anywhere; it is the set of their internal *parameters* that defines this specificity and is made to change. Since such networks are usually mainly planar (two-dimensional) arrays, this result also means that there exist mappings that are able *to preserve the topological relations while performing a dimensionality reduction of the representation space.*

From a theoretical point of view, our main task is thus to find abstract self-organizing processes in which maps resembling the brain maps are formed, whereas it is of less interest whether the maps are formed by evolution, post-natal growth, or learning. Most self-organizing algorithms discussed in this book are mathematical processes for which the biological implementation need not be specified. Nonetheless the theoretical learning conditions seem to have counterparts in certain biological effects that we know; the particular physiological modeling attempt made in Ch. 4 must be understood as one case example, which does not necessarily exclude other explanations.

Perhaps the most intriguing theoretical finding is that a regular network structure, such as a two-dimensional lattice of neurons, can create ordered maps of any (metric) high-dimensional signal space, often in such a way that the main dimensions of the map thereby formed correspond to the most prominent features of the input signals. Examples are: maps of

- acoustical frequencies [2.37]
- phonemes of speech [2.37]
- color (hue and saturation) [2.70]
- elementary optical features [2.37]
- textures [2.71]
- geographic location [2.37]

etc.

In another important category of maps, the locations describe some *contextual features* that occur in *symbol strings.* For instance, if the symbols represent words of a natural language, such maps can identify the words as belonging to categories like nouns, verbs, adverbs, and adjectives [2.73–75] etc. There is some experimental evidence for such category maps existing in the brain, too [2.76–78].

In addition to serving as models for brain maps, the SOMs can be used as mathematical tools and they have practical applications ranging from speech recognition, image analysis, and industrial process control [2.72, 79] to the automatic ordering of document libraries, the visualization of financial records and the categorization of the electric signals in the brain; Ref. [2.80] lists almost 4000 works based on the SOM algorithm. The computed SOMs are very similar to many brain maps and they also behave dynamically in the same way, for example, their magnification is adjusted in proportion to the occurrences of the stimuli.

3. The Basic SOM

We shall now demonstrate a very important phenomenon that apparently has a close relationship to the brain maps discussed in Sect. 2.15 and occurs in certain spatially interacting neural networks. While being categorizable as a special kind of *adaptation*, this phenomenon is also related to *regression.* In regression, some simple mathematical function is usually fitted to the distribution of sample values of input data. The "nonparametric regression" considered in this chapter, however, involves fitting a number of *discrete, ordered reference vectors*, similar to the codebook vectors discussed in Sect. 1.5, to the distribution of vectorial input samples. In order to approximate continuous functions, the reference vectors are here made to define the nodes of a kind of hypothetical "elastic network," whereby the topological order characteristic of this mapping, and a certain degree of regularity of the neighboring reference vectors ensue from their *local interactions*, reflecting a kind of "elasticity." One possibility to implement such an "elasticity" would be to define the local interactions between the nodes *in the signal space* [3.1–7], whereas more realistic spatial interactions, from a neural modeling point of view, are definable between the neurons *along the neural network.* The latter approach is mainly made in this text.

Formation of a topologically ordered mapping from the signal space onto the neural network by this "regression" is already one interesting and important result, but from a practical point of view an even more intriguing result is that the various neurons develop into specific *decoders* or *detectors* of their respective signal domains in the input space. These decoders are formed onto the network *in a meaningful order*, as if some *feature coordinate system* were defined over the network.

The process in which such mappings are formed is defined by the *Self-Organizing Map (SOM)* algorithm that is the main subject of this book. The "feature maps" thereby realized can often effectively be used for the preprocessing of patterns for their recognition, or, if the neural network is a regular two-dimensional array, *to project and visualize high-dimensional signal spaces on such a two-dimensional display.* As a theoretical scheme, on the other hand, the adaptive SOM processes, in a general way, may explain the organizations found in various brain structures.

3.1 A Qualitative Introduction to the SOM

The Self-Organizing Map (SOM) is a new, effective software tool for the visualization of high-dimensional data. In its basic form it produces a *similarity graph of input data.* It converts the nonlinear statistical relationships between high-dimensional data into simple geometric relationships of their image points on a low-dimensional display, usually a regular two-dimensional grid of nodes. As the SOM thereby compresses information while preserving the most important topological and/or metric relationships of the primary data elements on the display, it may also be thought to produce some kind of *abstractions.* These two aspects, visualization and abstraction, can be utilized in a number of ways in complex tasks such as process analysis, machine perception, control, and communication.

In its present form the SOM was conceived by the author in 1982. At present, about 4000 research papers have been published on it; a list of them is available in the Internet [2.80]. Many tutorial texts and surveys on the SOM have appeared (Chap. 10). Several software packages that contain all the central procedures, a number of monitoring, diagnostic, and display programs, and exemplary data are freely available in the Internet, too (Chap. 8).

The SOM may be described formally as a nonlinear, ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, low-dimensional array. This mapping is implemented in the following way, which resembles the classical vector quantization (Sect. 1.5). Assume first for simplicity that the set of input variables $\{\xi_j\}$ is definable as a real vector $x = [\xi_1, \xi_2, \ldots, \xi_n]^T \in \Re^n$. With each element in the SOM array we associate a parametric real vector $m_i = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}]^T \in \Re^n$ that we call a *model.* Assuming a general distance measure between x and m_i denoted $d(x, m_i)$, the *image* of an input vector x on the SOM array is defined as the array element m_c that matches best with x, i.e., that has the index

$$c = \arg\min_i \{d(x, m_i)\} . \tag{3.1}$$

Differing from the traditional vector quantization, our task is to define the m_i in such a way that the mapping is *ordered* and descriptive of the distribution of x. Before proceeding further, it must be emphasized that the "models" m_i need not be vectorial variables. It will suffice if the distance measure $d(x, m_i)$ is defined over all occurring x items and a sufficiently large set of models m_i.

Consider Fig. 3.1 where a two-dimensional ordered array of nodes, each one having a general model m_i associated with it, is shown. The initial values of the m_i may be selected as random, preferably from the domain of the input samples. Then consider a list of input samples $x(t)$, where t is an integer-valued index. Let us recall that in this scheme, the $x(t)$ and m_i may be vectors, strings of symbols, or even more general items. Compare each $x(t)$ with all the m_i and copy each $x(t)$ into a sublist associated with that

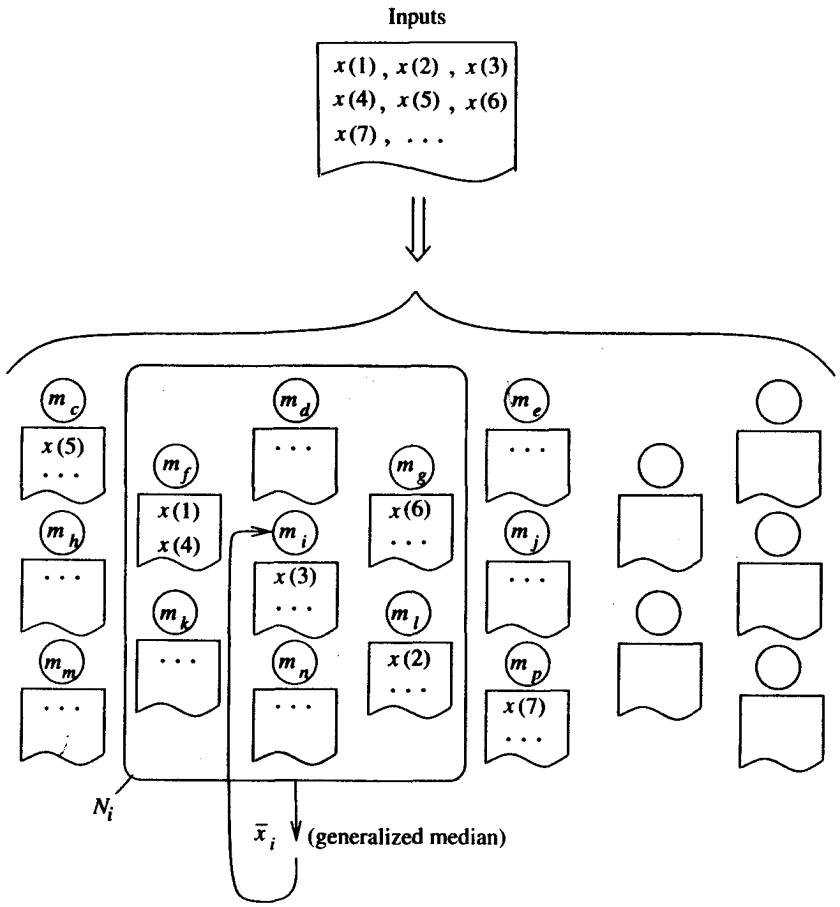


Fig. 3.1. Illustration of the batch process in which the input samples are distributed into sublists under the best-matching models, and then the new models are determined as (generalized) medians of the sublists over the neighborhoods N_i

node, the model vector of which is most similar to $x(t)$ relating to the general distance measure. When all the $x(t)$ have been distributed into the respective sublists in this way, consider the *neighborhood set* N_i around model m_i. Here N_i consists of all nodes up to a certain radius in the grid from node i. In the union of all sublists in N_i, the next task is to find the "middlemost" sample $\bar{x}_i$, defined as that sample that has the smallest sum of distances from all the samples $x(t), t \in N_i$. This sample $\bar{x}_i$ is now called the *generalized median* in the union of the sublists. (cf. Sect. 1.2.3) If $\bar{x}_i$ is restricted to being one of the samples $x(t)$, we shall indeed call it the *generalized set median;* on the other hand, since the $x(t)$ may not cover the whole input domain, it may be possible to find another item $\bar{x}_i'$ that has an even smaller sum of distances from the $x(t), t \in N_i$. For clarity we shall then call $\bar{x}_i'$ the *generalized median.*

Notice too that for the Euclidean vectors the generalized median is equal to their *arithmetic mean* if we look for an arbitrary Euclidean vector that has the smallest *sum of squares* of the Euclidean distances from all the samples $x(t)$ in the union of the sublists.

The next phase in the process is to form $\bar{x}_i$ or $\bar{x}'_i$ for each node in the above manner, always considering the neighborhood set N_i around each node i, and to replace each old value of m_i by $\bar{x}_i$ or $\bar{x}'_i$, respectively, in a simultaneous operation.

The above procedure shall now be iterated: in other words, the original $x(t)$ are again distributed into the sublists (which now change, because the m_i have changed), and the new $\bar{x}_i$ or $\bar{x}'_i$ are computed and made to replace the m_i, and so on. This is a kind of *regression process.*

There is an important question that we are not able to answer completely at the moment: even if we keep the $x(t)$ the same all the time, does this process converge? Do the m_i finally coincide with the $\bar{x}_i$? If the $x(t)$ and m_i are are Euclidean vectors, and a slightly modified (locally smoothed) distance measure is used in place of $d(x, m_i)$, the convergence has been proved by *Cheng* [3.8]. On the other hand, the original formulation of the SOM process that we shall discuss at length below is closely related to the above description.

Either the above process or the original SOM algorithm will then likely produce asymptotically converged values for the models m_i, the collection of which will approximate the distribution of the input samples $x(t)$, *even in an ordered fashion.* Let us look at Fig. 3.2, which represents the self-organized map of short-time acoustic spectra, viz. those taken from the (Finnish) speech at 20 ms intervals. The round symbols stand for the nodes of the SOM, and the curves inside them are *models* of spectra: low frequencies are on the left, high frequencies on the right, and the ordinates of the curve correspond to the voice intensity at the various spectral channels. One can immediately discern the similarity of the spatially adjacent models: the neighboring models look more similar than those farther apart. The collection of the models is also supposed to approximate the distribution of the input spectra.

At the first glance one might also think that the global order in the "map" reflects some kind of *harmonicity*: it seems as if every model were the average of the neighboring models, like in the theory of the harmonic functions. On a closer look, however, one can find several properties of the SOM that differ from the harmonicity. First of all, *there are no fixed boundary values for the models at the edges*: the values are determined freely in the regression process, when the neighborhood set N_i of the edge nodes is made to contain nodes from the inside of the grid. The nature of regression is slightly different at the edges and in the inside of the grid, resulting in certain border effects, Another deviation from the harmonicity, as later demonstrated by numerous examples, is that there are areas in the map where the models are very similar, but then there are again places where a bigger "jump" between the neighboring

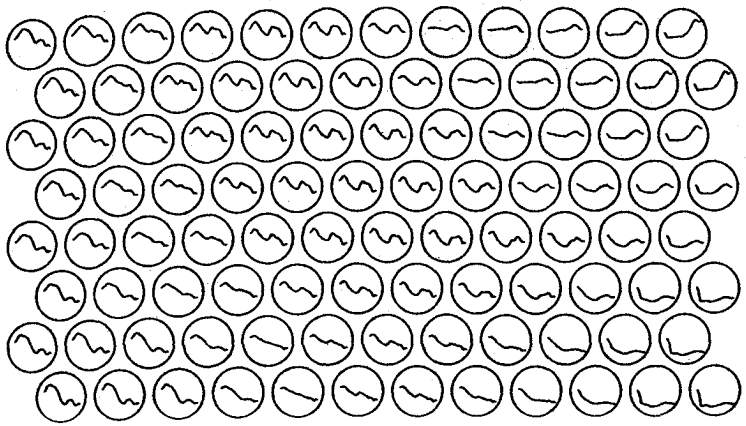


Fig. 3.2. In this exemplary application, each processing element in the hexagonal grid holds a model of a short-time spectrum of natural speech (Finnish). Notice that neighboring models are mutually similar

models is discernible. If the collection of the models has to approximate the distribution of the inputs, such uneven areas must exist in the map.

On the other hand, if one considers the process depicted in Fig. 3.1, one can easily realize how the definition of "harmonicity" must be modified in order to describe the input data: *the collection of models is ordered by definition, if each model is equal to the average of input data mapped to its neighborhood.*

3.2 The Original Incremental SOM Algorithm

We shall now proceed to the theory of the Self-Organizing Maps using the original SOM algorithm as the starting point. This algorithm can be seen to define a special recursive regression process, in which only a subset of models is processed at every step.

Consider Fig. 3.3. The SOM here defines a mapping from the input data space $\Re^n$ onto a two-dimensional array of nodes. With every node i, a parametric *model vector*, also called *reference vector* $m_i = [\mu_{i1}, \mu_{i2}, \ldots, \mu_{in}]^T \in \Re^n$ is associated. Before recursive processing, the m_i must be *initialized.* In the preliminary examples we select random numbers for the components of the m_i to demonstrate that starting from an arbitary initial state, in the long run the m_i will attain two-dimensionally ordered values. This is the basic effect of self-organization. Later in Sect. 3.7 we shall point out that if the initial values of the m_i are selected as regular, the process can be made to converge much faster.

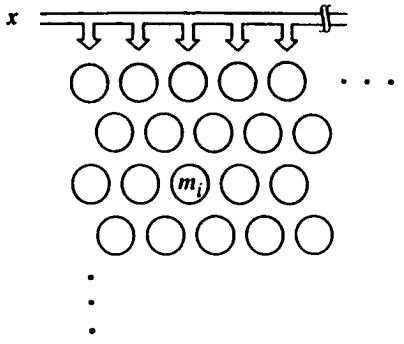


Fig. 3.3. The array of nodes (neurons) in a two-dimensional SOM array. The underlying mechanisms for parallel comparison of x and the m_i are not shown

The lattice type of the array can be defined to be rectangular, hexagonal, or even irregular; hexagonal is effective for visual display. In the simplest case, an input vector $x = [\xi_1, \xi_2, \ldots, \xi_n]^T \in \Re^n$ is connected to all neurons in parallel via variable scalar weights μ_{ij}, which are in general different for different neurons. In an abstract scheme it may be imagined that the input x, by means of some parallel computing mechanisms, is compared with all the m_i, and the *location of best match* in some metric is defined as the location of the "response." We shall later in Chap. 4 demonstrate what kind of physical, eventually physiological parallel computing mechanism is able to do this. By computer programming, of course, location of the best match is a trivial task. The exact *magnitude* of the response need not be determined: the input is simply mapped onto this *location*, like in a set of *decoders*.

Let $x \in \Re^n$ be a stochastic data vector. One might then say that the SOM is a "nonlinear projection" of the probability density function $p(x)$ of the high-dimensional input data vector x onto the two-dimensional display. Vector x may be compared with all the m_i in any metric; in many practical applications, the smallest of the *Euclidean distances* $||x - m_i||$ can be made to define the *best-matching node*, signified by the subscript c:

$$c = \underset{i}{\text{argmin}}\{||x - m_i||\} \ , \quad \text{which means the same as}$$
$$||x - m_c|| = \min_i \{||x - m_i||\} \ . \tag{3.2}$$

In Sect. 3.3 and Chap. 4 we shall consider another, more "biological" matching criterion, based on the *dot product* of x and m_i.

During *learning*, or the process in which the "nonlinear projection" is formed, those nodes that are *topographically close in the array up to a certain geometric distance* will activate each other to learn something from the same input x. This will result in a local *relaxation* or *smoothing effect* on the weight vectors of neurons in this neighborhood, which in continued learning leads to *global ordering*. Consider the eventual convergence limits of the following

learning process, whereupon the initial values of the $m_i(0)$ can be arbitrary, e.g., random:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \ , \tag{3.3}$$

where $t = 0, 1, 2, \ldots$ is an integer, the discrete-time coordinate. In the relaxation process, the function $h_{ci}(t)$ has a very central role: it acts as the so-called *neighborhood function*, a smoothing kernel defined over the lattice points. For convergence it is necessary that $h_{ci}(t) \to 0$ when $t \to \infty$. Usually $h_{ci}(t) = h(||r_c - r_i||, t)$, where $r_c \in \Re^2$ and $r_i \in \Re^2$ are the location vectors of nodes c and i, respectively, in the array. With increasing $||r_c - r_i||$, $h_{ci} \to 0$. The average width and form of h_{ci} define the "stiffness" of the "elastic surface" to be fitted to the data points.

In the literature, two simple choices for $h_{ci}(t)$ occur frequently. The simpler of them refers to a *neighborhood set* of array points around node c (Fig. 3.4). Let their index set be denoted N_c (notice that we can define $N_c = N_c(t)$ as a function of time), whereby $h_{ci}(t) = \alpha(t)$ if $i \in N_c$ and $h_{ci}(t) = 0$ if $i \notin N_c$. The value of $\alpha(t)$ is then identified with a *learning-rate factor* $(0 < \alpha(t) < 1)$. Both $\alpha(t)$ and the radius of $N_c(t)$ are usually decreasing monotonically in time (during the ordering process).

Another widely applied, smoother neighborhood kernel can be written in terms of the Gaussian function,

$$h_{ci}(t) = \alpha(t) \cdot \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) \ , \tag{3.4}$$

where $\alpha(t)$ is another scalar-valued "learning-rate factor," and the parameter $\sigma(t)$ defines the width of the kernel; the latter corresponds to the radius of $N_c(t)$ above. Both $\alpha(t)$ and $\sigma(t)$ are some monotonically decreasing functions of time.

The algorithm chosen here for preliminary simulations is only representative of many alternative forms. If the SOM network is not very large (say, a few hundred nodes at most), selection of process parameters is not very crucial. We can also use the simple neighborhood-set definition of $h_{ci}(t)$.

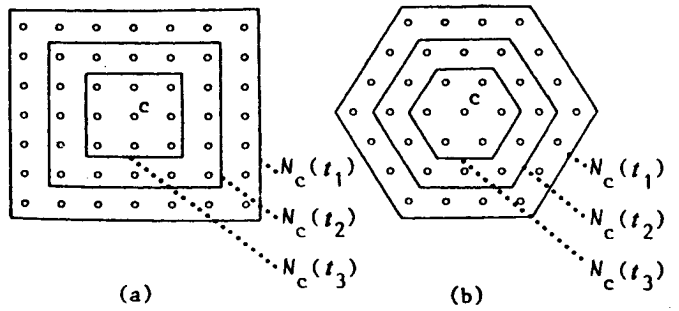


Fig. 3.4. a, b. Two examples of topological neighborhood ($t_1 < t_2 < t_3$)

Special caution, however, is required in the choice of the size of $N_c = N_c(t)$. If the neighborhood is too small to start with, the map will not be ordered globally. Instead various kinds of mosaic-like parcellations of the map are seen, between which the ordering direction changes discontinuously. This phenomenon can be avoided by starting with a fairly wide $N_c = N_c(0)$ and letting it *shrink* with time. The initial radius of N_c can even be more than half the diameter of the network! During the first 1000 steps or so, when the proper ordering takes place, and $\alpha = \alpha(t)$ is fairly large, the radius of N_c can shrink linearly to, say, one unit; during the fine-adjustment phase, N_c can still contain the nearest neighbors of cell c.

If the initial values have been selected at random, for approximately the first 1000 steps, $\alpha(t)$ should have reasonably high values (close to unity), thereafter decreasing monotonically. An accurate time function is not important: $\alpha = \alpha(t)$ can be linear, exponential, or inversely proportional to t. For instance, $\alpha(t) = 0.9(1 - t/1000)$ may be a reasonable choice. The *ordering* of the m_i occurs during this initial period, while the remaining steps are only needed for the fine adjustment of the map. After the ordering phase, $\alpha = \alpha(t)$ should attain small values (e.g., of the order of or less than .02) over a long period. Neither is it crucial whether the law for $\alpha(t)$ decreases linearly or exponentially during the final phase.

With very large maps, however, it may be important to minimize the total learning time. Then, selection of an optimal $\alpha(t)$ law may be crucial; cf. Sect. 3.8, where we shall consider "optimal" choices, essentially inversely proportional to t. Effective choices for these functions and their parameters have so far only been determined experimentally.

Since learning is a stochastic process, the final statistical accuracy of the mapping depends on the number of steps in the final convergence phase, which must be reasonably long; there is no way to circumvent this requirement. A "rule of thumb" is that, for good statistical accuracy, the number of steps must be at least 500 times the number of network units. On the other hand, the number of components in x has no effect on the number of iteration steps, and if hardware neural computers are used, a very high dimensionality of input is allowed. Typically we have used up to 100000 steps in our simulations, but for "fast learning," e.g., in speech recognition, 10000 steps and even less may sometimes be enough. Note that the algorithm is computationally extremely light. If only relatively few samples are available, they must be recycled for the desired number of steps.

Comment. In the preliminary examples described first our motive is to demonstrate that, starting from a random initial state, the mapping will be ordered in a finite number of learning steps. However, in practical applications one can start from an initial state that is already ordered and roughly complies with the input density function. If this is done (cf. Sect. 3.7), the learning process converges rapidly even if the neighborhood function were very narrow (of the order of its final form) and $\alpha(t)$ would start with low values, say, .2 or .1.

Examples of Ordering. It may be quite surprising that when starting with random $m_i(0)$, the reference vectors will attain *ordered* values in the long run, even in high-dimensional spaces. This ordering is first illustrated by means of two-dimensional input data $x = [\xi_1, \xi_2]^T \in \Re^2$ that have some arbitrarily structured distribution. For simplicity, if x is a stochastic vector, its probability density function $p(x)$ is in this example assumed uniform within the framed areas in Fig. 3.5 and zero outside them. The topological relations between the neurons in a square array can be visualized by auxiliary lines that are drawn between the neighboring reference or codebook vectors (points in the signal space). The reference vectors in these graphs now correspond to the crossings and end points of this network of auxiliary lines, whereby the relative topological order becomes immediately visible.

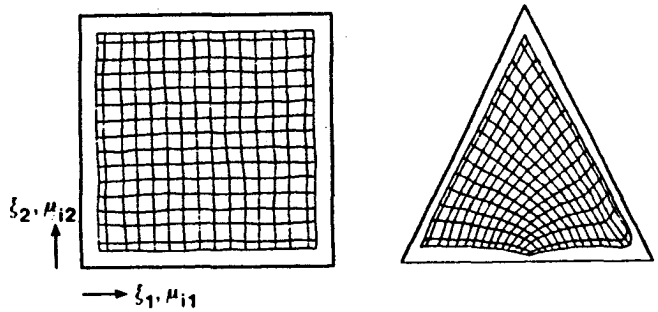


Fig. 3.5. Two-dimensional distributions of input vectors (framed areas), and the networks of reference vectors approximating them

The codebook vectors, while being ordered, also tend to approximate $p(x)$, the probability density function of x. This approximation, however, is not quite accurate, as will be seen later.

The examples shown in Fig. 3.5 represent the approximately *converged* state of the weight vectors. The different units have clearly become sensitized to different domains of input vectors in an orderly fashion. There is a boundary effect visible in Fig. 3.5, a slight contraction of the edges of the maps. We shall analyze this effect in Sect. 3.5.1 On the other hand, the density of weight vectors is correspondingly higher around the contraction. The relative contraction effect diminishes with increasing size of the array.

Examples of intermediate phases that occur during the self-organizing process are given in Figs. 3.6 and 3.7. The initial values $m_i(0)$ were selected at random from a certain (circular) support of values, and the structure of the network becomes visible after some time. Notice that the array can be, e.g., one-dimensional although the vectors are two-dimensional, as shown in

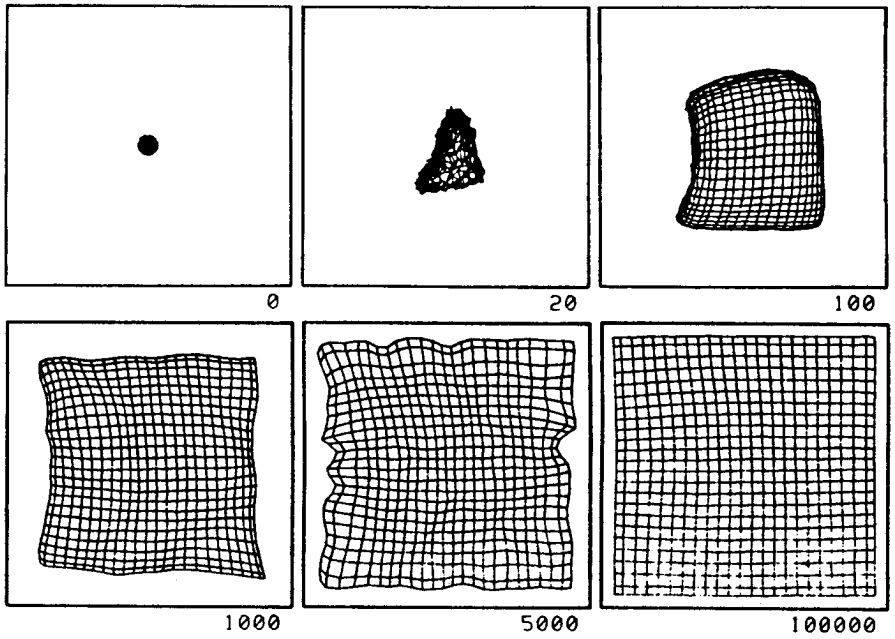


Fig. 3.6. Reference vectors during the ordering process, square array. The numbers at lower right-hand corner indicate learning cycles

Fig. 3.7. The "order" thereby created resembles a Peano curve, or fractal form.

Calibration. When a sufficient number of input samples $x(t)$ has been presented and the $m_i(t)$, in the process defined by (3.2) and (3.3), have converged to practically stationary values, the next step is *calibration* of the map, in order to locate images of different input data items on it. In practical applications for which such maps are used, it may be self-evident how a particular input data set ought to be interpreted and labeled. By inputting a number of typical, manually analyzed data sets, looking where the best matches on the map according to Eq. (3.2) lie, and labeling the map units correspondingly, the map becomes calibrated. Since this mapping is assumed to be continuous along a hypothetical "elastic surface", the unknown input data are approximated by the closest reference vectors, like in vector quantization.

Comment. An "optimal mapping" might be one that projects the probability density function $p(x)$ in the most "faithful" fashion, trying to preserve at least the local structures of $p(x)$ in the output plane. (You might think of $p(x)$ as a flower that is pressed!)

It has to be emphasized, however, that description of the *exact* form of $p(x)$ by the SOM is not the most important task. It will be pointed out that the SOM automatically finds those *dimensions* and *domains* in the signal

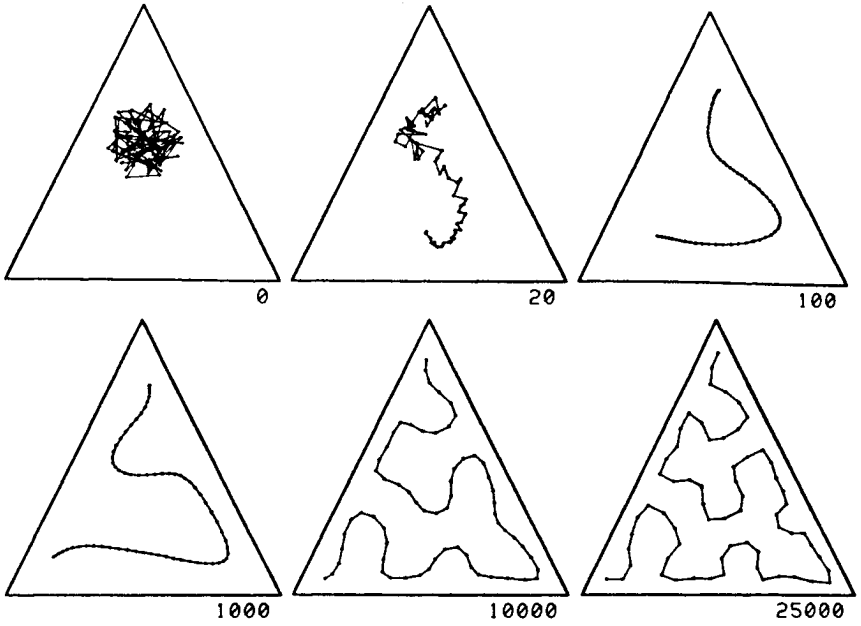


Fig. 3.7. Reference vectors during the ordering process, linear array

space where x has significant amounts of sample values, conforming to the usual philosophy in regression problems.

3.3 The "Dot-Product SOM"

It has sometimes been suggested that x be normalized before it is used in the algorithm. Normalization is not necessary in principle, but it may improve numerical accuracy because the resulting reference vectors then tend to have the same dynamic range.

Another aspect is that it is possible to apply many different metrics in matching; then, however, the matching and updating laws should be mutually *compatible* with respect to the same metric. For instance, if the dot-product definition of similarity of x and m_i is applied, the learning equations should read

$$x^{\mathrm{T}}(t)m_c(t) = \max_i\{x^{\mathrm{T}}(t)m_i(t)\}\,, \tag{3.5}$$

$$m_i(t+1) = \begin{cases} \dfrac{m_i(t) + \alpha'(t)x(t)}{\|m_i(t) + \alpha'(t)x(t)\|} & \text{if } i \in N_c(t)\,, \\ m_i(t) & \text{if } i \notin N_c(t)\,, \end{cases} \tag{3.6}$$

and $0 < \alpha'(t) < \infty$; for instance, $\alpha'(t) = 100/t$. This process automatically normalizes the reference vectors at each step. The normalization computations slow down the training algorithm. On the other hand, during matching, the dot product criterion applied during recognition is very simple and fast, and amenable to many kinds of simple analog computation, both electronic and optical. It also seems to have a connection to physiological processes, as later discussed in Chap. 4.

Besides the Euclidean distances and dot products, many other matching criteria can be used with the SOM. Sect. 3.12 points out how often other SOM algorithms may be derivable.

This author has frequently visualized the self-organizing process by means of a mechanical gadget, depicted in Fig. 3.8. A regular 4 by 4 array of rotary disks, each representing a neuron, has been pinned onto a substrate. An arrow, representing a normalized two-dimensional reference vector m_i, is painted on each disk. The initial orientations of the reference vectors can be randomized. The sequence of training vector values is selected at random, too, to represent samples of $p(x)$. The "winner" neuron is the one that has the smallest angle of its reference vector with the input vector; the reference vectors of all neurons in the neighborhood set (enclosed by the set line in Fig. 3.8) are corrected by a fraction of the angles between x and the m_i. After a few corrective steps the m_i values start looking smoothed and ordered, as seen from the series of pictures in Fig. 3.8.

3.4 Other Preliminary Demonstrations of Topology-Preserving Mappings

3.4.1 Ordering of Reference Vectors in the Input Space

The computer simulations given in this section will further illustrate the effect that the reference vectors tend to approximate various distributions of input vectors in an orderly fashion. In most examples, the input vectors were chosen two-dimensional for visual display purposes, and their probability density function was uniform over the area demarcated by its borderlines. Outside these borders the probability density function had the value zero. The vectors $x(t)$ were drawn from these density functions independently, at random, whereafter they caused adaptive changes in the reference vectors m_i.

The m_i vectors have mostly been illustrated as points in the same coordinate system where the $x(t)$ are represented; like earlier, in order to indicate to which unit each m_i belongs, the end points of the m_i have been connected by a lattice of lines that conforms to the topology of the processing unit array. A line connecting two weight vectors m_i and m_j is thus only used to indicate that the two corresponding units i and j are adjacent in the "neural" network.

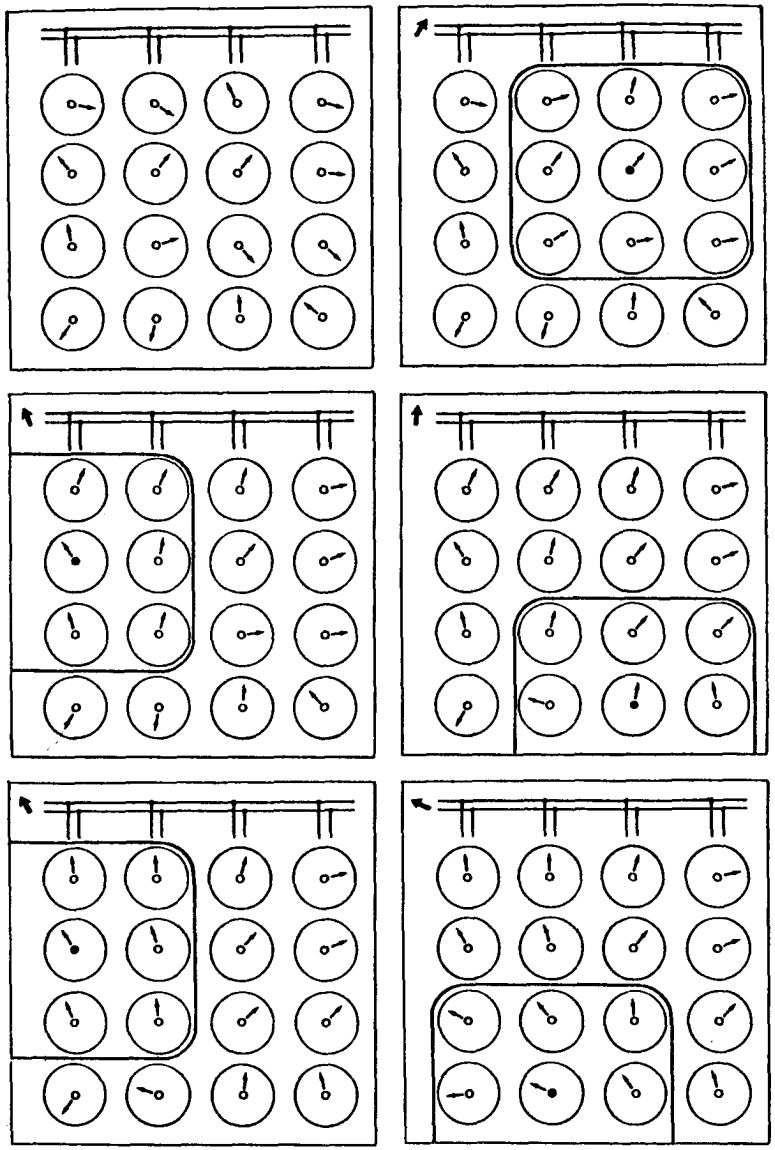


Fig. 3.8. This sequence of pictures illustrates successive phases in the operation of a two-dimensional dot-product SOM with two-dimensional input and reference vectors (arrows). The first picture represents the initial state. In the rest of the pictures, the small arrow in the upper left corner represents the input vector x, which specifies the best-matching reference vector, "winner". A set line drawn around the "winner" defines the neighborhood set N_c, in which (in this demonstration) the correction of each arrow was half of the angular difference of x and m_i. A smooth order of the m_i is seen to emerge

The results are still more interesting if the input vectors and the array have different dimensionalities: Fig. 3.9 illustrates a case in which the vectors were three-dimensional, but the array was two-dimensional. For clarity, $p(x)$ and the network formed of the m_i have been drawn separately, as explained in the caption of Fig. 3.9.

Comments. One remark ought to be made now in relation to Fig. 3.9. Some reference vectors seem to have remained outside $p(x)$. This, however, should not be interpreted as an error, as long as the SOM "net" is regarded to have a certain degree of "stiffness" and represent a nonparametric *regression*. In Sect. 5.4 we shall demonstrate how the SOM could be made to describe a structured $p(x)$ better, but is it then any longer a regression?

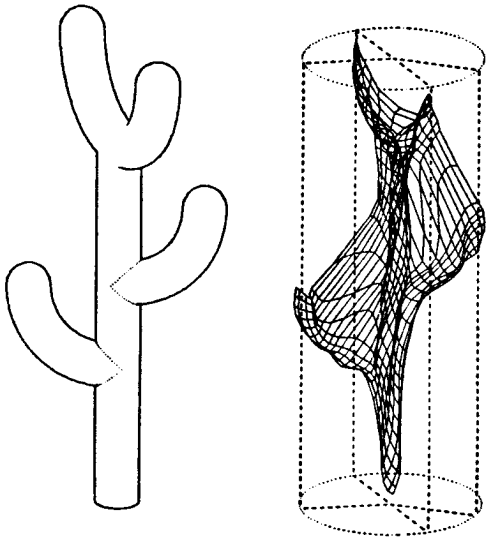


Fig. 3.9. SOM for a structured distribution of $p(x)$. For clarity, the three-dimensional $p(x)$, having uniform density value inside the "cactus" shape and zero outside it, is shown on the left, whereas the "net" of reference vectors is displayed on the right in a similar coordinate system

Before proceeding further one should also note that there is yet no factor present that would define a particular *orientation* of the map. Accordingly, the latter can be realized in the process in any mirror- or point-symmetric inversion. If a particular orientation had to be favored, the easiest way to reach this result would be to choose the initial values $m_i(0)$ asymmetrically. Since the symmetries may not be as interesting as the self-organizing behaviour itself, we shall ignore them in the preliminary examples at least. Practical questions of this kind will be discussed in Sect. 3.13.

The SOM as a Nonlinear, Adaptive Projection Screen. Figure 3.9 also illustrates another aspect of the SOM, namely, that of a *nonlinear projection*. One may regard the elastic network as a flexible *projection screen* that is first fitted through the distribution of the data points. After that, any point of $\Re^3$ (or in the general case, of an arbitrary-dimensional input space) will become projected onto the closest node of the "screen." This projection may not be orthogonal, though, because the screen is not continuous: it is only approximately orthogonal.

Automatic Selection of Feature Dimensions. There are two opposing tendencies in the self-organizing process. First, the set of weight vectors tends to describe the density function of the input vectors. Second, local interactions between processing units tend to preserve *continuity* in the double (two-dimensional) sequences of weight vectors. A result of these opposing "forces" is that the reference vector distribution, tending to approximate a smooth hypersurface, *also seeks an optimal orientation and form in the pattern space that best imitates the overall structure of the input vector density.*

A very important detail of the above reference-vector distribution is that it automatically tends to find those two dimensions of the pattern space where the input vectors have a high variance and which, accordingly, ought to be described in the map. As this effect might otherwise remain a bit obscure, the following extremely simple experiment is intended to illustrate what is meant. It is believed that the result that is here demonstrated using a one-dimensional topology (linear array of processing units) and a simple two-dimensional input density function, is easily generalizable for higher-order topology, arbitrary dimensionality of the input-vector density function, and structured distribution of the input samples.

Assume that the system consists of only *five* neurons connected as a linear open-ended array. Their reference vectors $m_i = [\mu_{i1}, \mu_{i2}]^T, i = 1, 2, \ldots, 5$ and the components of the input vectors $x = [\xi_1, \xi_2]^T$ are represented as an already familiar illustration in Fig. 3.10. The variances of ξ_1 and ξ_2 are now selected differently as shown by the borderlines in Fig. 3.10. As long as one of the variances is significantly higher, the weight vectors form an almost straight line that is aligned in the direction of the greater variance.

On the other hand, if the variances are almost equal, or if the length of the array is much greater than the range of lateral interaction, the straight form of the distribution is switched into a "Peano curve". The transition from straight to curved line is rather sharp, as shown in Fig. 3.11. Here the variances are fixed but the length of the array is varied; the borders of the Voronoi sets have also been drawn to the picture.

The next picture, Fig. 3.12 further illustrates what may happen when the input vectors have a higher dimensionality than the network topology (which in this case was two). As long as variance in the third dimension (ξ_3) is small enough, the map remains straight. However, with increasing variance and short lateral interaction range the map tends to become corrugated, and in

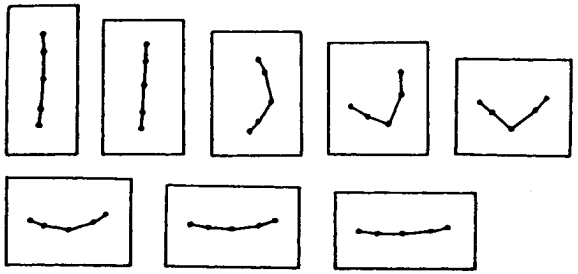


Fig. 3.10. Automatic selection of dimensions for mapping

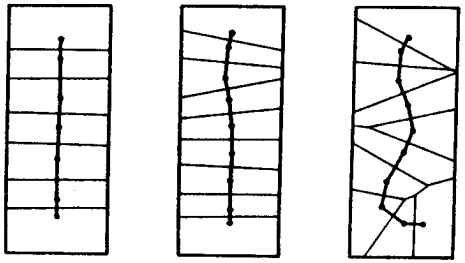


Fig. 3.11. Distribution of reference vectors with different length of a linear array

this connection one should note the "zebra stripes" that have been found in brain maps experimentally. *Here the "stripes" have a very simple and natural explanation, namely, they occur whenever a two-dimensional map tries to approximate a higher-dimensional signal distribution which has significant variance in more than two dimensions.*

The reader might be curious to know the mathematical explanation of this intriguing self-ordering effect. The phenomenon is actually rather delicate and needs a lengthy discussion, as will be seen in the rest of this book. We shall start the mathematical discussion in Sect. 3.5.

3.4.2 Demonstrations of Ordering of Responses in the Output Space

All simulations reported in this subsection were performed in our group in 1982, and most of them appeared in [2.37].

"The Magic TV". The following demonstration, Fig. 3.13 shows another, more concrete example of self-organization. It describes a hypothetical image-transferring system (dubbed here "The Magic TV") in which there are no control mechanisms for the encoding of the position of picture elements, but where the order of image points in the output display automatically results in the self-organizing learning process from their topological constraints. The result is a one-to-one mapping of the points of the input plane onto the points

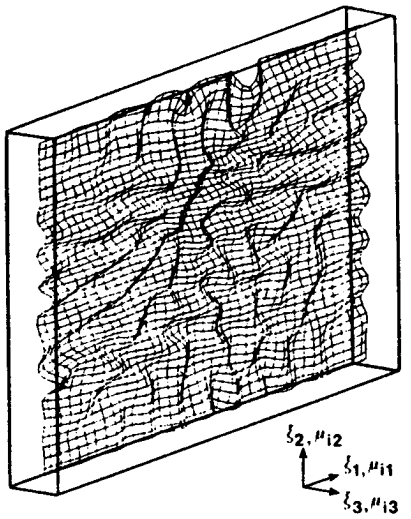
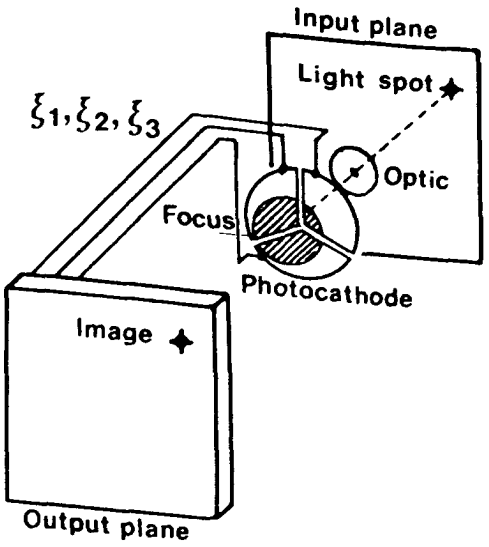


Fig. 3.12. Formation of "stripes"

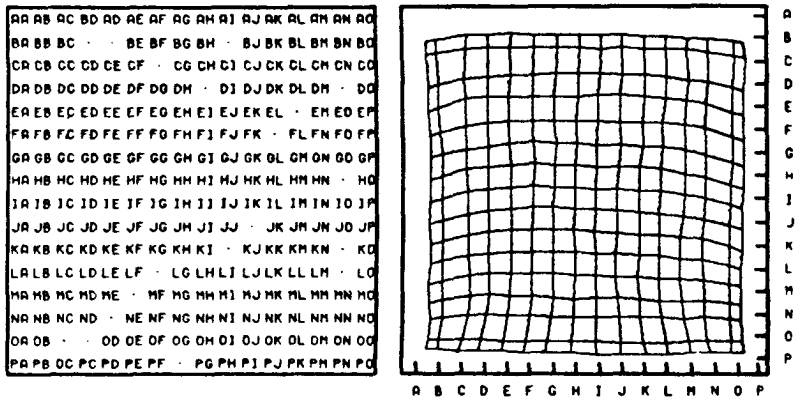
in the output plane; in the transmission of the signals, however, this order was never specified explicitly. The system has to deduce the order gradually from the relations that are implicit in the transmitted signals. This system consists of a primitive "TV camera" and an adaptive system of the above type. The camera is thought to have a very poor optical system, such that whenever a spot of light appears on the input plane, it forms a very diffuse focus onto the photocathode. Assume that the cathode has three sectors, each producing a signal directly proportional to the area that is illuminated by the broad focus. Now let the light spot move at random in different points in the input plane, with a probability density that is uniform over a square area. The resulting signals ξ_1, ξ_2, and ξ_3 are then transmitted to the processing-unit array, where they cause adaptive changes. Let this process continue for a sufficient time, after which a test of the system is performed. This test is accomplished, e.g., by recording the output of each unit in turn and *locating that point in the input plane where the light spot must be in order to cause the best match in the unit under consideration.*

The resulting output map can be tested in either of the following ways: A) One can look at which processing unit each of the test vectors in turn has the best match, and call this unit the *image* of the test vector. The array is labeled accordingly. B) One can test to which training vector (with known classification) each of the units has become most sensitive, making the best match.

Obviously these tests are related, but the respective output maps look slightly different. With Test A, two or more test vectors may become mapped onto the same unit while the classification of some units may be left undefined.



(a)



(b) (c)

Fig. 3.13. "The Magic TV" **(a)** System **(b)** Output plane. Pairs of letters correspond to processing units, labeled by the images of test vectors (with coordinates defined in **(c)**). **(c)** Input plane showing those points to which the various processing units (corresponding to nodes of the net) have become most sensitive

Test B always defines a unique matching input to every output unit but it eventually happens that some input vectors, especially near the edges, do not become mapped at all.

Mapping by a Feeler Mechanism. The purpose of this example is to demonstrate that a map of the environment of a subject can be formed in

a self-organizing process whereby the observations can be mediated by very rude, nonlinear, and mutually dependent mechanisms such as arms and detectors of their turning angles. In this demonstration, two artificial arms, with two joints each, were used for the feeling of a planar surface. The geometry of the setup is illustrated in Fig. 3.14. The tips of both arms touched the same point on the plane, which during the training process was selected at random, with a uniform probability density over the framed area. At the same time, two signals, proportional to the bending angles, were obtained from each arm; these signals were led to a self-organizing array of the earlier type, and adaptation of its parameters took place.

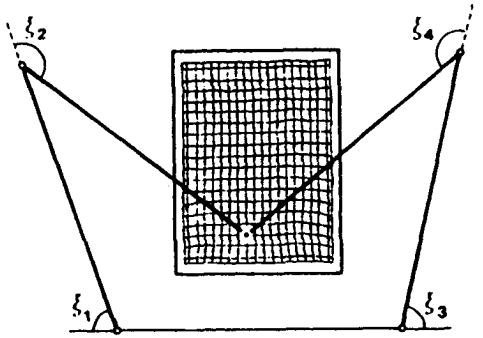


Fig. 3.14. Map of a feeler mechanism. The network of lines drawn onto the input plane shows a "virtual image" of the weight vectors: a crossing in this network is the input point to which the corresponding neuron in the SOM array is "tuned"

The lattice of lines that has been drawn onto the framed area in this picture represents a *virtual image* of the reference vectors, i.e., showing to which point on the plane each unit is most sensitive. When this point is touched, the corresponding processing unit makes the best match between x and m_i. One might also define the map so obtained as the map of the *receptive fields* of the array units. It can be tested for both arms separately, i.e., by letting each of them to touch the plane and looking at which point it had to be in order to cause the maximum response at a particular unit. These two maps coincide almost perfectly.

Formation of a Topographic Map of the Environment Through Many Different Channels. This example elucidates alternative mechanisms that are able to create the perception of space. In the following simulation, observations of the environment were mediated by one eye and two arms. Both arms touched some point on the input plane to which the gaze was also directed. The turning angles of the eye, and the bending angles of the arms were detected. Notice that no image analysis was yet performed. Every point on the input plane corresponds to six transmitted signals used for the ξ_i variables, $i = 1, \ldots, 6$.

Training was made by selecting the target point at random, and letting the resulting ξ_i signals affect the adaptive system. The asymptotic state was then tested *for each arm and the eye separately.* In other words, when the output of each processing unit was recorded in turn, one could, e.g., find the direction of gaze that caused the maximum match at this unit. (The other signals were thereby zero.) Figure 3.15 shows the results of this test, again projected on the target plane as a network of thin lines; their crossings correspond to processing units, and these networks have been determined as inverse mappings of the due weight vectors. It is remarkable that practically the same input-output mapping is obtained with very different nonlinear detectors.

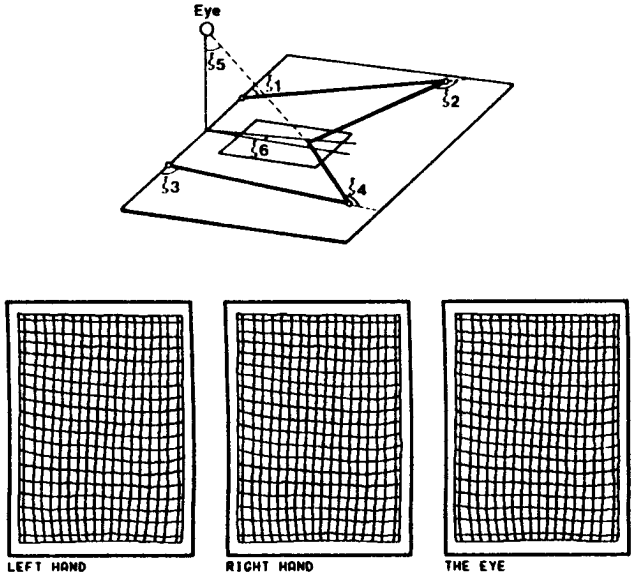


Fig. 3.15. Formation of a topographic map of the environment using one eye and two hands simultaneously

Construction of a Topographic Map of Stage by a Freely Moving Observer Using Vision Only. In this demonstration we show results of an experiment that was intended to illustrate how a complete image of space might be formed from its partial observations. Here the observatory mechanism was a very trivial one, consisting of a simple optic and a cylindrical retina with eight photosensitive segments. Consider Fig. 3.16 that shows the top view of a stage provided with walls, the scenery. Assume that the background is dark and the walls are white; a projection image of the scenery is formed onto the retina. The observers moves at random within a restricted area, and in this simple demonstration his orientation was fixed. The signals obtained from the segments of the retina are led to a self-organizing processing unit array. After adaptation, the map was tested by letting the observer stand in a particular location and recording the coordinates of the corresponding image point in the map.

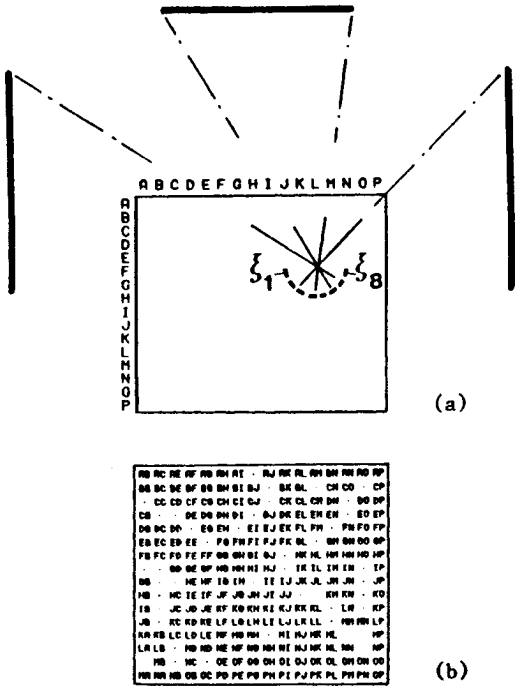


Fig. 3.16. Topographic map of a stage. **(a)** Thick lines: scenery. The framed area is where the observer moved: the coordinates are indicated by a pair of letters. **(b)** Output map, with pairs of letters corresponding to processing units. The units are labeled by coordinates of those points on the stage the images of which are formed on the units in question

Tonotopic Map. The following effect was found already in the first SOM experiments, but it has later been ignored. Nonetheless it demonstrates a very important feature of self-organization, namely that the inputs to the map units need not be identical, not even having the same dimensionality, as long as some kind of metric-topological order of the various inputs to the different "neurons" is preserved. This property is necessary in the biological models, because the number of inputs to a neuron is never fixed. Let us demonstrate this effect with a simple model that refers to a possible mode of formation, or at least refinement, of the physiological *tonotopic map.* This experiment demonstrates that the inputs can be nonidentical as long as the signals to every unit are correlated, and the topological order of their input vectors in the respective signal subspaces is the same. Consider Fig. 3.17 that depicts

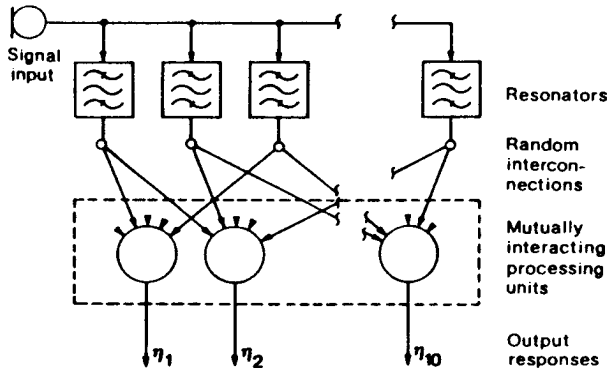


Fig. 3.17. System for tonotopic-map simulation

Table 3.1. Formation of frequency maps. There were twenty second-order filters with quality factor $Q = 2.5$ (where Q is inversely proportional to the relative width of resonance curve) and resonant frequencies distributed at random over the range [1, 2]. The training frequencies were drawn at random from the range [0.5, 1]. The numbers in the table indicate those test frequencies to which each processing unit became most sensitive.

Unit	1	2	3	4	5	6	7	8	9	10
Experiment 1, 2000 training steps	0.55	0.60	0.67	0.70	0.77	0.82	0.83	0.94	0.98	0.83
Experiment 2, 3500 training steps	0.99	0.98	0.98	0.97	0.90	0.81	0.73	0.69	0.62	0.59

a one-dimensional array of processing units. This system receives sinusoidal signals and becomes ordered according to their *frequency*. Assume a set of resonators or bandpass filters tuned at random. The filters may have a rather shallow resonance curve. The inputs to the array units (five to each) are now also picked up at random from the resonator outputs, different samples for different array units, so that there is no order or correlation in any structure or initial parameters. Next a series of adaptation operations is carried out, every time generating a new sinusoidal signal with a randomly chosen frequency. After a number of iteration steps, the units start to become sensitized to different frequencies in an ascending or descending order. Final results of two experiments are shown in Table 3.1.

Phonotopic Map. A more practical example is mapping of natural stochastic data, such as short-time spectra picked up from natural speech, onto the network. In this example the input vector x was 15-dimensional, and its components corresponded to the output powers of a 15-channel frequency filter

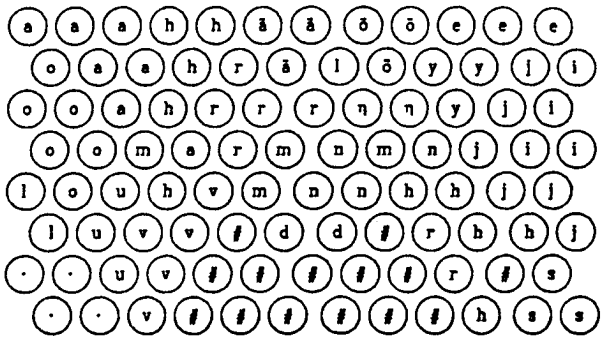


Fig. 3.18. The neurons, shown as circles, are labeled with the symbols of the phonemes with which they made the best match. Distinction of /k,p,t/ from this map is not reliable, and this phonemic class is indicated by symbol #. An analysis of the transient spectra of these phonemes by an auxiliary map is necessary

bank, averaged over 10 ms intervals, with the midfrequencies of the filters being selected from the range 200 Hz to 6400 Hz. The spectral samples were applied in their natural order of occurrence at the inputs of the SOM, and after learning the map was calibrated with known input samples and labeled according to the phonemic symbols of the samples. Details of the system used for the acquisition of speech samples will be given in Sects. 7.2 and 7.5. Figure 3.18 shows the result directly *on the neural network*, not in the signal space. The various neurons have become "tuned" to different categories of phonemes. (The speech used in this experiment was Finnish, phonetically rather similar to Latin.) The cells labeled with the same phonemic symbol actually correspond to somewhat different reference or codebook vectors, which like the various codebook vectors in VQ approximate the probability density function of a particular set of samples. As /k,p,t/ have a very weak signal power compared with the other phonemes, they are clustered together and represented by a broad phonetic class with symbol "#."

3.5 Basic Mathematical Approaches to Self-Organization

Although the basic principle of the above system seems simple, the process behaviour, especially relating to the more complex input representations, has been very difficult to describe in mathematical terms. The first approach made below discusses the process in its simplest form, but it seems that fundamentally similar results are obtainable with more complex systems, too. The other approaches made in this section are also simple, meant for basic understanding. More refined discussions can be found in the literature (Chap. 10).

3.5.1 One-Dimensional Case

We shall first try to justify the self-organizing ability analytically, using a very simple system model. The reasons for the self-ordering phenomena are actually very subtle and have strictly been proven only in the simplest cases. In this section we shall first delineate a basic Markov-process explanation that should help to understand the nature of the process.

In the first place we shall restrict our considerations to a *one-dimensional, linear, open-ended array* of functional units to each of which a *scalar-valued input signal* ξ is connected. Let the units be numbered $1, 2, \ldots, l$. Each unit i has a single scalar input weight or reference value μ_i, whereby the similarity between ξ and μ_i is defined by the absolute value of their difference $|\xi - \mu_i|$; the best match is defined by

$$|\xi - \mu_c| = \min_i \{|\xi - \mu_i|\} \; . \tag{3.7}$$

We shall define the set of units N_c selected for updating as follows:

$$N_c = \{\max(1, c-1), c, \min(l, c+1)\} \; . \tag{3.8}$$

In other words, unit i has the neighbors $i-1$ and $i+1$, except at the end points of the array, where the neighbor of unit 1 is 2, and the neighbor of unit l is $l-1$, respectively. Then N_c is simply the set of units consisting of unit c and its immediate neighbors.

The general nature of the process is similar for different values of $\alpha > 0$; it is mainly the speed of the process that is varied with α. In the *continuous-time formalism*, the equations read

$$\begin{aligned} d\mu_i/dt &= \alpha(\xi - \mu_i) \quad \text{for } i \in N_c \; , \\ d\mu_i/dt &= 0 \qquad\quad\;\; \text{otherwise} \; . \end{aligned} \tag{3.9}$$

Proposition 3.5.1. *Let ξ be a stochastic variable. Starting with randomly chosen initial values for the μ_i, these numbers will gradually assume new values in a process specified by (3.7)–(3.9), such that when $t \to \infty$, the set of numbers $(\mu_1, \mu_2, \ldots, \mu_l)$ becomes ordered in an ascending or descending sequence. Once the set is ordered, it remains so for all t. Moreover, the point density function of the μ_i will finally approximate some monotonic function of the probability density function $p(\xi)$ of ξ.*

The discussion shall be carried out in two parts: formation of ordered sequences of the μ_i, and their convergence to certain "fixed points", respectively.

Ordering of Weights.

Proposition 3.5.2. *In the process defined by (3.7)–(3.9), the μ_i become ordered with probability one in an ascending or descending order when $t \to \infty$.*

One might like to have a rigorous proof for that ordering occurs almost surely (i.e., with probability one). Following the argumentation presented by *Grenander* for a related problem [3.9], the proof of ordering can be delineated as indicated below. Let $\xi = \xi(t) \in \Re$ be a random (scalar) input that has the probability density $p(\xi)$ over a finite support, with $\xi(t_1)$ and $\xi(t_2)$ independent for any $t_1 \neq t_2$.

The proof follows from general properties of Markov processes, especially that of the *absorbing state* for which the transition probability into itself is unity. It can be shown [3.10] that if such a state, starting from arbitrary initial state, is reached by *some* sequence of inputs that has a *positive* probability, then allowing a random sequence of inputs, the absorbing state is reached almost surely (i.e., with probability one), when $t \to \infty$.

The absorbing state is now identified with any of the ordered sequences of the μ_i. (Notice that there are two different ordered sequences, which together constitute the absorbing state.) On the line of real numbers, select an interval such that ξ has a positive probability on it. By repeatedly choosing values of ξ from this interval, it is possible to bring all μ_i within it in a finite time. After that it is possible to repeatedly choose values of ξ such that if, e.g., μ_{i-2}, μ_{i-1}, and μ_i are initially disordered, then μ_{i-1} will be brought between μ_{i-2} and μ_i, while the relative order in the other subsequences is not changed. Notice that if unit i is selected, then units $i-1$ *and* $i+1$ will change; if there is disorder on both sides of i, we may consider that side which is ordered first, stopping the application of (3.9) at that point, and calling this an elementary sorting operation. For instance, if $\mu_{i-1} < \mu_{i-2} < \mu_i$, then selection of ξ from the vicinity of μ_i will bring μ_{i-1} between μ_{i-2} and μ_i (notice that μ_{i-2} is not changed). The sorting can be continued systematically along similar lines. An overall order will then result in a finite number of steps. Since the above ξ values are realized with positive probability, the proof of Proposition 3.5.2 is concluded.

Cottrell and *Fort* [3.11] have presented an exhaustive and mathematically stringent proof of the above ordering process in the one-dimensional case, but since it is very lengthy (forty pages), it will be omitted here. The reader might instead study the shorter constructive proof presented in Sect. 3.5.2 for an almost equivalent system.

Corollary 3.5.1. *If all the values $\mu_1, \mu_2, \ldots, \mu_l$ are ordered, they cannot become disordered in further updating.*

The proof follows directly from the observation that if all partial sequences are ordered, then process (3.9) cannot change the relative order of any pair $(\mu_i, \mu_j), i \neq j$.

Convergence Phase. After the μ_i have become ordered, their final convergence to the asymptotic values is of particular interest since the latter represent the image of the input distribution $p(\xi)$.

In this subsection it is assumed henceforth that the $\mu_i, i = 1, 2, \ldots, l$ are already ordered and, on account of Corollary 3.5.1, remain such in further

updating processes. The aim is to calculate the asymptotic values of the μ_i. To be quite strict, asymptotic values are obtained in the sense of mean squares or almost sure convergence only if the "learning rate factor" $\alpha = \alpha(t)$ in (3.9) decreases to zero; the sequence $\{\alpha(t)|t = 0, 1, \ldots\}$ must obviously satisfy certain conditions similar to those imposed on the Robbins-Monro stochastic approximation process, cf. Sect. 1.3.3.

The convergence properties of the μ_i are discussed in this section in a less restricted sense, namely, only the dynamic behavior of the *expectation values* $\mathrm{E}\{\mu_i\}$ is analyzed. These numbers will be shown to converge to unique limits. The variances of the μ_i can then be made arbitrarily small by a suitable choice of $\alpha(t), t \to \infty$.

It may be useful to refer to Fig. 3.19 that represents the values μ_i on a line of real numbers. As stated above, the μ_i shall already be in order; we may restrict ourselves to the case of increasing values. It is also assumed that $[\mu_1, \mu_l]$ is a proper subset of $[a, b]$, the support of $p(\xi)$, which is obviously due if ordering has occurred through a process described above.

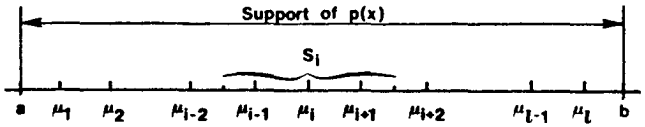


Fig. 3.19. Reference values after ordering

Since ordering of the μ_i was postulated, and because a selected node can only affect its immediate neighbours, it is obvious from (3.7)–(3.9) that any particular value μ_i can be affected only if ξ hits an interval S_i defined in the following way: assuming $l \geq 5$, we have

$$\begin{array}{lll} \text{for } 3 \leq i \leq l-2 : & S_i = [\frac{1}{2}(\mu_{i-2} + \mu_{i-1}), \frac{1}{2}(\mu_{i+1} + \mu_{i+2})] , \\ \text{for } i = 1 : & S_i = [a, \frac{1}{2}(\mu_2 + \mu_3)] , \\ \text{for } i = 2 : & S_i = [a, \frac{1}{2}(\mu_3 + \mu_4)] , \\ \text{for } i = l-1 : & S_i = [\frac{1}{2}(\mu_{l-3} + \mu_{l-2}), b] , \\ \text{for } i = l : & S_i = [\frac{1}{2}(\mu_{l-2} + \mu_{l-1}), b] . \end{array} \tag{3.10}$$

The expectation values of the $d\mu_i/dt \stackrel{\text{def}}{=} \dot{\mu}_i$, conditional on the $\mu_1, \ldots, \mu_l$, according to (3.9) read

$$\langle \dot{\mu}_i \rangle \stackrel{\text{def}}{=} E\{\dot{\mu}_i\} = \alpha(E\{\xi|\xi \in S_i\} - \mu_i)P(\xi \in S_i) , \tag{3.11}$$

where $P(\xi \in S_i)$ is the probability for ξ falling into the interval S_i. Now $E\{\xi|\xi \in S_i\}$ is the center of gravity of S_i, see (3.10), which is a function of the μ_i when $p(\xi)$ has been defined. In order to solve the problem in simplified closed form, it is assumed that $p(\xi) \equiv$ const over the support $[a, b]$ and zero outside it, whereby one first obtains:

for $3 \leq i \leq l-2$:

$$\begin{aligned} \langle \dot{\mu}_i \rangle &= \frac{\alpha}{4}(\mu_{i-2} + \mu_{i-1} + \mu_{i+1} + \mu_{i+2} - 4\mu_i)P(\xi \in S_i) , \\ \langle \dot{\mu}_1 \rangle &= \frac{\alpha}{4}(2a + \mu_2 + \mu_3 - 4\mu_1)P(\xi \in S_1) , \\ \langle \dot{\mu}_2 \rangle &= \frac{\alpha}{4}(2a + \mu_3 + \mu_4 - 4\mu_2)P(\xi \in S_2) , \\ \langle \dot{\mu}_{l-1} \rangle &= \frac{\alpha}{4}(\mu_{l-3} + \mu_{l-2} + 2b - 4\mu_{l-1})P(\xi \in S_{l-1}) , \\ \langle \dot{\mu}_l \rangle &= \frac{\alpha}{4}(\mu_{l-2} + \mu_{l-1} + 2b - 4\mu_l)P(\xi \in S_l) . \end{aligned} \tag{3.12}$$

Starting with arbitrary initial conditions $\mu_i(0)$, the most probable, "averaged" trajectories $\mu_i(t)$ are obtained as solutions of an equivalent differential equation corresponding to (3.12), namely,

$$dz/dt = P(z)(Fz + h) , \quad \text{where} \tag{3.13}$$

$$z = [\mu_1, \mu_2, \ldots, \mu_l]^{\mathrm{T}} ,$$

$$F = \frac{\alpha}{4}\begin{bmatrix} -4 & 1 & 1 & 0 & 0 & 0 & 0 & \ldots & & & & & \\ 0 & -4 & 1 & 1 & 0 & 0 & 0 & & & & & & \\ 1 & 1 & -4 & 1 & 1 & 0 & 0 & & & & & & \\ 0 & 1 & 1 & -4 & 1 & 1 & 0 & & & & & & \\ \vdots & & & & & & & & & & & & \\ & & & & & & & & & & & & \vdots \\ & & & & & & & & 0 & 1 & 1 & -4 & 1 & 1 & 0 \\ & & & & & & & & 0 & 0 & 1 & 1 & -4 & 1 & 1 \\ & & & & & & & & 0 & 0 & 0 & 1 & 1 & -4 & 0 \\ & & & & & & & \ldots & 0 & 0 & 0 & 0 & 1 & 1 & -4 \end{bmatrix}$$

$$h = \frac{\alpha}{2}[a, a, 0, 0, \ldots, 0, b, b]^{\mathrm{T}} , \tag{3.14}$$

and $P(z)$ is now a diagonal matrix with the $P(\xi \in S_i)$ its diagonal elements. The averaging, producing (3.13) from (3.12), could be made rigorous along the lines given by *Geman* [2.54]. Equation (3.13) is a first-order differential equation with constant coefficients. It has a fixed-point solution, a particular solution with $dz/dt = 0$, which is (taking into account that $P(z)$ is diagonal and positive definite)

$$z_0 = -F^{-1}h , \tag{3.15}$$

provided that F^{-1} exists; this has been shown strictly by the author [3.12] but the proof is too cumbersome and lengthy to be reproduced here. The general solution of (3.13) is difficult to obtain. Some recent studies, e.g. [3.11] indicate that convergence to the fixed point is generally true, and thus we restrict our considerations to showing the asymptotic solution only.

The asymptotic values of the μ_i, for uniform $p(\xi)$ over the support $[0,1]$ have been calculated for a few lengths l of the array and presented in Table 3.2 as well as in Fig. 3.20.

Table 3.2. Asymptotic values for the μ_i, with $a = 0$ and $b = 1$

Length of array (l)	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8	μ_9	μ_{10}
5	0.2	0.3	0.5	0.7	0.8	–	–	–	–	–
6	0.17	0.25	0.43	0.56	0.75	0.83	–	–	–	–
7	0.15	0.22	0.37	0.5	0.63	0.78	0.85	–	–	–
8	0.13	0.19	0.33	0.44	0.56	0.67	0.81	0.87	–	–
9	0.12	0.17	0.29	0.39	0.5	0.61	0.7	0.83	0.88	–
10	0.11	0.16	0.27	0.36	0.45	0.55	0.64	0.73	0.84	0.89

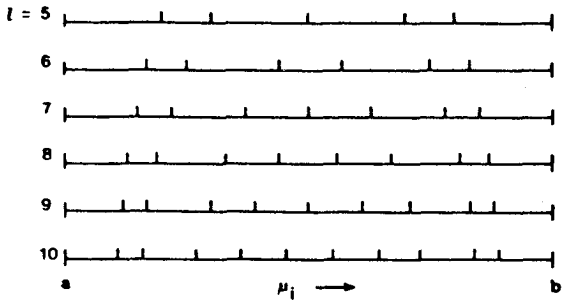


Fig. 3.20. Asymptotic values for the μ_i for different lengths of the array, shown graphically

It can be concluded that

- the outermost values μ_1 and μ_l are shifted inwards by an amount which is approximately $1/l$ whereas for uniform distribution this amount should be about $1/2l$; this is the "boundary effect" that occurs in the illustrations of Sect. 3.4, and vanishes with increasing l
- the values μ_3 through μ_{l-2} seem to be distributed almost evenly.

3.5.2 Constructive Proof of Ordering of Another One-Dimensional SOM

Since the ordering proof of the original SOM algorithm has turned out problematic although possible, we now carry out a simple proof for a somewhat modified but still logically similar system model. This proof appeared in

[3.13]. One of the main differences is redefinition of the "winner," because in the case of a tie, all "winners" are taken into account for learning in this modified model. The learning rule is then applied to the neighborhoods of all "winners," and the neighborhood set is modified in such a way that the "winner" itself is excluded from it. These modifications allow a *rigorous constructive proof of ordering* in the one-dimensional case.

The Problem. Assume an open-ended linear array of units named *nodes*. Let the nodes be indexed by $1, 2, \ldots, l$.

With every node, a real number $\mu_i = \mu_i(t) \in \Re$, which is a function of time, is associated. Let $\xi = \xi(t) \in \Re$ be a random input variable with a stationary probability density function $p(\xi)$ over a support $[a, b]$.

A self-organizing process is defined as follows: the degree of similarity of ξ with the numbers μ_i is at every instant t evaluated in terms of the distances $|\xi - \mu_i|$ that define one or several "winner" nodes c according to

$$|\xi - \mu_c| = \min_i \{|\xi - \mu_i|\} . \tag{3.16}$$

The numbers μ_i are continuously *updated* according to

$$\begin{aligned} \frac{d\mu_i}{dt} &= \alpha(\xi - \mu_i) \quad \text{for } i \in N_c , \\ \frac{d\mu_i}{dt} &= 0 \quad \text{otherwise} , \end{aligned} \tag{3.17}$$

where α is a "gain coefficient" (> 0), and N_c is a set of indices defined in the following way:

$$\begin{aligned} N_1 &= \{2\} , \\ N_i &= \{i-1, i+1\} \text{ for } 2 \le i \le l-1 , \\ N_l &= \{l-1\} . \end{aligned} \tag{3.18}$$

In case c is not unique, the union of all sets N_c must be used in stead of N_c in (3.17), excluding all the selected nodes themselves. The non-uniqueness of c is commented in the proof of Theorem 3.5.1.

The following discussion aims at showing that with time, the set of numbers $(\mu_1, \mu_2, \ldots, \mu_l)$ becomes ordered almost surely if certain rather mild conditions are fulfilled.

The degree of ordering is conveniently expressed in terms of the *index of disorder* D,

$$D = \sum_{i=2}^{l} |\mu_i - \mu_{i-1}| - |\mu_l - \mu_1| , \tag{3.19}$$

which is ≥ 0.

Definition. *The numbers $\mu_1, \ldots, \mu_l$ are ordered if and only if $D = 0$, which is equivalent to either $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_l$ or $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_l$.*

Note that this definition also allows the case in which a subset of the μ_i, or in fact all of them, become equal. In the present process this is indeed possible unless the input attains all its values sufficiently often. In numerous computer simulations, however, under the assumptions of Theorem 3.5.1, the asymptotic values have invariably ordered in a strictly monotonic sequence.

Theorem 3.5.1. *Let $\xi = \xi(t)$ be a random process satisfying the following assumptions:*

(i) $\xi(t)$ is almost surely integrable on finite intervals;
(ii) the probability density $p(\xi)$ of $\xi(t)$ is independent of t and strictly positive on $[a, b]$ and zero elsewhere, and $\xi(t)$ attains all values on $[a, b]$ almost surely during all time intervals $[t, \infty)$;
(iii) the initial values for the μ_i are randomly chosen from an absolutely continuous distribution on $[a, b]$.

Then, in the process defined by (3.16), (3.17), and (3.18), the μ_i will become almost surely ordered asymptotically.

Proof. Under the assumptions on ξ, and starting from almost surely different initial values, two numbers $\mu_i(t)$ and $\mu_j(t)$, $i \neq j$, are almost surely different except in the following case: if c is the "winner," or one of the "winners," then μ_{c-1} or μ_{c+1} may become equal to μ_c at some instant t. Then the index, say $c-1$, is added to the set of "winner" indices. Since now $d\mu_m/dt = d\mu_{m-1}/dt = 0$,the values of μ_c and μ_{c-1} stay equal as long as c remains one of the "winners." The result is that a set of values with consequent indices, say $\mu_{c-r}, \mu_{c-r+1}, \ldots, \mu_{c+p}$ with $r \geq 0$, $p \geq 0$, may be equal over time intervals of positive length. Then and only then c is non-unique with a positive probability.

Convention 3.5.1. In view of the above, the considerations will be reduced to the case of a unique c with always $\mu_{c-1} \neq \mu_c$ and $\mu_{c+1} \neq \mu_c$. If c is not unique, the indices must be redefined so that $c-1$ stands for $c-r-1$ and $c+1$ stand for $c+p+1$, and μ_c stands for all $\mu_{c-r}, \ldots, \mu_{c+p}$. This convention is followed throughout the proof without further mention.

Step 1. It will be shown first that $D = D(t)$ is a monotonically decreasing function of time, for all t, if $\xi = \xi(t)$ satisfies assumption (i).

The process (3.17) affects at most three (according to Convention 3.5.1) successive values of μ_i at a time; therefore it will suffice to inspect only those terms in D which may change. Let these constitute the partial sum S dependent on c in the following way (notice that $|\mu_l - \mu_1|$ may change only for $c = 2$ or $l-1$):

$$
\begin{aligned}
&\text{for } 3 \leq c \leq l-2, \quad S = \sum_{i=c-1}^{c+2} |\mu_i - \mu_{i-1}| ; \\
&\text{for } c = 1, \quad S = |\mu_3 - \mu_2| + |\mu_2 - \mu_1| ; \\
&\text{for } c = 2, \quad S = |\mu_4 - \mu_3| + |\mu_3 - \mu_2| + |\mu_2 - \mu_1| - |\mu_l - \mu_1| ; \\
&\text{for } c = l-1, \\
&\qquad S = |\mu_l - \mu_{l-1}| + |\mu_{l-1} - \mu_{l-2}| + |\mu_{l-2} - \mu_{l-3}| - |\mu_l - \mu_1| ; \\
&\text{for } c = l, \quad S = |\mu_l - \mu_{l-1}| + |\mu_{l-1} - \mu_{l-2}| . \qquad (3.20)
\end{aligned}
$$

Since the cases $c = l-1$ and $c = l$ are obviously symmetric with respect to $c = 2$ and $c = 1$, respectively, they need not be discussed separately.

At any given instant t, and any c, the signs of the differences $\mu_i - \mu_{i-1}$ in S attain one of at most 16 combinations. For any particular sign combination, the expression of dS/dt is given below. Of all different cases, half are symmetric in the sense that they produce the same analytical expression; accordingly, only the cases listed below need to be discussed separately. Around c one has (denoting $\dot{\mu}_i = d\mu_i/dt$)

$$
\begin{aligned}
\dot{\mu}_{c-2} &= 0 , \\
\dot{\mu}_{c-1} &= \alpha(\xi - \mu_{c-1}) , \\
\dot{\mu}_c &= 0 , \\
\dot{\mu}_{c+1} &= \alpha(\xi - \mu_{c+1}) , \\
\dot{\mu}_{c+2} &= 0 . \qquad (3.21)
\end{aligned}
$$

These linear differential equations have almost surely unique continuous solutions due to assumption (i).

We shall now consider the cases $3 \leq c \leq l-2$, $c = 1$, and $c = 2$ separately.

A. $3 \leq c \leq l-2$; assume $\mu_{c-1} \geq \mu_{c-2}$.

Case	$\mu_c - \mu_{c-1}$	$\mu_{c+1} - \mu_c$	$\mu_{c+2} - \mu_{c+1}$
$a0$	> 0	> 0	≥ 0
$a1$	> 0	> 0	≤ 0
$a2$	> 0	< 0	≥ 0
$a3$	> 0	< 0	≤ 0
$a4$	< 0	> 0	≥ 0
$a5$	< 0	> 0	≤ 0
$a6$	< 0	< 0	≥ 0
$a7$	< 0	< 0	≤ 0

If the symbols $a0$ through $a7$ are used as subscripts for the labeling of S, and it is denoted $\dot{S} = dS/dt$, one obtains taking into account (3.21):

$$
\begin{aligned}
&S_{a0} = -\mu_{c-2} + \mu_{c+2}, && \dot{S}_{a0} = 0 \ ; \\
&S_{a1} = -\mu_{c-2} + 2\mu_{c+1} - \mu_{c+2}, && \dot{S}_{a1} = 2\alpha(\xi - \mu_{c+1}) < 0 \ ; \quad *) \\
&S_{a2} = -\mu_{c-2} + 2\mu_c - 2\mu_{c+1} + \mu_{c+2}, && \dot{S}_{a2} = 2\alpha(\mu_{c+1} - \xi) < 0 \ ; \\
&S_{a3} = -\mu_{c-2} + 2\mu_c - \mu_{c+2}, && \dot{S}_{a3} = 0 \ ; \\
&S_{a4} = -\mu_{c-2} + 2\mu_{c-1} - 2\mu_c + \mu_{c+2}, && \dot{S}_{a4} = 2\alpha(\xi - \mu_{c-1}) < 0 \ ; \\
&S_{a5} = -\mu_{c-2} + 2\mu_{c-1} - 2\mu_c + 2\mu_{c+1} - \mu_{c+2}, && \\
& && \dot{S}_{a5} = 2\alpha[(\xi - \mu_{c-1}) \\
& && \quad +(\xi - \mu_{c+1})] < 0 \ ; \\
&S_{a6} = -\mu_{c-2} + 2\mu_{c-1} - 2\mu_{c+1} + \mu_{c+2}, && \dot{S}_{a6} = 2\alpha(\mu_{c+1} - \mu_{c-1}) < 0 \ ; \\
&S_{a7} = -\mu_{c-2} + 2\mu_{c-1} - \mu_{c2}, && \dot{S}_{a7} = 2\alpha(\xi - \mu_{c-1}) < 0 \ .
\end{aligned}
$$

*) Notice that for c the "winner," $\frac{1}{2}(\mu_{c-1} + \mu_c) \leq \xi \leq \frac{1}{2}(\mu_c + \mu_{c+1})$

(3.22)

B. $c = 1$: The proof is similar for nodes 1 and l and will be carried out for 1 only. Now only four cases need be considered.

Case	$\mu_2 - \mu_1$	$\mu_3 - \mu_2$
$b0$	> 0	≥ 0
$b1$	> 0	≤ 0
$b2$	< 0	≥ 0
$b3$	< 0	≤ 0

$$
\begin{aligned}
&S_{b0} = -\mu_1 + \mu_3, && \dot{S}_{b0} = 0; \\
&S_{b1} = -\mu_1 + 2\mu_2 - \mu_3, && \dot{S}_{b1} = 2\alpha(\xi - \mu_2) < 0; \\
&S_{b2} = \mu_1 - 2\mu_2 + \mu_3, && \dot{S}_{b2} = 2\alpha(\mu_2 - \xi) < 0; \\
&S_{b3} = \mu_1 - \mu_3, && \dot{S}_{b3} = 0.
\end{aligned}
$$

(3.23)

C. $c = 2$: It will suffice to consider the case $\mu_l \geq \mu_1$ only, since the case $\mu_l \leq \mu_1$ is symmetric to the other one.

Case	$\mu_2 - \mu_1$	$\mu_3 - \mu_2$	$\mu_4 - \mu_3$
$c0$	> 0	> 0	≥ 0
$c1$	> 0	> 0	≤ 0
$c2$	> 0	< 0	≥ 0
$c3$	> 0	< 0	≤ 0
$c4$	< 0	> 0	≥ 0
$c5$	< 0	> 0	≤ 0
$c6$	< 0	< 0	≥ 0
$c7$	< 0	< 0	≤ 0

$$
\begin{aligned}
&S_{c0} = -\mu_l + \mu_4, && \dot{S}_{c0} = 0; \\
&S_{c1} = -\mu_l + 2\mu_3 - \mu_4, && \dot{S}_{c1} = 2\alpha(\xi - \mu_3) < 0; \\
&S_{c2} = -\mu_l + 2\mu_2 - 2\mu_3 + \mu_4, && \dot{S}_{c2} = 2\alpha(\mu_3 - \xi) < 0; \\
&S_{c3} = -\mu_l + 2\mu_2 - \mu_4, && \dot{S}_{c3} = 0; \\
&S_{c4} = -\mu_l + 2\mu_1 - 2\mu_2 + \mu_4, && \dot{S}_{c4} = 2\alpha(\xi - \mu_1) < 0; \\
&S_{c5} = -\mu_l + 2\mu_1 - 2\mu_2 + 2\mu_3 - \mu_4, && \dot{S}_{c5} = 2\alpha[(\xi - \mu_1) + (\xi - \mu_3)] < 0; \\
&S_{c6} = -\mu_l + 2\mu_1 - 2\mu_3 + \mu_4, && \dot{S}_{c6} = 2\alpha(\mu_3 - \mu_1) < 0; \\
&S_{c7} = -\mu_l + 2\mu_1 - \mu_4, && \dot{S}_{c7} = 2\alpha(\xi - \mu_1) < 0.
\end{aligned}
$$

(3.24)

Step 2. Since $D(t)$ is monotonically decreasing and nonnegative, it must tend to some limit $D^* = \lim_{t \to \infty} D(t)$. It is now shown that $D^* = 0$ with probability one if $\xi(t)$ satisfies the assumption (ii).

Since D^* is constant with respect to t, $dD^*/dt = 0$ for all t. Still, D^* satisfies the same differential equation as $D(t)$. Assume now that $D^* > 0$. Then there is disorder in the set $(\mu_1, \ldots, \mu_l)$; without loss of generality, assume that there is an index j, $2 \leq j \leq l-1$, such that $\mu_j > \mu_{j+1}$ and $\mu_j > \mu_{j-1}$. (The proof would be similar for the case in which $\mu_j < \mu_{j+1}$ and $\mu_j < \mu_{j-1}$. Note that here, too, it is possible that μ_j stands for a set of equal values with consequent indices.)

With probability one $\xi(t)$ will eventually attain a value such that $j = c$ is the "winner," due to assumption (ii). If $j = 2$ (or $j = l-1$), then we have either case $c2$ with $dD^*/dt < 0$ which would mean contradiction, or case $c3$ with $dD^*/dt = 0$.

The magnitude relations of the μ_i values corresponding to case $c3$ are: $\mu_2 > \mu_1$, $\mu_3 < \mu_2$, and $\mu_4 \leq \mu_3$. Due to assumption (ii), with probability one $\xi(t)$ will eventually also attain a value such that $c = 3$ is the "winner," and $\xi < \mu_3$. If the μ_i values are as above, with $c = 3$ they can be written as $\mu_{c-1} > \mu_{c-2}$, $\mu_c < \mu_{c-1}$, and $\mu_{c+1} \leq \mu_c$. Then they correspond to case $a6$ or $a7$, with the difference that equality is possible in $\mu_{c+1} \leq \mu_c$. However, both $\dot{S}_{a6}$ and $\dot{S}_{a7}$ in (3.22) are now negative. This implies $dD^*/dt < 0$ which then again leads to contradiction.

If the "winner" c is equal to j with $3 \leq j \leq l-2$, then either case $a2$ is due, and $dD^*/dt < 0$, which means contradiction, or case $a3$ is due whereby $dD^*/dt = 0$. The magnitude relations of the μ_i values corresponding to case $a3$ are: $\mu_j > \mu_{j-1}$, $\mu_{j+1} < \mu_j$, and $\mu_{j+2} \leq \mu_{j+1}$. Again, due to assumption (ii), with probability one $\xi(t)$ will eventually also attain a value such that $j + 1 = c$ is the "winner" and $\xi < \mu_{j+1}$. If the μ_i values are as above, with $c = j + 1$ they can be written as $\mu_{c-1} > \mu_{c-2}$, $\mu_c < \mu_{c-1}$, and $\mu_{c+1} \leq \mu_c$. Then they correspond to case $a6$ or $a7$, with the difference that equality is possible in $\mu_{c+1} \leq \mu_c$. However, again both $\dot{S}_{a6}$ and $\dot{S}_{a7}$ in (3.22) are negative, leading to the contradiction $dD^*/dt < 0$.

From the above, we can conclude that D^* must be equal to zero, or otherwise a contradiction is bound to occur. The asymptotic state must therefore be an ordered one.

3.6 The Batch Map

It will be useful to understand what the convergence limits m_i^* in the sequence defined by (3.3) actually represent.

Assuming that the convergence to some ordered state is true, the expectation values of $m_i(t+1)$ and $m_i(t)$ for $t \to \infty$ must be equal, even if $h_{ci}(t)$ were then selected nonzero. In other words, in the stationary state we must have

$$\forall i,\ \mathrm{E}\{h_{ci}(x - m_i^*)\} = 0\,. \tag{3.25}$$

In the simplest case $h_{ci}(t)$ was defined: $h_{ci} = 1$ if i belongs to some *topological neighborhood set* N_c of cell c in the cell array, whereas otherwise $h_{ci} = 0$. With this h_{ci} there follows

$$m_i^* = \frac{\int_{V_i} xp(x)dx}{\int_{V_i} p(x)dx}\,, \tag{3.26}$$

where V_i is the set of those x values in the integrands that are able to update vector m_i; in other words, the "winner" node c for each $x \in V_i$ must belong to the neighborhood set N_i of cell i.

Let us now exemplify (3.25) or (3.26) by Fig. 3.21. In this special case, the neighborhood of cell i consists of the cells $i-1, i$, and $i+1$, except at the ends of the array, where only one neighbor exists.

In the case described in Fig. 3.21 we have two-dimensional vectors as inputs, and the probability density function of $x \in \Re^2$ is uniform over the framed area (support of the x values) and zero outside it. The neighborhood set N_c has the simple form defined above. *In this case at least the equilibrium condition (3.25) or (3.26) means that each m_i^* must coincide with the centroid of the respective influence region.* At least in this case it may be intuitively clear that the equilibrium represents the case whereby *the Voronoi sets around the m_i contact each other in the same order as the "topological links" between the nodes in the neuron array are defined.* In general, with more complicated network architectures, a similar topological correspondence has been discussed in [3.14].

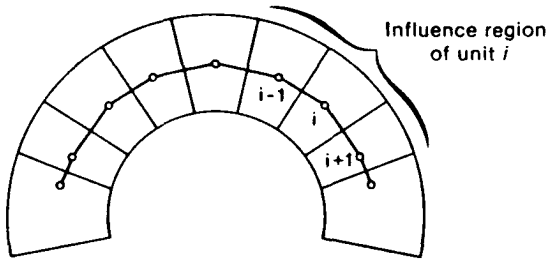


Fig. 3.21. Illustration for the explanation of the equilibrium state in self-organization and definition of "influence region"

The equilibrium condition (3.25) or (3.26) with a general probability density function $p(x)$ means that each m_i^* must coincide with the centroid of $p(x)$ over the respective influence region. This could then be taken for the definition of the ordered state. It may be intuitively clear that with general dimensionalities, such an equilibrium can only be valid with a particular configuration of the m_i.

Equation (3.26) is already in the form in which the so-called *iterative contractive mapping* used in the solving of nonlinear equations is directly applicable. Let z be an unknown vector that has to satisfy the equation $f(z) = 0$; then, since it is always possible to write the equation as $z = z + f(z) = g(z)$, the successive approximations of the root can be computed as a series $\{z_n\}$ where

$$z_{n+1} = g(z_n)\,. \tag{3.27}$$

We shall not discuss any convergence problems here. In the SOM context we have never encountered any, but if there would exist some, they could be overcome by the so-called Wegstein modification of (3.27):

$$z_{n+1} = (1-\lambda)g(z_n) + \lambda z_n \ \text{, where } 0 < \lambda \leq 1\,. \tag{3.28}$$

The iterative process in which a number of samples of x is first classified into the respective V_i regions, and the updating of the m_i^* is made iteratively as defined by (3.26), can be expressed as the following steps. The algorithm dubbed *"Batch Map"* [3.15, 16] resembles the Linde-Buzo-Gray algorithm discussed in Sect. 1.5, where all the training samples are assumed to be available when learning begins. The learning steps are defined as follows:

1. For the initial reference vectors, take, for instance, the first K training samples, where K is the number of reference vectors.
2. For each map unit i, collect a list of copies of all those training samples x whose nearest reference vector belongs to unit i.
3. Take for each new reference vector the mean over the union of the lists in N_i.
4. Repeat from 2 a few times.

It is easy to see that this algorithm describes the same process that was described in the general setting in Sect. 3.1. If now a general neighborhood function h_{ji} is used, and $\bar{x}_j$ is the mean of the $x(t)$ in Voronoi set V_j, then we shall weight it by the number n_j of samples V_j and the neighborhood function. Now we obtain

$$m_i^* = \frac{\sum_j n_j h_{ji} \bar{x}_j}{\sum_j n_j h_{ji}} , \tag{3.29}$$

where the sum over j is taken for all units of the SOM, or if h_{ji} is truncated, over the neighborhood set N_i in which it is defined. For the case in which no weighting in the neighborhood is used,

$$m_i^* = \frac{\sum_{j \in N_i} n_j \bar{x}_j}{\sum_{j \in N_i} n_j} . \tag{3.30}$$

A discussion of the convergence and ordering of the Batch Map type algorithm has been presented in [3.8].

This algorithm is particularly effective if the initial values of the reference vectors are already roughly ordered, even if they might not yet approximate the distribution of the samples. It should be noticed that the above algorithm contains no learning-rate parameter; therefore it has no convergence problems and yields stabler asymptotic values for the m_i than the original SOM.

The size of the neighborhood set N_i above can be similar to the size used in the basic SOM algorithms. "Shrinking" of N_i in this algorithm means that the neighborhood radius is decreased while steps 2 and 3 are repeated. At the last couple of iterations, N_i may contain the element i only, and the last steps of the algorithm are then equivalent with the K-means algorithm, which guarantees the most accurate approximation of the density function of the input samples. A few iterations of this algorithm will usually suffice.

Elimination of Border Effects for Low-Dimensional Signals. Inspection of Fig. 3.22 may further clarify the border effects in the one-dimensional SOM and help to understand how they could be eliminated.

If every cell of the SOM has two neighbors, except one at the ends, the "influence region" of cell i ($i > 2$ and $i < k - 1$) (or the range of x values that can affect cell i) is defined $V_i = [\frac{1}{2}(m_{i-2} + m_{i-1}), \frac{1}{2}(m_{i+1} + m_{i+2})]$. In the asymptotic equilibrium, according to (6.18), every m_i must coincide with the centroid of $p(x)$ over the respective V_i. The definition of the "influence

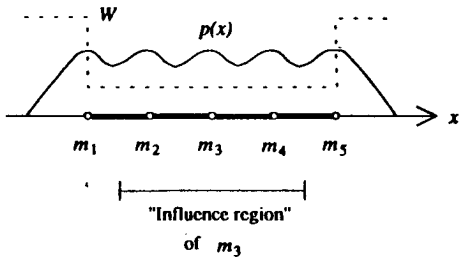


Fig. 3.22. One-dimensional SOM with five reference "vectors" m_i (scalars) that approximate the probability density function $p(x)$, and delineation of the weighting function W

regions" near the borders of the SOM is different, however, and therefore the m_i do not approximate $p(x)$ everywhere in the same way.

In computing the centroids, it is now possible to provide the x samples with *conditional weights* W that depend on index i and the relative magnitude of x and m_i. This weighting can be used both in the old stepwise SOM algorithm (with the given definition of the neighborhood set N_c), and with the Batch Map, too. In the former case, the weight should be applied to the learning-rate factor α, not to x. For to guarantee stability, one must then have $\alpha W < 1$, so this trick is not applicable during the first steps when α is still large. In the Batch Map algorithm, however, the x samples are always weighted directly, so no such restriction exists. Henceforth we assume that the Batch Map is used.

The following rules may first sound a bit complicated, but they are simple to program, and in practice they are also very effective and robust in eliminating the border effects to a large extent. Assume that the m_i values are already ordered.

Weighting Rule for the One-Dimensional SOM:

In updating, each x sample is provided with weight W. Normally $W = 1$, but $W > 1$ for the border (end) cells in the case that x is bigger than the biggest m_i or smaller than the smallest m_i, AND when updating of the border cell (but not of its neighbors) is due.

Consider the special case that $p(x)$ is *uniform* over some singly connected domain of x and zero outside it. It may then be easy to deduce on the basis of Fig. 3.22 and the above weighting rule that if we select for the special weight a value of $W = 9$, all the m_i will become equidistant in the asymptotic equilibrium; then they describe $p(x)$ in an unbiased way. Naturally, for other forms of $p(x)$, we should take other values for W. In many practical cases, however, the default value $W = 9$ compensates for the most part of the border effects in general.

It is possible to eliminate the border effects *totally*, if after a couple of Batch Map iterations the neighborhood set N_i is replaced by $\{i\}$, i.e., having a couple of simple K-means iterations at the end.

In a two-dimensional SOM, the weighting rules are slightly different. While we used, say, the value of $W = 9$ for the end cells in the one-dimensional array, in the updating of the two-dimensional array we must have a different weight W_1 for the corner cells, and another value W_2 for edge cells that are not in the corner. Inside the array, the weight is equal to unity.

Weighting Rules for the Two-Dimensional SOM:

The value W_1 is applied if both of the following two conditions are satisfied: A1. The value of x is in one of the four "outer corner sectors," i.e. outside the array and such that the m_i of some corner

cell is closest. A2. Updating of this selected m_i (but not of any other of its topological neighbors) is due.

The value W_2 is applied if both of the following conditions are satisfied: B1. The value of x lies outside the m_i array, but the closest m_i does not belong to any corner cell. B2. Updating of the selected edge cell or any of its topological neighbors, which must be one of the edge cells (eventually even a corner cell) is due.

If $p(x)$ in the two-dimensional input space were uniform over a square domain and zero outside it, it would be easy to deduce, in analogy with the one-dimensional case, that for an equidistant equilibrium distribution of the m_i values we must have $W_1 = 81$, $W_2 = 9$. Again, for other $p(x)$ the compensation is not complete with these weights. Then, as earlier, the Batch Map process may be run for a couple of iterations, followed by a couple of K-means iterations. Such a combination of methods is again both robust and unbiased and follows the two-dimensional input states very effectively.

3.7 Initialization of the SOM Algorithms

Random Initialization. The reason for using random initial values in the demonstrations of the SOM was that the SOM algorithms *can* be initialized using *arbitrary* values for the codebook vectors $m_i(0)$. In other words it has been demonstrated that initially unordered vectors will be ordered in the long run, in usual applications in a few hundred initial steps. This does *not* mean, however, that random initialization would be the *best* or *fastest* policy and should be used in practice.

Linear Initialization. As the $m_i(0)$ can be arbitrary, one might reason that *any ordered initial state* is profitable, even if these values do not lie along the main extensions of $p(x)$. A method we have used with success is to first determine the two eigenvectors of the autocorrelation matrix of x that have the largest eigenvalues, and then to let these eigenvectors span a two-dimensional linear subspace. A rectangular array (with rectangular or hexagonal regular lattice) is defined along this subspace, its centroid coinciding with that of the mean of the $x(t)$, and the main dimensions being the same as the two largest eigenvalues. The initial values of $m_i(0)$ are then identified with the array points. If one wants to obtain an approximately uniform lattice spacing in the SOM, the relative numbers of cells in the horizontal and vertical directions of the lattice, respectively, should be proportional to the two largest eigenvalues considered above.

Since the $m_i(0)$ are now already ordered and their point density roughly approximates $p(x)$, it will be possible to directly start the learning with the convergence phase, whereby one can use values for $\alpha(t)$ that from the beginning are significantly smaller than unity, and a neighborhood function

the width of which is close to its final value, to approach the equilibrium smoothly.

3.8 On the "Optimal" Learning-Rate Factor

The period during which a rough order in the SOM is obtained is usually relatively short, on the order of 1000 steps, whereas most of the computing time is spent for the final convergence phase, in order to achieve a sufficiently good statistical accuracy. It is not clear how the learning-rate factor should be optimized during the first phase, since the width of the neighborhood function is thereby changing, too, which complicates the situation.

On the other hand, in Sect. 3.7 we already pointed out that it is always possible to start the SOM algorithm with an already ordered state, e.g. with all the m_i lying in a regular array along a two-dimensional hyperplane. Also, since during the final convergence phase we usually keep the width of the neighborhood fixed, it seems possible to determine some kind of "optimal" law for the sequence $\{\alpha(t)\}$ during the convergence phase. Before deriving this law for the SOM we shall discuss a couple of simpler examples, from which the basic idea may be seen.

Recursive Means. Consider a set of vectorial samples $\{x(t)\}$, $x(t) \in \Re^n, t = 0, 1, 2, \ldots, T$. From the way in which the mean m of the $x(t)$ is computed recursively it may be clear that $m(t)$ is at all times correct with respect to samples taken into account up to step $t+1$:

$$\begin{aligned} m &= \frac{1}{T}\sum_{t=1}^{T} x(t) = m(T) \; ; \\ m(t+1) &= \frac{t}{t+1} m(t) + \frac{1}{t+1} x(t+1) \\ &= m(t) + \frac{1}{t+1}[x(t+1) - m(t)] \; . \end{aligned} \tag{3.31}$$

This law will now be reflected in the VQ processes, too.

"Optimized" Learning-Rate Factors for Recursive VQ. The steepest-descent recursion for the classical vector quantization was derived in Sect. 1.5 and reads

$$m_i(t+1) = m_i(t) + \alpha(t)\delta_{ci}[x(t) - m_i(t)] \; , \tag{3.32}$$

where δ_{ci} is the Kronecker delta. Let us rewrite (3.32) as

$$m_i(t+1) = [(1 - \alpha(t)\delta_{ci}]m_i(t) + \alpha(t)\delta_{ci}x(t) \; . \tag{3.33}$$

(Notice a shift in the index of $x(t)$ with respect to (3.31). It is a matter of convention how we label the $x(t)$.) If we consider $m_i(t+1)$ as a "memory" of all the values $x(t'), t' = 0, 1, \ldots, t$, we see that if m_i is the "winner", then

a memory trace of $x(t)$, scaled down by $\alpha(t)$, is superimposed on it. If m_i was "winner" at step t, it has a memory trace of $x(t-1)$, scaled down by $[1-\alpha(t)]\alpha(t-1)$ through the first term of (3.33); if $m_i(t)$ was not winner, no memory trace from $x(t-1)$ was obtained. It is now easy to see what happens to earlier memory traces from $x(t')$; every time when m_i is a "winner", all $x(t')$ are scaled down by the factor $[1-\alpha(t)]$, which we assume < 1. If $\alpha(t)$ were constant in time, we see that in the long run the effects of the $x(t')$ on $m(t+1)$ will be *forgotten*, i.e., $m(t+1)$ only depends on relatively few last samples, the "forgetting time" depending on the value of α. If, on the other hand, the earlier values of $\alpha(t')$ are selected as bigger, we might be able to *compensate for* the "forgetting" and have a roughly equal influence of the $x(t')$ on $m(t+1)$.

Comment. Since the Voronoi tessellation in VQ is changed at every step, the supports from which the $x(t)$ values are drawn for a particular m_i are also changing, so this case is not as clear as for the simple recursive means; therefore the $\alpha(t)$ values we derive will only approximately be "optimal."

Consider now the two *closest* instants of time t_1 and t_2, $t_2 > t_1$, for which the same m_i, denoted by m_c, was the "winner." If we stipulate that the memory traces of $x(t_1)$ and $x(t_2)$ shall be equal in different neurons at all times, obviously we have to select an *individual learning-rate factor* $\alpha_i(t)$ for each m_i. For steps t_1, and t_2 we must then have

$$[1-\alpha_c(t_2)]\alpha_c(t_1) = \alpha_c(t_2)\,. \tag{3.34}$$

Solving for $\alpha_c(t_2)$ from (3.34) we get

$$\alpha_c(t_2) = \frac{\alpha_c(t_1)}{1+\alpha_c(t_1)}\,. \tag{3.35}$$

Let us emphasize that $\alpha_i(t)$ is now *only* changed when m_i was the "winner"; otherwise $\alpha_i(t)$ remains *unchanged.* Since $\alpha_c(t_1)$ retains the same value up to step t_2-1, we can easily see that the following recursion of α_i can be applied at the same time when the "winner" m_c is updated:

$$\alpha_c(t+1) = \frac{\alpha_c(t)}{1+\alpha_c(t)}\,. \tag{3.36}$$

It must once again be emphasized that (3.36), as well as similar expressions to be derived for SOM and LVQ, do not *guarantee* absolutely optimal convergence, because the Voronoi tessellations in these algorithms are changing. Conversely, however, one may safely state that if (3.36) is *not* taken into account, the convergence is *less optimal* on the average.

"Optimized" Learning-Rate Factor for the SOM. It is now straightforward to generalize the above philosophy for the SOM. If we define an individual $\alpha_i(t)$ for each m_i and write

$$m_i(t+1) = m_i(t) + \alpha_i(t)h_{ci}[x(t) - m_i(t)]\,, \tag{3.37}$$

where h_{ci} (in the convergence phase) is time-invariant, *we may update* $\alpha_i(t)$ *whenever a correction to the* m_i *values is made, relative to that correction:*

$$\alpha_i(t+1) = \frac{\alpha_i(t)}{1+h_{ci}\alpha_i(t)}\,. \tag{3.38}$$

The law for $\alpha_i(t)$ derived above was based on a theoretical speculation; it may not work in the best way in practice, due to the very different values for $\alpha_i(t)$ obtained in the long run for different i. Let us recall that the point density of the codebook vectors in the original SOM algorithm was some monotonic function of $p(x)$. This density is influenced by differences in learning-rate factors.

In vector quantization, especially in Learning Vector Quantization discussed in Chap. 6 this idea works reasonably well, though: no harmful deformations of point densities of the m_i have thereby been observed.

Semi-Empirical Learning-Rate Factor. For the above mentioned reasons it also seems justified to keep $\alpha(t)$ in the SOM identical for all the neurons and to look for an *average optimal rate. Mulier* and *Cherkassky* [3.17] have ended up with an expression of the form

$$\alpha(t) = \frac{A}{t+B}\,, \tag{3.39}$$

where A and B are suitably chosen constants. At least this form satisfies the stochastic-approximation conditions. The main justification for (3.39) is that earlier and later samples will be taken into account with approximately similar average weights.

3.9 Effect of the Form of the Neighborhood Function

As long as one starts the self-organizing process with a wide neighborhood function, i.e., with a wide radius of the neighborhood set $N_c(0)$, or a wide standard deviation of $h_{ci}(0)$ such a value being of the same order of magnitude as half of the largest dimension of the array, there are usually no risks for ending up in "metastable" configurations of the map (for which the average expected distortion measure or average expected quantization error would end up in a local minimum instead of the global one.) However, with time-invariant neighborhood function the situation may be quite different, especially if the neighborhood function is narrow.

Erwin et al. [3.18] have analyzed the "metastable states" in a one-dimensional array. They first defined the neighborhood function being *convex* on a certain interval $I \equiv \{0,1,2,\ldots,N\}$, if the conditions $|s-q| > |s-r|$ and $|s-q| > |r-q|$ imply that $[h(s,s)+h(s,q)] < [h(s,r)+h(r,q)]$ for all $s,r,q \in I$. Otherwise the neighborhood function was said to be concave.

The main results they obtained were that if the neighborhood function is convex, there exist no stable states other than the ordered ones. If the

neighborhood function is concave, there exist metastable states that may slow down the ordering process by orders of magnitude. Therefore, if in the beginning of the process the neighborhood function is convex, like the middle part of the Gaussian $h_{ci}(t)$ at large standard deviation is, the ordering can be achieved almost surely; and after ordering the neighborhood function can be shrunk to achieve an improved approximation of $p(x)$.

The ordering conditions in general are most severe if the input signal space has the same dimensionality as the array; in practice, however, the dimensionality of the input signal space is usually much higher, whereby "ordering" takes place easier.

3.10 Does the SOM Algorithm Ensue from a Distortion Measure?

Since the SOM belongs to the category of vector quantization (VQ) methods, one might assume that the starting point in its optimization must be some kind of *quantization error in the vector space.* Assume that $x \in \Re^n$ is the input vector and the $m_i \in \Re^n, i \in \{\text{indices of neurons}\}$ are the reference vectors; let $d(x, m_i)$ define a generalized distance function of x and m_i. The quantization error is then defined as

$$d(x, m_c) = \min_i \{d(x, m_i)\} , \tag{3.40}$$

where c is the index of the "closest" reference vector to x in the space of input signals.

An even more central function in the SOM, however, is the *neighborhood function* $h_{ci} = h_{ci}(t)$ that describes the interaction of reference vectors m_i and m_c during adaptation and is often a function of time t. To this end one might try to define the *distortion measure* in the following way. Denote the set of indices of all lattice units by L; the distortion measure e is defined as

$$e = \sum_{i \in L} h_{ci} d(x, m_i) , \tag{3.41}$$

that is, as a sum of distance functions weighted by h_{ci}, whereby c is the index of the closest codebook vector to x. If we now form the *average expected distortion measure*

$$E = \int ep(x)dx = \int \sum_{i \in L} h_{ci} d(x, m_i) p(x) dx , \tag{3.42}$$

one way of defining the SOM is to define it as the set of the m_i that globally minimizes E.

Exact optimization of (3.42), however, is still an unsolved theoretical problem, and extremely heavy numerically. The best approximative solution that has been found so far is based on the Robbins-Monro stochastic approximation (Sect. 1.3.3). Following this idea we consider stochastic samples of the

distortion measure: if $\{x(t), t = 1, 2, \ldots\}$ is a sequence of input samples and $\{m_i(t), t = 1, 2, \ldots\}$ the recursively defined sequence of codebook vector m_i, then

$$e(t) = \sum_{i \in L} h_{ci}(t) d[x(t), m_i(t)] \tag{3.43}$$

is a stochastic variable, and the sequence defined by

$$m_i(t+1) = m_i(t) - \lambda \cdot \nabla_{m_i(t)} e(t) \tag{3.44}$$

is used to find an approximation to the optimum, as asymptotic values of the m_i. This would then define the SOM algorithm for a generalized distance function $d(x, m_i)$. It must be emphasized, however, that although the convergence properties of stochastic approximation have been thoroughly known since 1951, the asymptotic values of the m_i obtained from (3.44) only minimize E approximately. We shall see in Sect. 3.12.2 that at least the point density of the reference vectors obtained from (3.44) are differen from those derived directly on the basis of (3.42). *Then, on the other hand, (3.43) and (3.44) may be taken as still another definition of a class of SOM algorithms.*

Comment 3.10.1. For those readers who are not familiar with the Robbins-Monro stochastic approximation it may be necessary to point out that E need not be a *potential function*, and the convergence limit does not necessarily represent the *exact* minimum of E, only an approximation of it. Nonetheless the convergence properties of this class of processes are very robust and they have been studied thoroughly long ago in the theory of this method.

Comment 3.10.2. Above, the neighborhood function h_{ci} was assumed time-invariant, and it will be assumed such also in Sect. 3.11 where an optimization attempt will be made. It seems to be an important property of the SOM algorithms, however, that the kernels h_{ci} are time-variable.

Example 3.10.1. Let $d(x, m_i) = ||x - m_i||^2$. Then we obtain the original SOM algorithm:

$$m(t+1) = m_i(t) + \alpha(t) h_{ci}(t)[x(t) - m_i(t)]. \tag{3.45}$$

Example 3.10.2. In the so-called *city-block metric*, or *Minkowski metric of power one*,

$$d(x, m_i) = \sum_j |\xi_j - \mu_{ij}|. \tag{3.46}$$

Now we have to write the SOM algorithm in component form:

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \alpha(t) h_{ci}(t) \text{sgn}[\xi_j(t) - \mu_{ij}(t)], \tag{3.47}$$

where sgn[·] is the signum function of its argument.

3.11 An Attempt to Optimize the SOM

It has to be emphasized that we do not know any theoretical reason for which the recursive algorithm of the basic SOM *should* ensue from any objective function E that describes, e.g., the average expected distortion measure. The following facts only *happen* to hold true: 1. The usual stochastic-approximation optimization of E leads to the basic SOM algorithm, but this optimization method is only approximative, as will be seen below. 2. The (heuristically established) basic SOM algorithm describes a nonparametric regression that often reflects important and interesting topological relationships between clusters of the primary data.

In order to demonstrate what the "energy function" formalism pursued in this context actually may mean, we shall proceed one step deeper in the analysis of the average expected distortion measure [1.75].

The effect of the index c of the "winner," which is a discontinuous function of x and all the m_i, can be seen more clearly if the integral E in (3.42), with $d(x, m_i) = ||x - m_i||^2$ is expressed as a sum of partial integrals taken over those domains X_i where x is closest to the respective m_i (partitions in the Voronoi tessellation, cf. Fig. 3.23):

$$E = \sum_i \int_{x \in X_i} \sum_k h_{ik} ||x - m_k||^2 p(x) dx \; . \tag{3.48}$$

When forming the true (global) gradient of E with respect to an arbitrary m_j, one has to take into account two different kinds of terms: first, when the integrand is differentiated but the integration limits are held constant, and second, when the integration limits are differentiated (due to changing m_j) but the integrand is held constant. Let us call these terms G and H, respectively:

$$\nabla_{m_j} E = G + H \; , \tag{3.49}$$

whereby it is readily obtained

$$\begin{aligned} G &= -2 \cdot \sum_i \int_{x \in X_i} h_{ij}(x - m_j) p(x) dx \\ &= -2 \cdot \int_{x\,\mathrm{space}} h_{cj}(x - m_j) p(x) dx \; . \end{aligned} \tag{3.50}$$

In the classical VQ (Sect. 1.5), $H = 0$, due to the facts that $h_{ck} = \delta_{ck}$, and there, across the border of the Voronoi tessellation between X_i and X_j, the terms $||x - m_i||^2$ and $||x - m_j||^2$ were equal. With general h_{ck}, computation of H is a very cumbersome task, because the integrands are quite different when crossing the borders in the Voronoi tessellation.

To start with, I am making use of the fact that only those borders of the tessellation that delimit X_j are shifting in differentiation.

Consider Fig. 3.23 that delineates the partition X_j, and the shift of its borders due to dm_j. The first extra contribution to $\nabla_{m_j} E$, with the $||x - m_k||^2$

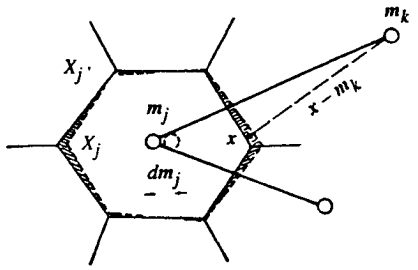


Fig. 3.23. Illustration of the border effect in differentiation with respect to m_j

evaluated over the topological neighborhood of m_j, is now obtained as the integral of the integrand of E taken over the shaded differential hypervolume. Because all the borders are segments of midplanes (hyperplanes) between the neighboring m_i, at least some kind of "average shift" seems to be estimable. As a matter of fact, if there were so many neighbors around m_j that X_j could be approximated by a hypersphere, and m_j were varied by the amount dm_j, then X_j would preserve its shape and be shifted by the amount $(1/2)dm_j$. In a general case with arbitrary dimensionality of the x space and constellation of the m_i, the shape of X_j would be changed; but my simplifying and averaging approximation is that this change of shape is not considered, in the first approximation at least, and X_j is only shifted by $(1/2)dm_j$. Another approximation that is needed to simplify the order-of-magnitude discussion is to assume $p(x)$ constant over the partition X_j, and this is justified if many m_i are used to approximate $p(x)$. If both of the above approximations are made, then the first contribution to H obtained from the variation of X_j, denoted H_1, is approximately the difference of two integrals: the integral over the displaced X_j, and the integral over the undisplaced X_j, respectively. This is equal to the integral of $\sum h_{jk} ||x - (m_k - (1/2)dm_j)||^2$ *times* $p(x)$ *minus* the integral of $\sum h_{jk} ||x - m_k||^2$ *times* $p(x)$. But one may think that this again is equivalent to the case where each of the m_k is differentiated by the same amount $-(1/2)dm_j$. Therefore this difference can also be expressed as the sum of the gradients of the integral of $\sum h_{jk} ||x - m_k||^2$ *times* $p(x)$ taken with respect to each of the m_k, and multiplied by $-(1/2)dm_j$:

$$\begin{aligned} H_1 &= -1/2 \cdot (-2) \cdot \int_{x \in X_j} \sum_{k \neq j} h_{jk}(x - m_k) p(x) dx \\ &= \int_{x \in X_c} \sum_{k \neq c} h_{ck}(x - m_k) p(x) dx \; . \end{aligned} \tag{3.51}$$

The case $k = c$ can be excluded above because this term corresponds to the basic Vector Quantization, and its contribution can be shown to be zero [1.75].

We must not forget that E is a sum over all i, and so the second extra contribution to H, named H_2, is due to the integral of the integrand of E

over all the partial differential hypervolumes (shaded segments in Fig. 3.23) bordering to $X_{j'}$ where j' is the index of any of the neighboring partitions of X_j. To compute H_2 seems to be even more cumbersome than to get the approximation of H_1. There are good reasons, however, to assume that on the average, $||H_2|| < ||H_1||$, since each of the integration domains in H_2 is only one segment of the differential of X_j, and the $x - m_k$ are different in each of these subdomains. In an order-of-magnitude analysis of the extra corrections at least, it is perhaps more interesting to concentrate on the major contribution H_1 quantitatively, whereby it is also possible to give an illustrative interpretation for it.

When trying to derive the results obtained into a recursive stepwise descent in the "E landscape", we may notice that in the expression of G, x runs over the whole x space, and $\nabla_{m_j} E$ has contributions from all those X_c for which m_j is a topological neighbor in the network. Conversely, in H_1, the integral is only over X_j, whereas the integrand contains terms that depend on the topological neighbors m_k of m_j. In analogy with Eqs. (1.166), (1.167), and (1.168) in Sect. 1.5 we can then write (neglecting terms due to H_2):

$$m_c(t+1) = m_c(t) + \alpha(t)\{h_{cc}[x(t) - m_c(t)] - 1/2\sum_{k \neq c} h_{ck}[x(t) - m_k(t)]\} ,$$
$$m_i(t+1) = m_i(t) + \alpha(t) h_{ci}[x(t) - m_i(t)] , \quad \text{for } i \neq c . \tag{3.52}$$

Notice in particular that the term $-1/2\sum_k h_{ck}[x(t) - m_k(t)]$ was derived under the assumption that $p(x)$ is constant over each partition in the Voronoi tessellation. For nodes at the edges of the tessellation, the corresponding partitions X_i extend to infinity, and this approximation is then no longer quite valid; so boundary effects, slightly different from those encountered with the basic SOM, are also here discernible.

Before reporting numerical experiments, we can give an interpretation to the extra terms due to the H_1 integral. Consider Fig. 3.24 that illustrates the old and the new algorithm. Assume that at some learning step, one of the m_i is offset from the rest of the parameter vector values. When x is closest to it, in the old algorithm this $m_i = m_c$ and its topological neighbors m_k would be shifted toward x. In the new algorithm, only the $m_k \neq m_c$ are shifted toward x, whereas m_c is "relaxed" in the opposite direction toward the center of the m_k neighborhood. This extra relaxation seems to be useful for self-organization, as demonstrated below.

A numerical experiment was performed to study the convergence properties of (3.52). A square array of nodes, with two-dimensional x and m_i, was used in the experiment. Both the old algorithm and (3.52) were run with identical initial state, values of $\alpha(t)$, and random number sequence. The $m_i(0)$ were selected as independent random values and $\alpha(t)$ decreased linearly from 0.5 to 0. The neighborhood kernel h_{ck} was constant for all nodes within one

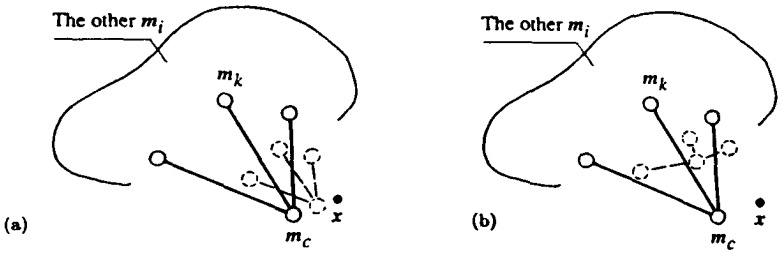


Fig. 3.24. Differences in correction: **(a)** The basic SOM, **(b)** Eq. (3.52)

lattice spacing from c in the horizontal and vertical direction, and $h_{ck} = 0$ otherwise. (It must be recalled that in practical applications, to make convergence faster and safer, the kernel is usually made to "shrink" over time.) In Fig. 3.25, the values of m_i for a square lattice as in Fig. 3.5 after 8000 steps of the process are displayed. The $p(x)$ had a constant value in the framed square and zero outside it. From many similar experiments (of which one is shown in Fig. 3.25), two conclusions transpire: 1. The new algorithm orders the m_i somewhat faster and safer; after 8000 steps (with the narrow constant kernel h_{ck}), none of the old-algorithm states was completely ordered, whereas about half of the new-algorithm states were. 2. The boundary effects in the new algorithm are stronger.

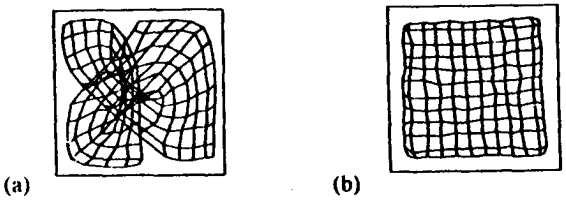


Fig. 3.25. Two-dimensional map after 8000 steps. **(a)** The basic SOM with small neighborhood, **(b)** The same neighborhood but with (3.52) used

Special Case. If the probability density function $p(x)$ of input data is *discrete-valued*, such as the case is in the famous traveling-salesman problem, the original SOM algorithm can be derived from the average expected distortion measure. (For a textbook account, see [1.40], Sect. 6.2 of that book.) In view of the above discussion this result may now be clear, because $p(x)$ at the borders of the Voronoi tessellation is then zero, and the extra term H, due to differentiation of the integration limits, then vanishes.

Modification of the Definition of the "Winner." It has been pointed out [3.19, 20] that if the matching criterion for the definition of the "winner" c is modified as

$$\sum_i h_{ci}||x - m_i||^2 = \min_j \left\{ \sum_i h_{ji}||x - m_i||^2 \right\} , \tag{3.53}$$

where h_{ci} is the same neighborhood function as that applied during learning, then the average expected distortion measure becomes an energy or potential function, the minimization of which can be carried out by the usual gradient-descent principle accurately. While this observation has certain mathematical interest, the following facts somewhat neutralize its value in neural-network modeling: 1. In the physiological explanation of the SOM (Chap. 4) the neighborhood function h_{ck} only defines the control action on the synaptic plasticity, not control of network acticity in the WTA function as implied by (3.53). 2. The h_{ji} above have to be normalized, $\sum_j h_{ji} = 1$, whereby in general $h_{ji} \neq h_{ij}$ for $i \neq j$.

For a practical ANN algorithm, computation of (3.53) for finding the "winner" is also more tedious compared with simple distance calculations.

Nonetheless, this kind of parallel theory, by analogy, may shed some extra light on the nature of the SOM process.

3.12 Point Density of the Model Vectors

3.12.1 Earlier Studies

In biological brain maps, the areas allocated to the representations of various sensory features are often believed to reflect the importance of the corresponding feature sets; the scale of such a map is somewhat loosely called the "magnification factor." As a matter of fact, different parts of a receptive surface such as the retina are transformed in the brain in different scales, almost like in mathematical *quasiconformal mappings.*

It is true that a higher magnification factor usually corresponds to a higher density of receptor cells, which might give a reason to assume that the magnification factor depends on the number of axons connecting the receptive surfaces to that brain area. However, in the nervous systems there also exist numerous "processing stations," called nuclei in the sensory pathways, whereby one cannot directly compare input-output point densities in such a mapping. In the light of the SOM theory it rather seems that the area allocated to the representation of a feature in a brain map is somehow *proportional to the statistical frequency of occurrence of that feature in observations.*

As the nervous pathways in the biological realms are very confused, we shall use the term "magnification factor" in the sequel only to mean *the inverse of point density of the* m_i. For instance, in the classical vector quantization with squared errors we had this density proportional to $[p(x)]^{\frac{n}{n+2}}$, where n is the dimensionality of x. The "magnification factor" would then be the inverse of this.

Ritter and Schulten [3.21] and *Ritter* [3.22] analyzed the point density for the linear map in the case that the map contained a very large number of codebook vectors over a finite area. If N neighbors on both sides of the "winner" and the "winner" itself were included in the neighborhood set, the asymptotic point density (denoted here M) can be shown to be $M \propto [p(x)]^r$, where the exponent is

$$r = \frac{2}{3} - \frac{1}{3N^2 + 3(N+1)^2} . \tag{3.54}$$

One thing has to be made clear first. The SOM process, although it may start with a wide neighborhood function, can have an arbitrary neighborhood function width at the end of the learning process (especially if the Gaussian kernel is used). Then at the last phases we may even have the zero-order topology case, i.e., no neighbors except the "winner" itself. However, with zero-order topological interaction the process no longer maintains the order of the codebook vectors, and this order may disappear in extensive learning with narrow neighborhoods. Therefore one has to make a compromise between the wanted approximation accuracy of $p(x)$, which is best in the VQ case, and the wanted stability of ordering, for which the neighborhood interactions are needed.

With $N = 0$ we get above the one-dimensional VQ (or the so-called scalar quantization) case, whereby $r = 1/3$. It seems that the low value of r (compared with unity) is often regarded as a handicap; it may be felt that approximation of probability density functions should be necessary in all statistical pattern recognition tasks, to which the neural networks are also supposed to bring the "optimal" solution. One has to note, however: 1. If classification is based on finding the Bayesian borders where the density functions of two classes have equal value, the same result can be obtained by comparing *any* (but the same) monotonic functions of densities. 2. Most practical applications have data vectors with very high dimensionality, say, dozens to hundreds. Then, e.g., in classical VQ, the exponent actually is $n/(n+2) \approx 1$, where n is the dimensionality of x. It is not yet quite clear what the corresponding exponent in two-dimensional SOMs with very high input density is, but it is certainly lower than unity.

In the work of Dersch and Tavan [3.23] the neighborhood function was Gaussian, and it is possible to see from Fig. 3.26 how the exponent α depends on the normalized second moment σ of the neighborhood function. (The integral values of σ correspond, very roughly, to N^2 above.)

Further works on the SOM point density can be found in [3.24, 25].

3.12.2 Numerical Check of Point Densities in a Finite One-Dimensional SOM

Strictly speaking, the scalar entity named the "point density" as a function of x has a meaning only in either of the following cases: 1. The number of

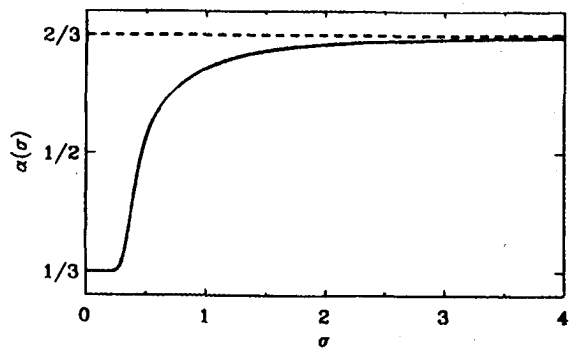


Fig. 3.26. Distortion exponent α for a Gaussian neighborhood interaction as the function of its normalized second moment σ [3.23]

points (samples) in any reasonable "volume" differential is large, or 2. The points (samples) are stochastic variables, and their differential probability of falling into a given differential "volume," i.e., the probability density $p(x)$ can be defined.

Since in vector quantization problems one aims at the minimum expected quantization error, the model or codebook vectors m_i tend to assume a more or less regular optimal configuration, and cannot be regarded as stochastic. Neither can one usually assume that their number in any differential volume is high.

Consider now the one-dimensional coordinate axis x, on which some probability density function $p(x)$ is defined. Further consider two successive points m_i and m_{i+1} on this same axis. One may regard $(m_{i+1} - m_i)^{-1}$ as the local "point density." However, to which value of x should it be related? The same problem is encountered if a functional dependence is assumed between the point density and $p(x)$. Below we shall define the point density as the inverse of the width of the Voronoi set, i.e., $[(m_{i+1} - m_i)/2]^{-1}$, and refer the density to the model m_i, which is only one choice, of course.

Asymptotic State of the One-Dimensional, Finite-Grid SOM Algorithm in Three Exemplary Cases. Consider a series of samples of the input $x(t) \in \Re$, $t = 0, 1, 2, \ldots$ and a set of k model (codebook) values $m_i(t) \in \Re$, $t = 0, 1, 2, \ldots$, whereupon i is the model index ($i = 1, \ldots, k$). For convenience assume $0 \leq x(t) \leq 1$.

The original one-dimensional self-organizing map (SOM) algorithm with at most one neighbor on each side of the best-matching m_i reads:

$$\begin{aligned} m_i(t+1) &= m_i(t) + \varepsilon(t)[x(t) - m_i(t)] \text{ for } i \in N_c \,, \\ m_i(t+1) &= m_i(t) \text{ for } i \notin N_c \,, \\ c &= \arg\min_i \{|x(t) - m_i(t)|\} \,, \text{ and} \\ N_c &= \{\max(1, c-1), c, \min(k, c+1)\} \,, \end{aligned} \tag{3.55}$$

where N_c is the neighborhood set around node c, and $\varepsilon(t)$ is a small scalar value called the learning-rate factor. In order to analyze the asymptotic values of the m_i, let us assume that the m_i are already ordered. Let the Voronoi set V_i around m_i be defined as

$$\begin{aligned} \text{for } 1 < i < k,\ V_i &= \left[\frac{m_{i-1} + m_i}{2}, \frac{m_i + m_{i+1}}{2}\right] , \\ V_1 &= \left[0, \frac{m_1 + m_2}{2}\right] , \ V_k = \left[\frac{m_{k-1} + m_k}{2}, 1\right] , \text{ and denote} \\ \text{for } 1 < i < k,\ U_i &= V_{i-1} \cup V_i \cup V_{i+1} \,, \\ U_1 &= V_1 \cup V_2 ,\ U_k = V_{k-1} \cup V_k \,. \end{aligned} \tag{3.56}$$

In other words, U_i is the set of such $x(t)$ values that are able to modify $m_i(t)$ during one learning step. Following the simple case discussed in Sect. 3.5.1 one can write the condition for stationary equilibrium of the m_i for a constant ε as:

$$\forall i,\ m_i = \mathrm{E}\,\{x \mid x \in U_i\} \,. \tag{3.57}$$

This means that every m_i must coincide with the centroid of the probability mass in the respective U_i.

For $2 < i < k - 1$ we have for the limits of the U_i:

$$\begin{aligned} A_i &= \frac{1}{2}(m_{i-2} + m_{i-1}) \,, \\ B_i &= \frac{1}{2}(m_{i+1} + m_{i+2}) \,. \end{aligned} \tag{3.58}$$

For $i = 1$ and $i = 2$ we must take B_i as above, but $A_i = 0$; and for $i = k - 1$ and $i = k$ we have A_i as above and $B_i = 1$.

Case 1: $p(x) = 2x$. The first case we discuss here is the one where the probability density function of x is linear, $p(x) = 2x$ for $0 \leq x \leq 1$ and $p(x) = 0$ for all the other values of x.

It is now straightforward to compute the centroids of the trapezoidal probability masses in the U_i:

$$\mathrm{E}\{x | x \in U_i\} = \frac{2(B_i^3 - A_i^3)}{3(B_i^2 - A_i^2)} \,. \tag{3.59}$$

The stationary values of the m_i are defined by the set of nonlinear equations

$$\forall i,\ m_i = \frac{2(B_i^3 - A_i^3)}{3(B_i^2 - A_i^2)} \tag{3.60}$$

and the solution of (3.60) is sought by the so-called *contractive mapping*. Let us denote

$$z = [m_1, m_2, \ldots, m_k]^T \,. \tag{3.61}$$

Then the equation to be solved is of the form

$$z = f(\mathbf{z}) . \tag{3.62}$$

Starting with the first approximation for $\mathbf{z}$ denoted $\mathbf{z}^{(0)}$, each improved approximation for the root is obtained recursively:

$$z^{(s+1)} = f(z^{(s)}) . \tag{3.63}$$

In this case one may select for the first approximation of the m_i equidistant values.

With a small number of grid points, (3.63) converges reasonably fast, but already with 100 grid points the required number of steps for the accuracy of, say, five decimal places may be about five thousand.

It may now be expedient to define the point density q_i around m_i as the inverse of the length of the Voronoi set, or $q_i = [(m_{i+1} - m_{i-1})/2]^{-1}$.

The problem expressed in a number of previous works, e.g., [3.21, 22, 23], is to find out whether q_i could be approximated by the functional form const.$[p(m_i)]^\alpha$. Previously this was only shown for the continuum limit, i.e. for an infinite number of grid points. The present numerical analysis allows us to derive results for finite-length grids, too. Assuming tentatively that the power law holds for the models m_i through m_j (leaving aside models near to the ends of the grid), we shall then have

$$\alpha = \frac{\log(m_{i+1} - m_{i-1}) - \log(m_{j+1} - m_{j-1})}{\log[p(m_j)] - \log[p(m_i)]} . \tag{3.64}$$

Naturally, more values of the m_i could be taken for improved accuracy. In Table 3.3, using $i = 4$ and $j = k - 3$, between which the border effects may be assumed as negligible, the exponent α has been estimated from (3.64) for 10, 25, 50, and 100 grid points, respectively.

Case 2: $p(x) = 3x^2$ *(convex).* Now we have the system of equations

$$\forall i, \quad m_i = \frac{3(B_i^4 - A_i^4)}{4(B_i^3 - A_i^3)} \tag{3.65}$$

and the approximations for α are in Table 3.3.

Case 3: $p(x) = 3x - \frac{3}{2}x^2$ *(concave).* The system of equations reads

$$\forall i, \quad m_i = \frac{8(B_i^3 - A_i^3) - 3(B_i^4 - A_i^4)}{12(B_i^2 - A_i^2) - 4(B_i^3 - A_i^3)} \tag{3.66}$$

and the approximations for α are also in Table 3.3.

These simulations show convincingly that for three qualitatively different $p(x)$ the exponent α, even for a reasonably small number of grid points, is fairly close to the value of $\alpha = 0.6$ as derived in the continuum limit in [3.22], in the case of one neighbor on both sides of the best-matching m_i.

Table 3.3. Exponent derived from the SOM algorithm

	Exponent α		
Grid points	Case 1	Case 2	Case 3
10	0.5831	0.5845	0.5845
25	0.5976	0.5982	0.5978
50	0.5987	0.5991	0.5987
100	0.5991	0.5994	0.5990

Numerically Accurate Optimum of the One-Dimensional SOM Distortion Measure with Finite Grids: Case 1. Equation 3.48 can also be written as

$$E = \sum_i \sum_j \int_{x \in V_i} h_{ij} \|x - m_j\|^2 p(x) dx , \tag{3.67}$$

where i and j run over all the values for which h_{ij} has been defined, and V_i is the Voronoi set around m_i.

In the simple one-dimensional case, when h_{ij} is defined as

$$h_{ij} = 1 \text{ if } |i - j| < 2 , \text{ and } h_{ij} = 0 \text{ otherwise} , \tag{3.68}$$

when we take Case 1, or $p(x) = 2x$ for $0 \leq x \leq 1$, $p(x) = 0$ otherwise, and when we assume the m_i as ordered in the ascending sequence, (3.67) becomes

$$\begin{aligned} E &= 2 \sum_i \sum_{j \in N_i} \int_{C_i}^{D_i} (x - m_j)^2 x dx \\ &= \sum_i \sum_{j \in N_i} m_j^2 (D_i^2 - C_i^2) - \frac{4}{3} m_j (D_i^3 - C_i^3) + \frac{1}{2}(D_i^4 - C_i^4) \end{aligned} \tag{3.69}$$

where the *neighborhood set of indices* N_i was defined in (3.55), and the borders C_i and D_i of the Voronoi set V_i are

$$\begin{aligned} C_1 &= 0 , \\ C_i &= \frac{m_{i-1} + m_i}{2} \quad \text{for } 2 \leq i \leq k , \\ D_i &= \frac{m_i + m_{i+1}}{2} \quad \text{for } 1 \leq i \leq k - 1 , \\ D_k &= 1 . \end{aligned} \tag{3.70}$$

When forming the accurate gradient of E, it must be noticed that index i is contained in N_{i-1}, N_i, and N_{i+1}, whereupon

$$\begin{aligned} \frac{\partial E}{\partial m_i} = \frac{\partial}{\partial m_i} \sum_{j \in N_{i-1}} \Big(& m_j^2 (D_{i-1}^2 - C_{i-1}^2) - \frac{4}{3} m_j (D_{i-1}^3 - C_{i-1}^3) \\ & + \frac{1}{2}(D_{i-1}^4 - C_{i-1}^4) \Big) \end{aligned}$$

$$+\frac{\partial}{\partial m_i}\sum_{j\in N_i}\left(m_j^2(D_i^2-C_i^2)-\frac{4}{3}m_j(D_i^3-C_i^3)+\frac{1}{2}(D_i^4-C_i^4)\right)$$
$$+\frac{\partial}{\partial m_i}\sum_{j\in N_{i+1}}\left(m_j^2(D_{i+1}^2-C_{i+1}^2)-\frac{4}{3}m_j(D_{i+1}^3-C_{i+1}^3)\right.$$
$$\left.+\frac{1}{2}(D_{i+1}^4-C_{i+1}^4)\right)\ . \qquad (3.71)$$

The result of this differentiation is given as follows (notice that $C_i = D_{i-1}$):

$$\frac{\partial E}{\partial m_1} = 2m_1D_2^2-\frac{4}{3}D_2^3-m_3^2C_2+2m_3C_2^2-C_2^3\ ,$$

$$\frac{\partial E}{\partial m_2} = m_1^2D_2-2m_1D_2^2+D_2^3+2m_2D_3^2-\frac{4}{3}D_3^3-m_3^2C_2+2m_3C_2^2$$
$$-C_2^3-m_4^2C_3+2m_4C_3^2-C_3^3\ ,$$

$$\frac{\partial E}{\partial m_i} = m_{i-2}^2D_{i-1}-2m_{i-2}D_{i-1}^2+D_{i-1}^3+m_{i-1}^2D_i-2m_{i-1}D_i^2+D_i^3$$
$$-m_{i+1}^2C_i+2m_{i+1}C_i^2-C_i^3-m_{i+2}^2C_{i+1}+2m_{i+2}C_{i+1}^2-C_{i+1}^3$$
$$+2m_i(D_{i+1}^2-C_{i-1}^2)-\frac{4}{3}(D_{i+1}^3-C_{i-1}^3)\quad \text{for}\quad 2<i<k-1\ ,$$

$$\frac{\partial E}{\partial m_{k-1}} = m_{k-3}^2D_{k-2}-2m_{k-3}D_{k-2}^2+D_{k-2}^3+m_{k-2}^2D_{k-1}$$
$$-2m_{k-2}D_{k-1}^2+D_{k-1}^3-m_k^2C_{k-1}+2m_kC_{k-1}^2$$
$$-C_{k-1}^3+2m_{k-1}(1-C_{k-2}^2)-\frac{4}{3}(1-C_{k-2}^3)\ ,\ \text{and}$$

$$\frac{\partial E}{\partial m_k} = m_{k-2}^2D_{k-1}-2m_{k-2}D_{k-1}^2+D_{k-1}^3$$
$$+2m_k(1-C_{k-1}^2)-\frac{4}{3}(1-C_{k-1}^3)\ . \qquad (3.72)$$

The question is whether one can obtain the optimal values of the m_i by the gradient-descent method, i.e.,

$$\forall i\ ,\quad m_i(t+1)=m_i(t)-\lambda(t)\cdot\partial E/\partial m_i|_t\ , \qquad (3.73)$$

where $\lambda(t)$ is a suitable small scalar factor. In the present problem E is of the fourth degree in the m_i and at least one kind of spurious local optimum has been found: for instance, when starting with the asymptotic m_i values obtained from the SOM algorithm and keeping $\lambda(t)$ at a value of the order of .001 or smaller, a very shallow local minimum of E has been reached, which has given the wrong value of about .6 for α. However, with $\lambda(t) > .01$ (even with $\lambda(t) = 10$) and starting with very different initial values for the m_i, the process robustly converges to a unique global minimum. After computation of the optimal values $\{m_i\}$, in order to facilitate a direct comparison with the values presented in Table 3.3 the exponent α of the tentative power law was computed from (3.64) of the previous section and presented in Table 3.4 for different lengths of the grid.

Table 3.4. Exponent derived from the SOM distortion measure

Grid points	Exponent α
10	0.3281
25	0.3331
50	0.3333
100	0.3331

Clearly the computed α is an approximation of the value of 1/3, the same as the exponent in vector quantization for $n = 1$ and $r = 2$, rather than of $\alpha = .6$ of the simple SOM algorithm.

The result that transpired in this numerical check is that the point density of the model (codebook) vectors resulting as asymptotic values in the basic SOM algorithm is different from that ensuing as the parameter values of the SOM distortion measure at its minimum. Nonetheless the m_i in both cases can be regarded as the nodes of an "elastic" network that is regressed onto the manifold of the input samples in an orderly fashion. The conclusion is thus that the Robbins-Monro stochastic approximation does not exactly lead to the basic SOM algorithm, but the algorithm and the distortion measure may be two optional ways to define the self-organizing map.

3.13 Practical Advice for the Construction of Good Maps

Although it is possible to obtain some kind of maps without taking into account any precautions, nonetheless it will be useful to pay attention to the following advice in order that the resulting mappings be stable, well oriented, and least ambiguous. Let us recall that the initialization problem was already discussed in Sect. 3.7. The reader is now adviced to study the two extensive software packages SOM_PAK [3.26] and LVQ_PAK [3.27].

Form of the Array. For visual inspection, the *hexagonal lattice* is to be preferred, because it does not favor horizontal and vertical directions as much as the rectangular array. The *edges* of the array ought to be rectangular rather than square, because the "elastic network" formed of the reference vectors m_i must be oriented along with $p(x)$ and be stabilized in the learning process. Notice that if the array were, e.g., circular, it would have no stable

orientation in the data space; so any oblongated form is to be preferred. On the other hand, since the m_i have to approximate $p(x)$, it would be desirable to find such dimensions for the array that roughly correspond to the major dimensions of $p(x)$. Therefore, visual inspection of the rough form of $p(x)$, e.g., by *Sammon's mapping* [1.34] (Sect. 1.3.2) ought to be done first.

Learning with a Small Number of Available Training Samples. Since for a good statistical accuracy the complete learning process may require an appreciable number, say, 100'000 steps, and the number of available samples is usually much smaller, it is obvious that the samples must be used *reiteratively* in training. Several alternatives then exist: the samples may be applied cyclically or in a randomly permuted order, or picked up at random from the basic set (so-called *bootstrap learning*). It has turned out in practice that ordered cyclic application is not noticeably worse than the other, mathematically better justifiable methods.

Enhancement of Rare Cases. It may also be obvious from the above that the SOM in one way or another tends to represent $p(x)$. However, in many practical problems important cases (input data) may occur with small statistical frequency, whereby they are not able to occupy any territory at all in the SOM. Therefore, such important cases can be enhanced in learning by an arbitrary amount by taking a higher value of α or h_{ci} for these samples, or repeating these samples in a random order a sufficient number of times during the learning process. Determination of proper enhancement in learning should be done in cooperation with the end users of these maps.

Scaling of the Pattern Components. This is a very subtle problem. One may easily realize that the orientation, or ordered "regression" of the reference vectors in the input space must depend on the scaling of the components (or dimensions) of the input data vectors. However, if the data elements have already been represented in different scales, there does not exist any simple rule to determine what kind of optimal rescaling should be used before entering the training data to the learning algorithm. The first guess, which is usually rather effective, especially with high input dimensionality, is to normalize the variance of each component over the training data. One may also try many heuristically justifiable rescalings and check the quality of the resulting maps by means of Sammon's mapping or average quantization errors.

Forcing Representations to a Wanted Place on the Map. Sometimes, especially when the SOM is used to monitor experimental data, it may be desirable to map "normal" data onto a specified location (say, into the middle) of the map. In order to force particular data into wanted places, it is advisable to use their copies for the initial values of reference vectors at these locations, and to keep the learning-rate factor α low for these locations during their updating.

Monitoring of the Quality of Learning. Different learning processes can be defined starting with different initial values $m_i(0)$, and applying different sequences of the training vectors $x(t)$ and different learning parameters. It is obvious that some optimal map for the same input data may exist. It may also be obvious that the best map is expected to yield the smallest average *quantization error*, approximately at least, because it is then fitted best to the same data. The mean of $||x - m_c||$, defined via inputting the training data once again after learning, is then a useful performance index. Therefore, an appreciable number (say, several dozens) of random initializations of the $m_i(0)$ and different learning sequences ought to be tried, and the map with the minimum quantization error might be selected.

The accuracy of the maps in *preserving the topology*, or neighborhood relations, of the input space has been measured in various ways. One approach is to compare the relative positions of the reference vectors with the relative positions of the corresponding units on the map [3.28]. For example, the number of times the Voronoi region of another map unit "intrudes" the middle of the reference vectors of two neighbor units can be measured [3.29]. A different approach is to consider, for each input vector, the distance of the best-matching unit and the second-best-matching unit on the map: If the units are not neighbors, then the topology is not preserved [3.30, 31] . When the distance between map units is defined suitably, such a measure can be combined with the quantization error to form a single measure of map goodness [3.32]. Although it is not at present possible to indicate the best measure of map quality, these measures may nevertheless be useful in choosing suitable learning parameters and map sizes.

3.14 Examples of Data Analyses Implemented by the SOM

This section contains demonstrations of the SOM, intended to visualize so-called *data matrices*. In the simplest cases we assume that all items possess values for all their attributes; later in Sect. 3.14.2 we consider another case in which almost all of the items lack some of their attributes.

3.14.1 Attribute Maps with Full Data Matrix

Abstract Hierarchical Data Structure. Consider a finite set of items, each one having a number of characteristics or *attributes*. The latter can be binary, or integer- or continuous-valued; at any rate the sets of attributes should be comparable in some metric.

If we first apply the SOM to abstract data vectors consisting of hypothetical attributes, we can thereby define very clear data structures. We shall consider an example with implicitly defined (hierarchical) structures in the

primary data, which the map algorithm is then supposed to reveal and display. Although the SOM is a single-level network, it can produce a hierarchical representation of the relations implicit in the primary data.

In Table 3.5, 32 items, with five hypothetical attributes each, are recorded in a data matrix. (Let us recall that this example is completely artificial.) Each of the columns represents one item, and for later inspection the items are labeled "A" through "6", although these labels are not referred to during learning.

Table 3.5. Input data matrix

Attribute	Item A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
a_1	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a_2	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
a_3	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
a_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
a_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

The attribute values $(a_1, a_2, \ldots, a_5)$ were defined artificially and they constitute the pattern vector x that acts as a set of signal values at the inputs of the network of the type of Fig. 3.3. During training, the vectors x were selected from Table 3.5 at random. Sampling and adaptation was continued iteratively until one could regard the asymptotic state of the SOM as stationary. Such a "learned" network was then calibrated using the items from Table 3.5 and labeling the map units according to the best match with the different items. Such a labeled map is shown in Fig. 3.27. It is discernible that the "images" of different items are related according to a taxonomic graph where the different branches are visible. For comparison, Fig. 3.28 illustrates the so-called *minimal spanning tree* (where the most similar pairs of items are linked) that describes the similarity relations of the items in Table 3.5.

Map of a Binary Data Matrix. Attributes are usually variables with scalar-valued discrete or continuous values, but they may also attain qualitative properties such as "good" or "bad". If the property of being "good" or

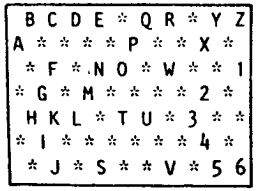


Fig. 3.27. Self-organized map of the data matrix in Table 3.5

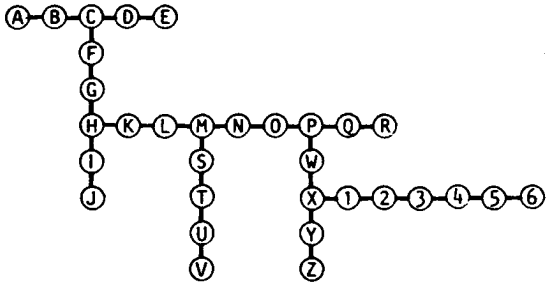


Fig. 3.28. Minimal spanning tree of the data matrix in Table 3.5

"bad", respectively, should be describable by a numerical attribute, it would be simplest to assume that such an attribute has the binary value, say 1 or 0, depending on the presence vs. absence of that attribute, respectively. Then the (unnormalized) similarity between two (binary) attribute sets may be defined in terms of the number of attributes common to both sets, i.e., as the dot product of the respective attribute vectors. It might seem more effective to use the value +1 to indicate the presence of an attribute, and -1 for its absence, respectively; however, if we *normalize* the input vectors, in their subsequent comparison using the dot product the attribute values 0 have a qualitatively similar effect as negative components in a comparison on the basis of vectorial differences. Euclidean distances can naturally be used directly for comparison, too.

To illustrate the self-organizing result with a concrete model simulation [2.73], consider the data given in Table 3.6. Each column is a schematic description of an animal, based on the presence (= 1) or absence (= 0) of some of the 13 different attributes given on the left. Some attributes, such as "feathers" and "2 legs" are correlated, indicating more significant differences than the other attributes, but we shall not take this correlation into account in learning in any way. In the following, we will take each column for the input vector of the animal indicated at the top. The animal name itself does not belong to the vector but instead specifies the label of the animal in the calibration of the map.

The members of the data set were presented iteratively and in a random order to a SOM of 10×10 neurons subject to the adaptation process described above. The initial connection strengths between the neurons and their $n = 29$ input lines were chosen to be small random values, i.e. no prior order was imposed. However, after a total of 2000 presentations, each neuron became more or less responsive to one of the occuring attribute combinations and simultaneously to one of the 16 animal names, too. Thus we obtain the map shown in Fig. 3.29 (the dots indicate neurons with weaker responses). It is very apparent that the spatial order of the responses has captured the essential "family relationships" among the animals. Cells responding to, e.g.,

Table 3.6. Animal names and their attributes

		dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
is	small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
	medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
has	2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
likes to	hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
	run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
	fly	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

"birds" occupy the left part of the lattice, "hunters" such as "tiger", "lion" and "cat" are clustered toward the right, and more "peaceful" species such as "zebra", "horse" and "cow" are situated in the upper middle. Within each cluster, a further grouping according to similarity is discernible.

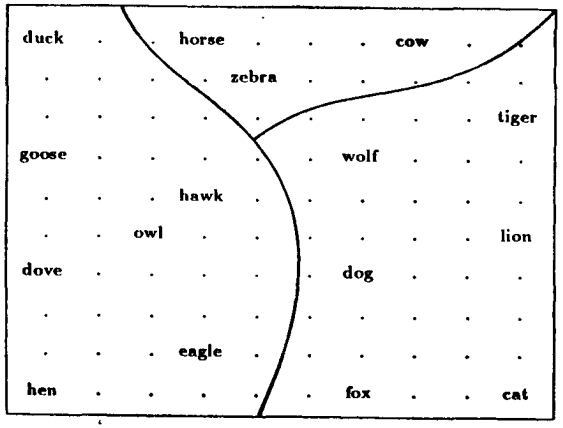


Fig. 3.29. After the network had been trained with inputs describing attribute sets from Table 3.6, the map was calibrated by the columns of Table 3.6 and labeled correspondingly. A grouping according to similarity has emerged

3.14.2 Case Example of Attribute Maps Based on Incomplete Data Matrices (Missing Data): "Poverty Map"

It has turned out that the SOM is a very robust algorithm, compared with many other neural models. One of the frequently encountered problems in practical (especially statistical) applications is caused by missing data [3.33, 34]. However, if the number of attributes taken into account is appreciable, say, at least on the order of hundreds, an appreciable fraction of data may be missing without making the similarity comparison impossible.

To start with, it is advisable to *normalize each attribute scale* such that its variance taken over all the items is unity. Similarly one may subtract the mean from each attribute: thus the scales are said to be $(0, 1)$ -normalized.

Comparison of an item with the codebook vectors is best made in the Euclidean metric, unless some better metric is deducible from the nature of the problem. Then, however, only the known components of x, and the corresponding components of each m_i, can be taken into account in the comparison: during this step the vectors are redimensioned correspondingly.

The data used in this case example were the same as in Sect. 1.3.2, namely, from the statistics published by *World Bank* [1.26]. The 39 indicators describe the poverty of the countries, or their citizens. All indicators are relative to population.

For 126 countries of the world listed, 12 or more indicator values of 39 possible ones were given to 78 countries and these countries were then taken to the data matrix used in training the SOM process. In Fig. 3.30, these countries are labeled by capital letters. For the countries labeled by lowercase symbols, more than 11 attribute values were missing, and they were mapped onto the SOM after training.

The "poverty map" of the countries is presented in Fig. 3.30. The symbols used in the map have been explained in Table 3.7. It has to be noted that horizontal or vertical directions have no explicit meaning in this map: only the local geometric relations are important. One might say that countries that are mapped close to each other in the SOM have a similar state of development, expenditure pattern, and policy.

3.15 Using Gray Levels to Indicate Clusters in the SOM

From the above map and the previous ones, however, we still do not see any boundaries between eventual *clusters*. That problem will be discussed in this section.

A graphic display called the *U-matrix*, to illustrate the clustering of codebook vectors in the SOM has been developed by *Ultsch and Siemon* [3.35], as well as *Kraaijveld et al.* [3.36]. They suggested a method in which the average distances between neighboring codebook vectors are represented by shades in a gray scale (or eventually pseudocolor scales might be used). If the

	BEL	SWE	ITA	YUG	rom	-	CHN TUR	bur IDN	MDG	-	BGD NPL	btn	afg gin MLI ner SLE						
AUT che DEU FRA	NLD	JPN	-	bgr csk	HUN POL PRT	-	-	-	gab lbr	khm	PAK	moz mrt sdn yem							
-	-	ESP	GRC	-	-	THA	MAR	-	IND	caf	SEN	MWI TZA uga							
DNK GBR NOR	FIN	IRL	-	URY	ARG	ECU mex	-	EGY	hti	lao png ZAR	-	tcd							
-	-	-	KOR	-	zaf	-	TUN	dza irq	GHA	NGA	-	ETH							
CAN USA	-	ISR	-	-	COL PER	lbn	lby	ZWE	omn	-	ago	hvo							
-	AUS	-	MUS tto	-	-	IRN PRY syr	hnd	BWA	KEN	BEN CIV	cog som	bdi RWA							
NZL	-	-	CHL	PAN	alb	mng sau	-	vnm	jor nic	-	-	tgo							
-	HKG SGP	are	CRI VEN	kwt	JAM MYS	-	DOM LKA PHL	-	BOL BRA SLV	-	GTM	CMR Iso nam ZMB							

Fig. 3.30. "Poverty map" of 126 countries of the world. The symbols written in capital letters correspond to countries used in the formation of the map; the rest of the symbols (in lower case) signify countries for which more than 11 attribute values were missing, and which were mapped to this SOM after the learning phase

average distance of neighboring m_i is small, a light shade is used; and vice versa, dark shades represent large distances. A "cluster landscape" formed over the SOM then clearly visualizes the classification.

The case example of Sect. 3.14.2, with the gray-level map superimposed on it, is now shown in Fig. 3.31, and it clearly shows "ravines" between clusters. The interpretation is left to the reader.

3.16 Interpretation of the SOM Mapping

The Self-Organizing Map combines nonlinear projection (Sect. 1.3.2) and clustering (Sect. 1.3.4) methods in an ordered vector quantization graph. Therefore the mapping that it produces is expected to be explainable in terms of some classical concepts of statistics.

3.16.1 "Local Principal Components"

The main difference between the SOM and the principal component analysis (PCA) is that the latter describes the global statistical properties of the data distribution: the first "principal axis" defines the direction in which the variance of the distribution is largest, the second "principal axis" that

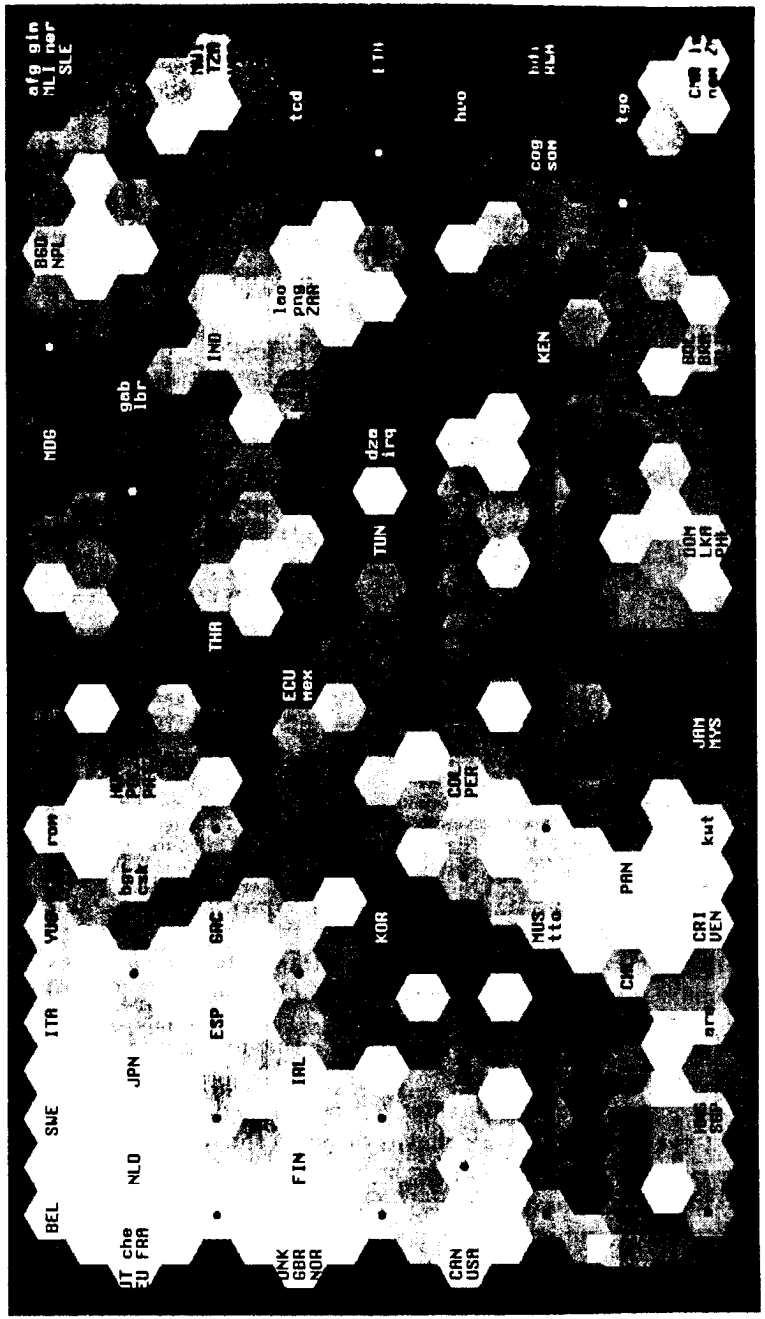


Fig. 3.31. "Poverty map," with the clustering shown by shades of gray

Table 3.7. Legend of symbols used in Figs. 3.29 and 3.30:

AFG	Afghanistan	GRC	Greece	NOR	Norway
AGO	Angola	GTM	Guatemala	NPL	Nepal
ALB	Albania	HKG	Hong Kong	NZL	New Zealand
ARE	United Arab Emirates	HND	Honduras	OAN	Taiwan, China
ARG	Argentina	HTI	Haiti	OMN	Oman
AUS	Australia	HUN	Hungary	PAK	Pakistan
AUT	Austria	HVO	Burkina Faso	PAN	Panama
BDI	Burundi	IDN	Indonesia	PER	Peru
BEL	Belgium	IND	India	PHL	Philippines
BEN	Benin	IRL	Ireland	PNG	Papua New Guinea
BGD	Bangladesh	IRN	Iran, Islamic Rep.	POL	Poland
BGR	Bulgaria	IRQ	Iraq	PRT	Portugal
BOL	Bolivia	ISR	Israel	PRY	Paraguay
BRA	Brazil	ITA	Italy	ROM	Romania
BTN	Bhutan	JAM	Jamaica	RWA	Rwanda
BUR	Myanmar	JOR	Jordan	SAU	Saudi Arabia
BWA	Botswana	JPN	Japan	SDN	Sudan
CAF	Central African Rep.	KEN	Kenya	SEN	Senegal
CAN	Canada	KHM	Cambodia	SGP	Singapore
CHE	Switzerland	KOR	Korea, Rep.	SLE	Sierra Leone
CHL	Chile	KWT	Kuwait	SLV	El Salvador
CHN	China	LAO	Lao PDR	SOM	Somalia
CIV	Cote d'Ivoire	LBN	Lebanon	SWE	Sweden
CMR	Cameroon	LBR	Liberia	SYR	Syrian Arab Rep.
COG	Congo	LBY	Libya	TCD	Chad
COL	Colombia	LKA	Sri Lanka	TGO	Togo
CRI	Costa Rica	LSO	Lesotho	THA	Thailand
CSK	Czechoslovakia	MAR	Morocco	TTO	Trinidad and Tobago
DEU	Germany	MDG	Madagascar	TUN	Tunisia
DNK	Denmark	MEX	Mexico	TUR	Turkey
DOM	Dominican Rep.	MLI	Mali	TZA	Tanzania
DZA	Algeria	MNG	Mongolia	UGA	Uganda
ECU	Ecuador	MOZ	Mozambique	URY	Uruguay
EGY	Egypt, Arab Rep.	MRT	Mauritania	USA	United States
ESP	Spain	MUS	Mauritius	VEN	Venezuela
ETH	Ethiopia	MWI	Malawi	VNM	Viet Nam
FIN	Finland	MYS	Malaysia	YEM	Yemen, Rep.
FRA	France	NAM	Namibia	YUG	Yugoslavia
GAB	Gabon	NER	Niger	ZAF	South Africa
GBR	United Kingdom	NGA	Nigeria	ZAR	Zaire
GHA	Ghana	NIC	Nigaragua	ZMB	Zambia
GIN	Guinea	NLD	Netherlands	ZWE	Zimbabwe

direction of all directions orthogonal to the first principal axis in which the residual variance is largest, and so on.

The SOM, however, can be characterized as a two-dimensional, finite-element "elastic surface" or network that is fitted to the distribution of the input samples. This network has *locally*, at every node of it, two principal directions that are found considering the differences between the neighboring vectors. For instance, the subset of reference vectors $\{m_j\}$ in the closest neighborhood of reference vector m_i and including the latter may be regarded as a smoothed set of samples that is trying to represent a *local two-dimensional hyperplane*. The principal axes of this hyperplane, and the two largest principal components of this subset can be computed in the normal way from the subset of neighboring reference vectors.

As the reference vectors resulting in the smoothing process carried out by the SOM may not accurately represent the original statistics of the input samples, we may only qualitatively regard the two principal components of the neighborhood set as the "local principal components." If we want more than two of them, we must fit a higher-dimensional SOM to the input data. Nonetheless, as the "local principal components" can be computed readily and in an almost unique way (depending only on the SOM process and the definition of the neighborhood), they allow one to get an illustrative insight into complex and "noisy" data distributions.

One may further apply the classical *factor analysis* (Sect. 1.3.1) to each neighborhood set of the reference vectors, in order to find the *factor loadings* of the input variables in each local area of the SOM. Notice that the local factor loadings may be totally different in different domains of the input data.

Notice that it would be absurd to carry out a local PCA, or to compute the local factors on the basis of the subset of input samples that are mapped to neighboring Voronoi sets. Consider, for instance, Fig. 3.11 and look what distribution the samples in the union of any three neighboring Voronoi sets have: no "principal axes" have any sense. On the other hand, the neighboring SOM reference vectors lie neatly in a direction that takes into account the global form of the distribution, but still the local directions are sensitive to local statistics. Therefore it is more sensible to define the "local factors" on the basis of the reference vectors. In other words, like in the mathematical discipline called differential geometry, the local directions must fulfill certain "compatibility conditions" that the SOM automatically takes into account.

3.16.2 Contribution of a Variable to Cluster Structures

This analysis is related to the local PCA and local factor analysis but is computationally simpler and more directly describes the discriminatory power of an input variable in the mapping.

Consider the vectorial differences between neighboring reference vectors. Similary one can form the respective differences of each vector component separately. If the correlation between the vectorial differences and the differences of some component in a local area of the SOM is large, this component (variable) has a significant contribution to the cluster structure and a large explanatory power in that domain of values.

The discriminatory power of a variable is manifested most strongly at the cluster borders, where one is looking for variables that make the biggest difference between the neighboring clusters.

Example. In Fig. 3.32, the animal example defined in Table 3.6 and depicted in Fig. 3.29 has been interpreted by both of the above methods.

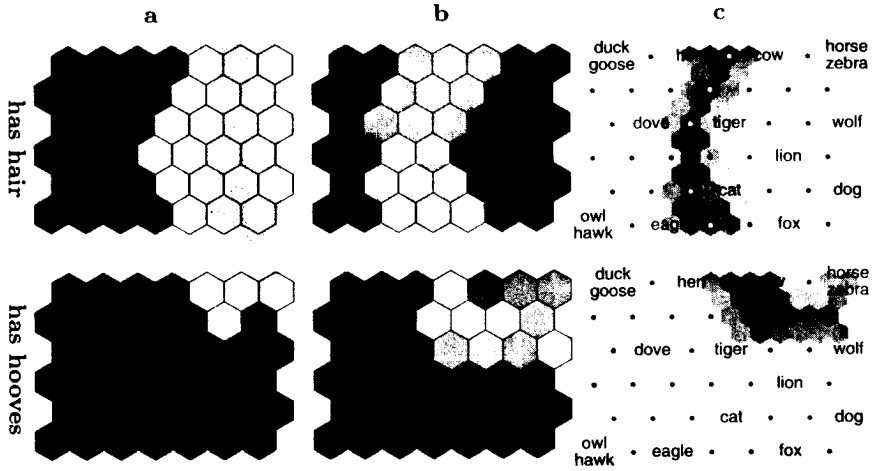


Fig. 3.32. Illustration of the two interpretation methods applied to the animal data. The top row visualizes the variable "has hair" and the bottom row "has hooves," respectively. **(a)** The component planes. The shade of gray describes the value of the respective component of the reference vectors (white: large, black: small). **(b)** The contribution of the variables in the two local factors (white: maximal contribution, black: minimal contribution). **(c)** The (spatially smoothed) contribution of the variables in the local cluster structures (dark: large contribution, white: minimal contribution)

3.17 Speedup of SOM Computation

3.17.1 Shortcut Winner Search

If there are M map units (neurons) in the SOM, and one stipulates that for a certain statistical accuracy the number of updating operations per unit shall be some constant (say, on the order of 100), then the total number of comparison operations to be performed during learning by an exhaustive search of the winners is $\sim M^2$.

By a tree-structured multilayer SOM architecture to be discussed in Sect. 5.3 it will be possible to reduce the number of searching operations to $\sim M \log M$. However, since the winner is thereby determined in a multistep decision process in which the later decisions depend on the earlier ones, the partition of the input signal space is not exactly the same as the Voronoi tessellation discussed earlier.

We will now show that the total number of comparison operations can be made $\sim M$, provided that the training vectors have been given in the beginning, i.e., their set is finite and closed. *Koikkalainen* [3.37, 38] has used a somewhat similar idea in the tree-structured SOM, but the scheme presented below [3.39] can be used in connection with any traditional SOM. Moreover,

in principle at least, the decision can also be regarded as a one-level process, comparable to the basic SOM.

Assume that we are somewhere in the middle of an iterative training process, whereby the last winner corresponding to each training vector has been determined at an earlier cycle. If the training vectors are expressed as a linear table, a *pointer* to the corresponding *tentative winner location* can be stored with each training vector (Fig. 3.33).

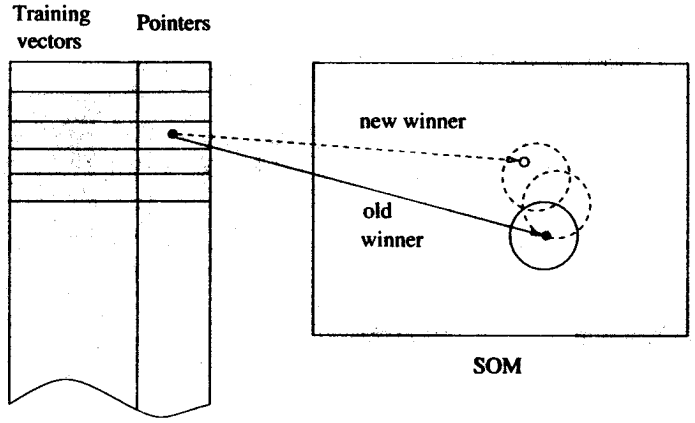


Fig. 3.33. Finding the new winner in the vicinity of the old one, whereby the old winner is directly located by a pointer. The pointer is then updated

Assume further that the SOM is already smoothly ordered although not yet asymptotically stable. This is the situation, e.g., during the lengthy fine-tuning phase of the SOM, whereupon the size of the neighborhood set is also constant and small. Consider that updating of a number of map units is made before the same training input is used again some time later. Nonetheless it may be clear that if the sum of the corrections made during this period is not large, the new winner is found at or in the vicinity of the old one. Therefore, in searching for the best match, it will suffice to locate the map unit corresponding to the associated pointer, and then to perform a local search for the winner in the neighborhood around the located unit. This will be a significantly faster operation than an exhaustive winner search over the whole SOM. The search can first be made in the immediate surround of the said location, and only if the best match is found at its edge, searching is continued in the surround of the preliminary best match, until the winner is one of the middle units in the search domain. After the new winner location has been identified, the associated pointer in the input table is replaced by the pointer to the new winner location.

For instance, if the array topology of the SOM is hexagonal, the first search might be made in the neighborhood consisting of the seven units at

or around the winner. If the tentative winner is one of the edge units of this neighborhood, the search must be continued in the new neighborhood of seven units centered around the last tentative winner. Notice that in each new neighborhood, only the three map units that have not yet been checked earlier need to be examined.

This principle can be used with both the usual incremental-learning SOM and its batch-computing version.

A benchmark with two large SOMs relating to our recent practical experiments has been made. The approximate codebook vector values were first computed roughly by a traditional SOM algorithm, whereafter they were fine-tuned using this fast method. During the fine-tuning phase, the radius of the neighborhood set in the hexagonal lattice decreased linearly from 3 to 1 units equivalent to the smallest lattice spacings, and the learning-rate factor at the same time decreased linearly from 0.02 to zero. There were 3645 training vectors for the first map, and 9907 training vectors for the second map, respectively. The results are reported in Table 3.8.

Table 3.8. Speedup due to shortcut winner search

Input dimensionality	Map size	Speedup factor in winner search	Speedup factor in training
270	315	43	14
315	768	93	16

The theoretical maximum of speedup in winner search is: 45 for the first map, and 110 for the second map, respectively. The training involves the winner searches, codebook updating, and overhead times due to the operating system and the SOM software used. The latter figures may still be improved by optimization of the management of the tables.

3.17.2 Increasing the Number of Units in the SOM

Several suggestions for "growing SOMs" (cf., e.g. [3.40–54]) have been made. The detailed idea presented below has been optimized in order to make very large maps, and is believed to be new. The basic idea is to estimate good initial values for a map that has plenty of units, on the basis of asymptotic values of a map with a much smaller number of units.

As the general nature of the SOM process and its asymptotic states is now fairly well known, we can utilize some "expert knowledge" here. One fact is that the asymptotic distribution of codebook vectors is generally smooth, at least for a continuous, smooth probability density function (pdf) of input, and therefore the lattice spacings can be smoothed, interpolated, and extrapolated locally.

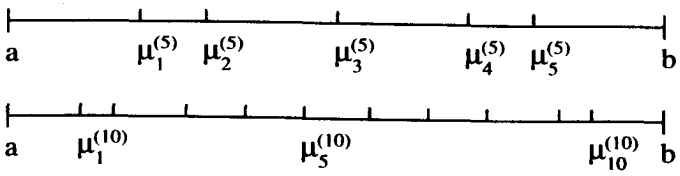


Fig. 3.34. Asymptotic values for the μ_i for two lengths of the array

As an introductory example consider, for instance, the one-dimensional SOM and assume tentatively a uniform pdf of the scalar input in the range $[a, b]$. Then we have the theoretical asymptotic codebook values for different numbers of map units that approximate the same pdf, as shown in Fig. 3.34.

Assume now that we want to estimate the locations of the codebook values for an *arbitrary* pdf and for a 10-unit SOM on the basis of known codebook values of the 5-unit SOM. A linear *local* interpolation-extrapolation scheme can then be used. For instance, to interpolate $\mu_5^{(10)}$ on the basis of $\mu_2^{(5)}$ and $\mu_3^{(5)}$, we first need the *interpolation coefficient* λ_5, computed from the two ideal lattices with uniform pdf:

$$\mu_5^{(10)} = \lambda_5 \mu_2^{(5)} + (1 - \lambda_5)\mu_3^{(5)} , \tag{3.74}$$

from which λ_5 for $\mu_5^{(10)}$ can be solved. If then, for an arbitrary pdf, the *true* values of $\mu_2'^{(5)}$ and $\mu_3'^{(5)}$ have been computed, the estimate of the true $\hat{\mu}_5'^{(10)}$ is

$$\hat{\mu}_5'^{(10)} = \lambda_5 \mu_2'^{(5)} + (1 - \lambda_5)\mu_3'^{(5)} . \tag{3.75}$$

Notice that a similar equation can also be used for the *extrapolation* of, say, $\mu_1^{(10)}$ on the basis of $\mu_1^{(5)}$ and $\mu_2^{(5)}$.

Application of local interpolation and extrapolation to *two-dimensional* SOM lattices (rectangular, hexagonal, or other) is straightforward, although the expressions become a little more complicated. Interpolation and extrapolation of a codebook vector in a two-dimensional lattice must be made on the basis of vectors defined at least in *three* lattice points. As the maps in practice may be very nonlinear, the best estimation results are usually obtained with the three closest reference vectors.

Consider two similarly positioned overlapping "ideal" two-dimensional lattices in the same plane, with the codebook vectors $m_h^{(d)} \in \Re^2, m_i^{(s)} \in \Re^2, m_j^{(s)} \in \Re^2$, and $m_k^{(s)} \in \Re^2$ its nodes, where the superscript d refers to a "dense" lattice, and s to a "sparse" lattice, respectively. If $m_i^{(s)}, m_j^{(s)}$, and $m_k^{(s)}$ do not lie on the same straight line, then in the two-dimensional signal plane any $m_h^{(d)}$ can be expressed as the linear combination

$$m_h^{(d)} = \alpha_h m_i^{(s)} + \beta_h m_j^{(s)} + (1 - \alpha_h - \beta_h) m_k^{(s)} , \tag{3.76}$$

where α_h and β_h are interpolation-extrapolation coefficients. This is a two-dimensional vector equation from which the two unknowns α_h and β_h can be solved.

Consider then two SOM lattices with the same topology as in the ideal example, in a space of arbitrary dimensionality. When the true pdf may also be arbitrary, we may not assume the lattices of true codebook vectors as planar. Nonetheless we can perform a *local linear estimation* of the true codebook vectors $m_h'^{(d)} \in \Re^n$ of the "dense" lattice on the basis of the true codebook vectors $m_i'^{(s)}, m_j'^{(s)}$, and $m_k'^{(s)} \in \Re^n$ of the "sparse" lattice, respectively.

In practice, in order that the linear estimate be most accurate, we may stipulate that the respective indices h, i, j, and k are such that in the ideal lattice $m_i^{(s)}, m_j^{(s)}$, and $m_k^{(s)}$ are the three codebook vectors *closest* to $m_h^{(d)}$ in the signal space (but not on the same line). With α_h and β_h solved from (3.76) for each node h separately we obtain the wanted interpolation-extrapolation formula as

$$\hat{m}_h'^{(d)} = \alpha_h m_i'^{(s)} + \beta_h m_j'^{(s)} + (1 - \alpha_h - \beta_h) m_k'^{(s)} . \tag{3.77}$$

Notice that the indices h, i, j, and k refer to *topologically identical lattice points* in (3.76) and (3.77). The interpolation-extrapolation coefficients for two-dimensional lattices depend on their topology and the neighborhood function used in the last phase of learning. For the "sparse" and the "dense" lattice, respectively, we should first compute the ideal codebook vector values in analogy with Fig. 3.34. As the closed solutions for the reference lattices may be very difficult to obtain, the asymptotic codebook vector values may be approximated by simulation. If the ratio of the horizontal vs. vertical dimension of the lattice is $H : V$, we may assume tentatively that two-dimensional input vectors are selected at random from a uniform, rectangular pdf, the width of which in the horizontal direction is H and the vertical width of which is V. These inputs are then used to train two-dimensional SOMs that approximate the ideal two-dimensional lattices.

Initialization of the Pointers for a Larger Map. When the size (number of grid nodes) of the maps is increased stepwise during learning using the estimation procedure, the initial pointers for all data vectors after each increase can be estimated quickly by utilizing the formula that was used in increasing the map size, equation (3.76). The winner is the map unit for which the inner product with the data vector is the largest, and so the inner products can be computed rapidly using the expression

$$x^T m_h^{(d)} = \alpha_h x^T m_i^{(s)} + \beta_h x^T m_j^{(s)} + (1 - \alpha_h - \beta_h) x^T m_k^{(s)} . \tag{3.78}$$

Expression (3.78) can be interpreted as the inner product between two *three-dimensional* vectors, $[\alpha_h; \beta_h; (1 - \alpha_h - \beta_h)]^T$ and $[x^T m_i^{(s)}; x^T m_j^{(s)}; x^T m_k^{(s)}]^T$, *irrespective of the dimensionality of* x. If necessary, the winner search can still be speeded up by restricting the winner search to the area of the dense map that corresponds to the neighborhood of the winner on the sparse map.

This is especially fast if only a subset (albeit a subset that covers the whole map) of all the possible triplets (i, j, k) is allowed in (3.76) and (3.78).

3.17.3 Smoothing

It may not always be desirable or possible to use an excessive computing time just to guarantee very high accuracy or correctness of the asymptotic state of the map. Consider, for instance, that the number of available training samples is so small that they anyway do not approximate the pdf well enough. However, as the smooth form of the map is necessary for good resolution in comparing matching at the adjacent units, one might rather stop the training when there are still appreciable statistical errors in the codebook vectors, and apply a *smoothing* procedure to achieve this fine resolution.

Smoothing and *relaxation* are related concepts, although with quite different scopes. In smoothing one aims at the reduction of stochastic variations, whereas in relaxation a typical task is to solve a differential equation by difference approximations. An example of the latter is the *Dirichlet problem*, computation of the values of a harmonic function inside a domain, when the boundary conditions have been given.

Smoothing also differs from relaxation because it is usually only applied a few times differentially; in relaxation problems the "smoothing" steps are iterated until the asymptotic state can be regarded as stable, whereby the final state may look very different from the initial one.

Consider, for example, that we stop the SOM algorithm when the codebook vectors have achieved the values m_h'. A smoothing step looks similar to (3.76) and (3.77), except that the lattice is now the same: the new value of a codebook vector is obtained from, say, three old codebook vectors that are closest to the one to be smoothed in the two-dimensional signal space of the ideal lattice, but do not lie on the same straight line. In analogy with (3.76) and (3.77), let m_h (of the ideal lattice) first be expressed as the linear combination

$$m_h = \gamma_h m_i + \delta_h m_j + (1 - \gamma_h - \delta_h) m_k . \tag{3.79}$$

From this vector equation, γ_h and δ_h can be solved. The smoothed value of m_h' in the corresponding true SOM lattice with arbitrary pdf reads

$$S(m_h') = \varepsilon[\gamma_h m_i' + \delta_h m_j' + (1 - \gamma_h - \delta_h) m_k'] + (1 - \varepsilon) m_h' , \tag{3.80}$$

where the degree of smoothing has been further *moderated* using the factor ε, $0 < \varepsilon < 1$.

This scheme is similar for units both in the inside and at the edges of the SOM lattice. In smoothing, the new values of all codebook vectors must first be computed on the basis of the old ones and buffered, whereafter they are made to replace the old values simultaneously.

3.17.4 Combination of Smoothing, Lattice Growing, and SOM Algorithm

It now seems to be the most reasonable strategy to first perform an *equal* number of *identical* smoothing steps *for both the ideal and the real lattice*, respectively, after which (3.76), (3.77), and (3.80) are applied:

$$m_h^{(d)} = \alpha_h S^k(m_i^{(s)}) + \beta_h S^k(m_j^{(s)}) + (1 - \alpha_h - \beta_h) S^k(m_k^{(s)}) , \quad (3.81)$$

$$\hat{m}_h'^{(d)} = \alpha_h S^k(m_i'^{(s)}) + \beta_h S^k(m_j'^{(s)}) + (1 - \alpha_h - \beta_h) S^k(m_k'^{(s)}) , \quad (3.82)$$

where $S^k(\cdot)$ means k successive smoothing operations.

Finally, some fine-tuning steps of the map can be made with the SOM, eventually using the shortcut winner search at its later cycles.

Comment. The results of this Sect. 3.17 have been utilized in the application to be described in Sect. 7.8, whereby very large SOMs are computed.

4. Physiological Interpretation of SOM

It may be recalled (Sect. 2.15) that three types of neuronal organization can be called "brain maps": sets of feature-sensitive cells, ordered projections between neuronal layers, and ordered anatomical maps of *abstract features*. The latter reflect the most central properties of the organism's experiences and other occurrences. Such feature maps are probably learned in a process that involves parallel input to neurons in a brain area and adaptation of neurons in the neighborhood of the cells that respond most strongly to this input.

Since it might take days, weeks, months, or years for the biological SOM processes in neural realms to form or alter neural maps, the organizing effects are not immediately measurable and may be disguised by other, stronger changes in short-term signal activities. Changes in the m_i and their effects on signal transmission occur gradually, probably only intermittently during short periods of "learning" when some specific plasticity-control factor is on.

Nonetheless it has now become possible to demonstrate theoretically that a SOM process is implementable by quite usual biological components, and there may even exist many candidates for the latter. The purpose of this chapter is to devise general principles according to which physiological SOM models may be built. It is based on [2.45].

4.1 Conditions for Abstract Feature Maps in the Brain

It were unrealistic to expect that maps as clearly organized as in the simulation studies should exist in the brain. The biological sensory preprocessing is already much more complex than that used in simulations. Nonetheless there exist many kinds of feature maps in the brain, either orderly representations of the skin, retina, or cochlea, or more abstract maps. For instance, there are acoustic maps associated with sound to which the organism is most frequently exposed to. The tonotopic maps are known to exist both in the auditory pathways and in the areas of hearing on the cortex [4.1, 2] They are often thought to represent the order of the acoustic resonances on the basilar membrane of the inner ear, and preservation of the same order in the pathways originating from the hair cells and ending up at the cortex. Nonetheless