An Introduction to Self-Organizing Maps

Chapter · November 2012

DOI: 10.2991/978-94-91216-77-0_14

2 authors:

Umut Asan
Istanbul Technical University
90 PUBLICATIONS **932** CITATIONS

SEE PROFILE

Secil Ercan
Gustave Eiffel University
27 PUBLICATIONS **317** CITATIONS

SEE PROFILE

Chapter 14

An Introduction to Self-Organizing Maps

Umut Asan and Secil Ercan

Department of Industrial Engineering, Istanbul Technical University, 34357, Macka, Istanbul, Turkey

E-mail: asanu@itu.edu.tr, ercansec@itu.edu.tr

As a special class of artificial neural networks the Self Organizing Map is used extensively as a clustering and visualization technique in exploratory data analysis. This chapter provides a general introduction to the structure, algorithm and quality of Self Organizing Maps and presents industrial engineering related applications reported in the literature.

14.1 Introduction

One of the apparent trends today in all disciplines of engineering sciences has the goal of developing computationally intelligent systems that "enable or facilitate intelligent behavior in complex and changing environments" [1]. The concepts, algorithms and models suggested by this new paradigm of knowledge-based processing should have the potential to solve real world problems, be able to learn from experience, have the capability of self-organization, and exhibit an ability to adapt to new situations [2]. One of the methods belonging to the field of computational intelligence which has proven to be a very powerful tool for data analysis is the neurobiologically inspired self-organizing map (SOM). The most common model of SOMs, also known as the Kohonen network, is the topology-preserving map proposed by the Finnish researcher Teuvo Kohonen in 1982 [3, 4]. The method is a special class of artificial neural networks and is used extensively as a clustering and visualization tool in exploratory data analysis. Some of the major practical application areas in which SOMs have been effectively applied are industrial instrumentation, pattern

recognition, data mining and processing, process control, robotics, telecommunications, medical applications, optimization, and product management.

The principal objective of SOMs is to transform a complex high-dimensional input space into a simpler low-dimensional (typically two-dimensional) discrete output space by preserving the relationships (i.e. topology) in the data, but not the actual distances [5, 6]. The spatial locations (i.e., coordinates) of the nodes in the output space are indicative of inherent statistical features contained in the input space [7]. In that respect, they are similar to the well-known dimension reduction techniques such as principal component analysis and multidimensional scaling. However, in comparison to these multivariate techniques, the nonparametric SOM procedures have a number of advantages. First, they do not make assumptions regarding the distributions of variables and nor do they require independence among variables [8, 9]. Second, they are easier to implement and are able to solve nonlinear problems of very high complexity [10]. Last, they more effectively cope with noisy and missing data [11], very small dimensionality [8] and samples of unlimited size [12].

Among the architectures and algorithms suggested for artificial neural networks, the SOM is trained using an unsupervised learning scheme. That means, unlike supervised networks, learning in SOM does not rely on predefined target outcomes that would guide the process. Thus, a form of learning by observation, rather than learning by examples takes place to discover the underlying hidden patterns in the data set [13]. In order to learn without a teacher, SOMs apply a competitive learning rule where the output nodes compete among themselves for the opportunity to represent distinct patterns within the input space [7]. During the learning, the feedforward nature of SOMs allows an information flow in only one direction, without looping or cycling, from the input nodes to the output nodes. Note that every node in the input layer is linked (with weights) to every node in the output layer, which makes an SOM a completely connected network [6].

The formation of an SOM involves three characteristic processes, which can be summarized as follows [7]:

(i) *Competition*: The output nodes (neurons) in a self-organizing map compete with each other to best represent the particular input sample. The success of representation is measured using a discriminant function, where an input vector is compared with the weight vector of each output node. The particular node with its connection weights most similar to the input sample is declared winner of the competition. There are a number of different functions to determine the winner, i.e. the best-matching unit (BMU) on the map. The most commonly used one is the Euclidean distance.

(ii) *Cooperation*: Similar to "[human] neurons dealing with closely related pieces of information are close together so that they can interact via short synaptic connections" [7], SOM is a topographic organization in which nearby locations in the output space represent inputs with similar properties. This is possible in the presence of neighborhood information. The winning node determines the spatial location of a neighborhood of cooperating nodes. These nodes, sharing common features, activate each other to learn something from the same input.

(iii) *Adaptation*: The weight vectors of the winner and its neighboring units in the map are adjusted in favor of higher values of their discriminant functions. Through this learning process the relevant nodes become more similar to the input sample. Thus, nodes which have a strong response to a particular piece of input data will have an increased chance of responding to similar input data in the future.

These processes are repeated for all training samples as they are passed sequentially to the SOM. In this way, different locations on the map are trained to produce a strong response to different kinds of data [14]. Later, we will examine in detail an example of how these processes are carried out.

The aim of this chapter is to provide a general introduction to self-organizing maps and give an overview of the technique. It is not the intention of this chapter to give all theoretical details of the SOM algorithm, which can be found in Kohonen [4]. The material presented in the rest of this chapter is organized as follows. After providing a brief overview of the general structure of an SOM, the chapter continues with a detailed description and an illustrative example of the basic version of the SOM algorithm. In Section 3, the validation of SOMs is discussed and important quality measures proposed in the literature are presented. The following section focuses on the applications of the SOM algorithm relevant to industrial engineering. The chapter concludes with some final remarks in Section 14.5.

14.2 Self-Organizing Map

14.2.1 *Structure*

The SOM network typically consists of two layers of nodes, the input layer and the output layer, see Figure 14.1. Different than most of the other neural networks, in SOM the input layer of source nodes is directly connected to the output layer of computation nodes without any hidden layer [6]. The nodes in the input layer denote the attributes (features) or, more generally, the variables contained in the input data. Each piece of input

data is represented by an m-dimensional input vector, $x = (x_1 x_2 \ldots x_m)'$, whose elements indicate the attribute values of a particular dataset. If there are large differences among the attribute values in the data set, normalization of the data is required in order to avoid the dominance of a particular attribute or subset of attributes [6]. This will give equal chances to all the attributes [15] and improve the numerical accuracy [16]. Several methods have been proposed in the literature for normalization. One of the most common methods is the z-score transformation which converts each attribute value to standard scores – with a zero mean and a unit standard deviation - by subtracting the mean and dividing by standard deviation for each attribute ($(x_j - \mu_{x_j})/\sigma_{x_j}$). Another widely used method is to divide each attribute value to the maximum of all values [15]. Unit length where all attributes will be scaled to the same length, the min-max transformation which scales the values between [0, 1], and the logarithmic transformation appropriate for exponentially distributed variables are further methods used for normalization [17].

The output layer, also known as "Kohonen layer" or "SOM layer", represents a low-dimensional visualization of the data. Nodes in the output layer are arranged in form of a topological architecture. This predefined structure is usually two-dimensional and organized as a grid that consists of rows and columns [18]. The number of nodes in the output layer denotes the maximum number of clusters and influences the accuracy and generalization capability of the SOM. According to a practical rule, the number of map units corresponds to 10 percent of the number of attribute values in a data set [15], which, however, should not be considered as a strict rule. The topologic structure can also be one-dimensional [19]. In one-dimensional string, the only parameter that must be decided is the number of nodes in the output layer; there is no need for the decision on the number of rows or columns. However, it is generally agreed that a two-dimensional map provides a better representation of the clusters.

Arrangement of the nodes in the output layer sets up the neighborhood relations which are studied in detail in the following subsection. The spaces between the neighbors are arbitrary and not important for the system. Focus is on the adjacency for two clusters that shows the similarity [8].

Typically the network topology is arranged in either a rectangular or hexagonal grid in the traditional SOM [4, 20]. As shown in Figure 14.2 these topologies have different properties. In the rectangular topology each internal node has four neighbors, while in the hexagonal topology six. Among these topologies, the hexagonal structure is usually more preferred, since it displays greater variance in neighborhood size. In both topolo-

gies, number of rows and columns are allowed to differentiate. Yet, research on the variety of topologies shows that the square-type topologies with $n \times n$ network size perform better [21]. Even though both shapes are frequently used, they cause problems on the edges. The neurons at the edge of a network are less central than other neurons, and therefore spherical and geodesic topologies are chosen to decrease the edge effect [22].

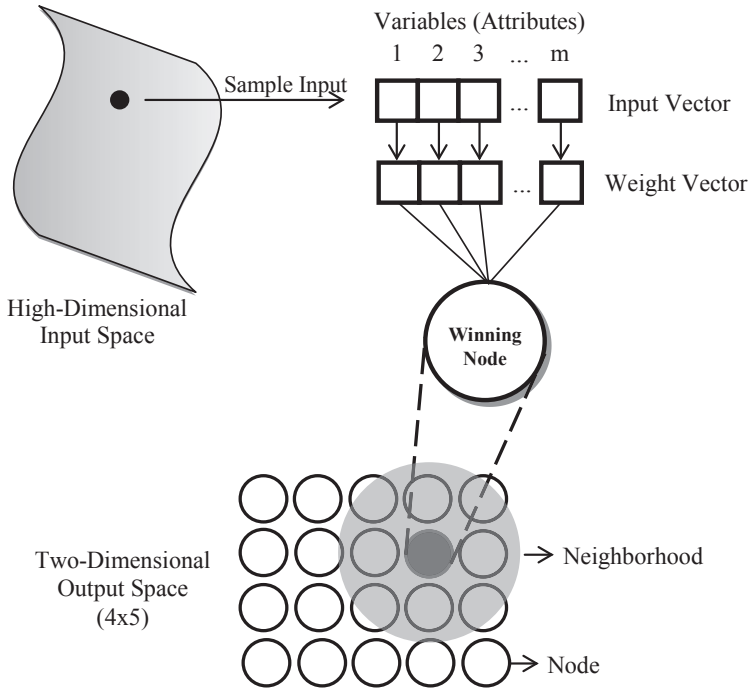
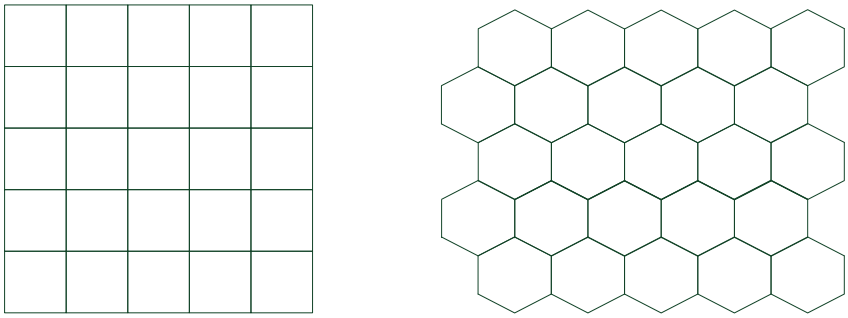


Fig. 14.1 Graphical illustration of a self-organizing map

Instead of using pre-defined structures with fixed number of neurons and grid dimensionality, it is possible to create gradually growing structures to better capture the fine-structure of the input distribution [23]. Several algorithms have been developed for growing SOM structures. For example, the Growing Grid algorithm starts with a 2×2 map and extends a column or a row for the next step. The Incremental Grid Growing (IGG) algorithm also starts with a small number of output nodes but generates any node irregularly when a new node is required. Another algorithm similar to IGG is the Growing SOM which additionally defines and uses a "spread factor" to control the growing [24]. All these growing structures give SOM a dynamic character.



(a) Rectangular topology (b) Hexagonal topology

Fig. 14.2 Typical topologies used in SOM

Table 14.1 Input space

	Personality Traits (Attributes)					
Customer	x_1	x_2	x_3	x_4	x_5	x_6
1	1	0.25	0.75	0	0,50	0.50
2	0.25	0.75	0.75	0.50	0.25	0.50
3	0.75	0.25	0.50	0.50	0.50	0.25
4	0.50	1	1	0.75	0.75	0
...	...	...	...	...	...	...
20	0	0.25	0.75	0.25	0.50	0.75

Example An illustrative SOM example about psychographic segmentation analysis will be presented throughout this section. The aim of using SOM is to divide a heterogeneous market into smaller homogeneous sub-sets consisting of customers who have relatively similar characteristics. The data collected through a survey consists of customer responses ($k = 20$) to items of personality traits ($j = 6$) including being social (x_1), innovative (x_2), confident (x_3), logical (x_4), cheerful (x_5), and practical (x_6). Table 14.1 presents the customer responses as input data. Input values range from 0 to 1 where 1 denotes "strongly agree" and 0 denotes "strongly disagree".

Since there are 20 input samples, at least $20 * (10\%) = 2$ clusters can be used to represent the output layer. However, in order to construct a square-type two dimensional map, a topologic structure with 2×2 network size has been chosen. As shown in Figure 14.3, the SOM in this case consists of 6 input nodes and 4 output nodes arranged in form of a 2×2 map. Because all input nodes are measured on the same scale, input vectors do not need to be normalized.

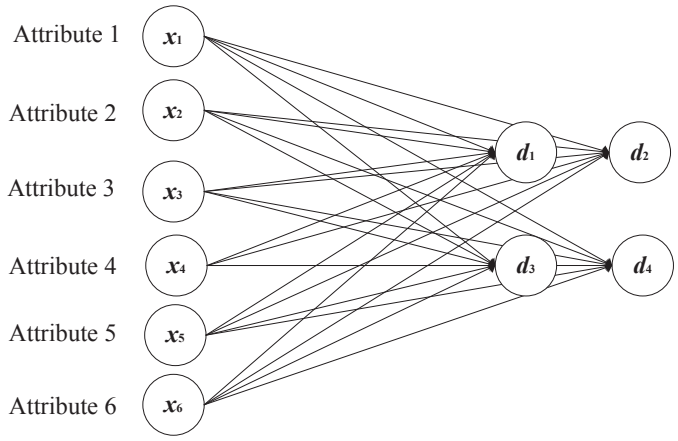


Fig. 14.3 The SOM architecture of the market segmentation example

14.2.2 *Analysis*

14.2.2.1 *Initial Weights*

The formation of the SOM starts first by initializing the weight vectors $w_i = (w_{i1}w_{i2}\ldots\ w_{im})'$ where $i = 1,2,\ldots,n$ and denotes the number of output nodes in the network. Weights are links that connect the input nodes to the output nodes and are updated through the learning process. Mazanec [8] states that the final results are somewhat subject to the chosen initial weight distribution which should be examined carefully. The initial weights, $w_i(t)$ for $t = 0$, of the nodes can be determined either arbitrarily or systematically (e.g. linearly). The most common initialization method assigns preferably small random values to the weight vectors of nodes in the network. That means, no prior order is imposed on the network in the initialization operation [7]. The only restriction imposed by this method is to avoid similar weights [16]. Note that, if the initial weights are randomly set around 0, the learning-rate and neighborhood parameters should be chosen reasonably large for a certain number of iterations (at least for 1000 steps) [15]. In practice, however, such random assignments can sometimes cause the algorithm to converge in a longer time. Another way of initializing the weights, which overcomes this problem, is to assign sample vectors drawn randomly from the training set to the weight vectors [7].

Alternatively, a more effective approach is to initialize the weight vectors along the two-dimensional linear subspace that is spanned by the eigenvectors corresponding to the largest eigenvalues computed from the input data [4]. Since the initial weights approximate

the probability density function of the input vector (x), the algorithm will converge faster. However, it is clear that this linear initialization method requires more calculations than the random initialization approaches.

14.2.2.2 *Similarity Matching*

As mentioned before, learning in SOM does not rely on predefined target outcomes that would guide the process. The output nodes compete among themselves to become activated. Only the node whose weight vector is most similar to the input vector will be activated and declared as the winner [20]. To find this best matching node, the distances between an input data (x) and all the weight vectors (w_i) of the SOM are computed using different measurement methods, such as Manhattan distance (city-block distance), Chebyshev distance, Euclidean distance, Mahalanobis distance [25] or vector dot product [20]. Of all these methods, as Kohonen [4] points out, Euclidean distance is more favorable in visual representations like those in SOMs, because a more isotropic display is obtained using it. For example, the algorithm in hexagonal lattice does not so much prefer horizontal and vertical directions. That's why the Euclidean distance is a generally accepted measure [4] and will be briefly explained here.

The Euclidean distance between a sample x, chosen randomly from the input dataset, and all the weight vectors at iteration t is calculated by using the following formula:

$$d_i(t) = \|x(t) - w_i(t)\| = \sqrt{\sum_{j=1}^{m} (x_{tj} - w_{tji})^2} \quad i = 1, 2, \ldots, n \tag{14.1}$$

where $\|\cdot\|$ denotes the Euclidean norm. Note that the weight vector w_i of each output node i has the same dimension as the input vector x. At the end of the similarity matching process, the best-matching (winning) node c at iteration t is determined by using the minimum-distance Euclidean criterion:

$$c(t) = \arg\min_i \{\|x(t) - w_i(t)\|\} \tag{14.2}$$

Example (continued) In our market segmentation example, for all four output nodes the initial weight vectors are assigned random values as shown in Table 14.2. The node whose weight vector is most similar to the customer response data, randomly chosen from the input space, will be activated and adjusted together with its neighboring units. For the sake of illustration, the Euclidean distance is applied to calculate the distances between the input data and all the weight vectors. Suppose that the first sample record $x(0) = (0.25\ 0.75\ 0.75\ 0.50\ 0.25\ 0.50)'$ randomly selected is the response of customer number

Table 14.2 Initial weights

j	w_1	w_2	w_3	w_4
1	0.15	0.06	0.11	0.38
2	0.97	0.84	1.00	0.78
3	0.65	0.50	0.99	0.79
4	0.27	0.35	0.64	0.68
5	0.09	0.39	0.19	0.44
6	0.46	0.85	0.47	0.22

two. The Euclidean distances for the four output nodes are calculated using Eq. (14.1):

$$d_1(0) = \sqrt{(0.25-0.15)^2+(0.75-0.97)^2+(0.75-0.65)^2+(0.5-0.27)^2+(0.25-0.09)^2+(0.5-0.46)^2} = 0.385$$

$$d_2(0) = \sqrt{(0.25-0.06)^2+(0.75-0.84)^2+(0.75-0.50)^2+(0.5-0.35)^2+(0.25-0.39)^2+(0.5-0.85)^2} = 0.521$$

$$d_3(0) = \sqrt{(0.25-0.11)^2+(0.75-1.00)^2+(0.75-0.99)^2+(0.5-0.64)^2+(0.25-0.19)^2+(0.5-0.47)^2} = 0.405$$

$$d_4(0) = \sqrt{(0.25-0.38)^2+(0.75-0.78)^2+(0.75-0.79)^2+(0.5-0.68)^2+(0.25-0.44)^2+(0.5-0.22)^2} = 0.408$$

For the first input record, the winning node is the first output node ($c(0) = 1$) that has the minimum Euclidean distance, $d_1(0) = 0.385$. In other words, the first output node is most similar to the given input node.

14.2.2.3 *Weight Updating*

The weight vectors of the winner and its neighboring units in the output space are adjusted to become more representative of the features that characterize the input space. This updating towards the input sample requires the consideration of the following two parameters: learning rate and neighborhood size. The learning rate, $\alpha(t)$, controls the rate of change of the weight vectors and, as in all neural networks, it takes values between 0 and 1. In SOMs, the learning rate gradually decreases as a function of the iteration step index t. That means, while the step index increases, the learning rate might decrease linearly, exponentially or geometrically [6], or it can be inversely proportional to t [4, 17]. Especially in cases where the initial weights are assigned random values, the learning rate should start with a reasonably high value that is close to unity and progressively reduce to small values. This procedure corresponds to larger corrections at the beginning of the training process (i.e. the ordering phase) than at the end where fine-tuning of the map takes place (i.e. the convergence phase) [4, 26]. The ordering phase may take as many as 1000 iterations of the SOM algorithm, and possibly more, while the convergence phase of the adaptive process must be at least 500 times the number of neurons in the network [4, 7].

Using a discrete-time formalism, given the weight vector $w_i(t)$ of the winning neuron i at iteration t, the updated weight vector $w_i(t+1)$ at iteration $t+1$ is defined by [4]:

$$w_i(t+1) = w_i(t) + \alpha(t)[x(t) - w_i(t)] \tag{14.3}$$

As mentioned above, the topological properties of the original data can only be preserved in the output space by considering neighborhood information. Thus, the reference vectors of the nodes in the neighborhood of the BMU also participate in the learning process. The rate of adaptation of the weights decreases away from the winning node, according to a spatio-temporal decay function [11], i.e. a neighborhood function $h_{ci}(t)$ where i denotes the index of the neighboring unit. In the literature, two fundamentally different types of neighborhood functions can be distinguished [16, 18]. The simpler of them, the discrete neighborhood function, defines a neighborhood set N_c of adjacent units around the winning node, whereby $h_{ci}(t) = \beta(t)$ if $i \in N_c$ and $h_{ci}(t) = 0$ if $i \notin N_c$. The value of $\beta(t)$ represents the degree of participation in the weight-updating procedure [18]. Examples of different discrete neighborhoods for rectangular and hexagonal topologies are shown in Figure 14.4. The other type of function widely applied for updating the neighborhood is the continuous neighborhood function which leads to a smoother neighborhood kernel. The most preferred function of this type is the Gaussian function which decreases in both the spatial domain and time domain:

$$h_{ci}(t) = \exp\left(-\frac{d_{ci}^2}{2\sigma^2(t)}\right) \tag{14.4}$$

where d_{ci}^2 denotes the lateral distance between the winning neuron c and the excited neuron i, and $\sigma(t)$ represents the effective width or radius of neighborhood at iteration t. The Gaussian function is symmetric about the maximum point defined by $d_{ci} = 0$, and decrease monotonically to zero with increasing lateral distance (d_{ci}) [7]. Here, $\sigma(t)$ is some monotonically decreasing function of time leading to a decrease in the value of the neighborhood function. It has been experimentally observed that choosing a too small neighborhood size at the beginning of the analysis may prevent inducing a global ordering. To overcome this problem Kohonen [16] suggests starting with a fairly large $\sigma(0)$ (e.g. more than half the diameter of the network). Other alternative neighborhood functions used in the literature are the bubble Eq. (14.5), cutgauss Eq. (14.6) and Epanechikov Eq. (14.7) [17]:

$$h_{ci}(t) = \mathbf{1}(\sigma(t) - d_{ci}) \tag{14.5}$$

$$h_{ci}(t) = \exp\left(-\frac{d_{ci}^2}{2\sigma^2(t)}\right)\mathbf{1}(\sigma(t) - d_{ci}) \tag{14.6}$$

$$h_{ci}(t) = \max\left\{0,\ 1 - (\sigma(t) - d_{ci})^2\right\} \tag{14.7}$$

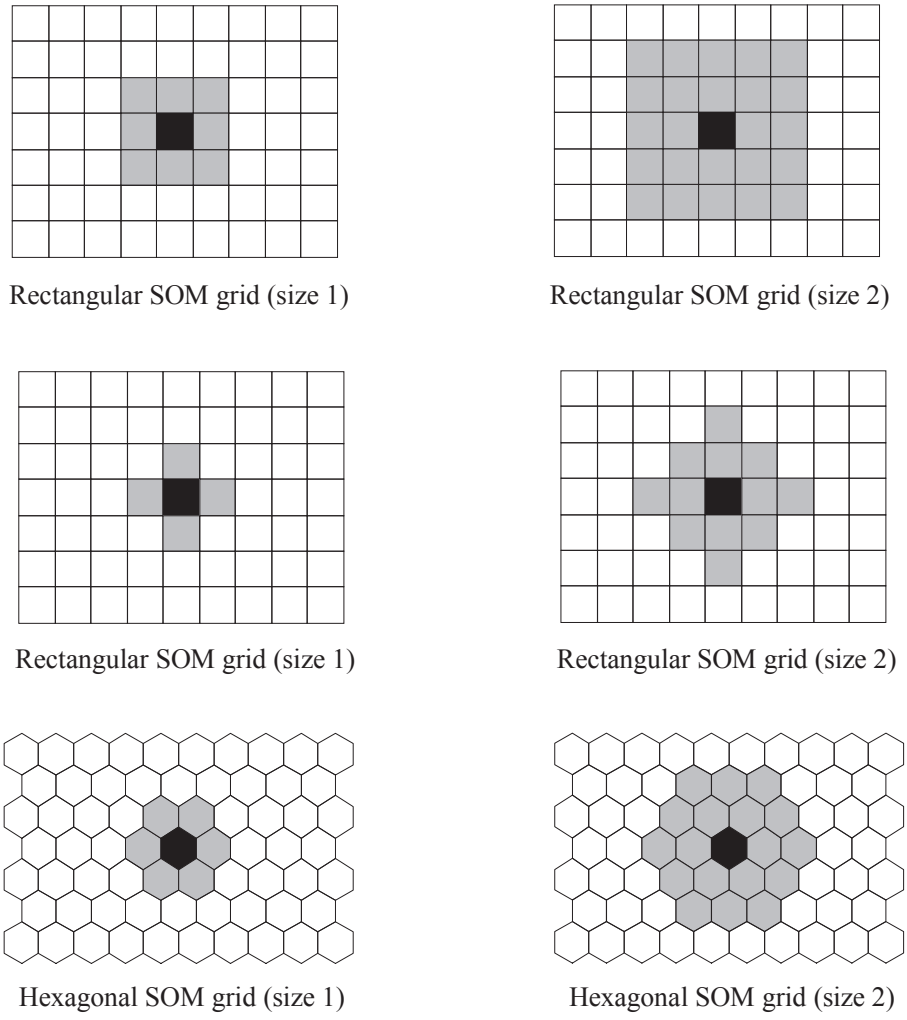


Fig. 14.4 Discrete neighborhoods (with size 1 and 2) for rectangular and hexagonal topologies

where $\mathbf{1}(x)$ is the step function: $\mathbf{1}(x) = 0$ if $x < 0$ and $\mathbf{1}(x) = 1$ if $x \geqslant 0$.

Finally, after defining the neighborhood set or kernel around the winning node c [16], a neighborhood function is chosen to update the weight vectors of the respective nodes. The following formula presents the update rule for the weight vector of unit i [4]:

$$w_i(t+1) = w_i(t) + \alpha(t)h_{ci}(t)[x(t) - w_i(t)] \tag{14.8}$$

All three processes – competition, cooperation and adaptation – described here are repeated for the remaining training data until the weights converge and no noticeable changes in the

low-dimensional output layer are observed. The entire set of the resulting weight vectors reflects the distribution of the input data.

Example (continued) Recall that for the first sample record selected, the winning node was the first output node ($c = 1$) that had the minimum Euclidean distance. Now the weight vectors of this best matching node and its neighborhood will be adjusted so that they become more similar to the current input sample $x(0)$. Setting the initial learning rate close to one ($\alpha(0) = 0.9$) and applying the adaptation formula given in Eq. (14.3), the updated weights for the winning node ($c = 1$) at $t = 0$ are calculated as shown below:

$$w_1(1) = w_1(0) + \alpha(0)[x(0) - w_1(0)]$$

$$\begin{bmatrix} w_{11}(1) \\ w_{12}(1) \\ w_{13}(1) \\ w_{14}(1) \\ w_{15}(1) \\ w_{16}(1) \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.97 \\ 0.65 \\ 0.27 \\ 0.09 \\ 0.46 \end{bmatrix} + 0.9 \left(\begin{bmatrix} 0.25 \\ 0.75 \\ 0.75 \\ 0.50 \\ 0.25 \\ 0.50 \end{bmatrix} - \begin{bmatrix} 0.15 \\ 0.97 \\ 0.65 \\ 0.27 \\ 0.09 \\ 0.46 \end{bmatrix} \right) = \begin{bmatrix} 0.24 \\ 0.77 \\ 0.74 \\ 0.48 \\ 0.23 \\ 0.50 \end{bmatrix}$$

Note that if the learning rate was chosen equal to zero, the weights would not change, i.e. $w_1(1) = w_1(0)$. Alternatively, if the learning rate was chosen equal to one, the new weights would equal to the input data, i.e.$w_1(1) = x(0)$. Next, applying the Gaussian neighborhood function with $\sigma(0) = 1$ (Eq. (14.4)) and the adaptation formula given in Eq. (14.8), the updated weights for the neighboring units at $t = 0$ are calculated as shown below:

$$h_{12}(0) = \exp\left(-\frac{d_{12}^2}{2\sigma^2(0)}\right) = \exp\left(-\frac{0.544^2}{2 \cdot 1^2}\right) = 0.862$$

$$h_{13}(0) = \exp\left(-\frac{d_{13}^2}{2\sigma^2(0)}\right) = \exp\left(-\frac{0.515^2}{2 \cdot 1^2}\right) = 0.876$$

$$h_{14}(0) = \exp\left(-\frac{d_{14}^2}{2\sigma^2(0)}\right) = \exp\left(-\frac{0.676^2}{2 \cdot 1^2}\right) = 0.796$$

where

$$d_{12} = \sqrt{(0.15-0.06)^2+(0.97-0.84)^2+(0.65-0.50)^2+(0.27-0.35)^2+(0.09-0.39)^2+(0.46-0.85)^2} = 0.544$$

$$d_{13} = \sqrt{(0.15-0.11)^2+(0.97-1.00)^2+(0.65-0.99)^2+(0.27-0.64)^2+(0.09-0.19)^2+(0.46-0.47)^2} = 0.515$$

$$d_{14} = \sqrt{(0.15-0.38)^2+(0.97-0.78)^2+(0.65-0.79)^2+(0.27-0.68)^2+(0.09-0.44)^2+(0.46-0.22)^2} = 0.676$$

hence, we obtain for output node 2:

$$w_2(1) = w_2(0) + \alpha(0)h_{12}(0)[x(0) - w_2(0)]$$

$$\begin{bmatrix} w_{21}(1) \\ w_{22}(1) \\ w_{23}(1) \\ w_{24}(1) \\ w_{25}(1) \\ w_{26}(1) \end{bmatrix} = \begin{bmatrix} 0.06 \\ 0.84 \\ 0.50 \\ 0.35 \\ 0.39 \\ 0.85 \end{bmatrix} + (0.9)(0.862) \left(\begin{bmatrix} 0.25 \\ 0.75 \\ 0.75 \\ 0.50 \\ 0.25 \\ 0.50 \end{bmatrix} - \begin{bmatrix} 0.06 \\ 0.84 \\ 0.50 \\ 0.35 \\ 0.39 \\ 0.85 \end{bmatrix} \right) = \begin{bmatrix} 0.21 \\ 0.77 \\ 0.69 \\ 0.47 \\ 0.28 \\ 0.58 \end{bmatrix}$$

The weights of the output nodes 3 and 4 are calculated similarly. The adjusted weight vectors of the winning node and its neighborhood are given below:

$$w_1(1) = (\,0.24\ 0.77\ 0.74\ 0.48\ 0.23\ 0.50\,)'$$

$$w_2(1) = (\,0.21\ 0.77\ 0.69\ 0.47\ 0.28\ 0.58\,)'$$

$$w_3(1) = (\,0.22\ 0.80\ 0.80\ 0.53\ 0.24\ 0.49\,)'$$

$$w_4(1) = (\,0.29\ 0.76\ 0.76\ 0.55\ 0.30\ 0.42\,)'$$

14.2.2.4 *Summary of the SOM Algorithm*

The basic steps of Kohonen's SOM algorithm can be summarized by the following iterative procedure:

(i) *Initialization*. Choose the dimension and size of the output space. Assign random values or alternatively sample vectors drawn randomly from the training set to the initial weight vectors $w_i(0)$. Specify the neighborhood function as well as the functional form of the learning rate $\alpha(0)$ and radius of the neighborhood $\sigma(0)$. Assign starting values for $\alpha(0)$ and $\sigma(0)$. Normalize training data. Define a critical threshold value (T) for the maximum number of iterations. Set iteration index $t = 1$.

(ii) *Sampling*. Randomly select an input vector $x(t)$ from the training data set.

(iii) *Similarity Matching*. Compute the Euclidean distances between the input vector and each output node's weight vector and find the best matching node $c(t)$ at iteration t by applying the minimum distance criterion:

$$c(t) = \arg\min_i \{\, \|x(t) - w_i(t)\| \,\} \quad i = 1, 2, \ldots, n$$

(iv) *Weight Updating*. Adjust the weights of the winning node and its neighborhood according to their distances to the winning node by using the update formula:

$$w_i(t+1) = w_i(t) + \alpha(t) h_{ci}(t)[x(t) - w_i(t)]$$

For the winning node the neighborhood function $h_{ci}(t)$ will be equal to 1.

(v) *Parameter Adjustment*. Set $t = t + 1$. Adjust the neighborhood size and the learning rate.

(vi) *Continuation*. Keep returning to Step 2 until the change of the weights is less than a prespecified threshold value or the maximum number T of iterations is reached. Otherwise stop.

14.3 Quality of SOM

SOM has the ability to produce simpler low-dimensional representations of complex high-dimensional data at the expense of information loss during the projection. This representation of topological relations and abstract structures is an approximation of the local, rather than the global, probability density of a given sample data set [9]. This explains why the outputs of an SOM can vary highly depending on the initial setting of parameters such as the number of output nodes, the learning rate, and the update speed of the neighborhood [4, 9]. It is, therefore, necessary to examine the quality of the results in order to determine the parameter values which give the most appropriate representation of the data set. However, it should be noted that there are no generally accepted rules for the determination of the parameters a priori. To address these issues different measurement methods have been developed in the literature to evaluate and compare the quality of SOM outputs. In this respect, the most studied properties of SOM projections are learning quality and projection quality, where the first one is sometimes referred to as vector quantization and the latter one as topology preservation. Technically, there is a tradeoff between these two, the topology preservation quality usually decreases with increasing data representation accuracy [27].

The methods suggested to measure the mentioned properties will be discussed in the remainder of this subsection. Note that not all the descriptions of the methods summarized contain formulas; for more detail on the concepts, please refer to the referenced publications.

Quantization Error

The Quantization Error (QE) provides a means to assess the quality of learning and shows how good the map fits to data. It is computed by determining the average distance of the sample vectors to its best matching unit [4]. The formula of the average quantization

error is as follows:

$$QE = \frac{\sum_{t=1}^{T} \|x(t) - w_c(t)\|}{T} \tag{14.9}$$

where $w_c(t)$ is the reference vector associated to the BMU of $x(t)$.

By increasing the number of output nodes for any given dataset, the QE can be reduced as the data samples are distributed more sparsely on the map [27]. Here, the QE measure completely disregards the topological relations and because of the tradeoff between vector quantization and projection quality of the SOM, reducing the QE may lead to distortions of the map's topology [27]. Besides, Sun [28] suggests using a small neighborhood size in order to achieve a small quantization error and a good approximation of input distribution. Note that a low QE may also imply an over fitted model [29].

Topographic Error

Topology preservation, i.e. projection quality, is a property that is more complex to define and hard to measure. The most common quality measure for topology preservation is the Topographic Error (TE), which can be expressed as the percentage of data vectors for which the first and second BMUs are not adjacent units [30]. The topographic error is computed as follows [31]:

$$TE = \frac{1}{T} \sum_{t=1}^{T} u(x(t)) \tag{14.10}$$

where T is the number of input samples, and $u(x(t)) = 1$ if the best- and second best-matching units of $x(t)$ are non-adjacent, otherwise $u(x(t)) = 0$. So, the less the violation of the neighborhood relations on the grid is, the better the SOM preserves the topology. According to Pölzlbauer [27] this measure is unreliable for small maps, since the TE is almost too simplistic and suffers from the discrete nature of the output space.

Topographic Product

Introduced by Bauer and Pawelzik [32] the Topographic Product (TP) is a measure of the preservation of neighborhood relations in maps. According to TP, for each node on the map, the weight vectors of its k-nearest neighbor units are compared whether they coincide or not. If folds are observed on the map, this will be taken as an indicator that the map is trying to approximate a higher-dimensional input space, and thus producing topographic error [31]. Therefore, the result of the computation of the TP can be used to optimize the

map size for a given input space. However, it was shown by Villmann *et al.* [33] that the topographic product fails to distinguish between the results caused by a folded input space, and the folds that are actually erroneous. Therefore, the TP presents reliable results only for nearly linear datasets.

Distortion Measure

One mathematical difficulty of the SOM algorithm is to prove that the algorithm converges to a good solution. Typically, in neural network learning algorithms, the weight update rule corresponds to some kind of gradient descent on an energy (or cost) function However, it has been shown that there exists no energy function for the Kohonen's original learning rule [34]. "Hence, strictly speaking, we cannot judge the degree of optimality achieved by the algorithm during learning" [23]. Only for a finite set of training patterns and an algorithm with a fixed neighborhood kernel radius, it is possible to come up with an energy function the SOM optimizes [35]. One such energy function used for measuring the overall quality of a given SOM is the Distortion Measure (E_d) [36], which is computed as follows:

$$E_d = \sum_{t=1}^{T} \sum_{i=1}^{n} h_{ci}(t) \, \|x(t) - w_i(t)\| \tag{14.11}$$

One important advantage of the Distortion Measure is that the error can be decomposed into three component terms [37]:

- Local Data Variance: it gives the quantization quality of the SOM,
- Neighborhood Variance: it analyzes the trustworthiness [38] (i.e. closeness of prototype vectors close to each other on the map grid) of the map topology,
- Neighborhood Bias: it links the other two properties together and can be regarded as the stress between them.

The Distortion Measure can be used to compare several maps trained with the same dataset according to their fit to the data.

Recall that finding the right size for the low-dimensional map is a quite important task in SOM. A too small map will be too rough to reveal some significant differences in data whereas a too large map will detect more detailed patterns where differences are too small. All the measures described above are critical in choosing the optimum size of the output space. As a result, for the same dataset, the map size where the values of QE and TE become minimum should be selected [29].

14.4 Applications of SOM

This section focuses on the applications of the SOM algorithm reported in the literature. Major practical application areas include process analysis, geoscience and remote control, image analysis and pattern recognition, information technology, intelligent systems design, signal processing, ecosystem modeling, robotics, optimization, expert systems and data mining (an exhaustive listing of the detailed applications can be found in Ref.39). Here, rather than attempting for an extensive overview, only the applications relevant to industrial engineering are presented.

Location-allocation (LA) problem. This problem consist of determining the locations of a set of facilities and the flows of products from these facilities to customers, such that the sum of location-specific fixed costs and operational as well as transportation costs are minimized. In this respect, the LA problem is a type of clustering problem based on minimum distances which can be solved by using an SOM algorithm [40, 41]. Here, the locations of K customers (i.e. K demand centers) taken as input patterns are clustered into N supply centers represented as output nodes in SOM. While the weights represent the location of supply centers, the input values denote the demands of customers. The learning algorithm adjusts the locations of the closest supply center (winning node) and its neighbors until no change with further training is observed.

Traveling-salesman problem (TSP). Given a list of cities and their pairwise distances, this problem searches the shortest (cheapest) possible route that visits each city exactly once and returns to the starting city. The neighborhood preserving property of the SOM and the convex-hull property of the TSP makes SOM a suitable algorithm to find the optimal route [42]. When the location of each city is represented by a point in a two-dimensional input space, and a tour by a sequence of N output nodes (i.e. closed chain), then a solution to the TSP can be viewed as a mapping from the input-space onto the nodes of the chain [23]. After a finite number of training cycles the weights of the output nodes are expected to coincide with the locations of the cities.

Cell designing problem. In cellular manufacturing machines are grouped together according to the families of parts produced. The cell formation problem is primarily concerned with machine–part clustering. As one of the clustering methods the SOM can successfully solve this problem by grouping machines and parts based on operations sequence [29]. Product parts are defined as the input data while machines as the output nodes. The SOM algorithm not only suggests a clustering solution but also provides a visualization of the solution.

Product positioning problem. Product positioning aims at exploring the perceptual differences between the brands of a product class and the way they are linked to consumer preferences or choice [43]. To this end, various sophisticated multivariate techniques (such as principle components, multidimensional scaling and unfolding, correspondence analysis) have been used to extract the competitive dimensions, product positions and consumer preferences [44]. The SOM approximates an unknown multivariate probability distribution by fitting topologically ordered units (product and customer positions) in a predefined low-dimensional perceptual space and links these positions to stated or revealed brand choice data. It allows performing segmentation (clustering) and positioning (mapping) analysis simultaneously. In comparison to the multivariate techniques, the SOM procedures are easy to implement, do not impose rigorous distributional assumptions and nor do they require particular scaling properties of the raw data. They effectively cope with very small dimensionality and samples of unlimited size [12]. For more detail on market segmentation using SOM the reader should refer to Kuo *et al.* [45], Kiang *et al.* [46], Hung and Tsai [47].

Multiple objective just-in-time (JIT) sequencing problem. This problem considers a JIT production-scheduling system where two inversely related objectives are present – minimization of setups between differing products and optimization of schedule flexibility. To address this problem, Kohonen's SOM has been used to search for sequences which are desirable in terms of both the number of setups and flexibility [48]. For this particular effort, each input sample associated with a particular item is represented by a two-dimensional vector denoting its coordinates (on a circle). The total number of inputs will be the total demand of all items needing sequencing. The weight vectors in this problem represent the sequence position of each of the individual items. During the learning process the weight vectors tend to organize themselves toward the (circular) patterns having minimal setups [48]. At the end of the training, each input vector will have been associated with a unique weight vector.

Product concept development, demand forecasting and approximation of probability density functions are further examples of industrial engineering relevant SOM applications, which, however, will not be discussed here.

14.5 Conclusion

The number of industrial engineering related SOM applications reported in the literature has shown a considerable increase in the past decade. Its ability to explore and

visualize high-dimensional data with no rigorous assumptions makes SOM an appealing technique in this field. This chapter is intended to contribute to this evolving body of knowledge by providing a gentle introduction to self-organizing maps and discussing its properties, limitations and applications. It can be concluded that combining the SOM with appropriate statistical techniques, heuristic methods or decision making tools appears to be a promising approach for widening further the application field towards industrial engineering problems. However understanding the mathematics of self-organizing processes will remain a highly active field of research.

Bibliography

[1] P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd ed. John Wiley & Sons, England, (2007).
[2] N.R. Pal and S. Pal, Computational Intelligence for Pattern Recognition, *International Journal of Pattern Recognition and Artificial Intelligence*, **16** (7), 773–779 (2002).
[3] T. Kohonen, Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, **43**, 59–69 (1982).
[4] T. Kohonen, *Self-Organizing Maps*, Springer, Berlin, (2001).
[5] A.H. Holmbom, T. Eklund, and B. Back, Customer portfolio analysis using the SOM, *International Journal of Business Information Systems*, **8** (4), 396–412 (2011).
[6] D.T. Larose, *Discovering Knowledge in Data*, John Wiley & Sons, New Jersey, (2005).
[7] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice-Hall, Englewood Cliffs, New Jersey, (1999).
[8] J.A. Mazanec, Positioning Analysis with Self-Organizing Maps: An Exploratory Study on Luxury Hotels, *Cornell Hotel and Restaurant Administration Quarterly*, **36** (6), 80–95 (1995).
[9] S. Wang and H. Wang, Knowledge Discovery Through Self-Organizing Maps: Data Visualization and Query Processing, *Knowledge and Information Systems*, **4**, 31–45 (2002).
[10] J.E. Dayhoff, *Neural network architectures—An introduction*, New York: Van Nostrand Reinhold, (1990).
[11] A.J. Richardson, C. Risien, and F.A. Shillington, Using self-organizing maps to identify patterns in satellite imagery, *Progress in Oceanography*, **59**, 223–239 (2003).
[12] J.A. Mazanec, Neural market structure analysis: Novel topology-sensitive methodology, *European Journal of Marketing*, **35** (7/8), 894–914 (2001).
[13] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, San Francisco, (2001).
[14] J.B. Greenwald, *Optical Character Categorization: Clustering as it Applies to OCR*, PhD thesis, Rochester Institute of Technology (1997).
[15] M. Zontul, O. Kaynar, and H. Bircan, Som tipinde yapay sinir ağlarını kullanarak Türkiye'nin ithalat yaptığı ülkelerin kümelenmesi üzerine bir çalışma, *C.Ü. İktisadi ve İdari Bilimler Dergisi*, **5** (2004).
[16] T. Kohonen, The Self-Organizing Map, *Proceedings of the IEEE*, **78**, 1464–1480 (1990).
[17] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, SOM Toolbox for Matlab 5, *Report A57*, Helsinki University of Technology Finland, (2000).
[18] T. Reutterer, Competitive market structure and segmentation analysis with self-organizing feature maps, in *Proceedings of the 27th EMAC Conference*, Ed. P. Andersson. Marketing Re-

search Stockholm, (1998).
[19] J.A. Lee and M. Verleysen, Self-organizing maps with recursive neighborhood adaptation, *Neural Networks*, **15**, 993–1003 (2002).
[20] C. Budayan, *Strategic group analysis: strategic perspective, differentiation and performance in construction*, PhD thesis, Middle East Technical University (2008).
[21] Y.M. Kiang, Extending the Kohonen self-organizing map networks for clustering analysis, *Computational Statistics & Data Analysis*, **38**, 161–180 (2001).
[22] C.R. Schmidt, S.J. Rey, and A. Skupin, Effects of irregular topology on self- organizing maps, *International Regional Science Review*, **34**, 215–229 (2010).
[23] M.M. Van Hulle, Self-Organizing Maps, in *Handbook of Natural Computing: Theory, Experiments, and Applications*, Ed. G. Rozenberg, T. Baeck, J. Kok, Springer, (2011).
[24] D. Alahakoon and S.K. Halgamuge, Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery, *IEEE Transactions on Neural Networks*, **11**, 601–614 (2000).
[25] S. Aly, N. Tsuruta, and R.-I. Taniguchi, Face recognition under varying illumination using Mahalanobis self-organizing map, *Artificial Life and Robotics*, **13** (1), 298–301 (2008).
[26] R. Rojas, *Neural Networks*, Springer-Verlag, Berlin, (1996).
[27] G. Pölzlbauer, Survey and comparison of quality measures for self-organizing maps, in *Proceedings of the Fifth Workshop on Data Analysis* (*WDA-04*), Elfa Academic Press, Sliezsky dom, Vysoké Tatry, Slovakia, 67–82 (2004).
[28] Y. Sun, On quantization error of self-organizing map network, *Neurocomputing*, **34** (1-4), 169–193 (2000).
[29] M. Chattopadhyay, P.K. Dan, and S. Mazumdar, Application of visual clustering properties of self organizing map in machine–part cell formation, *Applied Soft Computing*, **12**, 600–610 (2012).
[30] J.I. Mwasiagi, H. XiuBao, W. XinHou, and C. Qing-Dong, The Use of K-Means and Kohonen Self Organizing Maps to Classify Cotton Bales, *Beltwide Cotton Conferences*, New Orleans, Louisiana, January 9-12 (2007).
[31] K. Kiviluoto, topology preservation in self-organizing maps, in *Proceedings of ICNN'96, IEE International Conference on Neural Networks*, IEEF, Service Center, Piscataway, pp. 294–299 (1996).
[32] H.U. Bauer and K.R. Pawelzik, Quantifying the neighborhood preservation of selforganizing feature maps, *IEEE Transactions on Neural Networks*, **3** (4), 570–579, (1992).
[33] T. Villmann, R. Der, and T. Martinez, A new quantitative measure of topology preservation in Kohonen's feature maps, *Proceedings of the IEEE International Conference on Neural Networks 94*, Orlando, Florida, USA, 645–648 (1994).
[34] E. Erwin, K. Obermayer, and K. Schulten. Selforganizing maps: Ordering, convergence properties and energy functions, *Biological Cybernetics*, **67** (1), 47–55 (1992).
[35] T. Heskes, Energy functions for self-organizing maps, in *Kohonen Maps*, Ed. E. Oja and S. Kaski, pp. 303–316, Elsevier, Amsterdam, (1999).
[36] J. Lampinen and E. Oja, Clustering Properties of Hierarchical Self-Organizing Maps, *Journal of Mathematical Imaging and Vision*, **2** (2-3), 261–272 (1992).
[37] J. Vesanto, M. Sulkava, and J. Hollmen, On the decomposition of the self-organizing map distortion measure, in *Proceedings of the Workshop on Self-organizing Maps*, WSOM'03, pp. 11–16 (2003).
[38] J. Venna and S. Kaski, Neighborhood preservation in nonlinear projection methods: An experimental study, in *ICANN 2001*, Ed. G. Dorffner, H. Bischof, and K. Hornik, LNCS, **vol. 2130**, pp. 485–491, Springer, Heidelberg, (2001).
[39] URL `http://www.cis.hut.fi/research/refs/`
[40] S. Lozano, F. Guerrero, L. Onieva and J. Larrafieta, Kohonen maps for solving a class of location-allocation problems, *European Journal of Operational Research*, **108**, 106–117

(1998).

[41] K.-H. Hsieh and F.-C. Tien, Self-organizing feature maps for solving location–allocation problems with rectilinear distances, *Computers & Operations Research*, **31**, 1017–1031 (2004).

[42] K.-S. Leung, H.-D. Jinb, and Z.-B. Xuc, An expanding self-organizing neural network for the traveling salesman problem, *Neurocomputing*, **62**, 267–292 (2004).

[43] P. Kotler, *Marketing Management* – International Millennium Edition, Prentice Hall, New Jersey, (2000).

[44] V. Trommsdorff, U. Asan, and J. Becker, "Produkt- und Markenpositionierung", in *Handbuch Markenführung*, Ed. Bruhn M., Volume 1, Second Edition, Gabler Verlag, Wiesbaden (2004).

[45] R.J. Kuo, Y.L. An, H.S. Wang, and W.J. Chung, Integration of self-organizing feature maps neural network and genetic K-means algorithm for market segmentation, *Expert Systems with Applications*, **30** (2), 313–324 (2006).

[46] M.Y. Kiang, M.Y. Hu, and D.M. Fisher, An extended self-organizing map network for market segmentation—a telecommunication example, *Decision Support Systems*, **42**, 36–47 (2006).

[47] C. Hung and C.-F. Tsai, Market segmentation based on hierarchical self-organizing map for markets of multimedia on demand, *Expert Systems with Applications*, **34** (1), 780–787 (2008).

[48] P.R. McMullen, A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem, *International Journal of Production Economics*, **72**, 59–71 (2001).