Helsinki University of Technology Laboratory of Computational Engineering Publications
Teknillisen korkeakoulun Laskennallisen tekniikan laboratorion julkaisuja
Espoo 2003 REPORT B35

NEURAL NETWORK METHODS IN ANALYSING AND MODELLING TIME VARYING PROCESSES

Timo Koskela

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory of Computational Engineering Publications
Teknillisen korkeakoulun Laskennallisen tekniikan laboratorion julkaisuja
Espoo 2003 REPORT B35

NEURAL NETWORK METHODS IN ANALYSING AND MODELLING TIME VARYING PROCESSES

Timo Koskela

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Electrical and Communications Engineering, for public examination and debate in Auditorium S2 at Helsinki University of Technology (Espoo, Finland) on the 12th of December, 2003, at 12 noon.

Helsinki University of Technology
Department of Electrical and Communications Engineering
Laboratory of Computational Engineering

Teknillinen korkeakoulu
Sähkö- ja tietoliikennetekniikan osasto
Laskennallisen tekniikan laboratorio

Distribution:
Helsinki University of Technology
Laboratory of Computational Engineering
P. O. Box 9203
FIN-02015 HUT
FINLAND
Tel. +358-9-451 4826
Fax. +358-9-451 4830
http://www.lce.hut.fi

Online in PDF format: http://lib.hut.fi/Diss/2003/isbn9512268183/

E-mail: Timo.Koskela@hut.fi

ISBN 951-22-6817-5 (printed)
ISBN 951-22-6818-3 (PDF)
ISSN 1455-0474
PicaSet Oy
Espoo 2003

Abstract

Statistical data analysis is applied in many fields in order to gain understanding to the complex behaviour of the system or process under interest. For this goal, observations are collected from the process, and models are built in an effort to capture the essential structure from the observed data. In many applications, e.g. process control and pattern recognition, the modeled process is time-dependent, and thus modeling the temporal context is essential.

In this thesis, neural network methods in statistical data analysis and especially in temporal sequence processing (TSP) are considered. Neural networks are a class of statistical models, applicable in many tasks from data exploration to regression and classification. Neural networks suitable for TSP can model time dependent phenomena, typically by utilizing delay lines or recurrent connections within the network.

Recurrent Self-Organizing Map (RSOM) is an unsupervised neural network model capable of processing pattern sequences. The application of the RSOM with local models in temporal sequence prediction is presented. The RSOM is applied to divide the input pattern sequences into clusters, and local models are estimated corresponding to these clusters. In case studies, time series prediction problems are considered. Prediction results gained from the RSOM model show better performance than the model with conventional Self-Organizing Map. The RSOM can capture temporal context from the pattern sequence, which is useful in the presented prediction tasks.

As another application, a neural network model for optimizing a Web cache is proposed. Web caches store recently requested Web objects, and are typically shared by many clients. A caching policy decides which objects are removed when the storage space is full. In the proposed approach a model predicts the value of each cache object by utilizing features extracted from the object. Only syntactic features are used, which enables efficient estimation and application of the model. The caching policy can be optimized based on the predicted values and a cost model designed according to the objectives of the caching. In a case study, different stages and decisions made during the data analysis and model building are presented. The results gained suggest that the proposed approach is useful in the application.

Preface

This thesis for the degree of Doctor of Technology has been prepared in the Laboratory of Computational Engineering at the Helsinki University of Technology during the years 1997-2003.

I want to express my sincere gratitude to Prof. Kimmo Kaski who gave the opportunity to continue my studies and research, and for supervising the work throughout the years.

I'm greatly indebted to Dr.Tech. Jukka Heikkonen, my research supervisor, for giving advice, ideas and motivation to carry this work into conclusion, and to Dr.Tech. Markus Varsta, for providing the RSOM model and co-authoring the publications discussing its use. I'm grateful to Prof. Jouko Lampinen, for support and advice especially during the finalizing of this thesis. I also wish to thank Dr.Tech. Esa Alhoniemi and Dr.Tech. Panu Somervuo for reviewing the thesis and their suggestions and comments on the text.

My colleagues at the Laboratory of Computational Engineering deserve many thanks for the nice working atmosphere. Dr.Tech. Aki Vehtari and M.Sc. Arto Selonen are acknowledged for advice and many discussions not always related to research, and Ms. Eeva Lampinen for taking care of the practicalities.

My studies were partly funded by Graduate School in Electronics, Telecommunications and Automation (GETA), the National Technology Agency (TEKES) Grant 40687/00 (project title: *Internet Services Modelling*), and the Academy of Finland under the Finnish Centre of Excellence Programme during 2000-2005 (project title: *Research Centre for Computational Science and Engineering*, project number 44897).

Timo Koskela

Abbreviations

AIC	Akaike Information Criterion
AR	Auto Regressive
ARD	Automatic Relevance Determination
ARMA	Auto Regressive Moving Average
AR-SOM	Self-Organizing Map of competing Auto-Regressive models
bmu	best matching unit
BIC	Bayesian Information Criterion
BPTT	Backpropagation Through Time
CART	Classification and Regression Trees
CC	Cascade Correlation
DTW	Dynamic Time Warping algorithm
ECG	Electrocardiogram
EEG	Electroencephalograph
EM	Expectation Maximization method
FIR	Finite Impulse Response
GDS	Greedy Dual-Size
HMM	Hidden Markov Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIR	Infinite Impulse Response
IQR	Interquantile Range
LMedS	Least Median of Squares
LOO	Leave-One-Out cross-validation
LRU	Least Recently Used
LRV	Lowest Relative Value
MAP	Maximum A Posteriori
MCMC	Markov Chain Monte Carlo
MDL	Minimum Description Length
ML	Maximum Likelihood
MLP	Multi Layer Perceptron network
MLRE	Maximum Likelihood Robust Estimator

MSE	Mean Squared Error
NARMAX	Nonlinear Auto Regressive Moving Average with exogenous inputs
NARX	Nonlinear Auto Regressive with exogenous inputs
PCA	Principal Component Analysis
RANSAC	Random Sample Consensus
RBF	Radial Basis Function network
RSOM	Recurrent Self-Organizing Map
RTRL	Real Time Recurrent Learning
SOM	Self-Organizing Map
SSE	Sum Squared Error
TAR	Threshold Auto Regressive
TDNN	Time Delay Neural Network
TKM	Temporal Kohonen Map
TSP	Temporal Sequence Processing
URL	Uniform Resource Locator
WWW	World Wide Web

Contents

Chapter 1

Introduction

Statistical data analysis methods can be applied to gain understanding to a process under interest, which is too complex to be modeled analytically. Data from the process is first collected by measuring selected variables. These observations are then analysed by applying statistical methods to infer knowledge from the process. The results of the analysis depend on the prior knowledge about the process and the methods employed during the analysis. Essentially, all the choices made during the analysis, starting from the data collection, and ending to analysing the results gained from a statistical model have an effect on the inferred knowledge.

The main theme of this work is to consider neural network methods in data analysis. Neural networks constitute a class of statistical models, which can be applied in a broad range of applications, from exploratory data analysis and visualizations to classification and regression problems. In neural network modeling, the goal is to build a model based on the observed data, which can represent the essential properties of the process under interest.

The other main theme of this work is to present neural network methods capable of temporal sequence processing (TSP). Specifically, time series prediction is considered. A time series consist of measurements or observations of a natural, technical or economic process, that are made sequentially in time. The main motivation of the time series research is the desire to predict the future and to understand the underlying phenomena and processes of the system under study. The goal is to construct a model that can capture the essential features of the process by utilizing the available measurements and the prior knowledge. The model that can predict the future of a time series most accurately is usually considered to characterize the process best.

In chapter 2, data analysis and modeling with neural networks is reviewed. The process of statistical data analysis is presented, starting from data collection, moving to building the model, and ending at the analysing the model. In data collection section, preprocessing data and feature selection and extraction are discussed. In model building section, issues relating to model architecture, parameter estimation and model complexity are presented. In model analysis section, methods for model checking are discussed.

In chapter 3, neural networks in TSP are reviewed. Short review of traditional methods for TSP is first presented. Neural networks methods applicable to TSP, divided to unsupervised and supervised models, are then overviewed. The Multilayer perceptron network (MLP) and the Recurrent Self-Organizing Map (RSOM), which are employed in subsequent chapters, are also presented in more detail.

In chapter 4, case studies in TSP are presented. The models applied in the cases, including linear autoregressive model (AR), the MLP, and the RSOM with local models, are first presented. Application of the RSOM with local models in time series prediction is then proposed. Specifically, the RSOM learning algorithm is modified to facilitate learning temporal sequences of ordered input vectors. Also, the application of local models with RSOM to enable temporal sequence prediction is presented. Three time series prediction cases are then presented, which involve univariate one-step prediction problem. In these cases, the prediction performance of the RSOM is compared with other models. As a fourth case, multivariate one-step prediction problem is presented, where the properties of the RSOM model are studied in more depth. The results from the prediction cases show that the RSOM with local models can reach more accurate predictions than conventional SOM with local models. This suggest that the RSOM can capture certain temporal context from the pattern sequences which is useful in the presented tasks. Contributions of the author in this chapter include the proposed modification of the RSOM learning algorithm, implementation of the RSOM with local models to time series prediction, and performing all case studies and analysing the results.

In chapter 5, an application of a neural network model in Web cache optimization is presented. A short review of the cache optimization and previously presented methods is first given. The problem in cache optimization is to devise an optimal caching policy, which decides which objects to remove from the cache when it is full. Heuristic rules such as Least recently used (LRU) are still typically applied, but are not optimal for the Web cache. A novel approach to Web cache optimization is then proposed, where a model predicts the value of cache objects based on syntactic features collected from the objects. The caching policy can be optimized based on the predicted values and a cost model, which is designed according to the objectives of the caching. A case study is carried out to test the feasibility of the proposed approach. In the case study, Web objects are gathered based on real Web cache log. Two classifiers, MLP committee model and Generalized linear model, are employed to classify the objects based on their popularity. The syntactic features extracted are visually analysed with the SOM. The classification results show that the proposed approach can be useful in the case. The MLP committee can separate the classes based on the syntactic features more accurately than the Generalized linear model. The performance gain from the proposed model is demonstrated by running simulations with Web caches that utilize the classifier in the caching policy. Finally, some further considerations about applying the proposed model are discussed. Contributions of the author in this chapter include the proposed approach for Web cache optimization, and performing the case study where this approach is tested. This included analysing the collected data, building the models, and conducting the simulation studies.

Finally, in the last chapter a brief conclusion is drawn.

Chapter 2

Data Analysis with Neural Networks

2.1 Introduction

This chapter discusses data analysis with neural network methods. The goal of the analysis is to “understand” the system behind the observed phenomena, based on the available data and the prior knowledge of the system. In statistical data analysis methods and models by which we can infer knowledge from the observed data are used (Webb, 1999). These methods are typically applied when the phenomenon or the process under interest is too complex or unknown in order to be modeled analytically. The goal is to build a model that will capture the essential properties of the process, by using the observed data and the prior knowledge.

Artificial neural networks, or neural networks (NN) for short, can be viewed as a class of statistical models employed in data analysis and modeling (Bishop, 1995). Neural network methods are applicable in all areas of statistical modeling and analysis, but especially in classification and regression problems. Neural networks come in many forms and consequently there is no universally accepted definition. One could characterize them as biologically inspired models that typically consist of simple computing units connected in some manner, see e.g. (Haykin, 1994). Structure of the connections between units and the computation that a unit performs vary in different neural models.

Characteristic for neural networks is that the optimization of all the model parameters is carried out at the same time with a learning algorithm. When trained correctly, the mapping formed by the network can exhibit some capability for generalization beyond the training data. Also typical to neural networks and associated learning algorithms is that they are somewhat robust against redundant input variables or missing values in the training data. Neural networks are especially useful for classification and regression problems which are tolerant of some imprecision, which have lots of training data available, but to which heuristic rules (such as those that might be applied in an expert system) cannot easily be applied. Many different types of neural networks, such as Multilayer perceptron (MLP) (Rumelhart et al., 1986) and Radial basis function network (RBF) (Orr, 1996) have been proved to be universal function approximators (Cybenko, 1989; White, 1990), which means that almost any finite-dimensional vector function on a compact set can be approximated to arbitrary precision if there is enough data and enough computing resources.

The process of statistical data analysis can be divided in three stages. The first stage involves collecting the data set. It includes selecting the variables to be measured, normalising the data, selecting the features, and taking care of outliers and missing values. In the second stage the model is built using the data. Model family, training algorithm and optimising criteria are selected according to the objectives of the analysis. In the third stage, results from the model are analysed, and the model and features used are reviewed. If the model performs adequately, it can be implemented for its planned purpose.

The rest of this chapter is structured according to the process of the statistical analysis presented above. In the next sections, issues relating to data collection, model building, and analysing the model are discussed, focusing to neural network methods in statistical data analysis.

2.2 Data Collection

Statistical data analysis begins with the data collection phase. The choices made in collecting the raw data, preprocessing it, and making further choices on what methods to empoly in feature extraction or selection have their effect on the final results. Consequently, informed choices should be preferred instead of ad hoc methods. The choices made can be based on prior knowledge, or the results gained from using the model. Since the process from data collection to analysing the results involves quite many choices which affect the result, it is evident that the more prior knowledge can be exploited right from the beginning, the better.

The prior knowledge may include some expert's knowledge of the factors that have an effect on the modeled process. Having this knowledge can help in circumventing some problems in data collection, or help in making more sound choices. For instance, measurement of the variables may be difficult, time consuming or expensive, and a decision of which variables are included in the model is needed. On the other hand, there may be virtually unlimited amount of measurements available, and in order to limit the required storage space, decision on the sampling method employed must be made. Also, there may be some general information about the shape of the mapping (regression or classification) that should be applied, or some constraints that it should satisfy. If there is no knowledge of the properties of the process to be modeled, one has to begin with analysing the observations at hand, building a model and then analysing the results gained from the model. Also in this scenario prior knowledge is involved, that is the prior knowledge of the expert who is building the model.

Neural networks form a mapping from an input space to an output space. In order to perform this mapping, every part of the input space needs to be covered or represented. In the most general case, the amount of input data needed is proportional to the hyper-volume of the input space. Since the volume of the input space grows exponentially with respect to the number of input variables so does the amount of needed data. This unavoidable dependency is called the curse of dimensionality, referring to the exponential growth of hyper-volume as a function of dimensionality (Bishop, 1995). As a consequence, networks with lots of irrelevant inputs behave relatively badly: the dimension of the input space is high, and the network uses almost all its resources to represent irrelevant portions of the space. In general smooth, global functions are less sensitive to curse of dimensionality, while local functions are more sensitive. Also unsupervised learning algorithms are typically prone to this problem, as well as conventional RBF networks. A partial remedy is to preprocess the input in the right way, for example by scaling the components according to their "importance". However, importance of the input variables may be difficult to estimate. Another option is to try to reduce the dimensionality e.g. with Principal component analysis (PCA) method. Different methods for dimensionality reduction are applied during feature extraction and feature selection stages of the model building process.

Finally, the implementation of the model in a real application may raise issues which are usually not considered in more theoretical data analysis. In research and prototyping difficulties in data collection are often ignored. Also simplifying assumptions about data are made, which do not hold in the real environment. Often the measured variables are assumed to be independent of each other and measurement error distributions to be normal. Also the issue of missing values is often neglected altogether. In practice the data collection process is critical to the application and cannot be ignored.

In the rest of this section, issues relating to data collection are discussed. Different methods employed in preprocessing the data are presented first, after which feature extraction and selection methods are presented. Different topics are discussed in the typical order of the data collection process. However, also the data collection phase may be iterative in nature.

2.2.1 Preprocessing Data

Data preprocessing involves manipulating the data into a suitable form which can be modeled with neural networks. First step in preprocessing should be getting acquainted with the available data. Different methods for explorative data analysis (see e.g.Vesanto (2002)) can be applied in this stage. Typically, descriptive statistics such as mean, variance, and range of the variables are calculated. Also histograms and more advanced nonparametric methods (see e.g. Conover (1999)) can be useful. Different visualization methods should also be applied in this stage. For instance, measurement errors or outliers can be located from the data, in order to prevent problems in the normalising of the data.

Normalizing of the data is carried out next. This phase is dependent on the model and the nature of the modeling task. The distance measure, which measures the errors during the training of the model, and the normalising method applied for the data must match. Otherwise, the model may not be able to learn correct mapping from the data. For instance, in order to visualize the data with the Self-Organizing Map (SOM) (Kohonen, 1997), the input variables should be normalised to unit variance, since Euclidean distance is employed as the error metric. If the input variables are not normalised, the mapping realised by the SOM may be dictated by some variable which has a much larger variance than the others. Naturally, this is not the desired mapping of the data.

Variable encoding is required for the nonnumerical input variables since neural networks process only numerical information. Different models may require different encodings of the input variables. For instance, in classification tree algorithms (see e.g. Breiman et al. (1984)) it is usually possible to use unordered or ordered categorical variables as such, since the model can readily process them. In neural networks, however, typically applied training algorithms do not treat categorical variables appropriately. In essence, the encoding ensures a correct error metric to be produced within the training algorithm for different values of the nonnumerical variables.

Missing data refers to having missing values for certain input or output variables. There are several methods which can be employed for dealing with such situations. Depending on the amount of data and the nature of the modeling task, this issue can be either omitted by just dropping all samples which include missing values, or on the other extreme, using sophisticated methods for replacing the missing values with estimates. A typical approach for instance in time series prediction could be replacing a missing value of a variable by using its previously measured value, or an estimate calculated e.g. as a moving average over few samples.

The issue with non-typical data values, often called outliers, was already mentioned. The methods used for dealing with outliers are also dependent on the case and the nature of the modeling task. In some applications, dealing with the outliers may be essential in order to gain adequate results. On the other hand, in some cases the outliers might be omitted completely from the data. For instance, if the model was employed for classifying the quality of a product based on certain measurements from it, having an outlier in the measured variables could lead to manual inspection. In this case, the model only needs to recognize the outlier, and possibly give an estimate of the accuracy of the classification.

In the following sections, descriptive statistics, distance measures, normalising and encoding the data, and dealing with missing data and outliers are discussed in more depth.

Descriptive Statistics

Before the actual preprocessing, different statistical measures should be calculated from the raw data. These measures are needed in the following preprocessing steps. For each variable at least the measures describing the location and spread of values should be calculated. In most cases, estimates of first and second moment (mean and variance) are used as measures of location and spread, respectively. Estimating also higher moments e.g. skewness and kurtosis may be useful in some cases. Depending on the data, the effect of missing values and outliers should also be analyzed with appropriate methods. Methods which are resistant to outliers are often called robust.

Sample average is a simple and popular measure of location. If the data sample comes from a normal distribution, then the sample average is also optimal, since it is unbiased and consistent estimate of the true mean. Recall that an estimate is unbiased when its mean converges to the true value, and is consistent, when its variance tends asymptotically to zero. However, any real data is contaminated with outliers, data entry errors and glitches and the sample average is sensitive to these problems. The median and trimmed mean are two measures that are resistant (robust) to outliers. The median is the 50th percentile of the sample, which will only change slightly if a large perturbation is added to any value. The idea behind the trimmed mean is to ignore a small percentage of the highest and lowest values of a sample when determining the center of the sample (Venables and Ripley, 1999).

The range (the difference between the maximum and minimum values) is the simplest measure of scale. But if there is an outlier in the data, it will usually be the minimum or maximum value. Thus, the range is not robust to outliers. The standard deviation and the variance are popular measures of spread that are optimal for normally distributed samples. Neither the standard deviation nor the variance is robust to outliers. The Interquartile Range (IQR) is the difference between the 75th and 25th percentile of the data. Since only the middle 50% of the data affects this measure, it is robust to outliers (Venables and Ripley, 1999).

Distance Measures

Distance measures give a measure of similarity or dissimilarity between data samples (vectors). Distance measures satisfy the general axioms of symmetry, positive semi-definiteness, and triangular inequality (Theodoridis and Koutroumbas, 1999).

Let a and b be real-valued data vectors of dimension m, and d the distance operator. Distance d is symmetric, when $d(a, b) = d(b, a)$, and d is positive semi-definite, when $d(a, b) > 0$ if $a \neq b$, and $d(a, b) = 0$ if $a = b$. Distance d satisfies the triangular inequality when the distance between a and b cannot be shortened by going through c. Thus, the equality $d(a, c) + d(c, b) = d(a, b)$ holds when the point c is located on the line connecting a and b.

The most commonly applied distance for quantitative data is the *Euclidean* distance. Unweighted squared Euclidean distance is defined as

$$d^2(a, b) = \sum_j (a_j - b_j)^2 = (a - b)^T (a - b) \ , \tag{2.1}$$

where subscript T denotes the transpose of the column vector. The contours of equal Euclidean distance from a point are hyper-spheres (circles in two dimensions).

The *Manhattan* distance is defined as

$$d(a, b) = \sum_j |a_j - b_j| \ , \tag{2.2}$$

where $| \cdot |$ is absolute value. Manhattan distance can be useful e.g. when categorical variables are encoded by using binary representation. The contours of equal Manhattan distance from a point are diamonds in two dimensions.

The *Mahalanobis* distance is a generalization of Euclidean distance, defined as

$$d(a, b) = \sum_i \sum_j (a_i - b_i) \Sigma_{ij}^{-1} (a_j - b_j) = (a - b)^T \Sigma^{-1} (a - b) \ , \tag{2.3}$$

where Σ is the within-group covariance matrix (Webb, 1999). Mahalanobis distance measures the distance between groups of data points (distributions), and can be employed in classification as a class separability measure. In basic linear discriminant analysis, the covariance matrix Σ is typically assumed to be nondiagonal and symmetric, and same for each class. In a more general case, each class is assumed to have a different covariance matrix.

There exists other distance measures which can be useful in specific cases. In general, the choice of the distance measure depends on the application and may also depend on several factors including distribution of the data and computational considerations.

Normalising the Data

Normalising means adjusting a series (vector) of values according to some transformation function in order to make them comparable with some specific point of

Table 2.1: Encoding category variables.

Red	1	0	0
Green	0	1	0
Blue	0	0	1

reference (for example, a unit of length or a sum). Normalising is needed when the incompatibility of the measurement units across variables may affect the results (e.g., in calculations based on cross products) without carrying any interpretable information. It is also often beneficial to report the final results in a application specific meaningful/compatible units, thus normalising could also be useful in that sense.

Rescaling a vector means to add or subtract a constant and then multiply or divide by a constant, as in to change the units of measurement of the data, for example, to convert a temperature from Celsius to Fahrenheit.

Normalizing a vector most often means dividing by a norm of the vector, for example, to make the Euclidean length of the vector equal to one. In the neural networks literature, "normalising" also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1.

Standardizing a vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, the mean is first subtracted and the result divided by the standard deviation, thereby obtaining a "standard normal" random variable with mean 0 and standard deviation 1.

Variable Encoding

Variable encoding is performed for the nonnumerical input variables to facilitate processing with neural networks. Input variables can be divided to nominal, ordinal, interval, and ratio variables (Webb, 1999).

Nominal variables allow for only qualitative classification. They can be measured only in terms of whether the individual items belong to some distinctively different categories, but we cannot quantify or even rank order those categories. Typical examples of nominal variables are gender, race, color, city, etc. Categories are encoded using dummy variables. Each dummy variable is given the value zero except for the one corresponding to the correct category, which is given the value one. An example of encoding categories is shown in Table 2.1.

Ordinal variables allow to rank order the items measured in terms of which has less and which has more of the quality represented by the variable, but still they do not allow to describe "how much more". A typical example of an ordinal

Table 2.2: Encoding ordinal category variables.

Rich	1	1	1
Middle Class	0	1	1
Poor	0	0	1

variable is the socioeconomic status of families. An example of encoding ordered variables is shown in Table 2.2.

Interval variables allow not only to rank order the items that are measured, but also to quantify and compare the sizes of differences between them. For example, temperature, as measured in degrees Fahrenheit or Celsius, constitutes an interval scale. *Ratio variables* are very similar to interval variables; in addition to all the properties of interval variables, they feature an identifiable absolute zero point, thus they allow for statements such as x is two times more than y. Typical examples of ratio scales are measures of time or space. Most statistical data analysis procedures do not distinguish between the interval and ratio properties of the measurement scales.

Missing Data

Missing data refers to the values of variables within data sets which are not known. Although such cases that contain missing data are incomplete, they can still be exploited in data analysis. There are several approaches to this problem (Webb, 1999):

- Omit all incomplete vectors (patterns) from the analysis
- Use all available information, i.e. calculate estimates by using only those observations for which measurements have been made
- Substitute the missing values and proceed with the analysis as if the data set was complete

Omitting all incomplete data may be acceptable, but not if the data contains many observations with missing values. If missing data are randomly distributed across cases, one could easily end up with no "valid" cases in the data set, because each of them will have at least one missing data in some variable. Also, this approach implicitly assumes that the mechanism which is responsible for the omission of data is independent of the data itself. If this is not the case, then the effective data distribution will be modified by the case-wise deletion. However, the case-wise deletion of missing data is the only way to obtain a "true" correlation matrix, where all correlations are obtained from the same set of observations.

Using all available data for calculating the estimates is another approach. For instance, in estimating the correlation matrix, pairwise deletion of missing data is applied, where the correlation between each pair of variables is calculated from all cases that have valid data on those two variables. In many instances there is nothing wrong with that method, especially when the total percentage of missing data is low, and they are relatively randomly distributed between cases and variables (Webb, 1999). However, it may sometimes lead to serious problems. For example, a systematic bias may result from a "hidden" systematic distribution of missing data, causing different correlation coefficients in the same correlation matrix to be based on different subsets of subjects. In addition, real problems may occur when such matrices are subjected to another analysis (e.g., multiple regression, factor analysis, or cluster analysis) that expects a "true correlation matrix", with a certain level of consistency and "transitivity" between different coefficients.

Various methods exist to substitute missing data. In mean substitution, the missing data is replaced by the means for the respective variables during the analysis. This method is prone to serious problems, as the estimated mean may be biased. Also interpolation and extrapolation may be applied. In interpolation a curve is projected between known data points to infer the value of a function at points between. In extrapolation the value of unknown data points are predicted by projecting a function beyond the range of known data points. A more elaborate approach is to express any variable which has missing values in terms of regression over the other variables using the available data, and then to use the regression function to fill in the missing values (Bishop, 1995). This approach may cause problems since it underestimates the covariance in the data.

Outliers

An outlier is an observation which does not correspond to the phenomenon being studied, but instead has its origin in background noise or in a gross measurement (or assignment) error. In practice, nearly all experimental data samples are subject to contamination from outliers, a fact which reduces the real efficiency of theoretically optimal statistical methods. Methods which perform well even in the presence of outliers are called robust methods.

There are two different approaches for resolving the outlier problem (see e.g. Rousseeuw and Leroy (1987)). *Regression diagnostics* involves first identifying and removing outliers according to some rule of outlier rejection, and then reanalyzing the remaining data. This can also be an iterative process, sometimes called “build and criticize” approach. This approach may work well if the number of outliers is small.

In *robust statistics* the goal is to devise estimators that are tolerant of large residual values. After computing the robust estimate, the outliers are identified by examining the residual values. This approach usually works better than regression diagnostics when the number of outliers is large. One of the best known methods is least median of squares (LMedS) (Rousseeuw and Leroy, 1987). Other methods include M-estimators (Huber, 1981), where the least squares error function is replaced with another function. Also outlier removal methods based on Random sample consensus (RANSAC) (Fischler and Bolles, 1981) are popular especially among computer vision applications. Here minimal configurations are randomly sampled, and the one which maximises the number of inliers that fall inside some predefined threshold is selected.

In (Brandt, 2002) a Maximum likelihood robust estimator (MLRE) was proposed. It was shown to be asymptotically optimal in the sense that it reaches the same estimate that would have been achieved without contaminants (false observations) in the data. Here outliers are defined to be residuals of false observations, so that outliers can overlap with the residuals of good observations and their distribution is not assumed to be symmetric. The problem is now to estimate the residual density of the model. First, residual distributions for good and false observation are set to some initial value according to a priori assumptions. Thus, each residual is given a probability of belonging either to good or false observations. Then a Maximum likelihood (ML) estimator is fitted to the data, where the probability of the sample belonging to the good observations is used as a weighting factor in the error metric (negative log likelihood of the data). The residual distributions are then updated by applying Bayes' rule, and the procedure is iterated until convergence. For instance, a Gaussian mixture model, where one Gaussian models the distribution of good residuals, and the rest (one or several) model the outlier distribution can be applied. The number of Gaussian functions can be selected e.g. by Minimum description length (MDL) (Rissanen, 1996) principle, and the parameters of the model are estimated by Expectation-maximisation (EM) (Jordan and Jacobs, 1994) type iterative algorithm.

2.2.2 Feature Extraction and Selection

Collected data must often be processed further in order to get better results. Redundant variables may be removed, or higher level characteristic variables may be calculated based on them.

The objective is to represent the data in a reduced number of dimensions. Reasons for doing this may be easier subsequent analysis, improved classification or prediction performance through more stable representation, removal of redundant or irrelevant information, or an attempt to discover underlying structure by obtaining a graphical representation.

Given a set of measures, dimensionality reduction can be achieved in essentially two different ways. The first approach is to identify those variables that do not contribute to the modeling task. For instance, in a classification task, those variables that do not contribute to class separability may be neglected. Thus, the task is to seek k variables out of the available m measurements. This is termed feature selection in the measurement space or simply *feature selection*.

The other approach is to find a transformation from the m measurements to a lower-dimensional feature space. This is termed feature selection in the transformed space or *feature extraction*. This transformation may be linear or nonlinear combination of the original variables and may be supervised or unsupervised. In the supervised case, the task is to find the transformation for which a particular criterion is maximised (e.g. in classification case, certain separability measure).

It should be noted that both the feature extraction and feature selection are highly dependent on the application, and may also be dependent on the model. Different methods applied in feature extraction and selection are discussed in the next sections.

Feature Extraction

Collected data often includes redundant variables, which should be removed. Also to avoid the curse of dimensionality, number of variables used in the model must be kept as small as possible (Theodoridis and Koutroumbas, 1999). Feature extraction is the process of generating fewer, higher level variables than the data itself for the modeling purpose. This is achieved by using some transformation (linear or nonlinear) from the original variables to features. Extracted features may be general features, which are evaluated to ease further processing, or application-oriented, like those needed for image recognition. For instance, extracting edges in an image (sharpening) is an example of a general algorithm, whereas identifying the boundaries of individual chromosomes in medical imaging is an application-dependent example.

Principal Component Analysis in Feature Extraction

Principal component analysis (PCA) is commonly applied in feature extraction phase to transform the feature vectors to orthogonal coordinate system, and to select those that are used in the model (Webb, 1999). PCA produces an orthogonal coordinate system in which the axes are ordered in terms of the amount of variance in the original data for which the corresponding principal components account. Dimension reduction is gained when only those axes that account for the most of the variation are selected. New feature vectors are then gained by transforming the original feature vectors onto this lower dimensional space.

Derivation of the principal components is reviewed shortly next. Consider a set of N observed data samples on each of the m variables, denoted $X = (x_1, ..., x_N)$, where each x_i is an m-dimensional (feature) vector. Let μ_x be the estimated mean and C_x the estimated covariance matrix of the same data set. The eigenvectors e_i and the corresponding eigenvalues λ_i are solutions of the equation

$C_x e_i = \lambda_i e_i, i = 1, ..., n$. Eigenvectors can be solved e.g. from the characteristic equation $|C_x - \lambda I| = 0$. The equation becomes of order m, which is easy to solve only if m is small. In a more general case, solving eigenvalues and eigenvectors is a non-trivial task, and many methods exist. Suppose that eigenvectors have been solved, and let A_k be a matrix having the k first eigenvectors (in the order of descending eigenvalues) of the covariance matrix C_x as the row vectors. Each vector x_i can then be projected on the coordinate axes having the dimension k as $y_i = A_k(x_i - \mu_x)$. By selecting the eigenvectors having the largest eigenvalues as little as possible information in is lost in the mean-square sense.

It should be noted that the transformation gained with the PCA is dependent on the normalisation of the original data. If the data is concentrated in a linear subspace, using PCA provides a method to compress data without losing much information and simplifying the representation. However, since PCA is a linear transformation, it may be inadequate in nonlinear data.

Self-Organizing Map in Feature Extraction

The SOM (Kohonen, 1989, 1997) is one of the best-known neural networks in the unsupervised category. The role of the SOM in feature extraction is to construct optimal codewords in abstract feature spaces. Individual feature values can then be replaced by these codes, which results in data compression. Furthermore, hierarchical systems can be built in which the outputs from the maps are again used as inputs to subsequent layers. The topological property of the feature maps is then essential for low-error performance (Lampinen, 1997).

SOM belongs to the class of *vector coding* algorithms. In vector coding, the problem is to place a fixed number of vectors, called codewords, into the input space which is usually a high-dimensional real space R^m. The input space is represented by a training set $X = (x_1, ..., x_N) \in R^m$. An essential extra feature in the SOM is that the neurons are arranged to a 1-, 2- or multidimensional *lattice* such that each neuron has a set of neighbours. The goal of learning is not only to find the most representative code vectors for the input training set in the mean square sense, but at the same time to realise a *topological mapping* from the input space to the grid of neurons. For example, the inputs can be gray-scale windows from a digital image, measurements from a machine or a chemical process, or financial data describing a company or a customer. The dimension m is determined by the problem and can be large.

Each codeword will correspond to and represent a part of the input space: the set of those points in the space which are closer in distance to that codeword than to any other codeword. Each such set is convex and its boundary consists of intersecting hyperplanes. This produces a so-called Voronoi tessellation into the space. The overall criterion in vector coding is to place the codewords in such a way that the average distances from the codewords to those input points belonging to their own Voronoi set are minimised. This is achieved by learning algorithms that are entirely data-driven and unsupervised. With the SOM the vector coding criterion can be satisfied when the neighbourhood is set to zero at the end of the learning phase. This quantization phase fine tunes the locations of the code vectors. Coding facilitates data compression and makes possible postprocessing using the discrete signal codes. Typically, the codewords are found to correspond to relevant clusters among the input training data. There may be one or several code vectors corresponding each cluster.

An example of an application which employs SOM in feature extraction is face recognition, where microfeatures from an image are clustered by the SOM is presented in (Lampinen and Oja, 1995). Another example is autonomous robot control (Heikkonen and Koikkalainen, 1997), where the input vector is combined of both sensory inputs and corresponding control signals of the robot. These vectors are then clustered by the SOM. Third example is speech recognition (Kangas, 1994), where certain time-dependent spectral features are extracted from the raw speech signal. The SOM creates a representation of the spectral relations between the speech samples. Essentially, certain regions in the map space correspond to certain phonemes of the speech. The mapping realised by the SOM can be used in visualization of the spectral patterns and in speech recognition. In order to recognize words and sentences, the trajectory of the best matching unit on the map space as a function of time is processed further.

Feature Selection

Different feature selection procedures can be applied when the number of available features is too large to be used in the model. Also as noted before, using too many features is inefficient due to the curse of dimensionality.

The problem is to find the best subset (of size k) of features from the available m variables. Thus, transformations of the variables are not considered here, only selecting those m variables that contribute most to model predictions or classifications. The performance of the selected features is evaluated based on the available data, and consequently supervised methods are employed. For instance in a classification case, class separability gained by the selected features can be used as a performance metric. Feature selection methods can be divided to categories based on the subset generation method, and the selection criteria for the best subsets (Dash and Liu, 1997).

The methods applied in generation of the subsets can be further divided to three categories: complete, heuristic, or random. Using the complete set of features is usually intractable, since there are $2^m - 1$ possible subsets of features in a set of size m. For instance branch-and-bound (Theodoridis and Koutroumbas, 1999) selection method belongs to this category. Usually some heuristics are used to search a small but hopefully interesting fraction of the space of all subsets of features. Forward and backward selection, reviewed shortly in the next section, are popular heuristic feature selection methods. Third option is random selection of features. For instance, genetic algorithms and simulated annealing methods belong to this category (Dash and Liu, 1997).

The selection criteria for finding the best subset from the generated subsets can be divided to two broad groups. In the first group belong all the distance metrics, class separability metrics, information measures, and dependence measures which are calculated from the subset of features, independently of the model applied. In the second group belong the metrics based on the performance of the model with the selected subset. Methods for estimating the performance of the model after the model has been built are discussed in the last section of this chapter. Essentially these methods are based on estimating the generalization ability of the model which includes specific subset of features, by using the available data.

For feature selection with neural networks, prior knowledge of useful features can also be gained by first analysing the features with simpler models. For instance, neural networks may be difficult to train with large number of features. An option is to first estimate a linear model or a classification tree for the task with complete subsets of features, and select the best subset of features based on the model performance. The subset of features selected by the linear model may also prove to be useful with neural network model. Some methods useful in feature selection are briefly presented in the next sections.

Forward and Backward Feature Selection

Forward selection and backward elimination are the simplest heuristic rules for feature selection (Theodoridis and Koutroumbas, 1999). Forward selection starts with an empty subset to which one feature (the one which most reduces the error) is added at a time, until some chosen criterion stops decreasing. Backward elimination starts with the full subset from which one feature (the one which least increases the error) is removed at a time, until the chosen criterion stops decreasing.

The criterion for stopping is usually the error rate, since minimum expected classification or prediction error is the main objective in model design. Error rate can be estimated with a separate test data set, which is not used during the model building, including the feature extraction or selection phases. More often, though, all data is needed in model building, and consequently cross-validation or bootstrapping methods are applied to estimate the error rate. These methods are discussed in more depth in Section 2.3.3.

Class Separability Metrics in Feature Selection

Another approach for feature selection is to compare directly the separability of different feature sets. This approach can be useful in classification cases, since separability of the classes can be easily verified (Webb, 1999). The overlap between the distributions from which the data are drawn are estimated for each class. Those feature sets for which this overlap is minimal (maximising separability) are then favoured.

The approach has the benefit of being independent of the model. There exists a plethora of different heuristic measures of separability. These include different generalizations and extensions of Mahalanobis distance measure presented previously. In practice, finding a useful separability measure may be difficult, since real data may not fit well the presumed distribution. Furthermore, it is often computationally intensive to calculate the separability measures for all different feature sets. Therefore, in many practical situations, these methods cannot be applied effectively.

Classification Trees in Feature Selection

Classification or decision trees are capable of modeling complex nonlinear decision boundaries. An overly large tree is usually first constructed by using the training data, and then pruned to minimise a cost-complexity criterion. The resulting tree is easily interpretable and can in some cases provide insight into the data structure.

Classification and regression trees (CART) (Breiman et al., 1984) is a popular classification tree and algorithm. In CART, feature selection is an integral part of building the model. The building of the tree usually starts with a single root node which contains all the training data. For this node a rule that maximises the selected criterion is chosen. For instance, in classification case, all possible divisions of data based on the value of an input variable are explored. The selected rule then divides the training data to two child nodes. Building of the tree then continues until there is no data left to divide (all samples in a node belong to the same class), or the number of samples in one node is smaller than some predefined number.

In some cases, where the number of input variables is large, and it is assumed that only few can be used in practice, it may be helpful to build a classification tree in order to explore the importance of the variables. Furthermore, CART can include an alternate rule in each node based on using another input feature when the value of the preferred feature is missing. This technique, called surrogate splits, can be applied in classification of the sample in case of missing values. Thus, surrogate splits provides information about interchangeable features, which may be useful.

2.3 Model Building

Building the statistical model is perhaps the most complicated task in the data analysis. It is often an iterative process, as the model is first build, then tested and analysed, then tuned and built again.

The goal in neural network training is not to learn an exact representation of the data itself, but rather to build a statistical model of the process which generates the data. The effective flexibility or complexity of the model must be selected carefully, so that the best generalization is achieved. Generalization error can be decomposed into the sum of *bias* squared plus the *variance* (Geman et al., 1992). Too inflexible model in relation to the particular data set will have a large bias, while too flexible model will have a large variance. Bias and variance are complementary quantities, and consequently the best generalization is a compromise between small bias and small variance. In practice, it is often useful to start with a simple model, analyse the results, and then move on to more complicated models if the results are not satisfactory.

The process of building the model can be divided into three parts: Selecting the model architecture, estimating the parameters of the model, and choosing the complexity of the model. Since all three are dependent on each other, the process of model building becomes a cycle. In the next sections the phases of this cycle are discussed.

2.3.1 Model Architecture

Model architecture includes the model family, number and type of input and output variables, and other variables which affect the mapping which the model performs. As discussed previously, inputs to the model can be generated in a separate process, where general features and application specific features are first extracted. Usually the model is known when the features are extracted, and thus normalising the data and selecting the features can be carried out to suit the model. Output of the model depends on the application. For instance, when the model is a classifier, it may have separate output variables corresponding each class. For each input, the class with the largest output value is then selected. On a more sophisticated model, each output variable may instead give an estimate of the posterior probability of the class. Probability estimates gained for each class may then be used in subsequent decision making, for instance with associated cost model.

The model family can be represented by the functional form (or the mapping from the input to the output) which the model performs. The problem how to choose the model family M cannot be adequately formalized. In practice, the selection is based on human judgment and prior knowledge of the kinds of models that have been employed in the past, perhaps by other researchers.

Parametric Models

In traditional statistical modeling parametric models are frequently applied. Commonly used model class consists of parametrised probability densities

$$M = p(X|\theta), \theta = (\theta_1, ..., \theta_k) \ , \tag{2.4}$$

where $X = (x_1, ..., x_N)$ contains N observed data samples. θ contains the parameter values, and k is the number of estimated parameters for the model.

The observed sample X is assumed to be drawn from random variable χ, which has an unknown distribution. This unknown, "true" distribution is then approximated by the model class M. An estimator $\hat{\theta}$ for parameter values is calculated by using the observed data.

In general, parameter value optimization can be carried out in several different ways, which are discussed in Section 2.3.2. The most common estimation procedure, however, is Maximum likelihood (ML) estimation.

The tradition for using parametrised distributions is rather long, and properties of some models are well known. Commonly employed parametric models include Gaussian distribution, and others from exponential distribution family.

Nonparametric Models

Nonparametric modeling includes no assumptions about probability distribution which generated the observed data. Thus, there is no predetermined set of parameters to be optimized, rather the estimate is determined directly from the data, and the number of parameters in the model grows with the number of data. Examples of nonparametric methods include histograms, i.e. tabulating the data, kernel estimates and nearest neighbour estimates (see e.g. Conover (1999)).

Nonparametric methods are often computationally quite demanding. Often they also suffer considerably from the curse of dimensionality. For instance, consider approximating the function $f(x)$ by tabulating the values of $f(x)$ for x with m dimensions. Each dimension m is divided to n fragments with size s. The maximum interpolation error is now $e \leq sL$, where L is given by the differential $L \geq |\frac{\partial f}{\partial x}|$. In order for the approximation error to remain bounded when the dimension grows, the number of data samples needed is n^m.

Semiparametric Models

Certain types of neural networks can be regarded as being semiparametric models. For instance mixture (distribution) models, the MLP and the RBF networks, and certain regulated nonparametric models belong to this class (Bishop, 1995).

In semiparametric models the goal is to estimate the underlying probability distribution with a generic function (an universal approximator). Assumptions about the the underlying probability distribution, however, are less strict than with parametric models, since the models can represent a large number of potentially useful mappings for the observed data. Thus, the model is fitted to the data such that the effective number of parameters depends on the characteristics of the underlying distribution. The effective number of parameters is usually less than the number of available parameters and thus it corresponds to the complexity of the model (Moody, 1992).

Neural networks in general belong to nonlinear models. The functional form and consequently the complexity of the mapping realised by the neural network model may vary depending on the parameter values of the model. MLP network is one of the popular neural network models (Bishop, 1995). The architecture of the MLP consists of a description of how many layers it has, the number of neurons in each layer, each layer's activation function, and how the layers connect to each other. The best architecture to employ depends on the type of problem to be represented by the network. Typically, sigmoidal function is used as the activation function in the neurons. Thus, when using small weights in the network, the mapping results to almost linear function which has a low effective complexity. This is because the central region of sigmoidal activation function can be approximated by a linear transformation. The MLP is discussed in more depth in Section 3.3.1.

2.3.2 Parameter Estimation

Parameter estimation is the second phase is model building. In parameter estimation, the output of the model come to approximate the target values given the inputs in the training set. If the model is fitted adequately, it will have the ability to generalize to cases which were not included in the training set. This means that the outputs of the model approximate the target values given inputs that are not in the training set.

Generalization is not always possible, and it requires prior knowledge or assumptions to build a model with this ability. There are three necessary conditions to yield a model with generalization ability (Sarle, 1997b). First condition is using relevant inputs in the model, second is having some restricted class of input-output functions that contains an adequate approximation of the function to model, and third is that the cases to be generalized bear some resemblance to the training cases.

The parameter values for the model are estimated by using the collected training data. Thus, a criterion for selecting the best values for the parameters is needed. A popular selection is using Maximum likelihood (ML) estimates. For instance, least squares is a popular optimization method applied in linear regression, which can be derived from ML principle. Minimising the sum squared error corresponds to the ML principle, when the residual error distribution for the data is assumed Gaussian (Bishop, 1995). Other choices for parameter estimation include using Maximum a posteriori (MAP) principle or Bayesian inference. Issues relating to the training algorithm, and different methods for estimating the parameters are presented in the next sections.

Training Algorithm

Parameter estimation of the model is carried out with a training algorithm, which by using the training data gradually optimizes the parameters (or weights) in the model. In other words, the training algorithm implements the search method which estimates the values for the parameters of the model.

In linear models the parameter values can be estimated from the training data by using linear algebra. There may be situations when insufficient or noisy data prevents estimating the parameters, but this situation cannot be improved by the training algorithm. For nonlinear models, there only exist iterative algorithms by which the parameter values are estimated.

Gradient descent optimization is one of the methods for estimating the parameter values. The gradient of the error with regard to the parameters of the model is estimated with current parameter values. The parameter values are then corrected in the direction of the gradient, thus gradually driving the model toward the optimal parameter values. Common problem with the gradient descent optimization is difficulty in evaluating whether the model is in a local or in a global minimum. In practice, applied algorithms include different methods which try to prevent from getting trapped to local minima.

Depending how the algorithm corrects the parameters, two general types of algorithms are applied, namely stochastic gradient and batch learning. In stochastic learning algorithm an input vector is chosen randomly from the training set, output of the model is calculated, and based on the error (calculated as the difference between the output and desired target output), the weights of the model are corrected slightly toward the gradient. In batch learning algorithm the approximate gradient is evaluated for the entire input data at the same time, which allows more accurate estimate of the gradient.

Supervised and Unsupervised Learning

Training algorithms can be further divided to unsupervised and supervised algorithms. In supervised learning, the model estimates a mapping from input space to corresponding output space. Training data contains samples from this mapping, including representative input vectors and desired output vectors. Training algorithm presents these samples to the model, and applies some optimization method to estimate the parameters of the model in order to minimise the selected error metric. In this case, the error metric is computed from the difference between the model output and the desired output.

In unsupervised learning, only samples from the input space are available. The model is expected to provide a mapping from the input space back to the input space through some latent space usually with reduced dimensionality. As with the supervised learning, also in unsupervised learning the training algorithm presents these samples to the model, and uses some optimization method to estimate the parameters of the model in order to minimise the selected error metric. The error metric to be minimised is typically based on distance metrics between input vectors and model parameters. For instance, in vector quantization algorithms, the error is defined as the sum of all input vectors (Euclidean) distances from the quantized vectors.

Maximum Likelihood Estimation

Two different approaches to parameter estimation are Maximum likelihood (ML) estimation and Maximum a posteriori (MAP) estimation. A third, quite different approach known as Bayesian inference is discussed at the end of Section 2.3.3.

Consider a random variable χ which is modeled with a model family M, which consists of parametrized probability densities $p(X|\theta), \theta = (\theta_1, ..., \theta_k)$. ML estimation involves optimizing the parameter values $\hat{\theta}$ such that the probability of observing the data $X = (x_1, ..., x_N)$, given the model M is maximised (Bishop, 1995). When the observed samples are assumed to be independently drawn from the distribution $p(X|\theta)$, the joint probability density of the whole data set X is:

$$L(\theta) = \prod_{i=1}^{N} p(x_i|\theta, M) \ , \tag{2.5}$$

where $L(\theta)$ can be viewed as a function of θ for fixed X, in which case it is referred to as the likelihood of θ for the given X. In practice, maximising the likelihood $L(\theta)$ is often converted to minimising the negative logarithm of the likelihood:

$$E = -\log L(\theta) = -\sum_{i=1}^{N} \log p(x_i|\theta) \ . \tag{2.6}$$

The negative log-likelihood can be regarded as an error function. For most choices of the density function, the optimum $\hat{\theta}$ is found with iterative parameter estimation procedure, i.e. a training algorithm.

Properties of the ML estimate (Theodoridis and Koutroumbas, 1999):

- Asymptotically unbiased, which means that the estimate converges in the mean to the true value.
- Asymptotically consistent, so the estimate converges in mean square. Thus, for large sample size N, the variance of the ML estimate tends to zero.

- Asymptotically efficient, which means that the variance of the estimate reaches the Cramer-Rao lower bound, which is the lowest value any estimate can achieve.

- The probability distribution of the ML estimate approaches the Gaussian (with the true value as the mean) as sample size N goes to infinity.

If there exists a consistent and efficient estimator for the parameters θ, which can be represented by model family M, then maximum likelihood estimation will find this estimator. However, if the model family M is not selected appropriately, even the the best (maximised) likelihood does not fit the data well.

Problems of ML estimation include choosing the model family, including making assumptions about the "correct model" and the distribution of residuals, which must be made before the parameter estimation is possible. Thus, it is obvious that ML estimation does not address the problem of model selection.

Maximum a Posteriori Estimation

In the MAP estimation, the objective is to maximise the posterior probability $p(\theta|X)$ given the model M with parameters θ (Theodoridis and Koutroumbas, 1999). The posterior probability is given by the Bayes' formula:

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} \ , \tag{2.7}$$

where $p(\theta)$ is the prior probability for parameters θ, $p(X|\theta)$ is the likelihood, and the denominator $p(X) = \int p(X|\theta)p(\theta)d\theta$ is a normalising factor. MAP estimation chooses the estimate $\hat{\theta}$, which maximises the posterior $p(\theta|X)$, given the prior $p(\theta)$ and observed data X:

$$p(\hat{\theta}|X) = \max_{\theta} p(\theta|X) \ . \tag{2.8}$$

Since the posterior $p(\theta|X)$ is proportional to $p(X|\theta)p(\theta)$, the MAP estimate is usually found by minimising the negative logarithm:

$$E = -\log p(X|\theta) - \log p(\theta) \ . \tag{2.9}$$

The difference between ML and MAP estimates lies in the involvement of the prior $p(\theta)$ in the MAP. If it is assumed that prior distribution is uniform, that is, constant for all θ, then both estimates yield identical results.

2.3.3 Model Complexity

Selecting the correct complexity of the model is in general a separate problem from the parameter estimation. Several approaches to the problem exist.

With parametric models, a popular approach to model complexity selection is to build models with different number of parameters, and choose amongst them based on the estimated predictive ability. Here cross-validation or bootstrapping procedure can be applied to estimate the generalization error for each model, and the model with the smallest error is selected. In linear models, statistical theory provides several simple estimators of the generalization error under various sampling assumptions. These estimators adjust the training error for the number of weights being estimated, and in some cases for the noise variance if that is known (Venables and Ripley, 1999). These statistics can also be regarded as crude estimates of the generalization error in nonlinear models when there is a "large" training set. Correcting these statistics for nonlinearity requires substantially more computation (Moody, 1992), and the theory does not always hold for neural networks due to violations of the regularity conditions.

With nonparametric or semiparametric models, such as neural networks, selecting the complexity of the model based on the number of parameters is in general not feasible. In semiparametric models, the complexity of the mapping realised can change in a flexible manner based on the parameter values. In (Moody, 1992), the concept of effective number of parameters was proposed as a measure of the complexity of the model. Thus, the selection of model complexity can be turned into restricting the effective number of parameters. This can be achieved during the parameter estimation, by adding a penalty term which is dependent on the complexity of the model to the error function. Techniques applied in this manner are called *regularization* (Bishop, 1995). Regularization is effective in particular with semiparametric models. With adequate regularization, it is possible to apply flexible models with far more free parameters than is used in effect. Essentially, in regularization some prior knowledge about what kind of function the model should implement is exploited. With neural networks, weight decay is one popular technique.

Other approaches to model complexity selection include using criteria based on information theory, or using Bayesian inference for the model building. Several information theoretic metrics have been proposed, of which Akaike information criterion (AIC), Bayesian information criterion (BIC), and the MDL are perhaps the best known, and are reviewed shortly in this section. *Bayesian inference* is another approach to the model complexity selection, where the prior knowledge about the model complexity and parameters are given in a form of probability distributions for each entity, and separate hyperparameters describe the prior assumptions about the model complexity (Lampinen and Vehtari, 2001).

In the next sections, methods for model complexity selection are discussed, including regularization, cross-validation and bootstrapping, and using different information theory based criteria. Finally, Bayesian inference is discussed.

Regularization

Regularization is used to restrict the model complexity during the parameter estimation. Usually regularization is employed with flexible semiparametric or nonparametric models, which include large amount of free parameters.

Regularization is based on restricting the model complexity (e.g. the functional form of the mapping) by using some prior assumptions about what kind of mappings are probable to exist in the current data. This means that while the training data may suggest highly nonlinear dependencies (which may be due to noise or outliers in the data), the model tends to reach a functional form expressed in the prior assumptions, and optimizes the parameters given that assumption. In practice, results depend on the case and correctness of the prior assumptions.

Weight decay is an example of a regularization method. It can be applied e.g. with neural networks trained with backpropagation algorithm (Krogh and Hertz, 1992). Weight decay adds to networks prediction error E a penalty term, which depends on the weights w_i. The usual penalty is the sum of squared weights times a decay constant α:

$$E_r = E + \alpha \sum w_i^2 \tag{2.10}$$

Typically large weight values get larger penalty than smaller weights. In Bayesian setting, this can be also viewed as special case of zero mean Gaussian prior over the weights. In linear regression framework, this form of weight decay is equivalent to ridge regression (Hertz et al., 1991). Ridge regression originated as a method to solve numerical difficulties in performing the matrix inverse. Here the penalty terms are added to the diagonal elements of the matrix, thus forming a ridge.

In general form of MLP networks, large weights correspond to nonlinear mapping (or large change on the neuron's output), and small weight values correspond to linear mapping (small change on the neuron's output). Thus, weight decay restricts the functional form of the network, driving it toward more smooth mappings. Generalization ability of the network may depend heavily on the decay constant, especially with small training sets. One approach to choosing the decay constant is to train several networks with different amounts of decay and estimate the generalization error for each. The decay constant, which minimises the estimated generalization error is then chosen. Problem with weight decay is that different types of weights in the network will usually require different decay constants for good generalization.

Early stopping training is another method to restrict the model complexity (Bishop, 1995). As opposed to weight decay, no specific prior knowledge about the functional form is enforced. Early stopping was popular as a method for preventing overtraining of the MLP. In the typical setting, the data set is divided to design data set and the test data set. The design data set is further divided to training data and validation data. During the training of the model, the error for the validation data is frequently checked, and when the error starts to grow, training is stopped. Essential to the early stopping to be effective, is to employ a training algorithm where the model weights are adjusted only slightly during each iteration. Early stopping can be applied in committee models, where each model uses a different validation data set. Averaging the result from all models in the committee reduces the bias of the error, while keeping the variance the same. The early-stopping MLP committee is presented in Section 3.3.1.

Cross-validation and Bootstrapping

Cross-validation and bootstrapping are both methods for estimating generalization error based on "resampling". Both are useful for model selection purposes since they allow using all available data for parameter estimation of the model, and estimating its generalization ability is still possible with adequate precision.

In *V-fold cross-validation* (Picard and Cook, 1984; Amari et al., 1997) the data is first divided to V subsets of equal size, of which one is selected to be used for validation, and the rest of the data for training the model. After the model has been trained, validation error is calculated from the left-out subset of data. Similarly, each of the V subsets is selected at a time for validation, and the rest for training. After all V models have been trained, validation error estimate for the whole data is calculated by averaging over all V cases.

A special case of cross-validation is leave-one-out (LOO) validation, where the validation data set contains only one data sample. Leaveoneout is useful especially when training set is quite small. LOO is also computationally demanding, since with data size N, each model is trained N times. While leave-one-out provides more accurate estimates of the generalization ability, it also may prefer models with more complexity (parameters). This may happen when the training set is quite large and it contains similar samples. Now, when the used $N - 1$ samples are modeled more accurately, also in average the accuracy for the left-out sample becomes more accurate. As a consequence, noise in the samples tend to be modeled more accurately, and the number of parameters of the model becomes larger.

Bootstrapping (Efron and Tibshirani, 1993) in its simplest form includes analyzing subsamples of the data instead of subsets of data. Each subsample (bootstrap sample) is a random sample with replacement from the full sample. Depending on the case, number of subsamples B might be anywhere from 50 to 2000. In leave-one-out bootstrap, one data sample is first selected randomly for the test, and a bootstrap sample is drawn from the remaining data set. In a more general case, a predefined portion of samples can be selected for the nondesign data set containing both validation data and test data, and the bootstrap sample for training is drawn from the remaining data. For each of B bootstrap trials:

- Randomly draw $cN, c \in (0, 1)$ samples without replacement to form the nondesign data set
- Randomly draw N samples with replacement from the remaining $N - cN$ samples to form the training data set
- Decide the split of the nondesign set into validation and test data sets

Leave-one-out cross-validation and leave-one-out bootstrap methods were compared in (Efron and Tibshirani, 1997). Methods for correcting the bias in bootstrap estimates for the generalization error were also proposed. Furthermore, there are many sophisticated bootstrap methods that can be applied not only for estimating generalization error but also for estimating confidence bounds for network outputs.

Information Theory Based Criteria

Information theory offers a simple, intuitive point of view to model complexity determination: if we find a simple description for the observations, then we have found an interesting structure in the data.

Typically the metric calculated for each model involves the maximum likelihood error of the model, to which a penalty term is added. Penalty for a model is estimated as some approximation of information theoretic complexity of the model. Since the penalty is larger for more complex models, and the maximum likelihood error is larger for the most simple models, the criteria finds a compromise. It should be noted that none of the criteria can state which model family should be selected in each case. AIC, BIC, and MDL criteria are discussed in the following sections.

Akaike Information Criterion (AIC)

The AIC (Akaike, 1973; deLeeuw, 1992) has its roots in information theory, where Kullback-Leibler distance (or relative entropy) between the models (or probability distributions) f and g is defined for the continuous functions as the integral

$$I(f, g) = \int f(x) \log(\frac{f(x)}{g(x|\theta)})dx \ , \tag{2.11}$$

where log denotes the natural logarithm. K-L distance relates to the “information” lost when g is used to approximate f.

In model selection, a model that loses as little information as possible is searched. This is equivalent to minimising $I(f, g)$. The function f is considered to be given and g varies over a space of models indexed by θ. Another interpretation of minimising $I(f, g,)$ is finding an approximating model g that is the shortest distance away from the "true model" f.

K-L distance itself cannot be computed. As an approximation of it Akaike derived the relative expected K-L distance, which led to the AIC criterion. It was argued that the maximised log-likelihood is biased upwards as an estimator of the model selection target criterion, and that under certain conditions this bias is approximately equal to k, the number of estimated parameters in the model g. Under further conditions of large sample size N and "good" models, the AIC becomes:

$$AIC = -2\log(L) + 2k\ , \tag{2.12}$$

where L is the maximised likelihood for model g with k parameters $\theta = (\theta_1, ..., \theta_k)$, and N is the number of data samples $X = (x_1, ..., x_N)$.

Bayesian Information Criterion (BIC)

The Bayes Factor (integrated likelihood ratio) test is one method which quantifies the evidence for one hypothesized model against the other (Kass and Raftery, 1995). The BIC (Schwarz, 1978) is a large sample approximation to twice the logarithm of the integrated likelihood. BIC is defined as

$$BIC = -2\log(L) + \log(N)k\ . \tag{2.13}$$

where parameters L, N and k are the same as with AIC.

Derivation of BIC involves Laplace approximation to the integrated log likelihood. It ignores the terms of constant order, including those from the prior, which are dominated by terms from the likelihood when the data size N is large enough. Minimizing the BIC is equivalent to maximising integrated likelihood, which is equivalent to maximising posterior probability of a model when the priors are all equal. When the sample size $N = e^2$, AIC and BIC are the same. BIC favors simpler models more than AIC when the sample size N is larger than 8.

Minimum Description Length (MDL)

According to the MDL principle the best model or model class among a collection of tentatively suggested ones is the one that gives the smallest stochastic complexity (shortest description length) to the given data (Rissanen, 1986).

MDL has its roots in the algorithmic or descriptive complexity theory of Kolmogorov (see e.g. Cover and Thomas (1991)). Kolmogorov complexity can be seen as a characterization of probability based on the length of the shortest computer program that describes the object (or event) (Hansen and Yu, 2001). This quantity is the descriptive complexity of the object. While Kolmogorov complexity is not computable, approximations of the principle have been proposed for model selection purpose. In MDL, the attention is first restricted to those descriptions that correspond to probability models or distribution, and then the description length *interpretation* of these distributions are emphasized, rather than the actual finite-precision computations involved.

Features of MDL (Hansen and Yu, 2001):

- Unlike in traditional statistics, the data need not to be assumed to form a sample from a population with some distribution law. More generally, there's no need to assume anything about how the observed data was generated.
- The objective is not to estimate an assumed but unknown distribution, but to find good models for the data.
- The principle permits comparison of any two models or model classes, regardless of their type. Hence, it provides more general criterion than AIC, BIC and other criteria that depend only on the number of parameters.

There are different forms of MDL based on using a certain model. Consider a simple parametric model class M, consisting of a family of distributions indexed by parameter θ,

$$M = p(X|\theta), \theta = (\theta_1, ..., \theta_k) \tag{2.14}$$

where $X = (x_1, ..., x_N)$ denotes the data (string). Now MDL can be described by using a two-stage coding scheme, where the code length takes the form of penalized likelihood, the penalty being the cost to encode the estimated parameter values $\hat{\theta}$. Under certain assumptions about the encoding of the parameters, and "smooth" likelihood surface, the MDL then becomes:

$$MDL = -\log p(X|\hat{\theta}) + \frac{k}{2}\log(N) \ . \tag{2.15}$$

MDL can also be estimated by using only the observed data sample. The prior distribution for the parameter coding is estimated with normalisation over the parameter space, thus obtaining the distribution

$$L(X|\hat{\theta}) = \frac{-\log p(X|\hat{\theta})}{\sum_y -\log p(y|\hat{\theta})} \ , \tag{2.16}$$

which is called Normalized maximum likelihood (NML) distribution of the model (Rissanen, 1996).

Bayesian Inference

Bayesian inference offers a different methodology for building the model. As discussed before, in ML estimation the idea is to find a single set of parameters for the model that maximises the fit to the training data, perhaps modified by some sort of weight penalty to prevent overfitting.

In Bayesian data analysis, all uncertain quantities are modeled as probability distributions, and inference is performed by constructing the posterior conditional probabilities for the unobserved variables of interest, given the observed data sample and prior assumptions (Lampinen and Vehtari, 2001). Before we have seen any data, prior opinions about what the true relationship might be can be expressed in a probability distribution over the model parameters that define this relationship. The unknown degree of complexity is handled by defining vague priors for the hyperparameters, that determine the model complexity. After the data have been observed, revised opinions are captured by a posterior distribution over model parameters. Parameters that seemed plausible before, but which don't match the data very well, will now be seen as being much less likely, while the probability for values of the weights that do fit the data well will have increased.

The posterior probability for the parameters θ in a model M given the data X is, according to the Bayes' rule,

$$p(\theta|X, M) = \frac{p(X|\theta, M)p(\theta|M)}{p(X|M)} \; , \tag{2.17}$$

where $p(X|\theta, M)$ is the likelihood of the parameters θ, $p(\theta|M)$ is the prior probability of θ, and $p(X|M)$ is a normalising constant, called evidence of the model M. The term M denotes all the hypotheses and assumptions that are made in defining the model, including the selection of model family, specific noise model etc.

In this notation the normalisation term $p(X|M)$ can be seen as marginal probability of the data, conditional of M, integrated over everything the chosen assumptions M and prior $p(\theta|M)$ comprise

$$p(X|M) = \int_\theta p(X|\theta)p(\theta, M)d\theta \; . \tag{2.18}$$

$p(X|M_i)$ is the likelihood of the model i, which can be used in comparing the probabilities of different models, therefore it is also called the evidence of the model.

In Bayesian inference the posterior probability distribution of model parameters given the data must be estimated. Different sampling algorithms based on Markov Chain Monte Carlo (MCMC) (Gilks et al., 1996) are applied for the purpose. MCMC in general is a method to integrate over possibly high-dimensional probability distributions to make inference about parameters or to make predictions. Monte Carlo integration draws samples from the required distribution, and then forms sample averages to approximate expectations. MCMC draws these samples by running a cleverly constructed Markov chain for a long time. Gibbs sampling is a popular implementation of a Monte Carlo approach to generate samples from the posterior probability distribution.

After a large number of samplings, the samples tend in distribution to a random variable whose joint density is $p(\theta|X)$. Evaluation of the expectation of a function f of interest is then achieved by approximation

$$E[f(\theta)] = \frac{1}{n-b} \sum_{i=b+1}^{n} f(\theta^{i}) \ , \tag{2.19}$$

where n is the number of iterations and b is number of initial samples discarded during the *burn-in* period.

Bayesian model selection can be implemented by estimating the posterior probabilities of the models using MCMC methods. For instance, variable dimension MCMC methods have been proposed (Stephens, 2000). The variable dimension MCMC visits models according to their posterior probabilities, and thus models with small probability are not probably visited in finite time. The models with the highest number of visits are then investigated further.

It should be noted that also Bayesian inference involves methods where prior knowledge is required in order to achieve results in a limited time. Thus, the results gained are sensitive to the prior choices made during the model building.

2.4 Analysing the Model

As discussed earlier, model building is often an iterative process. Depending on the application and the objectives of the modeling, this iterative process may include several steps involving estimation of importance of input variables, choosing the correct model family and model complexity, and evaluating the model's performance in real application environment. Often the process of model building begins with using a simple model, which is then analysed in order to get insight into what kind of dependencies the data contains. More complex models can then be built by using the prior knowledge gained from the previous results.

Main objective of the analysis is to find optimal model for the task at hand. This objective can be divided to finding optimal input variables for the model, and an optimal model architecture. Assessing the importance of the input variables is in general useful, since most models benefit when unnecessary input variables are removed. Due to the curse of dimensionality, the training data is exploited more efficiently when there is less input variables in the model. However, it should be noted that some models can handle excessive input variables better than others, and some models may require that the input variables are encoded in a specific way. Also, using less input variables facilitates easier analysis of the resulting model. Furthermore, limiting the number of input variables is often desired since it reduces the cost of the implementation.

Comparing different models in order to find the optimal model architecture is usually carried out by estimating the predictive ability of each model. Different methods for estimating the predictive ability have been already discussed. Generalization error is the most popular metric for assessing the performance of the model. The generalization error is often estimated by using separate portion of the data for testing. Cross-validation and bootstrapping can also be applied. Furthermore, with Bayesian methods posterior probabilities for each model can be estimated.

Methods for assessing the importance of the input variables and estimating the predictive ability of the model are discussed in the following sections.

2.4.1 Importance of Input Variables

Determining the relevance of the inputs is of great importance in practical modeling problems, in both choosing the inputs in the model as well as analyzing the final model. The most common notions of importance are *predictive importance*, which can be assessed by the increase in the generalization error if the variable is omitted from the model, and *causal importance*, which can be seen in the change of the model outputs caused by the change in the input variable (Sarle, 1997a).

Change in the error function when an input is removed from the model can be used as a measure of the importance of an input. The change in the error function is a direct measure of predictive importance. However, this measure can be misleading when the inputs are correlated. It is important to estimate the parameters for the model after removing the input. Simply deleting an input from the model without retraining is equivalent to clamping the value of that input to zero for all cases. If zero is not a reasonable value for that input, the model's outputs are likely to be nonsense.

The causal importance is directly measurable only if the inputs are uncorrelated (which allows independent manipulation of the input values). If the inputs are not statistically independent, it is even more difficult to measure the importance of inputs, because the effects of different inputs cannot generally be separated.

For noisy data, all of the measures of importance of inputs are subject to sampling variation. It is difficult to estimate the amount of sampling variation (e.g., the standard errors of the importance measures). Possible ways to assess the variability of importance measures are cross-validation or bootstrapping.

Sensitivity Analysis

Sensitivity analysis is based on calculating the gradient of the output with respect to the inputs. This gradient can also be computed in a manner similar to the usual backpropagation algorithm.

The gradient is a vector of partial derivatives. Each partial derivative, by definition, gives the local rate of change of the output with respect to the corresponding input, holding the other inputs fixed. Consider a model $y_k = f(x|\theta)$ with k outputs and inputs x_i. The sensitivity of the output y_k with respect to input variable x_i can be approximated as:

$$\Delta y_k \approx \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i \tag{2.20}$$

A partial derivative applies only to a small neighbourhood of the input point at which it is computed, and thus they need to be evaluated at a large, representative sample of points from the input space. The importance of an input depends not only on the size of the partial derivatives, but also on the location of points in the input space with large partial derivatives. Consequently, it may be impossible to determine the importance based on partial derivatives (Bishop, 1995).

Saliency of Parameters

The estimation of the squared error surface with respect to model parameters can be employed in error analysis. This method can be applied in estimating the saliency of the parameters and can be used for all weights in the model. Change in the error function can be approximated by using the Hessian matrix. This method may give poor results if the number of training data is not much larger than the number of weights, if the optimal weights are infinite, or if the hidden units are not statistically well-identified (Bishop, 1995).

The error surface can be approximated by a second order Taylor expansion at the minimum of error function as follows:

$$E(\theta) = E(\hat{\theta}) + \frac{1}{2}(\theta - \hat{\theta})^T H(\theta - \hat{\theta}) \ , \tag{2.21}$$

where the Hessian matrix H for the model parameters θ_i, θ_j is also evaluated at $\hat{\theta}$:

$$H(\theta)_{i,j} = \frac{\partial^2 E}{\partial \theta_i \partial \theta_j}. \tag{2.22}$$

Since exact computation of the Hessian is in general not applicable with neural network models, different approximations can be applied. In diagonal approximation the posterior distribution of the parameters is assumed symmetric Gaussian distribution, i.e. parameters are not correlated. In practice, this is often not the case. In outer product approximation Hessian is approximated by the outer product of the first order derivatives. The approximation gives adequate results when the number of samples is large, and when the residual error is small. In practice, this applies when the target function is smooth, the error variance is small, or when the model is first fitted to the training data and the Hessian is approximated by using the same data. In testing phase a different set of data is used, in which case the latter does not hold.

Automatic Relevance Determination

Automatic relevance determination (ARD) prior was proposed by MacKay (1994) and Neal (1996) as an automatic method for determining the relevance of input variables in MLP. Irrelevant inputs should have smaller weights in the connections to the hidden units than the more important weights. Thus, in ARD the relevance measure of an input is related to the size of the weights connected to that input. In linear models these weights define the partial derivatives of the output with respect to the inputs, which is equal to the predictive importance of the input, and in the case of non-correlated inputs, also the causal importance.

In nonlinear models like MLP, however, the situation is more complex, since small weights in the first layer can be compensated by larger weights in the subsequent layers. Furthermore, the nonlinearity in the hidden layer unit changes the effect of one input depending on all other inputs. However, it was argued in (Vehtari and Lampinen, 2001) that ARD prior can be favourable in Bayesian inference with the MLP, since it loosens the assumption that all the input weight groups should have the same variance (nonlinearity).

2.4.2 Predictive Ability

The ability of the model to predict accurately (in classification or in regression), and also to generalize for the cases which were not presented during the training of the model, is the single most important capability of the model. In the following sections the concept of expected utilities is presented, and one Bayesian approach for model checking is discussed.

Estimating Expected Utilities

In prediction and decision problems, the predictive ability of the model can be assessed by estimating expected utilities (see e.g. Bernardo and Smith (1994)). This corresponds to estimating the generalization ability of the model, and taking into account the application specific costs. Utility is a measure of the relative values of consequences. By using application specific utilities, the expected benefit or cost of using the model for predictions or decisions can be computed. Also, in lack of application specific utilities, many general discrepancy and likelihood utilities can be used. In comparing different models the expected utilities have the benefit of revealing the true cost or benefit for using certain model over another. For instance, it may be possible that a statistically better model would only yield small benefit over a simpler model whose implementation in the real application is less expensive.

As the estimation of the generalization error requires a full model fitting (or perhaps k model fittings) for each model candidate, this approach is useful for analysing or choosing between a few models. The generalization error is typically estimated by using cross-validation or bootstrapping methods, which were discussed previously. Furthermore, on a more theoretical note, it has been argued that cross-validation cannot be used in general case for estimating the generalization error (Wolpert, 1996). This can be explained by the notion that in the case of very large number of models, the probability that one model exactly represents the training data set goes to one. This model is then selected as the best according to cross-validation criteria. However, since the model is overfitted to the training data set, the generalization ability of the model for a new data set is probably poor. Cross-validation estimates for the generalization error can still be applied in practical cases for choosing between a limited set of models. As has been argued before, the choice of the set of candidate models is of great importance. Prior knowledge that have been gained during the process of analysing the data should be used for this purpose.

In (Vehtari and Lampinen, 2002), practical methods for obtaining distributions of expected utility estimates were presented. Methods based on importance sampling leave-one-out and k-fold cross-validation were reviewed, and a quick method based on Bayesian bootstrap for obtaining samples from the distributions of the expected utility estimates was proposed. The models applied were MLP networks and Gaussian processes.

Posterior Predictive Simulation

Posterior predictive simulation can be used in model checking for Bayesian models. General discussion and also number of applications of the method are presented in (Gelman and Meng, 1996).

The idea in the posterior predictive model checking goes as follows: Let X be the observed data sample, θ the model parameters, $p(X|\theta)$ the likelihood, and $p(\theta|X)$ the posterior distribution. Assume that n samples from the posterior distribution of the model parameters $\theta_1, ..., \theta_n$ are drawn using e.g. MCMC method. Now for each $i = 1, ...n$, a hypothetical replication of the data X_i^{rep} is drawn given the simulated parameters θ_i. If the model is reasonably accurate, the hypothetical replications should look similar to the observed data X.

In essence, the idea is compare the observed data to simulations from a model.

The Bayesian way allows treating the model parameters as unknown quantities, since the parameters are themselves drawn from their posterior distributions. Hypothetical replications of data with the observed data can be compared by choosing a test variable $T(X)$, whose distribution is compared with observed X and posterior predictive distribution X^{rep}.

2.4.3 Visualizations

Data visualization is an important means of extracting useful information from large quantities of raw data. Visualizations are useful in all phases of data analysis.

In the data collection and preprocessing phase they can be used to get a first hand feeling of what is going on in the observed variables, what kind of distributions and ranges the variables have, and which need to be analysed more thoroughly. One option is to compute a reasonable number of the sample percentiles. This provides information about the shape of the data as well as its location and spread. Sample percentiles can be visualized e.g. by box plots and histograms. Visualizations can also be used after the preprocessing to visually confirm that the scaled input variables are indeed in permitted ranges.

Multidimensional Scaling

The human eye can distinguish very well patterns from two-dimensional plots, but higher dimensional data cannot be viewed as easily. Thus, different methods are applied for transforming the data to a lower, usually two-dimensional space.

Principal component analysis (PCA), as discussed in Section 2.2.2, can also be applied in visualization. The PCA analysis is carried out to the input vectors, and scores for each variable is calculated.

In a more general setting, any clustering algorithm, e.g. k-means (see e.g. Theodoridis and Koutroumbas (1999)) can be applied in clustering the input patterns (vectors). After the clustering, some multidimensional scaling method is applied to represent the distances between the clusters in a lower dimensional space, which can be visualized usually in two dimensions. Different methods for multidimensional scaling exist, of which the Sammon mapping is one of the most popular methods (see e.g. Webb (1999)). These methods can also be applied in visualizing the raw data. However, in order to distinguish patterns and shapes from the data, the number of data samples visualized has to be quite small.

Self-Organizing Map

The SOM (Kohonen, 1997) was discussed in Section 2.2.2 as a method for feature extraction. Perhaps even more typical application of the SOM is in the exploration and visualization of high dimensional data (see e.g. Cottrell et al. (1999); Vesanto

(2002)). The basic procedure involves training a two dimensional SOM with available data and using the result to plot various graphs, which are visually analysed and interpreted. Depending on the quality of the data and the projection realised by the SOM, clusters of similar data (with respect to the used distance measure) appear on the map space. An application expert can then find useful groupings of data from the map, which can be analysed further.

One typically used graph is the component plane visualization. Here components from the SOM weight vectors are shown in the shape of the map lattice. Comparing the component planes of two or more variables can reveal some dependencies between the variables. In another graph the distances between neighbour units, measured in the input space, are visualized in the map space using typically color coding. This graph can give insight into whether the data contains separable clusters. On a well organized map, the cluster boundaries should be clearly visible, since the neighbour units belonging to the same cluster should have small distances in the input space.

In the feature analysis, the SOM can be applied in evaluating visually the performance of the feature set used, for instance in inspecting the class separability. The SOM is trained with the data set using the features selected. After the map is trained, the point density function, $p(w)$, of the weight vectors approximates the point density function, $p(x)$, of the underlying sampled distribution in the input space (Cottrell, 1997). In a classification case, the number of input vectors belonging to each unit of the map is calculated (separately for each class) and visualized by color coding. Comparing visually the maps generated, by inspecting the differences and similarities between the maps, can give some clues of the separation or mixing of the classes. This method is applied in Section 5.4.2.

Chapter 3

Neural Networks in Temporal Sequence Processing

3.1 Introduction

This chapter discusses temporal sequence processing (TSP) with neural networks. TSP is a research area having applications in diverse fields varying from weather forecasting to time series prediction, speech recognition and remote sensing. The problems of the TSP can be divided into the following categories (Sun, 2001):

- sequence prediction, in which the objective is to predict elements of the sequence based on the preceding elements
- sequence generation, in which the goal is to generate elements of the sequence one by one in their natural order
- sequence recognition, in which the objective is to determine whether the sequence is a legitimate one according to some criteria
- sequential decision making, in which the goal is to select a sequence of actions in order to accomplish a goal, to follow a trajectory, or to maximise a reinforcement function

The sequence generation problem can essentially be formulated as a sequence prediction problem. Also sequence recognition problem can be turned into sequence generation/prediction problem, by basing recognition on prediction.

The time aspect of the problem under interest may vary corresponding to different properties of time, such as (Chappelier et al., 2001):

- time as a simple order relation (time as the index used to order events)
- time as metrics (time duration is meaningful)

- discrete time versus continuous time
- time over a finite versus infinite interval

The prototypical use of the classical neural network models (e.g. the MLP and the SOM) concerns *static* data processing. These models are not well suited to working with data varying over time. In response to the need to model time dependent phenomena, different temporal neural network models have appeared and constitute a continuously growing research field.

In this chapter, neural network models applicable to TSP are discussed, concentrating to their application in sequence (time series) prediction. Different approaches to temporal sequence prediction are first discussed, and neural network models based on supervised or unsupervised learning are then presented. The discussion concerns mostly discrete time processes, where observations range over a finite interval of time. In the sequence prediction problems, both the order and duration aspects of time are meaningful.

3.2 Temporal Sequence Prediction

A time series consist of measurements or observations of a natural, technical, or economic process, that are made sequentially in time. The main motivation in the time series research is the desire to understand the underlying phenomena and processes of the system under study. The major goals of time series analysis include system identification and reconstruction, and prediction and control of future events of the system.

In science, technology and economy there exist many interesting processes and phenomena, whose prediction is either very useful or profitable. These include different industrial processes that can be modeled, predicted and controlled based on some sensory data. Many phenomena of nature, such as daily rainfall or the probability of an earthquake would also be useful to predict. Medical applications include for example modeling biological signals such as EEG or ECG to understand better the patient's state. In economy stock market prices are an example of a possibly very profitable prediction task.

In general, consecutive samples of a time series are dependent on each other to an extent dictated by the process in question. Because of this dependency, it makes sense and it is possible to predict the future of the series. Whether the dependency is linear or nonlinear in nature, it can be estimated from the measured series using statistical methods. However, observations of the process usually contain noise that cannot be canceled out. Another problem is that the process may be dependent on some unknown variables, whose effect on the process cannot be estimated. Furthermore, there exists the question whether a process is predictable or not. In nature and in economy one can find complex processes that show self-organizing criticality (Bak, 1996). In this case the answer to the question is not obvious. However, this question is passed here by taking the view that a statistical model can be fitted to the available data, and the performance of the model in future predictions evaluates the usefulness of the method.

The problem is now to construct a model that can capture the essential features of the process using the measurements that are available. The model that can predict the future of a time series most accurately is usually considered to characterize the process best (Gershenfeld and Weigend, 1993). Usually, the model is built in such way that the input of the model consists of past samples from the series, and the output gives the desired prediction.

In one-step prediction the model output gives the prediction of the next value of the series. In *n*-step prediction, the model predicts *n* samples ahead in time. It is also possible to predict *n* samples to the future by making iteratively *n* one-step predictions to the future. However, this may not succeed with required accuracy, since the prediction errors quickly accumulate and lead to inaccurate predictions.

The problem of how to select the input variables fed to the model is not easily solved. In fact, the selection of the input variables is dependent on the nature of the model employed in the prediction. For traditionally applied static linear models, autocorrelation between the consecutive samples can be estimated, and used as a guideline when deciding the number of input variables. On the other hand, when the model is nonlinear and dynamic in nature, the selection of input variables cannot be solved in a general case. This problem is discussed in more depth in the following sections.

The reconstruction problem, i.e. how to represent the time series for the model, is typically handled by using delay coordinate embedding (Sauer, 1993). The input vectors (patterns) for the model are constructed by using windowing technique to split the time series into input vectors of constant length. The problem is thus converted into selection of the type and length of the window. Typically the input vectors contain past samples of the series up to certain length. The time is in effect converted into additional spatial dimension of the input vectors and the temporal context between consecutive samples is lost. After this conversion, conventional regression models, either linear or nonlinear, can be employed to model the dependence between the future values of the series, and the input vector containing past samples from the series.

For instance, a discrete time univariate time series consisting of N observations from variable x is denoted $\{x^N\} = \{x_1, x_2, ..., x_N\}$. One-step prediction y_t can now be estimated by the model f, which has k past samples of the series $\{x^N\}$ as the input:

$$y_t = f(x_{t-1}, x_{t-2}, ..., x_{t-k}) \ . \tag{3.1}$$

When the series is stationary, the delay coordinate embedding can be applied without losing too much from the original information. When the series is nonstationary, it can be difficult to determine a proper window length, i.e. the number of past samples used in the prediction. This is a considerable problem for global nonadaptive models. In general, the optimal window length for the model varies in the time series, and thus the windowing should be adaptive. This approach, however, can be applied effectively only when the changes in the statistics of the process are slow.

A time series can be nonstationary but still predictable if the changes in the statistics of the process are tractable (Papoulis, 1984). In this case, model parameters can be changed adaptively to reflect these changes. Adaptation of the parameters can be carried out i.e. with Kalman filter (Tsoi, 1998a). Another case is when the changes in the statistics of the process can be explained by different states in the process. In this case, it is possible to identify a local model for each state. The prediction of the time series is then produced by predicting first the state of the process, and then using the corresponding local model to gain the desired prediction (Sauer, 1993).

Another approach is to try to capture the essential context that is lost in the delay coordinate embedding with some other techniques. One way is to store the contextual information that exists between the samples and consecutive vectors to a memory in the model. The memory can store relevant information from the past, which is not visible in current input vector of the model. For instance, memory can store the current state of the process, which then determines the local model applied for prediction. One way of adding memory into the model is creating state variables into the model by using feedback (recurrent) connections. The memory structures used in neural networks are discussed at the end of this section.

Linear and nonlinear models

In order to construct a model for a process, data is gathered by measuring values of certain variables sequentially in time. Usually the data is incomplete and contains noise. In most cases, the underlying process is assumed stochastic. The goal for model building is to reveal the underlying process from the data. Model is estimated by using statistical methods to find regularities and dependencies that exist in the data.

Several computational techniques have been proposed to gain more insight into processes and phenomena which contain temporal information. Statistical methods based on linear and nonlinear models have been effective in many applications (Gershenfeld and Weigend, 1993). Among linear regression methods autoregressive (AR) and autoregressive moving average (ARMA) (Box et al., 1994) models have been the most popular methods in practice since Yule's paper in the 1920's (Yule, 1927). The theory of linear models is well known, and many algorithms for model building are available.

In practice almost all measured processes are nonlinear to some extent and hence linear modeling methods turn out to be in some cases inadequate. Nonlinear methods became widely applicable in 1980's with the growth of computer processing speed and data storage. Among the nonlinear methods, neural networks soon became very popular. Neural networks have gained a lot of interest in TSP due to their ability to learn effectively nonlinear dependencies from large volume of possibly noisy data. Many different types of neural networks, such as the MLP and the RBF have been proved to be universal function approximators (Cybenko, 1989; White, 1990). This gives neural networks certain attractiveness in time series modeling (see e.g. Mozer (1993); Tsoi and Back (1994); Lehtokangas (1995)).

Local and global models

Different models can be divided into global and local models. In global model approach only one model characterizes the measured process. Traditional models, such as the AR model and the MLP, when applied as in (Lapedes and Farber, 1987), are examples of global models. Global models give best results with stationary time series. However, when the series is nonstationary, identifying a proper global model becomes more difficult. During the last decade, local models have gained a growing interest, because they can often overcome some of the problems of the global models (Singer et al., 1992). Local models are based on dividing the data set to smaller sets of data, each being modeled with a simple local model.

An example of an extension of the AR model to build a local linear model is the threshold autoregressive model (TAR) (Tong, 1990). The TAR consist of separate AR models that are used in different parts of the data according to threshold values. The selection of the current AR model giving the prediction is carried out by comparing the value of the time series at certain point in the past with the threshold values. The model selection problem of the TAR model contains deciding the number of local models, the number of parameters in each model, and the time lag with which the model selection is carried out. The estimation of the model includes optimizing the threshold values and the parameters of local AR models.

In the local model approach, division of the data can be carried out with some clustering or quantization algorithm such as the k-means, the SOM (Walter et al., 1990; Vesanto, 1997), or neural gas (Martinetz et al., 1993). An approach where each unit of the SOM represents an AR model with its reference vector was presented in (Lampinen and Oja, 1989). After clustering the data, local models for the generated local data sets are estimated. The problem of data division for achieving the best prediction accuracy remains in a general case to be solved. One method using an iterative learning algorithm to solve the division of the data and the local model parameters at the same time is Hierarchical Mixtures of Experts (Jordan and Jacobs, 1994). Here a gating network decides which local models are used to in the prediction. The parameters of local models and the gating network are optimized with iterative Expectation-Maximization algorithm.

Neural Network Architectures

Different neural network architectures have been proposed to capture the inherently complex time structure in the modeled time series. Methods for capturing the temporal context can be roughly divided to two categories, however, also both of these methods can be employed in the same model.

The first method involves using tapped delay line registers, which store previous values of the input and the model state. In this method, the input of the model consists usually of segmented input vectors from the time series. Thus, the delay lines store previous input vectors, and previous values of the output values of the neurons. Using delay lines in the input layer of the network corresponds to using delay coordinate embedding to the time series. Delay lines within the network are not so easily interpreted, because the mappings formed inside the networks are nonlinear. The weights (parameters) which connect the delayed values to the network have to be estimated differently than in a static network, since the output of the model depends also on the previous values of the series.

The second method involves using positive feedback, that is making the network recurrent. The feedback allows the input-output function of the model to change according to time context. Thus, the function is no longer a static, but a dynamic function. Also in this case, the weights of the network which form the recurrent connections, have to be estimated by taking into account the previous inputs and the previous states of the network.

In both methods, memory is created within the network. To characterize memories in different architectures, two dimensions, depth and resolution has been proposed (Mozer, 1993). Roughly, depth refers to how far into the past the memory stores information relative to the memory size, and resolution how accurately information concerning the individual elements of the input sequence is preserved. Memories of equal size have different depth and resolution based on the way they store the information. The size of the memory can be calculated as the number of state variables in the model. The tapped delay line structure has high resolution, but low depth. Each data sample is stored in the delay line without any change, so there is no information loss. Since in practice the delay line has finite length, it can only store information from limited time period. The recurrent (feedback) connection has low resolution, but high depth. In the most simple case, the output of the state variable equals the weighted sum of current input and past output of the variable: $y_t = x_t + \alpha y_{t-1}$. Here the feedback coefficient α controls the depth of the memory. The state variable stores information farther from the past when the value of α is increased. However, this also makes the resolution of the memory lower.

In dynamic linear models the state of the system determines the output of the model for each input (Pole et al., 1994). The same holds also for nonlinear dynamic models. The state of the system is stored in the memory structure of the model. The memory structure then determines which kind of dynamic systems can be modeled. It can be argued that an optimal memory structure should minimise the size of the memory while retaining the essential information from the data. This corresponds to finding the best encoding for the dynamics in the observed phenomena.

In (Elman, 1990) it was proposed that there is no globally optimal way of representing the effect of time. It is then obvious that there cannot be a globally optimal memory structure, and since different memory structures have their limitations, one should search for optimal memory structure in each case. Consequently, there is vast number of different neural network architectures which try to capture the time context in the data. A general division of these architectures can be done based on the memory structure of the model as follows:

- Delay lines or memory registers between connections
- Different feedback connections
- Dynamic neuron models
- Combinations of the above

In this chapter the discussion concerns models which belong to the first or the second class, or are combinations of them. These models have also gained the most interest in time series prediction and modeling. Dynamic models of neuron are not discussed as separate models. Partly this is due to restricting the scope to the most important models, and partly due to the fact that in general the dynamic models of neuron can be represented by using the conventional perceptron, and adding recurrent and time delay memories to the network. Also, the discussion is restricted to models which process discrete time data.

In time series prediction with neural networks, the main problems have been deciding the length of the input vectors and the structure of the network. These problems remain mostly the same for all architectures discussed. In addition, with the recurrent networks, the stability of the model and the learning algorithm must be considered. Neural network models based on supervised learning are discussed in Section 3.3. and unsupervised learning algorithms are presented in Section 3.4. In both of these categories, delay lines and recurrent connections can be employed to enable processing of temporal context.

3.3 Supervised Learning

Several different supervised neural network architectures have been proposed, based on the artificial perceptron model. The most popular are the MLP and the RBF network. The cascade correlation (CC) network has also gained some interest. These neural network architectures consist of layered units, connected feedforward from input layer to output layer. Neural network architectures applicable to TSP, which include time delays or feedback connections, can be viewed as modifications to these static feedforward networks. A review and a taxonomy of supervised neural networks models capable of processing spatio-temporal problems was presented in (Kremer, 2001).

In the following sections, the popular MLP is first briefly reviewed, and then network architectures based on time delay structures and recurrent connections are presented.

3.3.1 Multilayer Perceptron Network

Feedforward multilayer networks are architectures, where the neurons are assembled into layers, and the connection between the layers go only into one direction, from the input layer to the output layer. There are no connections between the neurons in the same layer. Also, there may be one or several hidden layers between the input and the output layer.

These architectures are also static, so the mapping between the input and the output is a static function. In practice, this also means that the network does not have memory, where it could store contextual information from the past. Therefore, the input of the network must contain all the necessary contextual information which is used in representing the output.

The MLP is a nonlinear model consisting of number of neurons (units) organized to multiple layers, forming a mapping $y = f(x, w)$ between the input x and the output y, adjusted by the weights w. This mapping, with a certain architecture and weights, forms a static, nonlinear function. The complexity of the MLP network can be changed from an almost linear model to a highly nonlinear model by varying the number of layers, the number of units in each layer, and the values of the weights. A typical single hidden layer MLP network architecture with K outputs gives rise to the model $f_k(x, w), k = 1, ..., K$ with weights w. The model has the functional form

$$f_k(x, w) = w_{k0} + \sum_{j=1}^{q} w_{kj} g \left(w_{j0} + \sum_{i=1}^{p} w_{ji} x_i \right) , \qquad (3.2)$$

where p number of inputs, q is the number of hidden layer units, g is the activation function for the hidden layer units and indices j and i correspond to output and hidden units, respectively.

The MLP is often considered as a generic semiparametric model meaning that the effective number of parameters may be less than the number of available parameters. The effective number of parameters determines the complexity of the model. For small weights the network mapping is almost linear and has low effective complexity. This is because the central region of sigmoidal activation function can be approximated by linear transformation.

A practical problem with neural networks is the selection of the correct complexity of the model, i.e., the correct number of hidden units or correct regularization parameters. It is well known that plain ML optimization of the MLP may lead to severe overfitting. Recall that ML searches for the optimal parameter values which maximize the likelihood, and the respective learning algorithms search to minimise the error metric for the given data. Usually, the model that gives the smallest error for the training data does not generalize well in the new data. This is because the model starts to represent the noise in the training data. Since the MLP is quite flexible model, and efficient learning algorithms are applied in searching the optimal parameter values, overfitting is likely to happen. Consequently, regularization methods are needed in order to provide a good generalization ability. Traditionally, complexity of the MLP has been controlled with early stopping or weight decay methods (Bishop, 1995).

The early stopping method was discussed in Section 2.3.3. Briefly, the design data set is divided to training data and to validation data, and iterative learning algorithm gradually optimizes the network weights, until the error metric estimated from the validation data set starts to grow. Since training is stopped before a minimum of the training error is reached, the complexity of the model is regularized. It should be pointed out that early stopping in its basic form is rather inefficient, due to the fact that it is very sensitive to the initial conditions of the network, and only part of the data is available for training the model. These limitations can easily be alleviated by using a committee of early stopping networks, with different partitioning of the data to training and validation sets for each network. The output of the committee model is calculated as the mean of the all committee member's outputs. When applied with caution, the MLP early stopping committee turns out to be a good baseline method for neural networks (Lampinen and Vehtari, 2001).

3.3.2 Time Delay Networks

The simplest way to insert memory into the model is to add registers which store consecutive values of the model input (Mozer, 1993). These registers can be implemented as tapped delay lines, which are fed with new value at each time increment, and can store up to k past values of the input. This corresponds to making the delay coordinate reconstruction from the time series. Traditional way of using neural networks in TSP is to convert the temporal sequence into concatenated vector via a tapped delay line, and to feed the resulting vector as an input to a network. Using the tapped delay line as an input layer in a multilayer perceptron network was one of the first successful implementations of neural networks in time series prediction (Lapedes and Farber, 1987). In more advanced models (see e.g. Mozer (1993); Chappelier et al. (2001)), tapped delay lines are also employed in the connections between the layers of the network.

In the tapped delay line memory the time dimension is converted into spatial representation. In this reconstruction, neural network approximates the mapping between the input vectors and the desired output. Often the output is one-step prediction of the time series. In this function approximation the contextual information between the consecutive input vectors is lost. Also, the time difference between consecutive samples is implicitly assumed to be equal in all input vectors.

In many cases, this reconstruction can capture the essential information from the sequence, allowing to build models to predict it. Especially, if the series is stationary, the reconstruction is often successful. When the series is nonstationary, it can be difficult to determine a proper window length. Selecting the order of the delay coordinate embedding, or the length of the input vector, is usually carried out by using the model selection methods. However, the optimal window length for the model may vary in the time series, and thus the windowing should be adaptive. This approach, however, can be applied effectively only when the changes in the statistics of the process are slow.

The memory structure based on delay lines can be regarded as having high resolution, but low depth (Mozer, 1993). Since the values are stored without any encoding to delay registers, there is no information loss. However, the size of the memory increases linearly with the length of the memory. Use of delay lines between neurons in a multilayer architecture has been justified by the fact that in biological neurons there are also communication delays, which can be several tenths of milliseconds. It has been argued that these delays are essential in the way that brain processes information (Haykin, 1994). Another justification of the delay lines comes from the theory of dynamic linear systems. Delay coordinate reconstruction has been proved to capture the dynamics of the state vector with certain restrictions (Sauer, 1993). However, nonlinear dynamic systems may not be as easily modeled, since the state vector can have large changes due to the nonlinearity, and consequently the requirements of the delay coordinate reconstruction for to capture the dynamics of the system may not hold.

There are several drawbacks in using delay line memories. One drawback in using separate memory registers within the model is the inability to detect when a change occurs in the memory. Thus, it may be difficult to decide when the output of the model should be evaluated. However, this can be easily circumvented in models which process information only in discrete time steps. Output of the model can be evaluated at each time increment, when also the state of the memory registers is updated. Another drawback is the strictly limited length of the delay registers. This allows the model only process information, which is explicitly stored in the delay lines. This means that the model cannot process sequences which exceed the length of the delay lines. This can lead to overly long delay registers which complicate the model building process. Furthermore, detecting similar sequences of different length may be difficult or even impossible.

Despite the drawbacks, delay lines have been popular in neural networks which process temporal information. The structure of the memory is simple, and the memory's effect to the output of the model has only limited length. Thus the estimation of the model parameters can be carried out with slightly modified learning algorithm of a static network, which takes this effect into account. Most importantly, MLP with added tapped delay memories can be trained with modified backpropagation algorithm (Werbos, 1990). This architecture, known as Time delay neural network (TDNN), is discussed next.

Time Delay Neural Network

The MLP was applied in time series prediction first by (Lapedes and Farber, 1987). It was based on using a tapped delay line in the input layer. Otherwise, the network structure is static. Other popular neural networks which employ the delay line memory include Time delay neural network (TDNN) (Waibel et al., 1989; Lang et al., 1990), Finite impulse response neural network (FIR NN) (Wan, 1993; Back and Tsoi, 1991), and Gamma memory neural network (de Vries and Principe, 1992). In the Gamma memory neural network, the memory is built from FIR filters, which include local feedback connections, thus making it a hybrid architecture, utilizing both delay lines and recurrent connections.

The TDNN has been the most popular network using delay lines. In the TDNN, the network has an input layer, an output layer, and one or several hidden layers. The number of neurons used may vary. The input of the network is fed to the delay line. Values stored in the delay line are then fully connected to the hidden layer neurons, thus the input layer implements the delay coordinate embedding of the time series. The difference between TDNN and MLP comes from the hidden layer. In the TDNN, also the previous outputs of the hidden layer units are stored in delay lines. These values are then fully connected to the output layer (in univariate case output neuron), which combines all values together.

The TDNN has been successfully applied in speech recognition, specifically in isolated phoneme recognition (Waibel et al., 1989; Lang et al., 1990). In this application, spectral features are first extracted from the raw speech signal. After the feature extraction, each input vector represents the power spectrum of the speech signal at predefined frequency ranges. The time resolution of the spectrum is around 10-30 ms, but actual spectral values might be calculated from a longer time period in order to gain more accurate estimate of the power spectrum. The delay line of the input layer stores all samples from the spectrogram of one phoneme. Thus, the time is converted into spatial representation. The hidden layer acts as a compressor, which finds the essential patterns from the signal. The output layer neurons then implement the classification of the sample to one of the possible phonemes.

The FIR neural network has the same functional form as the TDNN, but the training algorithm, named Temporal backpropagation (Wan, 1993), is different. Also, the vectoral presentation of the FIR network allows presenting the architecture in similar form as the MLP. Figure 3.1 shows a schematic picture of an FIR neural network with two inputs, one output and three hidden layers.

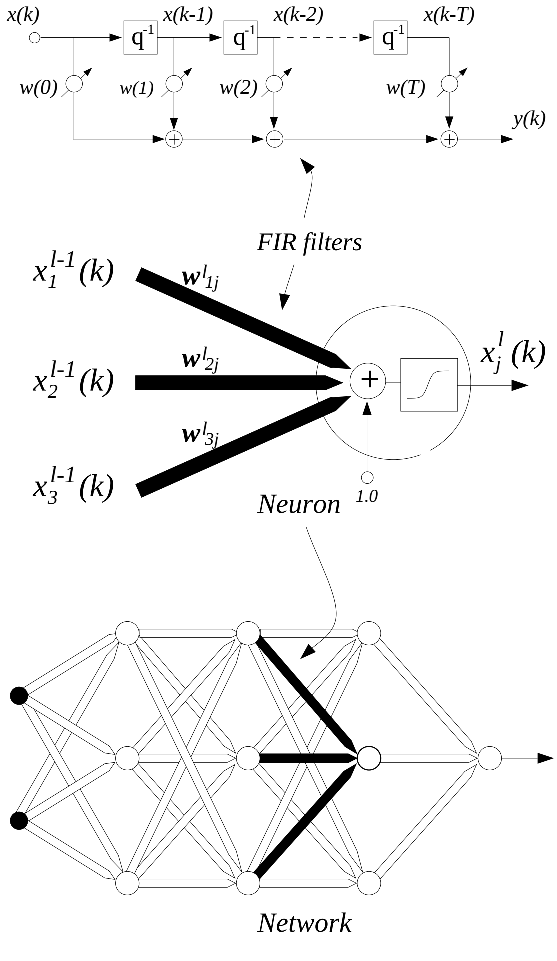

Figure 3.1: Schematic picture of the FIR neural network (Wan, 1993). At the bottom, the general structure of the network is shown. The network has two inputs, three hidden layers and one output. In the middle, a neuron j in hidden layer l with three inputs is shown. Each input connection is an FIR filter, that stores T past values of the output from the neuron in layer $l-1$.

The FIR network can be built from a conventional MLP by replacing each connection weight in the neurons with an FIR filter. Each filter consists of a tapped delay line and a weight vector. Output of the filter is calculated as a product from the values stored in the delay line and the weight vector. In effect, this has the same function as using a delay line, from which each value is fully connected to all neurons. However, by using the FIR structure, the learning algorithm can be presented as a vectoral generalization of the conventional backpropagation algorithm (Wan, 1993). Learning algorithms for neural networks which include delay lines are discussed next.

Learning Algorithms

TDNN can be trained with *Backpropagation through time* (BPTT) (Werbos, 1990) algorithm, which takes into account the past values of the time series and hidden layer units, which affect the current output of the network. BPTT is based on constructing a static network, which can then be trained with conventional backpropagation error gradient algorithm. Since the TDNN only includes limited time dependencies, BPTT can train the network without approximations.

In BPTT, the network is first simulated with the training data, and values stored in delay lines as well as networks output are saved at each time increment. Then the network is unfolded in time, originating an equivalent feedforward network called encoding network. This encoding network is a static version of the TDNN corresponding to each time step. The static version of the network contains replicated weights corresponding each time step. Correction terms are then calculated with standard backpropagation rule for each weight. Finally the correction terms calculated for the static network are accumulated to get correction terms for the weights in the TDNN network.

A serious drawback of the BPTT is that the number of calculations required is proportional to the square of the time steps in the simulation. In batch training, also the memory consumption can be quite large, since all values stored in the delay lines as well as the output of the network must be stored, before training the network one epoch is possible. Since the computational and memory requirements of the BPTT are quite extensive, different approximations of the training algorithm can be applied.

The *Temporal backpropagation* algorithm is an alternative algorithm for training the TDNN. Originally it was proposed by (Wan, 1993) for training the FIR neural network. Temporal backpropagation can be seen as a vectoral extension of the backpropagation algorithm. It is based on the idea of backward filtering the error gradient terms through the FIR connections to form the error gradient terms for the previous layer. The process is applied layer by layer, starting from the output and working through the network, in a similar fashion as in backpropagation algorithm. The algorithm uses certain approximations in calculating the error gradient. Convergence of the algorithm has been proved as long as the learning coefficient is kept small.

The greatest benefit of the Temporal backpropagation over the BPTT comes from the linear scaling of the computational complexity. The backward propagation of error terms can be calculated by parallel distributed processing. Also, the number of calculations per iteration grows only *linearly* with the number of layers and connections in the network. Disadvantage of the algorithm is that since it is based on utilizing the idea of reverse filtering the error terms, it cannot be implemented effectively as a batch algorithm. Therefore, training the network may take more time, and consequently the number of required calculations grows.

3.3.3 Recurrent Networks

As discussed previously, in the tapped delay line memory the time dimension is converted into spatial representation. In this reconstruction, neural network approximates the mapping between the input vectors and the desired output. Most time series are not stationary, and consequently the delay coordinate embedding may remove contextual information. The time-delay neural network approach has also other drawbacks, one of the most serious ones of being the difficulty to determine the proper length for the delay line. If the delay line is too long, it will increase the parameters in the model. This will complicate the estimation of the model, since more data is needed to obtain reliable estimates. Also the training time increases. If the input vector is too short, it may not capture the needed information from the series, which would allow predicting the series.

Using recurrent connections is another approach for temporal sequence processing. A number of dynamic neural networks models have been designed for TSP to capture inherently the essential context of the temporal sequence without the need of external time delay mechanics (see e.g. Mozer (1993); Chappelier et al. (2001)). In these models, the learning equations are often described by differential or difference equations and the interconnections between the network units may include a set of feedback connections, i.e., the networks are recurrent in nature. The recurrent connections allow the mapping of the model from input to output to change dynamically according to changes in the modeled phenomena. In essence, the recurrent connections allow storing information from the past input and the past state of the model.

The simplest form of the recurrent connection is making the feedback from the neuron's output back to its input. This kind of neurons have been called (depending on the context and the form of the feedback) time summating neuron, leaky integrator neuron, state unit, or context unit. In this case, the previous output of the neuron at time $k-1$ is fed back to its input at time step k. If the neuron has linear output function, it acts as an infinite impulse response (IIR) filter. In practice, the neuron may also include logistic function, which changes the feedback to act as a nonlinear filter. The context unit stores exponentially decaying values of the previous outputs of the neuron. The output can be seen as a state variable, whose impulse response can be represented as

$$c(k) = w\alpha^k \ , \tag{3.3}$$

where α is the feedback coefficient, whose value is usually $\alpha \in \{0, 1\}$, and w is the input weight. In a more general case, there are more inputs. The output of the neuron can also be calculated using the current input and previous output as

$$y(k) = wx(k) + \alpha y(k-1) \ . \tag{3.4}$$

Context unit stores recent inputs more accurately than those farther in the past. There is no strict limit for the memory's length. Depending on the value of α, the context unit can store information from longer or shorter time period. The depth of the memory is in this case large, since information is stored (or integrated) in one state variable from several time steps. The resolution of the memory is small, since previous values of the output are lost in the summing. Thus, the depth of the memory is gained at the expense of the resolution.

Using recurrent connections has certain benefits over the time delay memories, since it is in general a more flexible way of representing the temporal context (Tsoi, 1998b). One of the most important benefits is the ability to store information from long time periods. Thus, it is possible to build models without estimating the length of the input vector beforehand. For instance, the state of the process can be stored for long time periods within few context units. If the process has several states, the ability to store the state efficiently with few estimated parameters simplifies the model building. Context units can also capture the temporal information that exists between the consecutive vectors of the data, which is lost in delay coordinate embedding. Thus, dividing the time series into input vectors may be avoided altogether, if the input only contains the current value of the series. That is, the series is fed serially to the model, instead of using the delay coordinate embedding.

Drawbacks of the recurrent networks include more complicated parameter estimation. In theory, with fully recurrent connections the output of the network is affected by all previous inputs. Consequently, when estimating the parameters of the model, all these inputs (and previous values of context units) should be taken into account in the training algorithm. In practice, all previous inputs cannot be taken into account, since memory and time requirements of such an algorithm would be impractically large. Often only limited time period is taken into account during the training of the model. Since the effect of the feedback decays exponentially, this approximation may lead to quite acceptable results. However, this approximation also forces the model to represent and detect only patterns which "fit" into the selected time period. Hence, while the model's memory structure may be potentially able to represent longer and subtler contextual dependencies, the training algorithm ultimately controls what kind of mappings can be represented.

In recurrent neural networks the training algorithm also has to take stability into account. Since the feedback connection may drive the context unit into unstable stage or into saturation, the feedback weights must be regulated during the training. Also, in general the convergence into local minimum may not be certain, and it may be difficult to choose the correct time to stop the training. In particular, training the fully recurrent neural network may be complicated. The estimated prediction error can oscillate during the training, and convergence into minimum is not certain. In general, the outcome of the training depends upon the initial parameters of the network and the training algorithm (Tsoi, 1998a).

In spite of the complications in the training, recurrent neural networks have been popular subject for research. Most of the suggested architectures and algorithms are based on some kind of restricted feedback, which makes the parameter estimation easier. In practice, many of the proposed recurrent networks applied in time series prediction have employed both time delay memory and recurrent connections. Typically, the input to the model includes the tapped delay line, and the context units are located in the hidden layer of the multilayer network. These architecture are discussed next.

Recurrent Architectures

Recurrent connections can be added to layered architectures in several ways, yielding a plethora of different recurrent architectures. An overview of different recurrent neural network architectures was presented, and also a canonical form of networks and transforming from one architecture to another were discussed in (Tsoi, 1998b).

The most general network (and presumably also the most complicated) is the fully recurrent neural network, proposed by many authors, see below. In this architecture, there are no separate layers, no separate input layer or output layer. All neurons are connected to all other neurons with one time unit delay. The input is fed to some selected set of neurons, and similarly the output is read from some selected set of neurons.

It is possible to build different network architectures by limiting the number and location of the recurrent connections. Most of these networks are built by adding recurrent connections between or within the layers in a multilayer network. These partly recurrent networks include e.g. Elman network (Elman, 1990), which employs context units in the hidden layer. The outputs from the context units are fed back to the input layer. In Jordan network (Jordan, 1986), context units form a part of the input layer, and store previous values of the output layer of the network. In (Williams and Zipser, 1989), a general network where the neurons can act as an input, an output, a hidden neuron, or a combination of input and output was proposed. In addition, connections between the neurons may have a time delay with adjustable weight.

Four different categories of architectures were proposed for identification purposes in (Narendra and Parthasarathy, 1990). In the most general category, the output of the model is formed by a nonlinear combination of both past input and output values. This model is closely related to NARMAX neural networks, which are nonlinear replacements of the ARX and ARMAX models (AR and ARMA models with exogenous inputs) (Chen and Tsay, 1993). In Locally feedback neural network (Frasconi et al., 1992) the feedback connections are restricted between neurons. Only recurrent connections from the output of the neuron back to its input are allowed. Also different transfer functions may be utilized to build a longer memory. In (Back and Tsoi, 1991), modifying the MLP by replacing the weights with FIR or IIR filters was proposed. As discussed earlier, the case with the FIR synapses corresponds to the TDNN. Furthermore, architectures have been proposed to circumvent the limitations of the learning algorithms based on gradient descent (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997).

The vast number of different proposed architectures can be explained by the shear number of possibilities. In general, restricted feedback allows for simpler training algorithm, while in many practical situations it can still prove to be adequate for modeling the phenomena of interest. Since the training of the recurrent networks has been the most serious drawback of the models, it is natural that different models have been proposed. In the next section, learning algorithms for recurrent neural networks are discussed.

Learning Algorithms

Backpropagation through time (BPTT) (Werbos, 1990), which was briefly discussed as a learning algorithm for TDNN, is a general algorithm which can also train recurrent neural networks. The idea of unfolding the network in time can also be utilized with recurrent connections. Since the feedback connections in theory have an infinite memory, the BPTT must be truncated to take into account only predefined number of previous time steps.

For instance, the output of the output layer depends on the current and previous outputs of the hidden layer (and if the output layer has feedback connections, also on its previous values). In order to adjust the weights, the hidden layer is replicated, such that each replicated layer has the output from one step further into the past. The number of replications is restricted to some meaningful value. During one training step, the weights are adjusted by standard gradient descent rule, and the cumulative effect of the adjustments from previous time steps are finally summed together to get the final adjustment the weights.

Real time recurrent learning (RTRL) (Williams and Zipser, 1989) is another algorithm proposed for training recurrent networks. As opposed to BPTT, the training is carried out in “real time”, and the correction terms are calculated after each time increment with each new input value to the network. Also RTRL is quite heavy computationally. However, it has been shown that in the case of modifying the RTRL algorithm to batch update, the BPTT and the RTRL are dual of one another (Tsoi, 1998a).

These general algorithms (BPTT and RTRL) are seldom applied in practical problems as such. Often different approximations of the training rule are employed, allowing faster computations or using less memory. The most simple approximation is to take into account only limited time period from the past when training the network. Drawbacks of using BPTT or RTRL with recurrent networks include the quite high computational and memory requirements. More importantly, it has been shown that learning long term dependencies is difficult with algorithms based on gradient descent (Bengio et al., 1994). Since the weight adjustments are calculated as the gradients from the previous values, the adjustments effectively diminish to negligible small when going further into the past. Recall that with the TDNN the length of the delay lines restricts the length of the sequences that the network can process, and thus the TDNN can in general be trained with the BPTT. However, with the recurrent networks trained with the BPTT (or more generally, by gradient descent algorithm) the length of the sequences that the network can represent is limited by the algorithm, not by the network architecture.

3.4 Unsupervised Learning

Most recurrent neural networks are trained via supervised learning rules. Only quite few unsupervised neural networks models have been proposed for temporal sequence processing, although, it can be argued that in TSP unsupervised neural networks could reveal useful information from the sequences at hand in analogy to unsupervised neural networks’ reported power in cluster analysis, dimensionality reduction and visualization of their ’static’ input spaces. Moreover, in many TSP applications unsupervised learning could utilize more effectively the available temporal data than supervised learning methods, because no preclassification or prelabeling of the input data is needed. Based on the above the needs for the unsupervised learning methods in TSP are immense. A recent review and a taxonomy of unsupervised neural networks models capable of processing spatio-temporal problems can be found in (de A. Barreto et al., 2003). Most of the proposed models are based on the Self-Organizing Map.

The SOM in its basic form has no ability to process temporal context between the input vectors. Several different modifications have been proposed to allow sequence processing with the SOM. In the following sections, a short review of the SOM in sequence processing is first presented. Two modifications of the SOM, Temporal Kohonen Map (TKM) and Recurrent Self-Organizing Map (RSOM) are then presented in more detail.

3.4.1 Self-Organizing Map

In this section, the learning algorithm for the conventional SOM is first reviewed shortly. The search criteria for the best matching unit (bmu), and the update rule for the weights of the bmu and its neighbours are presented. In the proposed architectures based on the SOM, these learning rules must be modified in order to allow processing of sequences of input vectors.

Learning in SOM

The stochastic learning algorithm of the SOM is described next, since it can be compared with the respective learning rules applied with the TKM and the RSOM. However, in practice the batch learning algorithm is more frequently applied with the SOM, since it allows more efficient and accurate estimation of the weights of the map.

In the stochastic training algorithm of the SOM, one sample, the input vector $x(n)$ from the input space $V_I \in R^m$, is selected randomly and compared against the weight vector w_i of the unit i in the (typically two-dimensional) map space $V_M \in R^2$. The best matching unit b to given input pattern $x(n)$ is selected using some metric based criterion, such as

$$\|x(n) - w_b(n)\| = \min_{i \in V_M}\{\|x(n) - w_i(n)\|\} , \qquad (3.5)$$

where $\| \|$ denotes the Euclidean vector norm. Initially all weight vectors are set randomly to their initial positions in the input space. During the learning phase the weights in the map are updated toward the given input pattern $x(n)$ according to

$$w_i(n+1) = w_i(n) + \gamma(n)h_{ib}(n)(x(n) - w_i(n)) , \qquad (3.6)$$

where $i \in V_M$ and $\gamma(n)$, $0 \leq \gamma(n) \leq 1$, is a scalar valued adaptation gain. The neighbourhood function, $h_{ib}(n)$, gives the excitation of unit i when the best matching unit is b. A typical choice for h_{ib} is a Gaussian function $h_{ib}(n) = exp(-\|r_i - r_b\|^2/\sigma(n)^2)$, where σ controls the width of the function and r_i, r_b are the SOM index vectors of the unit i and the best matching unit b.

During learning the function h_{ib} normally approaches delta function, i.e., σ slowly approaches zero as training progresses. When good quantization is desired, the map should be trained with only b in h_{ib} once the map has organized (Kohonen, 1997). During this quantization stage the gain has to be sufficiently small to avoid losing the map ordering, how small exactly varies from case to case. In order to guarantee convergence of the algorithm, the gain $\gamma(n)$ should decrease as a function of time or training steps according to conditions (Ritter and Schulten, 1988):

$$\begin{aligned} \lim_{t\to\infty} \int_0^t \gamma(t')dt' &= \infty \ , \\ \lim_{t\to\infty} \int_0^t (\gamma(t'))^2 dt' &= C, \ C < \infty \ . \end{aligned} \tag{3.7}$$

If the map is trained properly, i.e the gain and the neighbourhood functions are properly decreased over training a mapping is formed, where weight vectors specify centers of clusters satisfying the vector quantization criterion:

$$E = min\{\sum_{j=1}^{M} ||x_j - w_{b(x_j)}||\}, \tag{3.8}$$

where we seek to minimise the sum squared distance E of all input patterns, x_j, $j = 1, ..., M$, to the respective bmus with weight vectors $w_{b(x_j)}$.

When the SOM is applied as vector quantization method as described above, it is closely related to k-means quantization, and becomes essentially equal to the online k-means algorithm (Flexer, 2001). The difference is the ordering of the codebook vectors in the SOM, which results from the employed neighbourhood function. Also, in k-means, the objective is to select the number of code book vectors k such that each code vector corresponds to one cluster in the observed data, while in the SOM the number of units in the map can be much larger. Furthermore, in a more typical case, the neighbourhood of the SOM is not set to exactly zero at the end of the training. In this case Equation 3.8 does not hold.

As has been discussed before, the SOM is typically applied in data exploration because of its prominent visualization properties. In visualization, the number of units in the SOM may be large, and the neighbourhood function essentially provides reguralization which allows smooth mapping of the input space. The SOM can also be employed as an intermediate method in data clustering, providing clustering via visualization. For instance, k-means can cluster the weight vectors of the SOM built from the observed data. Also other methods can be applied to find more quantative information from the generated SOM, such as summaries and interesting groupings of map units (see e.g. Vesanto and Alhoniemi (2000)).

SOM in Sequence Processing

The first popular applications of the SOM in sequence processing exploited the trajectory of the best matching units. In this approach, no changes are made to the SOM itself, rather it is used for dimensionality reduction. Subsequent processing of the *bmu* trajectories was applied for instance in the Phonetic Typewriter (see e.g.Kohonen (1989)) application, where the speech was converted into writing in real time.

The proposed methods for extending the SOM can be divided in three classes. In the first class, recurrent connections or time delay memories are added to the units in the map. The memory can store past inputs, past activations, or past errors of each unit. The map itself has a single layer structure similar to the conventional SOM. The search method for the *bmu* of the map, and the learning rule for updating the weights in the map are modified to take into account the effect of the stored past. The TKM (Chappell and Taylor, 1993) and the RSOM (Varsta et al., 1997a,b) belong to this class. These maps are discussed in Sections 3.4.2. and 3.4.3. The Recursive SOM (RecSOM) proposed in (Voegtlin, 2002), belongs also into this class, and was shown to be able to learn local representations of temporal sequences. In RecSOM, the map has one layer and each unit of the map has two weight vectors, w^x and w^y, that are compared to the input vector $x(n)$, and to the activities at the previous time step, $y(n-1)$, respectively. The *bmu* of the RecSOM is searched in the same way as in the SOM, by taking into account both weight vectors. In order to prevent unstable representation, a nonlinear transfer function updates rule of the weights w^y, which include the feedback. The weights w^x are updated as in the SOM.

In the second class, two or more SOMs are employed as a hierarchical structure. Recurrent connections or time delay memories can be used to store past activations of *bmu* trajectories of the first map, which are then fed to the subsequent maps. In (Kangas, 1994) a hierarchical model which has two layers with an integrating memory connecting the maps was proposed. The first map performs a nonlinear transformation of the data. The transformations are stored in the integrating memory, which serve as the input for the second map. In (Carpinteiro, 1999) a hierarchical SOM was proposed, which resembles the model by Kangas. The model utilizes an integrating memory in both maps, and includes a nonlinear transformation between the maps.

In the third class, the Euclidean distance measure which typically determines the *bmu*, is replaced with another operator. These operators can be defined according to the application. In (Lampinen and Oja, 1989), the Self-Organizing Map of competing autoregressive models (AR-SOM) was proposed. Here every unit of the SOM represents an autoregressive (AR) model with its reference vector. In AR-SOM, the parameters of the AR models are updated by an adaptive least mean squares method. The *bmu* search method is modified to take into account several past input vectors in order to make it more robust against outliers. Furthermore, operator maps utilizing dynamic time warping (DTW) and Hidden Markov models (HMM) have been proposed (see e.g. Somervuo (2000)). DTW is an algorithm for comparing a sequence of observations with a reference or a template sequence (see e.g. Sakoe and Chiba (1978)). In DTW-SOM each unit in the map is associated with a reference sequence, and DTW algorithm is applied for searching the *bmu*. The flexibility of the DTW matching allows comparing sequences of different lengths, which makes it appropriate for speech processing. Hidden Markov models were applied in the state space map (Somervuo, 2000) to replace the Euclidean distance operators in the map units. Here a trajectory of *bmu*s corresponding the input vector sequence is searched by Viterbi algorithm. Each input vector is then associated with certain *bmu*, which represents one of the HMM states.

In the rest of this section, two modifications to the SOM, namely the TKM and the RSOM, are presented in more detail. These maps belong to the first class of the modifications of the SOM discussed above.

3.4.2 Temporal Kohonen Map

The TKM was proposed by Chappell and Taylor (1993) as a modification to the original SOM. This modification is not only capable of separating different input patterns but is also capable of giving context to patterns appearing in sequences. In the TKM the involvement of the earlier input vectors in each unit is represented by using a recursive difference equation which defines the current unit activity as a function of the previous activations and the current input vector.

The outputs of the normal SOM are reset to zero after presenting each input pattern and selecting the *bmu* with the typical winner take all strategy, hence the map is sensitive only to the last input pattern. In the TKM the sharp outputs are replaced with leaky integrator outputs which, once activated, gradually lose their activity.

The modeling of the outputs in the TKM is close to the behaviour of natural neurons, which retain an electrical potential on their membranes with decay. In the TKM this decay is modeled with the difference equation:

$$V_i(n) = dV_i(n-1) - (1/2)\|x(n) - w_i(n)\|^2 \ , \tag{3.9}$$

where $0 < d < 1$ can be viewed as a time constant, $V_i(n)$ is the activation of the unit i at step n, $w_i(n)$ is the reference or the weight vector in the unit i and $x(n)$ is the input pattern. Now the best matching unit b is the unit with maximum activity. Equation 3.9 has the following general solution:

$$V_i(n) = -(1/2)\sum_{k=0}^{n-1} d^k \|x(n-k) - w_i(n-k)\|^2 + d^n V_i(0) \ , \tag{3.10}$$

where the involvement of the earlier inputs is explicit. Further analysis of Eq. 3.10 shows how the optimal weight vectors in the vector quantization sense can be solved explicitly when n is assumed to be sufficiently large to render the last residual term corresponding to initial activity insignificant. The analysis, when w_i is assumed constant, goes as follows:

$$\frac{\partial V(n)}{\partial w} = -\sum_{k=0}^{n-1} d^k (x(n-k) - w). \tag{3.11}$$

Now, when w is optimal in the vector quantization sense (Eq. 3.8) the derivative in Eq. 3.11 is zero as this minimises the sum in Eq. 3.10. Hence substituting the left hand side of Eq. 3.11 with 0 yields:

$$\begin{aligned} 0 &= -\sum_{k=0}^{n-1} d^k (x(n-k) - w) \ , \\ w &= \frac{\sum_{k=0}^{n-1} d^k (x(n-k))}{\sum_{k=0}^{n-1} d^k}. \end{aligned} \tag{3.12}$$

The result shows how the optimal weight vectors in the vector quantization sense are linear combinations of the input patterns. Since the TKM is trained with the normal SOM training rule, it attempts to minimise the normal vector quantization criterion in Eq. 3.8, which is other than the criterion suggested by Eq. 3.12. As a consequence it appears that it may be possible to properly train a TKM only for relatively simple input spaces.

3.4.3 Recurrent Self-Organizing Map

The RSOM was proposed in (Varsta et al., 1997a,b), and can be seen as an enhancement for the TKM. In brief, RSOM defines a difference vector for each unit of the map, which is used for selecting the *bmu* and also for adaptation of weights of the map. The difference vector captures the magnitude and direction of the error in the weight vectors and allows learning temporal context. Weight update is similar to the SOM algorithm—except that weight vectors are moved towards cumulative sum of past difference vectors and the current input vector.

Some of the problems of the original TKM can be solved by relocating the recurrent connection, giving rise to the modified TKM called Recurrent Self-Organizing Map. In the TKM, the leaky integrator memory stores past activations of each unit. Thus, the memory captures a linear sum of the squares of the past difference vectors. In the RSOM, the whole difference vector is instead included in the recurrent connection. Thus, the memory stores a linear sum of past difference vectors. While the difference between the maps may seem small, it is significant, since it allows the definition of a simple SOM-like learning rule for the RSOM (Varsta et al., 2001).

The difference vector $y_i(n)$ in each unit of the map is defined as

$$y_i(n) = (1-\alpha) y_i(n-1) + \alpha(x(n) - w_i(n)) \ , \tag{3.13}$$

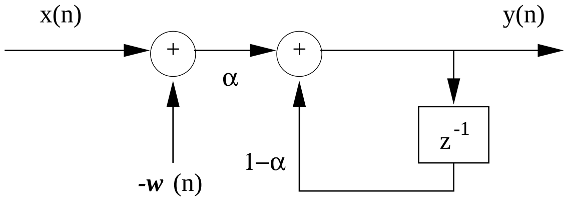

Figure 3.2: Schematic picture of an RSOM unit which acts as a recurrent filter.

Here $0 < \alpha \leq 1$ is the leaking coefficient analogous to d in the TKM, $y_i(n)$ is the leaked difference vector while $x(n)$ and $w_i(n)$ have their previous meanings. Schematic picture of an RSOM unit is shown in Fig. 3.2.

Large α corresponds to short memory while small values of α correspond to long memory and slow decay of activation. In the extremes of α RSOM behaves like a normal SOM ($\alpha = 1$) while in the other extreme all units tend to the mean of the input data.

Eq. 3.13 can be written in a familiar form by replacing $x_i'(n) = x(n) - w_i(n)$, yielding:

$$y_i(n) = (1-\alpha)y_i(n-1) + \alpha x_i'(n); \tag{3.14}$$

which describes an exponentially weighted linear IIR filter with the impulse response $h(k) = \alpha(1-\alpha)^k$, $k \geq 0$. For further analysis of the Eq. 3.14, see e.g. (Proakis and Manolakis, 1992).

Since the feedback quantity in RSOM is a vector instead of a scalar it also captures the direction of the error which can be exploited in weight update when training the map. The best matching unit b at step n is now searched by

$$y_b = min_i\{\|y_i(n)\|\}\ , \tag{3.15}$$

where $i \in V_M$. Then the map is trained with a slightly modified Hebbian training rule given in Eq. 3.6 where the difference vector, $(x(n) - w_i(n))$ is replaced with y_i. Thus the unit is moved toward the linear combination of the sequence of input patterns captured in y_i.

Repeating the mathematical analysis on RSOM earlier done with the TKM in Eqs. 3.11 and 3.12 yields:

$$y(n) = \alpha \sum_{k=1}^{n} (1-\alpha)^{(n-k)}(x(k) - w). \tag{3.16}$$

The square of the norm of $y(n)$ is

$$||y(n)||^2 = \alpha \sum_{k=1}^{n} (1-\alpha)^{(n-k)}(x(k) - w)^T(x(k) - w). \tag{3.17}$$

Optimizing the vector quantization criterion given in general form in Eq. 3.8 with respect to y yields the following condition:

$$\frac{\partial ||y(n)||^2}{\partial w} = -2\alpha \sum_{k=1}^{n} (1-\alpha)^{(n-k)}(x(k) - w) = 0, \tag{3.18}$$

when w is optimal. The optimal w can be analytically solved:

$$w = \frac{\sum_{k=1}^{n}(1-\alpha)^{(n-k)}\, x(k)}{\sum_{k=1}^{n}(1-\alpha)^{k}}. \tag{3.19}$$

From Eq. 3.19 one immediately observes how the optimal w's are linear combinations of the x's. Note how this result is essentially identical with the result in Eq. 3.12 so it might seem that the algorithms are essentially the same. However since the RSOM is trained with the y's it seeks to minimise the quantization criterion suggested by Eq. 3.18 while the TKM seeks to minimise the normal vector quantization criterion in Eq. 3.8. Nevertheless the resolution of the RSOM is limited to the linear combinations of the input patterns with different responses to the operator in the unit inputs. If a more sophisticated memory is required one has to resort to hierarchical maps or operator maps as discussed before.

A more detailed comparison of the RSOM and the TKM, both analytical and experimental, was presented in (Varsta et al., 2001). It was shown that the RSOM is a significant improvement over the TKM, because the RSOM allows simple derivation of a consistent learning rule.

Chapter 4

Case Studies in Temporal Sequence Processing

4.1 Introduction

In this chapter, the RSOM is presented as a method for TSP. Specifically, temporal sequence prediction problem is considered. A local model approach is proposed, which is based on applying the RSOM for clustering the input pattern sequence, and estimating local models corresponding to each cluster. A modification to the training algorithm of the RSOM is proposed, which enables applying the model in temporal sequence prediction. Also the methods for model selection and estimation are presented.

In order to study the properties of the proposed method, case studies are presented where linear and nonlinear models, as well as global model and local model approaches are applied in TSP. In the case studies, four different time series prediction problems are presented and the results analysed. Each time series represents a different system, which is modeled by predicting the future of the series. The time series were selected based on the type of the system that the series characterizes, the availability of long enough series to be analysed, and also the popularity of the series in other case studies presented previously. Since each case represents a different system, the performance of the model in these cases gives also valuable knowledge about its applicability in time series prediction tasks in general.

Results from the first three cases have been published in (Koskela et al., 1998b,a). Contributions of the author in these publications include the proposed episode learning for the RSOM and the implementation of the model selection. Also all presented case studies were performed by the author. The main objective here was to study the performance of the proposed RSOM model and learning algorithm, by comparing the gained prediction results with other methods. The prediction task in all cases was one-step prediction, and all series are univariate, consisting of observations of one variable. The first series represents a chaotic system, the second series consists of hourly measurements of electricity consumption, and the third series consist of measurements of the intensity of an infrared laser. From each series four cross-validation data sets were generated for model selection purposes. The same data sets were used for all models to prevent biasing the results based on different data sets.

The fourth case involves using RSOM with local models in multivariate time series prediction. The river flow is predicted based on daily measurements of three variables: river flow, temperature, and precipitation Also in this case, one-step prediction problem was considered, and cross-validation scheme was applied for model selection. The objective of this case was to study in more depth the properties of the RSOM model. For this purpose, plots and graphs are used for visualizing the model predictions. Also clustering of the input patterns, or "temporal quantization", realised by the RSOM is visualized. Prediction results gained with the RSOM model are in this case compared with earlier published results.

In the following sections, the models applied are first reviewed, and their implementation in the prediction task is discussed. Then the four cases are presented, and finally discussion based on the results is presented.

4.2 Models

The models which were applied include linear and nonlinear regression models. The basic AR model was selected to act as the "baseline" model and the MLP was employed as a nonlinear model. The RSOM with local linear models is compared to these baseline models. The more advanced neural network methods, such as the TDNN or recurrent NN, were not considered in these comparisons. Reasons for this include restricting the number of models, and relatively high complexity of the parameter estimation and model selection processes involved in these models. In the next sections, the usage of the selected models in time series prediction is presented.

4.2.1 Recurrent SOM

The RSOM was discussed in Section 3.4.3. In this section, implementation of the RSOM in TSP is presented. In the RSOM the feedback in the difference vectors allows the past to affect the mapping that is formed, thus it can be described to realize "temporal quantization" of the input vectors. In the conventional SOM, only the current input vector affects the mapping. The RSOM can separate input vector sequences based on the context stored in the difference vectors. The limitations of the stored context were analysed in (Varsta et al., 2001). Essentially, the context stored in the difference vectors is limited to linear combinations of the past input vectors.

The best matching unit (bmu) of the RSOM at each time is selected according to the (Euclidean) norm of the difference vectors. Hence the selection of the bmu is dependent on every input fed to the map starting from the beginning of the series. This suggests that the updating rule for the parameters of the map should include the whole history of the inputs. In this case, however, it is noticed that the effect of the feedback goes quickly toward zero in time. Since the computations are implemented with limited accuracy, this occurs with certain number of iterations. Hence the difference vector can be approximated by using certain number of past samples of the input vectors.

Based on this approximation, the learning algorithm of RSOM is implemented as follows. The map is presented an *episode* of consecutive input vectors starting from a random point in the data. The number of vectors belonging to the episode is dependent on the leaking coefficient α of the units. At the end of the episode, the impulse response $h(k)$ of the recurrent filter (see Fig. 3.2) is below 5 % of the initial value. The bmu is selected at the end of the episode based on the norm of the difference vector. The updating of the vector and its neighbours is carried out as in Eq. 3.6. After the updating, all difference vectors are set to zero, and a new random starting point from the data is selected. The above scenario is repeated until the mapping has formed.

Figure 4.1 shows the procedure for building the models and evaluating their prediction abilities with testing data. The time series is first divided to training and testing data. Input vectors to RSOM are formed by delay coordinate embedding, i.e. “windowing” the series. For model selection purposes the training data is divided to cross-validation data sets. Here v-fold cross-validation is applied, in which the training data is first divided to v disjoint subsets. Cross-validation error then gives an estimate of the generalization error of the model.

In training phase the free parameters of RSOM include length of the input vectors, time step between input vectors, leaking coefficient in the units and the number of RSOM units. The episode length is dependent on the leaking coefficient as described earlier. After the training the RSOM is employed to divide (cluster) the data into local data sets according to the bmu on the map. Linear regression models, that have the same number of parameters as the vectors of RSOM units, are then estimated using these local data sets. The prediction error for each model consisting of RSOM and associated local model is estimated using cross-validation error. Finally, the best model according to cross-validation error is selected. The selected model is trained again with the whole training data. This model is then applied in predicting the test data set that has not been presented to the model before.

The RSOM with local linear models were estimated as follows: The input vector length p, time step between consecutive input vectors s, number of units

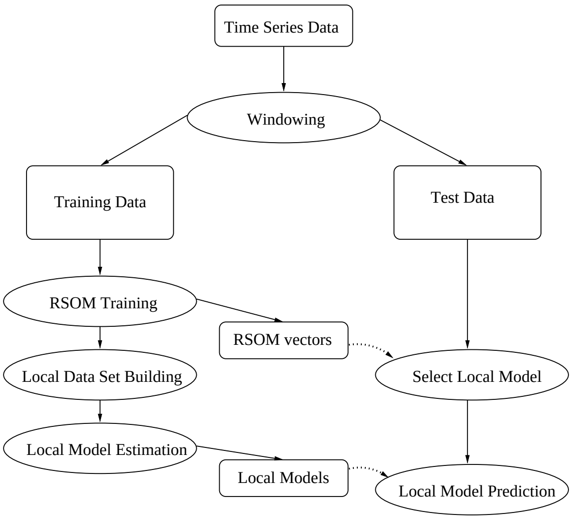

Figure 4.1: Schematic picture showing the procedure for building the RSOM and local models, and evaluating their performance with the test data.

n_u and the leaking coefficient α of the units in the map were free parameters giving rise to model $RSOM(p, s, n_u, \alpha)$. In the studied cases the number of RSOM units was varied as $n_u \in \{5, 9, 13\}$, and the leaking coefficient was varied as $\alpha \in \{1.0, 0.95, 0.78, 0.73, 0.625, 0.45, 0.40, 0.35\}$ corresponding to episode lengths 1, ..., 8. The time step between consecutive input vectors was varied as $s \in \{1, 3, 5\}$ while the length of the input vector p was varied differently in the cases as described later. The number of linear regression models equals the number of RSOM units n_u. The order of the regression models equals the length of the vectors p in RSOM units. Local model parameters were estimated with least squares algorithm, by using the data for which the corresponding RSOM unit was the bmu.

It should be noted that the RSOM corresponds to the conventional SOM when the value of the leaking coefficient α equals 1, and the episode length is also 1. In the subsequent prediction cases, the results were also reported for this case. The benefits of the RSOM compared to the SOM in the presented cases can be estimated from these results.

4.2.2 Multilayer Perceptron Network

The MLP was selected as the baseline global nonlinear model for the time series prediction cases. Using the MLP in TSP was discussed in Section 3.3.2. Here the time series is converted by delay coordinate embedding into input patterns, and consequently static MLP architecture and learning algorithms can be applied.

Using the MLP in this way corresponds to using a TDNN, which has tapped delay lines only in the input layer. The benefits and restrictions of time delay memories apply also in this case. More specifically, in the MLP the information between consecutive input patterns is lost. Also the length of the time context which can be represented is restricted by the length of the input vectors.

However, in spite of the restrictions, the MLP has been successful in many time series prediction cases (see e.g. Gershenfeld and Weigend (1993)). Furthermore, it has been proved that static MLP model implements some nonlinear generalization of usual statistical AR predictors (Lapedes and Farber, 1987). Thus, the MLP can be employed as a nonlinear replacement of the conventional linear models. Since the MLP is a semiparametric model, it can be adjusted according to the complexity of the modeled process. If the modeled process contains nonlinear dependencies, then the MLP can provide more accurate predictions than a linear model. Another benefit of the MLP is the ability to "degrade gracefully" in the case of unrepresentative input patterns (not seen in the training data). With appropriate training, the mapping from the input space to the output space is smooth, and thus the MLP can typically interpolate and extrapolate better than a linear model.

In the following cases, the MLP was implemented as follows. The weights of the network were estimated with Levenberg-Marquardt learning algorithm. An *MLP(p,s,q)* network with one hidden layer, p inputs and q hidden units was used. Here s is the time step between consecutive input vectors. In order to make comparison of the results fair, parameters p and s were chosen to be the same as in RSOM models. The number of hidden units was varied as $q \in \{3, 5, 7, 9\}$. The training was performed five times for each *MLP(p,s,q)* model and for each cross-validation data set with different initial weights. The smallest errors of each fold of the cross-validation data sets were selected to get an estimate of the prediction error. The model that gave the smallest cross-validation error was then trained again with the whole training data, and evaluated with the test data.

4.2.3 Autoregressive Model

The Autoregressive (AR) model was selected as the baseline global linear model for the time series prediction cases. The history of using AR models in time series prediction, system identification and control is long, starting from the 1960's. The properties of linear models are well known, and efficient algorithms exist for parameter estimation. Also, for AR and ARMA models, procedures for model selection in time series prediction have been proposed, and are widely applied (see e.g. Box et al. (1994)).

AR models relate the current value of a series $x(n)$, to values of the series exhibited in the past, $x(n-k) : k = 1, 2, ..., p$. Autoregressive model of order p, denoted $AR(p)$, is defined by

$$x(n) = a_0 + \sum_{k=1}^{p} a(k)x(n-k) + \epsilon(n) \ , \tag{4.1}$$

where the model parameters $a(k)$, autoregressive coefficients, do not vary with time. $\epsilon(n)$ is the noise term consisting of series of independent, zero mean, constant variance, normally distributed stochastic variables. *AR(p)* models with p inputs were estimated by using the least-squares algorithm. The order of the AR model was varied as $p \in \{1, ..., 50\}$. The same cross-validation scheme as with other models was applied. In this case the prediction errors serve as an example of the accuracy of a global linear model in the current tasks.

4.3 Mackey-Glass Chaotic Series

Description

The first of the test cases is the Mackey-Glass time series, produced by a time-delay difference system of the form (Mackey and Glass, 1977):

$$\frac{dx}{dt} = \beta x(t) + \frac{\alpha x(t-\gamma)}{1 + x(t-\gamma)^{10}} \tag{4.2}$$

where $x(t)$ is the value of the time series at time t. This system is chaotic for $\gamma > 16.8$. In the present test the time series was constructed with parameter values $\alpha = 0.2$, $\beta = -0.1$ and $\gamma = 17$ and it was scaled between [-1,1]. From the beginning of the series shown in Fig. 4.2 3000 samples was selected for training, and the rest 1000 samples were left for testing. For RSOM and MLP models length of the input vector was varied as $p \in \{3, 5, 7\}$.

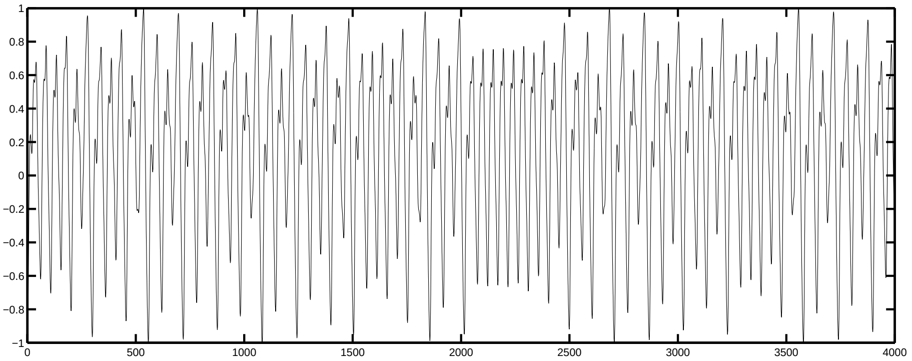

Figure 4.2: Mackey-Glass chaotic time series.

Prediction Results

The sum-squared errors gained for one-step prediction task are shown in Table 4.1. The *MLP*(3,1,7) model gives the smallest cross-validation error but fails to predict the test set accurately. This result suggest that the model selection procedure was not successful in this case. For the *AR*(2) model the results are opposite. AR model does not model here the underlying phenomena, instead it predicts the next value of the series using mainly the previous value. This scheme gives in this case the best prediction accuracy. *RSOM*(3,1,5,0.95) gives moderate accuracy for both cross-validation and test data sets. Closer inspection revealed that most of the error were due to large prediction errors for the first samples of the series, while the rest of the series were predicted with similar accuracy to the AR model. It is noted that the conventional SOM with local models yields almost identical results to RSOM. This was expected since in the RSOM with leaking coefficient value 0.95 the stored temporal context is quite limited.

Table 4.1: One-step prediction errors for Mackey-Glass time series.

	CV Error	Test Error
SOM(3,1,5)	6.6353	3.1162
RSOM(3,1,5,0.95)	6.5556	3.1115
MLP(3,1,7)	0.3157	3.7186
AR(2)	4.1424	1.600

4.4 Electricity Consumption

Description

Electricity consumption series contains measured load of an electric network. Measurements contain hourly consumption of electricity over a period of 83 days (2000 samples). The series was rescaled between [-1,1]. From the beginning of the series shown in Fig. 4.3, 1600 samples were selected for training, and the rest 400 samples were left for testing. For RSOM and MLP models length of the input vector was varied as $p \in \{4, 8, 12\}$.

Prediction Results

The sum-squared errors gained for one-step prediction task are shown in Table 4.2. The electricity consumption series contains 24 hours long cycle and also slower trend and noise in the form of measurement errors.

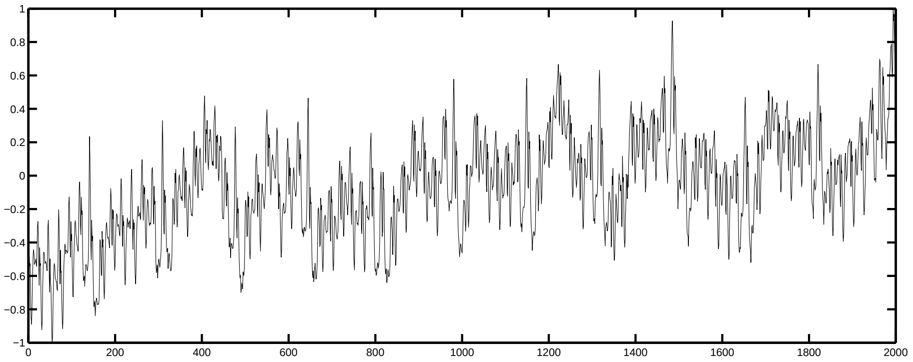

Figure 4.3: Electricity consumption time series.

AR(30) model is found to reach quite acceptable results, due to the fact that model includes the whole 24 hour cycle. As the results with *MLP*(8,1,9) model show, nonlinear model can reach better predictions with a shorter window length. In this case *RSOM*(8,1,13,0.73) model does not give any improvement due to the insufficient input vector length used in model estimation. Furthermore, with conventional SOM model the cross-validation error is almost the same as with the RSOM, but with a shorter input vector. However, results with the test data show that the RSOM can reach more accurate predictions.

Table 4.2: One-step prediction errors for electricity consumption time series.

	CV Error	Test Error
SOM(4,1,13)	18.2537	3.1231
RSOM(8,1,13,0.73)	18.0673	2.6735
MLP(8,1,9)	7.6007	1.4345
AR(30)	6.5698	2.1059

4.5 Laser Fluctuations

Description

Laser time series consists of measurements of the intensity of an infrared laser in a chaotic state. Explicit measurement conditions and the theory of the chaotic behaviour in the series is given in (Weigend and Gershenfeld, 1993). The data is available online [1]. The training data contains first 2000 samples of the laser series as depicted in Fig. 4.4. The rest 1000 samples were left for the testing. Both series were scaled between [-1,1]. For RSOM and MLP models length of the input vector was varied as $p \in \{3, 5, 7\}$.

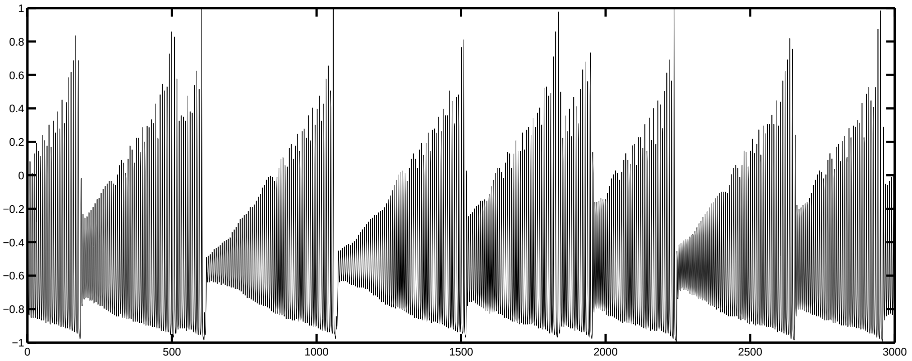

Figure 4.4: Laser time series.

Prediction Results

The sum-squared errors gained for one-step prediction task are shown in Table 4.3. The laser series is highly nonlinear and thus the errors gained with *AR*(12) model are considerably higher than for other models. The series is also stationary and almost noiseless, which explains the accuracy of the *MLP*(9,1,7) model predictions. In this case *RSOM*(3,3,13,0.73) gives results that are better than with AR model but worse than with MLP model. SOM with local models seem to perform similarly to RSOM according to cross-validation error, but with the test data the predictions with RSOM are substantially more accurate.

Table 4.3: One-step prediction errors for laser time series.

	CV Error	Test Error
SOM(3,1,13)	15.4727	12.1527
RSOM(3,3,13,0.73)	14.6995	7.3894
MLP(9,1,7)	4.9574	0.9997
AR(12)	69.8093	29.8226

[1] http://www-psych.stanford.edu/%7Eandreas/Time-Series/SantaFe/ containing files A.dat (first 1000 samples) and A.cont (as a continuation to A.dat 10000 samples)

4.6 Riverflow Series

Description

The fourth time series case involves predicting the river flow based on observations of the flow, mean temperature and precipitation. The mean daily flow of Jökulsa Eystri River in Iceland was observed during the period Jan. 1, 1972 - Dec. 31, 1974. Daily precipitation and mean daily temperature were observed in Hveravellir. There are 1096 observations of all three variables. An important hydrological feature of this river is that there is a glacier on the drainage area. Consequently, temperature has certain influence on the riverflow besides melting the snow. These datasets has been analysed by various nonlinear modelers. For further information on the data, see the book by (Tong, 1990). The data is available online in the Time Series Data Library [2]. Figure 4.5 shows the riverflow (m^3/s), temperature (degrees of Celsius), and precipitation (mm/day).

The riverflow depends on the previous flow, the precipitation and the temperature. During winter, precipitation does not affect the riverflow, since the rain consist of snow. During summer, precipitation will affect the riverflow after certain delay. During spring, when the temperature is below 1 degrees, riverflow is not significantly affected. When the temperature rises above 1 degrees, the riverflow on the next day increases substantially due to snow melting.

RSOM Models

For this case, only RSOM model was applied. Prediction results gained with other models are summarized in Table 4.6 and referred to (Chen and Tsay, 1993).

The input variables of the model were chosen based on the previous results. The input vectors for the RSOM contained ordered triplets of temperature t, precipitation p and flow v. The number of these triplets was varied as $\{2, 3, 4\}$, and the length of the RSOM input vectors was $p \in \{6, 9, 12\}$, respectively. For instance, the input vector containing observations from two previous days was $x(k) = \{t(k-2), p(k-2), v(k-2), t(k-1), p(k-1), v(k-1)\}$, where k is the time index. The desired output of the model was $y(k) = v(k)$. Stepsize between consecutive input vectors was not varied in this case. The time index k went through all the values in the sample $k = \{1, 2, ..., N\}$.

The number of RSOM units was varied as $n_u \in \{5, 9, 13\}$ and the leaking coefficient $\alpha \in \{1.0, 0.95, 0.78, 0.73, 0.625, 0.45, 0.40, 0.35\}$ corresponding to episode lengths $\{1...8\}$ as with the previous cases presented. Data from the first two years was selected for training, and the third year was kept aside for testing. All series were scaled to unit variance. The training data set was used in estimating the variance of each variable.

[2] http://www-personal.buseco.monash.edu.au/%7Ehyndman/TSDL/

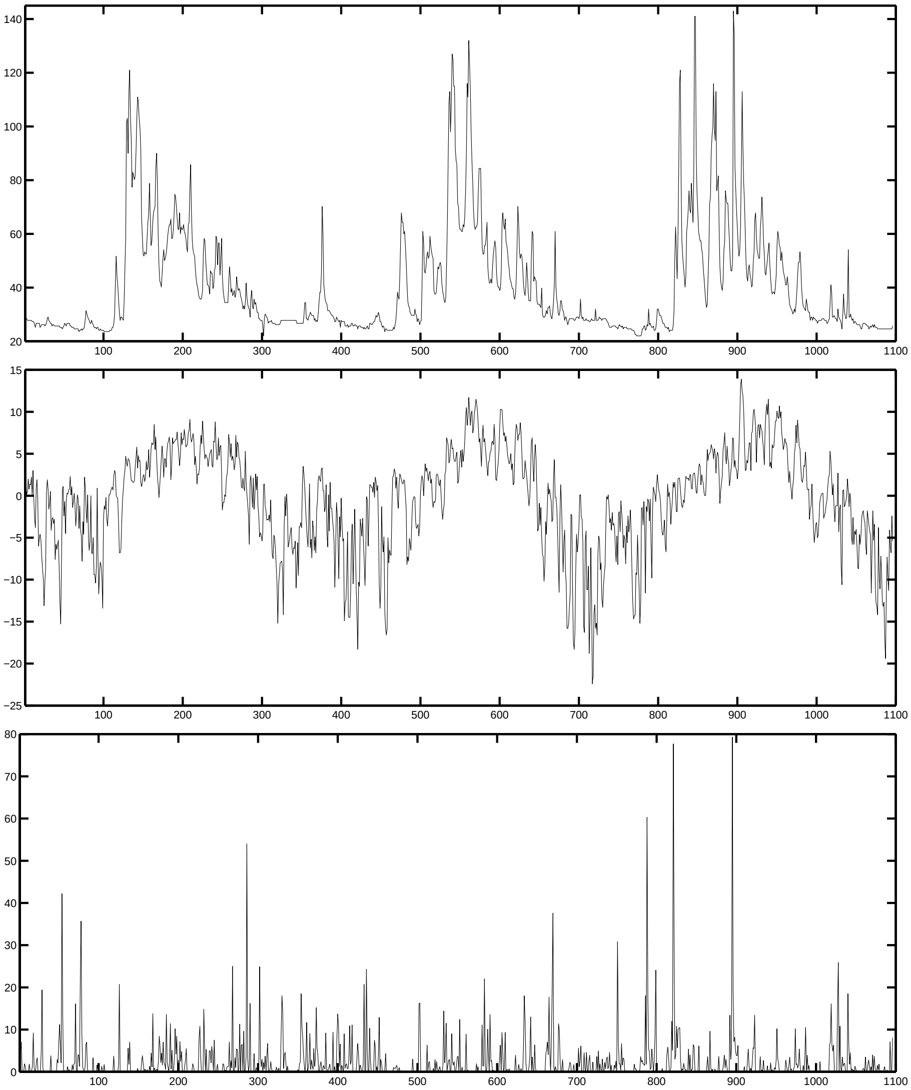

Figure 4.5: From top to bottom: Daily riverflow (m^3/s), daily temperature (Celsius) and daily precipitation (mm/day).

Prediction Results

4-fold cross-validation errors and test data errors are shown in Table 4.4. In order to conform with previously published results (Chen and Tsay (1993)), mean squared error is reported. The best models using observations from 2, 3 and 4 previous days as inputs are shown. As with previous cases, the best models according to cross-validation error were built again by using the whole training data, and applied in one-step prediction of the test data. The smallest cross-validation error is gained with model RSOM(6,1,5,0.78), which includes 2 previous values from each variable as the input, has 5 local models, and the leaking coefficient α has value 0.78.

Table 4.4: One-step prediction errors for river time series.

	CV Error	Test Error
RSOM(6,1,5,0.78)	43.76	70.29
RSOM(9,1,5,0.95)	49.84	69.81
RSOM(12,1,5,0.35)	49.27	77.21

Figure 4.6 shows how the RSOM clusters the training data, in the case where the input vectors contain four previous values of each variable. The clustering is presented for model RSOM(12,1,5,0.35), which gave the smallest cross-validation error in this case. In the figure, each variable, temperature t, precipitation p, and flow v is visualized separately. The weights of the RSOM units are plotted with different markers. The mean value of the input patterns for each unit is plotted with solid line, and the interquantile range (25% and 75%) is plotted with dotted line. Precipitation is plotted in logarithmic scale. From the figure it can be seen, that RSOM units are ordered according to temperature (top row) and flow (bottom row). For precipitation such ordering is not achieved, instead unit 3 represents a wide range of values.

Following figures are presented for the test data, which was not used during the model estimation. The results from the best model according to cross-validation error, RSOM(6,1,5,0.78), are viewed in these visualizations. Figure 4.7 shows the scatterplot of the flow as a function of the temperature, and Figure 4.8 as a function of the precipitation for the test data. The division of the data for the local models according to RSOM(6,1,5,0.78) is shown with different markers. Also in this case, the RSOM has 5 units, and thus there are 5 local models. It can be seen from the figures, that local models 5 and 4 are responsible for lower temperatures, when the rain consist mostly of snow. Model 3 is responsible for the heavy rain periods, while the model 1 is applied during flood. Model 2 seems to be a more general model. These results agree with the clustering achieved for the training data shown in Figure 4.6, where four previous values of each variable were included in the model.

Figure 4.9 and Figure 4.10 show the the change in riverflow between current and previous day divided by the change in precipitation and temperature, respectively, as a function of the temperature for the current day. From these figures it can further be seen, that local model 3 is also responsible for predicting the changes in the flow when the temperature is below zero, and that the flow during

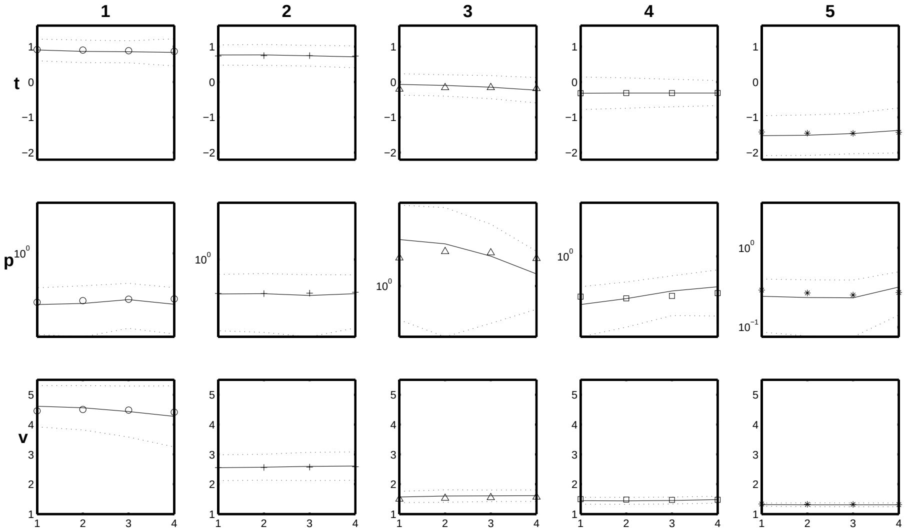

Figure 4.6: Clustering of the training data realised by the RSOM(12,1,5,0.35) model. Input vector contains four values of each variable. Temperature t is shown on the top, precipitation p in the middle, and flow v at the bottom row. Columns represent each RSOM unit (1...5), respectively.

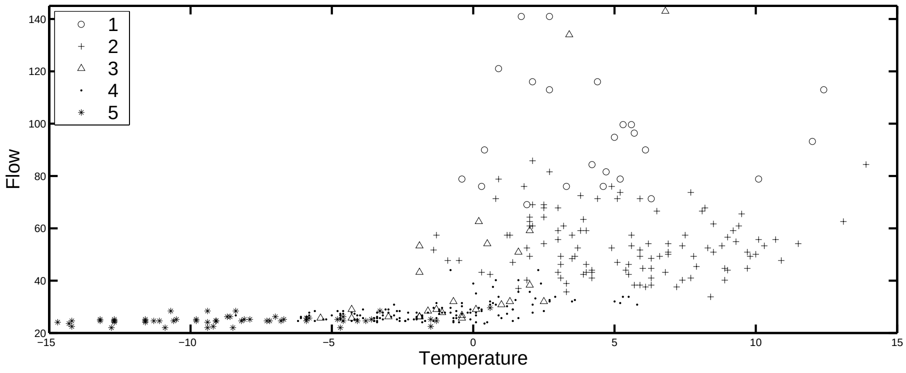

Figure 4.7: The flow as a function of temperature showing the corresponding RSOM local models for the test data.

the warmest days is predicted by local model 2.

Figure 4.11 shows the RSOM model predictions for the spring period in the test data. River flow is plotted with solid line and dots marking the observed data, and model predictions with dashed line. The local model applied in each prediction is indicated with different markers. From the shown time period it is

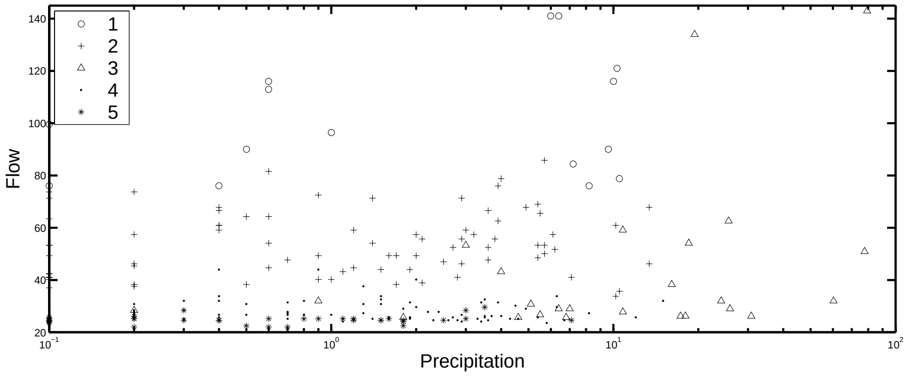

Figure 4.8: The flow as a function of precipitation showing the corresponding RSOM local models for the test data. The precipitation is plotted on logarithmic scale.

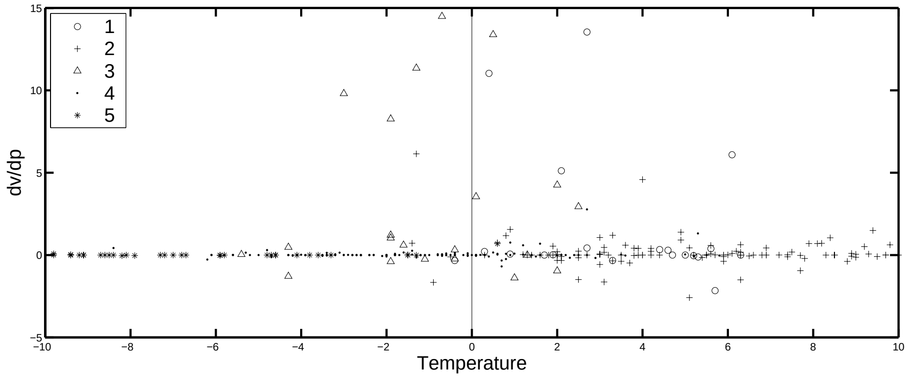

Figure 4.9: $\partial v / \partial p$ shown as a function of temperature calculated from the test data.

noticed that the largest prediction errors occur during the floods. The first flood in the figure is due to snow melting, and the second is mostly due to heavy rainfall.

Figure 4.12 shows the autocorrelation estimates for the prediction error residuals calculated from the test data. There seems to be a noticeable correlation between the prediction errors and the river flow with 2 days delay. However, closer inspection of the prediction errors revealed that the estimate was inaccurate. Since the test data contained only 365 samples, the two largest prediction errors, also visible in the Figure 4.11, caused the estimate to be inaccurate.

In order to analyze the models more thoroughly, one-step prediction errors were also calculated for other RSOM models, which were rejected based on the cross-validation criterion. Models giving the smallest error for testing data are shown in Table 4.5.

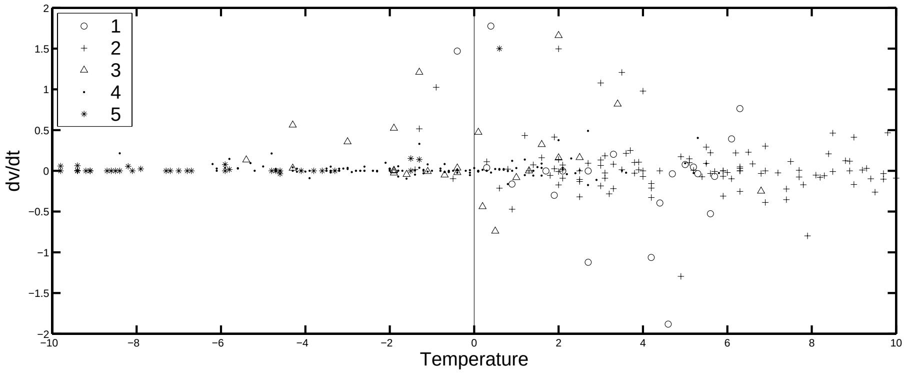

Figure 4.10: $\partial v/\partial t$ shown as a function of temperature calculated from the test data.

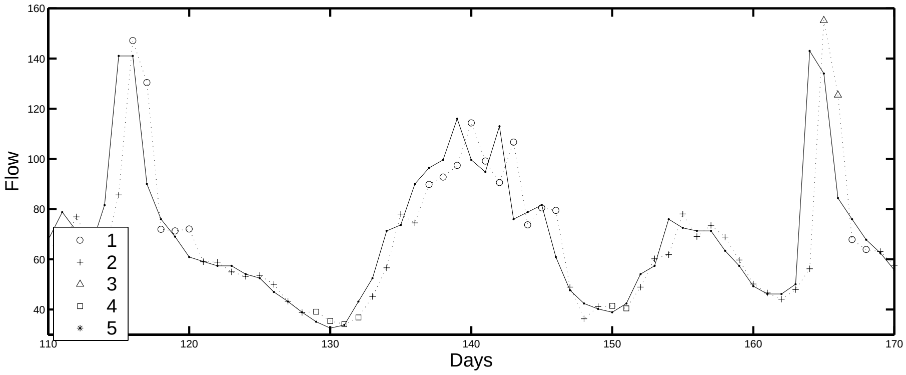

Figure 4.11: RSOM model predictions for the riverflow in the test data, showing the largest prediction errors.

It is noted that the best model according to cross-validation criterion does not give the best performance for the test data. Also, cross-validation seems to favor smaller models, since the number of RSOM units was 5 in all three cases. These results can be explained to be due to insufficient or unrepresentative training data,

Table 4.5: Smallest one-step prediction errors for river time series.

	Train Error	Test Error
RSOM(6,1,13,0.73)	17.96	65.75
RSOM(9,1,9,0.73)	18.06	67.76
RSOM(12,1,9,0.73)	17.19	63.36

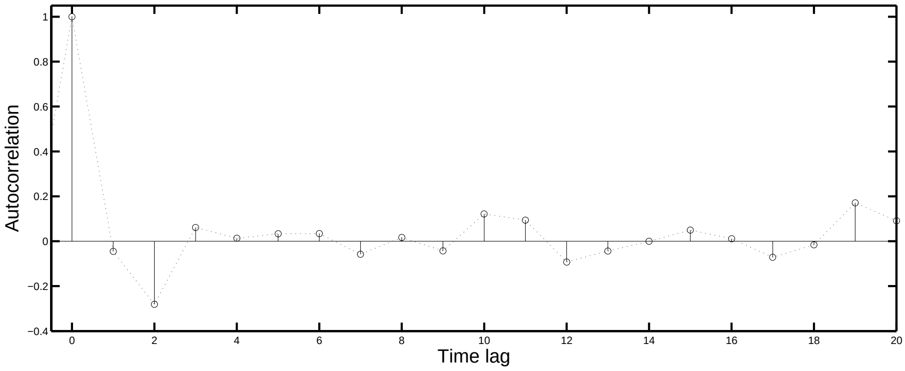

Figure 4.12: Autocorrelation estimates for the RSOM model prediction error residuals calculated from the test data.

or inaccurate model selection procedure.

It is evident that the training data set is somewhat unrepresentative. The test data set contains more rainfall than the training data, and also the variance of the precipitation is larger in test data. Specifically, during three days the rainfall exceeds all previously observed values. Prediction errors for these days only can dominate the overall result, and add to the uncertainty of the model selection process.

Results with 4-fold cross-validation may be inaccurate due to splitting the data to four large blocks for the cross-validation. As a consequence, the training data portion for each 4-fold case doesn't have equal amount of data from each season of the year. Taking this into account, a better suited strategy could be splitting the data by using a smaller block size for the cross-validation, and using e.g. 3 months/weeks for the training, and leaving the 4th month/week for the testing. However, smaller block size for cross-validation was not tested, since it would have reduced the usable training data and increased computational burden and complexity of the model selection. Having more blocks reduces the number of available training data, since the first samples in the block cannot be used in estimating the model parameters. This is because the episode learning of the RSOM. Also, the model selection procedure becomes more complicated and time consuming when the number of blocks increases. Recall that RSOM weights have to be estimated first, after which the local model parameters can be estimated. These steps have to be repeated v times for v-fold cross-validation.

One-step prediction errors for Linear, TAR and NARX models (presented in Chen and Tsay (1993)) and for RSOM model are compared in Table 4.6. While the TAR and the NARX models applied also current values of the precipitation and temperature in predicting the river flow, in RSOM only observations from the previous days were included. Taking this into account, as well as the best results

Table 4.6: Comparison of linear, TAR, NARX and RSOM models.

Linear	TAR	NARX	RSOM
81.99	66.67	65.52	70.29

gained with RSOM for the test data, it can be argued that the prediction ability of the RSOM with local models is similar to the best piecewise linear models.

4.7 Discussion

Time series prediction using the RSOM with local linear models has been presented. For the selected prediction tasks this approach gives promising results.

However, the prediction accuracy of the RSOM model in the one-step prediction tasks was in some cases somewhat disappointing. It can be argued that the prediction accuracy of the model should be at least the same as with pure linear models. Due to the selection of the RSOM parameters, this was not achieved in all cases. However, in the case of highly nonlinear laser series the RSOM model gave considerably better prediction results than linear models. Also, in the studied cases the MLP seems to perform better than the RSOM. This is mainly due to the selected one-step prediction problem, where the MLP could yield the underlying mapping effectively. Another reason for the RSOM not achieving as good results as the MLP is the linear models employed.

The prediction results of the RSOM model were in all studied case better than with models using the conventional SOM as the clustering method. These results can be explained to be due to at least two characteristics of the RSOM. Firstly, it can be argued that the RSOM finds underlying dependencies between the consecutive input vectors, thus providing a better representation of the temporal context. This representation is then utilized by the local linear models, and ultimately leads to better accuracy in predictions. Secondly, it can be argued that the RSOM is less sensitive to outliers and noise in the data, since the selection of the bmu is dependent on several past input vectors. The recurrent connection acts as a filter, dampening the effect of noise in the sequence of input vectors, leading to a more robust selection of local model at each time step.

Compared with other clustering algorithms, both RSOM and SOM provide more attractive visualization possibilities. In the case studies, visualizations based on the topology of the map were not exploited to the full potential of the SOM. For instance, in the case of two-dimensional map, the state changes of the process can be visualized with the location of the bmu in the map as a function of time. It could be expected that this visualization capability is perhaps the most important point in using the RSOM instead of other clustering algorithms.

Another attractive property of the RSOM is the unsupervised learning of the context from the data. It allows building models from the data with only a little *a priori* knowledge. The selection of the value of the leaking coefficient α can be handled as model selection problem, by applying cross-validation or other means for estimating the generalization ability of the model.

Chapter 5

Application in Web Cache Optimization

5.1 Introduction

In this chapter, a novel approach to Web cache optimization is proposed. The method is based on applying a statistical model to classify cache objects based on syntactic features extracted from the objects. The main content of this chapter has been published in (Koskela et al., 2003). The author acted in a leading role in all aspects of this publication. Contributions of the author include the proposed approach for Web cache optimization, conducting a case study where this approach is tested, running all simulations and analysing the results.

Web caching is a technique which aims to reducing WWW network traffic and improving response time for the end users. Modern computers and their CPUs, operating systems and distributed file systems have caches for data accesses between clients and servers. In all of these systems, caches work well because it is very likely that some piece of data will be accessed repeatedly. Access delays are minimised by keeping popular data close to the entity which needs it.

The same holds true for the Web as well. Especially proxy caches have some very attractive advantages when implemented correctly. The proxy caches can be accessed simultaneously and they can be shared by many users. They are often located near network gateways to reduce the bandwidth required over expensive dedicated Internet connections. As a disadvantage, caching can lead to increased latency or to providing outdated documents to users. Also the rate of change in today's Web is very fast, which tends to make the caching less attractive (Douglis et al., 1997).

Since the storage capacity of the cache is limited, cached objects must be removed from the cache when new objects are brought in. The decision which objects to remove is made by a caching policy algorithm (also called replacement policy or removal policy). The problem is to find the value of an object, and whether it should be cached, without dramatically increasing computation and communication overheads. Most Web caches still apply traditional heuristic rules as caching policies. For instance, Least Recently Used (LRU) is simple to implement and efficient in the case of uniform size objects, like in the memory cache. However, since it does not consider the size or the download latency of documents, it does not perform well in Web caching (Williams et al., 1996). Consequently, designing a better suited caching policy that would optimize the performance of the Web cache has been under a continuous research. Several different algorithms have been proposed for this task (Williams et al., 1996; Cao and Irani, 1997; Rizzo and Vicisano, 2000).

Most of the caching policies estimate a value (or cost) for each cache object. When the cache is full, objects with the lowest value are first removed from the cache. There is a number of different ways to estimate the object's value. The simplest rules, such as LRU, take into account only the request statistics of the object. The more advanced policies try to estimate a more realistic value of the object by utilizing some statistics from the network and the server involved.

Both semantic and syntactic information from the cached object might be utilized in the caching algorithm. In the syntactic case information can be gathered from the HTTP (Hypertext Transfer Protocol) responses of the Web server and from the HTML (Hypertext Markup Language) structure of the object. Extracting syntactic information does not involve complex calculations and decisions about the type of the features. On the other hand in applications such as document categorization, semantic information is frequently extracted (Kohonen et al., 2000). However, in the case of the caching algorithm, extracting semantic features e.g. word frequencies from each cached object would not be sensible, hence this approach is not discussed further.

In this chapter, a method which is based on using all useful syntactic information in the caching algorithm is proposed. The proposed method is based on using a model to predict the value of each cache object. If these values can be predicted correctly, usage of the cache can be optimized irrespective of the objectives. Inputs of applied models consist of numerical features which are extracted from the HTTP responses of the WWW server and from the HTML structure of the document. Using only syntactic information simplifies the model building and makes the implementation of the model more efficient.

In the next sections Web cache optimization is first introduced, and the proposed approach to Web cache optimization is presented. Then a case study is presented, where different models classify HTML documents according to their popularity. Finally, some further issues about applying the proposed approach in Web caches are discussed.

5.2 Web Cache Optimization: A Short Review

Web cache optimization involves designing the caching policy such that the chosen set of objectives are achieved. Typical objectives of Web caching include reducing network traffic, alleviating server bottlenecks, minimising user access latency, and maintaining transparency to the end user. A conventional way of designing the caching policy is described next. It can be called a heuristic approach, since the caching policies applied are typically heuristic rules whose parameter values are optimized. It can also be referred as an empirical approach, since the optimization is made based on the measured or simulated Web cache access log traces.

The design of the caching policy algorithm can be considered to have three stages: In the first stage, the objectives of caching are chosen. The main objectives usually include minimising the access latency for the users or minimising the outbound traffic volume. In the second stage, the performance metrics which describe the cache operations are chosen. Typical metrics are described in more detail in the next section. For instance, *hit rate* represents what percentage of the requests is served straight from the cache, without causing outbound network traffic. The performance metrics are chosen according to the objectives set. In the third stage, experiments with different caching policy algorithms are made, and the one which maximises chosen performance metrics is selected. Five proposed caching policies are described at the end of this section. Experiments can be carried out as simulations using real measurements, or using simulated data. Another possibility is running experiments in a real environment. This procedure must be repeated if there are changes in the objectives, network or server resources, or traffic patterns.

5.2.1 Performance Metrics

In order to define the most popular performance metrics in more detail, let N be the total number of requests (objects), b_i the size of the i-th request and let $\delta_i = 1$ if the request i is in the cache while $\delta_i = 0$ otherwise.

The request hit rate H^{r} is then defined as the percentage of documents that are in the cache (Williams et al., 1996),

$$H^{\mathrm{r}} = \frac{\sum_{i=1}^{N} \delta_i}{N} . \tag{5.1}$$

Then the relative request hit rate is defined as $H^{\mathrm{r}}/H^{\mathrm{r}}_{\mathrm{max}}$, where $H^{\mathrm{r}}_{\mathrm{max}}$ is the request hit rate obtained for a cache with an infinite size. The byte hit rate H^{b} is the percentage of bytes transferred from the cache, defined as follows

$$H^{\mathrm{b}} = \frac{\sum_{i=1}^{N} b_i \delta_i}{\sum_{i=1}^{N} b_i} . \tag{5.2}$$

Then relative byte hit rate is $H^{\mathrm{b}}/H^{\mathrm{b}}_{\mathrm{max}}$, where $H^{\mathrm{b}}_{\mathrm{max}}$ is the request hit rate obtained for a cache with an infinite size. The latency ratio H^{t} is defined as the ratio of the sum of download time of missing documents over the sum of all downloading latencies,

$$H^{\mathrm{t}} = \frac{\sum_{i=1}^{N} t_i (1 - \delta_i)}{\sum_{i=1}^{N} t_i}\,, \tag{5.3}$$

where t_i is the time to download the i-th referenced document from its server to the cache.

5.2.2 Caching Policies

The key issue in maximising the selected performance metrics is to apply a suitable caching policy. Several of them have been proposed in the literature, and some of them being implemented in real systems (e.g. LRU in Squid Squid (2000)), and some others being proved to be optimal using simulations with actual cache access logs (Davison, 1999). However, most of these policies are suboptimal since they only optimize a particular performance metric. Some policies also include statistics from request rates and server response times, which allows adaptation to changing conditions in the network or server resources (Cao and Irani, 1997; Rizzo and Vicisano, 2000). Below five different policies and their main advantages and disadvantages are reviewed.

- LRU: The Least Recently Used documents are removed first.

  Advantages: Efficient in the case of uniform objects like in the memory cache, applied in other areas of caching for decades. Simple to implement.

  Disadvantage: Does not consider size or download latency of documents.

- LFU: The Least Frequently Used documents are removed first.

  Advantages: Simplicity.

  Disadvantage: Does not consider size or download latency of documents and may keep obsolete documents indefinitely in the cache.

- SIZE (Williams et al., 1996): Big documents are removed first.

  Advantages: Removes big documents, therefore keeps a lot of small files in the cache, resulting in high request hit rate.

  Disadvantages: May keep small documents indefinitely in the cache (even if they are never accessed again). Low byte hit rate.

- GD-SIZE (Cao and Irani, 1997): Greedy-Dual algorithm. Each object is associated a value H. When an object is brought into the cache, H is set to $1/s$ where s is the size of the object. When a replacement has to be made, the object with the smallest H ($H_{\min}$) is removed from the cache, and every object has its H reduced by $H_{\min}$. If an object is accessed again, then its H is restored to its cost.

Advantages: Tries to remove objects which are no longer requested by clients, and therefore to overcome the drawback of $SIZE$ policy. More general cost functions can also be applied. However, the simple $1/s$ variant yielded the best performance in (Cao and Irani, 1997).

Disadvantages: Does not take into account the delays induced by the network and the frequency at which documents are accessed.

- LRV (Rizzo and Vicisano, 2000): LRV estimates for each object in the cache the *relative value*,

$$V = \frac{C}{B} P_r \,, \tag{5.4}$$

where B is the benefit and C is the cost involved in purging the object from the cache, and P_r is the estimated re-access probability for the object. When a cost model based on traffic is chosen, C/B is approximately constant and denotes the overhead in traffic when the object is fetched to the cache. Thus, V is directly proportional to re-access probability P_r of the object. P_r is estimated as follows:

$$P_r(i,t,s) = \begin{cases} P(1,s)(1 - D(t)), & \text{if } i = 1 \\ P(i)(1 - D(t)), & \text{otherwise} \end{cases} \tag{5.5}$$

Here i denotes the number of previous accesses, t is the time since the last access and s is the size of the object. The dependency of $P_r(i,t,s)$ on s is neglected when $i > 1$. $P(i)$ is the probability that the object is re-accessed again after the i-th access and $P(1,s)$ is the re-access probability for a new object whose size is s. $D(t)$ is estimated time between accesses. Its calculation involves estimating values for three parameters. $P(i)$ is estimated by partitioning the objects in a small number of sets according to the parameter value i. A counter in each group will then suffice to estimate $P(i)$ adaptively. A similar partitioning is done based on the object size s when $i = 1$. An important property in the approximation made in Eq. 5.5 is that the objects with the same i retain the LRU ordering. Thus, it is only needed to calculate P_r for the first object in each group, and to select the object with the lowest value for replacement.

Advantages: Includes access statistics for all objects. The replacement decisions are made in constant time.

Disadvantages: Needs additional data to be kept in memory. The cost model does not include access latencies for the objects.

In addition, several other policies have been proposed in the literature. For instance, in (Foong et al., 1999) logistic regression models were applied in a similar approach to LRV, where the value of the document is predicted by the model. The inputs of the model include the document size and type, number of requests and time since the last request. Other policies are discussed e.g. in (Bolot and Hoschka, 1996; Niclausse et al., 1998).

5.3 Proposed Approach

A novel approach to Web cache optimization is proposed next. Preliminary work on this approach can be found in (Koskela et al., 2001). In this approach, the value of each cache object is predicted with a model which is built by using real measurement data. Syntactic features are extracted from the HTTP responses of the server, from the Web cache logs, and from the structure of the HTML document. These features include e.g. URL (uniform resource locator), content type, size, modification and expiration dates, Cache-control HTTP headers, the number of hyperlinks and the number of images. All information available from each object is collected and transformed into numerical form. This numerical data is then used for building the model.

A neural network model is proposed, which can represent complex nonlinear dependencies (Bishop, 1995). The architecture of the model and its parameter values are estimated by using a learning algorithm. The inputs X_i of the model include numerical features extracted from each object. The output of the model Y is the prediction of the value of the object. During the training of the model, each object must be assigned a target value. Since training is carried out offline, the history of requests to the object is known. Thus the value can be calculated from the real access statistics. For instance, popularity of the object (the number of requests) during the chosen time period represents how valuable that object is for the cache.

The utility of this approach can be evaluated by using the model to predict the values of the objects that have not been used in the training of the model. If the mapping implemented by the model from the input features to the predictions generalizes to new objects, then the predictions gained from the model are useful. When the value of each object in the cache can be predicted to a certain accuracy, the caching policy can be optimized according to any objectives set.

Designing a caching policy which utilizes the predictions from the model involves choosing a cost model. The cost model essentially contains factors which are dependent on the environment, such as the cache storage size and network bandwidth. One possible approach is to utilize the prediction Y of the model as a substitute for the re-access probability P_r in Eq. 5.4:

$$V = \frac{C}{B} Y \tag{5.6}$$

The caching policy then replaces first the object with lowest relative value V, which is the predicted value Y weighted by the ratio C/B. In this case, choosing the values for C and B constitutes the design of the cost model for the cache.

B is the benefit of purging an object from the cache, and can be estimated to be proportional to the storage space freed in the cache. C is the cost of purging an object from the cache. However, estimating C is more complicated, since it should be easily calculated, and it should reflect the main objectives set for the caching. For instance, if the main objectives include minimising user access latency, C can be chosen to be proportional to the time needed to fetch the object. Another choice for C is proportional to the traffic involved in fetching the object, which was selected in (Rizzo and Vicisano, 2000).

The advantage in the proposed approach over previously presented ones is that no heuristic rules or selections based on empirical or simulated data are needed. Instead, by applying a suitable model and a learning algorithm, the model is fitted according to the data collected. Since only syntactic information is utilized, the extraction of features is simple.

The disadvantages in the proposed approach include difficulties in collecting a comprehensive data set. Predictions of the model can only be as good as the data available. Also the learning phase is computationally quite intensive. In practice, the model should be built in a separate computer with a large memory and a fast processor. The completed model can be implemented on-line in the Web cache, since evaluating the model output does not require much computational power.

5.4 Case Study

The objective of this case study was to build models which predict the value (popularity) of each cache object by using numerical input features collected from the objects, and to test whether this approach can improve the performance of the cache. The selected models were built and their parameters estimated from a training data set collected for the purpose. The prediction ability of the models was then estimated from a new data set, which was not used during the course of building the models. The performance gain from using the built models was then estimated by carrying out a simulation where a conventional cache and a cache which utilized the built model were compared.

To simplify the modeling task, it was first transformed into a classification problem. Each object was assigned either a high or a low value (class one or class zero) according to its popularity during a chosen time period. In addition, only HTML documents (objects with content type `text/html`) were considered. Linear and nonlinear classifiers were built by using the collected data. Inputs of the classifier included the collected features from each object, and its output gave the class of the object. Finally, the performance gain from applying the classifier was estimated. In order to study the effect of different workloads, synthetically generated cache request streams were employed in the simulations. In addition, the simulations were limited to comparing conventional LRU and GDS (Greedy Dual Size) caches with LRU and GDS caches which utilized the predicted class of the objects, respectively.

5.4.1 Data Collection

For this case study an access log containing one day requests was obtained from the FUNET (Finnish University and Research Network) Web cache. A Web robot was applied to retrieve each HTML document in the log and a HTML parser to parse its structure and store the features in a data file. Table 5.1 shows the selected features in more detail. Features 1–49 contain the number of certain HTML tag existing in the document. Features 50–56 include information extracted mostly from the Web cache access log. Features 51 ’Last modified’ and 52 ’Expires’ were assigned value 1 if the feature existed in the document headers and value 0 otherwise. Feature 55 ’Path depth’ states how many "/" characters the document URL contains. The collected data was cleaned from empty HTML documents and from features that had the same value throughout the data (usually either missing or zero). Also features which had extremely sparse data (only few differing values) were deemed useless and removed from the data.

Linear correlation analysis of collected features was also made. Correlation of each feature with the desired output of the classifier was calculated. It was found that most of the features, especially those collected from the HTML structure of the document, had only negligible linear correlation with the classification. These results suggest that the linear model may not be adequate to the task.

Feature analysis could also be utilized in selecting the features for the classifier. However, also cross-correlations of the features should be calculated before discarding features. For instance, some combinations of the simple HTML features can prove to be useful. Furthermore, linear correlation analysis is inadequate in this case, since also a nonlinear model was employed as a classifier. Thus, no features were removed from the data based on the correlation analysis.

Final data consisted of 48765 HTML documents with 56 features from each document. HTML documents with two or more requests in the access log were labeled to belong to the class 1 (high value), and all the others to the class 0 (low value). Labeling the objects with one request to the class zero can be justified by studies of Web server and proxy cache workloads, which suggest that “one-timers” should not be cached (see e.g. Busari and Williamson (2002)). Also, the

Table 5.1: Features collected from the HTML structure of the document and Web cache access log.

01 <a>	02 <address>	03 <area>	04 <b>	05 <base>	06 <big>	07 <blink>
08 <blockquote>	09 <body>	10 	11 <center>	12 <div>	13 <dl>	14 <dt>
15 <dd>	16 <em>	17 <font>	18 <frame>	19 <frameset>	20 <h1>	21 <h2>
22 <h3>	23 <h4>	24 <h5>	25 <h6>	26 <head>	27 <hr>	28 <html>
29 <i>	30 <iframe>	31 <img>	32 <li>	33 <link>	34 <map>	35 <meta>
36 <p>	37 <param>	38 <pre>	39 <script>	40 <select>	41 <small>	42 <strong>
43 <td>	44 <textarea>	45 <th>	46 <title>	47 <tr>	48 <u>	49 <ul>
50 Doc length	51 Last modified	52 Expires	53 Length of content	54 Content type	55 Path depth	56 Domain

selected class boundary was convenient in this case, since the number of samples in each class was approximately equal.

5.4.2 Feature Analysis with SOM

The Self Organizing Map was utilized as a method to evaluate visually the performance of the feature set. The SOM in feature analysis was briefly discussed in Section 2.4.3. In this procedure, the map is trained with the whole training data using the features selected. After the map is trained, the point density function, $p(w)$, of the weight vectors approximates the point density function, $p(X)$, of the underlying sampled distribution in the input space (Cottrell, 1997). This means that the SOM can visualize the separation of the classes by the features used.

The stochastic learning algorithm of the SOM was presented in Section 3.4.1. In practice, batch learning procedure is often employed instead. Here approximate gradient is evaluated for the entire input data and the weights are updated to the global optimum given the current partitioning of the data. When batch learning algorithm is applied, the map may reach a state where all units are exactly at the centroids of the samples in their regions of activity (Kohonen, 1997).

The feature analysis with the SOM was performed as follows. The extracted feature vectors, containing 56 values for each object, were first normalised so that each feature had a unit variance. The 2D SOM with 8x8 units was then trained by using the batch learning algorithm with all collected data. The SOM was trained

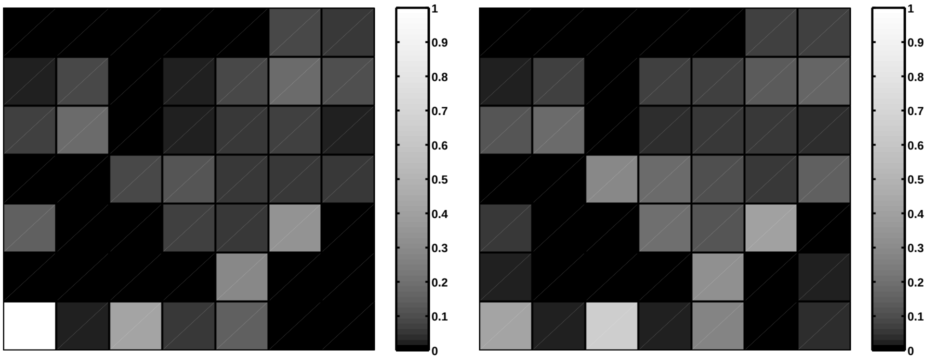

Figure 5.1: The class-conditional histograms of the features on a 8x8 SOM. Class 1 on left (value 1 represents 3903 objects), and class 0 on the right (value 1 represents 2886 objects).

with the SOM Toolbox [1] library implementation for Matlab (Alhoniemi et al., 2000). After this the number of objects belonging to each unit in the map were calculated for both classes. These histograms now approximate the conditional class distributions in the map space.

Figure 5.1 shows the result for both classes. Class 1 contained total 23919 objects, and class 0 24846 objects. The number of objects in each unit in the SOM lattice is shown with a color coding: white units include the most objects, and black units have none. From the figure it can be seen that the spread of the objects among the units is quite similar irrespective of the class, i.e. the classes are strongly overlapping. Also some differences can be found. Noticeably the lower left corner unit, which includes the most objects from class 1, does not have as large portion of objects from the class 0. However, from the visual inspection it can be speculated that a linear model may not be successful in the classification task.

5.4.3 Models

The models which were applied include linear and nonlinear classifiers. The generalized linear model was selected to act as the "baseline" classifier and the Multi-layer perceptron (MLP) network committee was applied as a nonlinear classifier.

[1] Available from http://www.cis.hut.fi/projects/somtoolbox/

Generalized Linear Model

In linear models, the output of the model is a linear function of the input variables. There are several ways in which such functions can be generalized. One method is using a monotonic, nonlinear activation function $g(a)$ which acts on the linear sum to give the form:

$$f(x, w) = g\left(w_0 + \sum_{i=1}^{d} w_i x_i\right), \tag{5.7}$$

One notable choice for the activation function is the logistic sigmoid function, which maps the interval $(-\infty, \infty)$ onto (0, 1):

$$g(a) = \frac{1}{1 - exp(a)} \tag{5.8}$$

Function $f(x, w)$ is called the discriminant function, since it separates the classes from each other. In statistics literature, the models with logistic sigmoid activation function are often referred as logistic discrimination. In case $|a|$ is small, $g(a)$ can be approximated by a linear function. In this sense the model contains the linear function as a special case. The logistic sigmoid is also important in case of two-class classification. Here, output of the discriminant function can be interpreted as the posterior probability of class membership $P(C_k|x)$, where C_k is the class, and x is the input vector (Bishop, 1995). These probabilities can be utilized in a subsequent decision-making stage to achieve a classification.

The logistic sigmoid was selected as the activation function for this case study. However, the probability interpretation of the class membership was not considered. The objects were classified directly to appropriate classes according to the value of the discriminant function. All simulations were carried out with Netlab [2] library implementation of the generalized linear models for Matlab (Nabney, 2001).

Multilayer Perceptron Network

The MLP was discussed previously in Section 3.3.1. In the case study, an early-stopping MLP committee model was applied. This model provides a good baseline nonlinear model, as demonstrated for instance in (Lampinen and Vehtari, 2001; Heikkonen et al., 1999).

Since the main objective of the case study was to test the proposed approach, and not to search for the best model, the model selection for the committee architecture was not carried out. Instead, the number of committee members and the number of hidden units of each MLP were chosen such that the model can

[2] Available from http://www.ncrg.aston.ac.uk/netlab/

Table 5.2: Confusion matrices for each cross-validation training case.

	1		2		3		4		5	
Linear	0.67	0.33	0.67	0.33	0.67	0.33	0.66	0.34	0.66	0.34
model	0.44	0.55	0.44	0.55	0.44	0.56	0.45	0.56	0.43	0.57
MLP	0.74	0.26	0.75	0.25	0.75	0.25	0.76	0.24	0.74	0.26
model	0.36	0.64	0.35	0.65	0.37	0.63	0.38	0.62	0.36	0.64

represent complex mappings, while keeping the computational complexity of the model reasonable. Therefore, the committee consisted of 10 MLP networks, each of which had one hidden layer with 20 units, and one output with one logistic output, yielding a model model $f(x, w)$, with weights w:

$$f(x, w) = g(w_0 + \sum_{j=1}^{m} w_j g\left(w_{j0} + \sum_{i=1}^{d} w_{ji} x_i\right)) \, , \tag{5.9}$$

where g is the logistic activation function and indices j and i correspond to output and hidden units, respectively.

The output of the committee model is calculated as the mean of all the committee member's outputs, and final classification result is gained by simple thresholding, i.e. if the output of the model is larger than 0.5, the sample is classified to class 1, and otherwise to class 0. The MLP networks in the committee were trained with Rprop algorithm (Riedmiller and Braun, 1993). For each MLP network a unique 10 % portion of the data was used for deciding when to stop training.

5.4.4 Classification Results

In order to be able to use all the data collected, 5-fold cross-validation was applied to estimate the classification error for each model. The data was first divided to 5 subsets of equal size. Each model was then trained 5 times, each time leaving out one of the subsets from training, and using the omitted subset to compute the classification error.

Table 5.2 shows the confusion matrices for each of the 5-fold cross-validation cases for both the linear and the MLP models. The upper left corner value of the confusion matrix represents the classification percentage of the class 0, and the lower right corner value represents the classification percentage for the class 1. In the case of the linear model the matrices are almost identical, supposedly because the model cannot learn any useful mapping from the training data. For the MLP model, there are some differences. However, differences are quite small regarding the amount of training samples available.

Table 5.3: Classification percentages of classifiers in the test data. Class 0 refers to a low value of the object (do not cache) and class 1 refers to a high value of the object (store in cache).

Method	Class 0	Class 1
Linear model	0.67	0.56
MLP model	0.74	0.64

Table 5.3 shows the classification results. The linear model classifies 67 percent of class 0 and 56 percent of class 1 correctly. The nonlinear MLP model classifies 74 percent of class 0 and 64 percent of class 1 correctly. In this case, the nonlinear model can distinguish objects that have a high value from those that have a low value much more accurately than the linear model. Thus the results with the MLP committee model turn out to be quite promising. However, more simulations with different data sets should be carried out to evaluate the usefulness of the MLP committee model in more detail. As is evident from the classification results with the linear model, there are no strong linear correlations between the features and the desired classification. Therefore, more advanced feature analysis taking cross-correlations and nonlinear dependencies into account should also be carried out.

5.4.5 Simulation with Analytical Workload

In order to evaluate the usefulness of the gained classifications also in terms of the usual performance metrics for Web caches, a simulation with a cache request stream was carried out.

There are two different ways of generating realistic cache workloads for research purposes (Barford and Crovella, 1998). In a trace-based approach, the measured Web cache log files are sampled in order to generate a new request stream. In an analytical approach, mathematical models for the workload characteristics of interest are utilized. For instance, the popularity and the size of the request stream objects are modeled with parametrised statistical distributions. A random sampling technique is then applied to produce a request stream that conforms to these models.

An analytical approach was chosen for the simulation, since it allows more flexibility in generating the request streams. ProWGen synthetic workload generation tool was applied for the purpose. Interested readers are directed to the Ref. (Busari and Williamson, 2002) for the details of the mathematical models involved. The parameters with ProWGen were chosen as follows: The total number of requests in the workload $n = 10000$, the percentage of distinct documents

Table 5.4: Parameters of the simulation with analytical workload.

Zipf slope α	Tail %	Size of the cache %	Classification accuracy %
0.6, 0.7, 0.8, 0.9, 1.0	10, 20, 30, 40, 50	1, 2, 5, 10	50, 60, 70, 80, 90, 100

of total number of requests $d = 30\%$, the percentage of 1-timers out of the distinct documents $o = 50\%$, the correlation between popularity and the file size $c = 0$, the size of the dynamic LRU stack model which introduces temporal locality $l = 1000$, the mean and the standard deviation of the lognormal values characterizing the body of the file size distribution $m = 7000$ and $s = 11000$, and the heavy tail index of the workload $h = 1.2$.

In order to study the performance of the cache with different classifier accuracies, as well as different request streams, simulations with different parameters were carried out. Table 5.4. shows these parameters and their values. Zipf slope α refers to the Zipf-like popularity distribution of the cache objects, given by equation

$$F_i = \Omega / i^{\alpha} , \tag{5.10}$$

where F_i is the popularity of the object ranked at position i and Ω is a normalising constant. Empirical studies with measured cache traces have shown that the popularity follows the Zipf-like distribution with different values of α (Breslau et al., 1999). The tail percentage gives the portion of objects, whose size is drawn from the tail of the size distribution, thus increasing the total volume of the request stream. The size of the cache was varied depending on the total volume of the request stream objects. The variation of the classification accuracy is described next in more detail.

The request stream produced by the ProWGen contains cache object identifiers and sizes. The class label was added for the objects as follows: For each object k in the request stream, the class equals one if the same object appears within the next L requests, otherwise zero. The length L was selected to be equal to the maximum number of cache objects $len = 500$, which was used in the simulation. After labeling the objects, the accuracy of the classification was degraded by labeling a predefined portion of the objects (selected by random sampling) to the wrong class.

Simulation with LRU Cache

A conventional LRU cache was first compared with a LRU cache, which utilizes the class information of the objects (referred as LRU-C). The LRU-C operates similarly as LRU, with the exception of handling the class zero objects. When an object which is not found in the cache is requested, the class of the object is

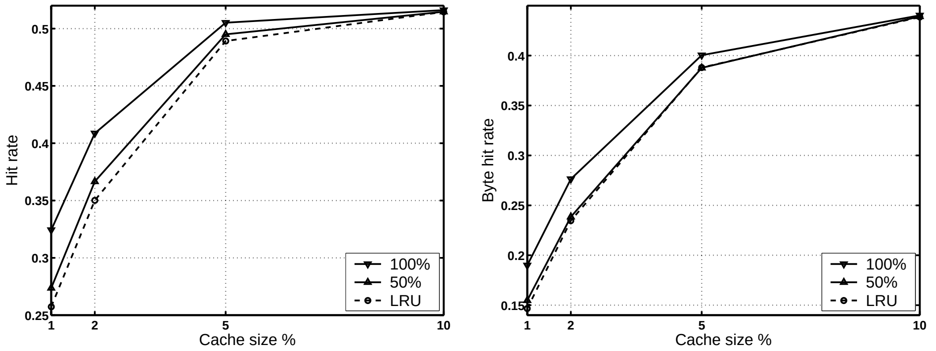

Figure 5.2: Hit rate (left) and byte hit rate (right) for LRU and LRU-C cache with 50% and 100% accuracy of the classifier, as a function of the cache size (1%, 2%, 5% and 10% of total size).

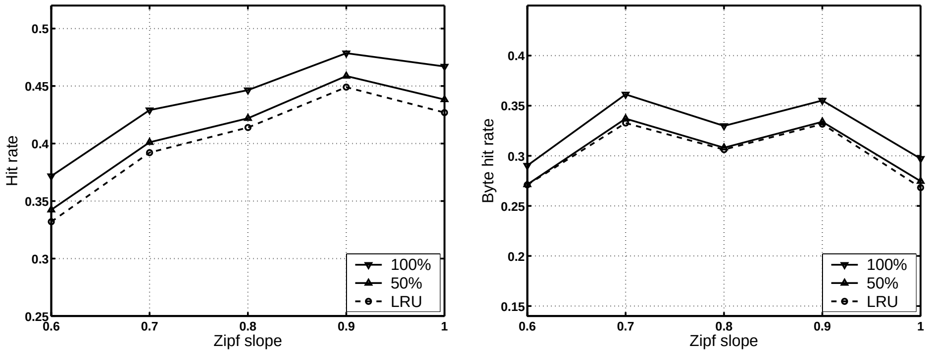

Figure 5.3: Hit rate (left) and byte hit rate (right) for LRU and LRU-C cache with 50% and 100% accuracy of the classifier, as a function of the Zipf slope $\alpha = (0.6...1.0)$.

checked. If the class is zero the object is placed into the middle of the LRU stack, otherwise on the top of the LRU stack. If the object was found in the cache, it is placed on the top of the stack, irrespective of the class. Removing of the cache objects is carried out the same way in both caches, the objects in the bottom of the stack are removed first when the cache is full.

The results of the simulations are shown in the Figures 5.2 and 5.3. As expected, the LRU-C outperformed the conventional LRU in all cases. Figure 5.2

shows the hit rate and the byte hit rate with different sizes of the cache. The plotted values were calculated as averages over the different tail percentages and values of α. Since the portion of one-timer objects was 50% of the distinct objects, on average the LRU-C gave slightly better hit rate than the LRU even with 50% accuracy in classification (i.e. random classifier). However, a substantial performance gain can be attained with more accurate classifications, especially with smaller cache sizes.

Figure 5.3 shows the hit rate and byte hit rate with different values of popularity distribution parameter α. The plotted values were calculated as averages over the different tail percentages and cache sizes. The hit rate grows with the value of α as expected. With larger values of α, there are fewer "hot" objects in the request stream, and the LRU stack can store most of them all the time. Since in the simulations there was no correlation between the object size and popularity, the size distribution affects the results with byte hit rate more than the value of α. As reported in (Busari and Williamson, 2002), byte hit rate is highly sensitive to individual large objects in the request stream, which can be seen in the Figure 5.3.

Simulation with GDS Cache

Another simulation was carried out with Greedy-Dual-Size (GDS) cache. The comparison was made with a conventional GDS cache and with a GDS cache, which utilizes the class information of the objects (referred as GDS-C). The GDS-C operates similarly as GDS, with the exception of handling the class one objects. When an object which is not found in the cache is requested, its value H is set to $cost/size$, where the cost is a constant value. In the GDS, cost was set to $cost = 1$ for all objects. In the GDS-C, the cost of the class one objects was set to $cost = 2$. If the object is found in the cache, its value is replaced with $cost/size$, where $cost = 2$, irrespective of the class. Removing of the cache objects is carried out the same way in both caches, the objects with the lowest value are removed first when the cache is full.

The results of the simulations with Greedy-Dual-Size are shown in the Figures 5.4 and 5.5. The GDS-C outperformed the conventional GDS in all cases. Figure 5.4 shows the hit rate and the byte hit rate with different sizes of the cache. Figure 5.5 shows the hit rate and byte hit rate with different values of popularity distribution parameter α.

5.5 Discussion

To summarize a novel approach to Web cache optimization has been presented. The results of a case study suggest that nonlinear classifier can find some regularities from the collected data, which can be utilized in predicting the value (class)

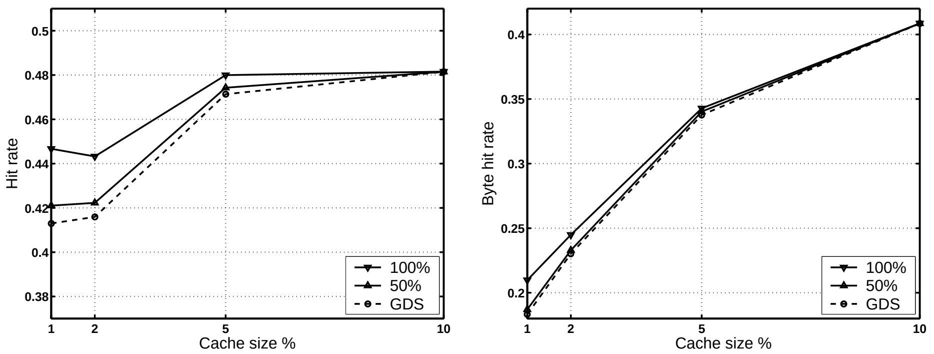

Figure 5.4: Hit rate (left) and byte hit rate (right) for GDS and GDS-C cache with 50% and 100% accuracy of the classifier, as a function of the cache size (1%, 2%, 5% and 10% of total size).

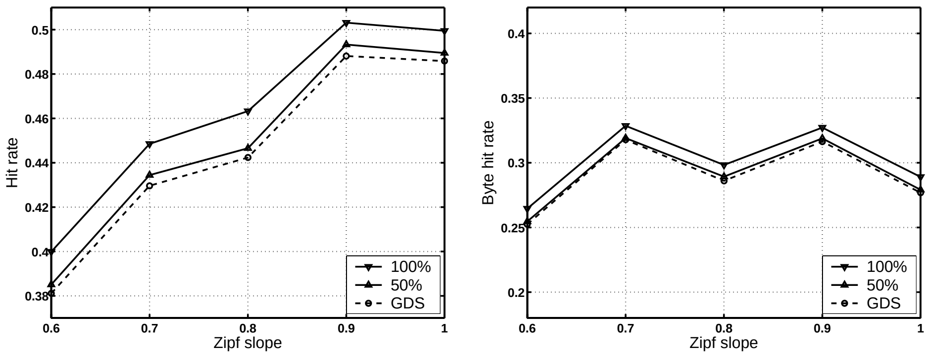

Figure 5.5: Hit rate (left) and byte hit rate (right) for GDS and GDS-C cache with 50% and 100% accuracy of the classifier, as a function of the Zipf slope $\alpha = (0.6...1.0)$.

of the object. This value can further be applied in optimizing the Web cache. The suggested method effectively replaces heuristic algorithms by a statistical model which is based on real measurement data. The preliminary results obtained from the case study turned out to be encouraging. However, further experiments should be carried out to fully evaluate the usefulness of the method. It is emphasized that building a good model requires a large amount of collected data. Also finding a set of suitable features, the model architecture and estimating the model parameters can be a computationally intensive task.

The cache operation can be optimized further if certain factors typical in Web caching are taken into account. One possible case is optimizing the use of cache storage. The distribution of requests for the cache objects is typically very heavily tailed (Bolot and Hoschka, 1996). The most popular objects get the most requests, while a large portion of objects which are stored in the cache are never requested again. Caching policy can take this into account, and use the predicted value for each object to decide which objects should not be stored in the cache at all. While the obtained classification percentage for class 1 was moderate 64% in this case study, class 0 was classified with a 74% accuracy. This classification with the associated cost model could be used for example to decide not to store big objects with a small value.

Another way how the cache operation can be further optimized is minimising the user access latency. As discussed in (Cohen and Kaplan, 2002), the access latency for an object which is validated (i.e. checked with the server whether it is still *fresh*) is often nearly as long as fetching the whole object. Validation request occurs when a cached object is requested from the cache, and the object has gone *stale*, i.e. it is uncertain whether it is still up to date. To ensure that the most popular objects in the cache are always fresh would help in minimising user access latencies. Since validation requests use computing and network resources, a policy for choosing the objects to be validated is needed. A straightforward policy would be using active refresh for the objects with the highest value.

The proposed approach may be difficult to implement in real Web caches. In order to be applicable, the prediction of the object value should be carried out within the caching policy algorithm. However, the initial parameters of the model can be estimated beforehand, which greatly reduces the computational burden. Since the statistics of requested objects change continuously, the model parameters should also be tuned frequently as a part of the caching algorithm.

Chapter 6

Conclusions

The vast majority of the neural network models proposed have been concerned with the learning of static mappings. However, time is important in many practical tasks that contain decision making and responses to temporal stimuli, such as pattern recognition, signal processing, and process control. A number of neural networks for temporal sequence processing have been proposed to overcome the limitations of static models. Most of the proposed models have been supervised, although unsupervised learning could be useful in many applications, such as in feature extraction or redundancy reduction of unlabeled temporal data.

One of the main themes in this work was presenting the Recurrent Self- Organizing Map (RSOM) as an unsupervised model for temporal sequence processing. The RSOM with local models was presented for the time series prediction task. The methodology for estimating the model and the procedures for the model selection were proposed. The results from the case studied showed that the context stored in the units of the RSOM allowed for useful representations of the delay coordinate embedded time series data. The performance of the "temporal quantization" was estimated based on the predictive ability of the constructed model in the one-step prediction tasks. In all studied cases, the local model approach gave better results when the RSOM was used as the quantizer instead of the conventional Self-Organizing Map.

The other main theme of this work was presenting the process of statistical data analysis with neural networks. The Web cache optimization case study served as an example of this process. After presenting the research problem, a novel approach based on modeling the value of cache objects based on syntactic features extracted from the objects was proposed. Issues with collecting and preprocessing the data, feature analysis, and model design were then addressed. The results gained from the case study suggest that the proposed approach is useful. More specifically, the collected data contains some dependencies that could be represented with neural network based classifier. Furthermore, a conventional linear model could not represent this underlying dependency. While the modeled process was time-dependent, the model was converted into processing only static data. This was mainly due to the collected data, which contained data from too short time period to facilitate modeling the temporal aspects. Finally, to analyse the benefits of using the proposed model in a Web cache, a simulation study with analytically generated sequence of objects was presented. The benefits of the proposed approach were confirmed.

To summarize the results of this thesis, neural network methods for data analysis and for modeling time varying processes have been addressed. The methodology of using the RSOM with local models in time series prediction, including modifications of the training algorithm to facilitate learning of temporal patterns is proposed. Another main contribution of this work is the proposal of a statistical model for optimizing the Web cache, including the process and methods involved.

References

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In Petrov, B. N. and Csaki, F., editors, *Second International Symposium on Information Theory*, pp. 267–281. Academiai Kiado. Reprinted in Kotz, S. and Johnson, N. L., editors, (1992). *Breakthroughs in Statistics Volume I: Foundations and Basic Theory*, pp. 610–624. Springer-Verlag.

Alhoniemi, E., Himberg, J., Parhankangas, J., and Vesanto, J. (2000). *SOM Toolbox for Matlab*. Laboratory of Information and Computer Science, Helsinki University of Technology. Available from http://www.cis.hut.fi/projects/somtoolbox/.

Amari, S., Murata, N., Müller, K.-R., Finke, M., and Yang, H. (1997). Asymptotic statistical theory of overtraining and cross-validation. *IEEE Trans. Neural Networks*, 8(5).

Back, A. and Tsoi, A. (1991). FIR and IIR synapses, a new neural network architecture for time series modelling. *Neural Computation*, 3(3):375–385.

Bak, P. (1996). *How Nature Works*. Springer Verlag, New York.

Barford, P. and Crovella, M. (1998). Generating representative Web workloads for network and server performance evaluation. In *Proc. SIGMETRICS '98, International Conference on Measurement and Modeling of Computer Systems*.

Bengio, Y., Simard, S., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*. John Wiley & Sons.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bolot, J.-C. and Hoschka, P. (1996). Performance engineering of the World-Wide Web: Application to dimensioning and cache design. In *Proc. WWW '96, 5th International World Wide Web Conference*, Paris, France.

Box, G., Jenkins, G., and Reinsel, G. (1994). *Time Series Analysis: Forecasting and Control*. Prentice Hall, Englewood Cliffs, New Jersey.

Brandt, S. (2002). *Theorems and Algorithms for Multiple View Geometry with Applications to Electron Tomography*. PhD thesis, Helsinki University of Technology.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Chapman and Hall.

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and Zipf-like distributions: Evidence and implications. In *Proc. IEEE INFOCOM '99 conference*.

Busari, M. and Williamson, C. (2002). ProWGen: A synthetic workload generation tool for simulation evaluation of web proxy caches. *Computer Networks*, 38(6):779–794.

Cao, P. and Irani, S. (1997). Cost-aware WWW proxy caching algorithms. In *Proc. USITS'97, Usenix Symposium on Internet Technologies and Systems*, Monterey, CA.

Carpinteiro, O. A. S. (1999). A hierarchical Self-Organizing Map model for sequence recognition. *Neural Processing Letters*, 9(3):209–220.

Chappelier, J., Gori, M., and Grumbach, A. (2001). Time in connectionist models. In Sun, R. and Giles, C. L., editors, *Sequence Learning: Paradigms, Algorithms, and Applications*, number 1828 in LNAI, pp. 105–134. Springer.

Chappell, G. and Taylor, J. (1993). The temporal Kohonen map. *Neural Networks*, 6:441–445.

Chen, R. and Tsay, R. S. (1993). Nonlinear additive ARX models. *Journal of the American Statistical Association*, 88(423):955–967.

Cohen, E. and Kaplan, H. (2002). Refreshment policies for Web content caches. *Computer Networks*, 38(6):795–808.

Conover, W. J. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, Inc.

Cottrell, M. (1997). Theoretical aspects of the SOM algorithm. In *Proc. WSOM'97, Workshop on Self-Organizing Maps*, pp. 246–267. Helsinki University of Technology.

Cottrell, M., Gaubert, P., Letremy, P., and Rousset, P. (1999). Analyzing and representing multidimensional quantitative and qualitative data : Demographic study of the Rhône valley. The domestic consumption of the Canadian families. In Oja, E. and Kaski, S., editors, *Kohonen Maps*, pp. 1–14. Elsevier.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. John Wiley & Sons, Inc.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, 1(2):131–156.

Davison, B. D. (1999). A survey of proxy cache evaluation techniques. In *Proc. 4th International Web Caching Workshop*.

de A. Barreto, G., Araujo, A., and Kremer, S. (2003). A taxonomy for spatiotemporal connectionist networks revisited: The unsupervised case. *Neural Computation*, 15:1255–1320.

de Vries, B. and Principe, J. (1992). The Gamma model – a new neural model for temporal processing. *Neural Networks*, 5:565–576.

deLeeuw, J. (1992). Introduction to Akaike (1973) information theory and an extension of the maximum likelihood principle. In Kotz, S. and Johnson, N. L., editors, *Breakthroughs in Statistics Volume I: Foundations and Basic Theory*, pp. 599–609. Springer-Verlag.

Douglis, F., Feldmann, A., Krishnamurthy, B., and Mogul, J. (1997). Rate of change and other metrics: A live study of the World-Wide Web. In *Proc. USITS'97, Usenix Symposium on Internet Technologies and Systems*.

Efron, B. and Tibshirani, R. (1997). Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560.

Efron, B. and Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–385.

Flexer, A. (2001). On the use of Self-Organizing Maps for clustering and visualization. *Intelligent Data Analysis*, 5:373–384.

Foong, A. P., Hu, Y.-H., and Heisey, D. M. (1999). Logistic regression in an adaptive web cache. *IEEE Internet Computing*, 3(5):27–36.

Frasconi, P., Gori, M., and Soda, G. (1992). Local feedback multilayered networks. *Neural Computation*, 4:120–130.

Gelman, A. and Meng, X.-L. (1996). *Markov Chain Monte Carlo in Practice*, chapter Model Checking and Model Improvement. Chapman & Hall.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58.

Gershenfeld, N. and Weigend, A. (1993). The future of time series: Learning and understanding. In Weigend, A. and Gershenfeld, N., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pp. 1–70. Addison-Wesley.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J., editors (1996). *Markov Chain Monte Carlo in Practice*. Chapman & Hall.

Hansen, M. H. and Yu, B. (2001). Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company, New York.

Heikkonen, J. and Koikkalainen, P. (1997). *Neural Systems for Robotics*, chapter Self-Organization and Autonomous Robots, pp. 297–331. Academic Press.

Heikkonen, J., Varjo, J., and Vehtari, A. (1999). Forest change detection via Landsat TM difference features. In *Proc. SCIA'99, 11th Scandinavian Conference on Image Analysis*, pp. 157–164.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to The Theory of Neural Computation*, volume 1. Addison-Wesley Publishing Company.

Hochreiter, S. and Schmidhuber, J. (1997). Long-short time memory. *Neural Computation*, 9:1735–1780.

Huber, P. (1981). *Robust Statistics*. Wiley.

Jordan, M. (1986). Attractor dynamics and parallelism in a conncetionist sequential machine. In *Proc. 8th Annual Conference of The Cognitive Science Society, Amherst*, pp. 531–546.

Jordan, M. and Jacobs, A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Kangas, J. (1994). *On the Analysis of Pattern Sequences by Self-Organizing Maps*. PhD thesis, Helsinki University of Technology.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795.

Kohonen, T. (1989). *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, Heidelberg.

Kohonen, T. (1997). *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 2nd edition.

Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., and Saarela, A. (2000). Self organization of a massive text document collection. *IEEE Trans. Neural Networks*, 11(3):574–585.

Koskela, T., Heikkonen, J., and Kaski, K. (2001). Using document features to optimize web cache. In *Proc. ICANN'01, 11th International Conference on Artificial Neural Networks*, pp. 1211–1216. Springer-Verlag.

Koskela, T., Heikkonen, J., and Kaski, K. (2003). Web cache optimization with nonlinear model using object features. *Computer Networks*, 43(6):805–817.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998a). Recurrent SOM with local linear models in time series prediction. In *Proc. ESANN'98, 6th European Symposium on Artificial Neural Networks*, pp. 167–172. D-facto.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998b). Time series prediction using RSOM with local linear models. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 2(1):60–68.

Kremer, S. (2001). Spatiotemporal connectionist models: A taxonomy and review. *Neural Computation*, 13:249–306.

Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pp. 950–957. Morgan Kauffmann Publishers.

Lampinen, J. (1997). Advances in neural network modeling. In *Proc. TOOLMET'97, Tool Environments and Development Methods for Intelligent Systems*, pp. 28–36, Oulu, Finland.

Lampinen, J. and Oja, E. (1989). Self-Organizing Maps for spatial and temporal AR models. In *Proc. SCIA'89, 6th Scandinavian Conference on Image Analysis*, Oulu, Finland.

Lampinen, J. and Oja, E. (1995). Distortion tolerant pattern recognition based on self-organizing feature extraction. *IEEE Trans. Neural Networks*, 6(3):539–547.

Lampinen, J. and Vehtari, A. (2001). Bayesian approach for neural networks – review and case studies. *Neural Networks*, 14(3):7–24. (Invited article).

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

Lapedes, A. and Farber, R. (1987). Nonlinear signal processing using neural networks: Prediction and signal modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM.

Lehtokangas, M. (1995). *Modeling with Layered Feedforward Neural Networks*. PhD thesis, Tampere University of Technology, Tampere, Finland.

MacKay, D. J. C. (1994). Bayesian non-linear modelling for the prediction competition. In *ASHRAE Transactions, V.100, Pt.2*, pp. 1053–1062. ASHRAE.

Mackey, M. and Glass, L. (1977). Oscillations and chaos in physiological control systems. *Science*, pp. 197–287.

Martinetz, T., Berkovich, S., and Schulten, K. (1993). 'Neural-Gas' network for vector quantization and its application to time-series prediction. *IEEE Trans. Neural Networks*, 4(4):558–569.

Moody, J. E. (1992). The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pp. 847–854. Morgan Kaufmann Publishers.

Mozer, M. (1993). Neural net architectures for temporal sequence processing. In Weigend, A. and Gershenfeld, N., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pp. 243–264. Addison-Wesley.

Nabney, I. T. (2001). *NETLAB: Algorithms for Pattern Recognition*. Springer. Available from http://www.ncrg.aston.ac.uk/netlab/.

Narendra, K. and Parthasarathy, K. (1990). Identification and control of dynamic system using neural networks. *IEEE Trans. Neural Networks*, 1:4–27.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer.

Niclausse, N., Liu, Z., and Nain, P. (1998). A new efficient caching policy for the World Wide Web. In *Proc. WISP'98, Workshop on Internet Server Performance*.

Orr, M. J. L. (1996). Introduction to radial basis function networks [online]. Technical report, Centre for Cognitive Science, University of Edinburgh.

Papoulis, A. (1984). *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2nd edition.

Picard, R. R. and Cook, R. D. (1984). Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583.

Pole, A., West, M., and Harrison, J. (1994). *Applied Bayesian Forecasting and Time Series Prediction*. Texts in Statistical Science. Chapman and Hall/CRC.

Proakis, J. and Manolakis, D. (1992). *Digital Signal Processing: Principles, Algorithms, and Applications*. MacMillan Publishing Company.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. ICNN'93, IEEE International Conference on Neural Networks*, pp. 586–591.

Rissanen, J. (1986). Stochastic complexity and modeling. *Annals of Statistics*, 14(3):1080–1100.

Rissanen, J. (1996). Fisher information and stochastic complexity. *IEEE Trans. Information Theory*, 42(1):40–47.

Ritter, H. and Schulten, K. (1988). Convergence properties of Kohonen's topology preserving maps: Fluctuations, stability and dimension selection. *Biological Cybernetics*, 60:59–71.

Rizzo, L. and Vicisano, L. (2000). Replacement policies for a proxy cache. *IEEE/ACM Trans. Networking*, 8(2):158–170.

Rousseeuw, P. and Leroy, A. (1987). *Robust Regression and Outlier Detection*. Wiley.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In *Parallel Distributd Processing*, volume 1, chapter 8, pp. 318–362. MIT Press.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech and Signal Processing*, 26(1):43–49.

Sarle, W. S. (1997a). How to measure importance of inputs? [online]. Technical report, SAS Institute Inc. Revised June 23, 2000. Available from ftp://ftp.sas.com/pub/neural/importance.html.

Sarle, W. S., editor (1997b). *Neural Network FAQ*. Periodic posting to the Usenet newsgroup comp.ai.neural-nets. Available from ftp://ftp.sas.com/pub/neural/FAQ.html.

Sauer, T. (1993). Time series prediction by using delay coordinate embedding. In Weigend, A. and Gershenfeld, N., editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pp. 175–193. Addison-Wesley.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464.

Singer, A., Wornell, G., and Oppenheim, A. (1992). A nonlinear signal modeling paradigm. In *Proc. ICASSP'92, IEEE International Conference on Acoustics, Speech and Signal Processing*.

Somervuo, P. (2000). *Self-Organizing Maps for Signal and Symbol Sequences*. PhD thesis, Helsinki University Of Technology.

Squid (2000). Squid Web proxy cache. Available from http://www.squid-cache.org/.

Stephens, M. (2000). Bayesian analysis of mixtures with an unknown number of components. *Annals of Statistics*, 28(1):40–74.

Sun, R. (2001). Introduction to sequence learning. In Sun, R. and Giles, C. L., editors, *Sequence Learning: Paradigms, Algorithms, and Applications*, number 1828 in LNAI, pp. 1–11. Springer.

Theodoridis, S. and Koutroumbas, K. (1999). *Pattern Recognition*. Academic Press.

Tong, H. (1990). *Nonlinear Time Series: A Dynamical System Approach*. Oxford University Press.

Tsoi, A. (1998a). Gradient based learning methods. In Tsoi, A., editor, *Adaptive Processing of Sequences and Data Structures*, number 1387 in LNAI, pp. 27–62. Springer.

Tsoi, A. (1998b). Recurrent neural network architectures: An overview. In Tsoi, A., editor, *Adaptive Processing of Sequences and Data Structures*, number 1387 in LNAI, pp. 1–27. Springer.

Tsoi, A. and Back, A. (1994). Locally recurrent globally feedforward networks: A critical review of architectures. *IEEE Trans. Neural Networks*, 5(2):229–239.

Varsta, M., Heikkonen, J., Lampinen, J., and del R. Millán, J. (2001). Temporal Kohonen map and recurrent Self-Organizing Map: Analytical and experimental comparision. *Neural Processing Letters*, 13(3):237–251.

Varsta, M., Heikkonen, J., and Millán, J. D. R. (1997a). Context learning with the Self-Organizing Map. In *Proc. WSOM'97, Workshop on Self-Organizing Maps*, pp. 197–202. Helsinki University of Technology.

Varsta, M., Millán, J. D. R., and Heikkonen, J. (1997b). A recurrent Self-Organizing Map for temporal sequence processing. In *Proc. ICANN'97, 7th Int. Conf. Artificial Neural Networks*, pp. 421–426. Springer-Verlag.

Vehtari, A. and Lampinen, J. (2001). Bayesian input variable selection using posterior probabilities and expected utilities. Technical Report B31, Helsinki University of Technology, Laboratory of Computational Engineering.

Vehtari, A. and Lampinen, J. (2002). Bayesian model assessment and comparison using cross-validation predictive densities. *Neural Computation*, 14(10):2439–2468.

Venables, W. N. and Ripley, B. D. (1999). *Modern Applied Statistics with S-PLUS*. Springer, third edition.

Vesanto, J. (1997). Using the SOM and local models in time-series prediction. In *Proc. WSOM'97, Workshop on Self-Organizing Maps*, pp. 209–214. Helsinki University of Technology.

Vesanto, J. (2002). *Data Exploration Process Based on the Self-Organizing Map*. PhD thesis, Helsinki University of Technology.

Vesanto, J. and Alhoniemi, E. (2000). Clustering of the Self-Organizing Map. *IEEE Trans. Neural Networks*, 11(3):586–600.

Voegtlin, T. (2002). Recursive Self-Organizing Maps. *Neural Networks*, 15(8–9):979–991.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1989). Phoneme recognition using time-delay networks. *IEEE Trans. Acoustics, Speech and Signal Processing*, 37(3):328–339.

Walter, J., Ritter, H., and Schulten, K. (1990). Non-linear prediction with Self-Organizing Maps. In *Proc. International Joint Conference on Neural Networks*, volume 1, pp. 589–594.

Wan, E. (1993). *Finite Impulse Response Neural Networks with Applications in Time Series Prediction*. PhD thesis, Stanford University.

Webb, A. (1999). *Statistical Pattern Recognition*. Oxford University Press.

Weigend, A. and Gershenfeld, N., editors (1993). *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

White, H. (1990). Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3(5):535–549.

Williams, R. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Williams, S., Abrams, M., Standridge, C. R., Abdulla, G., and Fox, E. A. (1996). Removal policies in network caches for World-Wide Web documents. In *Proc. ACM SIGCOMM '96, Applications, Technologies, Architectures, and Protocols for Computer Communication*, Stanford University, CA.

Wolpert, D. H. (1996). The existence of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1391–1420.

Yule, G. (1927). On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers. *Philos. Transactions of the Royal Society*, A226:267–298.

ISBN 951-22-6817-5 (printed)
ISBN 951-22-6818-3 (PDF)
ISSN 1455-0474