead. When all segments of the data block are assigned, all the local models are retrained for a number of iterations using their most currently assigned data segments. The algorithm continues until each consecutive data block of y_t has been considered.

3.2.2. Gated Expert Networks

Gated Experts (Weigend [10]) and Mixture of Experts (Jacobs *et. al.* [19] [20]) are a class of modular time series models typified by the architecture shown in Figure 3.3.

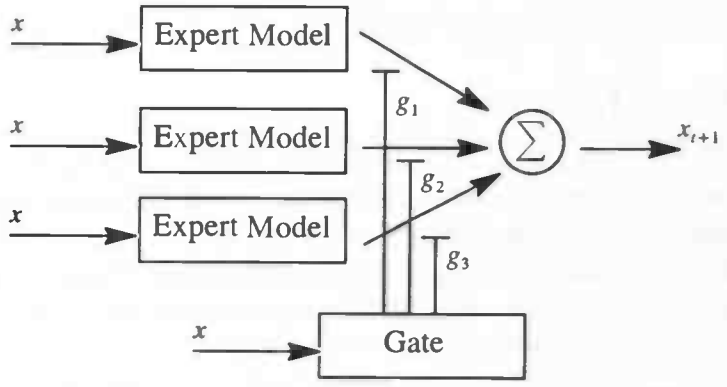

Figure 3.3 General architecture of Gated Experts and Mixture of Experts models

This architecture is more or less the same as that of the PREMONN (Figure 3.2). but Gated Experts and Mixture of Experts also solve the source identification and classification problems.

In this architecture a component called 'Gate' is responsible for the segmentation of the input space (divide). The gate assigns a weight g_k to each of the experts to denote the probability that it generated the current input vector x. In this way an input vector can be assigned to more than one expert, thereby *soft–partitioning* the input space.

The response of the model is a linear combination of the outputs of the individual experts, weighted by the probabilities given by the gating component (conquer). The difference between the Gated Experts (GEN) and Mixture of Experts (MoE) models is that the GEN employs non–linear ANNs for both the gate and experts. The MoE architecture uses linear models for the gate as well as the experts.

The GEN and MoE architectures are constructed (trained) using an iterative algorithm (Expectation Maximization, EM, see also [17]) that simultaneously trains the gating network

and the expert networks. During training input vectors are assigned to the individual experts according to their current probability given by the gate. In each iteration (epoch) of the training algorithm the experts are trained on their currently assigned input vectors. After which the gate is updated from the current error performance of the experts and the input vectors are reassigned to the experts based on the new probailities given by the gate. In this way the different local models 'compete' for train data and they will specialize more at each step of the algorithm. Training ends when the local models show no more significant reduction in their error performance.

Given the training method as described above, it can be seen that in GEN and MoE models, source identification is achieved by the competition among experts, time series classification is done during forecasting by the gate, as it assign a probability to each model according to the current input and finally that the quality of segmentation during training is evaluated from the performance of the local models.

The training a GEN or MoE model can be computationally intensive, as each input vector can be assigned to each of the experts (worst case) and each expert has to be retrained at every step of the training algorithm. The GEN and MoE models still require the designer to make some assumptions about the process underlying the series. The models use a fixed number of experts. Therefore, an estimate must be made before training on the number of experts that are needed to create a good model. This in contrast to [14] where the right number of experts (local models) is automatically found.

3.2.3. Competing Neural Networks

Another implementation of the divide–conquer–model framework is given by Kohlmorgen [11]. The article mainly focuses on the 'divide' task. The source identification task is performed during training where several local models (experts) compete for train data. The approach is similar to that of [10], but there is no gate. Instead the error performance of the experts is lowpass filtered to give an estimate of the posterior probability for each expert that the current input vector was generated by it.

Competition between the experts is increased during training. Also at every step the experts are trained on the data associated to them by the competition. After training the models represent the sources present in the data. It is clear that this method is computationally expensive especially when non–linear local models are used (the same problem as in [10]).

The time series classification task in these 'competing neural network' models is performed by the low–pass filter as it assigns a weight to each of the local models based on their performance on recent history.

The quality of segmentation is estimated during construction from the low–pass filtered error performances of the local models (At each iteration of the competition, just as the GEN and MoE models).

F. Reine and R. Zoeller [16] acknowledge the computationally expensive method of [10] and [11] and present a variant which can significantly reduce training time by using less general local models. The same competition between experts as in [11] is used to perform the source

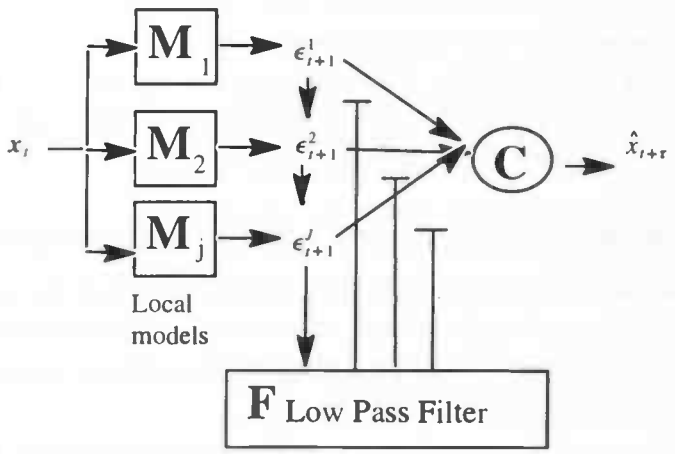

Figure 3.4 Competing Neural Network

identification task. A two–stage local model is introduced that consists of a non–linear stage followed by linear stage. At the beginning of training, the local models are trained to predict the whole time series. During training only the linear stage of the local models is updated thereby reducing the training effort. The article presents some experimental results that show that the two–stage local models can be used effectively on a number of multistationary time series.

3.3. The problem of Estimating Segmentation Quality

The modular time series models discussed in the previous chapter all share the property that in the learning process of the model, the quality of segmentation is evaluated by means of the error performance of the local models. The reason for doing this is clear: By assuming that local data is stationary (and hence segmentation quality is good) if the models are performing well, these models can avoid the chicken and egg problem presented in chapter 1. E.g. the stationarity of local data is estimated indirectly from the performance of the local models, and no assumption about the properties of the multistationary time series need to be made.

The problem with this approach is that it invariably leads to a construction process where the segmentation needs to be learned simultaneously with the local models. This means that the number of free parameters that must be optimized during construction is large e.g. Lets say that the number of parameters of the segmentation device is N, and that the number of parameters of a local model are M. Then the learning algorithm of the complete modular model must cope with $N \times M$ degrees of freedom in optimizing the performance of the complete modular model. This is the cause of the problems already discussed in chapter 1. (e.g. interpretation problems, loss of separation of concerns, long training times).

It is the goal of this thesis to find a way to estimate the quality of segmentation in modular models without accounting local model performance. This way a good segmentation can be found first, aftershock local models only need to be constructed once, leading to a *modular* construction process with only $N+M$ degrees of freedom, In the following chapter new measures of segmentation quality are presented that could potentially achieve this thesis goal.

4. Measures for Segmentation Quality

In this chapter a new measure is proposed for estimating quality of segmentation in modular time series models. This measure does not depend on local model performance. Instead segmentation quality is measured by estimating stationarity of local data using traditional statistics.

4.1. Measuring Segmentation Quality in Modular Models

It has been shown in the previous chapter that the common approach for measuring segmentation quality in modular models is to estimate it from the performance of the local models. It was also shown in chapter 1. that there are a number of problems associated with this approach which motivated the goal of this thesis;

To find a way to estimate segmentation quality in a modular model, independently of local models.

In order to find such a new measure of segmentation quality, the typical architecture of a modular model is inspected for locations for measuring segmentation quality in Figure 4.1.

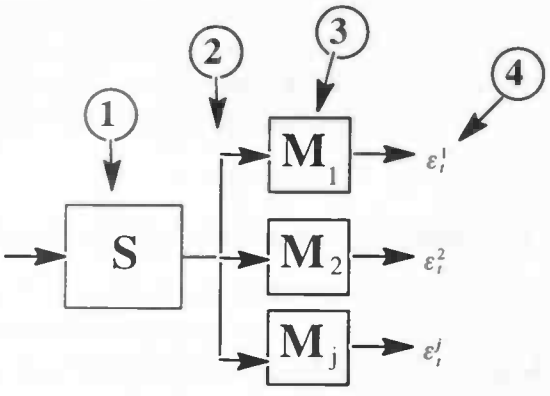

Figure 4.1 Locations for measuring quality of segmentation in a modular model

This inspection reveals 4 locations where segmentation quality could be measured;

At location 4. the error performances of the local models are used to provide an estimate for segmentation quality. The problems with this approach have already been described in the previous chapter.

At location 3. properties of the local models could be considered to provide the quality estimate. In [23] large changes in the activations of the hidden units of a non–linear model are used as an indication for segmentation quality. This will again lead to an interdependence between local models and segmentation device and is therefore likely to cause similar problems as location 4.

This leaves locations 1 & 2 as possible locations where the quality of the segmentation could be measured independently of the local models.

At location 1. quality parameters of the segmentation device itself could be used to measure segmentation quality. A disadvantage of this location is that the measure would depend on

the actual type of segmentation device. In the case where segmentation is to be performed by a Self Organizing Feature Map, measures for *map goodness* [22] could potentially be used to estimate segmentation quality.

At location 2. the quality of the segmentation is estimated from the statistical properties of the local data, e.g. by *directly* measuring the stationarity of the local data. The assumption is made that segmentation quality is good if the local data is stationary. Advantages of this location; The stationarity of local data may be measured independently of the type of segmentation device/method used and it doesn't require local models to be build. The main disadvantage is that at present no non–parametric measures of stationarity exist (the chicken and egg problem, see chapter 1).

Both locations 1 & 2 could provide the means to estimate the quality of segmentation independently from the local models. Due to time constraints only location 2 has been considered., and this is the basis for the approach taken in this thesis.

4.2. Measuring Stationarity

A choice was made in the previous chapter to estimate segmentation quality by measuring the stationarity of local data after segmentation. A major obstacle with this approach is the chicken and egg problem introduced in chapter 1; In order to measure stationarity, assumptions must be made on the nature of the data (e.g. which statistical properties are changing/switching in time). Literature research on the subject did not provide a *non–parametric* statistic for measuring stationarity. A common approach taken in literature is to consider only changes in the *mean* and *variance* statistics in time when qualifying stationarity (e.g. the weak definition of stationarity). Obviously this is a rather limited approach that will fail on real–world time series where no assumptions on the non–stationary behavior may be made.

Also, no method to actually *measure* stationarity was found in literature, only methods for *testing* stationarity are given ([1][9]).

To overcome these problems, a new approach is taken in this thesis.

First the hypothesis is made that stationarity of arbitrary time series may be measured by using a wider range of traditional statistics. This hypothesis is motivated by the results of an experiment involving *testing* for stationarity using various traditional statistics. The results of this experiment are presented in the following section.

Based on the stationarity *testing* procedure of this experiment, a new method for actually *measuring* stationarity was developed. This method will be presented in section 4.4.

4.3. Testing for Stationarity

A procedure to test for stationarity is suggested in [1] and [9]. The procedure is inspired by the formal definition of weak stationarity (definition 2.5) and in its original form the procedure consists of calculating the mean and variance statistics for successive *time windows* of a time series.

The resulting sequences of values for mean and variance of the time windows are then also considered to be time series. In the case of trending non–stationarity, this resulting series can

then be tested for the presence of a trend, or in the case of multistationarity, a threshold can be defined on the result series to detect the switching of regimes in the original time series.

This procedure was extended to include a number of traditional statistics that measure properties of the time series; e.g. *sample mean, variance, standard deviation, skewness, kurtosis, autocovariance, rms, median, peak count and zero count*. Definitions of these statistics can be found in Appendix B.

The following experiment was performed to check their usability in testing for stationarity. Figure 4.2 and Figure 4.3 show the results of performing the traditional stationarity test on a trending non–stationary time series and a multistationary time series. In these graphs only those statistics are shown that successfully detect the non–stationary features of the series.

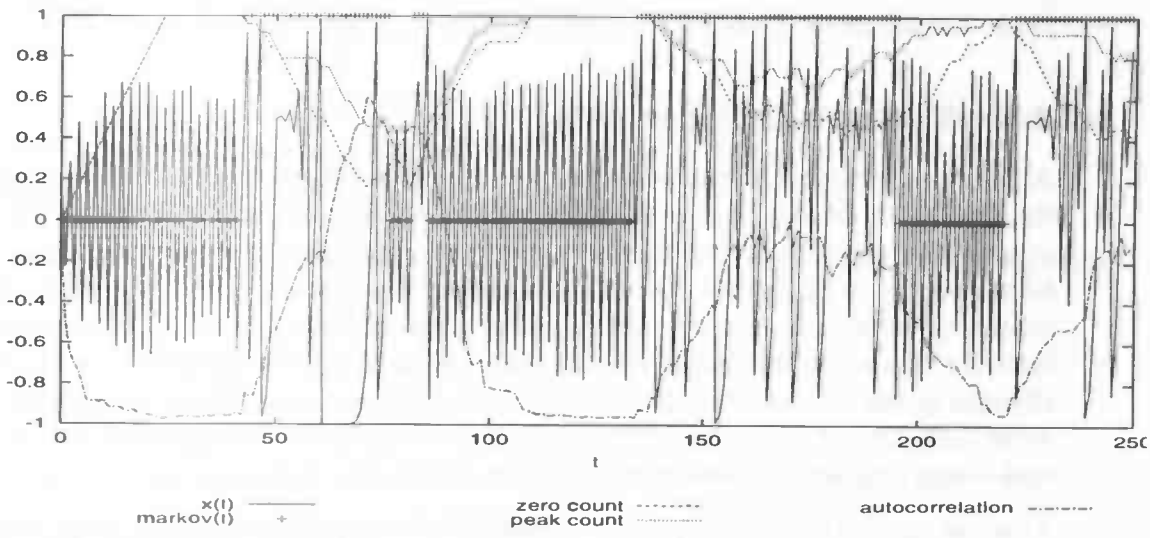

Figure 4.2 Stationarity test of a multistationary time series

In Figure 4.2, the presence of the multistationary switching behavior is detected by the zero count and peak count statistics in the frequency domain, and by the autocorrelation statistic in the time domain.

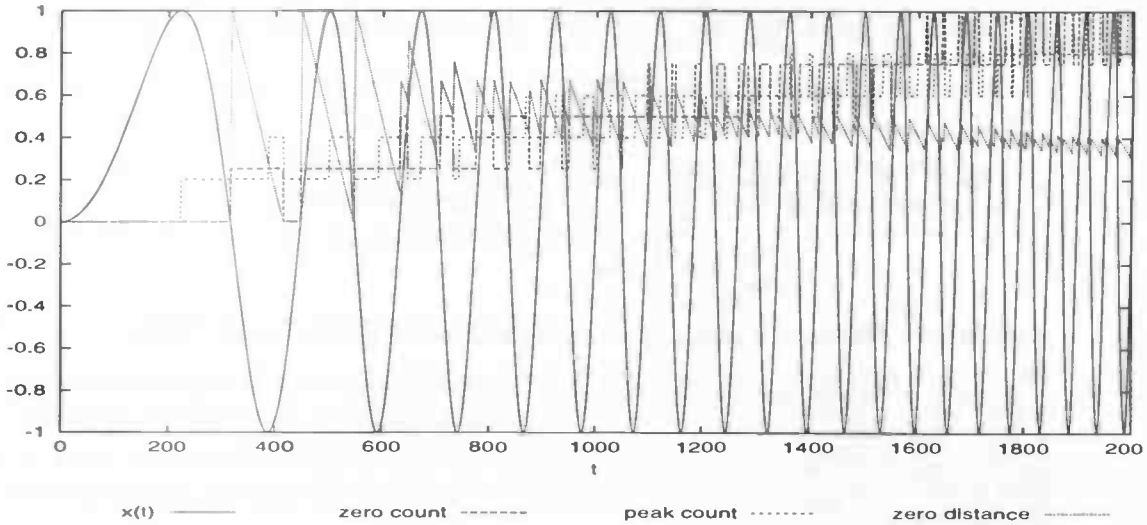

Figure 4.3 Stationarity test of a trending non–stationary time series

In Figure 4.3 the presence of trending non–stationarity is shown by the observed trend in the results for the *zero count* and *peak count* statistics.

The nature of the non–stationary feature influences the capability of a particular statistic to detect the non–stationary feature. Not all statistics of Appendix B were able to detect the non–stationary features of the two time series. Still in both cases at least some were able to detect the non–stationary feature. This corroborates the hypothesis that stationarity may be measured by using a wider range of traditional statistics for arbitrary time series.

Another consideration is the *time scale* at which to detect the non–stationary feature. The time scale is determined by the size, or lag τ of the time window used in the test. The lag should be large enough to permit long–term trends to be differentiated from (random) fluctuations in the time series. The lag should also be small enough to show local non–stationary properties such as regime switching.

Based on the testing method presented in this section, a new method was developed to *measure* the stationarity of time series using traditional statistics.

4.4. The Varratio Method for Measuring Stationarity

In Figure 4.2 it can be observed that the value of a particular statistic used in the test remains more or less constant during one region, but has a different value in the different regions of the series. The idea is to calculate the variance of the result series of this statistic as a measure of stationarity. If a time series is more or less stationary, the variance will be low. If different regimes are present, the variance will be higher. The more the statistical properties in each regime differ, the higher the variance will be. Note that this method will only work for multistationary time series.

The actual variances that are found will depend on the properties of the data, making it impossible to compare the measure of stationarity between different time series. However, this is not a problem because only a relative measure of stationarity between original and local data is needed in the case of a multistationary time series model (Figure 4.4).

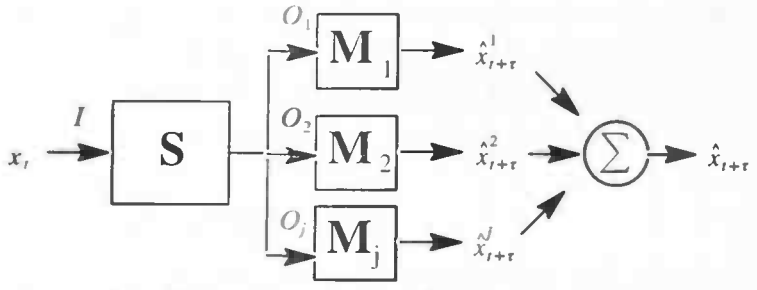

Figure 4.4 Measuring stationarity a modular time series model

During construction of such a model, time series data is presented to the model as a set of consecutive windows I of past observations of the series x_t. It is the task of the segmentation device S to assign each of these windows to a local dataset O_j in such a way that this local data is stationary. The stationarity of the local dataset only needs to be quantified relative to that of the original dataset I.

The above can be summarized in the following definition of the new method for measuring stationarity in modular models. The method is called *varratio.*:

Definition 4.1 Varratio:

Let $f_s(x)$ be a function that calculates a particular statistic s for vector x_t, and $j, 1,\ldots,J$ the index of the local dataset O then:

$$I_{f_s} = \{f_s(x) | x \in I\} \tag{4.1}$$

and:

$$O_{j,f_s} = \{f_s(o) | o \in O_j\} \tag{4.2}$$

Then the varratio η_j of local dataset O_j equals:

$$\eta_j = \frac{var(O_{j,f_s})}{var(I_{f_s})} \tag{4.3}$$

A value $\eta_j > 1$ signals a decrease in stationarity of local dataset O_j, relative to the original dataset I, a value $\eta_j < 1$ an increase.

4.5. Summary

In this chapter a choice was made to estimate quality of segmentation in modular time series models by means of directly measuring stationarity of segmented local data.

In order to circumvent the chicken and egg problem, a hypothesis was made that stationarity may be measured using a number of traditional statistics. (and subsequently motivated by the results of the experiment of section 4.3.

Finally, the varratio method was presented as a method to measure stationarity in modular models using traditional statistics.

The viability of the varratio as a method to estimate segmentation quality in a modular time series model will be tested in chapter 6. The experimental setup needed for these tests will be presented first in the following chapter.

5. Quality of Unsupervised Segmentation

In this chapter the experimental setup is presented to test the quality of the newly developed varratio as a measure for segmentation quality. A new modular time series model was developed that provides unsupervised segmentation of time series, and the quality of the unsupervised segmentation was subsequently evaluated in a number of experiments.

5.1. Unsupervised Segmentation

In the previous chapter a new method for measuring quality of segmentation was presented. This 'varratio' measure is to be used as a way to measure quality of segmentation independent of local models. In order to test the viability of the varratio an experimental setup is needed that provides for an *unsupervised* segmentation of time series (e.g. independent of local model performance, in contrast to the segmentation methods of the models in chapter 3. which are all essentially supervised by the error performance of the local models).

In this chapter this experimental setup is introduced in the form of a new multistationary time series model that provides an unsupervised segmentation of time series by means of a Self Organizing Feature Map. The quality of the unsupervised segmentation of this model was evaluated visually in a number of experiments. The results of these experiments are presented in section 5.3. of this chapter. The data generated by these experiments was subsequently used to test the quality of the varratio measure itself. The results of those experiments are presented in chapter 6.

5.2. The SOLEX Model

A new multistationary time series model was developed as an experimental setup for testing the quality of the varratio method. This model provides an unsupervised segmentation of time series by means of a Self Organizing Feature Map (or SOFM, Kohonen[5]). The model is based on the divide–model–conquer framework introduced in chapter 3. and is called SOLEX or Self Organizing Local EXperts network (Figure 5.1).

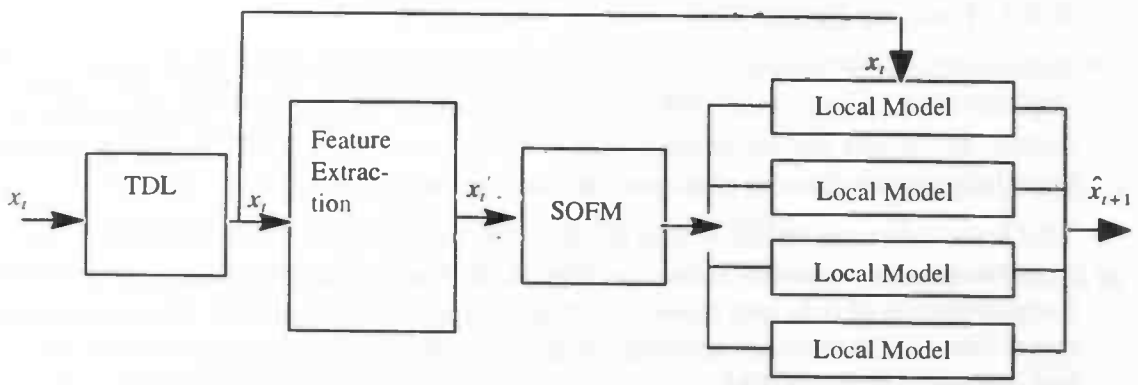

Figure 5.1 The SOLEX model.

The use of the SOFM as a device for time series segmentation is motivated by the fact that it can be used to split a set of input vectors into unlabeled clusters of vectors with similar

statistical properties. In section 4.3. it was shown that in a time series, similarities in the statistical properties of consecutive time windows are an indication for stationarity. If these time windows are presented to a SOFM, it is assumed that the resulting local clusters represent the stationary regions of the original time series.

In the following sections the various parts of the architecture are discussed in more detail.

5.2.1. Tapped Delay Line

The SOFM and the local models used in the SOLEX architecture provide only a *static* mapping from an input vector to an output vector. To be able to capture the *dynamic* nature of a time series some form of memory is needed.

In the SOLEX architecture memory is provided by means of a Tapped Delay Line. The delay line generates a set of consecutive *time windows* from a time series in the following way:

The series:

$$x_1, x_2, \ldots, x_t, \quad t = 1, 2, \ldots, T \tag{5.1}$$

Is converted into a the set of vectors (time windows) of length N:

$$I = \{x_1, x_2, \ldots, x_p\}, p = 1, 2, \ldots, T \tag{5.2}$$

where:

$$x_p = x_p, x_{p+1}, \ldots, x_{p+N} \tag{5.3}$$

The proper size of the time window N depends on the properties of the time series. Appropriate settings for N were investigated in the experiments presented in section 5.3.

There are other ways to include memory in a SOFM, an example can be found in [4] where delay elements are added between the output neurons of the SOFM. The TDL approach was chosen for reasons of simplicity and because the time delayed windows can also used to train the local models.

In the implementation of the TDL the input time series is zero padded at the end. This to obtain a number of delayed vectors equal to the number of observations in the input time series.

5.2.2. Feature Extraction

Although the SOFM always provides *a* segmentation, the nature of a particular segmentation depends on the features the SOFM finds in its inputs. To guide the segmentation process a feature extraction pre–processing step was added to the SOLEX model to exploit prior knowledge about the non–stationary feature of the time series.

This knowledge can be introduced by applying some form of feature extraction on the time windows coming from the TDL. For example the fast fourier transform can be used to force a segmentation of data into clusters with similar spectral properties. Also a combination of one or more of the statistics presented in appendix B can be calculated from the time window and presented to the SOFM.

If no feature extraction is done, the SOFM produces a clustering based on the spatial properties of the data in the input time window (euclidian distance).

5.2.3. The Self Organizing Feature Map

The SOLEX model depends on the SOFM to provide the unsupervised segmentation of time series data. It performs both the *source identification* and *time series classification* tasks of the the divide–model–conquer framework. Source identification is performed during training of the SOFM, as it learns to represent the stationary regions by means of a set of internal *prototypes*. Time series classification is performed by classifying the current input vector of time delayed observations into one of the clusters found during training. This cluster then corresponds to the appropriate local model for the current input.

Formally the SOFM is described as follows:

An input vector of the SOFM is denoted by:

$$x = x_1, x_2, \ldots, x_p, \quad p = 1, 2, \ldots, P \tag{5.4}$$

The SOFM stores a set of *prototypes* or *synaptic weight vectors* defined by:

$$w_c = \{w_{c,1}, w_{c,2}, \ldots, w_{c,p}\}, c = 1, 2, \ldots, C \tag{5.5}$$

Where C is the number of output neurons or *clusters* of the SOFM. When presented with input vector x, the index $i(x)$ of the *winning neuron* or *cluster* is determined by:

$$i(x) = \arg_c \min \|x - w_c\|, c = 1, 2, \ldots, C \tag{5.6}$$

Where $\|\cdot\|$ denotes the *euclidean distance* between vectors.

During training the SOFM is presented with a set of input vectors I. An iterative learning algorithm [17] adapts the weight vectors to minimize the total euclidean distance between the input vectors and the stored prototypes.

After training the SOFM is used to segment a set of input vectors (time windows of observations on the time series):

$$I = \{x_1, x_2, \ldots, x_n\}, n = 1, 2, \ldots, N \tag{5.7}$$

into a set of output clusters:

$$O = \{O_1, O_2, \ldots, O_c\}, c = 1, 2, \ldots, C \tag{5.8}$$

where:

$$x_n \in O_{i(x_n)} \text{ and } O_j \subseteq I \text{ and } \forall (1 \le k \le J, 1 \le j \le J, k \ne j; O_j \cap O_k = \emptyset) \tag{5.9}$$

These clusters of local data are then used to train the local models. During forecasting, each consecutive input vector is assigned to exactly one local model according to the response of the SOFM (Hard partitioning).

Note that the proper number of clusters C must be chosen in advance. Appropriate settings for C were investigated in the experiments presented in section 5.3.

Further notes on the implementation of the SOFM as used in this experimental setup can be found in appendix C.

5.2.4. Local Models

The SOLEX architecture places no restriction on the types of local models used. A choice can be made based on knowledge about the properties of the modeled time series. If the time

series is linear, a linear ARMA model (section 2.3.2.) can be chosen. If the time series is known to be non–linear, a non–linear model is chosen such as the dynamic MLP of section 2.3.3. If speed is important Mean models can be used (section 2.3.1.). Mean models can only predict the mean target value for their associated cluster, but they have the property that they are very easy to build.

Training of the local models is performed after segmentation (clustering) of some amount of train data by the SOFM. A distinct local model is then constructed for each of these clusters of local train data.

No local ARMA models were implemented. Instead MLP models with linear output neurons (linear activation function) were used in the experiments.

5.3. Unsupervised Segmentation of Multistationary Time Series

A number of experiments were performed to investigate the quality of the unsupervised segmentation of multistationary time series by the SOFM. Also, these experiments were used to gain some insight into the appropriate settings for the three important parameters in the SOLEX architecture; N, the depth of the delay line used to split the available time series into a set of input vectors (or time windows, see section 5.2.1.); C, the number of clusters the SOFM creates and finally F, the type of feature extraction. More specifically these experiments were conducted to obtain a 'feeling' as to what settings produce a 'good' unsupervised segmentation of multistationary time series.

The problem is of course to define exactly what constitutes a 'good' segmentation. In earlier chapters it was already shown that this would imply stationarity of the clustered local data. The varratio measure was developed as a way to estimate the stationarity of local data, but this method is not yet proven so it cannot be used here as a means of estimating the quality of the segmentation provided by the SOFM (The quality of the varratio itself is investigated in the next chapter).

To overcome this problem, only synthetic time series are used in these experiments. Because the number of stationary regions, or *input classes*, is known for these computer generated time series, an indication of segmentation quality can be obtained by visual inspection of the results after clustering. Ideally one would like to see a 1 to 1 mapping between the activation of the stationary regions present in the multistationary input time series, and the resulting activation of the clusters by the SOFM trough time. Details about the various experimental time series used in the following experiments can be found in appendix A.

5.3.1. A Simple Multistationary Time Series

In this first experiment the effects of different settings of the delay parameter$N$ on the SOFM segmentation are investigated. To keep the problem small, a simple multistationary time series with known properties is used so that the C and F parameters can be fixed. A 'simple' multistationary time series was generated (see appendix A.). This time series switches between two regions with a different mean value every 100 steps for a total of 600 observations (Figure 5.2). Because of this, the C parameter can be fixed to 2 clusters and the appropriate feature F is *known* to be the *mean* statistic.

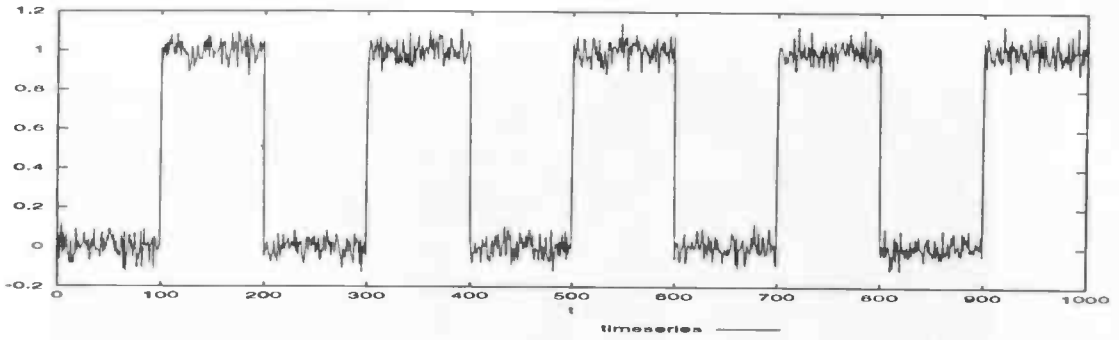

Figure 5.2 The 'Simple' Multistationary time series

Starting with a setting $N = 8$, 28 global trainsets were obtained using the TDL on the simple time series , increasing the amount of delay N with 2 with each run. These sets were then feature extracted with the mean statistic (Figure 5.3), leaving 28 feature extracted trainsets, each containing the mean values of the original time windows. The same procedure was followed to obtain 28 independent testsets using a different instance of the simple time series.

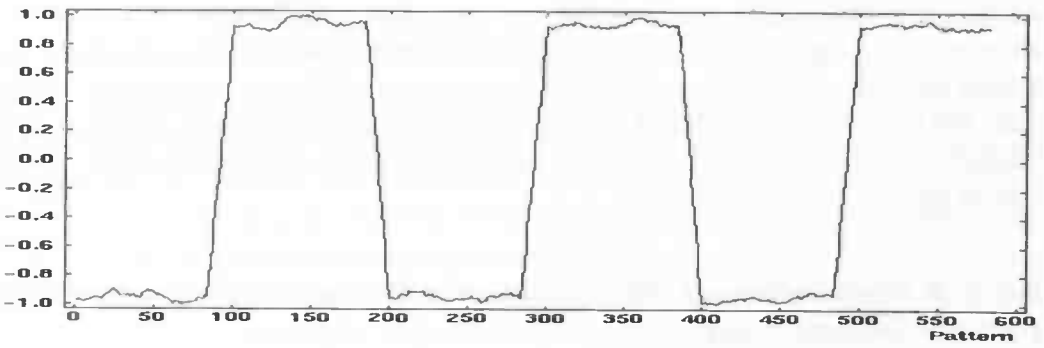

Figure 5.3 Mean statistic driving the segmentation

28 SOFMs with a 1x2 map (2 clusters) were constructed from the global trainsets, and the clustering was subsequently performed on the corresponding global testsets.

The visual results of this clustering can be seen in Figure 5.4, where the activation of the clusters in time is shown. This type of plot will be referred to as 'cluster activation' plot, and it shows both the activation of the stationary regions on the input of the SOFM (mkov), and the activity of the clusters as identified by the SOFM trough time.

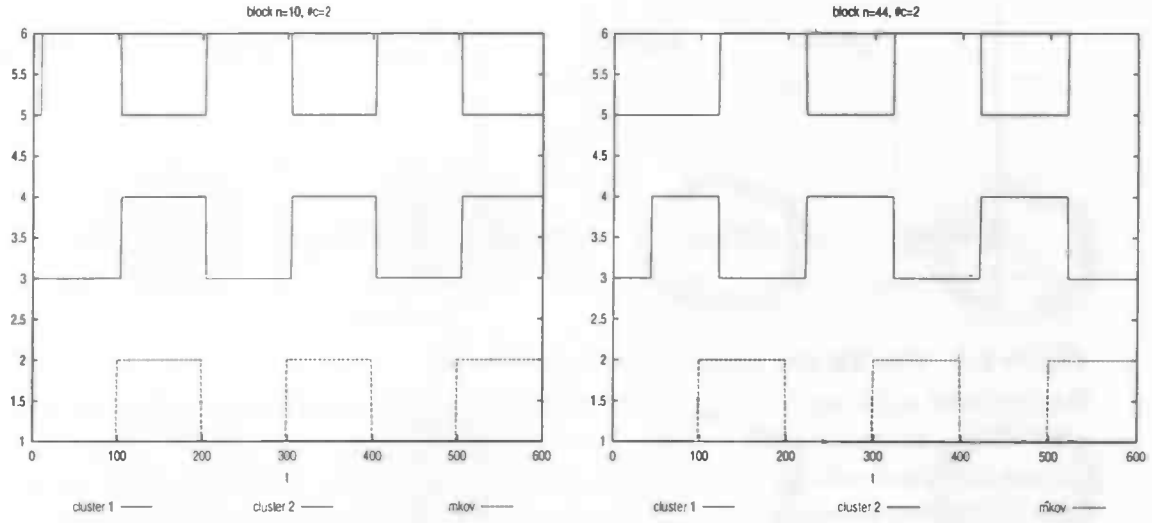

Figure 5.4 Cluster activation of simple time series for N=10 and N=44

As would be expected in this particular experiment, the SOFM is perfectly capable of finding the perfect segmentation for each value of N. The conclusion of this experiment is to use the lowest possible value of N as it leads to the shortest *lag* between the change in stationary region and the detection of this change by the SOFM. Note that exact mapping of the 2 input classes onto the 2 output clusters may be different for each instance of the experiment (As can be seen in Figure 5.4.).

5.3.2. A Multistationary Time Series with Overlapping Input Domains. (Chaotic Return Maps)

The next experiment looks into the effect of the 3 parameters N, C and F in a more complex setting, e.g. a multistationary time series with unknown statistical properties.

In this experiment the effects of different settings of these parameters on the SOFM clustering of the synthetic multistationary Chaotic Return Maps series (CRM, Figure 5.5) are explored. In this experiment the CRM series was generated with 4 possible stationary modes (classes). The CRM series in this experiment switches randomly between these modes every 512 samples, for a total of 8192 samples. It was made sure that both the train instance and the test instance of the series contained exactly 4 regions of each of the 4 stationary modes (To eliminate the possibility of an asymmetrical mapping of the 4 classes over the available SOFM clusters, caused by overrepresentation of one class over the others [17]).

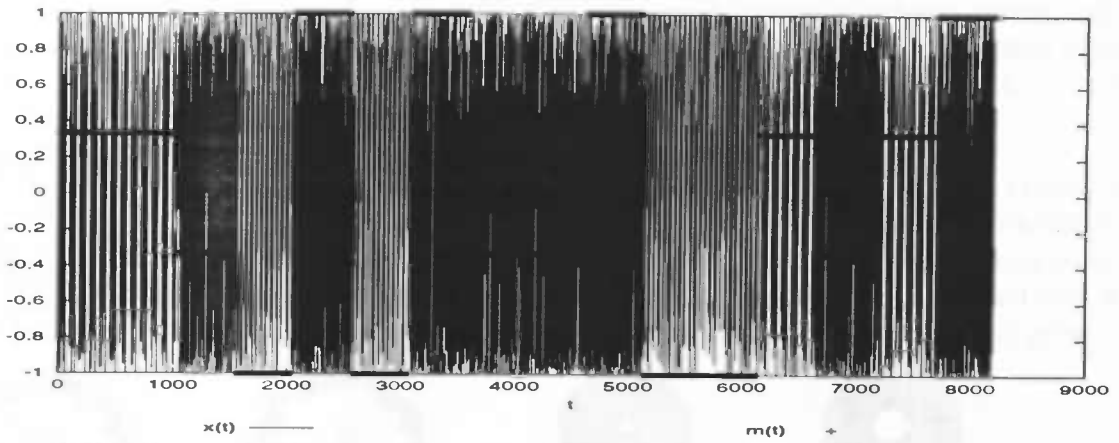

Figure 5.5 A typical instance of the multistationary Chaotic Return Maps series

First a choice was made as to what particular feature extraction F to perform. Because the CRM series is computer generated, the number of classes and their corresponding vectors in the trainset are known. Therefore it is possible to visually inspect *radarplots* of the trainset to check the separability of the 4 classes with different types of feature extraction. Note that this information is generally not available with real world series, but that it is used here to reduce the number of different parameters settings to check by fixing F.

Radarplots were made from a time delayed trainset (N=64) and the following types of feature extraction on this set (Figure 5.6); a) No feature extraction, the SOFM will cluster purely on spatial properties of the data. The radarplot shows almost completely overlapping input domains of the 4 classes. Consequently this would not be a good choice for F. b) FFT, the SOFM clustering will be on spectral properties of the data. This radarplot shows less overlap of the 4 classes in the frequency domain and c) Statistics, SOFM clustering based on the statistical properties; variance, skewness, kurtosis, median, rms, zero_cross and peak_count. This plot shows the best visual separability of the 4 classes. Therefore this type of feature extraction was chosen for the actual experiment.

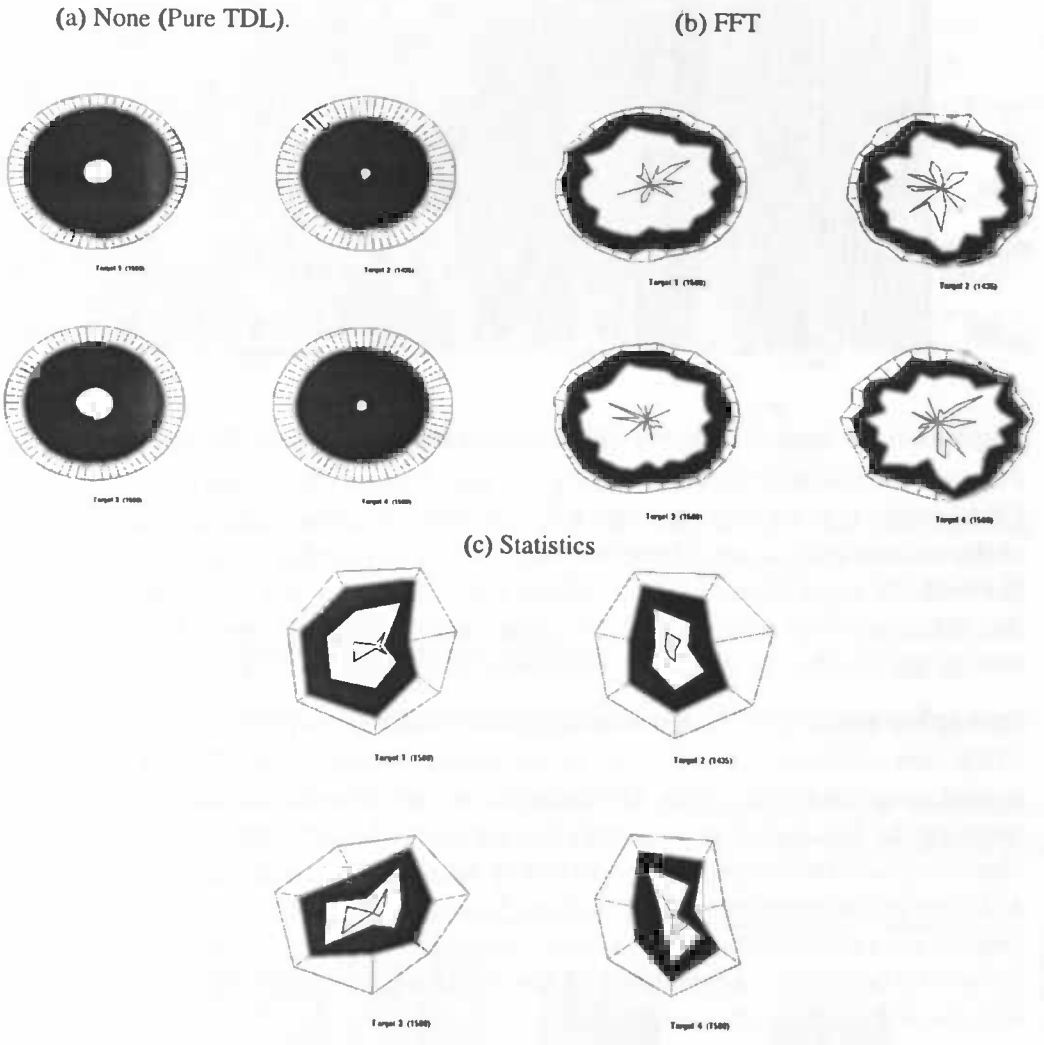

Figure 5.6 Radarplots of Feature Extraction on Chaotic Return Maps Timeseries

The actual experiment was performed in the same way as described in the previous section for the 'simple' time series, only this time both the N and C parameters were changed for each run. The clustering was performed 16 times, one run for each combination (N, C), with $N \in \{8, 16, 32, 64\}$ and $C \in \{9, 16, 25, 36\}$. N was chosen to be a power of 2 to speed up a cross–checking experiments with FFT feature extraction. C was chosen this way to have a square topological neighborhood for the SOFM, which leads to better clustering (this was found in a preliminary experiment on SOFM clustering, see appendix C.).

Visual inspection of the cluster activation of the results of the various runs of the experiment is not very usable as the number of clusters becomes quite large. Still the cluster activation plots for C=9 are somewhat informative (Figure 5.7). The left plot shows what is perceived to be a bad segmentation, where each of the 9 clusters are active in each of the stationary re-

gions of the input. The segmentation is visually better in the right plot, where a specialization occurs between the clusters. This is best shown by clusters 4, 5 and 6 which are (almost) only active for regions of class 4. Still this is not the ideal clustering where each individual input class is mapped exclusively to its own set of clusters. For instance, in the right plot, clusters 1,2,7 and 8 are at some point active for both class 1 and 3.

Another observation that can be made in the right plot is that if a set of clusters is exclusive for 1 particular region, these clusters will be activated in rapid succession during this region. From a modeling standpoint it would have been nicer if each of these cluster would be responsible for only one phase in the region, for instance one cluster that is activated only at the beginning of the region, one for the middle part, and one for the last part, just prior to the transition to the following region.

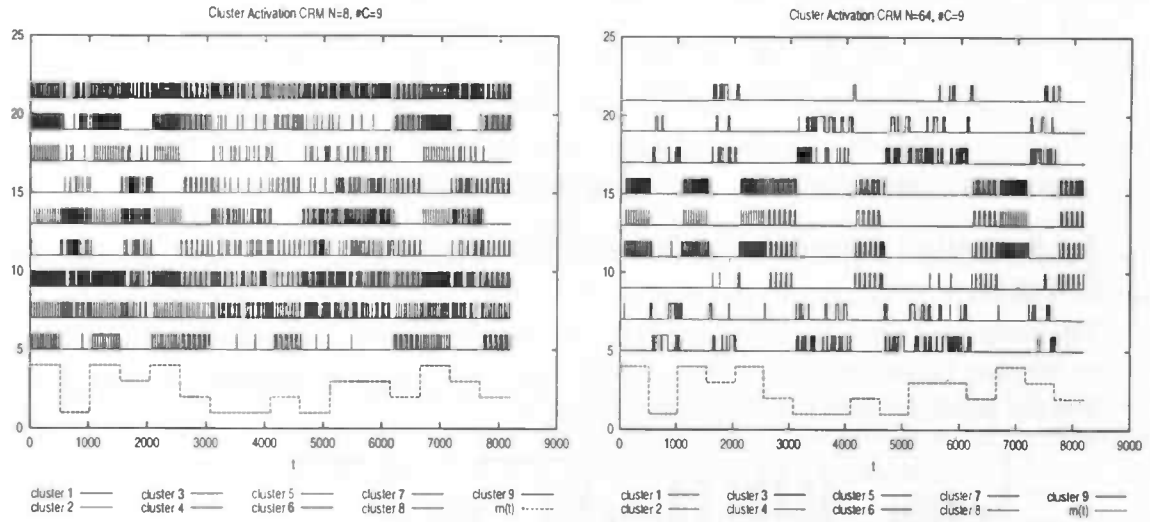

Figure 5.7 Cluster activation on Chaotic Return Maps series

Rather than looking at the activation of clusters in time, a better visualization is obtained in the following way; At each point in time exactly one cluster is active. Also, it is known what class is active at that particular time. Therefore a matrix can be calculated containing the number of times each cluster is active for each of the classes (for the whole testset). Then, at each time step, for each class, the number of times the current cluster is activated at some point for that class can be plotted relative to the total number of times the current cluster is active at some point in time. In this way, at each time step, for each class, a number is obtained with a value in the range from 0.0 to 1.0, providing an indication on the distribution of the currently active cluster over the classes. This type of plot will be referred to as 'class activation plot'. The advantage of this type of plot is that it can be used to directly compare the results of the various runs of the experiments where different numbers of clusters where used for each run.

From the 16 class activation plots obtained in this experiment, the two most important ones are shown in Figure 5.8. The left plot shows the results of the parameter setting which led to the worst segmentation (visually). The right plot shows the parameter setting which led

to the best segmentation. Comparison of all the plots leads to the conclusion that large values of N, and large numbers of clusters C are needed to obtain a good segmentation.

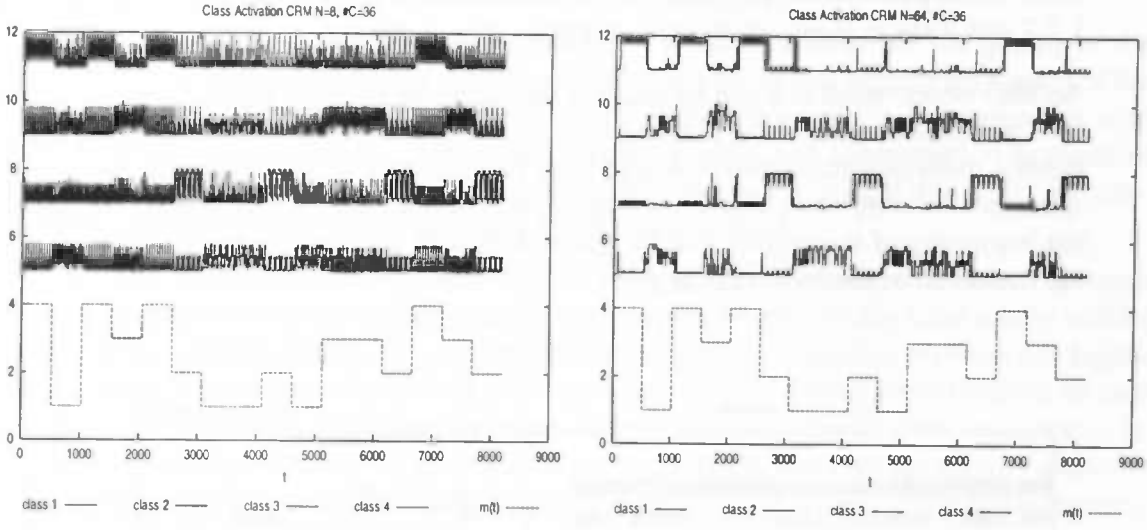

Figure 5.8 Class activation on Chaotic Return Maps series

5.3.3. A High Dimensional Chaotic Multistationary Time Series (Mackey–Glass)

The same experiment as described in the previous section was repeated for the multistationary Mackey Glass time series (Figure 5.9). The same feature extraction was used (Statistics), and the same settings for parameters N and C.

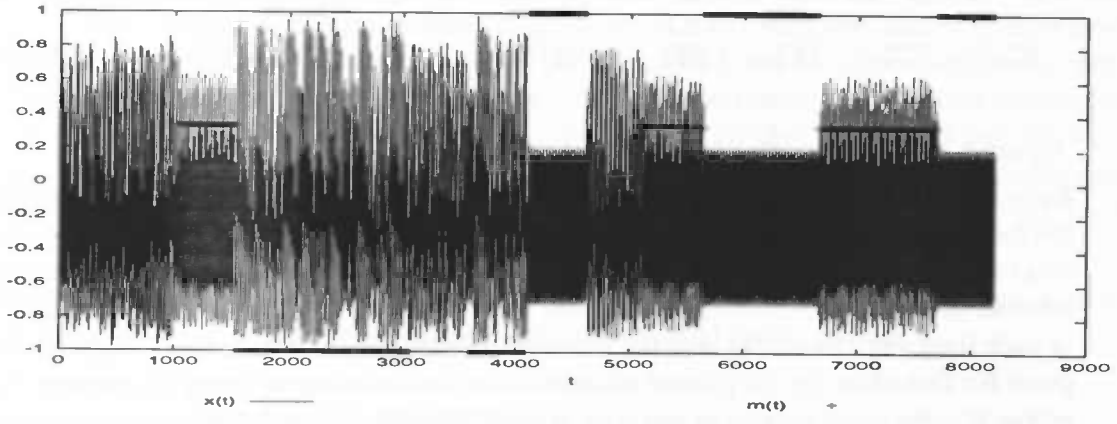

Figure 5.9 Instance of the multistationary Mackey–Glass series

Figure 5.10 Shows the class & cluster activation plots for the parameter setting $N = 64$ and $C = 9$ which led to the best segmentation.

From all the class activation plots, it can be concluded that on the Mackey–Glass series a large N is also beneficial to a good segmentation. In contrast to the CRM series, larger numbers of clusters did not improve segmentation quality.

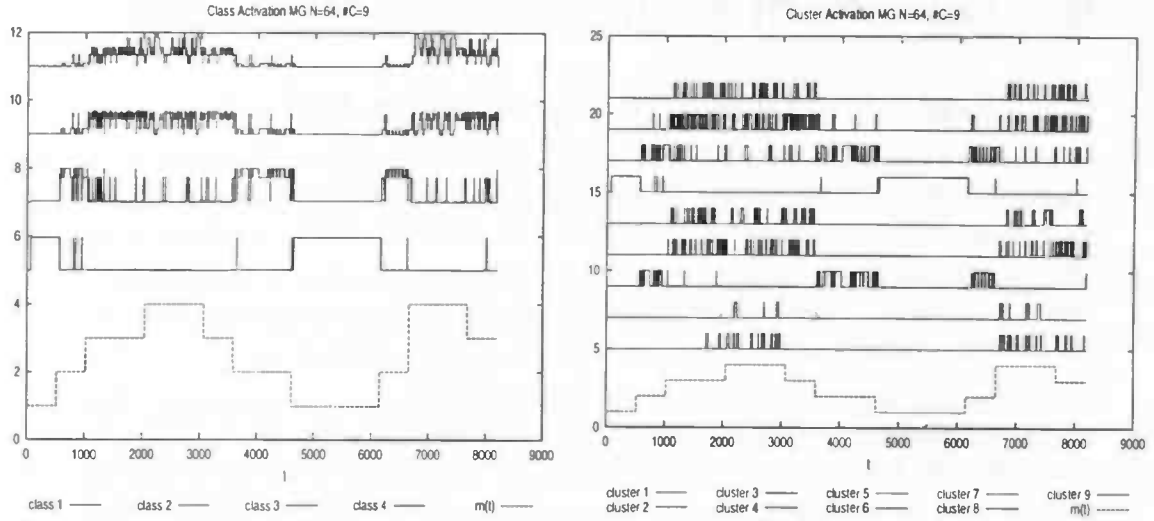

Figure 5.10 Class & Cluster activation on Mackey–Glass series

The overall conclusion that can be made after visually evaluating the effect of the various parameter settings on the SOFM segmentation is that large N improve the segmentation. This can be attributed to the feature extraction with statistics and the fact that these statistics will become more distinctive for each class if they are calculated for larger windows. The problem with larger N is that this leads to a longer delay between the switching of classes and the detection of this switch by the SOFM.

No general conclusions can be drawn on the right number of cluster C. On the CRM series more clusters improved the segmentation, but on the Mackey–Glass series it did not improve segmentation and the lowest setting of $C = 9$ produced the best segmentation.

The feature extraction with statistics turned out to be the best choice for both the CRM and MG series and a conclusion can be made that this would provide a reasonable default choice if no prior knowledge is available about the nature of the stationary regimes.

If the visual results on SOFM segmentation of multistationary time series are compared with results presented in literature, especially those of Kohlmorgen[11] and Reine, Zoeller [16] on the CRM series, it must be concluded that these results are rather disappointing. In contrast to these results of unsupervised SOFM segmentation, these articles present a perfect segmentation of the CRM series using their respective competition based supervised methods (see section 3.2.3.). Only in the case where the appropriate feature was known in advance (section 5.3.4., segmentation of simple series), was the SOFM able to achieve a perfect clustering.

Although the SOFM did not perform very well as a means of unsupervised clustering of time series, the data generated by these experiments can be used to test the quality of the varratio measure developed in chapter 4. As each instance of these experiments used a variety of settings for the parameters N and C, consequently a lot of different segmentations with varying stationarity of local data were generated. These datasets are used in the following chapter to test the quality of the varratio method.

6. Quality of Segmentation Measures

6.1. Evaluating Quality of Segmentation Measures

In chapter 4. the varratio method was introduced as a means to estimate the quality of the segmentation by measuring the stationarity of the local data. In this section the results of the experiments on the varratio method are presented. The purpose of these experiments is to show if the varratio method is a good way to measure quality of segmentation in modular time series models. Also, these experiments should provide some insight into which particular statistic would be appropriate to use in the varratio method.

The approach taken to test the varratio method is based on the assumption that if the local data in a modular time series model is stationary, the performance of the local model will be optimal. If the varratio method is a good method there must be a relation between the degree of stationarity it measures in local data and the performance of a local model trained on that data. In other words if a correlation between varratio and local model performance can be found, it is concluded that the varratio method is a valid way to measure stationarity and consequently a good way to estimate the quality of segmentation in modular time series models.

Besides being correct, the varratio measure must also be *stable*, that is, small variations in the actual segmentation of the time series should not lead to big variations in the varratios of the local data. Furthermore, the varratio should be independent of the type of local models and time series.

To perform the experiments, clustered local data is needed produced from a variety of segmentations (good & bad) and time series. For this the clustered data produced by the experiments on SOFM segmentation was reused (the time delayed clustered local datasets). The stationarity of the local data produced by the SOFM segmentation in these experiments will vary due to differences in the parameter settings of the SOFM. Also, this data originates from a number of different experimental sources (Simple, CRM and MG time series). Furthermore, in these experiments the segmentation was performed twice, once during training of the SOFM on one instance of the time series, and once on another instance of the series for the evaluation of the resulting segmentation. In this way, two independent local datasets were created for each cluster. These local datasets are reused here and will be referred to as 'local trainsets' and 'local testsets'.

Local models were then constructed for each available local trainset and the performance of these local model was measured with the Ratio of Squared Errors metric:

$$RSE = \frac{\sum_{n \in T} (observation_n - prediction_n)^2}{\sum_{n \in T} (observation_n - observation_{n-1})^2} \tag{6.1}$$

The RSE is calculated from the predictions the local model makes on the local testset. The RSE measures the performance of the local model relative to the performance of a 'virtual' reference model that takes the current observation as its prediction. The RSE will be smaller than 1.0 if the local model performs better than the reference, and larger than 1.0 if its performance is worse.

In the experiments, the RSE of the local models is also measured relative to the performance of a *global* reference model. This global reference model is trained on the global trainset (time delayed data of the whole time series). When the performance of a local model is evaluated on its local testset, the performance of the global model on this set is also measured. The RSE of the local model is then divided by the RSE of the global model. In this way a performance measure for the local model is obtained that is smaller than 1.0 if the local model performs better than the global reference, and larger if it performs worse.

Results are obtained by plotting the performance of any number of local models against the varratio of their corresponding local testset. Ideally, this would show a good positive correlation between local model performance and varratio.

The experiments were done individually for the three experimental multistationary time series Simple, Chaotic Return Maps and Mackey–Glass. In the following sections the results of these experiments are presented.

6.2. Correlation of Varratio with Local Model Performance

6.2.1. A multistationary time series with known properties

In this first experiment, the clustered local datasets resulting from the segmentation of the 'Simple' time series are reused (see section 5.3.1.). The stationarity of each of these local datasets differs as the delay parameter N was changed with each run of that experiment. Note that the segmentation experiment produced a total of 112 local datasets, consisting of an independent train and test set for each of the 2 SOFM clusters for each of the 28 different SOFMs.

Subsequently, a local averaging model (section 2.3.1.) was created for each of the 56 local trainsets. The averaging model type was chosen because it is the simplest model that still matches the modeling task (on the simple time series predicting the mean of the local region is the best any type of local model can do). Performance of these local models on the corresponding local test sets was expected to decrease for local data obtained by clustering with increasing values of the delay parameter N. The reason for this is that as N increased, more and more time windows in the global dataset spanned region boundaries, resulting in clustered local datasets containing more and more observations from both regions (regions with a mean of 0.0 and 1.0). as the local averaging model only predicts a single mean value for its cluster, its performance can be expected to decrease with increasing N.

Then the varratio was calculated for each of the 10 statistics of appendix B. and for each of the 56 local testsets. For each statistic, the varratios of the local testsets were plotted against the performance of the corresponding local averaging model of that testset. In this way any correlation between that particular varratio and local model performance will be easily visible from the plot. If the assumption is made that the varratio method is a good way to measure stationarity, then it is expected that the 'mean' varratio of the local testsets increases with larger N because the 'mean' stationarity of these sets is known to decrease with larger N.

The actual results of these experiments are shown in Figure 6.1 and Figure 6.2.

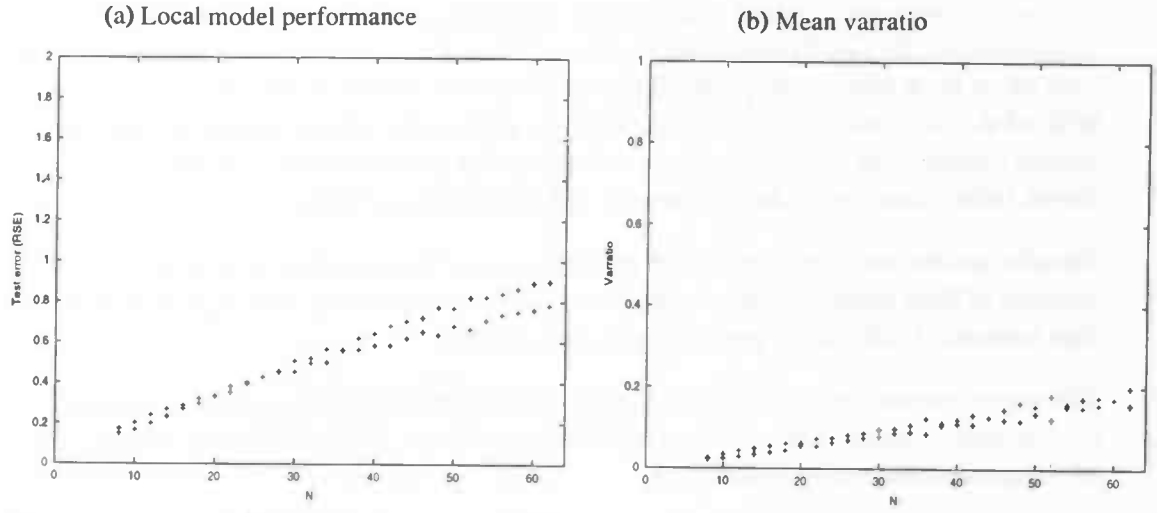

Figure 6.1 Local Model performance & Mean varratio vs. N

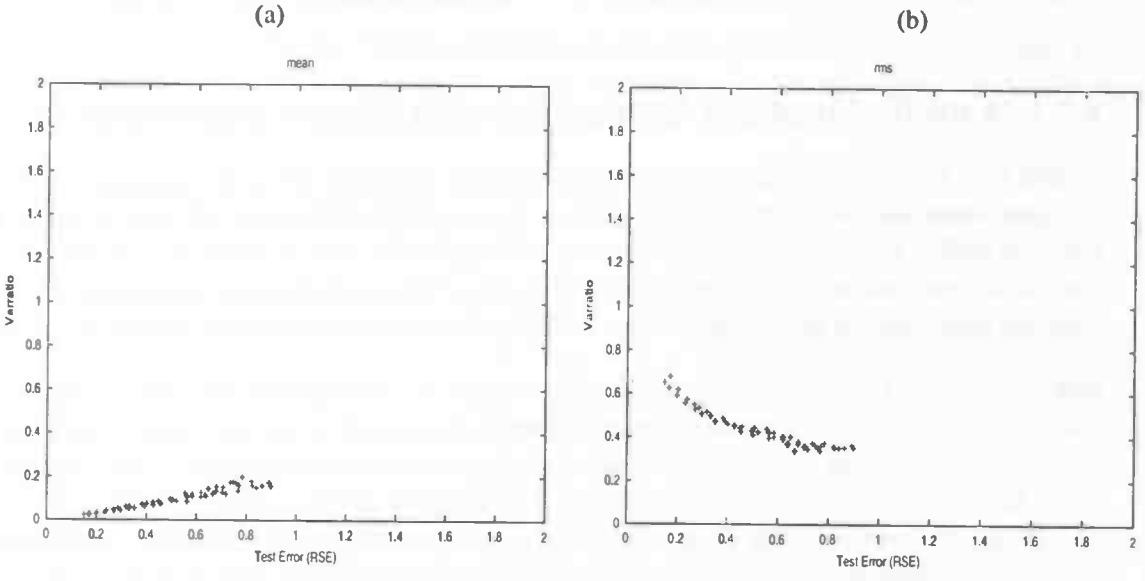

Figure 6.2 Mean and RMS Varratio vs. Local Model Performance

As expected, local model performance decreases, and the 'mean' varratio increases with larger N (Figure 6.1). Note that in these plots two 'lines' can be seen, diverging with larger N. This is caused by a slight overpresence of time windows with a zero mean, in turn caused by the initialization of the TDL with zeroes.

If local model performance is plotted against the mean varratio (Figure 6.2a), it shows that in this experiment, the mean varratio is a good measure of stationarity (high positive correlation between varratio and local model performance). In this experiment, it was known in advance with which statistic to measure stationarity (e.g. mean). The problem is that in general this is not known (chicken/egg). It is therefore informative to see what results would be obtained if another statistic would have been chosen. This is shown in Figure 6.2b, where the RMS varratio is plotted against local model performance. Again this shows a nice correlation between model performance and varratio, but in this case a negative correlation is seen, e.g.

as the RMS varratio indicates increasing stationarity, local model performance becomes worse. This was also the case when plotting the other statistics of appendix B. against local model performance. Either the the correlation was negative, or their was no correlation at all (noise).

The conclusion of this experiment is that it *is* possible to measure the stationarity of local data with the varratio method if the right statistic is can be chosen in advance. Another conclusion is that if the 'wrong' statistic is used in the varratio method, the resulting estimate of the stationarity of the local data may be entirely wrong.

6.2.2. Multistationary time series with unknown properties

In this experiment the clustered data generated from segmentation of the CRM series in section 5.3.2., and the MG series in section 5.3.3. is reused to see if any of the proposed varratio measures show a *general* correlation with local model performance (e.g.. no prior knowledge about the nature of the stationary regions).

The experiment consisted of training the three types of local models of section 2.3. (Averaging, Linear and MLP) for both the CRM and MG series, and for each of the clustered local trainsets generated in the segmentation experiments. The stationarity of each of these local datasets differs, as they originate from clustering with various SOFM settings ($N \in \{8, 16, 32, 64\}$ and $C \in \{9, 16, 25, 36\}$). In this way (9+16+25+36)*4 = 344 local models were trained of each model type, for each time series.

The performance of these local models was then plotted against the 344 varratios of the corresponding local testsets. A different plot was made for each of the 10 varratio statistics of appendix B.; mean, rms, variance, stdev, zero_cross, peak_count, autocorrelation, skewness, kurtosis and median, and each type of local model. Thus, a total of 60 scatter plots of varratio vs. local model performance were made this way (3 model types X 10 statistics X 2 time series).

To make comparisons easier, the correlation coefficient r between the varratio and local model performance was calculated for each of these plots. The correlation coefficient is a measure of the degree of linear relationship between a pair of random variables [3]. The correlation coefficient r is a value in the range [–1.0, 1.0], where $r < 0$ indicates negative correlation, $r = 0$ indicates no correlation and $r > 0$ indicates positive correlation.

Although the actual relation between local model performance and varratio could possibly be non–linear, the linear correlation still provides some indication on the predictability of local model performance from the varratio. In order to draw a positive conclusion on the suitability of a particular varratio as an estimator of local model performance, its correlation coefficient on the two series should at least be positive. Furthermore a high positive correlation would increase confidence in the suitability of the statistic.

The results of the experiments are rather disappointing. None of the scatter plots provides a visual indication of a clear relation between any of the statistics of the varratio method and local model performance. A selection of the best and worst plots based on the linear correlation they show is presented here. On the CRM series, the best linear correlation is found between the median varratio & linear local models. (Figure 6.3a), and the worst between the kurtosis varratio and local mlp models (Figure 6.3b).

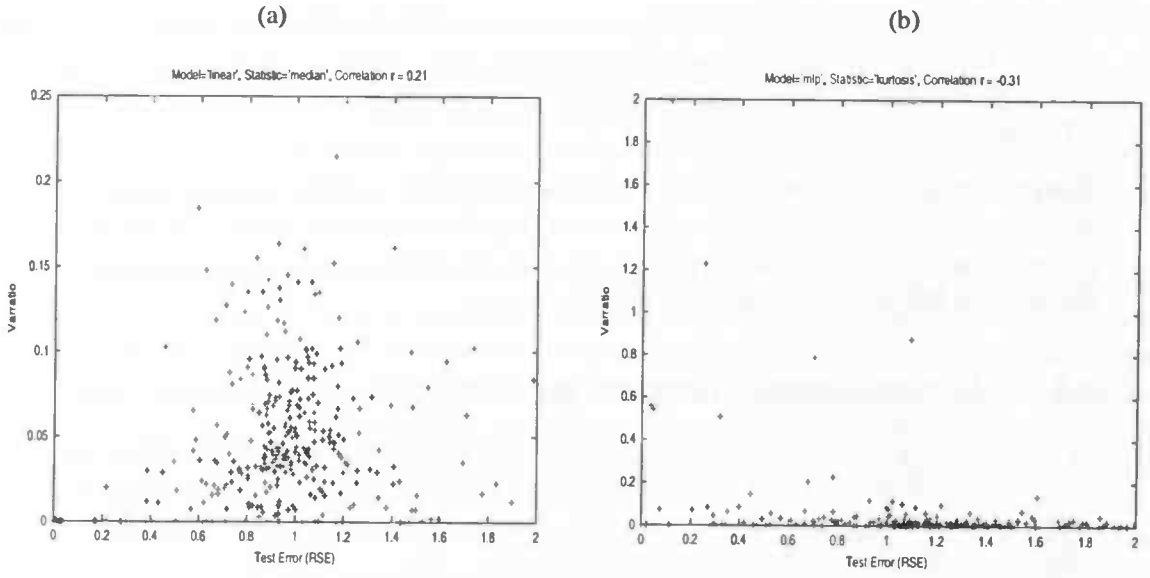

Figure 6.3 Correlation between varratio & local model performance on the CRM series

On the MG series, the best linear correlation is found between the median varratio & linear local models. (Figure 6.4a) and the worst between the mean varratio and mlp model (Figure 6.4b).

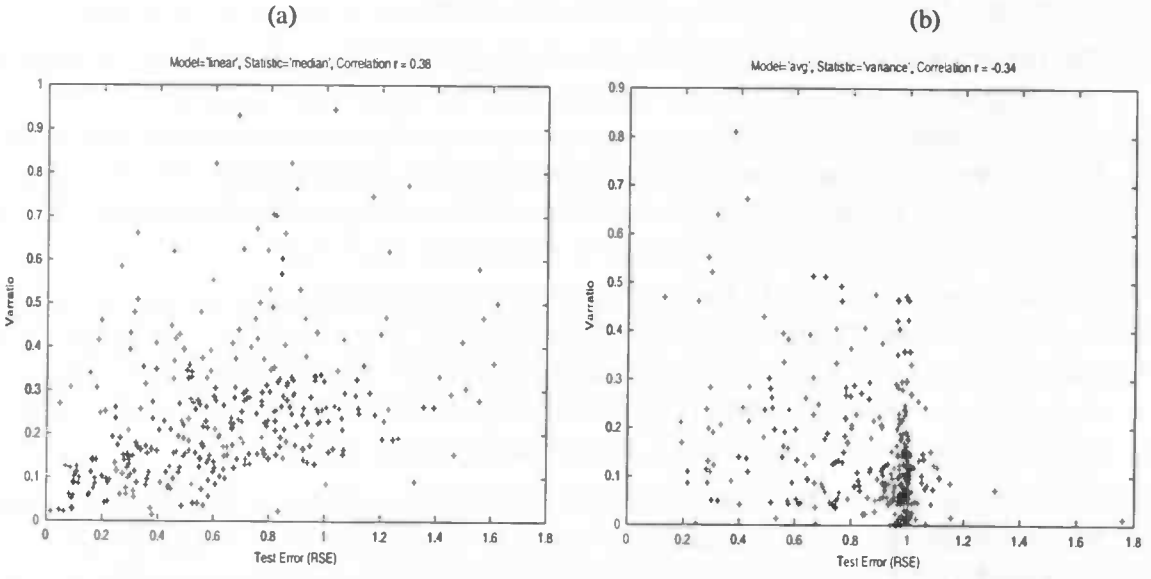

Figure 6.4 Correlation between varratio & local model performance on the MG series

Visually the results of these plots are disappointing, most of the plots showed little visible correlation between varratio and local model performance. To make comparisons more easy, the calculated linear correlations between all the varratios, time series and local model used in this experiment are shown in the following table:

CRM mean	CRM mlp	CRM linear	MG mean	MG mlp	MG linear
zero 0.14	mean 0.20	medi 0.21	auto 0.17	mean 0.10	medi 0.38

medi	0.12	medi	0.18	mean	0.18	zero	0.17	medi	0.08	auto	0.32
mean	0.12	peak	0.15	zero	0.18	peak	0.05	rmsq	0.03	rmsq	0.27
vari	0.12	zero	0.11	peak	0.17	medi	0.04	auto	0.03	mean	0.25
auto	0.12	rmsq	–0.08	auto	0.09	rmsq	0.02	vari	0.02	peak	0.20
peak	0.09	auto	–0.09	vari	0.08	mean	–0.06	zero	0.00	zero	0.17
skew	0.05	vari	–0.12	rmsq	0.06	skew	–0.12	stdv	–0.05	vari	0.13
rmsq	0.04	stdv	–0.25	skew	–0.03	kurt	–0.23	peak	–0.06	stdv	–0.03
stdv	–0.04	skew	–0.26	kurt	–0.09	stdv	–0.29	kurt	–0.07	skew	–0.06
kurt	–0.04	kurt	–0.31	stdv	–0.12	vari	–0.34	skew	–0.07	kurt	–0.16

Table 6.1. Linear correlation between varratio and local model performance

The same data is also shown more graphically in the radar plots of Figure 6.5:

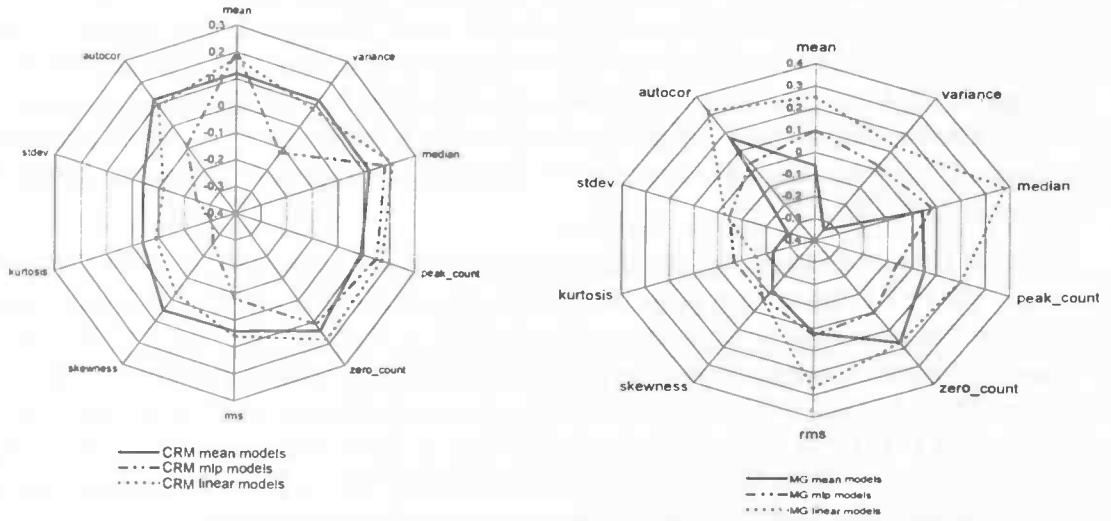

Figure 6.5 Radarplots of Linear correlation of the various varratio's

These results all show that almost none of the 10 statistics (measures of segmentation quality) show significant linear correlation between varratio and (future) local model performance. The exception is the correlation between the median varratio and the linear models on both time series. Although the median varratio showed best or second best correlation in of 5 out of 6 combinations of local model type and time series, the correlation between median varratio and mean and mlp models on both series is very low. Therefor, it may still not be considered as a generally usable estimate of segmentation quality.

The conclusion from these results must be that none of the 10 statistics provides a general estimator for local model performance on these two time series. Consequently neither of these statistics will provide a good estimate on the quality of segmentation on these series. It therefore not likely that any of these statistics would perform better on arbitrary real world multistationary time series.

6.3. Stability of Varratio

To further research the viability of the varratio method the following experiment was performed to check its stability. If the varratio method is a good method of measuring stationarity of local data, then it must also be stable. This means that slight variations in the stationarity of a local dataset must not lead to large variations in the measured varratio for the same local dataset.

The general setup of the experiment is to perform the SOFM clustering a great number of times on the same instance of a synthetic multistationary time series. Because the nature of this time series is known, the correct feature extraction, number of clusters C and delay parameter N can be chosen in advance, resulting in a perfect clustering each time the clustering is performed. Still, the stationarity of the clustered local data will differ slightly with each run, due to little variations in the initialization of the SOFM. Then the varratio of each of the local datasets of each run is calculated with the appropriate statistic and the variance among these varratios is calculated. If this variance is low, the varratio method is considered to be stable. Note that although it is difficult to specify how low this variance must be, it must at least be lower than the variance found in local model performance

Time Series: Using DTMF time series. Samplerate = 4000Hz because the highest DTMF frequency is 1633Hz and the samplerate should be at least two times higher (Nyquist). 16 Classes (All tones of the DTMF system), One region of every class in the timeseries. regionlength = 1024 samples, because window size N is fixed to 64. Regionlength should be large compared to window size to keep the number of windows that span region boundaries low, chosen 1:16 in this case). Total number of samples in timeseries is 16*1024=16384. Test data is same as Traindata.

TDL & Feature Extraction: Because the time series is multistationary in the frequency domain, the FFT is the most appropriate feature extraction. Time delay N is fixed to 64 Reason: The SOFM performs the segmentation on the FFT of the input windows. To be sure of a perfect segmentation we want the 8 frequencies in the DTMF data mapped onto separate inputs of the SOFM. It can be shown that for a samplerate of 4000Hz, and the 8 particular frequencies of the DTMF System, the first window size N where this is the case is 39. To be on the save side and to speed up the FFT N was taken to be 64. A *hamming* windowing function was applied to the windows (really needed, else bad segmentation).

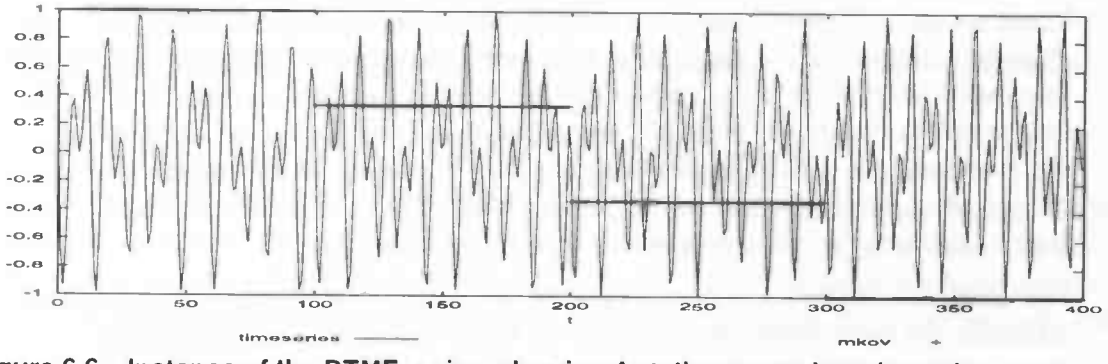

Figure 6.6 Instance of the DTMF series, showing 4 stationary regions (tones)

For the SOFM the number of clusters was chosen to match the number of classes present in the data (16).

25 Feature Maps were trained on this data, with 32 inputs (the power spectrum) and 16 clusters. For each feature map the varratios for every clusters were calculated resulting in 25*16=400 varratios per varratio statistic.

Results of unsupervised segmentation of the DTMF series: The SOFM assigns a cluster to the region boundaries, this could be seen on every instance. As a consequence some classes are mapped to the same cluster. Visual inspection revealed that these classes share a common frequency. Noteworthy is the fact that the SOFM assign a cluster to the switch boundary between the regions.

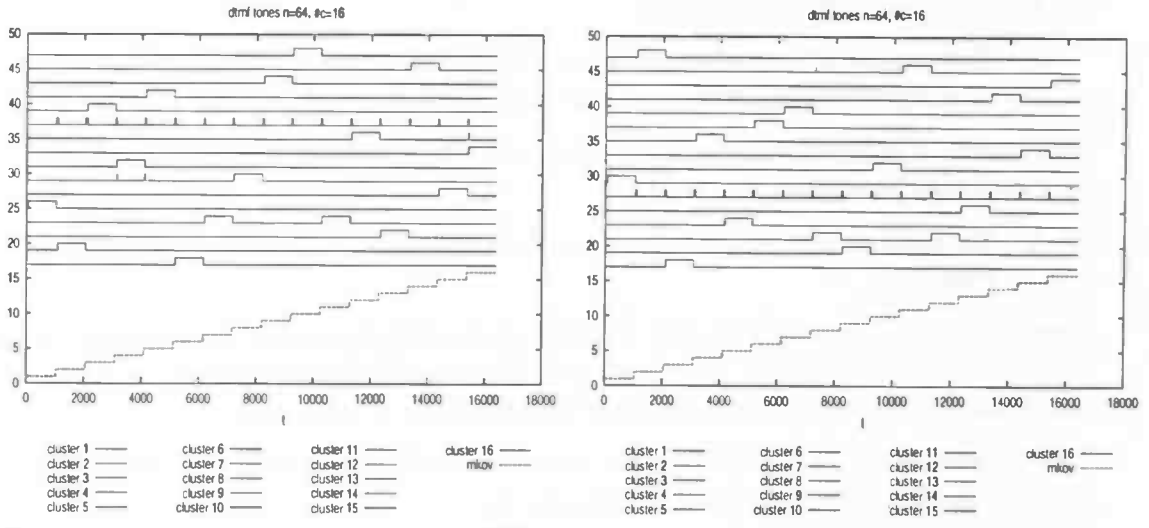

Figure 6.7 Cluster activation of 2 SOFM instances of DTMF series

Results of the stability experiments:

Statistic	Mean	Variance (% of mean)	Statistic	Mean	Variance (% of mean)
mean	0.97	22	peak_count	0.19	18
rms	1.04	64	autocorrelation	0.99	1
variance	1.04	65	skewness	0.97	137
stdev	1.04	65	kurtosis	1.02	147
zero_cross	0.76	35	median	0.86	117

Table 6.2. Varratio stability results

These results show that the zero_count and peak_count varratios are both <1 (indicating increased stationarity), and are more or less stable (low variance). These varratios can also be considered the 'appropriate' varratios given the nature of the multistationarity in the frequency domain. The varratios that are less appropriate are also less stable.

The conclusion is that this also indicates that the proposed varratios are not *general* enough to provide a means of estimating segmentation performance.

7. Conclusions & Future Research

It has been the goal of this thesis to find a way to estimate segmentation quality in modular time series models independent of local model performance, with the aim of breaking the interdependence between segmentation and local modeling.

In the course of achieving this goal, the following hypothesis were made:

1. The quality of segmentation in a modular time series model can be measured independently from the local models.

Based on the assumption that segmentation quality is good if the local data is stationarity, the following hypothesis was made:

2. The quality of segmentation in a modular time series model can be measured by directly estimating stationarity of local data.

And finally, in order to circumvent the chicken and egg problem (no non–parametric measure of stationarity exists, see chapter 1.):

3. Stationarity can generally be quantified by means of a number of traditional statistics.

Based on these hypothesis the new varratio method for measuring segmentation quality in modular time series models was developed. The varratio method allows a number of traditional statistics to be used as a measure of stationarity of local data.

The varratio method was put to the test in a number of experiments, the results of which are summarized according to the following criteria:

1. Correlation of Varratio with Local Model Performance. The varratios of the various statistics show little correlation with local model performance. Only if the appropriate statistics are known in advance (with respect to the nature of the multistationary time series), a significant correlation can be found.
2. Stability of Varratio. The varratio is only stable if the appropriate statistic is known.
3. Generality of Varratio. In each experiment on the correlation and stability of the varratio method, the best results were obtained if the nature of the multistationary time series was taken into account.

The results of the experiments (lack of correlation and generality) must lead to the rejection of hypothesis 3, thus:

Stationarity may not be quantified by means of traditional statistics.

The experiments did not provide enough evidence to reject hypothesis 2, e.g. It may still be possible to estimate segmentation quality by measuring stationarity of local data. This however, would require a better way to measure stationarity. Future research could find a non–parametric measure of stationarity, but given the results of this thesis, it may be unlikely that such a measure exists.

Hypothesis 1 may also hold, e.g. it may still be possible to measure segmentation quality independently of local model performance. Future research could include the possibility of us-

ing properties of the segmentation device as an estimate for segmentation quality. An example would be the 'map goodness' measure for Self Organizing Feature Maps as proposed by Ypma [22].

Thus, the overall conclusion of this thesis must be that:

It is not possible to generally estimate segmentation quality independently of local model performance by means of estimating stationarity of local data using traditional statistics. The newly developed measures of segmentation quality can therefore not be used to break the interdependence between segmentation and local modeling in modular time series models.

References

[1] A.C. Harvey. *Time Series Models*. Philip Allen Publishers ltd., 1981.

[2] R. van Asselt, a.o. *Wiskunde voor het hoger onderwijs 3*. Educaboek BV, 1991 (niet zo cool).

[3] Takeshi Amemiya. *Introduction to Statistics and Econometrics*, Harvard University Press, 1994

[4] Neil R. Euliano and Jose C. Principe. Spatio–Temporal Self–Organizing Feature Map *IJCNN96, IEEE/INNS Joint Conference on Neural Networks*, June 1996.

[5] Teuvo Kohonen. Self–organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69, 1982

[6] Jose C. Principe, Bert de Vries and Pedro Guedes de Oliveira. The Gamma Filter. *IEEE Transactions on Signal Processing*. 41 No. 2:195–201, February 1993. noref

[7] Mathew B. Kennel. Statistical Test for Dynamical Nonstationarity in Observed Time–Series data. *Physical Review E*, vol 56. nr 1, July 1997

[8] Ahn, Byung Chul. Testing the Null of Stationarity in the Presence of Structural Breaks for Multiple Time Series. *Unpublished*, 1994

[9] R.S. Venema. *Aspects of an Integrated Neural Prediction System*. PhD thesis, Department of Computing Science, University of Groningen, 1999.

[10] Andreas S. Weigend. Time Series Analysis And Prediction Using Gated Experts With Application To Energy Demand Forecasts. *Applied Artificial Intelligence*, 10:583–624, 1996.

[11] Klaus–Robert Mueller, Jens Kohlmorgen and Klaus Pawelzik. Analysis of Switching Dynamics with Competing Neural Networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E78–A, No.10, pp. 1306–1315.

[12] J. Kohlmorgen, S. Lemm K.–R. Mueller, S. Liehr, K. Pawelzik. Fast Change Point Detection in Switching Dynamics using a Hidden Markov Model of Prediction Experts. *ICANN'99. Proceedings of the International Coneference on Artificial Neural Networks*. pp. 204–209, 1999

[13] Ath. Keagias and Vas. Petridis. Predictive Modular Neural Networks for Time Series Classification. *Neural Networks*. Vol.10, No.1, pp.31–49, 1997.

[14] Ath. Keagias and Vas. Petridis. Time Series Segmentation using Predictive Modular Neural Networks. *Neural Computation*. Vol.9, pp.1691–1710, 1997.

[15] A. Bakirtzis, S Kiartzis, V. Petridis and A. Kehagias. A Bayesian Combination Method for Short Term Load Forecasting. *Electrical Power and Energy Systems*. Vol.19, pp.171–177, 1997.

[16] F. Reine, R. Zoeller. Characterization of Time Series By Dynamical Clustering. ?niet gepubliceerd?

[17] Simon Haykin. *Neural Networks, A Comprehensive Foundation* . Prentice Hall inc., 1994.

[18] J. McNAMES. Winning Entry of the K.U. Leuven Time–Series Prediction Competition. *International Journal of Bifurcation and Chaos*. Vol. 9, No. 8, pp. 1495–1500, 1998.

[19] R.A. Jacobs, M.A. Jordan, S.J. Nowlan, G.E. Hinton. Adaptive Mixtures of Local Experts *Neural Computation*., Vol 3, No. 1, pp.79–87, 1991.

[20] Michael I. Jordan and Robbert A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, Vol 6, No. 6, pp.181–214, 1994.

[21] Tong, Lim. *TAR Treshold autoregressive model*. 1980.

[22] A. Ypma, R.P.W. Duin, Novelty Detection using Self Organizing Maps, *Proceedings 3d Annual ASCI Conference*, pp.236–240, in: H.E.Bal et al. (eds.), June 1997.

[23] A. Doutriaux, D. Zipser., Unsupervised discovery of speech segments using recurrent networks. *Proceedings of the 1990 connectionist models summer school*, pp.303–309, 1990.

[24] M. van Veelen, J. Nijhuis, R. Venema and B. Spaanenburg, Neural network approaches to capture temporal information.?

[25] J.S. Bendat, A.G. Piersol *Random Data, Analysis and Measurement Procedures*. John Wiley & Sons Inc., 1971

A. Multistationary Time Series

In this appendix the synthetic multistationary time series are given that were used throughout the experiments.

A.1. Simple

The 'Simple' time series is introduced as an example of a very basic multistationary time series. The 'simple' time series is a block wave with some gaussian noise added ($\mu = 0, \sigma = 0.05$.). Every 100 steps the series alternates between two states (classes) with $\mu = 0$ and $\mu = 1$. See Figure 5.2 for a plot of an instance of this time series.

A.2. DTMF Tones

The 'DTMF' series is an example of a multistationary time series which is non–stationary in the frequency domain. The series simply consists of the consecutive 16 DTMF tones. Each tone consists of two sine waves. The tones were sampled at 4000Hz, and the duration of each tone is 1024 samples (region length).

A.3. Chaotic Return Maps

The Chaotic Return Maps series is introduced in an example of a multistationary time series with overlapping input domains. The process uses 4 *chaotic return maps*:

$$f_1(x) = 4x(1 - x), x \in [0, 1] \tag{1.1}$$

$$f_2(x) = \{2x, \text{if } x \in [0, 5) \text{ and } 2(1 - x), \text{if } x \in [0.5, 1]\} \tag{1.2}$$

$$f_3(x) = \varepsilon \tag{1.3}$$

$$f_4(x) = f_2 \cdot f_2 \tag{1.4}$$

where $f \cdot f$ denotes the iteration $f(f(x))$.

A multistationary timeseries is obtained by iteration of the following recursive definition:

$$x_{t+1} = f_i(x_t) \tag{1.5}$$

where f_i is chosen randomly from the set $\{f_1 f_2 f_3 f_4\}$ every 100 iterations.

To obtain a better segmentation from the SOFM, function (1.3) was changed from the original definition by kohlmorgen [11], $f_3(x) = f_1 \cdot f_1$, to the uniform random process $f_3(x) = \varepsilon$.

A.4. Mackey–Glass

The Mackey–Glass Equation is an example of a high dimensional chaotic system. It was originally introduced as a model of blood cell regulation and is used in numerous publications on time series forecasting as a testcase.

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t - \tau)}{1 + x(t - \tau)^{10}} \tag{1.6}$$

A time series is obtained by numerical integration of the Mackey–Glass delay differential equation (1.6) using the fourth order Runge–Kutta method. The model is then sampled at an interval λ.

To obtain a multistationary time series the delay parameter τ is switched every 100 samples. In the experiments the following values for τ have been used; $\tau = 10; 17; 23; 30$. These values are commonly found in literature [13], [11]. The sampling interval λ was set to 5.

B. Statistics

This appendix presents the various statistics used in the experiments are presented. These statistics were used as features to drive the segmentation of multistationary time series by the SOFM, and to measure the stationarity of the resulting clustered local data with the varratio method.

B.1. Time Domain

In the time domain relations between observations at different points in time are considered

The time domain statistics presented here are usually defined in terms of their expected value E over all possible realization of a time series. In practice only a limited number of observations are available and thus the real value of the statistic can be only estimated. The formulas presented here all show how to calculate the estimate for the particular statistic from a vector x of consecutive observations x_j with length N. Note that in this thesis this vector represents a time delayed window on the time series under consideration.

The Nth order moments are a set of statistics that describe a number of features of the actual probability distribution of a stochastic process;

The first order moment is the *sample mean* and it gives the average value of N observations:

$$Sample\ \ Mean = \mu \simeq \frac{1}{N}\sum_{j=0}^{N-1} x_j$$

The second order moment is the *variance*. It gives an estimate of the absolute average deviation of the samples from the mean:

$$Variance = \sigma^2 \simeq \frac{1}{N}\sum_{j=0}^{N-1}(x_j - \mu)^2$$

Closely related to the variance is the *standard deviation* σ. It is defined in terms of the variance:

$$Standard\ deviation = \sigma = \sqrt{variance}$$

The third order moment is the *skewness* of a set of N observations. It gives an estimate of the lack of symmetry in the joint probability distribution of x_i:

$$Skewness \simeq \frac{1}{N}\sum_{j=0}^{N-1}\left(\frac{x_j - \mu}{\sqrt{Variance}}\right)^3$$

The fourth order moment is *kurtosis*. kurtosis is a measure of how "fat" a probability distribution's tails are, measured relative to a normal distribution having the same standard deviation:

$$Kurtosis \simeq \frac{1}{N}\sum_{j=0}^{N-1}\left(\frac{x_j - \mu}{\sqrt{Variance}}\right)^4 - 3$$

Another important tool in time domain analysis is the *autocorrelation function* function $r(\tau)$. This function gives an estimate on the linear dependence of sample y_t on sample $y_{t-\tau}$. The

autocorrelation function $r(\tau)$ is derived from the autocovariance function $\gamma(\tau)$ in the following way:

The autocovariance function $\gamma(\tau)$ of a series y_t is defined by:

$$\gamma(\tau) = E[(y_t - \mu)(y_{t-\tau} - \mu)]$$

And is called the sample autocovariance $c(\tau)$ when estimated for a particular y_t using the equation:

$$c(\tau) = \frac{1}{N}\sum_{j=0}^{N-1}(y_i - \mu)(y_{i-\tau} - \mu)$$

The sample autocovariance function is normalized by dividing through the variance of y_t. This gives the autocorrelation function $r(\tau)$:

$$r(\tau) = c(\tau)/c(0)$$

Or:

$$r(\tau) \simeq \frac{\sum_{i=1}^{N-\tau}(y_i - \mu)(y_{i+\tau} - \mu)}{\sum_{i=1}^{N}(y_i - \mu)^2}$$

The autocorrelation as used in the experiments on the varratio method actually consisted of the autocorrelation at lag 1, or $r(1)$.

The *Root Mean Square* statistic gives an estimate of the average absolute value of x_t.

$$RMS = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x_i^2}$$

And finally the *median* statistic; Let the vector y be the result of sorting the time delayed vector x by value, then the median is defined by:

$$median = y_i \begin{cases} i = (N+1)/2, & N \text{ is odd} \\ i = N/2, & N \text{ is even} \end{cases}$$

B.2. Frequency Domain

In frequency domain analysis *cyclic* properties of a time series are considered. The most important analysis tool in this domain is *fourier analysis*. Fourier analysis is based on the fact that under conditions (Dirichlet), a function can be described by a linear combination of a possibly unlimited number of sine and cosine functions [2]:

$$x(t) = c + \sum_{n=1}^{\infty} a_n \frac{\cos 2\pi nt}{T} + \sum_{n=1}^{\infty} b_n \frac{\sin 2\pi nt}{T}$$

For discrete functions (time series), the coefficients a_n, b_n and c are found by applying the *Discrete Fourier Transform* to the series. In this way a time series can be decomposed into its frequency components.

Even in its fastest incarnation, the Fast Fourier Transform or FFT, the discrete fourier transform is rather expensive operation with a time complexity $\Theta(n \lg n)$. When speed is important the following two statistics are used to give a crude estimate on the base frequency of a series by counting the number of zero crossings or peaks in the series:

$$\textit{zero count} = \sum_{i=1}^{N-1} \begin{cases} 1 & x_i < 0 \wedge x_{i+1} > 0 \\ 1 & x_i > 0 \wedge x_{i+1} < 0 \\ 0 & \textit{otherwise} \end{cases}$$

$$\textit{peak count} = \sum_{i=1}^{N-1} \begin{cases} 1 & \nabla x_i < 0 \wedge \nabla x_{i+1} > 0 \\ 1 & \nabla x_i > 0 \wedge \nabla x_{i+1} < 0 \\ 0 & \textit{otherwise} \end{cases}$$

C. SOFM Implementation

To obtain some insight and gain confidence into the correct workings of the SOFM implementation, two small experiments were performed. The first experiment was conducted to check the basic clustering capabilities of the SOFM. The second experiment was performed to check the convergence to a stable mapping with the chosen training scheme.

Basic clustering ability of the SOFM implementation was checked with the following experiment. 4 random vectors (input prototypes or *classes*) of length 4 were created. These are shown in figure C.1a. Train & testsets were composed from these vectors, each containing approx. 500 instances of each of the 4 classes. 10% noise was added to each vector in the sets. Using the trainset, a SOFM with a linear map arrangement (1x8 output neurons = 8 clusters) was trained and the resulting 8 prototypes of the SOFM were plotted (C.1.b).

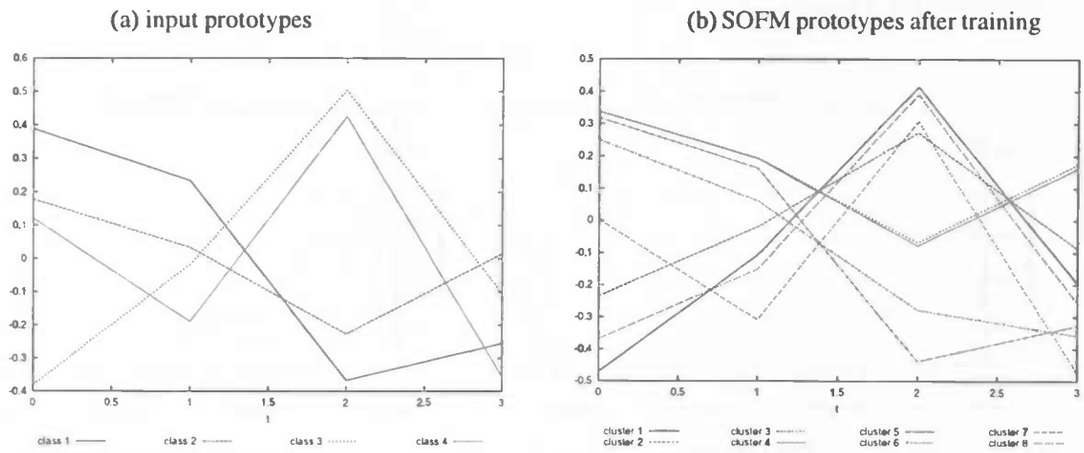

Figure C.1 Input & SOFM prototypes

The unequal distribution of map prototypes across the 4 input classes clearly shows the freedom the SOFM has in forming this mapping during training. The particular distribution is different each time the SOFM is trained. The results of clustering the testset are shown in the confusion matrix of table C.1 (classes on rows, clusters on columns):

	1	2	3	4	5	6	7	8
1	169	0	0	0	0	0	189	157
2	0	261	243	0	0	8	0	0
3	0	0	0	1	226	227	0	0
4	0	0	0	479	22	0	0	0

Table C.1 Confusion Matrix of Basic SOFM Clustering with Noise (10%)

This clustering shows only an insignificant misclassification of classes 4 into cluster 5, class 3 into cluster 4 and class 2 into cluster 6.

Given the simplicity of the given clustering task these results could be expected from a correct implementation of the SOFM. Therefore the conclusion is drawn that the implementation of the SOFM is correct.

In SOFM training there are 2 parameters that affect the convergence of the SOFM training algorithm; The learning rate parameter $\eta(n)$ and the width of the neighborhood $\sigma(n)$. These parameters are both dependent on time n *(epochs)* and must be slowly decreased during each consecutive epoch. The SOFM implementation used in this thesis follows the suggestion mentioned in [17], to exponentially decrease these parameters during the first phase of training (also referred to as the ordering phase) and then to hold these parameters at small values during a second phase.

What is not clear from literature is how many epochs of training are needed. Therefore a second experiment was performed; 4 different instances of the SOFM were used to cluster a trainset of the multistationary mackey–glass series with 4 stationary modes (or *classes*) into 4,8,12 and 20 clusters respectively. Each of the SOFMs was trained for 6, 30, 75, and 150 epochs and the Average Quantization Error (The average euclidian distance between the train vectors and the SOFM prototypes) was computed during each epoch, the results of which are shown in Figure C.2.

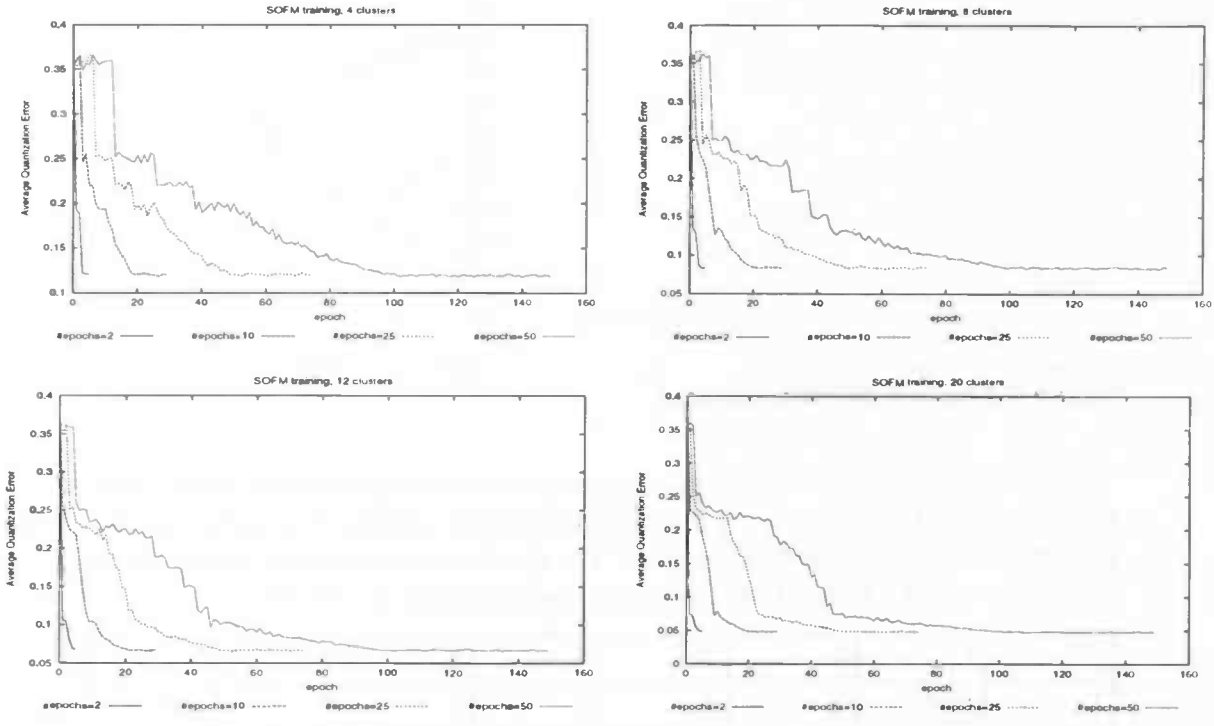

Figure C.2. Average Quantization Error vs #epochs in SOFM training

As expected, the final average quantization error decreases as the number of clusters available to the SOFM increases. More importantly, the final quantization error does not depend on the number of epochs used in training. From these results it the conclusion was drawn that 25 epochs of training is sufficient for the SOFM to converge. This number was then used throughout all the following experiments.

A final note on the form of the topological neighborhood of the SOFM implementation. The best results on clustering multistationary time series were obtained using a square, but 'folded' arrangement of the output neurons, e.g. a sphere.