Chapter 1
Self-Organizing Maps

Marc M. Van Hulle

Abstract A topographic map is a two-dimensional, nonlinear approximation of a potentially high-dimensional data manifold, which makes it an appealing instrument for visualizing and exploring high-dimensional data. The Self-Organizing Map (SOM) is the most widely used algorithm, and it has led to thousands of applications in very diverse areas. In this chapter, we will introduce the SOM algorithm, discuss its properties and applications, and also discuss some of its extensions and new types of topographic map formation, such as the ones that can be used for processing categorical data, time series and tree structured data.

1.1 Introduction

One of the most prominent features of the mammalian brain is the *topographical* organization of its sensory cortex: neighboring nerve cells (neurons) can be driven by stimuli originating from neighboring positions in the sensory input space, and neighboring neurons in a given brain area project to neighboring neurons in the next area. In other words, the connections establish a so-called *neighborhood-preserving* or *topology-preserving* map, or *topographic map* for short. In the visual cortex, we call this a *retinotopic* map; in the somatosensory cortex a *somatotopic* map (a map of the body surface), and in the auditory cortex a *tonotopic map* (of the spectrum of possible sounds).

The study of topographic map formation, from a theoretical perspective, started with basically two types of self-organizing processes, gradient-based learning and competitive learning, and two types of network architectures (Fig. 1.1) (for a review, see Van Hulle, 2000). In the first architecture, which is commonly referred to as the Willshaw-von der Malsburg model (Willshaw and von der Malsburg, 1976), there

K.U.Leuven, Laboratorium voor Neuro- en Psychofysiologie, Campus Gasthuisberg, Herestraat 49, BE-3000 Leuven, Belgium, E-mail: marc@neuro.kuleuven.be

are two sets of neurons, arranged in two (one- or) two-dimensional layers or *lattices*[1] (Fig. 1.1A). Topographic map formation is concerned with learning a mapping for which neighboring neurons in the input lattice are connected to neighboring neurons in the output lattice.

The second architecture is far more studied, and is also the topic of this chapter. We now have continuously valued inputs taken from the input space $\mathbb{R}^d$, or the data manifold $V \subseteq \mathbb{R}^d$, which need not be rectangular or have the same dimensionality as the lattice to which it projects (Fig. 1.1B). To every neuron i of the lattice A corresponds a reference position in the input space, called the weight vector $\mathbf{w}_i = [w_{ij}] \in \mathbb{R}^d$. All neurons receive the same input vector $\mathbf{v} = [v_1, \ldots, v_d] \in V$. Topographic map formation is concerned with learning a map $\mathcal{Y}_A$ of the data manifold V (grey shaded area in Fig. 1.2), in such a way that neighboring lattice neurons, i, j, with lattice positions $\mathbf{r}_i, \mathbf{r}_j$, code for neighboring positions, $\mathbf{w}_i, \mathbf{w}_j$, in the input space (*cf.*, the inverse mapping, Ψ). The forward mapping, Φ, from the input space to the lattice, is not necessarily topology-preserving – neighboring weights do not necessarily correspond to neighboring lattice neurons–, even after learning the map, due to the possible mismatch in dimensionalities of the input space and the lattice (see *e.g.*, Fig. 1.3). In practice, the map is represented in the input space in terms of neuron weights that are connected by straight lines, if the corresponding neurons are nearest neighbors in the lattice (*e.g.*, see the left panel of Fig. 1.2 or Fig. 1.3). When the map is topology preserving, it can be used for visualizing the data distribution by projecting the original data points onto the map. The advantage of having a flexible map, compared to *e.g.*, a plane specified by principal components analysis (PCA), is demonstrated in Fig. 1.4. We observe that the three classes are better separated with a topographic map than with PCA. The most popular learning algorithm for this architecture is the Self-Organizing Map (SOM) algorithm by Teuvo Kohonen (Kohonen 1982, Kohonen, 1984), whence this architecture is often referred to as Kohonen's model.

Chapter overview

We start with the basic version of the SOM algorithm where we discuss the two stages of which it consists: the competitive and the cooperative ones. We then discuss the topographic ordering properties of the algorithm: how it unfolds and develops topographically-ordered maps, whether there exists a mathematical proof of ordering, and whether topological defects in the map could still occur after the learning process has ended. We also discuss the convergence properties of the algorithm, and in what sense the converged weights are modeling the input density (is the weight density a linear function of the input density?).

[1] A lattice is an undirected graph in which every non-border vertex has the same, fixed number of incident edges, and which usually appears in the form of an array with a rectangular- or simplex topology.

We then discuss the applications of the SOM algorithm, for which thousands of them have been reported in the open literature. Rather than attempting for an extensive overview, we group the applications into three areas: vector quantization, regression and clustering. The latter is the most important one since it is a direct consequence of the data visualization- and exploration capabilities of the topographic map. We highlight a number of important applications such as the WEBSOM (Kaski *et al.*, 1998), for organizing large document collections, the PicSOM (Laaksonen *et al.*, 2002), for content-based image retrieval, and the Emergent Self Organizing Maps (ESOM) (Ultsch and Mörchen, 2005), for which we consider the MusicMiner (Risi *et al.*, 2007), for organizing large collections of music, and an application for classifying police reports of criminal incidents.

We then give an overview of a number of extensions of the SOM algorithm. The motivation behind these was to improve the original algorithm, or to extend its range of applications, or to develop new ways to perform topographic map formation.

We then detail three important extensions of the SOM algorithm. First, we discuss the growing topographic map algorithms. These algorithms consider maps with a dynamically-defined topology so as to better capture the fine-structure of the input distribution. Second, since many input sources have a temporal characteristic, which is not captured by the original SOM algorithm, several algorithms have been developed based on a recurrent processing of time signals (recurrent topographic maps). It is a heavily researched area since some of these algorithms are capable of processing tree-structured data. Third, another topic of current research is the kernel topographic map, which is in line with the "kernelization" trend of mapping data into a feature space. Rather than Voronoi regions, the neurons are equipped with overlapping activation regions, in the form of kernel functions, such as Gaussians. Important future developments are expected for these topographic maps, such as the visualization and clustering of structure-based molecule descriptions, and other biochemical applications.

Finally, we formulate a conclusion to the chapter.

1.2 SOM Algorithm

The SOM algorithm distinguishes two stages: the *competitive* stage and the *cooperative* stage. In the first stage, the best matching neuron is selected, *i.e.*, the "winner", and in the second stage, the weights of the winner are adapted as well as those of its immediate lattice neighbors. We consider the minimum Euclidean distance version of the SOM algorithm only (also the dot product version exists, see Kohonen, 1995).

Competitive stage

For each input $\mathbf{v} \in V$, we select the neuron with the smallest Euclidean distance ("Winner-Takes-All", WTA), which we call the "winner":

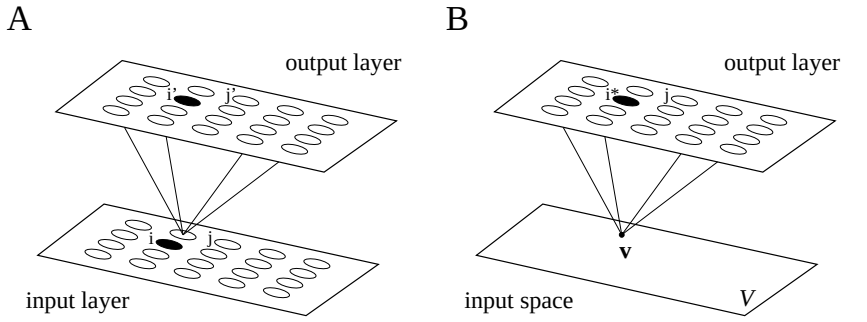

Fig. 1.1 (A) Willshaw-von der Malsburg model. Two isomorphic, rectangular lattices of neurons are shown: one represents the input layer and the other the output layer. Neurons are represented by circles: filled circles denote active neurons ("winning" neurons); open circles denote inactive neurons. As a result of the weighted connections from the input to the output layer, the output neurons receive different inputs from the input layer. Two input neurons are labeled (i, j) as well as their corresponding output layer neurons (i', j'). Neurons i and i' are the only active neurons in their respective layers. (B) Kohonen model. The common input all neurons receive is directly represented in the input space, $\mathbf{v} \in V \subseteq \mathbb{R}^d$. The "winning" neuron is labeled as i^*: its weight (vector) is the one that best matches the current input (vector).

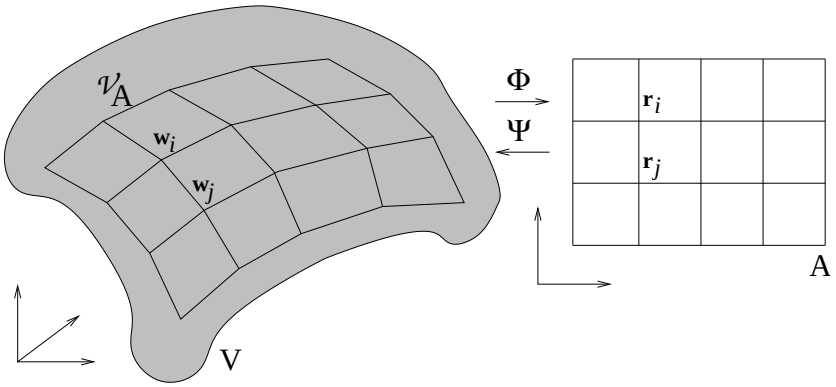

Fig. 1.2 Topographic mapping in the Kohonen architecture. In the left panel, the topology-preserving map $\mathcal{V}_A$ of the data manifold $V \subseteq \mathbb{R}^d$ (grey shaded area) is shown. The neuron weights $\mathbf{w}_i, \mathbf{w}_j$ are connected by a straight line since the corresponding neurons i, j in the lattice A (right panel), with lattice coordinates $\mathbf{r}_i, \mathbf{r}_j$, are nearest neighbors. The forward mapping Φ is from the input space to the lattice; the backward mapping Ψ is from the lattice to the input space. The learning algorithm tries to make neighboring lattice neurons, i, j, code for neighboring positions, $\mathbf{w}_i, \mathbf{w}_j$, in the input space.

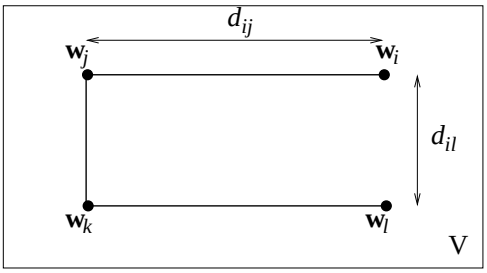

Fig. 1.3 Example of a one dimensional lattice consisting of four neurons i, j, k, l in a two dimensional rectangular space. The distance between the weight vectors of neurons i, j, d_{ij}, is larger than between that of neurons i, l, d_{il}. This means that, at least in this example, neighboring neuron weights do not necessarily correspond to neighboring neurons in the lattice.

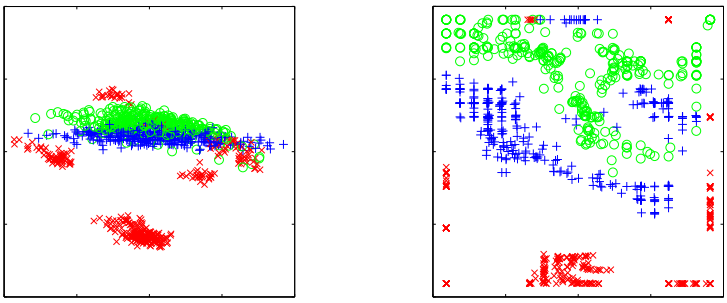

Fig. 1.4 Oil flow data set visualized using PCA (left panel) and a topographic map (right panel). The latter was obtained with the GTM algorithm (Bishop *et al.*, 1996; Bishop *et al.*, 1998). Since the GTM performs a nonlinear mapping, it is better able to separate the three types of flow configurations: laminar (red crosses), homogeneous (blue plusses) and annular (green circles) (Bishop, 2006, reprinted with permission.)

$$i^* = \arg\min_i \|\mathbf{w}_i - \mathbf{v}\|. \tag{1.1}$$

By virtue of the minimum Euclidean distance rule, we obtain a Voronoi tessellation of the input space: to each neuron corresponds a region in the input space, the boundaries of which are perpendicular bisector planes of lines joining pairs of weight vectors (the grey shaded area in Fig. 1.5 is the Voronoi region of neuron j). Remember that the neuron weights are connected by straight lines (links or edges): they indicate which neurons are nearest neighbors in the lattice. These links are important for verifying whether the map is topology preserving.

Cooperative stage

It is now crucial to the formation of topographically-ordered maps that the neuron weights are not modified independently of each other, but as topologically-related

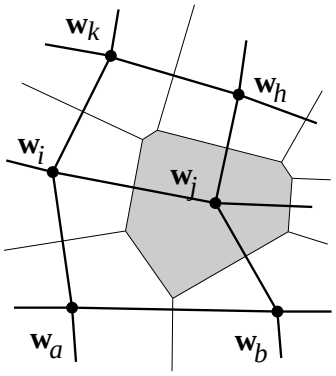

Fig. 1.5 Definition of quantization region in the Self-Organizing Map (SOM). Portion of a lattice (thick lines) plotted in terms of the weight vectors of neurons $a, \ldots, k$, in the two-dimensional input space, *i.e.*, $\mathbf{w}_a, \ldots, \mathbf{w}_k$.

subsets on which similar kinds of weight updates are performed. During learning, not only the weight vector of the winning neuron is updated, but also those of its lattice neighbors and, thus, which end up responding to similar inputs. This is achieved with the neighborhood function, which is centered at the winning neuron, and decreases with the lattice distance to the winning neuron[2].

The weight update rule in incremental mode[3] is given by:

$$\Delta \mathbf{w}_i = \eta \, \Lambda(i, i^*, \sigma_\Lambda(t)) \, (\mathbf{v} - \mathbf{w}_i), \quad \forall i \in A, \tag{1.2}$$

with Λ the neighborhood function, *i.e.*, a scalar-valued function of the lattice coordinates of neurons i and i^*, $\mathbf{r}_i$ and $\mathbf{r}_i^*$, mostly a Gaussian:

$$\Lambda(i, i^*) = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_{i^*}\|^2}{2\sigma_\Lambda^2}\right), \tag{1.3}$$

with range σ_Λ (*i.e.*, the standard deviation). (We further drop the parameter $\sigma_\Lambda(t)$ from the neighborhood function to simplify our notation.) The positions $\mathbf{r}_i$ are usually taken to be the nodes of a discrete lattice with a regular topology, usually a 2 dimensional square or rectangular lattice. An example of the effect of the neighborhood function in the weight updates is shown in Fig. 1.6 for a 4×4 lattice. The parameter σ_Λ, and usually also the learning rate η, are gradually decreased over time. When the neighborhood range vanishes, the previous learning rule reverts to standard unsupervised competitive learning (note that the latter is unable to form

[2] Besides the neighborhood function, also the neighborhood set exists, consisting of all neurons to be updated in a given radius from the winning neuron (see Kohonen, 1995).

[3] With incremental mode it is meant that the weights are updated each time an input vector is presented. This is to be contrasted with batch mode where the weights are only updated after the presentation of the full training set ("batch").

topology-preserving maps, pointing to the importance of the neighborhood function).

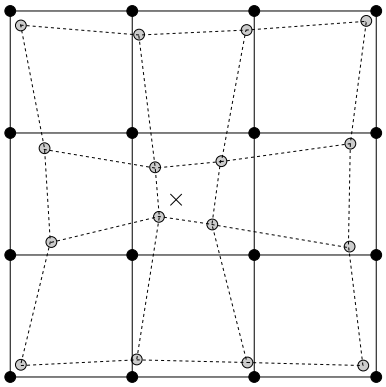

Fig. 1.6 The effect of the neighborhood function in the SOM algorithm. Starting from a perfect arrangement of the weights of a square lattice (full lines), the weights nearest to the current input (indicated with the cross) receive the largest updates, those further away smaller updates, resulting in the updated lattice (dashed lines).

As an example, we train a 10×10 square lattice with the SOM algorithm on a uniform square distribution $[-1,1]^2$, using a Gaussian neighborhood function of which the range $\sigma_\Lambda(t)$ is decreased as follows:

$$\sigma_\Lambda(t) = \sigma_{\Lambda 0} \exp\left(-2\sigma_{\Lambda 0}\frac{t}{t_{max}}\right), \tag{1.4}$$

with t the present time step, t_{max} the maximum number of time steps, and $\sigma_{\Lambda 0}$ the range spanned by the neighborhood function at $t = 0$. We take $t_{max} = 100,000$ and $\sigma_{\Lambda 0} = 5$ and the learning rate $\eta = 0.01$. The initial weights (*i.e.*, for $t = 0$) are chosen randomly from the same square distribution. Snapshots of the evolution of the lattice are shown in Fig. 1.7. We observe that the lattice is initially tangled, then contracts, unfolds, and expands so as to span the input distribution. This two-phased convergence process is an important property of the SOM algorithm and it has been thoroughly studied from a mathematical viewpoint in the following terms: 1) the topographic ordering of the weights and, thus, the formation of topology-preserving mappings, and 2) the convergence of these weights (energy function minimization). Both topics wil be discussed next. Finally, the astute reader has noticed that at the end of the learning phase, the lattice is smooth, but then suddenly becomes more erratic. This is an example of a phase transition, and it has been widely studied for the SOM algorithm (see, *e.g.*, Der and Herrmann, 1993).

Finally, since the speed of convergence depends on the learning rate, also a version without one has been developed, called batch map (Kohonen, 1995):

$$\mathbf{w}_i = \frac{\sum_\mu \Lambda(i^*, i)\mathbf{v}^\mu}{\sum_\mu \Lambda(i^*, i)}, \quad \forall i, \tag{1.5}$$

and it leads to a faster convergence of the map.

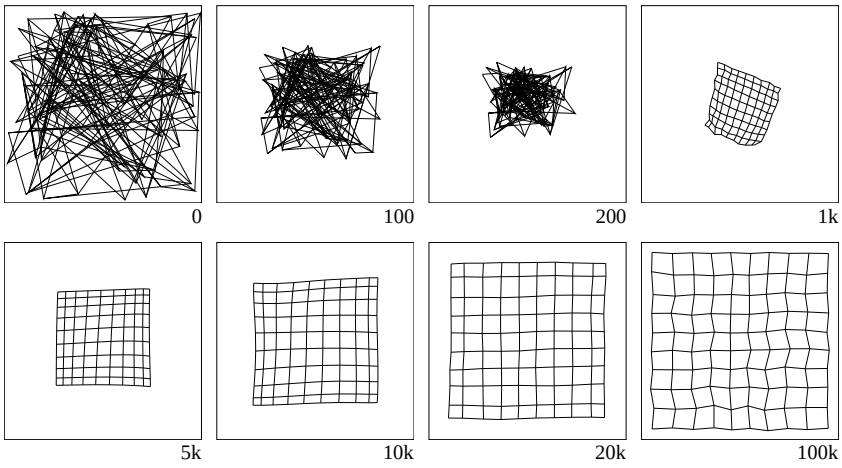

Fig. 1.7 Evolution of a 10×10 lattice with a rectangular topology as a function of time. The outer squares outline the uniform input distribution. The values given below the squares represent time.

1.2.1 Topographic ordering

In the example of Fig. 1.7, we have used a two-dimensional square lattice for mapping a two-dimensional uniform, square distribution. We can also use the same lattice for mapping a non-square distribution, so that there is a topological mismatch, for example, a circular and an L-shaped distribution (Fig. 1.8A,B). We use the same lattice and simulation set-up as before but now we show only the final results. Consider first the circular distribution: the weight distribution is now somewhat non-uniform. For the L-shaped distribution, we see in addition that there are several neurons outside the support of the distribution, and some of them even have a zero (or very low) probability to be active: hence, they are often called "dead" units. It is hard to find a better solution for these neurons without clustering them near the inside corner of the L-shape.

We can also explore the effect of a mismatch in lattice dimensionality. For example, we can develop a one-dimensional lattice ("chain") in the same two-dimensional square distribution as before. (Note that it is now impossible to preserve all of the topology). We see that the chain tries to fill the available space as much as possible

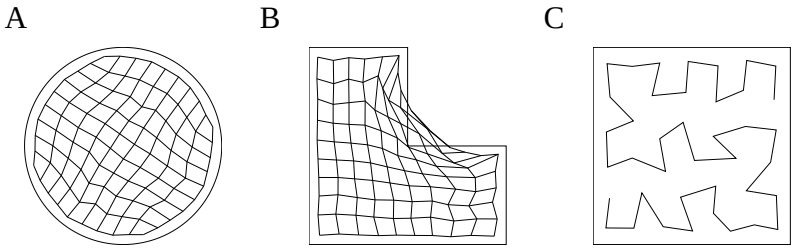

Fig. 1.8 Mapping of a 10×10 neurons lattice onto a circular (A) and an L-shaped (B) uniform distribution, and a 40 neurons one-dimensional lattice onto a square uniform distribution (C).

(Fig. 1.8C): the resulting map approximates the so-called space-filling *Peano curve*[4] (Kohonen, 1995, pp. 81, 87).

1.2.1.1 Proofs or ordering

It is clear that the neighborhood function plays a crucial role in the formation of topographically-ordered weights. Although this may seem evident, the ordering itself is very difficult to describe (and prove!) in mathematical terms. The mathematical treatments that have been considered are, strictly speaking, only valid for one-dimensional lattices developed in one-dimensional spaces. Cottrell and Fort (1987) presented a mathematical stringent (but quite long) proof of the ordering process for the one-dimensional case. For a shorter constructive proof, we refer to (Kohonen, 1995, pp. 100–105; for an earlier version, see Kohonen, 1984, pp. 151–154). The results of Kohonen (1984) and Cottrell and Fort (1987) have been extended by Erwin and co-workers (1992) to the more general case of a monotonically decreasing neighborhood function. However, the same authors also state that a strict proof of convergence is unlikely to be found for the higher-than-one-dimensional case.

1.2.1.2 Topological defects

As said before, the neighborhood function plays an important role in producing topographically-ordered lattices, however, this does not imply that we are guaranteed to obtain one. Indeed, if we decrease the neighborhood range too fast, then there could be topological defects (Geszti, 1990; Heskes and Kappen, 1993). These defects are difficult to iron out, if at all, when the neighborhood range vanishes. In the case of a chain, we can obtain a so-called *kink* (Fig. 1.9).

Consider, as a simulation example, a rectangular lattice sized $N = 24 \times 24$ neurons with the input samples taken randomly from a two-dimensional uniform distribution $p(\mathbf{v})$ within the square $[0,1]^2$. The initial weight vectors are randomly drawn

[4] A Peano curve is an infinitely and recursively convoluted fractal curve which represents the continuous mapping of, *e.g.*, a one-dimensional interval onto a two-dimensional surface.

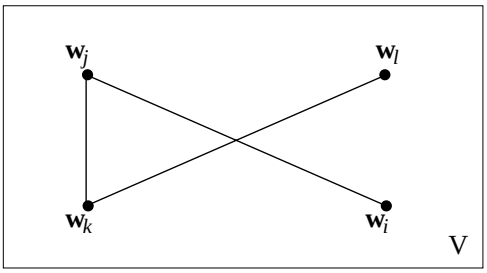

Fig. 1.9 Example of a topological defect ("kink") in a chain consisting of four neurons i, j, k, l in a two dimensional rectangular space.

from this distribution. We now perform incremental learning and decrease the range as follows:

$$\sigma_\Lambda(t) = \sigma_{\Lambda 0} \exp\left(-2\,\sigma_{\Lambda 0}\,\frac{t}{t_{max}}\right), \tag{1.6}$$

but now with t the present time step and $t_{max} = 275,000$. For the learning rate, we take $\eta = 0.015$. The evolution is shown in Fig. 1.10. The neighborhood range was too rapidly decreased since the lattice is twisted and, even if we would continue the simulation, with zero neighborhood range, the *twist* will not be removed.

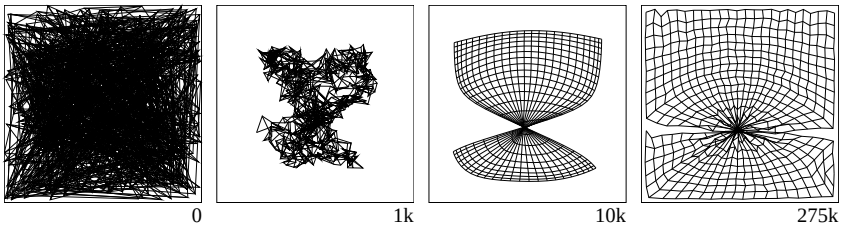

Fig. 1.10 Example of the formation of a topological defect called "twist" in a 24×24 lattice. The evolution is shown for different time instances (values below the squares).

1.2.2 Weight convergence, energy function

Usually, in neural network learning algorithms, the weight update rule performs gradient descent on an energy function E (also termed error-, cost-, distortion- or objective function):

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}}, \tag{1.7}$$

so that convergence to a local minimum in E can be easily shown, for example, in an average sense. However, contrary to intuition, the SOM algorithm *does not* perform gradient descent on an energy function as long as the neighborhood range has not vanished (when $\sigma_\Lambda = 0$, an energy function exist: one is minimizing the mean squared error (MSE) due to the quantization of the input space into Voronoi regions). Hence, strictly speaking, we cannot judge the degree of optimality achieved by the algorithm during learning.

In defense of the lack of an energy function, Kohonen emphasizes that there is no theoretical reason why the SOM algorithm *should* ensue from such a function (Kohonen, 1995, p. 122), since: 1) the SOM algorithm is aimed at developing (specific) topological relations between clusters in the input space, and 2) it yields an approximative solution of an energy function (by virtue of its connection with the Robbins-Munro stochastic approximation technique), so that convergence is not a problem in practice.

Besides this, a lot of effort has been devoted to developing an energy function which is minimized during topographic map formation (Tolat, 1990; Kohonen, 1991; Luttrell, 1991; Heskes and Kappen, 1993; Erwin *et al.*, 1992). Both Luttrell (1991) and Heskes and Kappen (1993) were able to show the existence of an energy function by modifying the definition of the winner. Heskes and Kappen first ascribed a *local error* e_i to each neuron i, at time t:

$$e_i(\mathbf{W},\mathbf{v},t) = \frac{1}{2}\sum_{j\in A}\Lambda(i,j)\|\mathbf{v}-\mathbf{w}_j\|^2, \tag{1.8}$$

with $\mathbf{W} = [\mathbf{w}_i]$ the vector of all neuron weights, and then defined the winner as the neuron for which the local error is minimal:

$$i^* = \arg\min_i \sum_j \Lambda(i,j)\|v-w_j\|^2. \tag{1.9}$$

This was also the equation introduced by Luttrell (1991), but with $\sum_j \Lambda(i,j) = 1$. The actual weight update rule remains the same as in the original SOM algorithm and, thus, still considers several neurons around the winner. The solution proposed by Kohonen (1991) takes a different starting point, but it leads to a more complicated learning rule for the winning neuron (for the minimum Euclidean distance SOM algorithm, see Kohonen, 1995, pp. 122–124).

1.2.2.1 Weight density

The next question is: in what sense are the converged weights modeling the input probability density? What is the distribution of the weights *vs.* that of the inputs? Contrary to what was originally assumed (Kohonen, 1984), the weight density at convergence, also termed the (inverse of the) magnification factor, is not a linear function of the input density $p(\mathbf{v})$.

Ritter and Schulten (1986) have shown that, for a one-dimensional lattice, developed in a one-dimensional input space:

$$p(w_i) \propto p(v)^{\frac{2}{3}}, \tag{1.10}$$

in the limit of an infinite density of neighbor neurons (continuum approximation). Furthermore, when a discrete lattice is used, the continuum approach undergoes a correction, *e.g.*, for a neighborhood set with σ_Λ neurons around each winner neuron ($\Lambda(i,i^*) = 1$ *iff* $|r_i - r_{i^*}| \leq \sigma_\Lambda$):

$$p(w_i) \propto p^{\alpha}, \tag{1.11}$$

with:

$$\alpha = \frac{2}{3} - \frac{1}{3\sigma_\Lambda^2 + 3(\sigma_\Lambda + 1)^2}\ .$$

For a discrete lattice of N neurons, it is expected that, for $N \rightarrow \infty$ and for minimum MSE quantization, in d-dimensional space, the weight density will be proportional to (Kohonen, 1995):

$$p(\mathbf{w}_i) \propto p^{\frac{1}{1+\frac{2}{d}}}(\mathbf{v}), \tag{1.12}$$

or that in the one-dimensional case:

$$p(w_i) \propto p(v)^{\frac{1}{3}}. \tag{1.13}$$

The connection between the continuum and the discrete approach was established for the one-dimensional case by Ritter (1991), for a discrete lattice of N neurons, with $N \rightarrow \infty$, and for a neighborhood set with σ_Λ neurons around each "winner" neuron.

Finally, regardless of the effect resorted by the neighborhood function or -set, it is clear that the SOM algorithm tends to undersample high probability regions and oversample low probability ones. This affects the separability of clusters: *e.g.*, when the clusters overlap, the cluster boundary will be more difficult to delineate in the overlap region than for a mapping which has a linear weight distribution (Van Hulle, 2000).

1.3 Applications of SOM

The graphical map displays generated by the SOM algorithm are easily understood, even by non-experts in data analysis and statistics. The SOM algorithm has led to literally thousands of applications in areas ranging from automatic speech recognition, condition monitoring of plants and processes, cloud classification, and micro-array

data analysis, to document- and image organization and retrieval (for an overview, see Centre, 2003 `http://www.cis.hut.fi/research/som-bibl/`). The converged neuron weights yield a model of the training set in three ways: vector quantization, regression, and clustering.

Vector quantization

The training samples are modeled in such a manner that the average discrepancy between the data points and the neuron weights is minimized. In other words, the neuron weight vectors should "optimally" quantize the input space from which the training samples are drawn, just like one would desire for an adaptive vector quantizer (Gersho and Gray, 1991). Indeed, in standard unsupervised competitive learning (UCL), and also the SOM algorithm, when the neighborhood has vanished ("zero-order" topology), the weight updates amount to centroid estimation (usually the mean of the samples which activate the corresponding neuron) and minimum Euclidean distance classification (Voronoi tessellation), and attempt to minimize the mean squared error due to quantization, or some other quantization metric that one wishes to use. In fact, there exists an intimate connection between the batch version of the UCL rule and the zero-order topology SOM algorithm, on the one hand, and the generalized Lloyd algorithm for building vector quantizers, on the other hand (Luttrell, 1989,1990) (for a review, see Van Hulle, 2000). Luttrell showed that the neighborhood function can be considered as a probability density function of which the range is chosen to capture the noise process responsible for the distortion of the quantizer's output code (*i.e.*, the index of the winning neuron), *e.g.*, due to noise in the communication channel. Luttrell adopted for the noise process a zero-mean Gaussian, so that there is theoretical justification for choosing a Gaussian neighborhood function in the SOM algorithm.

Regression

We can also interpret the map as a case of non-parametric regression: no prior knowledge is assumed about the nature or shape of the function to be regressed. Non-parametric regression is perhaps the first successful statistical application of the SOM algorithm (Ritter *et al.*, 1992; Mulier and Cherkassky, 1995; Kohonen, 1995): the converged topographic map is intended to capture the principal dimensions (principal curves, principal manifolds) of the input space. The individual neurons represent the "knots" that join piecewise smooth functions, such as splines, which act as interpolating functions for generating values at intermediate positions. Furthermore, the lattice coordinate system can be regarded as a (approximate) global coordinate system of the data manifold (Fig. 1.2).

Clustering

The most widely used application of the topographic map is clustering, *i.e.*, the partitioning of the data set into subsets of "similar" data, without using prior knowledge about these subsets. One of the first demonstrations of clustering was by Ritter and Kohonen (1989). They had a list of 16 animals (birds, predators and preys) and 13 binary attributes for each one of them (*e.g.*, large size or not, hair or not, 2 legged or not, can fly or not,...). After training a 10×10 lattice of neurons with these vectors (supplemented with the 1-out-16 animal code vector, thus, in total, a 29 dimensional binary vector), and labeling the winning neuron for each animal code vector, a natural clustering of birds, predators and preys appeared in the map. The authors called this the "semantic map".

In the previous application, the clusters and their boundaries were defined by the user. In order to visualize clusters more directly, we need an additional technique. We can compute the mean Euclidean distance between a neuron's weight vector and the weight vectors of its nearest neighbors in the lattice. The maximum and minimum of the distances found for all neurons in the lattice is then used for scaling these distances between 0 and 1; the lattice then becomes a grey scale image with white pixels corresponding to *e.g.* 0 and black pixels to 1. This is called the U-matrix (Ultsch and Siemon, 1990), for which several extensions have been developed to remedy the oversampling of low probability regions (possibly transition regions between clusters) (Mörchen and Ultsch, 2005).

An important example is the WEBSOM (Kaski *et al.*, 1998). Here, the SOM is used for organizing document collections ("document map"). Each document is represented as a vector of keyword occurrences. Similar documents then become grouped into the same cluster. After training, the user can zoom into the map to inspect the clusters. The map is manually or automatically labeled with keywords (*e.g.*, from a man-made list) in such a way that, at each zoom-level, the same density of keywords is shown (so as not to clutter the map with text). The WEBSOM has also been applied to visualizing clusters in patents based on keyword occurrences in patent abstracts (Kohonen *et al.*, 1999).

An example of a content-based image retrieval system is the PicSOM (Laaksonen *et al.*, 2002). Here, low level features (color, shape, texture, etc...) of each image are considered. A separate two-dimensional SOM is trained for each low level feature (in fact, a hierarchical SOM). In order to be able to retrieve one particular image from the database, one is faced with a semantic gap: how well do the low level features correlate with the image contents? To bridge this gap, *relevance feedback* is used: the user is shown a number of images and he/she has to decide which ones are relevant and which ones not (close or not to the image the user is interested in). Based on the trained PicSOM, the next series of images shown are then supposed to be more relevant, and so on.

For high-dimensional data visualization, a special class of topographic maps, called Emergent Self Organizing Maps (ESOM) (Ultsch and Mörchen, 2005), can be considered. According to Alfred Ultsch, emergence is the ability of a system to produce a phenomenon on a new, higher level. In order to achieve

emergence, the existence and cooperation of a large number of elementary processes is necessary. An Emergent SOM differs from the traditional SOM in that a very large number of neurons (at least a few thousand) are used (even larger than the number of data points). The ESOM software is publicly available from `http://databionic-esom.sourceforge.net/`.

As an example, Ultsch and co-workers developed the *MusicMiner* for organizing large collections of music (Risi *et al.*, 2007). Hereto, low level audio features were extracted from raw audio data, and static and temporal statistics were used for aggregating these low level features into higher level ones. A supervised feature selection was performed to come up with a non-redundant set of features. Based on the latter, an ESOM was trained for clustering and visualizing collections of music. In this way, consistent clusters were discovered that correspond to music genres (Fig. 1.11). The user can then navigate the sound space and interact with the maps to discover new songs that correspond to his/her taste.

Fig. 1.11 Organizing large collections of music by means of an ESOM trained on high level audio features (MusicMiner). Shown is the map with several music genres labeled. (Risi *et al.*, 2007, reprinted with permission.)

As an example ESOM application, in the realm of text mining, we consider the case of police reports of criminal incidents. When a victim of a violent incident makes a statement to the police, the police officer has to judge whether, *e.g.*, domestic violence is involved. However, not all cases are correctly recognized and are, thus, wrongly assigned the label “non-domestic violence”. Because it is very time consuming to classify cases and to verify whether or not the performed classifications are correct, text mining techniques and a reliable classifier are needed. Such an automated triage system would result in major cost- and time savings. A collection of terms (thesaurus), obtained by a frequency analysis of key words, was

constructed, consisting of 123 terms. A 50×82 toroidal lattice (in a toroidal lattice, the vertical and horizontal coordinates are circular) was used, and trained with the ESOM algorithm on 4814 reports of the year 2007; the validation set consisted of 4738 cases (of the year 2006). The outcome is shown in Fig. 1.12 (Poelmans, 2008, unpublished results).

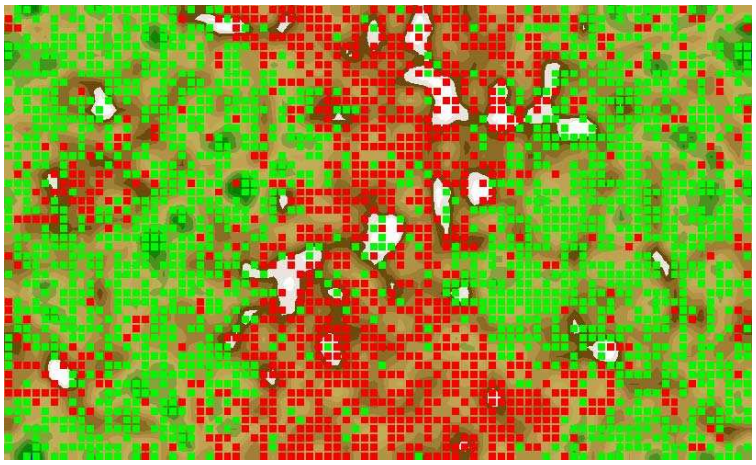

Fig. 1.12 Toroidal lattice trained with the ESOM algorithm showing the distribution of domestic violence cases (red squares) and non-domestic violence cases (green squares). The background represents the sum of the distances between the weight vector of each neuron and those of its nearest neighbors in the lattice, normalized by the largest occurring sum (white) (*i.e.*, the U-matrix). We observe that the domestic violence cases appear in one large cluster and a few smaller clusters, mostly corresponding to violence in homosexual relationships. (Figure is a courtesy of Jonas Poelmans.)

1.4 Extensions of SOM

Although the original SOM algorithm has all the necessary ingredients for developing topographic maps, many adapted versions have emerged over the years (for references, see Kohonen, 1995 and `http://www.cis.hut.fi/research/som-bibl/` which contains over 7000 articles). For some of these, the underlying motivation was to improve the original algorithm, or to extend its range of applications, while for others the SOM algorithm has served as a source of inspiration for developing new ways to perform topographic map formation.

One motivation was spurred by the need to develop a learning rule which performs gradient descent on an energy function (as discussed above). Another was to remedy the occurrence of dead units, since they do not contribute to the representation of the input space (or the data manifold). Several researchers were inspired

by Grossberg's idea (1976) of adding a "conscience" to frequently winning neurons to feel "guilty" and to reduce their winning rates. The same heuristic idea has also been adopted in combination with topographic map formation (DeSieno, 1988; Van den Bout and Miller, 1989; Ahalt *et al.*, 1990). Others exploit measures based on the local distortion error to equilibrate the neurons' "conscience" (Kim and Ra, 1995; Chinrungrueng and Séquin, 1995; Ueda and Nakano, 1993). A combination of the two conscience approaches is the learning scheme introduced Bauer and co-workers (1996).

A different strategy is to apply a competitive learning rule that minimizes the mean absolute error (MAE) between the input samples $\mathbf{v}$ and the N weight vectors (also called the *Minkowski metric* of power one) (Kohonen, 1995, pp. 120, 121) (see also Lin *et al.*, 1997). Instead of minimizing a (modified) distortion criterion, a more natural approach is to optimize an information-theoretic criterion directly. Ralph Linsker was among the first to explore this idea in the context of topographic map formation. He proposed a principle of *maximum information preservation* (Linsker, 1988) – *infomax* for short – according to which a processing stage has the property that the output signals will optimally discriminate, in an information-theoretic sense, among possible sets of input signals applied to that stage. In his 1989 article, he devised a learning rule for topographic map formation in a probabilistic WTA network by maximizing the average mutual information between the output and the signal part of the input, which was corrupted by noise (Linsker, 1989). Another algorithm is the Vectorial Boundary Adaptation Rule (VBAR) which considers the region spanned by a quadrilateral (4 neurons forming a square region in the lattice) as the quantization region (Van Hulle, 1997a,b), and which is able to achieve an equiprobabilistic map, *i.e.*, a map for which every neuron has the same chance to be active (and, therefore, maximizes the information-theoretic entropy).

Another evolution are the growing topographic map algorithms. In contrast with the original SOM algorithm, its growing map variants have a dynamically-defined topology, and they are believed to better capture the fine-structure of the input distribution. We will discuss them in the next section.

Many input sources have a temporal characteristic, which is not captured by the original SOM algorithm. Several algorithms have been developed based on a recurrent processing of time signals and a recurrent winning neuron computation. Also tree structured data can be represented with such topographic maps. We will discuss recurrent topographic maps in this chapter.

Another important evolution are the kernel-based topographic maps: rather than Voronoi regions, the neurons are equipped with overlapping activation regions, usually in the form of kernel functions, such as Gaussians (Fig. 1.19). Also for this case, several algorithms have been developed, and we will discuss a number of them in this chapter.

1.5 Growing Topographic Maps

In order to overcome the topology mismatches that occur with the original SOM algorithm, as well as to achieve an optimal use of the neurons (*cf.*, dead units), the geometry of the lattice has to match that of the data manifold it is intended to represent. For that purpose, several so-called growing (incremental or structure-adaptive) self-organizing map algorithms have been developed. What they share is that the lattices are gradually build up and, hence, do not have a pre-defined structure (*i.e.*, number of neurons and possibly also lattice dimensionality) (Fig. 1.14). The lattice is generated by a successive insertion (and possibly an occasional deletion) of neurons and connections between them. Some of these algorithms can even guarantee that the lattice is free of topological defects (*e.g.*, since the lattice is subgraph of a Delaunay triangularization, see further). We will briefly review the major algorithms for growing self-organizing maps. The algorithms are structurally not very different; the main difference is with the constraints imposed on the lattice topology (fixed or variable lattice dimensionality). We first list the properties common to these algorithms, using the format suggested by Fritzke (1996).

- The network is an undirected graph (lattice) consisting of a number of nodes (neurons) and links or edges connecting them.
- Each neuron i has a weight vector $\mathbf{w}_i$ in the input space V.
- The weight vectors are updated by moving the winning neuron i^*, and its topological neighbors, towards the input $\mathbf{v} \in V$:

$$\Delta \mathbf{w}_{i^*} = \eta_{i^*}(\mathbf{v} - \mathbf{w}_{i^*}), \tag{1.14}$$

$$\Delta \mathbf{w}_i = \eta_i(\mathbf{v} - \mathbf{w}_i), \quad \forall i \in \mathcal{N}_{i^*}, \tag{1.15}$$

  with $\mathcal{N}_{i^*}$ the set of direct topological neighbors of neuron i^* (neighborhood set), and with η_{i^*} and η_i the learning rates, $\eta_{i^*} > \eta_i$.
- At each time step, the local error at the winning neuron i^* is accumulated:

$$\Delta E_{i^*} = (\text{error measure}). \tag{1.16}$$

  The error term is coming from a particular area around $\mathbf{w}_{i^*}$, and is likely to be reduced by inserting new neurons in that area. A central property of these algorithms is the possibility to choose an arbitrary error measure as the basis for insertion. This extends their applications from unsupervised learning ones, such as data visualization, combinatorial optimization and clustering analysis, to supervised learning ones, such as classification and regression. For example, for vector quantization, $\Delta E_{i^*} = \|\mathbf{v} - \mathbf{w}_{i^*}\|^2$. For classification, the obvious choice is the classification error. All models reviewed here can, in principle, be used for supervised learning applications by associating output values to the neurons, *e.g.*, through kernels such as radial basis functions. This makes most sense for the algorithms that adapt their dimensionality to the data.
- The accumulated error of each neuron is used to determine (after a fixed number of time steps) where to insert new neurons in the lattice. After an insertion, the

error information is locally redistributed, which increases the probability that the next insertion will be somewhere else. The local error acts as a kind of memory where much error has occurred; the exponential decay of the error stresses more the recently accumulated error.

- All parameters of the algorithm stay constant over time.

1.5.1 Competitive Hebbian learning and Neural Gas

Historically the first algorithm to develop topologies has been introduced by Martinetz and Schulten, and it is a combination of two methods, competitive Hebbian learning (CHL) (Martinetz, 1993) and the Neural Gas (NG) (Martinetz and Schulten, 1991).

The principle behind CHL is simple: for each input, create a link between the winning neuron and the second winning neuron (*i.e.*, with the second smallest Euclidean distance to the input), if that link does not already exist. Only weight vectors lying in the data manifold develop links between them (thus, non-zero input density regions). The resulting graph is a subgraph of the (induced) Delaunay triangularization (Fig. 1.13), and it has been shown to optimally preserve topology in a very general sense.

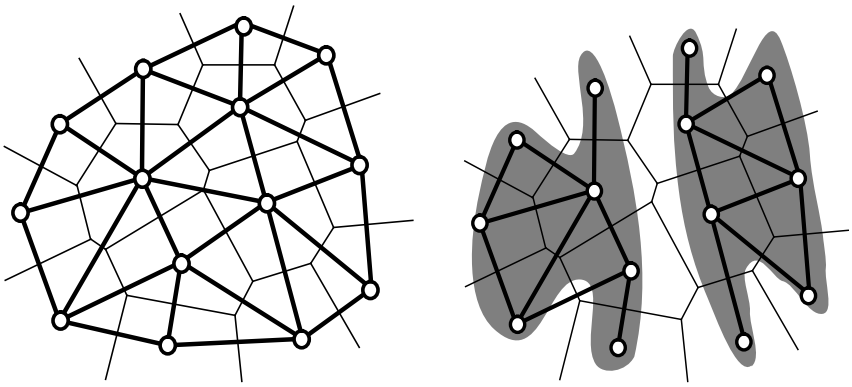

Fig. 1.13 *Left panel*: Delaunay triangularization. The neuron weight positions are indicated with open circles; the thick lines connect the nearest-neighbor weights. The borders of the Voronoi polygons, corresponding to the weights, are indicated with thin lines. *Right panel:* Induced Delaunay triangularization. The induced triangularization is obtained by masking the original triangularization with the input data distribution (two disconnected gray shaded regions). (Fritzke, 1995a, reprinted with permission)

In order to position the weight vectors in the input space, Martinetz and Schulten (1991) have proposed a particular kind of vector quantization method, called Neural Gas (NG). The main principle of NG is: for each input $\mathbf{v}$ update the k nearest-neighbor neuron weight vectors, with k decreasing over time until only the winning

neuron's weight vector is updated. Hence, we have a neighborhood function but now in input space. The learning rate also follows a decay schedule. Note that the NG by itself does not delete or insert any neurons. The NG requires a fine tuning of the rate at which the neighborhood shrinks to achieve a smooth convergence and proper modeling of the data manifold.

The combination of CHL and NG is an effective method for topology learning. The evolution of the lattice is shown in Fig. 1.14 for a data manifold that consists of three-, two- and one-dimensional subspaces (Martinetz and Schulten, 1991). We see that the lattice successfully has filled and adapted its dimensionality to the different subspaces. For this reason, visualization is only possible for low-dimensional input spaces (hence, it is not suited for data visualization purposes where a mapping from a potentially high-dimensional input space to a low-dimensional lattice is desired). A problem with the algorithm is that one needs to decide *a priori* the number of neurons, as it required by the NG algorithm (Fritzke, 1996): depending on the complexity of the data manifold, very different numbers may be appropriate. This problem is overcome in the Growing Neural Gas (GNG; Fritzke, 1995a) (see next subsection).

1.5.2 Growing neural gas

Contrary to CHL/NG, the growing neural gas (GNG) poses no explicit constraints on the lattice. The lattice is generated, and constantly updated, by the competitive Hebbian learning technique (CHL, see above; Martinetz, 1993). The algorithm starts with two randomly placed, connected neurons (Fig. 1.15, left panel). Unlike the CHL/NG algorithm, after a fixed number λ of time steps, the neuron i with the largest accumulated error is determined and a new neuron inserted between i and one of its neighbors. Hence, the GNG algorithm exploits the topology to position new neurons between existing ones, whereas in the CHL/NG, the topology is not influenced by the NG algorithm. Error variables are locally redistributed and another λ time steps is performed. The lattice generated is a subgraph of a Delaunay triangularization, and can have different dimensionalities in different regions of the data manifold. The end-result is very similar to CHL/NG (Fig. 1.15, right panel).

1.5.3 Growing cell structures

In the growing cell structures (GCS) algorithm (Fritzke, 1994), the model consists of hypertetrahedrons (or simplices) of a dimensionality chosen in advance (hence, the lattice dimensionality is fixed). Note that a d_A-dimensional hypertetrahedron has d_A+1 vertices, with d_A the lattice dimensionality, and $d_A \leq d$, with d the input space dimensionality. Examples for $d = 1, 2$ and 3 are a line, a triangle and a tetrahedron, respectively.

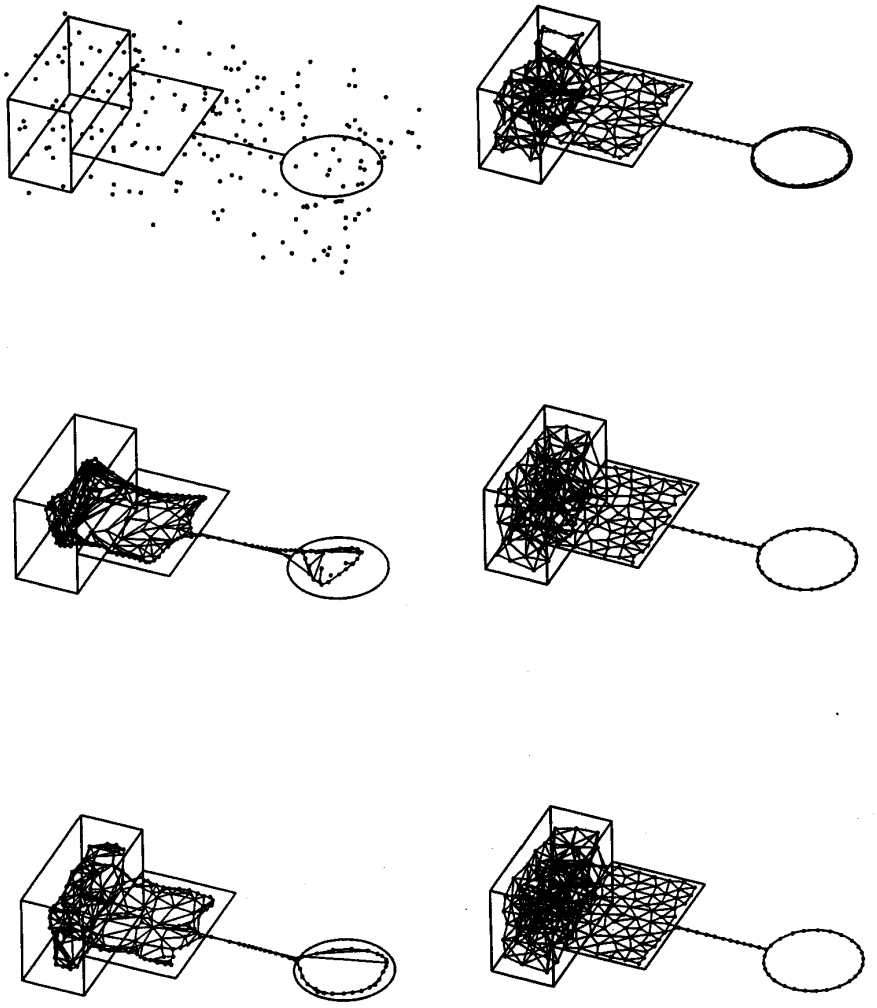

Fig. 1.14 Neural Gas algorithm, combined with competitive Hebbian learning, applied to a data manifold consisting of a right parallelepiped, a rectangle and a circle connecting a line. The dots indicate the positions of the neuron weights. Lines connecting neuron weights indicate lattice edges. Shown are the initial result (top left), and further the lattice after 5000, 10,000, 15,000, 25,000 and 40,000 time steps (top-down the first column, then top-down the second column). (Martinetz and Schulten, 1991, reprinted with permission.)

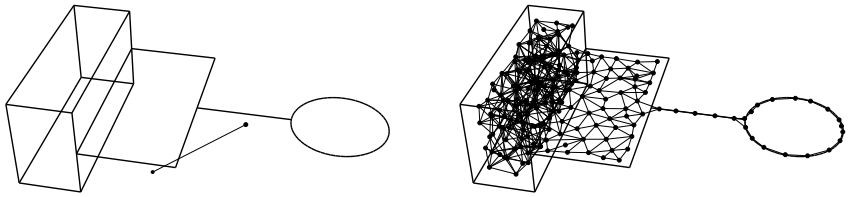

Fig. 1.15 Growing Neural Gas algorithm applied to the same data configuration as in Fig. 1.14. Initial lattice (left panel) and lattice after 20,000 time steps (right panel). Note that the last one is not necessarily the final result because the algorithm could run indefinitely. (Fritzke, 1995a, reprinted with permission)

The model is initialized with exactly one hypertetrahedron. Always after a prespecified number of time steps, the neuron i with the maximum accumulated error is determined and a new neuron is inserted by splitting the longest of the edges emanating from i. Additional edges are inserted to rebuild the structure in such a way that it consists only of d_A-dimensional hypertetrahedrons: Let the edge which is split connect neurons i and j, then the newly inserted neuron should be connected to i and j and with all *common* topological neighbors of i and j.

Since the GCS algorithm assumes a fixed dimensionality for the lattice, it can be used for generating a dimensionality-reducing mapping from the input space to the lattice space, which is useful for data visualization purposes.

1.5.4 Growing grid

In the growing grid algorithm (GG; Fritzke, 1995b) the lattice is a rectangular grid of a certain dimensionality d_A. The starting configuration is a d_A-dimensional hypercube, *e.g.*, a 2×2 lattice for $d_A = 2$, a $2 \times 2 \times 2$ lattice for $d_A = 3$, and so on. To keep this structure consistent, it is necessary to always insert complete (hyper-)rows and (hyper-)columns. Since the lattice dimensionality is fixed, and possibly much smaller than the input space dimensionality, the GG is useful for data visualization.

Apart from these differences, the algorithm is very similar to the ones described above. After λ time steps, the neuron with the largest accumulated error is determined, and the longest edge emanating from it is identified, and a new complete hyper-row or -column is inserted such that the edge is split.

1.5.5 Other algorithms

There exists a wealth of other algorithms, such as the Dynamic Cell Structures (DCS) (Bruske and Sommer, 1995), which is similar to the GNG, the Growing Self-Organizing Map (GSOM, also called Hypercubical SOM) (Bauer and Villmann,

1997), which has some similarities to GG but it adapts the lattice dimensionality, Incremental Grid Growing (IGG) which introduces new neurons at the lattice border and adds/removes connections based on the similarities of the connected neurons' weight vectors (Blackmore and Miikkulainen, 1993), and one that is also called the Growing Self-Organizing Map (GSOM) (Alahakoon *et al.*, 2000), which also adds new neurons at the lattice border, similar to IGG, but does not delete neurons, and which contains a spread factor to let the user control the spread of the lattice, to name a few.

In order to study and exploit hierarchical relations in the data, hierarchical versions of some of these algorithms have been developed. For example, the Growing Hierarchical Self-Organizing Map (GHSOM) (Rauber *et al.*, 2002), develops lattices at each level of the hierarchy using the GG algorithm (insertion of columns or rows). The orientation in space of each lattice is similar to that of the parent lattice, which facilitates the interpretation of the hierarchy, and which is achieved through a careful initialization of each lattice. Another example is Adaptive Hierarchical Incremental Grid Growing (AHIGG; Merkl *et al.*, 2003) of which the hierarchy consists of lattices trained with the IGG algorithm, and for which new units at a higher level are introduced when the local (quantization) error of a neuron is too large.

1.6 Recurrent Topographic Maps

1.6.1 Time series

Many data sources such as speech have a temporal characteristic (*e.g.*, a correlation structure) that cannot be sufficiently captured when ignoring the order in which the data points arrive, as in the original SOM algorithm. Several self-organizing map algorithms have been developed for dealing with sequential data, such as the ones using:

- fixed-length windows, *e.g.*, the time-delayed SOM (Kangas, 1990), among others (Martinetz *et al.*, 1993; Simon *et al.*, 2003; Vesanto, 1997);
- specific sequence metrics (Kohonen, 1997; Somervuo, 2004);
- statistical modeling incorporating appropriate generative models for sequences (Bishop *et al.*, 1997; Tiňo *et al.*, 2004);
- mapping of temporal dependencies to spatial correlation, *e.g.*, as in traveling wave signals or potentially trained, temporally activated lateral interactions (Euliano and Principe, 1999; Schulz and Reggia, 2004 Wiemer, 2003);
- recurrent processing of time signals and recurrent winning neuron computation based on the current input and the previous map activation, such as with the Temporal Kohonen map (TKM) (Chappell and Taylor, 1993), the recurrent SOM (RSOM) (Koskela *et al.*, 1998), the recursive SOM (RecSOM) (Voegtlin, 2002), the SOM for structured data (SOMSD) (Hagenbuchner *et al*, 2003), and the Merge SOM (MSOM) (Strickert and Hammer, 2005).

Several of these algorithms have been proposed recently, which shows the increased interest in representing time series with topographic maps. For some of these algorithms, also tree structured data can be represented (see later). We focus on the recurrent processing of time signals and briefly describe the models listed above. A more detailed overview can be found elsewhere (Barreto and Araújo, 2001; Hammer *et al.*, 2005). The recurrent algorithms essentially differ in the context, *i.e.*, the way by which sequences are internally represented.

1.6.1.1 Overview of algorithms

The TKM extends the SOM algorithm with recurrent self-connections of the neurons, such that they act as leaky integrators (Fig. 1.16A). Given a sequence $[\mathbf{v}_1,\ldots,\mathbf{v}_t]$, $\mathbf{v}_j \in \mathbb{R}^d, \forall j$, the integrated distance ID_i of neuron i with weight vector $\mathbf{w}_i \in \mathbb{R}^d$ is:

$$ID_i(t) = \alpha\|\mathbf{v}_t - \mathbf{w}_i\|^2 + (1-\alpha)ID_i(t-1), \tag{1.17}$$

with $\alpha \in (0,1)$ a constant determining the strength of the context information, and with $ID_i(0) \stackrel{\Delta}{=} 0$. The winning neuron is selected as $i^*(t) = \arg\min_I ID_i(t)$, after which the network is updated as in the SOM algorithm. Equation (1.17) has the form of a leaky integrator, integrating previous distances of neuron i, given the sequence.

The RSOM uses in essence the same dynamics, however, it integrates over the directions of the individual weight components:

$$ID_{ij}(t) = \alpha(v_{jt} - \mathbf{w}_i) + (1-\alpha)ID_{ij}(t-1), \tag{1.18}$$

so that the winner is then the neuron for which $\|[ID_{ij}(t)]\|^2$ is the smallest. It is clear that this algorithm stores more information than the TKM. However, both the TKM and the RSOM compute only a leaky average of the time series and they do not use any explicit context.

The RecSOM is an algorithm for sequence prediction. A given sequence is recursively processed based on the already computed context. Hereto, each neuron i is equipped with a weight and, additionally, a context vector $\mathbf{c}_i \in \mathbb{R}^N$ which stores an activation profile of the whole map, indicating in which context the weight vector should arise (Fig. 1.16B). The integrated distance is defined as:

$$ID_i(t) = \alpha\|\mathbf{v}_t - \mathbf{w}_i\|^2 + \beta\|\mathbf{y}(t-1) - \mathbf{c}_i\|^2, \tag{1.19}$$

with $\mathbf{y}(t-1) = [\exp(-ID_1(t-1)),\ldots,exp(-ID_N(t-1))]$, $\alpha,\beta > 0$ constants to control the respective contributions from pattern and context matching, and with $ID_i(0) \stackrel{\Delta}{=} 0$. The winner is defined as the neuron for which the integrated distance is minimal. The equation contains the exponential function in order to avoid numerical explosion: otherwise, the activation ID_i could become too large because the distances with respect to the contexts of all N neurons could accumulate. Learning is performed on the weights as well as the contexts, in the usual way (thus, involv-

ing a neighborhood function centered around the winner): the weights are adapted towards the current input sequences; the contexts towards the recursively computed contexts $\mathbf{y}$.

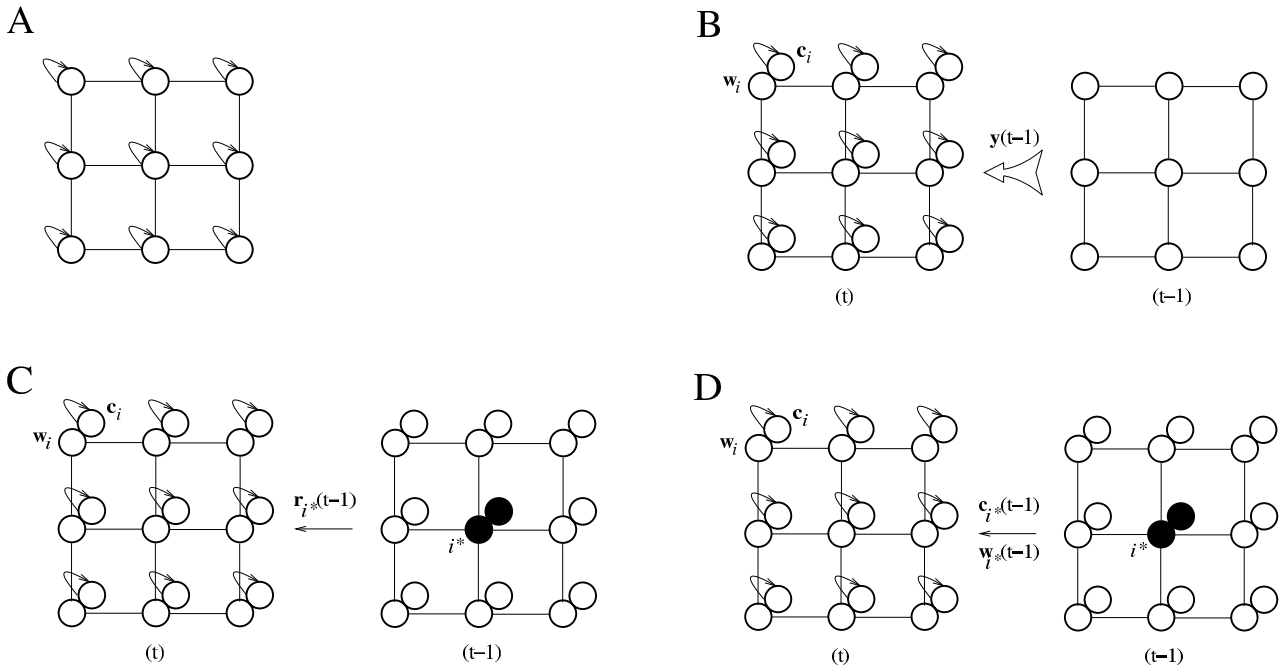

Fig. 1.16 Schematic representation of four recurrent SOM algorithms: TKM (A), RecSOM (B), SOMSD (C), and MSOM (D). Recurrent connections indicate leaky integration; double circles indicate the neuron's weight- and context vectors; i^* and the filled circles indicate the winning neuron; (t) and $(t-1)$ represent the current and the previous time steps, respectively.

The SOMSD has been developed for processing labeled trees with fixed fan-out k. The limiting case of $k = 1$ covers sequences. We further restrict ourselves to sequences. Each neuron has, besides a weight, also a context vector $\mathbf{c}_i \in \mathbb{R}^{d_A}$, with d_A the dimensionality of the lattice. The winning neuron i^* for a training input at time t is defined as (Fig. 1.16C):

$$i^* = \arg\min \alpha \|\mathbf{v}_t - \mathbf{w}_i\|^2 + (1-\alpha)\|r_{i^*(t-1)} - \mathbf{c}_i\|^2, \tag{1.20}$$

with r_{i^*} the lattice coordinate of the winning neuron. The weights $\mathbf{w}_i$ are moved in the direction of the current input, as usual (*i.e.*, with a neighborhood), and the contexts $\mathbf{c}_i$ in the direction of the lattice coordinates of the winning neuron of the previous time step (also with a neighborhood).

The MSOM algorithm accounts for the temporal context by an explicit vector attached to each neuron which stores the preferred context of that neuron (Fig. 1.16D). The MSOM characterizes the context by a "merging" of the weight and the context of the winner in the previous time step (whence the algorithm's name: Merge SOM). The integrated distance is defined as:

$$ID_i(t) = \alpha\|\mathbf{w}_i - \mathbf{v}_t\|^2 + (1-\alpha)\|\mathbf{c}_i - \mathbf{C}_t\|^2, \tag{1.21}$$

with $\mathbf{c}_i \in \mathbb{R}^d$, and with $\mathbf{C}_t$ the expected (merged) weight/context vector, *i.e.*, the context of the previous winner:

$$\mathbf{C}_t = \gamma\mathbf{c}_{i^*(t-1)} + (1-\gamma)\mathbf{w}_{i^*(t-1)}, \tag{1.22}$$

with $\mathbf{C}_0 \stackrel{\Delta}{=} 0$. Updating of $\mathbf{w}_i$ and $\mathbf{c}_i$ are then done in the usual SOM way, thus, with a neighborhood function centered around the winner. The parameter α is controlled so as to maximize the entropy of the neural activity.

1.6.1.2 Comparison of algorithms

Hammer and co-workers (2004) pointed out that several of the mentioned recurrent self-organizing map algorithms share their principled dynamics, but differ in their internal representations of context. In all cases, the context is extracted as the relevant part of the activation of the map in the previous time step. The notion of "relevance" thus differs between the algorithms (see also Hammer *et al.*, 2005). The recurrent self-organizing algorithms can be divided in two categories: the representation of the context in the data space, such as for the TKM and MSOM, and in a space that is related to the neurons, as for SOMSD and RecSOM. In the first case, the storage capacity is restricted by the input dimensionality. In the latter case, it can be enlarged simply by adding more neurons to the lattice. Furthermore, there are essential differences in the dynamics of the algorithms. The TKM does not converge to the optimal weights; RSOM does it but the parameter α occurs both in the encoding formula and in the dynamics. In the MSOM algorithm they can be controlled separately. Finally, the algorithms differ in memory and computational complex-

ity (RecSOM is quite demanding, SOMSD is fast and MSOM is somewhere in the middle), the possibility to apply different lattice types (such as hyperbolic lattices, Ritter, 1998), and their capacities (MSOM and SOMSD achieve the capacity of Finite State Automata, but TKM and RSOM have smaller capacities; RecSOM is more complex to judge).

As an example, Voegtlin (2002) used the Mackey-Glass time series, a well known one-dimensional time-delay differential equation, for comparing different algorithms:

$$\frac{\mathrm{d}v}{\mathrm{d}t} = bv(t) + \frac{av(t-\tau)}{1+v(t-\tau)^{10}}, \tag{1.23}$$

which for $\tau > 16.8$ generates a chaotic time series. Voegtlin used $a = 0.2$, $b = -0.1$ and $\tau = 17$. A sequence of values is plotted in Fig. 1.17, starting from uniform input conditions. For training, the series is sampled every 3 time units. This example was also taken up by (Hammer *et al.*, 2004) for comparing their MSOM. Several 10×10 maps were trained using 150,000 iterations; note that the input dimensionality $d = 1$ in all cases. Fig. 1.18 shows the quantization error plotted as a function of the index of the past input (index= 0 means the present). The error is expressed in terms of the average standard deviation of the given sequence and the winning neuron's receptive field over a window of 30 time steps (*i.e.*, delay vector). We observe large fluctuations for the SOM, which is due to the temporal regularity of the series and the absence of any temporal coding by the SOM algorithm. We also observe that the RSOM algorithm is not really better than the SOM algorithm. On the contrast, the RecSOM, SOMSD and MSOM algorithms (the MSOM was trained with a Neural Gas neighborhood function, for details see Strickert and Hammer, 2003a) display a slow increase in error as a function of the past, but with a better performance for the MSOM algorithm.

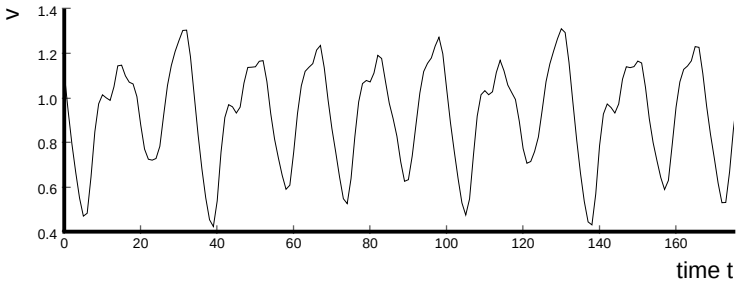

Fig. 1.17 Excerpt from the Mackey-Glass chaotic time series. (Strickert and Hammer, 2003b, reprinted with permission)

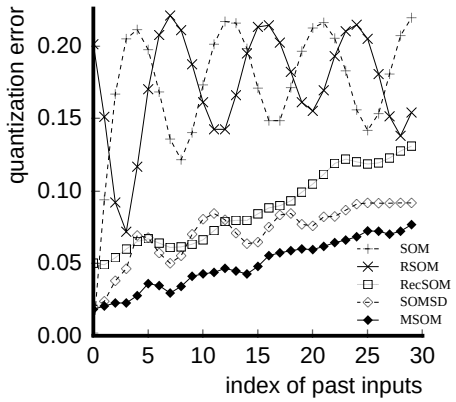

Fig. 1.18 Temporal quantization error of different algorithms for the Mackey-Glass time series plotted as a function of the past (index= 0 is present). (Strickert and Hammer, 2003b, reprinted with permission)

1.6.2 Tree structures

Binary trees, and also trees with limited fan-out k, have been successfully processed with the SOMSD and the MSOM by extending the neuron's single context vector to several context vectors (one for each subtree). Starting from the leafs of a tree, the integrated distance ID of a tree with a given label and the k subtrees can be determined, and the context defined. The usual learning can then be applied to the weights and contexts. As a result of learning, a topographic mapping of trees according to their structure and labels arises. Up to now, only preliminary results of the capacities of these algorithms for tree structures have been obtained.

1.7 Kernel Topographic Maps

Rather than developing topographic maps with disjoint and uniform activation regions (Voronoi tessellation), such as in the case of the SOM algorithm (Fig. 1.5), and its adapted versions, algorithms have been introduced that can accommodate neurons with overlapping activation regions, usually in the form of kernel functions, such as Gaussians (Fig. 1.19). For these *kernel-based topographic maps*, or *kernel topographic maps*, as they are called (they are also sometimes called *probabilistic topographic maps* since they model the input density with a kernel mixture), several learning principles have been proposed (for a review, see Van Hulle, 2009). One motivation to use kernels is to improve, besides the biological relevance, the density estimation properties of topographic maps. In this way, we can combine the unique visualization properties of topographic maps with an improved modeling of clusters in the data. Usually, homoscedastic (equal-variance) Gaussian kernels are

used, but also heteroscedastic (differing variances) Gaussian kernels and other kernel types have been adopted. In the next sections, we will review the kernel-based topographic map formation algorithms and mention a number of applications. The diversity in algorithms reflects the differences in strategies behind them. As a result, these algorithms have their specific strengths (and weaknesses) and, thus, their own application types.

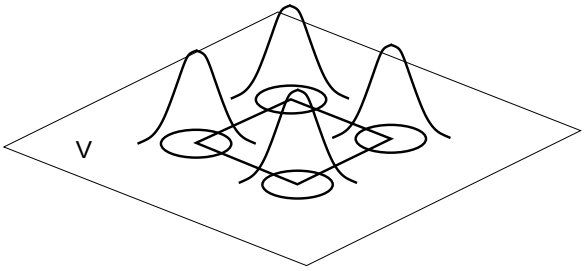

Fig. 1.19 Kernel-based topographic map. Example of a 2×2 map (*cf.* rectangle with thick lines in V-space) for which each neuron has a Gaussian kernel as output function. Normally, a more condensed representation is used where, for each neuron, a circle is drawn with center the neuron weight vector and radius the kernel range.

1.7.1 SOM algorithm revisited

The starting point is again Kohonen's SOM algorithm. To every neuron a homoscedastic Gaussian kernel is associated with center corresponding to the neuron's weight vector. Kostiainen and Lampinen (2002) showed that the SOM algorithm can be seen as the equivalent of a maximum likelihood procedure applied to a homoscedastic Gaussian mixture density model, but with the exception that a winner neuron (and, thus, kernel) is selected (the definition of the "winner" i^* eq. (1.1) is equivalent to looking for the Gaussian kernel with the largest output). The position of the winner's kernel is then updated, and possibly also those of other kernels, given the neighborhood function. In a traditional maximum likelihood procedure, there are no winners, and all kernels are updated (Redner and Walker, 1984). This means that, *e.g.*, for a vanishing neighborhood range, a Gaussian kernel's center is only updated when that neuron is the winner, hence, contrary to the classical case of Gaussian mixture density modeling, the tails of the Gaussian kernels do not lead to center updates (they disappear "under" other kernels), which implies that the kernel radii will be underestimated.

1.7.2 Elastic net

Durbin and Willshaw's elastic net (1987) can be considered as one of the first accounts on kernel-based topographic maps. The elastic net was used for solving the Traveling Salesman Problem (TSP). In TSP, the objective is to find the shortest, closed tour that visits each city once and that returns to its starting point (*e.g.*, the right panel in Fig. 1.20). When we represent the location of each city by a point $\mathbf{v}^\mu$ in the two-dimensional input space $V \subseteq \mathbb{R}^2$, and a tour by a sequence of N neurons – which comprise a ring or closed chain A–, then a solution to the TSP can be envisaged as a mapping from V-space onto the neurons of the chain. Evidently, we expect the neuron weights to coincide with the input points ("cities") at convergence.

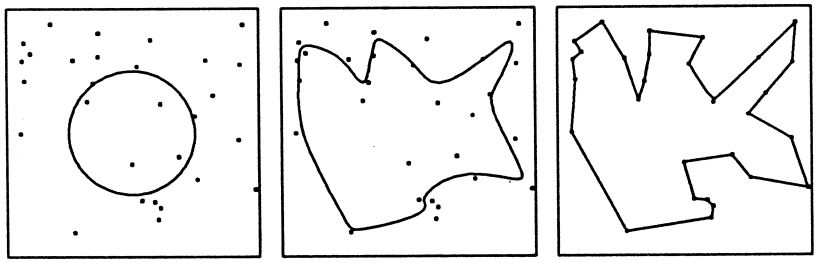

Fig. 1.20 One dimensional topographic map used for solving the Traveling Salesman Problem. The lattice has a ring topology (closed chain); the points represent cities and are chosen randomly from the input distribution demarcated by the square box. The evolution of the lattice is shown for three time instants, at $t = 0$ (initialization), 7000 and 10,000 (from left to right). The weights of the lattice at $t = 0$ form a circle positioned at the center of mass of the input distribution. (Reprinted from Ritter and Schulten, 1988, ©1988 IEEE.)

The algorithm of the elastic net can be written as follows (in our format):

$$\Delta \mathbf{w}_i = 2\eta \left(\sum_\mu \Lambda^\mu(i)(\mathbf{v}^\mu - \mathbf{w}_i) + \kappa(\mathbf{w}_{i+1} - 2\mathbf{w}_i + \mathbf{w}_{i-1}) \right), \quad \forall i, \tag{1.24}$$

where each weight $\mathbf{w}_i$ represents a point on the elastic net. The first term on the right hand side is a force that drags each point $\mathbf{w}_i$ on the chain A towards the cities $\mathbf{v}^\mu$, and the second term is an elastic force that tends to keep neighboring points on the chain close to each other (and thus tends to minimize the overall tour length). The function $\Lambda^\mu(i)$ is a *normalized* Gaussian:

$$\Lambda^\mu(i) = \frac{\exp(-\|\mathbf{v}^\mu - \mathbf{w}_i\|^2/2\sigma_\Lambda^2)}{\sum_j \exp(-\|\mathbf{v}^\mu - \mathbf{w}_j\|^2/2\sigma_\Lambda^2)}, \tag{1.25}$$

with $\mathbf{w}_i$ the center of the Gaussian and σ_Λ its range, which is gradually decreased over time (as well as η, and also κ). By virtue of this kernel, the elastic net can be

viewed as a homoscedastic Gaussian mixture density model, fitted to the data points by a penalized maximum likelihood term (for a formal account, see Durbin *et al.*, 1989). The elastic net algorithm looks similar to Kohonen's SOM algorithm except that $\Lambda(i,j)$ has been replaced by $\Lambda^{\mu}(i)$, and that a second term is added. Interestingly, the SOM algorithm can be used for solving the TSP even without the second term (Ritter *et al.*, 1992), provided that we take more neurons in our chain than cities, and that we initialize the weights on a circle (a so-called N-gon) positioned at the center of mass of the input distribution. An example of the convergence process for a 30 city case using a $N = 100$ neuron chain is shown in Fig. 1.20.

The elastic net has been used for finding trajectories of charged particles with multiple scattering in high energy physics experiments (Gorbunov and Kisel, 2006), and for predicting the protein folding structure (Ball *et al.*, 2002). Furthermore, it has been used for clustering applications (Rose *et al.*, 1993). Finally, since it also has a close relationship with "snakes" in computer vision (Kass *et al.*, 1987) (for the connection, see Abrantes and Marques, 1995), the elastic net has also been used for extracting the shape of a closed object from a digital image, such as finding the lung boundaries from magnetic resonance images (Gilson *et al.*, 1997).

1.7.3 Generative topographic map

The Generative Topographic Map (GTM) algorithm (Bishop *et al.*, 1996; 1998) develops a topographic map that attempts to find a representation for the input distribution $p(\mathbf{v})$, $\mathbf{v} = [v_1, \ldots, v_d]$, $\mathbf{v} \in V$, in terms of a number L of latent variables $\mathbf{x} = [x_1, \ldots, x_L]$. This is achieved by considering a non-linear transformation $\mathbf{y}(\mathbf{x}, \mathbf{W})$, governed by a set of parameters $\mathbf{W}$, which maps points in the latent variable space to the input space, much the same way as the lattice nodes in the SOM relate to positions in V-space (inverse mapping Ψ in Fig. 1.2). If we define a probability distribution $p(\mathbf{x})$ on the latent variable space, then this will induce a corresponding distribution $p(\mathbf{y}|\mathbf{W})$ in the input space.

As a specific form of $p(\mathbf{x})$, Bishop and co-workers take a discrete distribution consisting of a sum of delta functions located at the N nodes of a regular lattice:

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x}_i). \tag{1.26}$$

The dimensionality L of the latent variable space is typically less than the dimensionality d of the input space so that the transformation $\mathbf{y}$ specifies an L-dimensional manifold in V-space. Since $L < d$, the distribution in V-space is confined to this manifold and, hence, is singular. In order to avoid this, Bishop and co-workers introduced a noise model in V-space, namely, a set of radially-symmetric Gaussian kernels centered at the positions of the lattice nodes in V-space. The probability distribution in V-space can then be written as follows:

$$p(\mathbf{v}|\mathbf{W},\sigma) = \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{v}|\mathbf{x}_i,\mathbf{W},\sigma), \tag{1.27}$$

which is a homoscedastic Gaussian mixture model. In fact, this distribution is a *constrained* Gaussian mixture model since the centers of the Gaussians cannot move independently from each other but are related through the transformation $\mathbf{y}$. Moreover, when the transformation is smooth and continuous, the centers of the Gaussians will be topographically ordered by construction. Hence, the topographic nature of the map is an intrinsic feature of the latent variable model and is not dependent on the details of the learning process. Finally, the parameters $\mathbf{W}$ and σ are determined by maximizing the log-likelihood:

$$\ln \mathscr{L}(\mathbf{W},\sigma) = \ln \prod_{\mu=1}^{M} p(\mathbf{v}^{\mu}|\mathbf{W},\sigma), \tag{1.28}$$

and which can be achieved through the use of an Expectation-Maximization (EM) procedure (Dempster *et al.*, 1977). Because a single two-dimensional visualization plot may not be sufficient to capture all of the interesting aspects of complex data sets, a hierarchical version of the GTM has also been developed (Tiňo and Nabney, 2002).

The GTM has been applied to visualizing oil flows along multi-phase pipelines, where the phases are oil, water and gas, and the flows can be one of 3 types, stratified, homogeneous and annular (Bishop *et al.*, 1996) (Fig. 1.4, right panel). It has been applied to visualizing electropalatographic (EPG) data for investigating the activity of the tongue in normal and pathological speech (Carreira-Perpiñán and Renals, 1998) (Fig. 1.21). It has also been applied to the classification of *in vivo* magnetic resonance spectra of controls and Parkinson patients (Axelson *et al.*, 2002), to word grouping in document data sets (using the newsgroup data set benchmark) and the exploratory analysis of web navigation sequences (Kabán, 2005), and to spatio-temporal clustering of transition states of a typhoon from image sequences of cloud patterns (Kitamoto, 2002). In another application, the GTM is used for micro-array data analysis (gene expression data) with the purpose of finding low-confidence value genes (D'Alimonte *et al.*, 2005).

1.7.4 Regularized Gaussian mixture modeling

Tom Heskes (2001) was able to show the direct correspondence between minimum distortion topographic map formation and maximum likelihood Gaussian mixture density modeling for the homoscedastic case. The starting point was the traditional distortion (vector quantization) formulation of the self-organizing map:

$$F_{\text{quantization}} = \sum_{\mu}\sum_{i} P(i|\mathbf{v}^{\mu}) \sum_{j} \Lambda(i.j) \frac{1}{2} \|\mathbf{v}^{\mu} - \mathbf{w}_j\|^2, \tag{1.29}$$

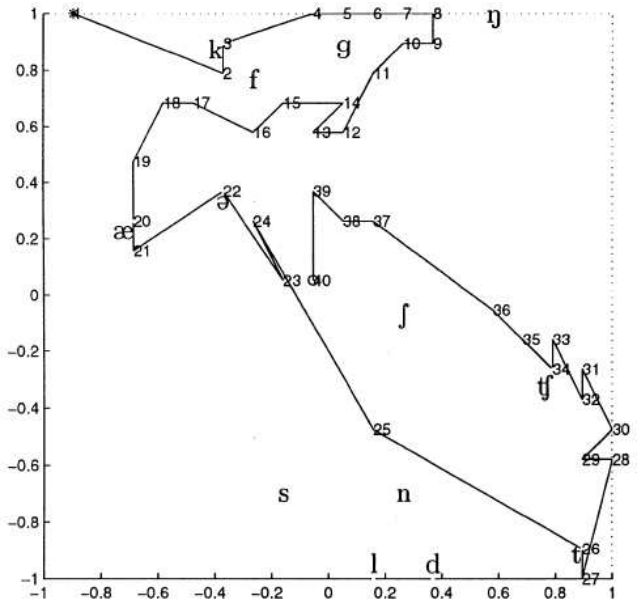

Fig. 1.21 Visualization of the trajectory in a 20×20 GTM lattice of the activity of the tongue (electropalatographic (EPG) data) of speaker RK for the utterance fragment "I prefer *Kant to* Hobbes for a good bedtime book" (Carreira-Perpiñán and Renals, 1998, reprinted with permission.)

with $P(i|\mathbf{v}^\mu)$ the probability that input $\mathbf{v}^\mu$ is assigned to neuron i with weight $\mathbf{w}_i$ (*i.e.*, the posterior probability, and with $\sum_i P(i|\mathbf{v}^\mu) = 1$ and $P(i|\mathbf{v}^\mu) \geq 0$). Even if we assign $\mathbf{v}^\mu$ to neuron i, there exists a confusion probability $\Lambda(i,j)$ that $\mathbf{v}^\mu$ is assigned to neuron j. An annealed version of the self-organizing map is obtained if we add an entropy term:

$$F_{\text{entropy}} = \sum_\mu \sum_i P(i|\mathbf{v}^\mu) \log\left(\frac{P(i|\mathbf{v}^\mu)}{Q_i}\right), \tag{1.30}$$

with Q_i the prior probability (the usual choice is $Q_i = \frac{1}{N}$, with N the number of neurons in the lattice. The final (free) energy is now:

$$F = \beta F_{\text{quantization}} + F_{\text{entropy}}, \tag{1.31}$$

with β playing the role of an inverse temperature. This formulation is very convenient for an EM procedure. The expectation step leads to:

$$P(i|\mathbf{v}^\mu) = \frac{Q_i \exp\left(-\frac{\beta}{2}\sum_j \Lambda(i,j)\,\|\mathbf{v}^\mu - \mathbf{w}_j\|\right)}{\sum_s Q_s \exp\left(-\frac{\beta}{2}\sum_j \Lambda(s,j)\,\|\mathbf{v}^\mu - \mathbf{w}_j\|\right)}, \tag{1.32}$$

and the maximization step to:

$$\mathbf{w}_i = \frac{\sum_\mu \sum_j P(j|\mathbf{v}^\mu)\Lambda(j,i)\mathbf{v}^\mu}{\sum_\mu \sum_j P(j|\mathbf{v}^\mu)\Lambda(j,i)}, \tag{1.33}$$

which is also the result reached by Graepel and co-workers (1998) for the Soft Topographic Vector Quantization (STVQ) algorithm (see the next section). Plugging eq. (1.32) into eq. (1.31) leads to an error function, which allows for the connection with a maximum likelihood procedure, for a mixture of homoscedastic Gaussians, when the neighborhood range vanishes ($\Lambda(i,j) = \delta_{ij}$). When the neighborhood is present, Heskes showed that this leads to a term added to the original likelihood.

As an application, Heskes considers market basket analysis. Given are a list of transactions corresponding to the joint set of products purchased by a customer at a given time. The goal of the analysis is to map the products onto a two-dimensional map (lattice) such that neighboring products are "similar". Similar products should have similar conditional probabilities of buying other products. In another application, he considers the case of transactions in a supermarket. The products are summarized in product groups and given are the co-occurrence frequencies. The result is a two-dimensional density map showing clusters of products that belong together, *e.g.*, a large cluster of household products (Fig. 1.22).

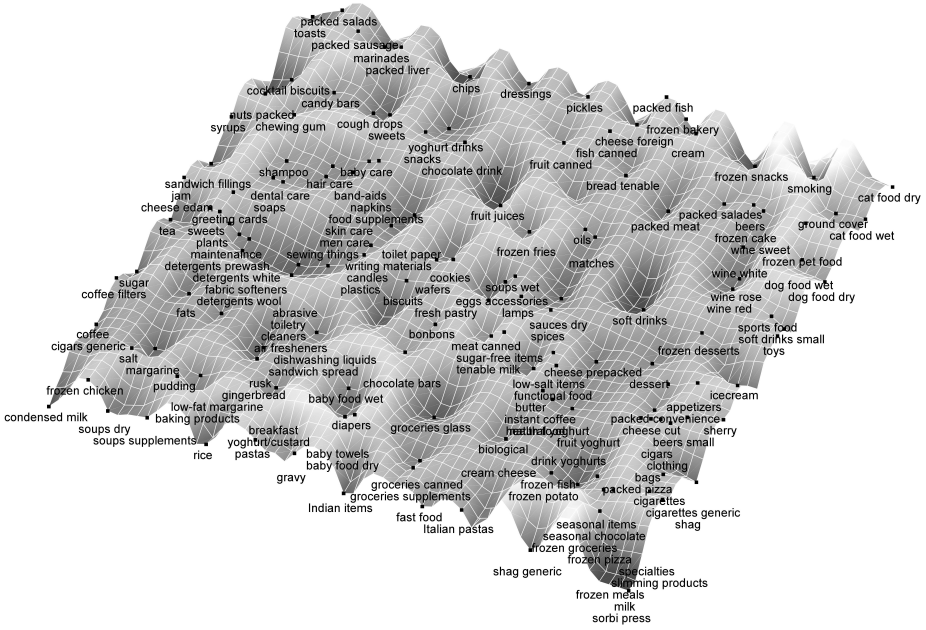

Fig. 1.22 Visualization of market basket data in which 199 product groups are clustered based on their co-occurrence frequencies with other products (Heskes, 2001). (©2001 IEEE)

1.7.5 Soft topographic vector quantization

Another approach that considers topographic map formation as an optimization problem, is the one introduced by Klaus Obermayer and co-workers (Graepel *et al.*, 1997,1998). They start from the following cost function:

$$E(\mathbf{W}) = \frac{1}{2}\sum_{\mu}\sum_{i} c_{\mu,i} \sum_{j} \Lambda(i,j)\,\|\mathbf{v}^{\mu} - \mathbf{w}_j\|^2, \tag{1.34}$$

with $c_{\mu,i} \in \{0,1\}$ and for which $c_{\mu,i} = 1$ if $\mathbf{v}^{\mu}$ is assigned to neuron i, else $c_{\mu,i} = 0$ ($\sum_i c_{\mu,i} = 1$); the neighborhood function obeys $\sum_j \Lambda(i,j) = 1$. The $\mathbf{w}_i$, $\forall i$, for which this function is minimal, are the optimal ones. However, the optimization is a difficult task, because it depends both on binary and continuous variables and has many local minima. To avoid this, a technique known as deterministic annealing is applied: the optimization is done on a smooth function parametrized by a parameter β, the so-called free energy. When β is small, the function is smooth and only one global minimum remains; when large, more of the structure of the original cost function is reflected in the free energy. One starts with a low value of β and attempts to keep track of the minimum through higher values of β.

The application of the principle of maximum entropy yields the free energy (Graepel *et al.*, 1997):

$$F = -\frac{1}{\beta}\log\sum_{c_{\mu,i}} \exp(-\beta E), \tag{1.35}$$

which leads to probabilistic assignments of inputs $\mathbf{v}^{\mu}$ to neurons, $P(i|\mathbf{v}^{\mu})$, $\forall i$, that is, the posterior probabilities, and which are given by:

$$P(i|\mathbf{v}^{\mu}) = \frac{\exp(-\frac{\beta}{2}\sum_j \Lambda(i,j)\|\mathbf{v}^{\mu} - \mathbf{w}_j\|^2)}{\sum_s \exp(-\frac{\beta}{2}\sum_j \Lambda(s,j)\|\mathbf{v}^{\mu} - \mathbf{w}_j\|^2)}. \tag{1.36}$$

The fixed point rule for the kernel centers is then:

$$\mathbf{w}_i = \frac{\sum_{\mu}\sum_j P(j|\mathbf{v}^{\mu})\Lambda(j,i)\mathbf{v}^{\mu}}{\sum_{\mu}\sum_j P(j|\mathbf{v}^{\mu})\Lambda(j,i)}, \quad \forall i. \tag{1.37}$$

The updates are done through an EM scheme. We observe that the latter equation is identical to Heskes' rule for regularized Gaussian mixture modeling, eq. (1.33).

The STVQ has been generalized to the soft topographic mapping for proximity data (STMP), which can be used for clustering categorical data, given a matrix of pairwise proximities or dissimilarities, which is one of the first accounts of this nature in the topographic map literature. A candidate application are the DNA micro-array data sets where the data can be described by matrices with the columns representing tissue samples and the rows genes, and the entries in the matrix cor-

respond to the strength of the gene expression. In (Seo and Obermayer, 2004), a modified version of the STMP is used for clustering documents ("document map").

1.7.6 Heteroscedastic Gaussian kernel topographic map formation

In the literature, only few approaches exist that consider heteroscedastic kernels, perhaps because the kernel radius in the homoscedastic case is often used in an annealing schedule, as shown above in the STVQ-, and the Elastic Net algorithms. When using heteroscedastic kernels, a better density estimate is expected. Several algorithms for heteroscedastic kernels have been developed (for a review, see Van Hulle, 2009). We briefly mention a few here.

Bearing in mind what we have said earlier about the SOM in connection to Gaussian mixture modeling, one can extend the original batch map, eq. (1.5), to the heteroscedastic case (Van Hulle, 2009):

$$\begin{aligned} \mathbf{w}_i &= \frac{\sum_\mu \Lambda(i^*,i)\mathbf{v}^\mu}{\sum_\mu \Lambda(i^*,i)}, \\ \sigma_i^2 &= \frac{\sum_\mu \Lambda(i^*,i)\|\mathbf{v}-\mathbf{w}_i\|^2/d}{\sum_\mu \Lambda(i^*,i)}, \quad \forall i, \end{aligned} \tag{1.38}$$

with $i^* = \text{argmax}_i K_i$ (which is no longer equivalent to $i^* = \text{argmin}_i \|\mathbf{v}-\mathbf{w}_i\|$, but which is required since we now have heteroscedastic kernels), *i.e.*, an *activity*-based definition of "winner-takes-all", rather than a minimum *Euclidean distance*-based one. Notice again that, by the definition of the winner, the tails of the kernels are cut off, since the kernels overlap.

Recently, we introduced (Van Hulle, 2005a) a learning algorithm for kernel-based topographic map formation of heteroscedastic Gaussian mixtures that allows for a unified account of distortion error (vector quantization), log-likelihood and Kullback-Leibler divergence, and that generalizes Heskes' algorithm (2001) to the heteroscedastic case.

There is also the heuristic approach suggested by Yin and Allinson (2001), which is minimizing the Kullback-Leibler divergence, based on an idea introduced by Benaim and Tomasini (1991) for the homoscedastic case. Albeit that these authors only suggested an incremental, gradient-based learning procedure (thus, with a learning rate), we can cast their format into a fixed point learning scheme:

$$\begin{aligned} \mathbf{w}_i &= \frac{\sum_\mu \Lambda(i^*,i)P(i|\mathbf{v}^\mu)\mathbf{v}^\mu}{\sum_\mu \Lambda(i^*,i)P(i|\mathbf{v}^\mu)}, \\ \sigma_i^2 &= \frac{\sum_\mu \Lambda(i^*,i)P(i|\mathbf{v}^\mu)\|\mathbf{v}^\mu-\mathbf{w}_i\|^2/d}{\sum_\mu \Lambda(i^*,i)P(i|\mathbf{v}^\mu)}, \end{aligned} \tag{1.39}$$

with the winner neuron defined as $i^* = \text{argmax}_i P(i|\mathbf{v}^\mu)$, thus, the neuron with the largest posterior probability.

In a still different approach, an input to lattice transformation Φ is considered that admits a kernel function, a Gaussian (Van Hulle, 2002) $\langle \Phi(\mathbf{v}), \Phi(\mathbf{w}_i) \rangle = K(\mathbf{v}, \mathbf{w}_i, \sigma_i)$:

$$K(\mathbf{v}, \mathbf{w}_i, \sigma_i) = \exp\left(-\frac{\|\mathbf{v} - \mathbf{w}_i\|^2}{2\sigma_i^2} \right). \tag{1.40}$$

When performing topographic map formation, we require that the weight vectors are updated so as to minimize the expected value of the squared Euclidean distance $\|\mathbf{v} - \mathbf{w}_i\|^2$ and, hence, following our transformation Φ, we instead wish to minimize $\|\Phi(\mathbf{v}) - \Phi(\mathbf{w}_i)\|^2$, which we will achieve by performing gradient descent with respect to $\mathbf{w}_i$. The leads to the following fixed point rules to which we have added a neighborhood function:

$$\begin{aligned} \mathbf{w}_i &= \frac{\sum_\mu \Lambda(i, i^*) K(\mathbf{v}^\mu, \mathbf{w}_i, \sigma_i) \mathbf{v}^\mu}{\sum_\mu \Lambda(i, i^*) K(\mathbf{v}^\mu, \mathbf{w}_i, \sigma_i)}, \\ \sigma_i^2 &= \frac{1}{\rho d} \frac{\sum_\mu \Lambda(i, i^*) K(\mathbf{v}^\mu, \mathbf{w}_i, \sigma_i) \|\mathbf{v}^\mu - \mathbf{w}_i\|^2}{\sum_\mu \Lambda(i, i^*) K(\mathbf{v}^\mu, \mathbf{w}_i, \sigma_i)}, \end{aligned} \tag{1.41}$$

with ρ a scale factor (a constant) designed to relax the local Gaussian (and d large) assumption in practice, and with $i^* = \arg\max_{\forall i \in A} K(\mathbf{v}, \mathbf{w}_i, \sigma_i)$.

Rather than having a real-valued neural activation, one could also threshold the kernel into a binary variable: in the kernel-based maximum entropy rule (kMER) a neuron i is activated by input $\mathbf{v}$ when $\|\mathbf{w}_i - \mathbf{v}\| < \sigma_i$, where σ_i is the kernel radius of neuron i, and which defines a hyperspherical activation region, S_i (Van Hulle, 1998). The membership function, $\mathbb{1}_i(\mathbf{v})$, equals unity when neuron i is activated by $\mathbf{v}$, else it is zero. When there are no neurons active for a given input, the neuron that is positioned closest to that input is defined active. The incremental learning rules for the weights and radii of neuron i are as follows:

$$\begin{aligned} \Delta \mathbf{w}_i &= \eta \sum_j \Lambda(i, j) \, \Xi_i(\mathbf{v}) \, \text{sign}(\mathbf{v} - \mathbf{w}_i), \\ \Delta \sigma_i &= \eta \left(\frac{\rho_\text{r}}{N} \big(1 - \mathbb{1}_i(\mathbf{v})\big) - \mathbb{1}_i(\mathbf{v}) \right), \end{aligned} \tag{1.42}$$

with sign(.) the sign function taken componentwise, η the learning rate, $\Xi_i(\mathbf{v}) = \frac{\mathbb{1}_i}{\sum_j \mathbb{1}_j}$ a fuzzy membership function, and $\rho_r = \frac{\rho N}{N - \rho}$. It can be shown that the kernel ranges converge to the case where the average probabilities become equal, $\langle \mathbb{1}_i \rangle = \frac{\rho}{N}, \forall i$. By virtue of the latter, kMER is said to generate an equiprobabilistic topographic map (which avoids dead units). The algorithm has been considered for a wide range of applications, such as shape clustering (Van Hulle and Gautama, 2004), music signal clustering (Van Hulle, 2000), and the linking of patent- and scientific publications databases (Deleus and van Hulle, 2001). More recently, also

a fixed point version, called batch map kMER, was introduced (Gautama and Van Hulle, 2006), and applied to handwritten numerals clustering.

1.7.7 Kernels other than Gaussians

In principle, kernels other than Gaussians could be used in topographic map formation. For example, Heskes pointed out that his regularized mixture modeling approach could, in principle, accommodate any kernel of the exponential family, such as the Gamma, multinomial and the Poisson distribution (Heskes, 2001).

In another case, the kernel is considered for which the differential entropy of the kernel output will be maximal given a Gaussian input, *i.e.*, the incomplete gamma distribution kernel (Van Hulle, 2002b).

Another type of kernels are the Edgeworth-expanded Gaussian kernels, which consist of a Gaussian kernel multiplied by a series of Hermite polynomials of increasing order, and of which the coefficients are specified by (the second- but also higher-order) cumulants (Van Hulle, 2005b).

In still another case, a mixture of Bernouilli distributions is taken (Verbeek *et al.*, 2005) for the specific purpose to better encode binary data (*e.g.*, word occurrence in a document). This also leads to an EM algorithm for updating the posteriors as well as the expected joint log-likelihood with respect to the parameters of the Bernouilli distributions. However, as the posteriors become quite peaked for higher dimensions, for visualization purposes, a power function of them was chosen. Several applications have been demonstrated, including word grouping in document data sets (newsgroup data set) and credit data analysis (from the UCI Machine Learning repository `http://archive.ics.uci.edu/ml/`).

1.7.8 Future developments

An expected development is to go beyond the limitation of the current kernel-based topographic maps that the inputs need to be vectors (we already saw the extension towards categorical data). But in the area of structural pattern recognition, more powerful data structures can be processed, such as strings, trees and graphs. The SOM algorithm has already been extended towards strings (Kohonen and Somervuo, 1998) and graphs, which include strings and trees (Günter and Bunke, 2002; Seo and Obermayer, 2004; Steil and Sperduti, 2007) (see also the SOMSD and the MSOM algorithms above). However, also new types of *kernels* for strings, trees and graphs have been suggested in the Support Vector Machine literature (thus, outside the topographic map literature) (for reviews, see Shawe-Taylor and Cristianini, 2004; Jin *al.*, 2005). The integration of these new types of kernels into kernel-based topographic maps is yet to be done, but could turn out to be a promising evolution for

biochemical applications, such as visualizing and clustering sets of structure-based molecule descriptions.

1.8 Conclusion

In this chapter we have introduced the Self-Organizing Map (SOM) algorithm, discussed its properties, limitations and application types, and reviewed a number of extensions, and other types of topographic map formation algorithms, such as the growing-, the recurrent-, and the kernel topographic maps. We have also indicated how recent developments in topographic maps enable us to consider categorical data, time series and tree structured data, widening further the application field towards micro-array data analysis, document analysis and retrieval, exploratory analysis of web navigation sequences, and the visualization of protein structures and long DNA sequences.

Acknowledgments

The author is supported by the Excellence Financing program (EF 2005) and the CREA Financing program (CREA/07/027) of the K.U.Leuven, the Belgian Fund for Scientific Research – Flanders (G.0234.04 and G.0588.09), the Flemish Regional Ministry of Education (Belgium) (GOA 2000/11), the Belgian Science Policy (IUAP P5/04), and the European Commission (NEST-2003-012963, STREP-2002-016276, IST-2004-027017, and ICT-2007-217077).

References

1. Abrantes, A.J., and Marques, J.S. (1995). Unified approach to snakes, elastic nets, and Kohonen maps. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'95*, **5**, pp. 3427-3430.
2. Ahalt, S.C., Krishnamurthy, A.K., Chen, P., and Melton, D.E. (1990). Competitive learning algorithms for vector quantization. *Neural Networks*, **3**, 277-290.
3. Alahakoon, D., Halgamuge, S.K., and Srinivasan, B. (2000). Dynamic Self Organising Maps with Controlled Growth for Knowledge Discovery. *IEEE Transactions on Neural Networks* (Special Issue on Knowledge Discovery and Data Mining), **11**(3), 601-614.
4. Axelson, D., Bakken, I.J., Gribbestad, I.S. Ehrnholm, B. Nilsen, G. and Aasly, J. (2002). Applications of neural network analyses to in vivo ^{1}H magnetic resonance spectroscopy of Parkinson disease patients. *Journal of Magnetic Resonance Imaging*, **16**(1), 13-20.
5. Ball, K.D., Erman, B., and Dill, K.A. (2002). The elastic net algorithm and protein structure prediction. *Journal of Computational Chemistry*, **23**(1), 77-83.
6. Barreto, G., and Araújo, A. (2001). Time in self-organizing maps: An overview of models. *Int. J. of Computer Research*, **10**(2), 139-179.

7. Bauer, H.-U., and Villmann, T. (1997). Growing a Hypercubical Output Space in a Self-Organizing Feature Map. *IEEE Transactions on Neural Networks*, **8**(2), 218-226.
8. Bauer, H.-U., Der, R., and Herrmann, M. (1996). Controlling the magnification factor of self-organizing feature maps. *Neural Computat.*, **8**, 757-771.
9. Benaim, M., and Tomasini, L. (1991). Competitive and self-organizing algorithms based on the minimization of an information criterion. *Proc. ICANN'91*, pp. 391-396.
10. Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*, New York: Springer.
11. Bishop, C.M., Svensén, M., and Williams, C.K.I. (1996). GTM: A principled alternative to the self-organizing map. In: *Proceedings 1996 International Conference in Artificial Neural Networks (ICANN'96)*, pp. 165-170.
12. Bishop, C.M., Hinton, G.E., and Strachan, I.G.D. (1997). In *Proceedings IEE Fifth International Conference on Artificial Neural Networks*, Cambridge (U.K.), pp. 111-116.
13. Bishop, C.M. , Svensén, M. and Williams, C.K.I. (1998). GTM: The generative topographic mapping, *Neural Computat.*, 10: 215-234.
14. Blackmore, J., and Miikkulainen, R. (1993). Incremental Grid Growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proc. IEEE Int'l Conference on Neural Networks*, San Francisco, CA, 1993, **1**, 450-455.
15. Bruske, J., and Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, **7**(4), 845-865.
16. Carreira-Perpiñán, M.Á., and Renals, S. (1998). Dimensionality reduction of electropalatographic data using latent variable models. *Speech Communications*, **26**(4), 259-282.
17. Centre, N.N.R. (2003). Bibliography on the Self-Organizing Map (SOM) and Learning Vector Quantization (LVQ), Helsinki Univ. of Tech.
`http://liinwww.ira.uka.de/bibliography/Neural/SOM.LVQ.html`
18. Chappell, G. and Taylor, J. (1993). The temporal Kohonen map. *Neural Networks*, **6**, 441-445.
19. Chinrungrueng, C., and Séquin, C.H. (1995). Optimal adaptive *k*-means algorithm with dynamic adjustment of learning rate. *IEEE Trans. Neural Networks*, **6**, 157-169.
20. Cottrell, M., and Fort, J.C. (1987). Etude d'un processus d'auto-organization. *Ann. Inst. Henri Poincaré*, **23**, 1-20.
21. D'Alimonte, D., Lowe, D., Nabney, I.T., and Sivaraksa, M. (2005). Visualising uncertain data. In: *Proceedings European Conference on Emergent Aspects in Clinical Data Analysis (EACDA2005)* (September 28 - 30 2005, Pisa, Italy) `http://ciml.di.unipi.it/EACDA2005/papers.html`.
22. Deleus, F.F., and Van Hulle, M.M. (2001). Science and technology interactions discovered with a new topographic map-based visualization tool. *Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, San Fransisco, USA, August 26-29, 2001, pp. 42-50.
23. Dempster, A.P., Laird, N.M., and Rubin, D.B. (1977). Maximum likelihood for incomplete data via the EM algorithm. *J. Roy. Statist. Soc., B*, **39**, 1-38.
24. Der, R., and Herrmann, M. (1993). Phase transitions in self-organizing feature maps. *Proc. ICANN'93* (Amsterdam, The Netherlands), pp. 597-600, New York: Springer.
25. DeSieno, D. (1988). Adding a conscience to competitive learning. *Proc. IEEE Int. Conf. on Neural Networks* (San Diego), Vol. I, pp. 117-124.
26. Durbin, R., and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, **326**, 689-691.
27. Durbin, R. Szeliski, R., and Yuille, A.L. (1989). An analysis of the elastic net approach to the traveling salesman problem. *Neural Computat.*, **1**, 348-358.
28. Erwin, E., Obermayer, K., and Schulten, K. (1992). Self-organizing maps: Ordering, convergence properties and energy functions. *Biol. Cybern.*, **67**, 47-55.
29. Euliano, N.R., and Principe, J.C. (1999). A spatiotemporal memory based on SOMs with activity diffusion. In *Kohonen Maps*, E. Oja and S. Kaski (eds.), Elsevier, pp. 253-266.
30. Fritzke, B. (1994). Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, **7**(9), 1441-1460.

31. Fritzke, B. (1995a). A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, G. Tesauro, D.S. Touretzky and T.K. Leen (Eds.), MIT Press, pp. 625-632.
32. Fritzke, B. (1995b). Growing Grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, **2**(5), 9-13.
33. Fritzke, B. (1996). Growing self-organizing networks - why? In *European Symposium on Artificial Neural Networks (ESANN96)*, Bruges, 1996, D facto publications, Brussels, Belgium, pp. 61-72.
34. Gautama, T., and Van Hulle, M.M. (2006). Batch Map Extensions of the Kernel-based Maximum Entropy Learning Rule. *IEEE Transactions on Neural Networks*, **16**(2), 529-532.
35. Gersho, A., and Gray, R.M. (1991). *Vector quantization and signal compression*. Boston, Dordrecht, London: Kluwer.
36. Geszti, T. (1990). *Physical models of neural networks*. Singapore: World Scientific Press.
37. Gilson, S.J., Middleton, I., and Damper, R.I. (1997). A localised elastic net technique for lung boundary extraction from magnetic resonance images. In: *Proceedings Fifth International Conference on Artificial Neural Networks* (Cambridge, UK, 7-9 July 1997), pp. 199-204.
38. Gorbunov, S. and Kisel, I. (2006). Elastic net for stand-alone RICH ring finding. *Nuclear Instruments and Methods in Physics Research A*, **559**, 139-142.
39. Graepel, T., Burger, M., and Obermayer, K. (1997). Phase transitions in stochastic self-organizing maps. *Physical Rev. E*, **56**(4), 3876-3890.
40. Graepel, T., Burger, M., and Obermayer, K. (1998). Self-organizing maps: Generalizations and new optimization techniques. *Neurocomputing*, **21**, 173-190.
41. Grossberg, S. (1976). Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors. *Biol. Cybern.*, **23**, 121-134.
42. Günter, S., and Bunke, H. (2002). Self-organizing map for clustering in the graph domain, *Pattern Recognition Letters*, **23**, 4-5-417.
43. Hagenbuchner, M., Sperduti, A., and Tsoi, A.C. (2003). A Self-Organizing Map for Adaptive Processing of Structured Data. *IEEE Transactions on Neural Networks*, **14**(3), 491-505.
44. Hammer, B., Micheli, A., Strickert, M., and Sperduti, A. (2004). A general framework for unsupervised processing of structured data, *Neurocomputing*, **57**, 3-35.
45. Hammer, B., Micheli, A., Neubauer, N., Sperduti, A., and Strickert, M. (2005) Self Organizing Maps for Time Series. In *Proceedings of WSOM 2005*, Paris (France), September 05-08, pp. 115-122.
46. Heskes, T. (2001). Self-organizing maps, vector quantization, and mixture modeling. *IEEE Trans. Neural Networks*, **12**(6), 1299-1305.
47. Heskes, T.M., and Kappen, B. (1993). Error potentials for self-organization. *Proc. IEEE Int. Conf. on Neural Networks* (San Francisco, 1993), pp. 1219-1223.
48. Jin, B., Zhang, Y.-Q., and Wang, B. (2005). Evolutionary granular kernel trees and applications in drug activity comparisons, *Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'05)*, 1-6.
49. Kabán, A. (2005). A Scalable Generative Topographic Mapping for Sparse Data Sequences In: *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)*, Vol. 1, pp. 51-56.
50. Kass, M., Witkin, A., and Terzopoulos, D. (1987). Active contour models. *International Journal of Computer Vision*, **1**(4), 321-331.
51. Kangas, J. (1990). Time-delayed self-organizing maps. In *Proc. IEEE/INNS Int. Joint Conf. on Neural Networks 1990*, **2**, 331-336.
52. Kaski, S., Honkela, T., Lagus, K., and Kohonen, T. (1998). WEBSOM - self-organizing maps of document collections, /it Neurocomputing, **21**, 101-117.
53. Kim, Y.K., and Ra, J.B. (1995). Adaptive learning method in self-organizing map for edge preserving vector quantization. *IEEE Trans. Neural Networks*, **6**, 278-280.
54. Kitamoto, A. (2002). Evolution Map: Modeling State Transition of Typhoon Image Sequences by Spatio-Temporal Clustering. *Lecture Notes in Computer Science*, **2534/2002**, 283-290.

55. Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, **43**, 59-69.
56. Kohonen, T. (1984). *Self-organization and associative memory.* Heidelberg: Springer.
57. Kohonen, T. (1991). Self-organizing maps: Optimization approaches. In *Artificial Neural Networks*, T. Kohonen, K. Mäkisara, O. Simula and J. Kangas (Eds.), pp. 981-990, Amsterdam: North-Holland.
58. Kohonen, T. (1995). *Self-organizing maps.* Heidelberg: Springer (second edition: 1997)
59. Kohonen, T., and Somervuo, P., (1998). Self-organizing maps on symbol strings. *Neurocomputing*, **21**, 19-30.
60. Kohonen, T., Kaski, S., Salojärvi, J., Honkela, J., Paatero, V., and Saarela, A. (1999). Self organization of a massive document collection. *IEEE Transactions on Neural Networks*, **11**(3), 574-585.
61. Koskela, T., Varsta, M., Heikkonen, J., and Kaski, K. (1998). Recurrent SOM with local linear models in time series prediction. In *Proc. 6th European Symposium on Artificial Neural Networks (ESANN 1998)*, M. Verleysen (ed.), D-facto, pp. 167-172
62. Kostiainen, T., and Lampinen, J. (2002). Generative probability density model in the self-organizing map. In *Self-organizing neural networks: Recent advances and applications*, U. Seiffert & L. Jain (Eds.), pp. 75-94. Physica Verlag.
63. Laaksonen, J., Koskela,, M., and Oja, E. (2002). PicSOM–Self-Organizing Image Retrieval With MPEG-7 Content Descriptors. *IEEE Transactions on Neural Networks*, **13**(4), 841-853.
64. Lin, J.K., Grier, D.G., and Cowan, J.D. (1997). Faithful representation of separable distributions. *Neural Computat.*, **9**, 1305-1320.
65. Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, **21**, 105-117.
66. Linsker, R. (1989). How to generate ordered maps by maximizing the mutual information between input and output signals. *Neural Computat.*, **1**, 402-411.
67. Luttrell, S.P. (1989). Self-organization: A derivation from first principles of a class of learning algorithms. *Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN89)* (Washington, DC), Part I, pp. 495-498, IEEE Press.
68. Luttrell, S.P. (1990). Derivation of a class of training algorithms. *IEEE Trans. Neural Networks*, **1**, 229-232.
69. Luttrell, S.P. (1991). Code vector density in topographic mappings: Scalar case. *IEEE Trans. Neural Networks*, **2**, 427-436.
70. Martinetz, T.M. (1993). Competitive hebbian learning rule forms perfectly topology preserving maps. In *Proc. Int'l Conf. on Artificial Neural Networks (ICANN93)*, Springer: Amsterdam, pp. 427-434.
71. Martinetz, T., and Schulten, K. (1991). A "neural-gas" network learns topologies. In *Proceedings of International Conference on Artificial Neural Networks (ICANN-91)*, Espoo, Finland, June 24-28, 1991, T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (Eds.), vol. I, North-Holland, Amsterdam, pp. 397-402.
72. Martinetz, T., Berkovich, S., and Schulten, K. (1993). "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, **4**(4), 558-569.
73. Merkl, D., He, S., Dittenbach, M., and Rauber, A. (2003). Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization. In *Proc. 4th Workshop on Self-Organizing Maps (WSOM03)*, Kitakyushu, Japan, September 11-14, 2003.
74. Mulier, F., and Cherkassky, V. (1995). Self-organization as an iterative kernel smoothing process. *Neural Computat.*, **7**, 1165-1177.
75. Rauber, A., Merkl, D. and Dittenbach, M. (2002). The Growing Hierarchical Self-Organizing Map: Exploratory Analysis of High-Dimensional Data. *IEEE Transactions on Neural Networks*, **13**(6), 1331-1341.
76. Redner, R.A., and Walker, H.F. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, **26**(2), 195-239.
77. Risi, S., Mörchen, F., Ultsch, A., and Lewark, P. (2007). Visual mining in music collections with Emergent SOM. *Proceedings Workshop*

on Self-Organizing Maps (WSOM '07) (Bielefeld, Germany, 3-6 September, 2007) ISBN: 978-3-00-022473-7, CD ROM, available online at `http://biecoll.ub.uni-bielefeld.de`.
78. Ritter, H. (1991). Asymptotic level density for a class of vector quantization processes. *IEEE Trans. Neural Networks*, **2**(1), 173-175.
79. Ritter, H. (1998). Self-organizing maps in non-Euclidean spaces, *Kohonen maps*, Oja E., Kaski, S. (Eds.). Elsevier, Amsterdam, 97-108.
80. Ritter, H., and Schulten, K. (1986). On the stationary state of Kohonen's self-organizing sensory mapping. *Biol. Cybern.*, **54**, 99-106.
81. Ritter, H., and Schulten, K. (1988). Kohonen's self-organizing maps: Exploring their computational capabilities, *Proc. IEEE Int. Conf. on Neural Networks (ICNN)* (San Diego, CA, 1988), **I**, 109-116.
82. Ritter, H., and Kohonen, T. (1989). Self-organizing semantic maps. *Biological Cybernetics*, **61**, 241-254.
83. Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural computation and self-organizing maps: An introduction.* Reading, MA: Addison-Wesley.
84. Rose, K., Gurewitz, E., and Fox, G.C. (1993). Constrained clustering as an optimization Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(8), 785-794.
85. Schulz, R., and Reggia, J.A. (2004). Temporally asymmetric learning supports sequence processing in multi-winner self-organizing maps. *Neural Computation*, **16**(3), 535-561.
86. Seo, S., and Obermayer, K. (2004). Self-organizing maps and clustering methods for matrix data. *Neural Networks*, **17**(8,9), 1211-1229.
87. Shawe-Taylor, J., and Cristianini, N. (2004). *Kernel methods in computational biology*, MIT Press.
88. Simon, G., Lendasse, A., Cottrell, M., Fort, J.-C., and Verleysen, M. (2003). Double SOM for Longterm Time Series Prediction. In *Proc. of the Workshop on Self-Organizing Maps (WSOM 2003)*, Hibikino (Japan), 11-14 September 2003, pp. 35-40.
89. Somervuo, P.J. (2004). Online algorithm for the self-organizing map of symbol strings. *Neural Networks*, **17**(8-9), 1231-1240.
90. Steil, J.J., and Sperduti, A. (2007). Indices to evaluate self-organizing maps for structures, *WSOM07*, (Bielefeld, Germany, 3-6 September, 2007), CD ROM, 2007, available online at `http://biecoll.ub.uni-bielefeld.de`.
91. Strickert, M., and Hammer, B. (2003a). Unsupervised recursive sequence processing, In *European Symposium on Artificial Neural Networks (ESANN 2003)*, M. Verleysen (ed.), D-side publications, pp. 27-32.
92. Strickert, M., and Hammer, B. (2003b). Neural Gas for Sequences. *Proceedings of the Workshop on Self-Organizing Maps (WSOM'03)*, Hibikino, Kitakyushu, Japan, September 2003. pp. 53-57.
93. Strickert, M., and Hammer, B. (2005). Merge SOM for temporal data. *Neurocomputing*, **64**, 39-72.
94. Tiňo, P., and Nabney, I. (2002). Hierarchical GTM: constructing localized non-linear projection manifolds in a principled way. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(5), 639-656.
95. Tiňo, P., Kabán, A., and Sun, Y. (2004). A Generative Probabilistic Approach to Visualizing Sets of Symbolic Sequences. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - (KDD-2004)*, R. Kohavi, J. Gehrke, W. DuMouchel, J. Ghosh (eds.), ACM Press, pp. 701-706.
96. Tolat, V. (1990). An analysis of Kohonen's self-organizing maps using a system of energy functions. *Biol. Cybern.*, **64**, 155-164.
97. Ultsch, A., and Siemon, H.P. (1990). Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis, In Proceedings Intern. Neural Networks, Kluwer Academic Press, Paris, pp. 305-308.
98. Ultsch, A., and Mörchen, F. (2005). *ESOM-Maps: Tools for clustering, visualization, and classification with Emergent SOM*. Technical Report Dept. of Mathematics and Computer Science, University of Marburg, Germany, No. 46.

99. Ueda, N., and Nakano, R. (1993). A new learning approach based on equidistortion principle for optimal vector quantizer design. *Proc. IEEE NNSP93* (Linthicum Heights, MD, 1993), pp. 362-371.
100. Van den Bout, D. E., and Miller III, T.K. (1989). TInMANN: The integer Markovian artificial neural network. *Proc. Int. Joint Conf. on Neural Networks (IJCNN89)*, Englewood Cliffs, NJ: Erlbaum, pp. II205-II211.
101. Van Hulle, M.M. (1997a). Topology-preserving map formation achieved with a purely local unsupervised competitive learning rule. *Neural Networks*, **10**(3), 431-446.
102. Van Hulle, M.M. (1997b). Nonparametric density estimation and regression achieved with topographic maps maximizing the information-theoretic entropy of their outputs. *Biol. Cybern.*, **77**, 49-61.
103. Van Hulle, M.M. (1998). Kernel-based equiprobabilistic topographic map formation. *Neural Computat.*, **10**(7), 1847-1871.
104. Van Hulle, M.M. (2000). *Faithful representations and topographic maps: From distortion- to information-based self-organization*, New York: Wiley.
105. Van Hulle, M.M. (2002). Kernel-based topographic map formation by local density modeling. *Neural Computation*, **14**(7), 1561-1573.
106. Van Hulle, M.M. (2005a). Maximum likelihood topographic map formation. *Neural Computation*, **17**(3), 503-513.
107. Van Hulle, M.M. (2005b). Edgeworth-expanded topographic map formation. *WSOM05* (Paris, France, 5-8 September, 2005), 719-724.
108. Van Hulle, M.M. (2009). Kernel-based topographic maps: Theory and applications. *Encyclopedia of Computer Science and Engineering*, Benjamin W. Wah (Ed.), in press.
109. Van Hulle, M.M., and Gautama, T. (2004). Optimal smoothing of kernel-based topographic maps with application to density-based clustering of shapes. *J. VLSI Signal Processing Systems for Signal, Image, and Video Technology*, **37**, 211-222.
110. Verbeek, J.J., Vlassis, N., and Kröse, B.J.A. (2005). Self-organizing mixture models. *Neurocomputing*, **63**, 99-123.
111. Vesanto, J. (1997). Using the SOM and local models in time-series prediction. In: *Proc. Workshop on Self-Organizing Maps (WSOM 1997)*, Helsinki (Finland), June 4-6, 1997, pp 209-214.
112. Voegtlin, T. (2002). Recursive self-organizing maps. *Neural Networks*, **15**(8-9), 979-992.
113. von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, **14**, 85-100.
114. Wiemer, J.C. (2003). The time-organized map algorithm: extending the self-organizing map to spatiotemporal signals. *Neural Computation*, 15(5), 1143-1171.
115. Yin, H., and Allinson, N.M. (2001). Self-organizing mixture networks for probability density estimation. *IEEE Trans. Neural Networks*, **12**, 405-411.