Helsinki University of Technology Laboratory of Computational Engineering Publications
Teknillisen korkeakoulun Laskennallisen tekniikan laboratorion julkaisuja
Espoo 2002 REPORT B32

SELF-ORGANIZING MAPS IN SEQUENCE PROCESSING

Markus Varsta

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 2002 REPORT B32

SELF-ORGANIZING MAPS IN SEQUENCE PROCESSING

Markus Varsta

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Electrical and Communications Engineering, Helsinki University of Technology, for public examination and debate in Auditorium S2 at Helsinki University of Technology (Espoo, Finland) on the 12th of December, 2002, 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 9400
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/2001/isbn9512262525/

E-mail: Markus.Varsta@hut.fi

ISBN 951-22-6251-7 (printed) ISBN 951-22-6252-5 (PDF)
ISSN 1455-0474
PicaSet Oy
Espoo 2002

Abstract

Models are abstractions of observed real world phenomena or processes. A good model captures the essential properties of the modeled phenomena. In the statistical learning paradigm the processes that generate observations are assumed unknown and too complex for analytical modeling, thus the models are trained from more general templates with measured observations. A substantial part of the processes we seek to model have temporal dependencies between observations thus defining templates that can account for these dependencies improves their ability to capture the properties of such processes.

In this work we discuss using the self organizing map with sequentially dependent data. Self-Organizing map (SOM) is perhaps the most popular non supervised neural network model that has found varied applications in the field of data mining for example. The original SOM paradigm, however, considers independent data, where context of a sample does not influence its interpretation. However, throwing away the temporal context of an observation when we know we are dealing with sequential data seems wasteful. Consequently methods for incorporating time into the SOM paradigm have been rather extensively studied. Such models if powerful enough would be very usable when tracking dynamic processes.

In this work a Self-Organizing map for temporal sequence processing dubbed Recurrent Self-Organizing Map (RSOM) was proposed and analyzed. The model has been used in time series prediction combined with local linear models. Deeper analysis provides insight into how much and what kind of contextual information the model is able to capture. The other topic covered by the publications in a sense considers an inverse problem. In this topic SOM was used to create sequential dependence and order into initially unordered data by modeling a surface and creating a path over the surface for a surface manipulating robot.

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 1997-2002. I want to express my sincere gratitude to Prof. Jouko Lampinen and Dr.Tech Jukka Heikkonen for excellent supervison and advice, and to Academy Prof. Kimmo Kaski whom with the above mentioned have expressed next to limitless patience with me. My gartitude also extends to Prof. Pasi Koikkalainen and Phd. José Del Ruiz Millán for their contributions in the work. The reviewers of my work, Docent Sami Kaski and Prof. Heikki Kälviäinen, deserve commendation for their efforts to prune out the potential stumbling blocks in the manuscript.

Sincere thanks are also due to my closest colleagues: Dr.Tech. Aki Vehtari, M.Sc. Timo Kostiainen, M.Sc. Timo Koskela, M.Sc. Arto Selonen, Dr.Tech. Ari Lukkarinen and M.Sc. Tommi Nykopp for making the laboratory a place where I always felt welcome. Their doors were always open for questions regardless of how trivial they might have seemed and the coffee breaks tended to last way beyond their best before dates. The last but not least I wish to thank Marja for doing her best to push me through the doldrums and M.Soc. Eeva Lampinen for epitomizing the good atmosphere in the laboratory.

My studies were partly funded by Graduate School in Electronics, Telecommunications and Automation (GETA) and 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).

Markus Varsta

List of Publications

Pub. 1 Varsta, M., and Koikkalainen, P. (1996) Surface Modeling and Robot Path Generation Using Self-Organization In *Proceedings of ICPR '96*, pages 30–34. IEEE.

Pub. 2 Koikkalainen, P., and Varsta, M. (1996) Robot Path generation for surface processing applications via neural networks In *Proceedings of the SPIE*, Vol. 2904, pages 66–73. SPIE.

Pub. 3 Varsta, M., Heikkonen, J., and Millán, J. del R. (1997) Epileptic Activity Detection in EEG with Neural Networks In *Proceedings of the 1997 International Conference on Engineering Applications of Neural Networks* , pages 179–186. The Royal Institute of Technology, Stockholm.

Pub. 4 Varsta, M., Millán, J. del R., and Heikkonen, J., (1997) A Recurrent Self Organizing Map for Temporal Sequence Processing In *ICANN'97: International Conference on Artificial Neural Networks* , LNCS vol. 1327, pages 421–426. Springer.

Pub. 5 Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998) Temporal Sequence Processing using Recurrent SOM In *KES '98: Proceedings of the Second International Conference on Knowledge Based Engineering Systems*, vol. 1, pages 290–296. IEEE.

Pub. 6 Varsta, M., Heikkonen, J., Lampinen, J., and Millán, J. del R. (2001) Temporal Kohonen Map and the Recurrent Self-Organizing Map: Analytical and Experimental Comparison *Neural Processing Letters*, Vol. 13, pages 237–251, Kluwer Academic Publishers.

Symbols and Abbreviations

ANN	artificial neural network
AR-SOM	Self-Organizing Map of competing Auto-Regressive models
bmu	best matching unit
DP	Dynamic Programming algorithm
DTW	Dynamic Time Warping algorithm
EEG	electroencephalography
EM	Expectation Maximization method
FMC	Folded Markov Chain
LLE	Locally Linear Embedding
GTM	Generative Topographic Mapping
HMM	Hidden Markov Model
LBG	Linde, Buzo, and Gray K-means vector quantizer
MLP	Multi Layer Perceptron network
pdf	probability density function
RBF	Radial Basis Function network
RSOM	Recurrent Self-Organizing Map
SOM	Self-Organizing Map
TKM	Temporal Kohonen Map
$\phi_i(t)$	response of a unit
$\sigma(\cdot)$	sigmoid type transformation function
$v_i(t)$	total activation of a unit
$E(\cdot)$	error or energy function
M	SOM in formulas
$\mathbf{x}$	input vector
$\mathbf{X}$	a data set drawn from $p_{\mathbf{x}}(\mathbf{x})$
Ω_X	cardinality of set X
$\mathbf{m}$	codebook or weight vector on map
$\mathcal{I}$	unit index space of map
γ	a learning rate factor
$b(\mathbf{x})$	function that returns the index of the *bmu* for $\mathbf{x}$
$h(i, j)$	the value of the neighborhood interaction function between i and j

$H(\mathbf{x}, i)$	shorthand for $h(b(\mathbf{x}), i)$
V_i	Voronoi cell of unit $i \in M$
$d(\cdot, \cdot)$	a distance measure
$d_M(\cdot, \cdot)$	a distance measure on latent map space of a SOM
$d_{\mathcal{X}}(\cdot, \cdot)$	a distance measure in (input space) $\mathcal{X}$
y	encoded value
$p_x(x)$	probability density of a random variable x
$p_{x,y}(x\|y)$	probability density of x conditioned on y
$p_{x,y}(x, y)$	joint probability density of x and y
D	expected distortion
$\mathbf{W}$	matrix of linear weights in LLE
$\delta(\cdot)$	Dirac delta
F_i	force acting on unit i
$P_X(X)$	likelihood of the set X
β	a noise variance parameter in the SOM *pdf*
Z	a density normalizing factor
$\boldsymbol{\mu}$	mean of component density in the SOM *pdf*
σ^2	variance
$\mathcal{N}(\mu, \sigma^2)$	normal distribution with expected value μ and variance σ^2
$\mathcal{X}$	input space
$\mathcal{M}_{\mathcal{X}}$	input manifold i.e region where $p_{\mathbf{x}}(\mathbf{x}) > 0$
Σ	covariance matrix
ψ_{Inv}	number of inversions on one dimensional map
$n_M(i, j)$	ordering of units in map space
$n_{\mathcal{X}}(i, j)$	ordering of units in input space
Q_1	quantity measuring organization mismatch in input space
Q_2	quantity measuring organization mismatch in latent map space
$\mathcal{P}$	topographic product
$\mu_{\mathcal{P}}$	logarithmic average of $\mathcal{P}$ over map
$\mathcal{D}_M$	Delaunay graph of map M
$\mathbf{M}_{\mathcal{D}_M}$	adjacency matrix to describe $\mathcal{D}_M$
$\Phi(\cdot)$	topographic function of SOM
S	set of sequences
$U(\cdot)$	activity function in TKM
λ	a time delay parameter in TKM
$\mathbf{y}$	leaked difference vector in RSOM
α	a time delay parameter in RSOM and AR_SOM
$\mathbf{P}$	warping path
$\boldsymbol{\pi}$	start state probability distribution

Contents

Chapter 1

Introduction

Models are abstractions of observed real world phenomena and processes. A good model captures the essential properties of the modeled phenomena. In the statistical learning paradigm the processes that generate observations are assumed unknown and too complex for analytical modeling, thus the models are trained from more general templates with measured observations. Artificial neural networks (ANNs) are a class of statistical models that owe their name to their biologically inspired origins. They are constructed of simple computational units, neurons, with weighted connections that store the essential modeling information.

Neural network models are roughly divided in two groups. The supervised models are trained for a specific transformation, where each input pattern is associated with an output pattern. The goal is not to memorize the samples but to find the underlying process that generated the samples, that is to find a model. Probably the most popular supervised neural network model is the multilayer perceptron network (MLP), where the interconnected computational units, perceptrons, are stacked in layers to form a feedforward network (Rumelhart et al., 1986; Haykin, 1998). Other supervised models include for example radial basis function networks (RBF) (Orr, 1996).

In contrast to supervised the unsupervised network models do not attempt to find a specific transformation from observed inputs to unknown outputs. Rather they look for similarities in the observations and try to bring orderliness to unordered data. Perhaps the premier unsupervised neural network model is the Self-Organizing Map (SOM) (Kohonen, 1982, 1984, 1997). The popularity of the SOM is probably a result of its computationally simple form and user friendliness: The results are easily visualized and interpreted though not always correctly. The primary uses for the SOM are found in various data analysis and mining tasks were the SOM is frequently used as a tool for visual inspection and data grouping. In these tasks SOMs ability to group similar data in clusters comes in handy. A good example of utilizing SOM in data mining are the WEBSOM and PicSOM

applications for content based organization of document (Kaski et al., 1998) or picture collections (Laaksonen et al., 2000). More recent relatives to SOM are for example generative topographic mapping (GTM) (Bishop et al., 1996) and locally linear embedding (LLE) (Roweis and Saul, 2000). These models will be discussed later in relation with the SOM.

In this thesis we will look into applying SOM to sequentially dependent data where the context of a sample carries meaningful information on its interpretation. Temporal independence is a frequently made assumption in statistical analysis but for real processes it often is very inaccurate thus losing significant knowledge carried by the observations. Bringing sequential or temporal dependence into neural network models has yielded several flavors of MLP where the models have recurrent feedback connections for context preservation (Jordan, 1986; Ellman, 1990; Werbos, 1990; Mozer, 1993). For unsupervised paradigm extending the SOM to sequentially dependent data is a natural direction. Several extensions to the SOM for sequence processing have been proposed where the SOM is either augmented with a time delay mechanism or the operators in the units of the map are changed to account for time (Chappell and Taylor, 1993; Kangas, 1994; Carpinteiro, 1999; Somervuo, 2000b). Extending SOM for sequentially dependent data is significant. If successful, such extension would broaden the scope of the model making it better suited to a variety of tasks such as speech recognition and other problems where the model needs to capture a dynamically evolving process.

In the second chapter the SOM algorithm is derived from several basis. First we discuss the original heuristic approach, where the derived model mimics the self organizing properties of biological nervous systems. That is units corresponding with nerons that respond to similar input or stimuli are attracted toward each other and form groups. After considering the heuristic origins we define an approximation for an error function the algorithm seeks to minimize. From this approximation the batch rule, which in general outperforms the original stochastic rule, can be derived for static input data. Extending the discrete function to a continuous case leads to the derivation of a SOM like algorithm that essentially has all the desirable properties of the SOM (Luttrell, 1989) avoiding much or the theoretical problems associated with it. The continuum limit extension of the SOM also allows the derivation of the algorithm from probabilistic basis (Luttrell, 1994).

After defining the SOM we review some essential analytical work on the SOM. Primarily we will concentrate on the organizational measures from both relevant directions. On one hand we will look into the error function of the SOM and what its minimization actually amounts to. We will also discuss a method that allows for systematic model selection with the SOM, that is choosing the best final neighborhood width and map topology for the data. The model selection method involves the derivation of an approximate probability density model for the SOM when the map is treated as a data generating mechanism. On the other

hand we will look into the quality of embedding the map into the input manifold, that is how well the map preserves the distance relations between points in the manifold. The error function based measures only indirectly address the issue of map organization as the minimization of the error function only endorses local order.

Finally we will look into extensions of the SOM, where the Euclidean distance operators in the units are replaced with some other operators to determine unit activations. In particular we will look into models proposed for sequence processing. These models take into account not only the current sample but also the samples near it thus accounting for the sequential dependencies of the sample when computing unit activations. The authors main contribution, Recurrent Self-Organizing Map RSOM (Pub. 4 also in Varsta et al. (1997)) is discussed in this chapter. In RSOM the units of a normal SOM are augmented with a integrating memory to preserve a trace of the past inputs for capturing temporal or sequential dependence. In addition to toy problem experiments we used the RSOM with moderate success in time series prediction with local linear models (Pub. 5), see also Koskela et al. (1998a,b)). In oder to establish what the RSOM actually amounted to we compared it analytically against the Temporal Kohonen Map (Chappell and Taylor, 1993). The analysis revealed that the model was limited to finding linear dependencies in the data and as such is not powerful enough for its intended purposes (Pub. 6, also covered in Varsta et al. (1998, 2000)).

After the literary review a short description for each of the bundled paper is given. In these descriptions the authors contribution and motivation behind the work is outlined. These papers cover two topics. The first topic is the use of SOM for generating program paths for a surface manipulating robots (Pub. 1,2). In this work the SOM is used to order the unorganized data, object surface points and then use the ordered map to generate a path, a sequence of points, for the robot to follow. The results were promising as feasible paths were generated but the results were never applied in a real setting. Publication three is a stand alone paper, where the SOM was used for robust feature quantization in the analysis of epileptiform electroencephalography (EEG) signal. The second topic covers the development and analysis of the RSOM model. The RSOM is a modification of the computational unit of the SOM that gives the model a limited capability to distinguish segments in temporal sequences.

Chapter 2

The Self-Organizing Map

The Self-Organizing Map (SOM) (Kohonen, 1982, 1984, 1997) has probably been the most popular unsupervised neural model since its introduction. The popularity of the SOM is largely a result of its algorithmic elegance and superficially easily interpretable results. Both properties are more or less an immediate consequence of the way the SOM was originally conceived. The model was originally defined through it desired behavior rather than result, which, as it turns out, makes the SOM somewhat difficult to analyze and deeply understand.

2.1 Fundamentals of the SOM

The Self-Organizing Map (Kohonen, 1982, 1984, 1997) mimicks the clustering behavior observed in biological neural networks by grouping units that respond to similar stimuli together. Nerve cells, neurons, in the cortex of the brain seem to be clustered by their function. For example brain cells responsible for vision form the visual cortex and those responsible for hearing form the auditory cortex.

Two variants of the SOM were developed more or less in parallel (Kohonen, 1982, 1984) (see Fig. 2.1). The model that was later dropped considered a string of units with lateral feedback connections that were both excitatory and inhibitory. Signal transfer in the network M was computed with

$$v_i(t) = \sigma\left[\phi_i(t) + \sum_{j \in M} \gamma(j) v_{i+j}(t-1)\right] \tag{2.1}$$

where $\phi_i(t)$ is units initial response to the input alone and $v_i(t)$ is the transformed activation of unit $i \in M$ at time t. The lateral interactions are captured by the summation and the strength of each connection is captured by the corresponding $\gamma(j)$. The output is then transformed with a non linear sigmoid type function $\sigma(\cdot)$. The initial response of a unit could, for example, be computed as an inner

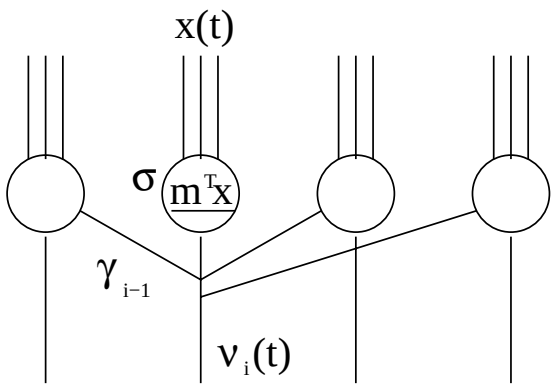

Figure 2.1: The original SOM consisting of laterally interacting units. Unit activation is computed with a sigmoid transformation of units immediate response and the current activations of its "neighbors".

product between its codebook vector and the input signal

$$\phi_i(t) = \mathbf{m}_i(t)^T \mathbf{x}(t),$$

where $\mathbf{m}_i(t)$ and $\mathbf{x}(t)$ are current codebook value and input respectively. The original construct for the SOM did not meet its goals. Learning was very slow as each input had to be presented multiple times at a time to train the map. Another problem was that the trained map could exhibit several regions of activity instead of just one. This is an immediate consequence of the Mexican hat (Fig. 2.2) function for lateral interactions which primarily influenced units nearby, thus far away regions of activity had little interaction.

At the end, however, the SOM that came to be was originally dubbed as a shortcut or engineering solution for the model described above. Not only does the shortcut scheme learn quicker but it also does away with multiple regions of activity by mandating a unique *bmu* for each possible input pattern. The SOM M is a triple $(\mathbf{M}, \mathcal{I}, h(\cdot,\cdot))$ consisting of the codebook $\mathbf{M}$, which usually but not necessarily is a matrix of codebook vectors, the index space $\mathcal{I}$ and the neighborhood interaction function $h(\cdot,\cdot)$ that captures the inter unit relatioships corresponding with $\gamma(j)$ in the alternate model.

The learning rule of the SOM consists of two distinct phases: The search for the *bmu* through competition, and the update of the codebook patterns of the *bmu* and its neighbors. In the basic SOM the activations of the units are inversely proportional to their Euclidean distances from the input pattern, hence the *bmu* can therefore be defined with

$$b(\mathbf{x}) = \arg\min_{i \in M} \|\mathbf{x} - \mathbf{m}_i\|, \tag{2.2}$$

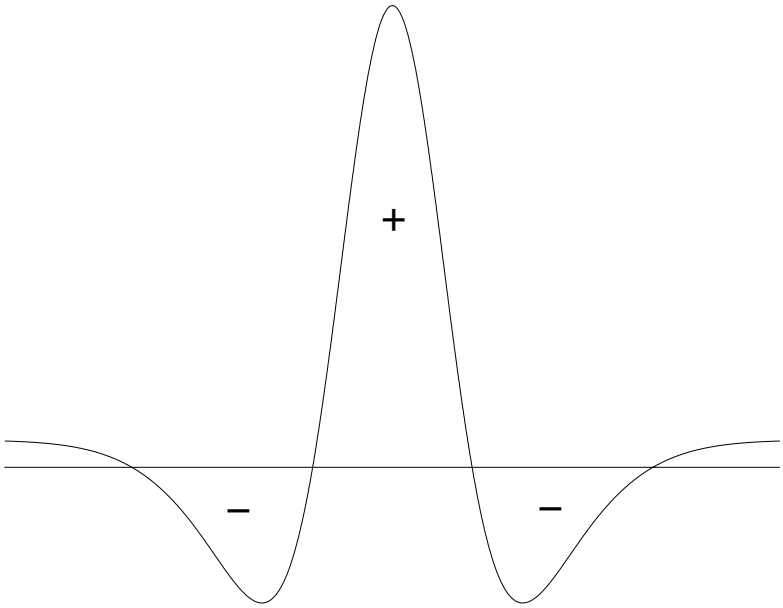

Figure 2.2: A Mexican hat function for lateral feedback strengths, computed as a difference of two zero meaned Gaussians of different variances.

where $b(\mathbf{x}) \in M$ is the index of the *bmu*, $\mathbf{m}_i$ is the codebook vector of unit $i \in M$ and $\mathbf{x}$ is the input pattern vector.

The update part of the rule moves the *bmu* and its neighbors toward $\mathbf{x}$ to slightly enforce maps response to the pattern. The update rule can be written as follows

$$\Delta\mathbf{m}_i = \gamma \cdot h(b(\mathbf{x}), i)(\mathbf{x} - \mathbf{m}_i), \tag{2.3}$$

where γ is a learning rate parameter and $h(b(\mathbf{x}), i)$ captures the neighborhood interaction between the *bmu* $b(\mathbf{x})$ and the unit i being updated. We can also write Eq. 2.3 as

$$\Delta\mathbf{m}_i = \gamma \cdot H(\mathbf{x}, i)(\mathbf{x} - \mathbf{m}_i)\ , \tag{2.4}$$

where $H(\mathbf{x}, i)$ is a shorthand notation for $h(b(\mathbf{x}), i)$. We will use these slightly differing notations for the neighborhood in parallel throughout this document.

Equations 2.2 and 2.4 define a Hebbian learning rule, where the strength of the training step is determined not only by the learning rate parameter $0 < \gamma \leq 1$ but also by the relationship of the updated unit j with the *bmu* $b(\mathbf{x})$ on the map. The inter unit relationships are captured by the neighborhood $h(i, j)$ which define how strongly units are attracted to each other.

In essence the learning rule of the SOM defines the model as a collection of competitive units that are related through the neighborhood function. In practice

the units are placed on a regular low dimensional grid and the neighborhood is defined as a monotonically decreasing function on the distance of the units on the map lattice, thus creating a latent space, which has the dimension of the map grid and flexibility determined by the neighborhood function. The latent space interpretation of the SOM is depicted in Fig. 2.3, where the two dimensional hexagonally arranged map grid, latent map space, is projected or embedded into three dimensional data sampled from a two dimensional manifold. The SOM can produce flawless, in the sense that the map follows the manifold, embedding when the dimension of the map grid matches the dimension of the input data manifold. We relied upon this property in the application of the SOM for surface manipulating robots "RoboSOM" (Pub. 1, Pub. 2), where the SOM was used to create an ordered model of two dimensional surface bent in third dimension in analogy to Fig. 2.3. Typical choices for the neighborhood function are a Gaussian

$$h(i,\, j) = \exp\left[\frac{d_M(i,\, j)^2}{2\sigma^2}\right], \tag{2.5}$$

or the so called bubble function

$$h(i,\, j) = \begin{cases} 1 & d_M(i,\, j) \leq c \\ 0 & d_M(i,\, j) > c \end{cases}, \tag{2.6}$$

where $d_M(i,\, j)$ is a distance measure in the map space, σ^2 is the variance of the Gaussian and in analogy c is a distance threshold parameter. The radius of the neighborhood is usually but not necessarily decreased during training. Likewise the learning rate parameter γ is normally decreased in accordance to predetermined cooling schedule with the aim of allowing the map sufficient time and freedom to organize before fine tuning the codebook. The basic SOM algorithm for Euclidean error and static dataset $\mathbf{X}$ is summarized in Alg. 1.

Algorithm 1: Basic SOM for Euclidean error.

1. Initialize units with random points from the input distribution represented by the data set $\mathbf{X}$.
2. Pick a sample $\mathbf{x}$ from $\mathbf{X}$ in random and choose the corresponding *bmu* $b(\mathbf{x})$ with Eq. 2.2.
3. Update the *bmu* and its neighbors toward $\mathbf{x}$ with Eq. 2.3.
4. Iterate steps 2 and 3 following a predetermined cooling schedule for both learning rate and neighborhood radius.

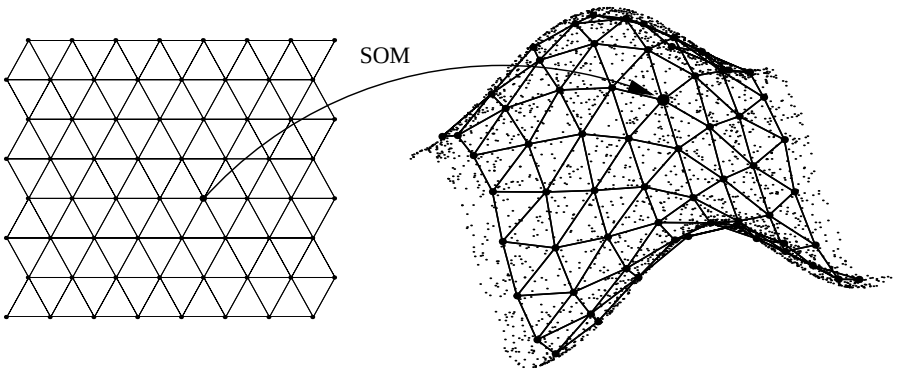

Figure 2.3: Latent space interpretation of the SOM. The low dimensional map grid, the latent map space, is projected into higher dimensional input space with the SOM.

2.1.1 Data Analysis With the SOM

High dimensional data is often difficult to comprehend and practically impossible to visualize as such. Large collections of data are often unordered and therefore difficult to browse for relevant pieces of information. They may also hide important clues as to how the data is generated. The goals of exploratory data analysis and data mining is to bring order into unordered data and to find previously unknown meaningful dependencies in the data. Perhaps the key application area for the SOM is found in this area, where the SOM on its own right does not do the trick but helps a human expert to sort it out.

The basic procedure typically involves training a two dimensional SOM with available data and using the result to plot various graphs, that are visually analyzed and interpreted by a human expert (see for example Cottrell et al. (1999)). The SOM takes care of the brute work by projecting the high dimensional data onto two dimensional map space. Depending on the quality of the projection and data, similar data with respect to the used error measure forms pronounced clusters on the map space. These clusters provide clues of the dependencies in the data and help a human expert to isolate data he/she finds particularly interesting. Typical depictions from the map are the so called component planes, where components from the codebook vectors are drawn in the shape of the map lattice. Comparing the component planes of two or more codebook variables may provide some insight into the dependencies between the variables. However, it is up to the human expert to isolate false hits from the real thing. Furthermore finding higher than first order dependencies with this method is difficult. Another frequently used

graph considers the distances between immediate neighbors on the map in the input space. This graph gives clues to the quality of clustering on the map. Units belonging to the same cluster are close in the input space indicated by small values on the graph, conversely on a well organized map the cluster boundaries should be pronounced. These two visualizations are just an example how the SOM can be used to visualize high dimensional data.

An example of succesfull applications of the SOM is the WEBSOM (Kaski et al., 1998) for content based organization of document collections and document retrieval. The WEBSOM architecture consists of two hierarchically connected SOMs. The word category map clusters words found in the documents using not only the word itself but also its immediate neighbors. The use of context brings order into the mapping as synonyms often appear in similar context. The documents are projected on this word category map as collections of word triplets and a histogram of hits on the map units is formed. This histogram conveys information on the realtive frequencies of different words categories in the document and serves as the input for the second or the document map to cluster documents. The underlying intuition is that the word category frequencies are typical to documents covering the same topic. A good, though slightly outdated, overview of the versitality of the applications utilizing SOM can be found in Kohonen (1997).

2.2 Derivation of the SOM from an Error Function

In the previous section we discussed the derivation of the SOM primarily from the organization point of view. In this section the algorithm is derived again from an error function that is approximately[1] consistent with the update rule summarized in Alg. 1.

SOM is a set of competitive units connected into a lattice with a topological neighborhood function. It can be related with Linde, Buzo, and Gray vector quantizer (Linde et al., 1980) but in the SOM the units are arranged into a low dimensional grid and locally attracted toward each other with a strength determined by the neighborhood function. Later we will refer to the grid or the lattice of connected units as the map space. When properly trained the SOM forms a mapping of the input manifold, where the units close in the map space are close in the input space. However, units close in the input space are not necessarily close in the map space since the dimensionality of the input manifold is often higher than the dimensionality of the map which consequently folds. The map space is usually one or two dimensional to facilitate visualization, for example.

The SOM, like vector quantizers in general, partitions the input space into convex regions of activity that are characterized with the following property: Ev-

[1]The error function is consistent with the original update rule with exception of the points that lie exactly at the boundarires of Voronoi cells.

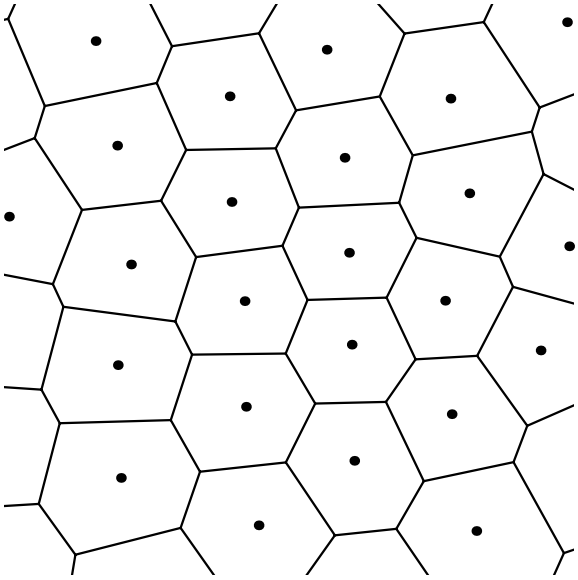

Figure 2.4: Voronoi partitioning by a two dimensional SOM. Codebook vectors are drawn with small circles and the boundaries of the Voronoi cells with lines.

ery point in the space is closer to the centroid of its region than to the centroid of any other region (see Fig. 2.4). The centroids of the regions are defined by the codebook of the map. Partitioning of this kind is called Voronoi tessellation of the input space. The partitioning of an optimally trained map minimizes some error function but due to properties of the SOM this error function cannot be an energy function (Erwin et al., 1992a), in other words the training rule of the SOM cannot be derived as a gradient of any continuous function.

In its converged state the basic SOM locally minimizes the neighborhood weighted sum of squared distances

$$E(\mathbf{X}, M) = \sum_{\mathbf{x}\in\mathbf{X}} \sum_{j\in M} H(\mathbf{x}, j)\|\mathbf{x} - \mathbf{m}_j\|^2 \tag{2.7}$$

which is only defined for a discrete set of input patterns $\mathbf{X}$ such that no $\mathbf{x} \in \mathbf{X}$ coincides with any Voronoi cell boundary. Differentiating Eq. 2.7 with respect to codebook vectors yields

$$\frac{\partial E(X, M)}{\partial \mathbf{m}_j} = 2 \sum_{\mathbf{x}\in X} H(\mathbf{x}, j)(\mathbf{x} - \mathbf{m}_j) \ , \tag{2.8}$$

when the discontinuities of $E(\mathbf{X}, M)$ are ignored due to assumed absence of samples on the Voronoi boundaries. It is relatively easy to derive a learning rule for

the SOM from Eq. 2.8. The basic Hebbian SOM rule Eqs. 2.2 and 2.3 approximates stochastic gradient descent to minimize Eq. 2.7. This observation provides guidelines to treatment of the learning rate $\gamma(t)$, which has to satisfy theRobbins and Monro (1951) conditions

$$\begin{cases} \int_0^\infty \gamma(t)dt & = & \infty \\ \lim_{t\to\infty} \gamma(t) & = & 0 \end{cases} \tag{2.9}$$

The conditions in Eq 2.9 define how the learning rate should behave to guarantee convergence at a local minimum at the limit. The first condition guarantees that the map has enough freedom to converge from any initial configuration and the second one guarantees that once the map has reached a locally near optimal state it will no longer unlearn it.

The other learning rule dubbed the *batch rule* is a fixed point optimization algorithm where the competition and update stages are separated and performed for the entire training data in turns. In the competition step each sample $\mathbf{x} \in \mathbf{X}$ is assigned the corresponding *bmu* with Eq 2.2. In the optimization step the codebook vectors are updated to global optimum with

$$\mathbf{m}_j = \frac{\sum_{\mathbf{x}\in\mathbf{X}} H(\mathbf{x},\, j)\mathbf{x}}{\sum_{\mathbf{x}\in\mathbf{X}} H(\mathbf{x},\, j)} \tag{2.10}$$

assuming that the movement of the boundaries of the Voronoi cells is little enough not to reassign any sample $\mathbf{x} \in \mathbf{X}$ to a different unit. This rule is obtained by setting

$$\frac{\partial E(\mathbf{X},\, M)}{\partial \mathbf{m}_j} = 0$$

and solving for $\mathbf{m}_j$. In practice with an unorganized map the unit locations are typically shaken up a lot, thus a considerable number of samples change their respective *bmu*:s and the two step procedure has to be iterated several times for the newly configured SOM codebook.

Both of these learning rules have their merits. With a static training set and a stationary problem the batch rule is almost invariably better. The progress in learning is easier to monitor from the number of reassigned samples and the rule in general is computationally cheaper. With the batch rule the map can easily be trained to convergence with several neighborhood widths to facilitate model selection for example. Since the training sequence does not influence the outcome of the batch rule its result is always the same for a given data set and initial state. In this sense the batch rule is deterministic.

The stochastic rule with a static input set can be used to approximate the batch rule by starting with large neighborhood and a large learning rate. The samples are presented to the map in a random sequence to avoid creating unwanted dependencies into the data. At the end the neighborhood is typically reduced to contain

only the *bmu* for fine tuning. This implicitly assumes that the sample is in fact the population and the map does not need any predictive power to generalize to unknown future samples.

While there are no clear benefits in using stochastic rule with static input set it may be modified to follow evolving processes. If the learning rate is not constrained to follow the Robbins-Monro conditions in Eq. 2.9 but is along with the neighborhood radius dynamically adjusted with the incoming data the SOM can respond to changes in the properties of the source of data. If the statistical properties of the process change the parameters are updated accordingly to give the map the ability to adapt to the changed situation. This, however, is not a simple task since distinguishing out liars and other possible anomalies from true changes in the underlying process is not trivial, and thus it is not straightforward to sensibly update the learning rate and neighborhood radius. In practice even with non stationary processes the batch rule can often be used to better effect when the map is retrained with data that is representative of the current properties of the underlying process. This would be especially effective with periodically rather than continuously changing processes.

The above derivation and discussion considers the update part of the learning rule and the competition part still remains to be discussed. As mentioned earlier the batch rule is a fixed point optimization algorithm consisting of separate competition and the update steps. The competition step of the rule assigns a sharp *bmu* with Eq. 2.2. In the update step, however, all units in the neighborhood of the *bmu* are updated toward the sample which partially contradicts the competition step. This observation will be discussed in the next section where an alternative SOM algorithm that does not exhibit this property is derived from communication channel interpretation of the SOM.

2.3 Communication Channel Interpretation of the SOM

In the previous section we derived the SOM as constrained vector quantizer related with the standard LBG K-means quantizer. In this section we describe an alternate way of treating the map as a noisy communication channel (Luttrell, 1989), which distorts data transmitted through it. Such a channel is formally depicted in Fig. 2.5. The expected distortion D for a random variable $\mathbf{x}$ with density $p_{\mathbf{x}}(\mathbf{x})$ transmitted through the channel is

$$D = \int p_{\mathbf{x}}(\mathbf{x}) d\left(\mathbf{x}, \mathbf{g}\left(f(\mathbf{x})\right)\right) d\mathbf{x} \tag{2.11}$$

where $f(\cdot)$ is the encoder and $g(\cdot)$ is the decoder. It is noted that optimal encoder decoder pair minimizes D. When Euclidean distance is used for $d(\cdot,\cdot)$ optimization of Eq. 2.11 leads to normal vector quantification with sharp *bmu* selection i.e nearest neighbor encoding.

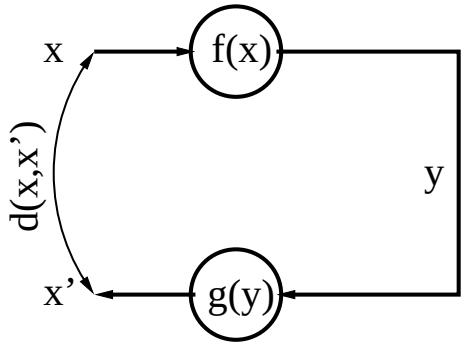

Figure 2.5: Communication channel interpretation of a vector quantizer consisting of a an encoder $y = f(\mathbf{x})$ and decoder $\mathbf{x}' = \mathbf{g}(y)$. The loss of data in the channel is captured by the distortion measure $d(\mathbf{x}, \mathbf{x}')$, which for the normal L_2 quantizer is the Euclidean distance between the original data and its reconstruction after transmission through the channel.

Extending the channel formalism to the SOM requires a second encoding stage Fig. 2.6 to account for the neighborhood. With the second stage encoder expression for D becomes

$$D = \int p_{\mathbf{x}}(\mathbf{x}) d\left(\mathbf{x}, \mathbf{g}\left(\kappa\left(f(\mathbf{x})\right)\right)\right) d\mathbf{x}. \tag{2.12}$$

The neighborhood of the SOM can be considered an uncertainty in the encoding process or noise acting in the latent map space. Consequently the second encoding stage $\kappa(\cdot)$ accounting for the neighborhood can be replaced by an additive noise component on the first stage encoder result to get

$$D = \int d(\mathbf{x}, \mathbf{g}(f(\mathbf{x}) + h)) p_h(h)\, dh\; p_{\mathbf{x}}(\mathbf{x})\, d\mathbf{x} \tag{2.13}$$

where h is the additive noise component. After choosing Euclidean distance for the distortion measure d the functional derivatives of Eq.2.13 with respect to the encoder $f(\mathbf{x})$ and the decoder $\mathbf{g}(f(\mathbf{x}) + h)$ are as follows

$$\frac{\delta D}{\delta y(\mathbf{x})} = p_{\mathbf{x}}(\mathbf{x}) \int \frac{\partial (\mathbf{x} - \mathbf{g}(y))^2}{\partial y}\, p_h(h)\, dh \tag{2.14}$$

$$\frac{\delta D}{\delta \mathbf{g}(y)} = 2 \int p_{\mathbf{x}}(\mathbf{x})\; p_h(y - y(\mathbf{x}))(\mathbf{x} - \mathbf{g}(y))\, d\mathbf{x}, \tag{2.15}$$

where $y = y(\mathbf{x}) + h$. The functional derivative in Eq. 2.14 captures the dependence of the distortion on the encoder while the derivative in Eq.2.15 captures the dependence on the decoder. The optimal encoder now assigns $\mathbf{x}$ with the encoded

value

$$y = \arg\min_{y} \int (\mathbf{x} - \mathbf{g}(y(\mathbf{x}) + h))^2 \, p_h(h) \, dh \tag{2.16}$$

by integrating away the uncertainty from the encoding noise. The optimal decoding or reconstruction is obtained by setting Eq. 2.15 zero and solving for $\mathbf{g}(y)$

$$\mathbf{g}(y) = \frac{\int p_{\mathbf{x}}(\mathbf{x}) p_h(y - y(\mathbf{x})) \, \mathbf{x} \, d\mathbf{x}}{\int p_{\mathbf{x}}(\mathbf{x}) p_h(y - y(\mathbf{x})) \, d\mathbf{x}}. \tag{2.17}$$

In other words the optimal reconstruction integrates over the input space and weights each possible input with its corresponding encoding probability

$$p_h(y - y(\mathbf{x})).$$

Equations 2.16 and 2.17 define a learning algorithm for a continuum limit two stage encoder. The analog result for a discrete two stage encoder leads to a SOM like structure which differs from the original SOM in its competition part. For the SOM, where the noise process in the latent space is modeled by the neighborhood function, the competition part becomes

$$b(\mathbf{x}) = \arg\min_{i \in M} \sum_{j \in M} h(i, j)(\mathbf{x} - \mathbf{m}_j)^2. \tag{2.18}$$

In other words the distortion is minimized over the neighborhood of the *bmu* instead of the *bmu* alone. The update rule is identical to the original SOM rule. The competition rule in Eq. 2.18, dubbed minimum distortion rule, is consistent with the normal SOM update rule, thus resulting in gradient descent rule for the SOM. A map trained with the minimum distortion rule no longer Voronoi partions its input space because the distrotion or error is minimized with respect to a set of neighborhoods instead of discrete set of points. The significance of this result will be further elaborated in section 3.9, where we will discuss an approximation of the generative probability density model underlying the SOM. The original SOM rule will not generate smooth densities unlike a SOM trained with the minimum distortion rule. It is, however, worth noting that in practise both rules yield rather similar results for at least two reasons. Once the map is well organized the competition rules are more likely to choose the same *bmu* and on the other hand, the normal reduction of the neighborhood radius reduces the contribution of the neighborhood in the error.

This work also motivates an alternative way of training the map. Normal practice is to start with wide neighborhood to quickly order the map (Erwin et al., 1992b), and then reduce the neighborhood progressively to achieve the wanted compromise between smoothness and fit into the data. This approach corresponds with starting with a broad density for the latent space noise $p_h(h)$ and the variance of $p_h(h)$ is reduced as the learning progresses. Similar effect can be achieved

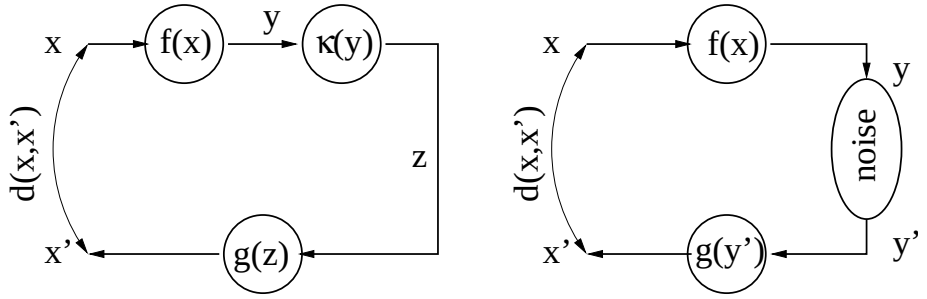

Figure 2.6: A communication channel with a two stage encoder. A SOM like construct results when the second stage encoder is considered a noise process acting on the result of the first stage encoder. This noise process corresponds to the neighborhood function in the case of SOM, where the winner take all mapping is perturbed in the update with the neighborhood function.

through a renormalized approach, where the neighborhood is kept constant and the latent space is dilated as the learning progresses. The renormalized scheme leads to much faster convergence through reduced computational load in comparision with the original scheme. In the discrete case the dilation of the latent space corresponds with interpolating new units into the map preferably after the current map has converged.

2.4 Probabilistic Approach to SOM

The communication channel formulation of the SOM can be set in more general probabilistic framework, where the encoding and decoding functions are replaced with conditional densities (Luttrell, 1994). Expected value of the channel distortion Eq. 2.11 is thus reformulated as follows

$$D = \int p_{\mathbf{x}',y}(\mathbf{x}'|y)\, p_{y,\mathbf{x}}(y|\mathbf{x})\, p_{\mathbf{x}}(\mathbf{x}) d(\mathbf{x},\mathbf{x}')\; d\mathbf{x}'\; dy\; d\mathbf{x} \tag{2.19}$$

where the conditional density $p_{y,\mathbf{x}}(y|\mathbf{x})$ encodes input $\mathbf{x}$ with y and $p_{\mathbf{x}',y}(\mathbf{x}'|y)$ reconstructs $\mathbf{x}$ as $\mathbf{x}'$ from its encoded counterpart. The integration over y and $\mathbf{x}'$ compute the average reconstruction error $d(\mathbf{x}, \mathbf{x}'(y(\mathbf{x})))$ weighted with the encoding density $p_{y,\mathbf{x}}(y|\mathbf{x})$ for $\mathbf{x}$. The remaining integration over $\mathbf{x}$ for the expected overall channel distortion weights all possible inputs with corresponding density. Likewise the expectation for the channel distortion for the model with the added noise in the encoding stage corresponding with SOM in Eq. 2.13 is reformulated as

$$D = \int p_{\mathbf{x}',y}(\mathbf{x}'|y+h)\, p_{y,\mathbf{x}}(y|\mathbf{x})\, p_{\mathbf{x}}(\mathbf{x})\, p_h(h) d(\mathbf{x},\mathbf{x}') dy\; d\mathbf{x}\; dh\;, \tag{2.20}$$

where h is again the noise process acting on the encoding space corresponding with the neighborhood. The interpretation of Eq. 2.20 is similar to that of Eq. 2.19 with the added integration over the encoding noise with density $p_h(h)$.

Probabilistic re-formulation carries several advantages (Luttrell, 1994).

- "Soft" probabilistic formulation allows easier theoretical manipulations of the expressions than "hard" deterministic functions.
- The probabilistic approach lends itself well to a simulated annealing approach in numerical simulations.
- Contact with standard results can be made when Euclidean distance is chosen as distortion measure $d(\mathbf{x}, \mathbf{x}')$.
- If a single pair of transformations was considered but one was uncertain about which particular pair, then a probabilistic approach assigning probability through density to each possible pair would be necessary.

Probabilistic derivation of the continuum limit SOM is based on construct called folded Markov chain, which we will discuss below.

2.4.1 Folded Markov Chain

Like the communications channel approach the probabilistic approach is considered at the continuum limit through a construct called Folded Markov Chain (FMC) depicted in Fig. 2.7, which performs an N-stage transformation of an input vector $\mathbf{x}_0$ to an output vector $\mathbf{x}_N$ and then inverts the transformation (Luttrell, 1994). The Bayes rule of probabilities is used to link the probabilistic transformations as follows

$$p_{i,i+1}(\mathbf{x}_i|\mathbf{x}_{i+1})\,p_{i+1}(\mathbf{x}_{i+1}) = p_{i+1,i}(\mathbf{x}_{i+1}|\mathbf{x}_i)\,p_i(\mathbf{x}_i). \tag{2.21}$$

In other words the transformations of the "up" pass are inverted on the "down" pass. Relating the transformations with the Bayes rule and fixing Dirac delta

$$\delta(\mathbf{x}_N - \mathbf{x}'_N)$$

as the last transformation at the end of chain guarantees that the marginal densities $p_i(\mathbf{x}_i)$ and $p_i(\mathbf{x}'_i)$ are identical.

2.4.2 Single Stage Folded Markov Chain as a Vector Quantizer

Vector quantizer can be derived as a special case of single stage FMC in Fig. 2.8 (Luttrell, 1994). The derivation starts at the definition of the average Euclidean

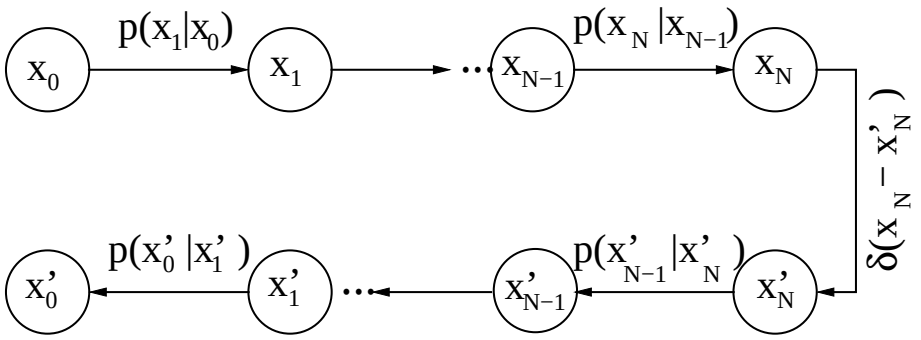

Figure 2.7: A folded Markov chain with both passes. The top half represents the forward pass to encode and the bottom half represents the backward pass to decode. The probabilistic transformations on the "up" direction are related with those of the "down" direction with the Bayes rule (see Eq. 2.21). Dirac delta at the end guarantees that the decoding starts where the encoding ended i.e $\mathbf{x}'_N = \mathbf{x}_N$.

error D between the input $\mathbf{x}_0$ and its reconstruction $\mathbf{x}'_0$, in a one stage FMC

$$D = \int p_0(\mathbf{x}_0)\, p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\delta(\mathbf{x}_1 - \mathbf{x}'_1)\, p_{0,1}(\mathbf{x}'_0|\mathbf{x}'_1)\|\mathbf{x}_0 - \mathbf{x}'_0\|^2\, d\mathbf{x}_0\, d\mathbf{x}_1\, d\mathbf{x}'_1\, d\mathbf{x}'_0. \tag{2.22}$$

The delta operator is easy to integrate out and with the Bayes rule

$$p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\, p_1(\mathbf{x}_1) = p_{0,1}(\mathbf{x}_1|\mathbf{x}_0)\, p_0(\mathbf{x}_0),$$

we get for the distortion measure

$$D = \int p_1(\mathbf{x}_1)\, p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\, p_{0,1}(\mathbf{x}'_0|\mathbf{x}_1)\|\mathbf{x}_0 - \mathbf{x}'_0\|^2\, d\mathbf{x}_0\, d\mathbf{x}'_0\, d\mathbf{x}_1. \tag{2.23}$$

After expanding the squared norm as

$$\|\mathbf{x}_0\|^2 - 2\mathbf{x}_0\mathbf{x}'_0 + \|\mathbf{x}'_0\|^2$$

and performing the integrations where possible this can be further simplified to

$$D = 2\int p_1(\mathbf{x}_1)\left[\int p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\|\mathbf{x}_0\|^2\, d\mathbf{x}_0 - \|\int p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\mathbf{x}_0\, d\mathbf{x}_0\|^2\right] d\mathbf{x}_1, \tag{2.24}$$

which may be rewritten as

$$D = 2\int p_1(\mathbf{x}_1)\, p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\|\mathbf{x}_0 - \int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\mathbf{u}_0\, d\mathbf{u}_0\|^2\, d\mathbf{x}_0\, d\mathbf{x}_1. \tag{2.25}$$

The derivation of Eq. 2.24 from Eq. 2.23 is well known. It says that the average

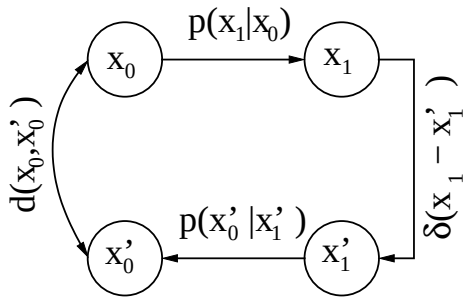

Figure 2.8: A single stage FMC as a vector quantizer when $d(\mathbf{x}_0, \mathbf{x}_0')$ is minimized with respect to $\mathbf{x}_1(\mathbf{x}_0)$. Linking the probabilistic transformations with the chain form of the Bayes rule ensures that $p_{0,1}(\mathbf{x}_0'|\mathbf{x}_1')$ cannot be varied independently of $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$.

Euclidean error between two independent identically distributed samples is twice the variance of the density the samples are drawn from. The introduction of the integration variable $\mathbf{u}_0$ into Eq. 2.25 will be discussed shortly. The desired result can finally be obtained with application of the Bayes rule

$$D = 2 \int p_0(\mathbf{x}_0)\, p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\ \|\mathbf{x}_0 - \int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\ \mathbf{u}_0\ d\mathbf{u}_0\|^2\ d\mathbf{x}_1\ d\mathbf{x}_0. \qquad (2.26)$$

Equation 2.26 has the right form to relate FMCs with VQs. It has a source distribution of input vectors $p_0(\mathbf{x}_0)$, a soft encoder $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$ and a reference vector

$$\int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\ \mathbf{u}_0\ d\mathbf{u}_0 \qquad (2.27)$$

attached to each encoding $\mathbf{x}_1$. The interpretation of the integration variable $\mathbf{u}_0$ is that of a soft codebook value, which is used to compute the reconstruction $\mathbf{x}_0'$ of $\mathbf{x}_0$ by integrating it out from Eq. 2.27.

The differences between the FMC formulation and the standard VQ are

1. The encoder is not a winner take all encoder. Rather each input vector is transformed into each output vector according to its density $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$.
2. The reconstruction $\int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\mathbf{u}_0\ d\mathbf{u}_0$ of $\mathbf{x}$ is associated with the encoder through Bayes rule. In standard VQ the reference vectors and the encoder are usually related through nearest neighbor relation.

When the soft decoder

$$\int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\mathbf{u}_0\ d\mathbf{u}_0$$

is replaced with a function $\mathbf{x}_0'(\mathbf{x}_1)$, to reconstruct the original input from its encoding we get

$$D = 2 \int p_0(\mathbf{x}_0)\, p_{1,0}(\mathbf{x}_1|\mathbf{x}_0) \|\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_1)\|^2\ d\mathbf{x}_0\ d\mathbf{x}_1. \qquad (2.28)$$

Functionally fifferentiating Eq. 2.28 with respect to decoder function $\mathbf{x}_0'(\mathbf{x}_1)$ yields

$$\frac{\delta D}{\delta \mathbf{x}_0'(\mathbf{x}_1)} = 4 \int p_0(\mathbf{x}_0) p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)[\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_1)]\, d\mathbf{x}_0. \tag{2.29}$$

The stationary point where $\frac{\delta D}{\delta \mathbf{x}_0'(\mathbf{x}_1)} = 0$ is obtained when

$$\mathbf{x}_0'(\mathbf{x}_1) = \frac{\int p_0(\mathbf{x}_0) p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\ \mathbf{x}_0\ d\mathbf{x}_0}{\int p_0(\mathbf{x}_0) p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\ d\mathbf{x}_0}. \tag{2.30}$$

Application of the Bayes rule reduces the stationarity condition to

$$\begin{aligned} \mathbf{x}_0'(\mathbf{x}_1) &= \frac{\int p_0(\mathbf{x}_0) p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\ \mathbf{x}_0\ d\mathbf{x}_0}{p_1(\mathbf{x}_1)} \\ &= \frac{p_1(\mathbf{x}_1) \int p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\ \mathbf{x}_0\ d\mathbf{x}_0}{p_1(\mathbf{x}_1)} \\ &= \int p_{0,1}(\mathbf{x}_0|\mathbf{x}_1)\mathbf{x}_0\ d\mathbf{x}_0\ . \end{aligned} \tag{2.31}$$

Thus the modified expression for D reduces to the original providing that the reconstructing function $\mathbf{x}_0'(\mathbf{x}_1)$ minimizes D. This simplification is significant as it allows replacing the Bayesian coupling between the encoder $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$ and the reconstructions

$$\int p(\mathbf{u}_0|\mathbf{x}_1)\mathbf{u}_0\ d\mathbf{u}_0$$

with Eq. 2.28 providing that $\mathbf{x}_0'(\mathbf{x}_1)$ satisfies the stationarity condition in Eq. 2.31.

At this point the minimization of D with respect to $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$ remains which is done below in simplified form. If the soft encoder $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$ is replaced with the standard VQ encoder, namely nearest neighbor prescription

$$p_{1,0}(\mathbf{x}_1|\mathbf{x}_0) \rightarrow \delta[\mathbf{x}_1 - \mathbf{x}_1(\mathbf{x}_0)]$$

the expression for distortion D in Eq. 2.26 can now be written

$$D = 2 \int p(\mathbf{x}_0) \|\mathbf{x}_0 - \mathbf{x}_0'[\mathbf{x}_1(\mathbf{x}_0)]\|^2\ d\mathbf{x}_0. \tag{2.32}$$

which turns out to be the expression for the continuum version of the standard VQ. The gradient of D with respect to $\mathbf{x}_0'(\mathbf{x}_1)$ is

$$\frac{\delta D}{\delta \mathbf{x}_0'(\mathbf{x}_1)} = 4 \int p_0(\mathbf{x}_0)\delta[\mathbf{x}_1 - \mathbf{x}_1(\mathbf{x}_0)][\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_1)]\, d\mathbf{x}_0. \tag{2.33}$$

Which leads to to the following batch training rule for minimizing D

$$\begin{aligned} \mathbf{x}_1(\mathbf{x}_0) &= \arg\min_{\mathbf{x}_1} \|\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_1)\|^2 \\ \mathbf{x}_0'(\mathbf{x}_1) &= \frac{\int p(\mathbf{x}_0)\delta[\mathbf{x}_1 - \mathbf{x}_1(\mathbf{x}_0)]\mathbf{x}_0\ d\mathbf{x}_0}{\int p(\mathbf{x}_0)\delta[\mathbf{x}_1 - \mathbf{x}_1(\mathbf{x}_0)]\ d\mathbf{x}_0}. \end{aligned} \tag{2.34}$$

thus completing the derivation of the nearest neighbor VQ from a single stage FMC.

2.4.3 Two Stage Folded Markov Chain as a SOM

The derivation of the continuum limit LBG algorithm from the single stage FMC can be extended to a two stage FMC in Fig. 2.9 corresponding with the SOM (Luttrell, 1994). The derivation is quite similar with the previous one for the normal vector quantizer so it will only be presented in an abbreviated form. For the sake of simplicity the second encoding stage, which corresponds with the neighborhood or encoding noise and is not optimized, is fused into the first stage with the chain form of the Bayes rule to obtain

$$\begin{aligned} p_{2,0}(\mathbf{x}_2|\mathbf{x}_0) &= \int p_{2,1}(\mathbf{x}_2|\mathbf{x}_1)\, p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\; d\mathbf{x}_1 \\ p_{0,2}(\mathbf{x}_0|\mathbf{x}_2)\, p(\mathbf{x}_2) &= p_{2,0}(\mathbf{x}_2|\mathbf{x}_0)\, p_0(\mathbf{x}_0) \end{aligned}$$

Using these abbreviations the expression for distortion D becomes

$$D = \int p_0(\mathbf{x}_0)\, p_{2,0}(\mathbf{x}_2|\mathbf{x}_0)\delta(\mathbf{x}_2 - \mathbf{x}_2')\, p_{0,2}(\mathbf{x}_0'|\mathbf{x}_2')\|\mathbf{x}_0 - \mathbf{x}_0'\|^2\; d\mathbf{x}_0\; d\mathbf{x}_2\; d\mathbf{x}_2'\; d\mathbf{x}_0'. \tag{2.35}$$

which after the steps taken with the vector quantizer simplifies to

$$D = 2\int p_0(\mathbf{x}_0)\, p_{2,0}(\mathbf{x}_2|\mathbf{x}_0)\; \|\mathbf{x}_0 - \int p_{0,1}(\mathbf{u}_0|\mathbf{x}_1)\mathbf{u}_0\; d\mathbf{u}_0\|^2\; d\mathbf{x}_2\; d\mathbf{x}_0. \tag{2.36}$$

The subsequent steps already taken through Eq. 2.26 with the single stage model lead in the case of the two stage model to

$$\begin{aligned} \mathbf{x}_0'(\mathbf{x}_2) &= \frac{\int p_o(\mathbf{x}_0)\, p_{2,0}(\mathbf{x}_2|\mathbf{x}_0)\mathbf{x}_0\; d\mathbf{x}_0}{\int p_0(\mathbf{x}_0)\, p_{2,0}(\mathbf{x}_2|\mathbf{x}_0)\; d\mathbf{x}_0} \\ &= \int p_{0,2}(\mathbf{x}_0|\mathbf{x}_2)\mathbf{x}_0\; d\mathbf{x}_0. \end{aligned} \tag{2.37}$$

Replacing the soft encoder $p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)$ with a winner take all scheme and keeping in mind that $p_{2,0}(\mathbf{x}_2|\mathbf{x}_0) = \int p_{2,1}(\mathbf{x}_2|\mathbf{x}_1)\, p_{1,0}(\mathbf{x}_1|\mathbf{x}_0)\; d\mathbf{x}_1$ leads to

$$D = 2\int p_0(\mathbf{x}_0)\, p_{2,1}(\mathbf{x}_2|\mathbf{x}_1(\mathbf{x}_0))\|\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_2)\|^2\; d\mathbf{x}_0\; d\mathbf{x}_2 \tag{2.38}$$

The functional derivative corresponding with Eq. 2.33 is then

$$\frac{\delta D}{\delta \mathbf{x}_0'(\mathbf{x}_2)} = 4\int p_0(\mathbf{x}_0)\, p_{2,1}(\mathbf{x}_2|\mathbf{x}_1(\mathbf{x}_0))[\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_2)]\; d\mathbf{x}_0. \tag{2.39}$$

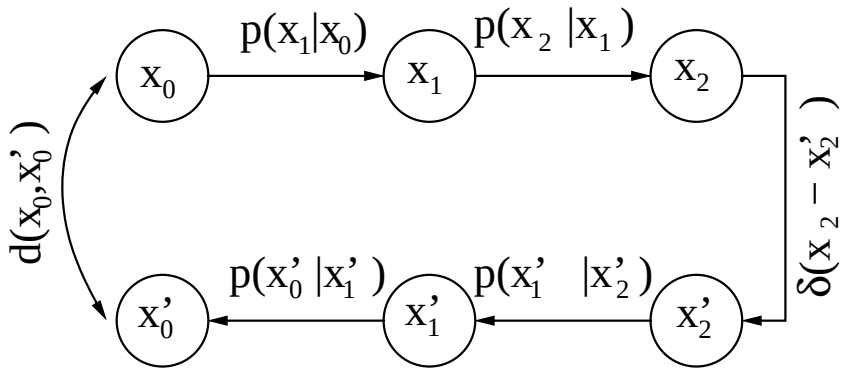

Figure 2.9: A two stage FMC as the SOM when $d(\mathbf{x}_0, \mathbf{x}_0')$ is minimized with respect to $\mathbf{x}_2(\mathbf{x}_0)$. The pair of second stage transformations corresponding with the SOM neighborhood or noise perturbing encoding is not optimized unlike the first pair of transformations to minimize $d(\mathbf{x}_0, \mathbf{x}_0')$.

which leads to the batch training rule already derived in section 2.3. The encoding rule for the two stage FMC becomes

$$\mathbf{x}_1(\mathbf{x}_0) = \arg\min_{\mathbf{x}_1} \int p_{2,1}(\mathbf{x}_2|\mathbf{x}_1)\|\mathbf{x}_0 - \mathbf{x}_0'(\mathbf{x}_2)\|^2 \, d\mathbf{x}_2; \tag{2.40}$$

and the decoding or reconstruction rule is

$$\mathbf{x}_0'(\mathbf{x}_2) = \frac{\int p_0(\mathbf{x}_0)\, p_{2,1}[\mathbf{x}_2|\mathbf{x}_1(\mathbf{x}_0)]\mathbf{x}_0 \, d\mathbf{x}_0}{\int p_{2,1}(\mathbf{x}_0)\, p(\mathbf{x}_2|\mathbf{x}_1(\mathbf{x}_0)] \, d\mathbf{x}_0}. \tag{2.41}$$

This rule can be interpreted as a vector quantizer whose encoding is corrupted with $p_{2,1}(\mathbf{x}_2|\mathbf{x}_1)$ in the encoding stage. The model turns out to be a continuum limit SOM with the minimum distortion rule (Luttrell, 1989) derived in section 2.3.

2.5 Chapter Summary

In the previous section the Self Organizing Map was looked into and derived from several points of view. At first we reviewed the origins of the map and then derived essentially the same result from seemingly quite different perspectives. We treated the SOM as a vector quantizer regularized by the neighborhood that brings order into the mapping, reduces the chances of getting stuck at local minima and provides means to avoid overfitting into the data. After vector quantizer interpretation we looked into noise corrupted communication channel interpretation of the map, which led to a variant of SOM that solves the inconsistency in the

competition and update parts of the training rule. In practise, however, the models produce quite similar results. Finally we derived the SOM as a special case of much broader family of probabilistic transformations provided by a construct called folded Markov chain.

There is little doubt that the key aspect of the SOM algorithm is its elegant simplicity. Namely how little is required to create a model that exhibits self organization and lends itself well to several alternate operators in units, thus providing a framework for a self organizing collection of competing models. While only very limited proofs of organization for the SOM have been managed (see e.g. Flanagan (2001)) in practise the map seems to organize as well as its low dimensional grid and locally operating neighborhood allows it to.

There are several alternatives to the SOM of which we will briefly discuss two. Genrative Topographic Mapping (GTM) (Bishop et al., 1997) was designed to be a principled alternative to the SOM. In GTM the latent space interpretation of the SOM is made explicit, by projecting a latent manifold defined with a set a radial basis functions into the data space. The properties of these basis functions determine the stiffness of the latent manifold and thus work like the neighborhood of the SOM. To keep the otherwise very complicated computations tractable the latent manifold is sampled on a regular grid corresponding with map units. The resulting model turns out to be a constrained kernel density estimator that can be trained with an appropriate expectation maximization (EM) algorithm.

Another alternative to the SOM is the recently published locally linear embedding (LLE) (Roweis and Saul, 2000). In the LLE each data point is represented as accurately as possible as a linear combination of its n closest neighbors such that the sum of weighting coefficients for every data point is costrained to equal one. This produces a sparse matrix $\mathbf{W}$ of linear weigthing coefficients. The neighborhood preserving mapping is then constructed by keeping $\mathbf{W}$ constant and finding the two dimensional set of points that would as accurately as possible have the same weighting coefficient matrix $\mathbf{W}$ as the original data. The LLE method projects points with similar weighting coefficients close in the low dimensional projection space. This observation links the method with linear auto regressive models that also capture non linear processes with simple linear models.

Chapter 3

Properties of the SOM

Despite its computational simplicity, or perhaps for the very reason, the SOM is a tough nut to crack when rigorously analyzed. These difficulties stem from the origins of the SOM as it was heuristically defined through its learning rule rather than learning target. The SOM model exhibits two traits, on one hand it quantizes the data with the respect to its error function and on the other hand it models the structure of the manifold.

3.1 Lack of Energy Function

The analysis of the SOM would be facilitated a great deal if we could derive an energy function for the update rule. A continuous function gradient of which would lead to the update rule in Eq. 2.3 would be such an energy function. This function could be used to analyze the behavior of the SOM instead of considering lengthy simulated runs.

A natural candidate for an energy function for the SOM is the continuous generalization $E'(p_{\mathbf{x}}(\mathbf{x}), M)$ of the discrete error Eq. 2.7 for a continuous random variable $\mathbf{x}$ with probability density $p_{\mathbf{x}}(\mathbf{x})$ (Luttrell, 1989). This function can be written as

$$E'(p_{\mathbf{x}}(\mathbf{x}), M) = \sum_{i \in M} \int H(\mathbf{x}, i) \|\mathbf{x} - \mathbf{m}_i\|^2 p_{\mathbf{x}}(\mathbf{x})\, d\mathbf{x}, \tag{3.1}$$

where $\mathbf{m}_j \in M$ are the codebook vectors of map M and $\mathbf{x}$ are the possible inputs each weighted in the error with the corresponding density $p_{\mathbf{x}}(\mathbf{x})$. However it is a quite straightforward task to show that the expected error in Eq. 3.1 is not an energy function. The result is easy to understand and is a consequence of the discontinuous Voronoi partitioning of the input space coupled with the neighborhoods that extend over the boundaries of the Voronoi cells.

The instantaneous error function in Eq. 2.7 is the sum of the sample errors. If we consider a sample $\mathbf{x}$ that lies at a boundary of a cell we realize that differential movement of the boundaries will change the *bmu* of $\mathbf{x}$ and thus change the neighborhood in which the error is computed. In general this will cause a discontinuity in the error and consequently the error function is not differentiable for such a sample, thus disqualifying Eq. 3.1 as an energy function.

There is an elegant argument (Erwin et al., 1992a) to show that the training rule of the SOM cannot be the gradient of any energy function. First we define the force field corresponding with the update rule and show that the change in the potential energy in the corresponding energy field when traversing from one point to another is generally dependent on the path. Such field is not energy conserving.

For a continuous random variable $\mathbf{x}$ with density $p_{\mathbf{x}}(\mathbf{x})$ the expected force F_i acting on weight $\mathbf{m}_i$ is

$$\begin{aligned} F_i &= \int H(\mathbf{x}, i)(\mathbf{x} - \mathbf{m}_i)\, p(\mathbf{x}) d\mathbf{x} \\ &= \sum_{j \in M} h(j, i) \int_{\mathbf{x} \in V_j} (\mathbf{x} - \mathbf{m}_i)\ p(\mathbf{x})\ d\mathbf{x}, \end{aligned} \tag{3.2}$$

where the integral is split into a sum of integrals over Voronoi cells V_j defined by the the map. Note that the neighborhood can be moved out from the integration as the value of the neighborhood is only dependent on pair of units i and j.

When we consider a string of N units in $[0, 1]$ for scalar input x in the same interval we can define

$$\begin{array}{lcccl} 0 & < & V_1 & < & \frac{1}{2} \cdot (u_1) \\ \frac{1}{2}(u_{l-1} + u_l) & < & V_l & < & \frac{1}{2}(u_l + u_{l+1}) \\ \frac{1}{2}(u_{N-1} + u_N) & < & V_N & < & 1 \end{array} \tag{3.3}$$

for the Voronoi cells, when the weights m_i, are permuted into ascending order as

$$\begin{aligned} u_1 &< u_2 \\ u_{l-1} &< u_l < u_{l+1}\ \forall\ l \in \{2...N-1\} \\ u_{N-1} &< u_N, \end{aligned}$$

and denoted with u_l. The original order can be restored with a permutation function $i = P(l)$, which is unique to each configuration of the map. Consequently we can write $u_l = m_{P(l)} = m_i$.

In the case of uniform $p_x(x)$ the solution to the integral in Eq. 3.2 for each i

is

$$
\begin{aligned}
F_i = \sum_{l=2}^{N-2} \hat{h}(l,i) \left[\frac{1}{8}(u_{l+1}^2 - u_{l-1}^2) + \frac{1}{4}u_l(u_{l+1} - u_{l-1}) + \frac{1}{2}u_i(u_{l-1} - u_{l+1}) \right] + \\
\hat{h}(1,i) \cdot \left[\frac{1}{8}(u_1 + u_2)^2 - u_i \frac{1}{2}(u_1 + u_2) \right] + \\
\hat{h}(N,i) \cdot \left[\frac{1}{2} - u_i - \frac{1}{8}(u_N + u_{N_1})^2 + u_i \frac{1}{2}(u_N + u_{N-1}) \right],
\end{aligned}
\tag{3.4}
$$

where $\hat{h}(l,i)$ is the permuted neighborhood function. The terms of Eq. 3.4 correspond with units internal to the map and both ends respectively. We can treat these forces as an N dimensional force or a vector field $\vec{F}$ and look into the properties of this field.

A force field is considered conservative if the change in potential energy when traversing from one point to another is not dependent on the path like one can expect in a physical situation of Earths gravity field for example. This property has several alternate definitions one of which is of particular interest to us. *Rotor* or a *curl* operator (Zachmanoglou and Thoe, 1986) of a three dimensional vector field is defined as the cross product between the field and the $\nabla = \frac{\partial}{\partial u}$ operator. The *rotor* of any three dimensional subfield $\vec{F}_{3D}$ of $\vec{F}$ is therefore

$$
\begin{aligned}
\mathrm{rot}(\vec{F}_{3D}) &= \nabla \times \vec{F}_{3D} \\
&= \begin{vmatrix} x & y & z \\ \frac{\partial}{\partial u_l} & \frac{\partial}{\partial u_m} & \frac{\partial}{\partial u_n} \\ F_l & F_m & F_n \end{vmatrix} \\
&= \left(\frac{\partial F_n}{\partial u_m} - \frac{\partial F_m}{\partial u_n} \right) x + \left(\frac{\partial F_l}{\partial u_n} - \frac{\partial F_n}{\partial u_l} \right) y + \left(\frac{\partial F_m}{\partial u_l} - \frac{\partial F_l}{\partial u_m} \right) z \ ,
\end{aligned}
\tag{3.5}
$$

where l, m and n is any triple from $1...N$ and x, y and z are the normal $3D$ basis vectors. The terms of the rotor vector capture the interaction between the weights during training. In essence each term of the vector tells how much the change in the other two weights influence the force acting on the third. Only if the rotor operator vanishes everywhere for all three dimensional subfields the field can be derived as the gradient of some continuous potential function

$$
\frac{\partial \breve{E}(u)}{\partial u} = -\vec{F}
$$

Unfortunately in general this is not the case with the SOM as the weights interact through the neighborhood and these interactions are not pairwise canceled in our example, as could be shown by substituting Eq. 3.4 into Eq. 3.5.

In fact the forces do not even follow a more general condition (Erwin et al., 1992a)

$$\vec{F}_{3D}(u) \cdot \mathrm{rot}(\vec{F}_{3D}(u)) = 0, \tag{3.6}$$

which means that the turbulence in the field is not orthogonal to the forces and thus the weight interactions cannot be accounted for by any arbitrary function $\eta(u)$ to get

$$\frac{\partial \breve{E}(u)}{\partial u} = -\eta(u)\vec{F}.$$

The lack of energy function, however, is not of major practical consequence though the analysis of the map would be facilitated somewhat if such a function existed. Once the map has been well organized it follows relatively closely the function defined in Eq. 3.1. After proving the lack of energy function for the SOM the effort shifted from the search of such function to the definition of models with a well defined energy functions that exhibit SOM like behavior (Wiskott and Sejnowski, 1997; Goodhill and Sejnowski, 1997; Graepel et al., 1997).

3.2 Generative Probability Density for the SOM

Typically the problem of model selection is not considered in conjunction with the SOM and as a matter of fact practically applicable methods for model selection have not been available until recently. The problem of model selection involves model complexity: How much of the data generating process can be inferred from the available data. In other words how well the available observations generalize to future observations. The users of the SOM frequently omit the problem thus implicitly assume they work with the population instead of a sample. We can approach the model selection problem by treating the SOM as a data generating mechanism and deriving an approximation of the corresponding density model (Lampinen and Kostiainen, 2001). This density model for the first time provides means for quantitative model selection with SOM.

3.2.1 Approximate Likelihood Function

The development of the density model for the standard SOM begins with the instantaneous error function

$$E(\mathbf{X}, M) = \sum_{\mathbf{x} \in \mathbf{X}} \sum_{i \in M} H(\mathbf{x}, i)(\mathbf{x} - \mathbf{m}_i)^2, \tag{3.7}$$

where $\mathbf{x} \in \mathbf{X}$ is a sample vector, $\mathbf{m}_i$ is the codebook vector of unit i on the map M and $H(\mathbf{x}, i)$ is the neighborhood function. In its converged state the map locally minimizes the error function in Eq. 3.7 only defined for a discrete data set $\mathbf{X}$ and fixed neighborhood parameters.

The maximum likelihood method (ML) chooses the model that maximizes the likelihood of the model having generated the data. If the data samples are considered independent the likelihood of the data is the product of its sample likelihoods

$$P_{\mathbf{X}}(\mathbf{X}|M) = \prod_{\mathbf{x}\in\mathbf{X}} P_{\mathbf{x}}(\mathbf{x}|M),$$

where $P_{\mathbf{X}}(\mathbf{X}|M)$ is the likelihood of the data $\mathbf{X}$ and $P_{\mathbf{x}}(\mathbf{x}|M)$ is the likelihood of a sample $\mathbf{x} \in \mathbf{X}$ conditioned on the map M. The target is to find a likelihood function which is consistent with the error function Eq. 3.7. The negative log likelihood

$$L = -\log\left(P_{\mathbf{X}}(\mathbf{X}|M)\right)$$

when related with Eq. 3.7 approximately yields

$$\begin{aligned} P_{\mathbf{X}}(\mathbf{X}|M,\beta) &= Z' \cdot \exp\left[-\beta E(\mathbf{X}, M)\right] \\ &= Z' \cdot \prod_{\mathbf{x}\in\mathbf{X}} exp\left[-\beta \sum_{i\in M} H(\mathbf{x}, i)(\mathbf{x}-\mathbf{m}_i)^2\right] \end{aligned} \tag{3.8}$$

where Z' and β are normalizing factors. Observe that 3.8 is only an approximation of the real likelihood function, if such exists, as the normalizing factor Z depends on the codebook vectors of M but its logarithm is ignored in the training rule. The quality of the approximation made here is an open question and requires more work to assess. However our intuition is that the dependence of Z on the codebook is not particularly sharp, hence we believe the approximation is well justified.

3.2.2 Full Density

Since the likelihood of the data is the product of the sample probabilities, the probability density function is given by

$$p_{\mathbf{x}}(\mathbf{x}|M,\beta) = Z \cdot \exp\left[-\beta \sum_{i\in M} H(\mathbf{x}, i)(\mathbf{x}-\mathbf{m}_i)^2\right]. \tag{3.9}$$

From this form it becomes apparent that the density is a product of Gaussian components. The function has discontinuities at the boundaries of the Voronoi cells due to hard best matching unit assignment and neighborhood function of the SOM algorithm. We can write the function in a different form by completing the sum over the codebook vectors. With a little algebra and setting

$$\boldsymbol{\mu}_i \sum_j h(i, j) = \sum_j h(i, j)\mathbf{m}_j$$

where $h(i, j) = H(\mathbf{x}, j) \;\; \forall \;\; \mathbf{x} \in V_i$ is the neighborhood strength between units i and j, we obtain

$$\begin{aligned} p_{\mathbf{x}}(\mathbf{x}|\mathbf{x} \in V_i, M, \beta) &= Z \cdot \exp\left[\sum_{j \in M} -\beta h(i, j)(\mathbf{x} - \mathbf{m}_j)^2\right] \\ &= Z \cdot \exp\left[-\beta \sum_{j \in M} h(i, j)\left(\mathbf{x}^2 - 2\mathbf{x}\mathbf{m}_j + (\boldsymbol{\mu}_i + (\mathbf{m}_j - \boldsymbol{\mu}_i))^2\right)\right] \\ &= Z \cdot \exp\left[-\beta(\mathbf{x} - \boldsymbol{\mu}_i)^2 \sum_{j \in M} h(i, j) - \beta \sum_{j \in M} h(i, j)(\mathbf{m}_j - \boldsymbol{\mu}_i)^2\right] \end{aligned} \tag{3.10}$$

After making the the following substitutions:

- $\boldsymbol{\mu}_i = \frac{\sum_{j \in M} h(i,j)\, \mathbf{m}_j}{\sum_{j \in M} h(i,j)}$ is the expected value of the component density bounded by the Voronoi cell V_i.
- $s_i^2 = \left(\beta \sum_{j \in M} h(i, j)\right)^{-1}$ is the variance of the Gaussian in V_i with expected value $\boldsymbol{\mu}_i$. Note that the data influences s_i^2 only through the noise variance parameter β.
- Finally $W_i = \sum_{j \in M} h(i, j)(\mathbf{m}_j - \boldsymbol{\mu}_i)^2$ weights the importance of V_i against the other Gaussians. The intuitive rationale is that if the neighborhood is tight, W_i is small and the estimated density $p_{\mathbf{x}}(\mathbf{x})$ in V_i is high, as can be expected with the SOM (Cottrell, 1997, 1998).

we get

$$p_{\mathbf{x}}(\mathbf{x}|\mathbf{x} \in V_i, M) \;\; = \;\; Z \cdot \exp\left(-\beta W_i\right) \cdot \exp\left[\frac{1}{s_i^2}(\mathbf{x} - \boldsymbol{\mu}_i)^2\right]. \tag{3.11}$$

Since the integral of the density function has to converge to 1, the normalizing constant Z and the noise variance parameter β are coupled with

$$\int p_{\mathbf{x}}(\mathbf{x}|M, \beta)\, d\mathbf{x} = Z \cdot \sum_{i \in M}\left[\exp\left(-\beta W_i\right) \int_{V_i} \exp\left(\frac{1}{s_i^2}(\mathbf{x} - \boldsymbol{\mu}_i)^2\right)\, d\mathbf{x}\right] = 1, \tag{3.12}$$

where the integral over the density is decomposed into a sum of integrals over each Voronoi cell on the map. The component integrals of Eq. 3.12 are not analytically integrable, but they can be estimated with a Monte Carlo method using the following relatively simple rejection sampling technique in Alg. 2.

Algorithm 2: Monte Carlo algorithm for estimating component volumes.

1. For each component integral pick L samples from normal distribution $\mathcal{N}(\boldsymbol{\mu}_i, s_i^2)$.
2. Compute the acceptance ratio q_i as $q_i = L_i/L$ i.e. the proportion of the samples in L in the Voronoi cell V_i.
3. The integral can now be estimated with $q_i \cdot \left(2\pi s_i^2\right)^{(-\mathrm{d}_X/2)}$, where d_X is the dimension of the input data. In other words the estimate is the acceptance ratio times the total volume of the un-normalized Gaussian for V_i.

The variance of the acceptance ratio q_i depends heavily on the neighborhood. If the neighborhood is large it is possible that $\boldsymbol{\mu}_i$ is relatively far from the codebook vector $\mathbf{m}_i$ and thus the Voronoi cell may be significantly offset from its component density $\mathcal{N}(\boldsymbol{\mu}_i, s_i^2)$. On the other hand with a small neighborhood the chances of $\boldsymbol{\mu}_i$ having large offset from $\mathbf{m}_i$ are much smaller and thus the Voronoi cell sits much better over the component density. Once the component integrals have been estimated for a given β the corresponding Z is readily available and we have the complete density.

To search for the optimal β we can use cross validation (see for example Bishop (1995)). For this we split the available data into two as independent as possible subsets. One of the subsets is used to train the map. For the other subset we compute the approximate likelihood with Eq. 3.8, which requires estimation of the normalizing constant Z for the candidate β. The aim is to find a likelihood maximizing value for β. We can pick the candidate values for β with normal line search methods such as golden bisection (see for example Bazaraa et al. (1993)).

We can find an interpretation for the maximum likelihood solution β^{ML} of the noise variance parameter β in terms of the neighborhood weighted distortion

$$E(\mathbf{x}, M) = \sum_i H(\mathbf{x}, j)(\mathbf{x} - \mathbf{m}_i)^2$$

by setting

$$\frac{\partial P_{\mathbf{X}}(\mathbf{X}|M, \beta^{ML})}{\partial \beta^{ML}} = 0.$$

With algebraic manipulations we get

$$\frac{\sum_{\mathbf{x}_i \in \mathbf{X}} E(\mathbf{x}_i, M)}{\Omega_{\mathbf{X}}} = \frac{\frac{\partial Z}{\partial \beta^{ML}}}{Z}. \tag{3.13}$$

where $\Omega_{\mathbf{X}}$ is the cardinality of $\mathbf{X}$. Using $Z = \left(\int p_{\mathbf{x}}(\mathbf{x}|M, \beta)d\mathbf{x}\right)^{-1}$ we obtain

$$\begin{aligned} Z &= \int \exp\left(\beta E(\mathbf{x}, M)\right) d\mathbf{x} \quad \text{and} \\ \frac{\partial Z}{\partial \beta} &= \int E(\mathbf{x}, M) \exp\left(\beta E(\mathbf{x}, M)\right) d\mathbf{x} \end{aligned} \tag{3.14}$$

where the input distribution has been estimated with

$$\hat{p}(x|M, \beta) \propto \exp(-\beta E(x, M)).$$

Substituting Eq. 3.14 into Eq.3.13 yields

$$\frac{\sum_{i \in \mathbf{X}} E(\mathbf{x}, M)}{\Omega_{\mathbf{X}}} = \frac{\int E(\mathbf{x}, M) \exp\left(\beta^{ML} E(\mathbf{x}, M)\right) d\mathbf{x}}{\int \exp\left(\beta^{ML} E(\mathbf{x}, M)\right) d\mathbf{x}} \tag{3.15}$$

Heuristically Eq. 3.15 states that at $\beta = \beta^{ML}$ the expected error over the estimated input distribution equals the sample average of $E(\mathbf{x}, M)$ over the input data.

Figures 3.1–3.2 depict the *pdf* of a normal SOM while figures 3.3–3.4 depict the *pdf* of a SOM trained with minimum distortion rule. The figures emphasize the discontinuity of the *pdf* of the normal SOM while minimum distortion rule yields continuous density. Notice, however, that in the minimum distortion case for some neighborhoods the coresponding receptive fields do not contain the expected value of the component density. This is an immediate consequence of the component density weighting coefficients.

3.2.3 Model Selection with the Density Model

The complexity of the model for the data produced by the SOM is determined by the number of units, the map topology and the neighborhood function. When the input data is a sample of a much larger population the objective is to choose the model that best predicts the future observations from the population. The approximate likelihood function provides a way to compare the goodness of different maps.

If the map topology is considered fixed we can use the likelihood function in the following manner. The available data is split in two parts *training* and *validation* as independently as possible and the map is trained with the former part to a local optimum with several neighborhood widths in decreasing order. The algorithm described above is used to estimate the likelihood function for each of the maps for the estimated likelihood of the *validation* data. The map maximizing the likelihood is considered best. This method can directly be generalized to cross validation where the available data is split in N independent subsets that alternate in the roles of the *training* and *validation* data (see for example Bishop (1995)).

Similar approach can be used to determine the best topology as well. For a large map determining the optimal neighborhood width with the likelihood model is computationally costly. In such cases the neighborhood can be optimized with smaller map and scaled with the desired dimensions of the larger map. Figures 3.5 and 3.6 demonstrate the model selection procedure.

The likelihood model provides tools for model selection when minimizing the expected error. In other words for selecting the map that approximately minimizes the expected neighborhood weighted quantization error over future samples from the same process. However, it only indirectly addresses the embedding property, that is how well the map follows the input manifold, of the map by favoring maps where units with strong neighborhood appear close in the output space but it tells very little of larger scale folding of the map space into the input space. The embedding property of the map cannot be analyzed through the error function in the input space but has to be addressed through measures that directly asses the quality of the maps organization and its model of the input manifold. These measures are given a brief look in section 3.3

3.3 Topology Preservation on the SOM

The key difference between the SOM and pure vector quantization schemes is the latent map space defined by the neighborhood. The neighborhood serves two purposes. It brings order into the mapping by pulling nearby units in the latent space toward each other enforcing a smoothness constraint on the map thus regularizing it. In this section I will look into the ordering properties of the map and in particular ordering measures.

The ordering properties of the map must be considered in both directions to get a full picture of how the map has organized into the input manifold. Only when the true dimension of the manifold very closely matches the dimension of the map, preservation of order in both directions; from map space to input space and from input space to map space is possible. In general, however, discrepancy between the dimensions will unavoidably cause folds in one or the other of the mappings. Typically the dimension of the input manifold far exceeds the dimension of the map causing folding of the map to evenly cover the input manifold. For an idea of the folding consider a sheet of paper, an essentially two dimensional surface, folded to evenly fill a cube.

In the very restricted case of one dimensional map in one dimensional input space we can define a Liapunov-function (Cottrell et al., 1994; Kohonen, 1984) as the number of inversions in the direction of the map as

$$\psi_{Inv}(M) = \sum_{i=2}^{\Omega_M - 1} \operatorname{sgn}(m_i - m_{i-1}) \neq \operatorname{sgn}(m_{i+1} - m_i), \tag{3.16}$$

where Ω_M is the number of units on the map. However for more complex maps and input manifolds different measures have to be defined.

3.3.1 Topographic Product

The topographic product (Bauer et al., 1992; Polani, 2001) measures the degree of folding of the map. For the product two permutations of units are computed for each unit. First permutation function $n_M(i, j)$ orders the units in ascending order with respect to distance in the map space. Formally we can write

$$d_M(i, n_M(i, j)) \leq d_M(i, n_M(i, j+1)) \; \forall i \in M, \tag{3.17}$$

where $d_M(i, j)$ is a distance measure between units i and j in the map space. Permutation function $n_M(i, j)$ returns the index of the j:th closest unit to i with respect to $d_M(i, j)$. The other permutation function $n_\mathcal{X}(i, j)$ orders the units in ascending order with respect to distance in the input space $\mathcal{X}$. For this permutation we can write

$$d_\mathcal{X}(\mathbf{m}_i, \mathbf{m}_{n_\mathcal{X}(i,j)}) \leq d_\mathcal{X}(\mathbf{m}_i, \mathbf{m}_{n_\mathcal{X}(i,j)}) \; \forall i \in M, \tag{3.18}$$

where $d_\mathcal{X}(\mathbf{m}_i, \mathbf{m}_j)$ is a distance measure in the input space between the codebook vectors of units i and j. Like $n_M(i, j)$ above, $n_\mathcal{X}(i, j)$ returns the index of the j:th closest unit but now the criterion is different.

The permutations in Eqs. 3.17, 3.18 computed for every unit on the map, are used in the definition of two quantities Q_1 and Q_2 to measure the folding of the map. These quantities are written as follows

$$Q_1(i, j) = \frac{d_\mathcal{X}(\mathbf{m}_i, \mathbf{m}_{n_M(i,j)})}{d_\mathcal{X}(\mathbf{m}_i, \mathbf{m}_{n_\mathcal{X}(i,j)})} \tag{3.19}$$

$$Q_2(i, j) = \frac{d_M(i, n_M(i, j))}{d_M(i, n_\mathcal{X}(i, j))} \; . \tag{3.20}$$

The first quantity captures the ratio of distances in the input space between i and its j:th closest neighbor on the map $n_M(i, j)$ and in the input space $n_\mathcal{X}(i, j)$. Likewise the second quantity captures the ratio of distances in the map space for the same pair of units.

With Q_1 and Q_2 the topographic product is defined as

$$\mathcal{P}(i, j) = \left[\prod_{k=1}^{j} Q_1(i, k) \cdot Q_2(i, k) \right]^{\frac{1}{2k}} \; \forall j \in \{1 ... \Omega_M - 1\}, \tag{3.21}$$

where Ω_M is the cardinality of M i.e the number of units on the map. Thus for each unit i we compute a chain of $\Omega_M - 1$ products, one for each j, ending up

with total of $\Omega_M(\Omega_M - 1)$ values for $\mathcal{P}(\cdot, \cdot)$. The final measure to summarize the result $\mu_{\mathcal{P}}$ was defined as the logarithmic average of the products $\mathcal{P}(\cdot, \cdot)$ over the map

$$\mu_{\mathcal{P}} = \frac{1}{\Omega_M \cdot (\Omega_M - 1)} \sum_{i=1}^{\Omega_M} \sum_{j=1}^{\Omega_M - 1} \log(\mathcal{P}(i, j)). \tag{3.22}$$

A value of $\mu_{\mathcal{P}}$ near zero indicates good adaptation as it means that Q:s tend close to one indicating that the n:th closest unit in the input space is the n:th closest in the map and conversely that the n:th closest on the map is the n:th closest in the input space. Negative values of $\mu_{\mathcal{P}}$ indicate folding of the map into the input space while positive values indicate that the dimension of the map grid is too big for the input manifold.

The ordering relation $n_M(i, j)$ is generally not unique but different implementations generally lead to similar results (Polani, 2001). The main problem with the topographic product is the fact that it does not account for folds in the input manifold (Villmann et al., 1997). The product only looks into the shape of the map but as it turns out the input manifold may be of lower dimension than the input space and thus it also may have folds. A correctly ordered map will exhibit the same folding but the topographic product has no means to separate correct folding following true folds in the manifold from incorrect folding caused by dimension mismatch between the map and the input manifold. Topographic function (Villmann et al., 1997) was defined to counter this problem.

3.3.2 Topographic Function

For the topographic function we need to define Delaunay and the Hebb graph for the SOM (Martinetz and Schulten, 1994; Polani, 2001). The Delaunay graph $\mathcal{D}_M$ for the SOM M is a graph where units i and j are adjacent if their respective Voronoi cells, closed by their boundaries, share a $\mathrm{d}_{\mathcal{X}} - 1$ dimensional section, where $\mathrm{d}_{\mathcal{X}}$ denotes the dimension of the input space. In other words if units i and j have adjacent Voronoi cells the corresponding Delaunay graph has an edge connecting them. Slightly relaxed definition for the pseudo Delaunay graph only mandate that $V_i \cap V_j \neq \emptyset$ also linking units that share a lower than $\mathrm{d}_{\mathcal{X}} - 1$ dimensional section such as corner in the two dimensional case. The Hebb graph is a generalization of the Delaunay graph where each edge of the Delaunay graph is associated with the probability $P_{i,j}(i, j)$ of a an input sample falling in the $2nd$ order Voronoi cell of i and j defined as

$$V_{i,j} = \{x \in \mathcal{X} | \forall k \in M \backslash \{i, j\} \;\; d_{\mathcal{X}}(\mathbf{x}, \mathbf{m}_i) \leq d_{\mathcal{X}}(\mathbf{x}, \mathbf{m}_k) \wedge d_{\mathcal{X}}(\mathbf{x}, \mathbf{m}_j) \leq d_{\mathcal{X}}(\mathbf{x}, \mathbf{m}_k)\},$$

where $d_{\mathcal{X}}(\mathbf{x}, \mathbf{m}_i)$ is a distance metric in the input space. In other words the second order Voronoi cell $V_{i,j}$ is the region of the input space where i and j are the two closest units.

A seemingly simple algorithm for determining the Delaunay and Hebb graph as an $\Omega_M \times \Omega_M$ adjacency matrix $\mathbf{M}_{\mathcal{D}_M}$ involves drawing samples $\mathbf{x}$ from the corresponding density $p_{\mathbf{x}}(\mathbf{x})$. Observe that the input manifold $\mathcal{M}_{\mathcal{X}}$ as used in our context is also defined by the density function as data can only come from the region of $\mathrm{d}_{\mathcal{X}}$ dimensional input space where $p_{\mathbf{x}}(\mathbf{x}) > 0$. For each sample we determine the two closest units i and j, on the map and increment the corresponding matrix entries $\mathbf{M}_{\mathcal{D}_M}(i, j)$ and $\mathbf{M}_{\mathcal{D}_M}(j, i)$ in $\mathbf{M}_{\mathcal{D}_M}$ by one.

A conceivable difficulty is that usually we do not have the density $p_{\mathbf{x}}(\mathbf{x})$ and in fact quite frequently the density is what we are after. What we do have, however, is a sample $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{\Omega_{\mathbf{X}}}\}$ drawn from $p_{\mathbf{x}}(\mathbf{x})$, which we can use as an estimate for $p_{\mathbf{x}}(\mathbf{x})$. Thus the algorithm reduces to picking samples from the data set $\mathbf{X}$ and incrementing the corresponding entries in $\mathbf{M}_{\mathcal{D}_M}$. If the samples are picked randomly without removal, we will get an estimate for the Hebb graph. If each sample is considered once we will get an estimate of the Delaunay graph. A problem with this approach is the rapidly growing complexity of the graph. The number of possible edges in $\mathcal{D}_M$ grows as square of Ω_M, thus the demand for data grows rapidly with growing map to reliably approximate $\mathcal{D}$ (or $\mathbf{M}_{\mathcal{D}_M}$).

The topographic function requires the masked Delaunay graph, which considers masked Voronoi cells $\bar{V}_i = V_i \cap \mathcal{M}_{\mathcal{X}}$ that are clipped with the input manifold. Since in general we do not know the manifold we have to guess which connections suggested by the picking algorithm are between units whos masked Voronoi cells do not share boundary. We can do this with an ageing process (Villmann et al., 1997; Martinetz, 1993), where connections that are rarely hit are pruned out.

For the topographic function we define a metric in the Delaunay graph as $d_{\mathcal{D}_M}(i, j)$ being the smallest number of hops on the graph $\mathcal{D}_M$ to traverse from i to j. This metric is relatively easy to calculate from the unweighted $\mathbf{M}_{\mathcal{D}_M}$ elements of which are either one, if an edge exists between i and j on $\mathcal{D}_M$ or zero otherwise. We also need two distance metrics $d^1_M(i, j)$ and $d^\infty_M(i, j)$ in the map space defined as

$$d^1_M(i, j) = \sum_{l=1}^{\mathrm{d}_M} |c_i(l) - c_j(l)| \tag{3.23}$$

$$d^\infty_M(i, j) = \max_l |c_i(l) - c_j(l)|, \tag{3.24}$$

where $\mathbf{c}_i$ is the vector of latent map space co-ordinates of unit i and d_M is the dimension of the latent map space. With these definitions for metrics we define

$$f_i(k) = \begin{cases} \Omega\{j | d_{\mathcal{D}_M}(i, j) = 1, d^\infty_M(i, j) > k\} & \text{for } k > 0 \\ \Omega\{j | d^1_M(i, j) = 1, d_{\mathcal{D}_M}(i, j) > -k\} & \text{for } k < 0 \end{cases} \tag{3.25}$$

where Ω denotes cardinality of the sets. Function $f_i(k)$ for $k > 0$ returns the number of units immediately connected with i i.e. $d_{\mathcal{D}_M}(i, j) = 1$ on Delaunay

graph that have distance $d_M^{\infty}(i, j) > k$ on the map. As a consequence $f_i(k)$ for $k > 0$ measures mismatch in the mapping from the input space to the map space. Conversely for $k < 0$ $f_i(k)$ counts the number of units in the immediate neighborhood of i i.e $d_M^1(i, j) = 1$ that are further than k hops away on the graph $\mathcal{D}_M$ i.e. $d_{\mathcal{D}_M}(i, j) > k$, thus measuring the mismatch in the mapping from the map space to the input space. With $f_i(k)$ the topographic function is then defined

$$\Phi(k) = \begin{cases} \frac{1}{\Omega_M} \sum_{i \in M} f_i(k) & \text{for } k \neq 0 \\ \Phi(1) + \Phi(-1) & \text{for } k = 0 \end{cases} \tag{3.26}$$

The topographic function contains more information than a plain number obtained from the topographic product Eq. 3.22. It gives a direction of the mismatch. If $\Phi(k)$ is positive for positive k the dimension of the map is insufficient for the manifold and thus the map has folded. The largest value of k gives the maximum extent of theses folds. Likewise positive $\Phi(k)$ for negative k indicates that the map has too high dimension for the input manifold. In practice this however is hardly ever the case. The topographic function is a powerful tool for determining the quality of mapping by the SOM but the inherent difficulty in estimating density, caused by the curse of dimensionality (Bishop, 1995), for the masked Delaunay graph may limit its usefulness.

3.4 Chapter Summary

In this chapter we reviewed some key work in the analysis of the SOM. The first two topics are related as they both consider the error or distortion measure of the SOM. The lack of energy function is a consequence of the inconsistent competition and update parts of training rule as it causes non differentiable steps on error at the boundaries of the Voronoi cells. The anomalies in the density model are caused by the same steps in the error function, which should come as no surprise considering the relationship of the error function with the approximate negative log likelihood of the map. The resulting density model is a collection of Gaussian components bounded by the Voronoi cells. The expected values of the components are the neighborhood weighted means of the codebook vectors, thus depending on the final neighborhood radius they may be considerably offset from the centroids of their respective Voronoi cells. This in turn causes non differentiable steps in the density. When the map is trained with the minimum distortion rule, it gains both an energy function and continuous density. On the balance, however, the edge phenomena caused by the units near the edge areas of the map having non symmetrical neighborhoods becomes more pronounced as the neighborhoods are used both in competition and update. As a consequence the map shows stronger tendency to compress than the normal SOM as we can observe when comparing the excerpts in Figs. 3.2 and 3.4, thus the partitioning

of the input manifold by the minimum distortion map is less even than the partitioning by the normal SOM. In this sense the compromise made in the learning rule of the original SOM was in my opinion beneficial.

In data analysis tasks the SOM is frequently used as a tool for reducing data dimensionality. The point of exercise in dimensionality reduction is to find a lower dimensional manifold, the data is sampled from, than the input space. A folded sheet of paper is an example of a two dimensional manifold that produces a three dimensional set of points when sampled. The neighborhood of the SOM brings about local but not global order as the radius of the neighborhood is limited to units close in the latent space. Global order, that is how well the map has organized into the input manifold, must therefore be measured separately to determine how good a job the SOM has done in dimensionality reduction. The operators reviewed in the third topic of chapter three address the global order on the map. The topographic product is limited to linear manifolds as it is unable to separate folds on the map following folds in the manifold from dimension mismatch. The topographic function accounts for non linear, folded, manifolds but reliable estimation of the topographic function may be difficult. The measures reviewed here are not the only proposed. For example the intrinsic distance measure (Polani (2001), originally proposed by Kaski and Lagus (1996)) considers the curvature of the map by measuring the Euclidean distance of the two closest codebook vectors directly in the input space and along the map surface.

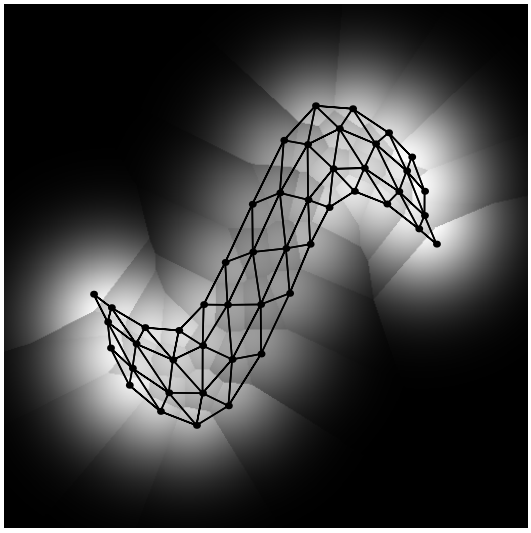

Figure 3.1: The generative density of a normal SOM. Observe the discontinuities at the Voronoi cell boundaries.

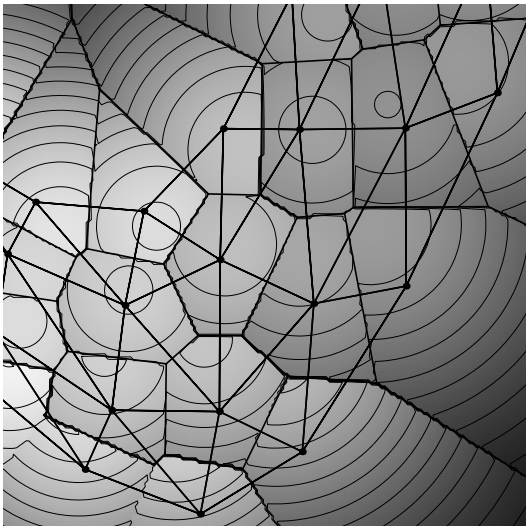

Figure 3.2: An excerpt of the above generative density with contours and boundaries of Voronoi cells added.

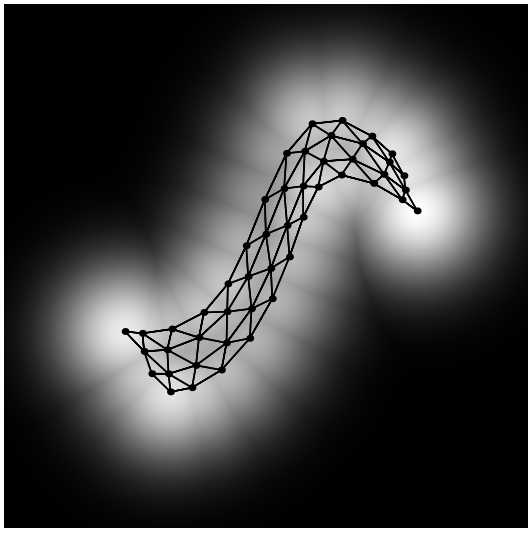

Figure 3.3: The generative density of a minimum distortion SOM. The points corresponding with map units are the neighborhood weighted means of the codebook vectors. This depiction however is not entirely correct as the error is not minimized with respect to any single point.

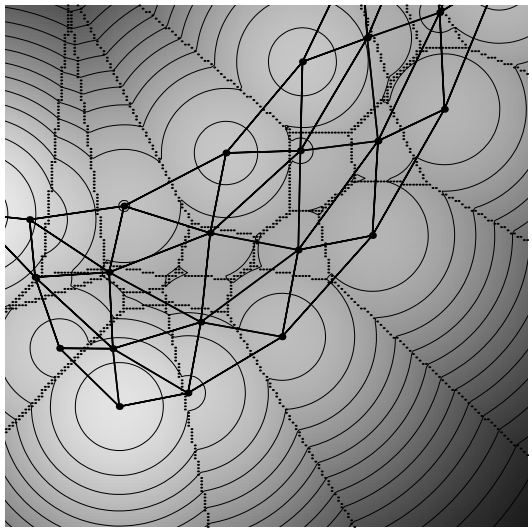

Figure 3.4: An excerpt of the above generative density with contours and boundaries of the receptive fields added. Notice how in some cases the means of the component densities are not included in the respective receptive field.

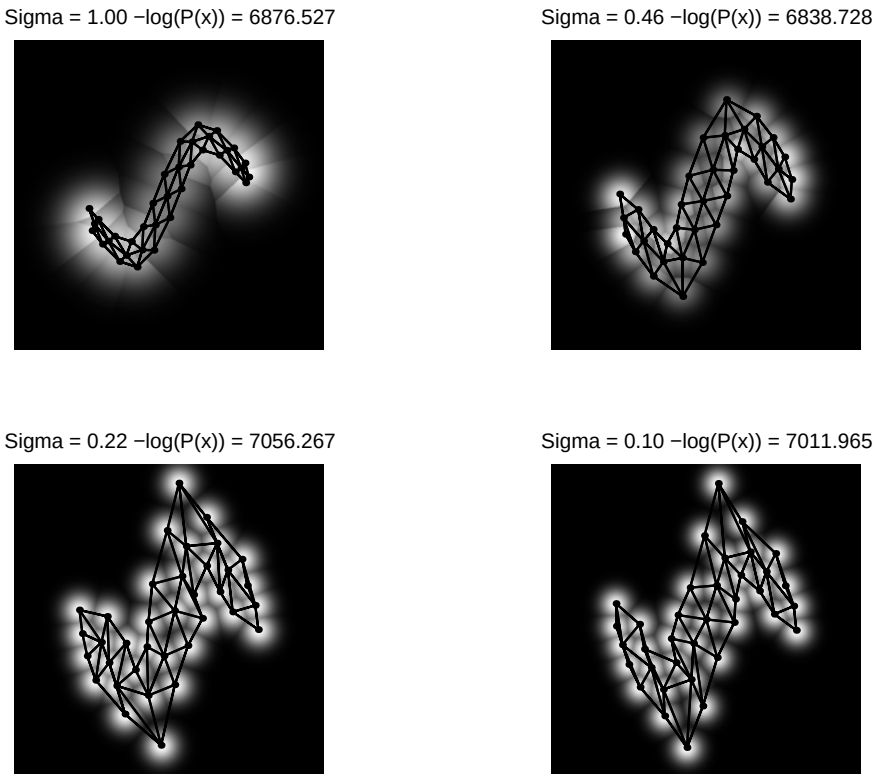

Figure 3.5: Model selection example with the normal SOM. The axis titles are the final neighborhood radius and the approximated negative log likelihoods for the test set.

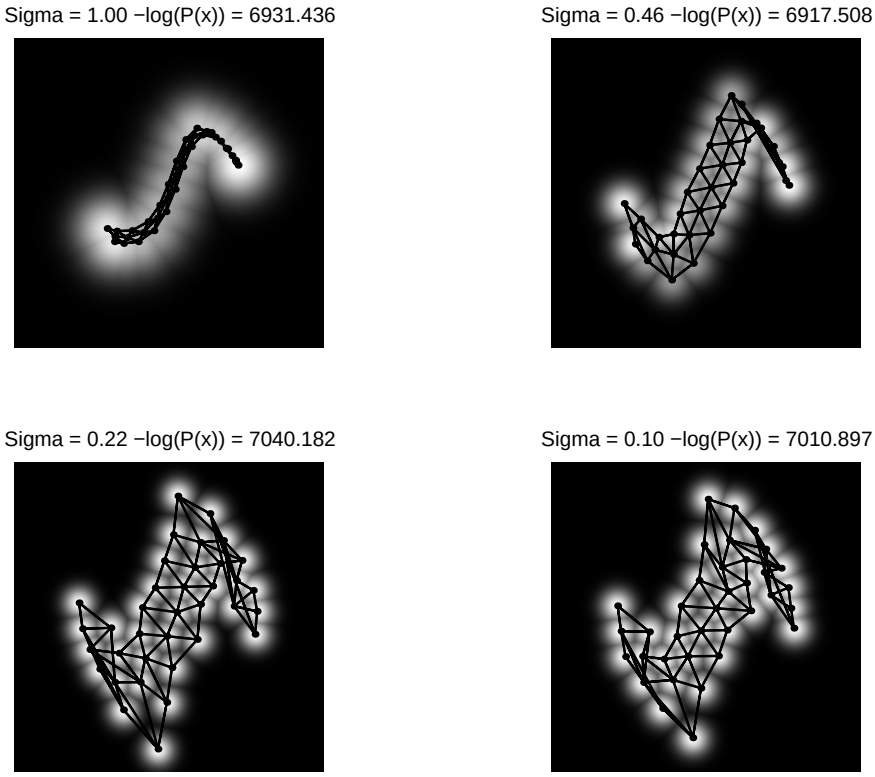

Figure 3.6: Model selection example with the minimum distortion rule SOM. The axis titles are the final neighborhood radius and the approximated negative log likelihoods for the test set.

Chapter 4

The Self-Organizing Map in Sequence Processing

The SOM was originally developed for independent identically distributed data. Thus the mapping is not influenced by the order of the input pattern presentation. However often in a real world situation data is collected as observations from sequential processes, where the assumption of independence with respect to time or previous observations is not always reasonable. Therefore a SOM like unsupervised network model for sequential data is an appealing prospect for data mining and comparable applications where the data, samples, are not associated with class labels or any other desired target value. In this section I will review and discuss various methods for bringing time or other possible forms of sequential dependence into the SOM.

4.1 Trace of Best Matching Units

The first widely used applications of the SOM in sequence processing used the trajectory of *bmu*s in the latent map space. The foremost application of these trajectories was the Phonetic Typewriter (Kohonen et al., 1988; Kohonen, 1988; Leinonen et al., 1992) for visualization of speech signals and their variations. The activity trajectories have also found applications in process control and monitoring (Tryba and Goser, 1991).

The *bmu* trajectories in the Phonetic Typewriter and similar applications do no sequence processing on their own right. Rather the SOM is used for reduction of dimensionality and thus acts as a pre-processing stage for subsequent processing. A potential weakness in the *bmu* trace method comes from the unavoidable folding of the map in high dimensional input manifold. If the true dimension of the data manifold matches the dimension of the map the map will organize nicely without unwanted folding but in higher dimensional manifolds folds are created on the

map to cover the entire manifold. As a consequence the trace signal will not preserve the properties of the original signal in the sense that changes in the trace are not a true reflection of the changes in the actual signal.

4.2 Single Layer Models

The use of *bmu* trace involves no changes to the SOM algorithm. The map is simply used to reduce the dimensionality of the data by projecting it onto latent map space of substantially lower dimension. If the mapping preserved the distance relations between subsequent samples the original signal could be substituted with the resulting trace for further processing. In general this however is not the case. In this section we will look into models that modify the basic SOM for sequence processing. These models have leaky integrators added to the units to give them a mechanism for remembering past activations or input patterns.

4.2.1 Temporal Kohonen Map

The goal of the Temporal Kohonen map (TKM) is to distinguish different sequences. To gain such time resolution leaky integrators, that gradually lose their activity, are added into the outputs of the otherwise normal competitive units of the SOM. In the TKM these integrators and consequently the decay of activation is modeled with the difference equation

$$U_i(n) = \lambda \cdot U_i(n) - \frac{1}{2}\|\mathbf{x}(n) - \mathbf{m}_i(n)\|^2 \,, \tag{4.1}$$

where $0 \leq \lambda < 1$ is a time constant, $U_i(n)$ is the activation of unit i at step n while $\mathbf{m}_i(n)$ is the codebook vector of unit i, and $\mathbf{x}(n)$ is the input pattern at step n. The formula in Eq. 4.1 preserves a trace of the past activations as their weighted sum. In fact it incorporates a linear low pass filter in the outputs of the otherwise normal competitive units.

The activity maximizing unit

$$b(n) = \arg\max_{i \in M} \; U_i(n) \tag{4.2}$$

is the *bmu* at step n in analogy with the normal SOM. The update rule for the TKM was not specifically addressed when the model was proposed (Chappell and Taylor, 1993). In the reported experiments, however, the codebook vectors were updated toward the last sample of the input sequence using the normal stochastic SOM update rule in Eq. 2.3

$$\mathbf{m}_i(n+1) = \mathbf{m}_i(n) + \gamma(n) h\left(b(n), i\right)\left(\mathbf{x}(n) - \mathbf{m}_i(n)\right) \,, \tag{4.3}$$

where $\gamma(n)$ is the learning rate and $h(b(n), i)$ is the neighborhood strength for unit i. The update rule in Eq. 4.3 corresponds to stochastic gradient descent for $\lambda = 0$, thus ignoring the leaky integrators at the outputs. The feasibility of the the TKM approach was demonstrated with a small synthetic and a small real world toy problem.

4.2.2 Recurrent Self-Organizing Map

In the recurrent self organizing map (RSOM) (Pub. 4,5 and 6) the leaked quantity is the difference vector instead of its squared norm. These integrators are modeled with

$$\mathbf{y}_i(n) = (1-\alpha)\mathbf{y}_i(n-1) + \alpha(\mathbf{x}(n) - \mathbf{m}_i(n)) \ , \tag{4.4}$$

where $\mathbf{y}_i(n)$ is the leaked difference vector for unit i at step n. The leaking coefficient α is analogous to the value of $1-\lambda$ in the TKM but in the RSOM formulation the sum of the factors is one to ensure stability when α is positive but less than one (Proakis and Manolakis, 1992). The RSOM formulation like the TKM formulation associates a linear low pass filter with each unit to preserve a trace of the past but in the RSOM the operator is moved from the unit outputs into their inputs. This seemingly small change is quite significant because it allows the definition of a simple SOM-like learning rule.

After moving the leaky integrators into the difference vector computation we can treat the map much like the normal SOM. Following the derivation of the learning rule for the normal SOM, first we define the *bmu* as

$$b(n) = \arg\min_{i \in M} \ \|\mathbf{y}_i(n)\| \ , \tag{4.5}$$

After choosing the *bmu* we define the corresponding error function we seek to minimize for input $\mathbf{x}$ in terms of the leaked difference vector $\mathbf{y}$ as

$$E(n) = \frac{1}{2} \sum_{i \in M} h\left(b(n), i\right) \mathbf{y}_i(n)^2 \ , \tag{4.6}$$

where the neighborhood $h(b(n), i)$ over which the error is computed was chosen with Eq. 4.5. Differentiating Eq. 4.6 with respect to $\mathbf{y}_i(n)$ yields

$$\frac{\partial E(n)}{\partial \mathbf{y}_i(n)} = h\left(b(n), i\right) y_i \ ,$$

thus the stochastic update rule following the normal SOM to minimize the error $E(n)$ is

$$\mathbf{m}_i(n+1) = \mathbf{m}_i(n) + \gamma(n) h\left(b(n), i\right) \mathbf{y}_i(n) \tag{4.7}$$

where $\gamma(n)$ is the learning rate. In this derivation we ignored the discontinuities of the error function $E(n)$ at the boundaries of the Voronoi cells.

Model	*bmu* selection criterion	Codebook update target
TKM	$\arg\max_{i \in M} \; U_i(n)\|_{\lambda=\hat{\lambda}}$	$\max U_i(n)\|_{\lambda=0}$
RSOM	$\arg\min_{i \in M} \; \|y_i(n)\| \;\|_{\alpha=\hat{\alpha}}$	$\min \; \|\mathbf{y}_i(n)\| \;\|_{\alpha=\hat{\alpha}}$

Table 4.1: The properties of the TKM and the RSOM. In the table $\hat{\lambda}$ and $\hat{\alpha}$ denote the intended values for the time delay parameters while λ and α denote the values that actually get used.

The key properties of the learning rules of the TKM and the RSOM models are summarized in Table 4.1. Note that the significant difference is that while the α in the RSOM is the same for both *bmu* selection and codebook update this is not true with the λ in the TKM.

4.2.3 Analytical Comparison of TKM and RSOM

In this section we will discuss the learning properties of the TKM and the RSOM models. First we derive the activity maximizing codebook vector for a set of sequences and a single unit for both the TKM and the RSOM. In 4.2.3 we look into the update rule of the TKM to see what the map actually learns and compare experimental results with RSOM results. This work was motivated by problems observed in the behavior of the TKM if the input space was even slightly more complicated than those the model was experimented with in (Chappell and Taylor, 1993).

Activity Maximizing Codebook

Brief mathematical analysis is sufficient to show how maximizing activity in the TKM should lead to similar codebook as minimizing the norm of the leaked difference vector in the RSOM when the maps share the same topology and data. For the analysis we consider a set $S = \{X_1, X_2, ..., X_{\Omega_S}\}$ of sequences. The samples in a sequence $X_j \in S$ are denoted $\mathbf{x}_j(1), \mathbf{x}_j(2), ..., \mathbf{x}_j(n_j)$, where n_j is the length of the sequence X_j.

Activity Maximizing Codebook for TKM

The goal of the TKM is to distinguish different sequences by choosing the unit best adapted for each sequence. The criterion for adaptation chosen in (Chappell and Taylor, 1993) is maximum activity on the map as defined in Eq. 4.2. Maximizing activities of the *bmu*s thus seem like a natural goal for the training procedure.

While this of course is not necessary since any learning target that produces the best possible result is equally good, assumption that the goal is to learn the activity maxiziming codebook is plausible.

For the set S of sequences and the codebook vector $\mathbf{m}$ of a TKM unit, the units activity $U(S, \mathbf{m}, \lambda)$ over S is the sum

$$U(S, \mathbf{m}, \lambda) \quad = \quad \frac{-1}{2} \sum_{j \in S} \sum_{k=1}^{n_j} \lambda^{(n_j - k)} \|\mathbf{x}_j(k) - \mathbf{m}\|^2 \,, \tag{4.8}$$

where $U(n)$ has been redefined in terms of the static input set S, the codebook vector of the single unit $\mathbf{m}$ and the time delay parameter λ.

Since the activity $U(S, \mathbf{m}, \lambda)$ is a parabola, it is everywhere continuous and differentiable with respect to $\mathbf{m}$. Consequently its maximum lies either at its extreme or at the single zero of $\frac{\partial U(S, \mathbf{m}, \lambda)}{\partial \mathbf{m}}$. Setting

$$\frac{\partial U(S, \mathbf{m}, \lambda)}{\partial \mathbf{m}} = 0$$

we obtain

$$\mathbf{m} = \frac{\sum_{j \in S} \sum_{k=1}^{n_j} \lambda^{(n_j - k)} \mathbf{x}_j(k)}{\sum_{j \in S} \sum_{k=1}^{n_j} \lambda^{(n_j - k)}} \,, \tag{4.9}$$

for the activity maximizing codebook for the single unit over the set S. Providing that all sequences in S have the same length $\hat{n}$, the inner sum of the denominator in Eq. 4.9 is constant allowing us to simplify the equation to

$$\mathbf{m} = \frac{1}{\Omega_S} \sum_{j \in S} \mathbf{m}_j$$

where Ω_S is the cardinality of S and $\mathbf{m}_j$ is the activity maximizing codebook vector for the sequence $X_j \in S$ defined with

$$\mathbf{m}_j = \frac{\sum_{k=1}^{\hat{n}} \lambda^{(\hat{n} - k)} \mathbf{x}_j(k)}{\sum_{k=1}^{\hat{n}} \lambda^{(\hat{n} - k)}} \,. \tag{4.10}$$

This codebook vector is the mean of the per sequence axtivity maximizing codebooks. It also is a good approximation when all sequences are sufficiently long for the chosen λ.

Optimal Codebook for RSOM

Now we repeat essentially the same steps for an RSOM unit to derive the corresponding optimal RSOM codebook vector for the set S. The leaked difference

$\mathbf{y}(X, \mathbf{m}, \alpha)$ of the unit, where $X = \mathbf{x}(1), ..., \mathbf{x}(n)$ is the input sequence, $\mathbf{m}$ is the codebook vector and α is the time constant, is

$$\mathbf{y}(X, \mathbf{m}, \alpha) = \alpha \sum_{k=1}^{n} (1-\alpha)^{(n-k)} (\mathbf{x}(k) - \mathbf{m}).$$

Since the goal is to minimize the norm of the leaked difference vector, for the set S we can rewrite the error function in terms of the the set S, the codebook vector $\mathbf{m}$ and the time delay parameter α as

$$E(S, \mathbf{m}, \alpha) = \sum_{j \in S} \|\mathbf{y}(X_j, \mathbf{m}, \alpha)\|^2 , \tag{4.11}$$

which is minimized at the optimal codebook vector. $E(S, \mathbf{m}, \alpha)$ defines a parabola just like $U(S, \mathbf{m}, \lambda)$ did for the TKM and thus the optimal codebook vector is either at an extreme or at the single zero of the derivative of the error function with respect to the codebook vector $\mathbf{m}$. After setting

$$\frac{\partial E(S, \mathbf{m}, \alpha)}{\partial \mathbf{m}} = 0$$

we obtain

$$\mathbf{m} = \frac{\sum_{j \in S} \left[\sum_{k=1}^{n_j} (1-\alpha)^{(n_j-k)} \sum_{k=1}^{n_j} (1-\alpha)^{(n_j-k)} \mathbf{x}_j(k) \right]}{\sum_{j \in S} \left[\sum_{k=1}^{n_j} (1-\alpha)^{(n_j-k)} \right]^2} . \tag{4.12}$$

The optimal RSOM codebook vectors in Eq. 4.12 is quite close to the codebook vector defined in Eq. 4.9 for the TKM. The small difference comes from the location of the leaky integrators.

Much like with the TKM we can simplify Eq. 4.12 if we assume that all sequences have the same length $\hat{n}$. We get

$$\mathbf{m} = \frac{1}{\Omega_S} \sum_{j \in S} \mathbf{m}_j ,$$

where $\mathbf{m}_j$ is the optimal codebook vector for sequence $X_j \in S$ defined with

$$\mathbf{m}_j = \frac{\sum_{k=1}^{\hat{n}} (1-\alpha)^{(\hat{n}-k)} \mathbf{x}_j(k)}{\sum_{k=1}^{\hat{n}} (1-\alpha)^{(\hat{n}-k)}} .$$

This codebook vector is identical with the corresponding TKM codebook when $\lambda = 1 - \alpha$. From the analysis we observe that the optimal codebook vectors for both models are linear combinations of the samples in the sequences.

Learning Algorithms

Since the update rule of the RSOM approximates gradient descent to minimize the sum of the squared norms of the leaked difference vectors regularized by the neighborhood, the map explicitly seeks to learn the codebook vector defined in the previous section. With the TKM this is not the situation: We show that generally the steady state codebook of the TKM does not maximize the activity and use simulations to show how this affects the behavior of the TKM. To simplify the analysis we only consider the zero neighborhood case.

By definition, in a steady state further training causes no changes in the codebook. In practice this means that the derivative of the objective function is zero with respect to the codebook given a static set of input patterns. Though in the stochastic training scheme reaching a steady state is not possible in finite time, criteria for a steady state can be defined and their impact considered when we study the equivalent batch approach. For the batch approach we split the TKM algorithm in two. In the first stage the data is Voronoi partitioned among the units with the network activity function. In the second stage the new codebook given the partitioning are computed. While proving convergence for any SOM model is very difficult (Erwin et al., 1992a; Cottrell, 1997, 1998), if the TKM converges its codebook has to satisfy the criteria we define here.

We have a set $S = \{X_1, ..., X_{\Omega_S}\}$ of discrete sequences and a map M. Last sample of each sequence $X_j \in S$ is $\mathbf{x}_j(n_j)$ where n_j is the length of the sequence. In a steady state the TKM weights have to be in the centroids of the last samples of the sequences in the Voronoi cells of the units. This observation is a direct consequence of the codebook update toward the last samples of the sequences corresponding with maximizing the activity when the time delay coefficient λ was set to zero. When we set $\lambda = 0$ the TKM activity for unit i in Eq. 4.8 reduces to

$$U(S_i, \mathbf{m}_i, 0) = \frac{-1}{2} \sum_{j \in S_i} (\mathbf{x}_j(n_j) - \mathbf{m}_i)^2$$

and the corresponding steady state codebook at $\frac{\partial U(S_i, \mathbf{m}_i, 0)}{\partial \mathbf{m}_j} = 0$ is

$$\mathbf{m}_i = \frac{1}{\Omega_{S_i}} \sum_{j \in S_i} \mathbf{x}_j(n_j) \ , \forall i \in M \ , \tag{4.13}$$

where $i \in M$ is the unit, $\mathbf{m}_i$ is the units codebook vector, $S_i \subset S$ is the set of sequences in the Voronoi cell of i and Ω_{S_i} is the cardinality of S_i. This codebook is necessary for a steady state as the update step would not vanish for any other codebook given the Voronoi partitioning that produced S_i:s. The activity maximizing TKM codebook with respect to the activity rule was defined in the

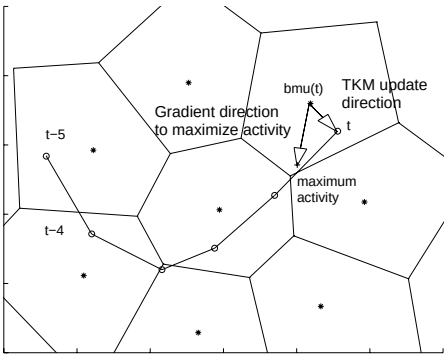

Figure 4.1: A piece of a TKM during training. The units, and their Voronoi cells, are marked with asterisks ($*$) and the input sequence with little circles ($\circ$). The plus (+) is drawn at the activity maximizing codebook vector. The arrows show the the activity maximizing and the actual TKM update directions.

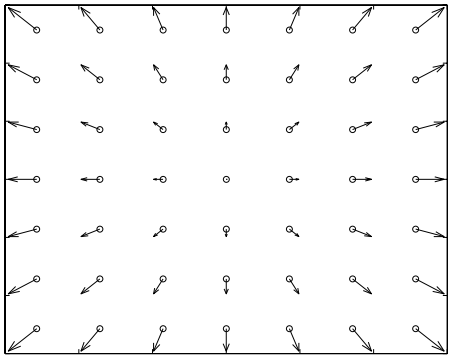

Figure 4.2: Approximation of the mean bias between the activity maximizing update directions and the TKM update directions for a regular 7×7 grid of input patterns in a 2D input manifold. We considered all sequences of length seven and computed the approximation for $\lambda = 0.15$ for the time delay.

previous section. The codebook

$$\mathbf{m}_i = \frac{\sum_{j \in S_i} \sum_{k=1}^{n_j} \lambda^{(n_j - k)} \mathbf{x}_j(k)}{\sum_{j \in S_i} \sum_{k=1}^{n_j} \lambda^{(n_j - k)}} \, , \forall i \in M \tag{4.14}$$

maximizes activity with our simplifying assumptions.

Experiments

The problem with the TKM is the discrepancy between the activity maximizing codebook and the necessary steady state codebook. Fig. 4.1, which has a part of a TKM during training, shows this graphically. The arrow "Gradient direction to maximize activity" shows the steepest descent direction to maximize activity while the arrow "TKM update direction" shows the actual update direction toward the last sample of the sequence.

We ran several simulations to demonstrate the impact of the discrepancy between the *bmu* selection and the weight update in the TKM. The first simulation involves a one dimensional map in a discrete one dimensional input space of seven input patterns. We initialized a 25 unit map with activity maximizing codebook (see axis 1 in Fig. 4.3) for inputs 1...7. The leaking coefficient λ was set at 0.1429, which leads to nearly regular activity maximizing distribution of the codebook into the input space [1, 7]. The map initilized with activity maximizing codebook was further trained by randomly picking one of the inputs, thus creating

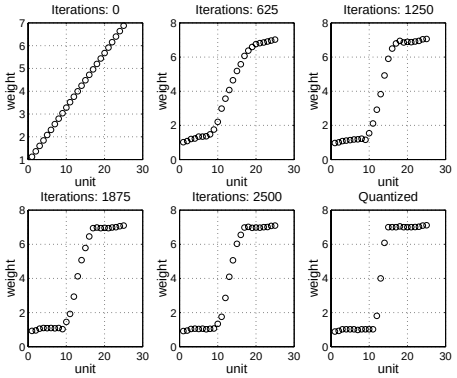

Figure 4.3: A map initialized with activity maximizing codebook and trained with the TKM approach. Notice how most of the units are drawn into the edges.

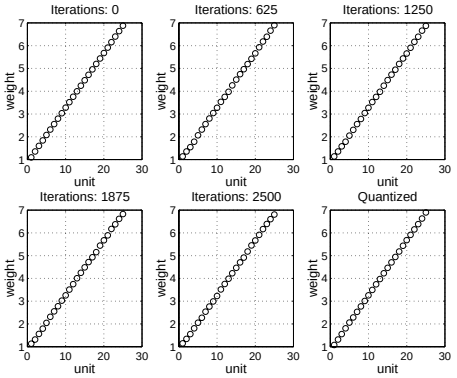

Figure 4.4: A map initialized with optimal codebook and trained with the RSOM approach.

a long random sequence, and updating the codebook using the stochastic training scheme. The samples of the random sequence was corrupted with additive Gaussian noise $\sim \mathcal{N}(0, 0.125)$.

Fig. 4.3 shows the progress of a sample run for the TKM. The TKM quickly "forgets" the initial codebook because it does not satisfy the steady state criteria we derived earlier. Notice how the units are drawn toward the extremes of the input manifold leaving only couple of units to cover bulk of the space. Identical one dimensional experiment with the RSOM in Fig. 4.4 yields a practically unchanged result.

We can intuitively explain the reason for the units being drawn toward the edges in the TKM with Figs. 4.1 and 4.2. For sequences that end near the edges of the input manifold the activity maximizing TKM codebook and consequently the *bmu*s are systematically closer to the center of the manifold than the last samples of the sequences which the units are updated toward. We can see this bias in Fig. 4.1 in the difference between the activity maximizing update direction and the actual update direction. This bias causes units to be attracted toward the edge and especially corner samples. Once a unit is close enough it will no longer be the *bmu* for any non trivial sequence of moving value.

Fig. 4.2 shows an approximation of mean bias between the activity maximizing update directions and the TKM update directions for a regular 7×7 grid of input patterns in a two dimensional input space. We considered all sequences of length seven and computed the approximation using $\lambda = 0.15$ for the time delay parameter. The bias is zero only at the center of the manifold and becomes larger the closer the input is to the edge. The lengths and the directions of the arrows show the relative magnitude and direction of the bias for the sequences ending at

that particular input. Formally

$$\mathbf{u}_j \approx \sum_{k \in S_j} \mathbf{x}_j - \mathbf{m}_{X_k}$$

where $\mathbf{u}_j$ is the arrow drawn at input location $\mathbf{x}_j$, S_j is the set of sequences the last sample of which is $\mathbf{x}_j$, X_k is a sequence in S_j and $\mathbf{m}_{X_k}$ is the activity maximizing TKM codebook vector for X_k. The arrows form what resembles a gradient field of smooth bump. The behavior of the TKM in two dimensional simulations strongly supports the intuitive result in the figure.

In two dimensional simulations we trained hundred TKM and RSOM maps to estimate the weight distributions in the input manifold with Gaussian kernels. The maps were trained with Luttrell's incremental approach (Luttrell, 1989). The maps were trained with random sequences by picking one of the possible input patterns and corrupting it slightly with additive Gaussian noise $\sim \mathcal{N}(0, 0.125)$. We experimented with two $2D$ input grids where the sparse grid had patterns only in its four corners. This simulation essentially repeats the original TKM experiment (Chappell and Taylor, 1993). The other grid was denser and had 49 input patterns arranged in a regular 7×7 grid.

For the sparse grid we trained maps with sixteen units arranged in four by four grid using $\lambda = 0.3$ for the time delay in the TKM and $\alpha = 0.7$ for the RSOM leaking parameter following the original experiment in Chappell and Taylor (1993). The resulting estimates for weight distributions in the input space are in Fig. 4.5 where lighter shade means higher probability of a codebook vector. The distributions are meaningful because the activity maximizing codebook vectors as derived in the previous section are linear combinations of the patterns in the input sequences. As a consequence the way the maps partition the input manifold directly reflect the way they partition the sequence space as well. When quantized with the zero neighborhood the, TKM concentrates all of its units at the four input patterns as expected from the update rule. When the neighborhood is not turned off the map forms a four by four grid of units where each unit is sensitive to one of the sixteen possible combinations of two input patterns. With the RSOM the result is not visibly dependent on the treatment of the neighborhood. The map creates a four by four grid of units which in the case when the neighborhood was retained was slightly denser.

The situation changed when we used the more densely sampled input grid and 100 unit maps units arranged in 10×10 grid. For this simulation we set the time delay factor at $\lambda = 0.15$ and $\alpha = 0.85$ accordingly. The resulting estimates for the codebook distributions are in Fig. 4.6. Again without the neighborhood the TKM concentrated all its units near the corner inputs reflecting the intuitive result in Fig. 4.2. With the neighborhood, more units were left to cover the core of the manifold as the neighborhood stiffens the map up but the improvement is not as significant as it was with the sparse grid. Increasing the the radius of the

neighborhood made the phenomenon more pronounced. The properties of the model are such that the conflicting activity and update rules force units toward the corners of the manifold but stiffening the map up with strong neighborhood partially counters this effect.

Now recall the activity maximizing codebook we derived for TKM and RSOM in Eqs. 4.9 and 4.12 respectively. TKM concentrated most of its units in the edges and the corners of the manifold leaving only a few units to cover the bulk when all input patterns were not in the corners of the manifold. In the sparse manifold the conflicting activity and update rules were countered with the neighborhood but the same neighborhood radius did not help with the densely sampled manifold. Increasing the neighborhood radius would help in a simple manifold but this approach could not be used in more complicated manifolds since the large neighborhood radius would not allow the map to follow the manifold. In our opinion using the neighborhood to correct the inherent problem in the model design is not the correct approach. In these simulations the TKM model wasted a considerable part of it expressive power. The RSOM on the contrary systematically learned codebook that nearly optimally spanned the input manifold. The problem with the TKM could be resolved if the TKM was trained with a rule that did not require gradient information. The chances are, however, that such a rule would be computationally very demanding because it would require repeated evaluations of the target function.

4.3 Hierarchical Models

The models presented in the previous section employed a single SOM coupled with an integrating memory. Both of these models are limited to discovering linear dependencies in the data and as it turned out the RSOM model is effectively a basic SOM coupled with a linear low pass operator. In this section we will look into models that employ two maps and an integrating memory between them.

In these models the outputs of the first map are stored in an integrating memory to provide the input for the second map. The first SOM produces a nonlinear transformation of the input data. The transformation is stored in in the integrating memory to subsequently serve as the input for the second map. When broken down they essentially have an RSOM in the bottom with a basic SOM on the top.

4.3.1 The Model of Kangas

A two layer model with an integrating memory connecting the maps was proposed by Kangas (1994). The integrating memory between the maps is realized with leaky integrators like in the TKM model. The transformation values y_i stored in the memory are inverted normalized Euclidean distances between data and code-

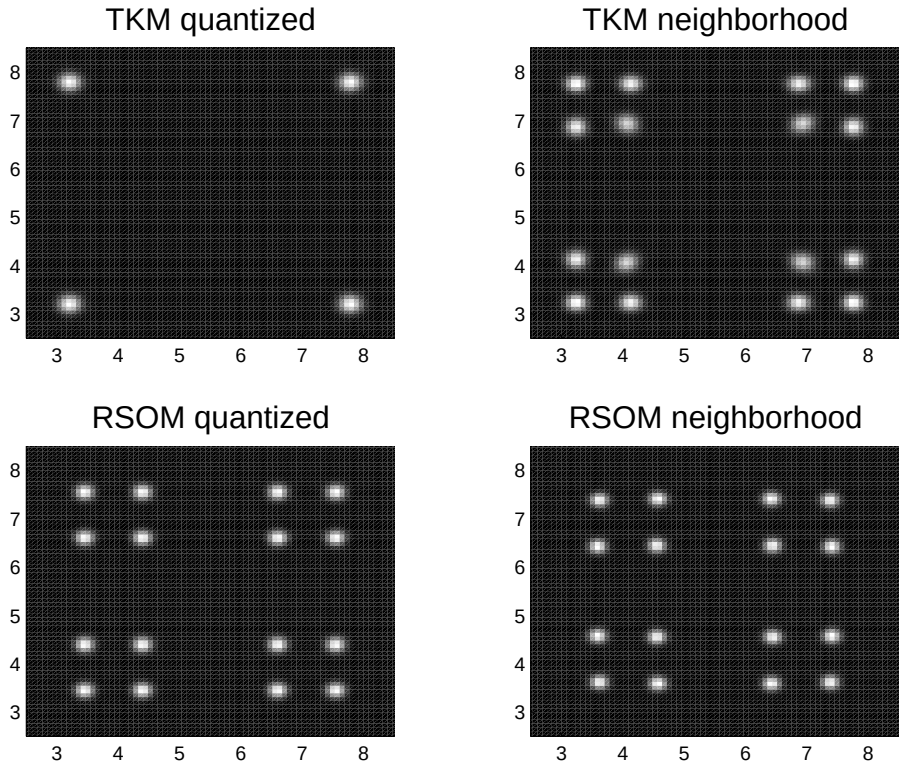

Figure 4.5: Kernel density estimates for codebook distributions of 4×4 TKM and RSOM maps in two dimensional input space with four input patterns in the corners of a square. The figure depicts the distribution of the codebook, estimated from hundred independent runs, into the square delimited by the four input patterns. Activity maximizing distribution is a four by four grid, where each of the sixteen locations encodes one of the sixteen possible combinations of two out of four input patterns. Lighter shade signifies higher probability.

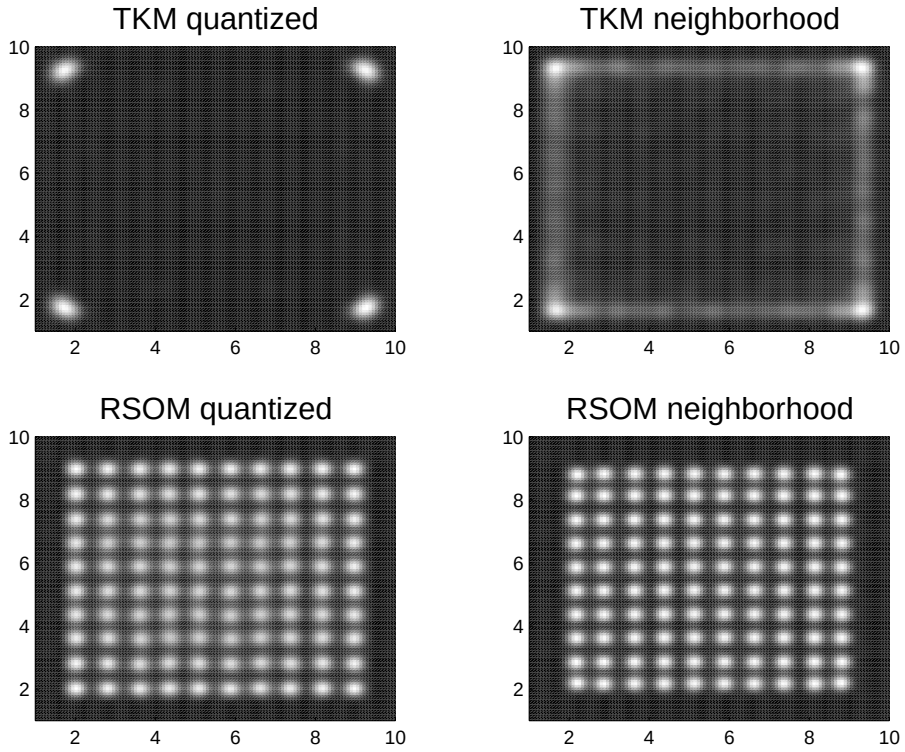

Figure 4.6: Kernel density estimates for codebook distributions of 10×10 TKM and RSOM maps in two dimensional manifold, sampled at 49 points on a regular 7×7 grid. The activity maximizing distribution of codebook vectors is a regular 10×10 grid.

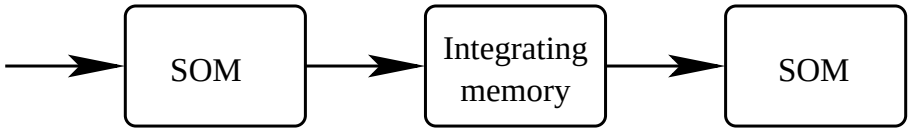

Figure 4.7: A hierarchical model employing two maps and an integrating memory (Kangas, 1994). The first map performs a non linear transformation on the data. The transformations are stored in integrating memory and serve as the input for the second map.

book vectors such that the *bmu* gets 1.0 and the other units smaller values accordingly, thus for y_i we can write

$$y_i = \left[\frac{\|\mathbf{x} - \mathbf{m}_i\|}{\|\mathbf{x} - \mathbf{m}_{b(\mathbf{x})}\|} \right]^{-1}.$$

The obtained activity values are sharpened by squaring them a few times. This dampens the small values and leaves only a few large values for more pronounced activity map for the second layer. The transformation amounts to a slightly softened winner take all encoding to account for a situations where the sample is equally close to two units far apart on the map. Such a situation may arise from folding of the map space. Normal winner take all transformation would lead to situations where the activity moves rapidly across the map space when in fact the original data stays almost constant. This is the same phenomena that can plaque the use of *bmu* trajectories.

4.3.2 The Model of Carpinteiro

The other hierarchical model employing two SOM associates an integrating memory with both maps (Carpinteiro, 1998, 1999). This model model bears a lot of resemblance with the model of Kangas and turns out to be close to two RSOMs connected sequentially with a non linear transformation before the low pass operator of the second map. The model has been proposed in at least two slightly differing flavors.

In one flavor the transformation before the second map is computed in the input space much like the transformation in the model of Kangas but instead of using the distances akin to negative log likelihoods the model considers the actual likelihood values (Carpinteiro, 1999) written as

$$y_i = \exp\left[\frac{-(\mathbf{x} - \mathbf{m}_i)^2}{2\sigma^2} \right], \tag{4.15}$$

where y_i is the transformed value for unit i and σ^2 is the variance of the Gaussian placed over each unit. Effectively the first map is treated as an un-normalized

reduced kernel density estimator (Holmström and Hämäläinen, 1993; Hämäläinen, 1995), where the codebook vectors specify the locations of the component densities. The idea is basically sound but in practice unlikely to work. The first problem stems from the curse of dimensionality. If the real dimension of the input manifold $\mathcal{X}$ and thus $\mathbf{x}$ is relatively large the chances that $\mathbf{x}$ is close to any $\mathbf{m}_j$ are rather small, and thus unnormalized component likelihoods will either tend to zero or be very close to one depending on the selection of the variance parameter σ^2. The squaring of the transformation values in the model of Kangas reduces the effects of similar problem. The other potential problem lies in the lack of normalization of the transformation. It would be beneficial if the values added up to unity as then the underlying map could be interpreted as a Gaussian mixture with equally weighted kernels generating the data $\mathbf{x}$.

In the other flavor the transformation function is computed primarily in the latent map space rather than the input space (Carpinteiro, 1998). The transformation is defined as

$$y_i = \begin{cases} 1 - \gamma d_M(b(\mathbf{x}), i) & \text{for } H(\mathbf{x}, i) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

where $d_M(i, j)$ is a distance measure in the latent map space, y_i is the transformed output for i and γ is a constant determining the rate of decay of y_i with increasing map distance to the *bmu*. As it turns out the second transformation is an alternate definition for the neighborhood function to replace the original neighborhood and as such it forfeits all information from the input data other than the map coordinates of the corresponding *bmu*.

4.4 Operator Maps

The original SOM employed Euclidean distance in the unit competition but in the greater scheme the SOM can be regarded as a framework where competing models of varying kind are connected in a SOM like structure. This notion gave rise to the idea of the operator map (Lampinen and Oja, 1989; Kohonen, 1993; Joutsensalo and Miettinen, 1995).

4.4.1 Self-Organizing Map of Auto Regressive Units.

Future values of an autoregressive (AR) process depend on its previous values and i.i.d additive innovations. In other words the future states of such process are defined by its previous states plus a random noise component. This property is captured by an AR-model, which in the linear case is simply a weighted sum of the previous values of the process plus noise. The linear AR model for a discrete

process or a scalar signal $x(n)$ is then

$$x(n+1) = \sum_{k=1}^{N} m(k)x(n-k) + \epsilon(n) \ , \tag{4.17}$$

where N is the degree of the model and $\epsilon(n)$ is the noise term. The N vector of coefficients $\mathbf{m}$ can be solved, for example, with least squares method corresponding with Gaussian innovations.

In the Self-Organizing Map of competing auto regressive models (AR-SOM) (Lampinen and Oja, 1989) the codebook vectors of the units are the coefficients of the corresponding AR-models and the noise model was assumed Gaussian. With Gaussian innovations the optimal coefficient vector for the AR-model is estimated from the observations writing down the estimation errors as

$$\epsilon(n) = x(n+1) - \sum_{k=1}^{N} m(k)x(n-k) \tag{4.18}$$

and solving the corresponding linear system in the least mean square sense. This system of equations is called the Yule Walker equations.

In the AR-SOM adaptive least mean squares *alms* method, which is a form of the Widrow-Hoff rule (Widrow and Stearns, 1985; Shynk, 1995), was adapted. After every input sample a codebook vector is updated with

$$\mathbf{m}(n) = \mathbf{m}(n-1) + \gamma\epsilon(n)\mathbf{x}(n) \tag{4.19}$$

where $\mathbf{x}(n)$ is the vector of N previous samples $x(n-N-1)...x(n-1)$ and $\epsilon(n)$ is the error estimated with Eq. 4.18, γ is a learning rate factor. With the update rule defined in Eq. 4.19 we can define a stochastic learning algorithm Alg. 3. for the AR-SOM. The learning rule sketched above is basically sound but falls victim to the noise innovations. For the optimal codebook vector of AR-parameters the time average of the estimation error is minimum but at any given instant there is a large probability of a wrong unit giving the minimum error. Thus to measure the matching goodness of an unit i an exponential average over the recent estimations errors is computed with a leaky integrator as

$$\hat{\epsilon}_i(n) = \alpha\hat{\epsilon}_i(n-1) + (1-\alpha)\epsilon_i(n),$$

where α is a time delay parameter. With this modification the competition rule is changed to

$$b(n) = \arg\min_{i \in M} \ \hat{\epsilon}_i(n) \ .$$

Algorithm 3: Stochtastic learning algorithm for the AR-SOM.

1. For each input sample search the corresponding *bmu* with

$$b(n) = \arg\min_{i \in M} \; x(n+1) - \sum_{k=1}^{N} m_i(k)x(n-k)$$

2. Use Eq. 4.19 modified to accommodate for the neighborhood to update the AR-coefficients of the *bmu*.

$$\mathbf{m}_i(n) = \mathbf{m}_i(n-1) + \gamma h(b(n), i)\epsilon_i(n)\mathbf{x}(n)$$

4.4.2 Self-Organizing Map with Dynamic Time Warping

Dynamic programming is a standard algorithm for computing distances of words from a discrete alphabet by computing the minimum cost for translating one word to the other. In dynamic programming a cost is associated with each possible operation which are removing a symbol, inserting a symbol or replacing a symbol with another from a limited alphabet. Dynamic time warping DTW is a dynamic programming algorithm, where a sequence of observations is usually compared with a reference or template sequence. Associating such reference sequences with the units of a SOM and using DTW for competition gives rise to a model dubbed DTW-SOM (Somervuo and Kohonen, 1999). In experiments the model was used to cluster Finnish utterances of numerals zero through nine.

In the DTW sequences of feature vectors are compared along warping function which essentially tells how the the compared sequences are aligned in comparison. The warping path $\mathbf{P}$ can be described as a sparse $N \times M$ matrix, where N and M are the lengths of the compared sequences. Each element of the matrix $p_{i,j}$ tells how much the distance $d(x(i), m(j))$, where $x(i)$ is the i:th sample of the compared sequence and $m(j)$ is the j:th element in the reference sequence influences the overall distance between the sequences along the path. Since $\mathbf{P}$ describes the alignment of the two sequences it is reasonable to restrict $\mathbf{P}$ to having exactly $\min(N, M)$ non zero elements with the $\sum_{\mathbf{P}} p_{i,j}$ being a constant. A natural choice for all nonzero elements of $\mathbf{P}$ is one but other alternatives may arise when matching certain parts of the sequences is more important than the other for example. The minimal warping path satisfies the following condition

$$D_{min} \equiv \sum_{\mathbf{P}_{min}} p_{i,j} d(x(i), m(j)) = \min_{\mathbf{P}} \sum_{\mathbf{P}} p_{n,m} d(x(i), m(j)) \tag{4.20}$$

where the distance measure $d(\cdot, \cdot)$ is application dependent.

Since it is assumed a priori that the samples in the sequences are ordered in time we can set further requirements on the warping path **P**. A natural condition is that the path must be continuous and cannot go backward in time, it can however stall. Since time marches on slope constraints can also be imposed on the path forcing the time to advance on both sequences at some minimal rate. A potentially coherent way to account for all prior knowledge would be through a suitable prior distribution for the **P**. In the case of speech recognition for example a conditional Gaussian ridge, where the ridge ran along the sub diagonal of current location, would would be an intuitive choice i.e a prior that favored advancing on both sequences at constant rate. The flexibility of the DTW matching as it allows comparing sequences of different length makes it appropriate for speech processing because people do not speak at the same constant rate. The method provides invariance against moderate speed variations within the limits allowed for warping path **P**. In speech processing for which is was intended the DTW-SOM is more suitable for a rather limited vocabulary as each word requires its own prototype sequence. The state map discussed in the following section addresses this problem by using the map as a Hidden Markov Model, which generates state sequences for further processing.

4.4.3 State Space Map and Hidden Markov Models

Markov chain is a realization of random variable future values of which can be predicted from its previous values. Most common Markov process, Markov one, only requires the current value to predict the next one at best possible accuracy. From now on only such Markov one processes are considered. Quite frequently it is stated that in a Markov one process the next sample depends only on its immediate predecessor but strictly speaking this is not the truth. Rather the correct formulation for the dependencies in a Markov process is

$$p(x(n)|x(n-1), x(n-2), ..., x(1)) = p(x(n)|x(n-1)),$$

where $x(n)$ denotes the state of process and n denotes discrete time, thus conditioning on the immediate predecessor is sufficient to capture all relevant information from the past but earlier samples still have an indirect influence through the immediate predecessor. A Markov model that realizes a Markov process can be described by an M vector $\boldsymbol{\pi}$ of start state probabilities and an $M \times M$ transition matrix **A** that describes the state transition probabilities in other words **A** is the probability distribution function for the process state.

In a hidden Markov model (HMM) the state of the process is not directly observable but the process emits a character or a value y at every state transition, thus the HMM model is described as a triplet ($\boldsymbol{\pi}$, **A**, **B**), where $\boldsymbol{\pi}$ and **A** have their previous meanings and **B** captures the emission probabilities. For a discrete set of N possible outputs **B** is an $N \times M$ probability distribution matrix. For continuous

random output **B** is probability density $p_{y,x}(y|x(n))$ for the output character y conditioned on the current state of the process $x(n)$.

The state space map (Somervuo, 2000a) replaces the Euclidean distance operators in the units with competing states or several state sub models of a larger HMM. Each state i is modeled with a kernel density estimator of K_i Gaussian kernels, which can be initialized by training a large normal SOM and splitting it into clusters of several units. The model, designed primarily for speech processing, finds the most likely state sequence through the state space map using Viterbi algorithm for maximum likelihood estimate on the hidden state sequence on the HMM for the given observation sequence (Viterbi, 1967). The segmented observation sequence is then used to train the emission probability distributions of the corresponding *bmu*:s and their neighbors on the space state map.

The emission probability density for state i of K_i component densities is modeled as a Gaussian mixture as follows

$$\begin{aligned} p_{\mathbf{y},i}(\mathbf{y}|i) &= \sum_{k=1}^{K_i} c_{i,k}\, p_{i,k}(\mathbf{y}|i) \\ &= \sum_{k=1}^{K_i} c_{i,k}\, Z_{i,k} \exp\left[-0.5 \cdot (\mathbf{y}-\mathbf{m}_{i,k})^T \Sigma_{i,k}^{-1} (\mathbf{y}-\mathbf{m}_{i,k})\right], \quad (4.21) \end{aligned}$$

where $\mathbf{m}_{i,k}$:s are the expected outputs of the component densities with corresponding covariance matrix $\Sigma_{i,k}$, and $Z_{i,k}$:s are the normalizing factors, for the component densities. The sum of component weighting coefficients $c_{i,k}$:s is one to produce a probability density. The parameters of the emission densities in the units can be trained with the expectation maximization (EM) algorithm (Moon, 1996). The state transition matrix **A** was kept uniform until very late in the training process of the model to prevent a small subset of available states from becoming dominating early in the training.

The state map cannot strictly be regarded as a SOM of competing HMMs since there is no unique winner for an observation sequence. Rather the model is hybrid where latent space of the SOM is used primarily to distribute and share data among the states for computing the emission distributions and the units on the map are the states of the HMM. In a subsequent extension to the model (Somervuo, 2000a) the initial map space of the model was replaced with a time dependent one (Somervuo, 1999). This extension speeds up the training process by explicitly connecting the states that are more likely to occur sequentially in the training data instead of connecting states with similar emission probabilities.

The use of Gaussian mixtures in the units prompted the comparison of the model with the generative topographic mapping (GTM) (Bishop et al., 1996, 1997) which is a SOM like model developed using probabilistic principles treating the model as a data generating mechanism. The state map also bears resemblance to the *bmu* trace idea of using SOM in the processing of sequential data.

The outcome of the model is a state sequence in the latent map space which is essentially a *bmu* trace. An actual map of competing HMMs would be closely related to the dynamic time warping map. A drawback of such model would be size as at least one HMM and unit would be required for each word in the vocabulary. The state space map as defined does not provide immediate classification but it reduces the dimension of the original feature data significantly to simplify subsequent classification.

4.5 Conclusions

In this thesis we discussed applying SOMs in sequence processing. The focus was in the applications and alterations of the SOM that detect sequential dependencies in the data. However, the RoboSOM proposal (Pub. 1,2) inverts the situation by using the SOM to establish temporal dependence.

The SOM models for sequence processing reviewed in chapter four can roughly be split in three groups. In the first group are the *bmu* trace method and the SOM of competing HMMs. Both of these models generate a state sequence, or a sequence of state probability distributions, which can be used as input to further processing. In the second group, formed by the AR-SOM, TKM, RSOM and DTW-SOM, the operators in the units are altered for sequential input. In these models the SOM structure is primarily used to share data among competing units. In other words the sequence space is quantized by the SOM and the quantization process is regularized by the neighborhood. The performance of these models is limited by the operators in the units. The SOM structure does not enrich the representations, rather the opposite, but this may be used to avoid overfitting into the data. The third group includes the two hierarchical models which are hybrids of the first two groups. The first map creates a state sequence which is integrated and serves as the input to the second map.

The application of the SOM for surface manipulating robots RoboSOM makes explicit use of the maps embedding property. The proposed method exploits the prior knowledge that the data is sampled from a two dimensional surface and thus the SOM can be expected to organize nicely on this surface as in Fig. 2.3. As a consequence the map neighborhood is used to create sequential order into the data. This order is then used to create a path for the robot. In a sense the RoboSOM is an inverse sequence processing method, where the neighborhood of the map creates the time or sequential dependence into the data thus extending the neighborhood in time. The applications of SOMs in NP-complete problems such as traveling sales person and Hamiltonian circuit (see for example Aras et al. (1999)) in a way utilize the neighborhood in the same manner we did in the RoboSOM proposal.

Chapter 5

Summary of Publications

5.1 Publication 1: Surface Modeling and Robot Path Generation Using Self-Organization

In publication one we considered the problem of creating a program path for a surface manipulating arm robot. The idea was to use the SOM to quantize a two dimensional manifold, surface of an object, folded through third dimension. The natural ordering of units that emerges on the map is taken advantage of when decomposing the map into a sequence of points comprising the path for the robot. The idea was simple enough: Use the SOM as a flexible surface fitted over the object to be manipulated. However several problems surfaced that required modifications to the basic SOM algorithm. These problems included varying curvature of the work piece surface requiring varying neighborhood width to enable closer matching to the actual surface. Another problem was the conflicting surface shape. A rectangular map lattice was being fitted over a non rectangular surface which created folds on the map especially near areas with high curvature.

The authors contribution in this work in addition to coming up with the idea was modifying the Tree Structured SOM (TS-SOM) for the surface modeling task as well as generating the test data and running all simulations. The text was co-authored with Prof. Pasi Koikkalainen.

5.2 Publication 2: Robot Path Generation for Surface Processing Applications via Neural Networks

The ideas we presented in publication one were refined in publication two to resolve the problems from fitting a rectangular map over non rectangular surface of the work piece. To achieve this the two dimensional surface was split into narrow essentially one dimensional strips and a one dimensional map was trained for

each of these strips. This created a hierarchical structure: The underlying two dimensional map was used to create a coarse path for the arm of the robot and the one dimensional maps, one for each row of the two dimensional map, were used to create more detailed tool orientation paths to get as even as possible coverage for the surface. This solution combated the surface shape problem as it allowed different number of path points for each strip and thus broke the rectangular shape restriction of the original idea.

The authors contribution in this work was to further modify the Tree Structured SOM (TS-SOM) as hierarchical model where the two dimensional map was split into one dimensional maps for the bottom layer. The author also generated the simulation data and ran all tests. The manuscript was co-authored with Prof. Pasi Koikkalainen.

5.3 Publication 3: Epileptic Activity Detection in EEG with Neural Networks

In publication three the SOM was used in more traditional context for feature quantization and normalization. Several feature sets were evaluated for detection of epileptic spikes in electroencephalography (EEG) recordings previously scored by neurologists. The normalization of the feature values simplifies the subsequent classifier but the real gain from the SOM was the improved tolerance for out liars. Like vector quantizers in general the map concentrates its units where the data is concentrated providing for more detailed encoding for higher density areas and the ordering of the units means that encodings of close values are also close.

For this work the author wrote the code for feature extraction and quantifying with the SOM. The classification with feed forward networks was done in cooperation with Jukka Heikkonen. The manuscript was co-authored with Dr.Tech. Jukka Heikkonen and Ph.D. José del R. Millán.

5.4 Publication 4: A Recurrent Self-Organizing Map for Temporal Sequence Processing

The Recurrent Self-Organizing Map (RSOM) proposed in publication four (also covered in Varsta et al. (1997)) is a modification of the Temporal Kohonen Map (TKM) (Chappell and Taylor, 1993). The RSOM approach remedied some undesired properties of its predecessor but at this stage the reasons were not quite clear. We knew that moving the leaky integrators from the response to the input side was the key but some analysis work was required to understand exactly why.

The author is solely responsible for the original idea for the RSOM inspired by the TKM and wrote the code. The initial experiments were performed in co-

operation by the author and Ph.D. José del R. Millán. The former were joined by Dr.Tech Jukka Heikkonen to finalize the manuscript.

5.5 Publication 5: Temporal Sequence Processing using Recurrent SOM

The RSOM model (Pub. 4) was used here for temporal sequence processing. In time series prediction the RSOM was used to partition the signal space and each part was associated with a local linear model (see also Koskela et al. (1998a,b)). This publication also has the first attempts to explain the why behavior or RSOM differs significantly from that of TKM.

The author wrote the code for the RSOM and ran the experiments involving the model. The authors contribution in the manuscript was writing the parts pertaining to the RSOM and the TKM.

5.6 Publication 6: Temporal Kohonen Map and Recurrent Self-Organizing Map: Analytical and Experimental Comparison.

In Publication six we presented the results of analysis to explain why the RSOM and the TKM while seemingly similar behave quite differently, the same problem was also considered in Varsta et al. (1998, 2000). The analysis revealed that the training rule of the TKM was ill formed causing the model to misbehave in any but the simplest of cases. It turned out that the RSOM is the correct way to construct a model with the properties originally desired from the TKM but it also showed how limited the ability of both of these models is in sequence processing for which they were intended.

The author wrote the paper, designed and performed the simulations for the analysis pushed in the right direction primarily by Prof. Jouko Lampinen and Dr.Tech. Jukka Heikkonen.

References

Aras, N., Oommen, B. J., and Altinel, I. K. (1999). Kohonen network incorporating explicit statistics and its application to the travelling salesman problem. *Neural Networks*, 12(9):1273–1284.

Bauer, H.-U., Pawelzik, K., and Geisel, T. (1992). A topographic product for the optimization of self-organizing feature maps. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pp. 1141–1147. Morgan Kaufmann, San Mateo, CA.

Bazaraa, M. S., Sherali, H. D., and Shetty, C. (1993). *Nonlinear Programming Theory and Algorithms*. Wiley, 2 edition.

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

Bishop, C. M., Hinton, G., and Strachan, I. G. (1997). Gtm through tme. In *Proceedings IEE Fifth International Conference on Artificial Neural Networks, Cambridge, U.K.*

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1996). GTM: A principled alternative to the self-organizing map. Technical Report NCRG/96/015, Neural Computing Research Group, Aston University.

Carpinteiro, O. A. S. (1998). A hierarchical self-organizing map model for sequence recognition. In *Proceedings of ICANN'98*, pp. 816–820.

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

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

Cottrell, M. (1997). Theoretical aspects of the som algorithm. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps*, pp. 246–267. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland. ISBN 951-22-3589-7.

Cottrell, M. (1998). Theoretical aspects of the som algorithm. *Neurocomputing*, 21:119–138.

Cottrell, M., Fort, J. C., and Pagès, G. (1994). Two or three things that we know about the Kohonen algorithm. In Verleysen, M., editor, *Proc. ESANN'94, European Symp. on Artificial Neural Networks*, pp. 235–244, Brussels, Belgium. D facto conference services.

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, Amsterdam.

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

Erwin, E., Obermeyer, K., and Schulten, K. (1992a). Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybenetics*, (67):47–55.

Erwin, E., Obermeyer, K., and Schulten, K. (1992b). Self-organizing maps: stationary states, metastability and convergence rate. *Biological Cybrenetics*, 67:35–45.

Flanagan, J. A. (2001). Self-organization in the one-dimensional SOM with a decreasing neighborhood. *NEURAL NETWORKS*, 14(10):1405–1417.

Goodhill, G. J. and Sejnowski, T. J. (1997). A unifying objective function for topographic mappings. *Neural Computation*, 9:1291–1303.

Graepel, T., Burger, M., and Obermayer, K. (1997). Deterministic annealing for topographic vector quantization and self-organizing maps. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pp. 345–350. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland.

Hämäläinen, A. (1995). *Self-Organizing Map and Reduced Kernel Density Estimation*. PhD thesis, Jyväskylä University, Jyväskylä, Finland.

Haykin, S. (1998). *Neural Networks - A Comprehensive Foundation (2nd. ed.)*. Prentice-Hall, Upper Saddle River, NJ.

Holmström, L. and Hämäläinen, A. (1993). The self-organizing reduced kernel density estimator. In *Proc. ICNN'93, International Conference on Neural Networks*, volume I, pp. 417–421, Piscataway, NJ. IEEE Service Center.

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

Joutsensalo, J. and Miettinen, A. (1995). Self-organizing operator map for nonlinear dimension reduction. In *Proc. ICNN'95, IEEE International Conference on Neural Networks*, volume I, pp. 111–114, Piscataway, NJ. IEEE Service Center.

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

Kaski, S., Honkela, T., Lagus, K., and Kohonen, T. (1998). WEBSOM—self-organizing maps of document collections. *Neurocomputing*, 21(1):101–117.

Kaski, S. and Lagus, K. (1996). Comparing self-organizing maps. In von der Malsburg, C., von Seelen, W., Vorbruggen, J. C., and Sendhoff, B., editors, *Artificial Neural Networks—ICANN 96. 1996 International Conference Proceedings*, pp. 809–14. Springer-Verlag, Berlin, Germany.

Kohonen, T. (1982). Self-organizing formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69.

Kohonen, T. (1984). *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg. 3rd ed. 1989.

Kohonen, T. (1988). The 'neural' phonetic typewriter. *Computer*, 21(3):11–22.

Kohonen, T. (1993). Things you haven't heard about the Self-Organizing Map. In *Proceedings of the ICNN'93, International Conference on Neural Networks*, pp. 1147–1156, Piscataway, NJ. IEEE, IEEE Service Center.

Kohonen, T. (1997). *Self-Organizing Maps*, volume 30 of *Lecture Notes in Information Sciences*. Springer, second edition.

Kohonen, T., Torkkola, K., Shozakai, M., Kangas, J., and Ventä, O. (1988). Phonetic typewriter for Finnish and Japanese. In *Proc. ICASSP-88, International Conference on Acoustics, Speech, and Signal Processing*, pp. 607–610, Piscataway, NJ. IEEE, IEEE Service Center.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998a). Prediction using rsom with local linear models. *Int. Journal of Knowledge-Based Intelligent Engineering Systems*, 2(1):60–68.

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998b). Recurrent som with local linear models in time series prediction. In *ESANN'98 European Symposium on Artificial Neural Networks*.

Laaksonen, J., Koskela, M., Laakso, S., and Oja, E. (2000). PicSOM—content-based image retrieval with self-organizing maps. *Pattern Recognition Letters*, 21(13–14):1199–1207.

Lampinen, J. and Kostiainen, T. (2001). *Self Organizing Neural Networks: Recent Advances and applications*, chapter Generative Probability Density Model in the SOM, pp. 75–94. Physica-Verlag.

Lampinen, J. and Oja, E. (1989). Self-organizing maps for spatial and temporal AR models. In Pietikäinen, M. and Röning, J., editors, *Proc. 6 SCIA, Scand. Conf. on Image Analysis*, pp. 120–127, Helsinki, Finland. Suomen Hahmontunnistustutkimuksen seura r.y.

Leinonen, L., Kangas, J., Torkkola, K., and Juvas, A. (1992). Dysphonia detected by pattern recognition of spectral composition. *J. Speech and Hearing Res.*, 35:287–295.

Linde, A., Buzo, Y., and Gray, R. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95.

Luttrell, S. P. (1989). Self-organisation: A derivation from first principles of a class of learning algorithms. In *Proc. Ijcnn'89. Int Joint Conf. On Neural Networks*, volume Ii, pp. 495–498, Piscataway, Nj. Ieee Technical Activities Board, Neural Network Committee, Usa; Int Neural Network Soc, Ieee Service Center.

Luttrell, S. P. (1994). A Bayesian analysis of self-organising maps. *Neural Computation*, 6(5):767–794.

Martinetz, T. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps. In Gielen, S. and Kappen, B., editors, *Proc. ICANN'93, International Conference on Artificial Neural Networks*, pp. 427–434, London, UK. Springer.

Martinetz, T. and Schulten, K. (1994). Topology representing networks. *Neural Networks*, 7(2).

Moon, T. K. (1996). The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60.

Mozer, M. (1993). Neural networks architectures for temporal sequence processing. In Weigend, A. S. and Gershenfeld, N. A., editors, *Time Series Prediction*, pp. 243–264. Addison-Wesley Publishing Company.

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

Polani, D. (2001). *Self Organizing Neural Networks: Recent Advances and Applications*, chapter 2 Measures for the Organization of Self Organizing Maps, pp. 13–41. Studies in Fuzziness and Soft Computing. Physica-Verlag.

Proakis, J. G. and Manolakis, D. G. (1992). *Digital Signal Processing: Principles, Algorithms, and Applications*. Macmillan Publishing Compan.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2336.

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.

Shynk, J. J. (1995). Adaptive filtering. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pp. 74–78. The Mit Press.

Somervuo, P. (1999). Time topology for the self-organizing map. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings.*, volume 3, pp. 1900–5, Piscataway, NJ. IEEE Service Center.

Somervuo, P. (2000a). Competing hidden markov models on the self-organizing map. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pp. 169–174, Piscataway, NJ. Helsinki Univ of Technology, IEEE.

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

Somervuo, P. and Kohonen, T. (1999). Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10(2):151–159.

Tryba, V. and Goser, K. (1991). Self-Organizing Feature Maps for process control in chemistry. In Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pp. 847–852, Amsterdam, Netherlands. North-Holland.

Varsta, M., Heikkonen, J., and del Ruiz Millán, J. (1997). Context learning with the self organizing map. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps*, pp. 197–202. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland. ISBN 951-22-3589-7.

Varsta, M., Heikkonen, J., and Lampinen, J. (2000). Analytical comparison of the Temporal Kohonen Map and the Recurrent Self Organizing Map. In Verleysen, M., editor, *Proceedings of the ESANN'2000, European Symposium on Artifical Neural Networks*, pp. 273–280, Bruges, Belgium. D-Facto.

Varsta, M., Heikkonen, J., Millán, J. D. R., and Lampinen, J. (1998). On the convergence properties of the recurrent self-organizing map. In *Proceedings of the ICANN'98*.

Villmann, T., Der, R., Herrmann, M., and Martinetz, T. (1997). Topology preservation in self-organizing feature maps: exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–66.

Viterbi, A. J. (1967). Error bounds for convolutional codes and asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*.

Werbos, P. J. (1990). Backpropagation through time. *Proceedings of the IEEE*, 78(10):1550–1560.

Widrow, B. and Stearns, S. D. (1985). *Adaptive Signal Processing*. Signal Processing. PreniceHall.

Wiskott, L. and Sejnowski, T. (1997). Objective functions for neural map formation. In *Artificial Neural Networks, ICANN 97*, volume 1327 of *Lecture Notes in Computer Science*, pp. 243–248.

Zachmanoglou, E. and Thoe, D. W. (1986). *Introduction to Partial Differential Equations with Applications*, chapter Chapter II: Integral Curves and Surfaces of Vector Fields. Dover.

Publication 1: Surface Modeling and Robot Path Generation Using Self-Organization

Varsta, M., and Koikkalainen, P. (1996)
In *Proceedings of ICPR '96*, pages 30–34. IEEE.

Publication 2: Robot Path generation for surface processing applications via neural networks

Koikkalainen, P., and Varsta, M. (1996)
In *Proceedings of the SPIE*, Vol. 2904, pages 66–73. SPIE.

Publication 3: Epileptic Activity Detection in EEG with Neural Networks

Varsta, M., Heikkonen, J., and Millán, J. del R. (1997)
In *Proceedings of the 1997 International Conference on Engineering Applications of Neural Networks* , pages 179–186.
The Royal Institute of Technology, Stockholm.

Publication 4: A Recurrent Self Organizing Map for Temporal Sequence Processing

Varsta, M., Millán, J. del R., and Heikkonen, J., (1997)
In *ICANN'97: International Conference on Artificial Neural Networks* , LNCS vol. 1327, pages 421–426. Springer.

Publication 5: Temporal Sequence Processing using Recurrent SOM

Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998) In *KES '98: Proceedings of the Second International Conference on Knowledge Based Engineering Systems*, vol. 1, pages 290–296. IEEE.

Publication 6:Temporal Kohonen Map and the Recurrent Self-Organizing Map: Analytical and Experimental Comparison

Varsta, M., Heikkonen, J., Lampinen, J., and Millán, J. del R. (2001)
Neural Processing Letters, Vol. 13, pages 237–251, Kluwer Academic Publishers.

ISBN 951-22-5764-5 (printed)
ISBN 951-22-5765-3 (PDF)
ISSN 1455-0474