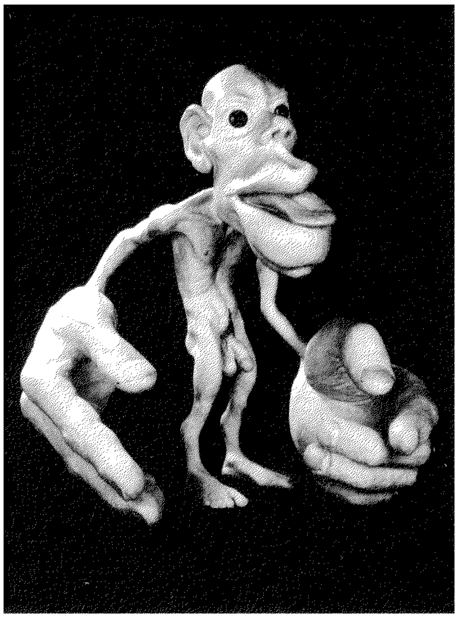
Utilizing the Topology Preserving Property of Self-Organizing Maps for Classification

Peter van der Putten
Cognitive Artifical Intelligence
Utrecht University
March 14, 1996



Universiteit Utrecht

Supervisors:
Prof.dr. Joost N. Kok
Dr. Marc Bezem
Drs. S. Haring

Abstract

The Kohonen Self-Organizing Map (SOM) is a popular algorithm for constructing a nearest neighbor codebook in pattern space. The algorithm utilizes a predefined ordering on the codebook to distribute the codes proportionally on the input manifold. In the end this ordering should reflect the structure of the input. Prototypical application of the SOM uses the codebook but neglects the ordering. We explore the practical possibilities for taking advantage of the ordering, concentrating mainly on classification tasks. We present three approaches: coding class boundaries with a duo of SOMs, construction of radial basis function networks with ordering information and using a SOM as a preprocessor for backpropagation networks. We obtain positive results on a number of real world data sets from the field of medical diagnosis, speech– and image processing. From this we conclude the ordering property of SOMs contains useful information. However, it is still unclear how to profit from it in the best possible way.

Contents

Chapter 1

Introduction and Motivation

Why use a Kohonen Self-Organizing Map (SOM, [Kohonen, 1982b])?
Researchers interested in using neural networks for neurobiological modelling will stress the similarity between SOMs and ordered neural structures like the somatosensory cortex. For other applications however the only thing that counts is an objective scoring of the performance on a certain task, for example the efficiency of clustering or the speed of classification. If used for these kind of tasks the SOM reduces to nothing more than a simple look up table (also: codebook). When a new input is presented all codebook vectors (codes) are compared to find the one most similar to the input. The values of the codes are calculated using the predefined ordering on the codebook: in a trained SOM codes that are next to each other in the codebook come to represent similar values (see the next chapter for an introductory discussion of the SOM).
A previous survey showed that although ordering is necessary for SOM codebook formation it is hardly used in SOM applications [van der Putten, 1994]. Because the ordering is a distinctive feature over other look up algorithms (also called nearest neighbor algorithms) and we think it contains useful information we want exploit this feature more. So the main aim of our research becomes:

Investigate if the ordering property of SOMs can be utilized to improve performance.

We define utilization of the ordering property as any use of ordering information other than in normal SOM training. The effect of ordering on quality and speed in normal SOM training is shown in a number of experiments performed by Ritter et al [Ritter et al., 1992].
Our central theme has an impact on both the utility of the ordering property for the SOM as on the utility of the SOM as a neural algorithm. If utilization of the ordering property leads to better results compared to standard SOM performance ordering will be seen as an important property of a SOM rather than just a by product. And if using ordering information results in better outcomes compared to other nearest neighbor algorithms the choice for a SOM will be confirmed.
As an approach to our subject we developed various variants of the SOM algorithm which take the ordering information into account. We ran a number of experiments to compare the objective performance of our architectures to the basic SOM and other algorithms. The task we chose our algorithm to perform was classification because it is the most basic task in pattern recognition and a common application for the SOM. All experiments were run on data taken from real world problems to produce as realistic results as possible.
This paper starts with a description of the the SOM algorithm as a nearest neighbor algorithm, without reference to neural network concepts that may work confusing (chapter 2). In this chapter we will also discuss and give definitions of the ordering property and show some SOM applications. Readers familiar with the SOM can skip this chapter. In chapter 3 we will give a short discussion on classification and the limits of classification error. The next three chapters describe three different (groups of) SOM variants which we developed and the experiments we performed with them. Chapter 4 discusses coding class boundaries in pattern space with a second SOM, in

chapter 5 we construct radial basis function networks with SOMs and in chapter 6 we use a SOM as a preprocessor for a backpropagation network. These chapters are followed by a discussion (chapter 7) and a conclusion (chapter 8). The paper ends with two appendices describing the data (appendix A) and the software (appendix B) used in the experiments.

Acknowledgements This research was done as a final degree project to obtain a 'doctorandus' degree in Cognitive Artificial Intelligence. I would like to thank my supervisors prof.dr. Joost N. Kok from the computer science department of Leiden University, dr. Marc Bezem from the philosophy department of Utrecht University and drs. Sebastiaan Haring from the computer science department of Utrecht University. Furthermore I would like to thank Jakub Zavrel for his comments and Michiel van Wezel for supplying me with backpropagation source code.

Chapter 2

The Kohonen algorithm

2.1 SOM architecture and dynamics

Although the Kohonen Self-Organizing Map (SOM, [Kohonen, 1982b, Kohonen, 1995]) is generally seen as a neural network, its workings can be readily understood without the usual neural network concepts like nodes, links, weights, propagation and activation.

Actually, in prototypical application a SOM is nothing more or less than a *codebook*.
Let's assume we want to model a certain amount of data consisting of records, all containing the same fields. We can see this data as a set of vectors $x \in X$ with $X \subset \mathcal{R}^D$ and size n; D equals the number of fields, n equals the number of records. We call x an input vector, X the input manifold and $\mathcal{R}^D$ a pattern space. Now, in prototypical application a SOM reduces to nothing more or less than a sort of look up table, a *codebook*. A codebook consists of a finite set C of vectors $m \in \mathcal{R}^D$ ('codes', or 'reference vectors'). Every new input is mapped to the nearest code. In this way the codebook divides pattern space in a so-called Voronoi tessellation (see figure 2.1). In more formal notation:

Definition 2.1.1 (Codebook, Nearest Neighbor) *Let X be a set of observations $x \in \mathcal{R}^D$. Let C be a finite set of n code vectors $m_i \in \mathcal{R}^D$, $i = 1 \ldots n$. We call C a codebook. Let d be a distance metric defined on $\mathcal{R}^D$. Then m_c is the nearest neighbor to x iff $d(x, m_c) = \min_1^n d(x, m_i)$.*

The distance metric d is usually the Euclidean metric $d_{eucl}(x, y) = ||x - y||$ but it can in fact be any metric. For instance it can be the Manhattan Block Distance $d_{block}(x, y) = \sum_1^D |x_i - y_i|$ which is easily implementable in hardware, the negative of the inner product $-d(x, y) = -x \cdot y$ that is more suited to a neural network implementation or just any other distance measure defined on $\mathcal{R}^D$.
The prototypical function of the codebook is to divide pattern space in such way that the probability density of the input is reflected. The quality of the codebook could be measured by testing if the number of input vectors mapped to each code is equal, or that the mean distance of codes to their nearest neighbors is minimized etc. These criteria are called clustering criteria. The basic task a codebook performs is called clustering or coding, dividing pattern space into disjoint parts, with respect to certain clustering criteria. We call this the *clustering property* of a SOM.
The key problem, of course, is how to form such a codebook. The Kohonen Algorithm defines such a codebook formation procedure.
First some ordering L is defined on codebook C. On the presentation of an input vector x the nearest neighbor m_c is moved a little bit in the direction of x. Moreover, those codes that are within a certain range from m_c *within* L are also allowed to update their value. In the end this ordering should reflect the structure of the input, in other words codes that are next to each other in L become to represent values that are near on X. This property is also called the *ordering*

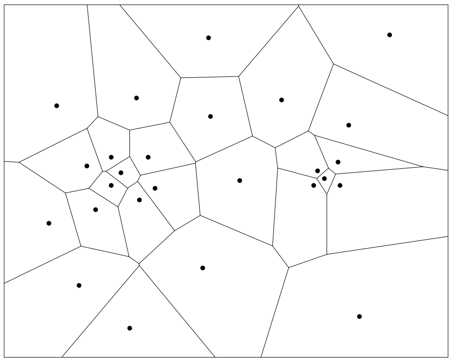


Figure 2.1: Voronoi tessellation. A codebook divides pattern space up into disjoint Voronoi cells. All points in a Voronoi cell are closer to the code in the center of the cell than to every other code.

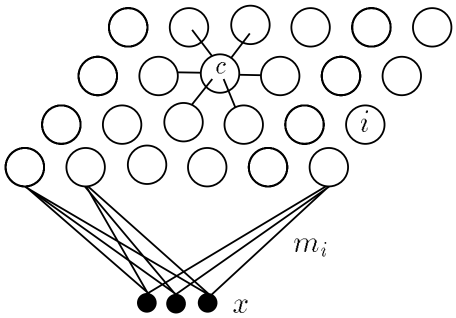


Figure 2.2: The SOM as a neural network. It is rendered as a lattice of units and edges. Units represent code vectors, edges between units represent the ordering relation.

property or the *topology preserving property.*

To simplify computation of distance within L we assign a position $l_i \in \mathcal{Z}^E$ to each code $m_i \in C$, $i = 1 \ldots n$. Ordering L then becomes a set of pairs (l_i, l_j) with $i, j \in \{1 \ldots n\}$; m_i is a direct neighbor of m_j in L iff $(l_i, l_j) \in L$. We call $\mathcal{Z}^E$ a lattice space. Usually L is a regular $L_1 \times L_2$ ordering, with for example

$$L = \{([0,0],[1,0]),([1,0],[0,0]),([0,0],[0,1]),([0,1],[0,0]),\ldots,([L_1,L_2],[L_1,L_2-1])\}.$$

In this case we can take Euclidian distance in lattice space $\mathcal{Z}^2$ as the distance function defined on L. One can define any other ordering or distance measure if necessary. Now we can describe the updating steps in more detail:

Definition 2.1.2 (SOM updating rules) *Let $X \subseteq \mathcal{R}^D$ be a set of observations. Let M be a pair (C, L) with C a codebook of size n and L a set of pairs (l_i, l_j) with $l_i, l_j \in \mathcal{Z}^E$, $i,j \in \{1 \ldots n\}$. We call M a SOM. Let $m_c \in C$ be the nearest neighbor for a certain $x \in X$. Then codebook C is changed according to the following adaptation rules:*

$$m_i(t+1) = m_i(t) + \alpha h(c,i)(x(t) - m_i(t))$$

with $0 \le \alpha, h(c,i) \le 1$, α the learning speed and $h(c,i)$ a neighborhood function.

The neighborhood function $h(c,i)$ defines neighborhood shape. The most simple form is that of a block function:

$$h_\delta(c,i) = \begin{cases} 1 & \text{if } m_i \in \mathcal{N}_c \\ 0 & \text{otherwise} \end{cases}$$

with $\mathcal{N}_c$ a set of codes within certain distance δ to c in ordering L:

$$\mathcal{N}_c = \{m_i \in C \mid d(l_i, l_c) \le \delta\}$$

with d a distance function defined on L.
The use of a block function implies that codes $m_i \in \mathcal{N}_c$, $i \neq c$ move more towards x than m_c does. A popular neighborhood function used to correct this effect is the Gaussian 'bubble' function:

$$h_\sigma(c,i) = e^{\frac{-\|l_c - l_i\|^2}{2\sigma^2}}$$

We see that for scale $\sigma \downarrow 0 : h_\sigma(c,i) = h_\delta(c,i)$ with $\delta = 0$. Prototypically, SOM training is divided in two phases. In the ordering phase, initial neighborhood size (δ, σ) and learning speed (α) are large; these values decrease over time. In the second phase the map is finetuned with small α and δ, σ.

Let's illustrate all these definitions with a simulation example (figure 2.3). Assume input manifold X consists of a Gaussian input distribution in two dimensional pattern space $\mathcal{R}^D$. We construct a SOM with a two dimensional regular rectangular ordering L of 10×10 codes (each code has eight neighbors). The codebook is drawn in pattern space; every code is connected to its direct neighbor in L.
At first the codes are initialized randomly in a rectangular area and the ordering seems to make no sense. Then in the ordering phase the map starts to unfold. When this is finished the ordering reflects the global structure of the input. Now neighborhood effects are reduced and the codebook is slowly finetuned to the optimal values. In the end all codes cover the circle shaped input, with more codes in the center than at the edges. So the clustering property is fulfilled. All codes are connected in the right order too. So also the ordering property is fulfilled.

The SOM can also be seen as a neural network (figure 2.2). Usually a SOM is represented as a regular two dimensional lattice of code units. With each code unit a code vector is associated. Assuming the inner product is used as a distance measure (input patterns must be normalized to constant length) we can define an input layer that is totally connected to each code unit. For one code unit, the weights on these connections constitute its code vector. The ordering L defines the neighborhood relations; in this example we see that there are hexagonal ordering relations.
SOM dynamics can also be expressed within this framework. Inputs are coded as activity patterns on the input layer. Activations are propagated to the code units through the weighted connections. Each code units computes its activation by applying a transfer function to the incoming activation. Lateral inhibitory connections between units then extinguish activation so that one 'bubble' of activation remains. The unit with maximum activation (the nearest neighbor) is called the winner or best matching unit. In the adaptation step, Hebbian learning changes the weights relative to the level of activation of the units.

The main aim of our study is to investigate whether the ordering property of SOMs can be utilized to improve performance. It is not possible to check objectively whether some application assumes the ordering property to hold. Therefore we say that any application that refers to ordering information other than in normal SOM training, is utilizing the ordering property. Using our description of the SOM this can be rewritten more specificly: any algorithm that contains a reference to a lattice position l_i apart from the normal SOM updating rules described in definition 2.1.2, is utilizing the ordering property.

2.2 Related Algorithms

The SOM has two different sources of inspiration: clustering & dimension reduction algorithms from classical pattern recognition and neurobiological modelling.

Its most similar predecessor is the k-means clustering algorithm ([Loyd, 1957, Linde et al., 1980]), which operates as a SOM without ordering. The basic steps are:

1. Initialize the codebook to the first k training samples.
2. For each code compute the mean of all training samples for which it is nearest neighbor.
3. Assign this mean to each code and repeat the previous step a few times.

Instead of this batch version there was also an early adaptive approach which, given a certain input, shifts the mean of the nearest neighbor in the direction of the input by a small amount

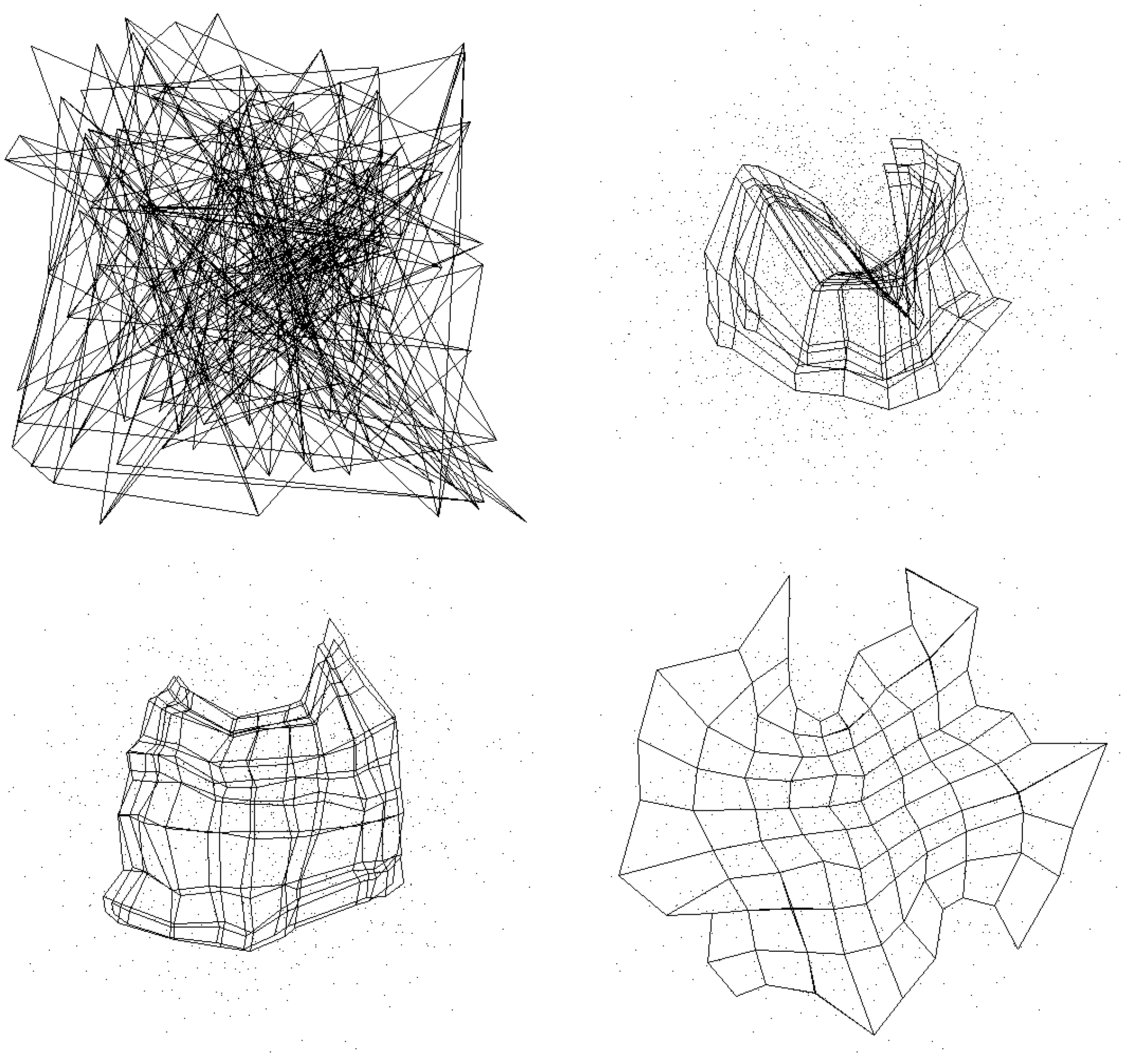


Figure 2.3: Simulation example: 10×10 SOM learns 2 dimensional Gaussian distribution with linearly decreasing neighborhood and learning speed and block neighborhood function. Different stages in ordering are shown: random initialization (0 input presentations), ordering phase (1240 input presentations), ordering finalized (2340 input presentations) and finetuning finalized (15270 input presentations).

[MacQueen, 1967, Murtagh and Hernández-Pajares, 1995]; this corresponds to SOM adaptation with neighborhood 0. The most serious disadvantage of k-means clustering, which is partly solved by the SOM, is that if one code covers an abnormally large portion of the input manifold then it will be the nearest neighbor for practically all the inputs and block the distribution of the other codes. Neighborhood training in the SOM ensures that groups of codes will be pulled in the direction of areas where input is located. Other common codebook algorithms are Simple Nearest Neighbor [Cover and Hart, 1967] in which all train data is used as a codebook (see also section 3) and Learning Vector Quantization, a non-topological, supervised variant of the SOM [Kohonen, 1990, Kohonen, 1989]. A group of techniques which also construct an ordering on the clusters is hierarchical clustering. Clusters are found iteratively by merging clusters that have minimal distance or by splitting clusters in parts which have maximal distance. So a by-product of these algorithms is that taxonomic ordering evolves on all the clusters formed in the process [Kohonen, 1995].
Another group of related algorithms are the so-called dimension reduction algorithms. The SOM can be considered as an algorithm that maps a pattern space $\mathcal{R}^D$ to lattice space $\mathcal{Z}^E$ which usually has a lower dimension. A number of classical algorithms performs the same task: a.o. non-metric multi dimensional scaling [Shepard, 1962], Sammon's mapping [Sammon, 1969], the Elastic Net [Durbin and Willshaw, 1987] and Minimal Wiring [Durbin and Mitchison, 1990]. These algorithms have in common that they assume the mapping to be bijective, in other words no clustering is performed [Goodhill et al., 1995]. So they only share the ordering property with SOMs and not the clustering property.

Another source of inspiration were algorithms designed for neurobiological modelling. In animal and human cortex a number of different topological maps are found [Knudsen et al., 1987]. A vast collection of computational models was developed to give an account for the formation of such maps, of which the study of Whillshaw and von der Malsburg became classical [Whillshaw and von der Malsburg, 1976, Ritter et al., 1992]. The authors define local chemical mechanisms which lead to topological correspondence between two neural layers. In another classical theoretical study [Amari, 1980] a system was developed based on a continuous neural sheet instead of a discrete one and subjected his algorithm to thorough mathematical investigation. Finally a number of biologically plausible variants of the SOM algorithm were developed one of which [Sirosh and Miikkulainen, 1994] is based on totally local algorithms (for nearest neighbor search, adaptation, neighborhood selection) and emerging neighborhood relations.

2.3 A closer look at the ordering property

As we saw in the previous section ordering L should ultimately correspond to the ordering relations in input manifold X. This was called the *topology preserving property* or the *ordering property*. At present there is no agreed mathematical definition for perfect topology preservation nor is there a common evaluation measure to quantify the topology preserving quality of a certain mapping. In this section we will give some illustrative examples of approaches to these problems and give a practical evaluation measure of our own.

To illustrate the problems that can arise we will suggest a definition for perfect topology preservation in any dimension.
For simplicity let us assume that input X is not a discrete set of points but a continuous manifold in pattern space $\mathcal{R}^D$. Perfect topology preservation of a certain map will satisfy two conditions: codes that are direct neighbor in the ordering should correspond to close points in pattern space and vice versa. To translate this into more mathematical terms we first give a definition of a Voronoi cell:

Definition 2.3.1 (Voronoi cell) *Let $m_i \in C$ be a code. Its Voronoi cell V_i is defined as:*

$$V_i = \{\ x \in \mathcal{R}^D \ \big|\ \forall m_j \in C : \quad \|x - m_i\| \leq \|x - m_j\|\ \}$$

Note that in this definition the Voronoi tessellation is not longer a *disjunctive* division of pattern space! Adjacent Voronoi cells will share their borders. Now our definition of perfect topology preservation will be (adapted from [Martinetz, 1993, Martinetz and Schulten, 1994]):

Definition 2.3.2 (Perfect Topology Preservation) *A map M=(C,L) with input manifold X is preserving topology perfectly if the following conditions hold:*

1. $\forall m_i, m_j \in C$: *if code m_i is a direct neighbor of m_j in ordering L ($\Leftrightarrow (l_i, l_j) \in L$) then V_i is adjacent to V_j ($\Leftrightarrow V_i \cap V_j \neq \emptyset$).*
2. $\forall m_i, m_j \in C$: *if V_i is adjacent to V_j ($\Leftrightarrow V_i \cap V_j \neq \emptyset$) and $i \neq j$ then m_i is a direct neighbor of m_j in L ($\Leftrightarrow (l_i, l_j) \in L$).*

Let's discuss these two conditions one by one.
The first condition is directly enforced by the Kohonen algorithm. Because codes neighboring in ordering L are allowed to move to a certain input x if they are part of the neighborhood of the nearest neighbor code, these codes will move towards each other in pattern space. [1] This condition is violated for instance when a SOM is twisted (see figure 2.5): local topology is mainly preserved, but global topology is violated. The crossing of the edges in the center of the indicates that the Voronoi tiles of these codes probably do not overlap.
The second condition, in contrary to common opinion, is *not* explicitly enforced by the SOM updating rules. Examples of SOMs violating this condition will occur frequently when trying to map a pattern space by a SOM of lower dimension (see figure 2.4), which is the case in almost all SOM applications.
Note that our definition is clearly restricted to local topology preservation, we do not want to add any conditions for preservation of global topology. We only refer to codes or Voronoi tiles that are direct neighbors. Any mapping that has a conflict in local topology preservation will necessarily fail global topology preservation. In the majority of cases the SOM will not be preserving topology globally because of local optimal in global organisation (figure 2.5) or dimension reduction (figure 2.4).

In general a certain mapping will not be preserving topology perfectly. Therefore, it is practically relevant to define a certain evaluation measure to quantify the quality of the map. Again there is no general consensus on this definition.
One of the most straightforward definitions is valid for a certain subclass of SOMs, those with linear ordering L and input manifold X lying in one dimensional pattern space $\mathcal{R}^D$: [2]

$$D = \sum_{i=2}^{n} (\mid m_i - m_{i-1} \mid) - \mid m_n - m_1 \mid$$

Trivially, if the codes are sorted (ascending or descending), the map is ordered and $D = 0$. This measure was used to prove convergence of ordering. In [Kohonen, 1982a] necessary and sufficient conditions are given to lower D (one dimensional $\mathcal{R}^D$, block neighborhood function h_{ci}). It was shown that there are more training steps that decrease D than there are that increase D. Furthermore it is proven that if $D = 0$, D will not be changed by subsequent learning. Under the same assumptions, the stronger result of convergence into an ordered state was proven [Cottrell and Fort, 1987, Kohonen, 1989]. The ordered states are shown to be absorbing states of a Markov Process. Furthermore, it is shown that there must be a sequence with non zero probability leading to an ordered state. Assuming random selection of inputs, this proves convergence into an ordered state [Kangas, 1994]. These proofs were simplified by rewriting ΔD in the standard formula for ellipsoids [Budinich and Taylor, 1995]:

[1] Consider a SOM with d Euclidean distance measure and block neighborhood, input x, nearest neighbor m_c, direct neighbor m_j; $d(m_c, x), d(m_j, x)$ will be lowered by factor α, angle between m_c,m_j is constant, so $d(m_c, m_j)$ will be lowered by factor α.

[2] In the following definitions assume $L = \{(l_1, l_2), (l_2, l_1), \ldots, (l_{n-1}, l_n), (l_n, l_{n-1})\}$

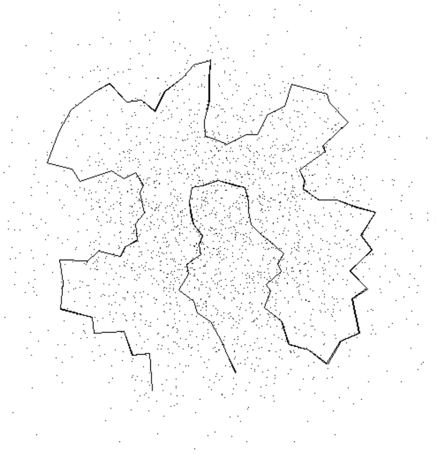


Figure 2.4: One dimensional SOM maps two dimensional Gaussian input. Both ends of the SOM are close in pattern space but distant in ordering L.

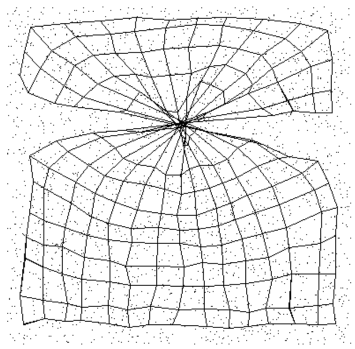


Figure 2.5: SOM (15×15) maps rectangular input. The majority of local topology is preserved, global topology is violated. In the center neighborhood edges cross which indicates the respective Voronoi tiles will not overlap.

$$\Delta D = \alpha(||x - F_1|| + ||x - F_2|| - 2a)$$

with foci $F1, F2$ and principal axis a:

$$F_1 = m_2 + \frac{m_1 - m_2}{\alpha}$$

$$F_2 = m_4 + \frac{m_5 - m_4}{\alpha}$$

$$2a = \frac{||m_2 - m1||}{\alpha} + ||m_3 - m_2|| + ||m_4 - m_3|| + \frac{||m_5 - m_4||}{\alpha}$$

D will only be lowered now if x is within the ellipsoid (see also figure 2.3). When the codes are sorted, the ellipsoid will shrink to a line segment. Only in one dimensional pattern space this segment has volume. So only in this case there is a probability that the algorithm converges. In higher dimensions there is still a possibility to converge: assume all codes are sorted and all new inputs fall on the border of the ellipse, so $D = 0$ and $\Delta D = 0$. Only the probability of this possibility is practically zero, especially when the data contains a minimal amount of noise.
A collection of measures that are valid for mappings with $\mathcal{R}^D$ and L of higher dimension are summarized and compared in [Goodhill et al., 1995]. These measures were designed for other algorithms than the SOM (a.o. Sammon's mapping, multi dimensional scaling, minimal wiring), so they were not used to prove SOM convergence. The advantage of the measure put forward by Goodhill is that an algorithm has been constructed that is proven to optimize this measure ([Goodhill et al., 1996]). It was proven that the SOM algorithm does not optimize some objective function [Ritter et al., 1992]. The disadvantage of the Goodhill algorithm is that it assumes mapping M to be bijective, in other words no clustering is performed.

For the purely practical purpose of this thesis we constructed a very simple and straightforward measure for topology preservation. In the previous section it was explained that a SOM is driven by two separate forces. One tries to spread the codes over the input manifold to reflect the probability density (the clustering property). The other tries to keep distances between codes neighbouring

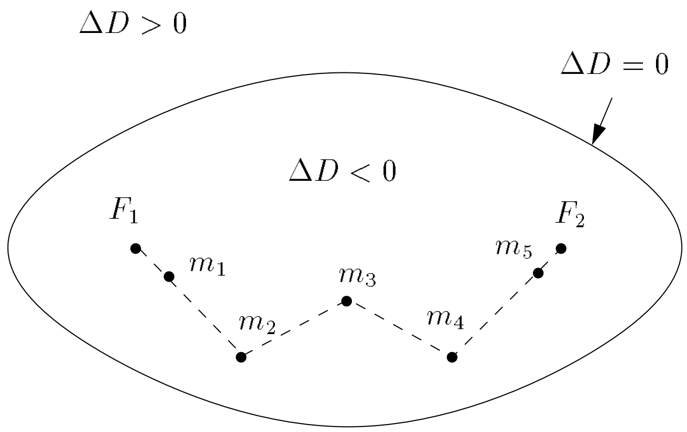


Figure 2.6: Convergence of ordering: only if a new input falls within this ellipsoid D will be lowered. Foci F_1, F_2 and axis a can be rewritten in code values.

in L low to enforce topology preservation. So it seems reasonable to express the ordering quality of a SOM in the mean distance between neighbours. For this we defined the topological error of a code:

Definition 2.3.3 (Topological Error) *The topological error of a code m_c is defined as the mean distance to its neighboring codes in L:*

$$E_{TOP}(m_c) = \frac{\sum_{(l_i,l_c)\in L} d(m_c, m_i)}{N}$$

with $(l_i, l_c) \in L \Leftrightarrow m_i$ is direct neighbor to m_c in L and N=number of direct neighboring codes in L.

In figure 2.7 we plotted mean topological error (over all SOM codes) and quantization error (mean Euclidian distance $d(x, m_c)$ over all training patterns) for our simulation example of figure 2.3. We see that quantization error decreases more or less monotonically. Topological error first decreases drastically during the ordering phase and increases slowly during finetune phase (when neighborhood size is practically zero).
This measure, though powerful through its simplicity, suffers from some disadvantages. Firstly, the absolute values for topological error are not comparable for codebooks of different size or input manifolds occupying different portions of pattern space. Secondly, we assume that the SOM performs normal clustering. A SOM that does not distribute the codebook over the input manifold but keeps the codes infinitely close to a single point will always score topological error approximating zero, whether the codes are ordered or not. Therefore we stress that topological error values should only be used as a measure to evaluate a single SOM during different stages of normal SOM training.

2.4 SOM Applications

SOMs can be applied for a whole range of tasks. In this section we will give some examples of the different applications. We will also focus on some studies in which we think the authors falsely claim that they utilize the ordering property. We try to make this clear by discussing these application using the non-neural concepts of our SOM description (section 2.1).

We see the following main applications of the SOM:

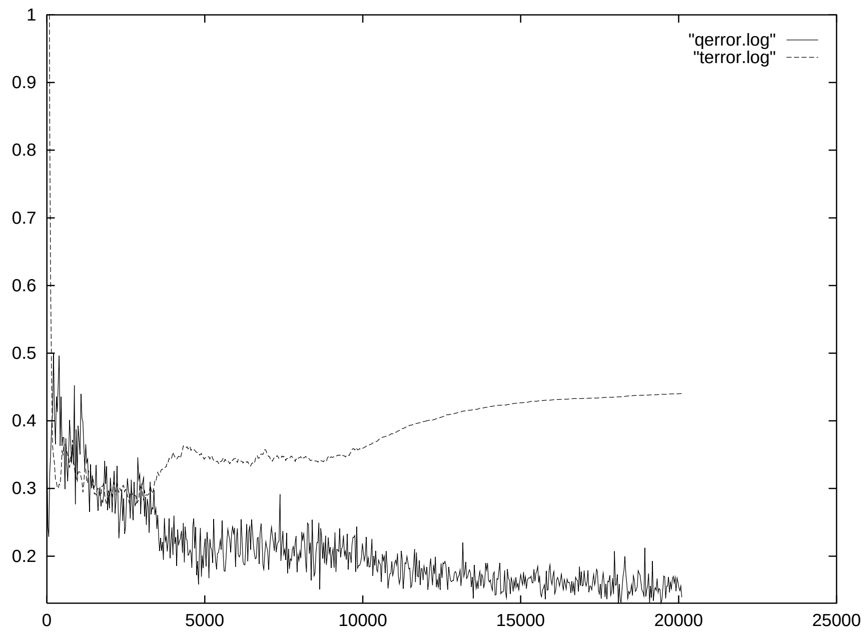


Figure 2.7: Topological error and quantization error over time for the simulation example in figure 2.3. Topological error decreases fast during ordering and increases slowly during finetuning; quantization error decreases more or less monotonically.

- **Clustering and Data Visualization** As described in section 2.1 the main task performed by the SOM is clustering. The existence of an ordering on the clusters can be useful for instance for signal processing. Consider a one dimensional SOM mapping scalar frequency values (see figure 2.8). If there is a shift of the signal in the frequency domain (e.g. all frequencies increase) the SOM will adapt very quickly because the neighborhood effects cause all the codes in the codebook to shift [Kohonen et al., 1990]. Also in data visualization, displaying ordering of clusters on screen can facilitate user interpretation of clusters described by multiple codes. An example is process control in which dangerous areas on the SOM represent dangerous process states that need to be avoided [Tryba and Goser, 1991].

- **Optimization** The SOM tries to find an optimal balance between the ordering property and the clustering property. The optimization power of the SOM can be used for instance to solve the Travelling Salesman Problem (TSP) [Angèniol et al., 1988]: assume we have a ring shaped SOM, and the two dimensional coordinates of cities to be visited constitute the input manifold. The Kohonen algorithm will now automatically find an approximation to the shortest route (see figure 2.9). If we map all possible places for e.g. a robot to be in with a two dimensional SOM we can solve route planning problems [Vleugels et al., 1993]. Obviously, if we can code these generic problems with a SOM other optimization problems can be coded straightforwardly as well (e.g. function approximation [Cherkassky and Lari-Najafi, 1990], chip design [Zhang and Mlynski, 1991]).

- **Classification and Sequential Processing** In a classification task there are multiple input manifolds in pattern space each of which belongs to a separate class (see chapter 3). A SOM is trained to the union of the manifolds and the codes are labelled to the class they respond best to. An example is Kohonen's classical 'phonetic typewriter', which was trained to spectral values of phonemes [Kohonen, 1989]. When a sequence of inputs is processed we enter the realm of sequential processing. In the example of the phonetic typewriter the

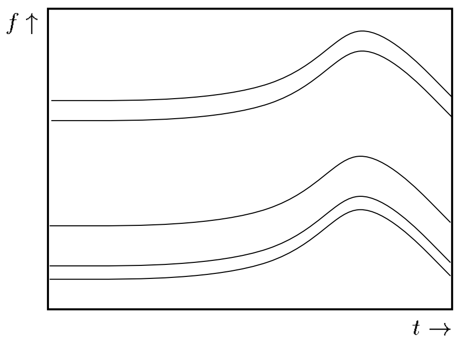


Figure 2.8: Vector Quantization. Linear SOM (5 codes) maps frequency values. A shift in the signal will lead to fast codebook adaptation because neighborhood adaptation affects multiple codes.

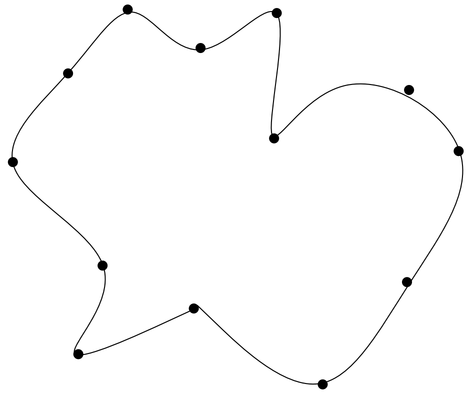


Figure 2.9: Travelling Salesman Problem. A ring-shaped SOM maps a distribution of 'cities'. Topology preserving property keeps route found as short as possible.

SOM codebook produced a sequence of phoneme class labels belonging to winning codes. We can however also put the ordering to use if we interpret these input sequences as paths through lattice space $\mathcal{Z}^E$ (see figure2.4), as is done in [Torkkola and Kokkonen, 1991] and [Kangas, 1994].

We do not only claim the ordering property is hardly utilized in SOM studies ([van der Putten, 1994]), but we can also give examples of studies in which the authors falsely claim (to our opinion) that they use the ordering property.
In the latest SOM book written by Kohonen himself [Kohonen, 1995] he describes the so-called supervised SOM. The input vector x consists of a signal (pattern) part x_s and a class part x_θ. First a codebook C is trained on the concatenation of x_s and x_θ. During recognition the nearest neighbor is found using only x_s; x_θ was not considered. Kohonen writes:

> The unsupervised SOM constructs a topology-preserving representation of the statistical distribution of all input data. The supervised SOM tunes this representation to discriminate better between pattern classes. [...] *This special supervised training was used in our original speech recognition system known as the "Phonetic Typewriter", and it indeed made use of the topological ordering of the SOM, contrary to newer architectures that are based on Learning Vector Quantization.* [emphasis placed by Kohonen]

To our opinion any codebook formation algorithm that would take $x_s + x_\theta$ as training input would show the behaviour described above. There is absolutely no reference to or use of ordering L!
Even if ordering L is referred to explicitly in a SOM study we have to be careful if the way ordering information is utilized is useful. An example is the Hierarchical SOM by Lampinen [Lampinen and Oja, 1992]. The authors use two maps to cluster an input manifold. The first SOM clusters the input manifold in the usual way. Every time a code m_c is a winner in the first SOM its lattice position l_c is sent through to the second SOM for clustering. In a HSOM every second map code will be nearest neighbor for a cluster of first map lattice positions and, indirectly, nearest neighbor for a conjunction of Voronoi cells. A single Voronoi cell is always convex (see figure 2.1) but in a HSOM the units in the second map can represent non convex clusters of Voronoi cells on the input manifold. So information from lattice space $\mathcal{Z}^E$ is used, and because the second map codes are to correspond to adjoining clusters, it is assumed that the first map lattice positions have some meaning, some relation to the input manifold. In other words it is assumed that the ordering property holds.

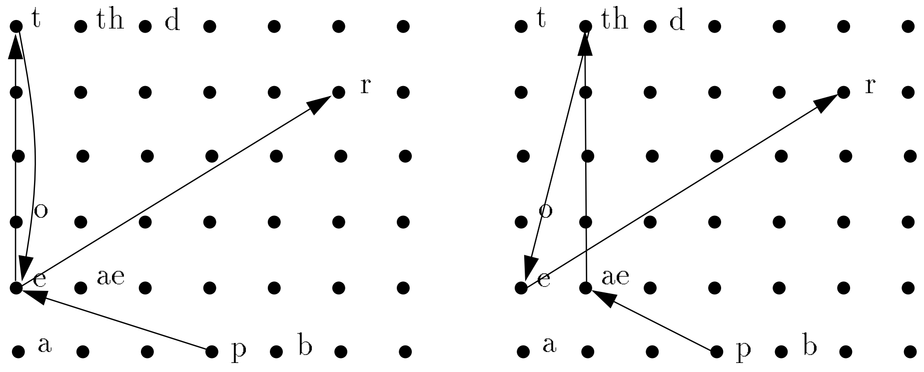


Figure 2.10: Phonetic Map. SOM (7×6) maps phoneme space. Codes are labelled to phoneme class they respond best to. Adjacent codes correspond to similar phonemes. Assume we present the network with the word 'Peter' from two different speakers. The class label outputs 'ppeeetter' and 'ppaeaeaeththter' are largely dissimilar; paths through lattice space have similar form.

On the other hand, we know that (i) because of the clustering property every first SOM code will approximately be nearest neighbor for the same number of inputs and (ii) the order in which inputs are presented to a SOM does not matter. From this we can conclude that if we know the ordering dimensions of the first SOM (say 10×10) we can train the second SOM with two nested loops in which we present all possible first map lattice positions ((0,0)...(9,9)), independent of the content of X. From this we can deduce that the only thing the second map learns is regular division of map one lattice in pieces of the same size. But this task is trivial, and could be done a priori. So we use the ordering information to perform a trivial task.
Hopefully we have shown now that we we are a bit sceptical about the real utilization of ordering information in previous SOM applications and the utility of preserving ordering in general. Note however, that we do not make any claim about the influence of neighborhood cooperation on the quality and speed of normal SOM codebook formation (for a range of experiments on this matter see [Ritter et al., 1992]).

Chapter 3

Classification

3.1 Introduction

Classification is one of the foremost tasks performed within neural computation. To perform classification we assume that our training data X is divided in different groups or classes, which may or may not correspond to clusters of points in X. When a new pattern is input to the classifier it should output the class it belongs to.
Consider the example of classifier for breast cancer. A new patient can be described by a number of features (personal details, lab exams). Based on the archive with descriptions and diagnosis of previous patients the classifier should give an advice whether the breast tumor is malignant or benign.
In this chapter we will show that there is a lower bound on classification error. Next we discuss nearest neighbor classification, give an upper bound for nearest neighbor classification error and show the relation to the SOM.

3.2 Bayes Error

It is intuitively clear that given a number of overlapping classes in input space we will never achieve 100% correct classification. Theoretically, we are able to compute a lower bound on the classification error, the so-called Bayes error. The idea behind this is simple.
Consider a multi class problem with classes $\theta_1 \ldots \theta_m$ and observation variable x (see figure 3.2 for an example). Let's assume we know the probability density function $p(x|\theta_i)$, which defines the relative frequencies with which observations x occur for a certain class θ_i. We also know the a priori probability $P(\theta_i)$. We want to know the a posteriori probability $P(\theta_i|x)$: the probability that a certain observation x belongs to class θ_i. This can be calculated using the Bayes rule:

$$P(\theta_i|x) = \frac{p(x|\theta_i)P(\theta_i)}{p_{all}(x)}$$

with

$$p_{all}(x) = \sum p(x|\theta_i)P(\theta_i)$$

We can define a cost function $C(\theta_i, \theta_j)$, which defines the cost of making classification θ_i when it should be θ_j. Consider for instance the example of the breast tumor classifier with θ_1 =benign tumor and θ_2=malignant tumor. In this case $C(\theta_1, \theta_2)$ will be much larger than $C(\theta_2, \theta_1)$. Now the expected total loss for classifying x as θ_k can be defined as:

$$Loss(x, \theta_k) = \sum_{i=1}^{m} P(\theta_i|x)C(\theta_k, \theta_i) = \frac{p(x|\theta_i)P(\theta_i)}{p_{all}(x)}C(\theta_k, \theta_i)$$

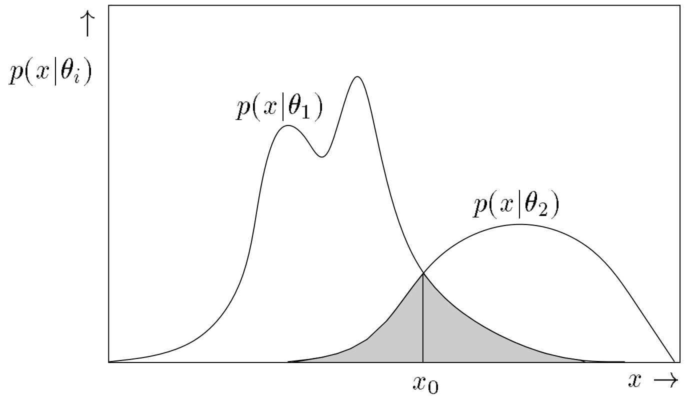


Figure 3.1: Two class problem: x is observant, ρ_{θ_1} and ρ_{θ_2} are a priori distributions of classes θ_1 and θ_2, $C(\theta_i, \theta_j) = 0$ if $i = j$, $C(\theta_i, \theta_j) = 1$ otherwise, $P(\Theta_1) = P(\Theta_2)$. Then x_0 will be decision boundary, shaded area will correspond to Bayes error.

Let's assume that all costs for misclassification are equal and that the a priori class distributions $P(\Theta_i)$ are equal. Then to minimize loss we should classify observation x as the class θ_i with highest a posteriori probability $P(\theta_i|x)$. In the case of figure 3.2 you would choose class θ_1 for values $x < x_0$ and θ_2 for $x > x_0$. We say that x_0 demarcates a *decision boundary*.
For a certain observation vector, the probability of error are the a posteriori probabilities of the other class. Now we see that over the total range of observations the probability of error corresponds to the shaded area in figure 3.2. This is precisely the Bayes Error. It can be theoretically proven that this will be the lowest possible error.

3.3 Nearest Neighbor Algorithms

The problem is that generally we do not know $p(x|\theta_i)$, so then we cannot compute the optimal decision boundaries that would yield the lowest possible error. In most cases we only have some small finite size set X of observations x_i and a corresponding set Θ of class labels θ_i. We could estimate $p(x|\theta_i)$ by fitting some appropriate distribution over our data. But then we would have made assumptions about the shape of the distributions. We could also resort to non-parametric algorithms; algorithms that don't make assumptions about the structure of the input manifolds. Probably the most straightforward approach is to classify an unknown input x as the class of its nearest neighbor in X. Note that now the whole of input data X is considered to be the codebook C. In more formal notation:

Definition 3.3.1 (1-Nearest Neighbor Rule) *Let codebook $C = m_1 \ldots m_n, m_i \in \mathcal{R}^D$ be a codebook, $\Theta = \theta_1 \ldots \theta_m$ be a set of class labels.*
Let l be a function $l(m_i) = \theta_j$ which assigns a class label to each code. Let $x \in \mathcal{R}^D$ be a new input. Then iff m_c is nearest neighbor to x in C, classify x as class $l(m_c)$.

Definition 3.3.1 can easily be extended to the k-NN rule: this rule produces the label occurring most frequently among the k nearest neighbors.
The most important proof on the 1-NN rule sets an upper bound on the probability of error:

Theorem 3.3.1 (Upper Bound on NN error) *Let R be NN probability of error, R^* Bayes probability of error. Let M be the number of classes, $M \geq 2$. Then for NN codebook size approximating infinity, $R^* \leq R \leq 2R^* - MR^{*2}/(M-1)$. [Cover and Hart, 1967]*

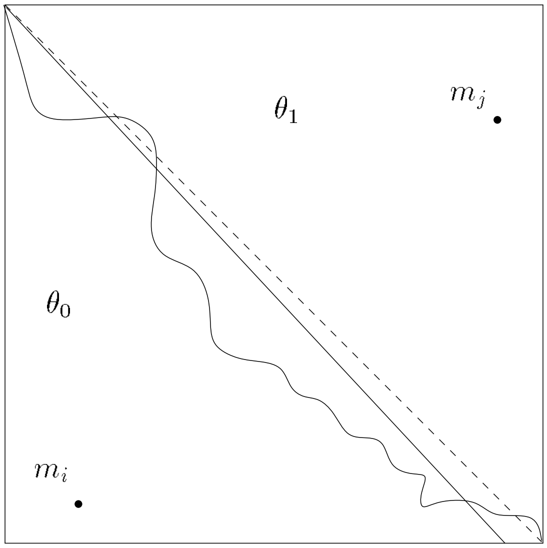


Figure 3.2: Generalization. Assume straight line is codebook decision boundary (in 2d pattern space), dotted line is Bayes decision boundary and curve defines Simple Nearest Neighbor decision boundary. If train set is unrepresentative the codebook performs useful generalization.

From this we can conclude that for any number of classes and for infinite number of samples simple 1-NN can not score worse than two times Bayes error. [1] There will not be *any* algorithm that will perform better than half NN error!

We discussed the bayes error as a theoretical lower bound on probability of error for classification. In most cases we do not know the (theoretical) Bayes error. However, if we assume our input set X is large enough (this is a dangerous assumption!) we can compute an approximation to NN error. Complexity of this computation will grow quadratically with size of set X. This is were the SOM comes into play: because of finite codebook size, complexity will be constant with respect to input set size. Especially when $p(x|\theta_i)$ are functions of time, the SOM acts as an efficient fingerprint of this huge amount of data. Furthermore, we might expect small codebooks to perform some useful generalization if X is small and unrepresentative (see figure3.3).
Prototypical application of the SOM for classification is a two stage procedure. First a SOM codebook is formed to approximate the probability density of the input manifold. In the second step we will assign an appropriate θ_i to each code $m_i \in C$. Usually this is done by majority voting: a code is labelled with the class that occurs most in its Voronoi cell. We present each input once to the codebook. [2] For each nearest neighbor we keep track of the number of times it scores for a certain input class. In the end we label each code with the class it responded most often to. [3]

[1] With $M, R, R^* \geq 0 \Rightarrow MR^{*2}/(M-1) \geq 0$. So we see $R \leq 2R^*$ and $R^* \geq \frac{1}{2}R$

[2] Or we take a representative subset of the input

[3] In case of ties or codes that never win we can label the code as 'unclassified'. If we use the kronecker delta function as a cost function, in case of ties, labelling the code with any of the competing class labels will reduce classification error on the training set.

Chapter 4

Coding class boundaries

4.1 Introduction

In a sense using a codebook for classification is not very logical. Most of the codebook will be used to code the cluster centers, because the input is most dense there. However, for classification we are more interested in coding the decision boundaries. This is the area where classification can be improved. See for example figure 4.1. Let's assume there are two classes; one left, and one right. The size of the Voronoi regions indicate where most of the inputs lie: small Voronoi tiles suggest that input is dense there. We see that the decision boundary is relatively sparsely coded. All code vectors that occupy a Voronoi cell without edges lying on the decision boundary could be deleted without changing the decision boundary and thus without changing classification error! Maybe we should instead use these 'spurious' codes to code the decision boundary at higher resolution (figure 4.2); of course at the risk of overgeneralization.

4.2 Coding with a Duo of SOMs

We try to achieve this kind of pattern space coding by using a duo of interacting SOMs. The first SOM explores pattern space to find class boundaries, and acts as a filter for the second SOM. The second SOM is then used to code these interesting areas. We have two slightly different implementations of this idea, called BorderSOM and EdgeSOM (BSOM, ESOM), of which only ESOM makes use of the ordering L. These implementations differ in the filtering criteria which define which inputs are sent through to the second map:

- **BorderSOM** The second SOM only receives an input if the two nearest neighbors in the first SOM have different class labels.
- **EdgeSOM** The second SOM only receives an input if the nearest neighbor in the first SOM has neighbors in L of a different class.

Let's illustrate BSOM and ESOM with a simulation example (figure 4.3). Assume we want to map three Gaussian input distributions in pattern space with BSOM or an ESOM, each consisting of two 10×10 SOMs. To reduce the number of times we have to label the codebook, we train the maps separately. The first map is trained on the input and its codes are labelled by majority voting. This first map is not shown in this figure. Then training continues differently for ESOM and BSOM.
In the ESOM algorithm we assume that the different classes are represented by different areas on the map. So we assume that the classes approximate clusters in input space and that the ordering property holds for the first map. Only if a code in the first map has a direct neighbor in L of different class we label this code UNLOCKED, otherwise we label it LOCKED. Now we train the second map only with those inputs that have an UNLOCKED first map code as nearest

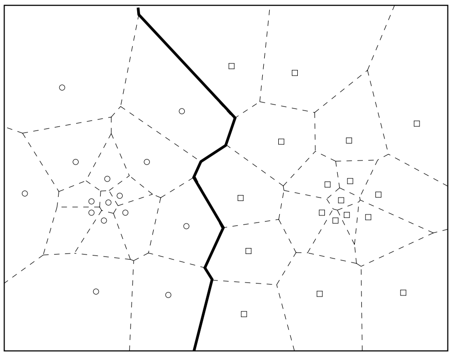


Figure 4.1: Two class problem. Only a small subset of Voronoi edges is also part of decision boundary.

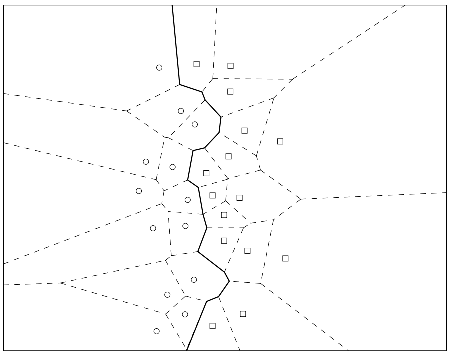


Figure 4.2: Two class problem. Decision boundary coded at higher resolution.

neighbor. So the second map will only code that part of the input manifold that contains a decision boundary (see figure 4.3). When training is finished we label the second codebook by majority voting. During recognition, the ESOM produces the code belonging to the nearest neighbor found in the union of both codebooks. This is done because although the second SOM codes class overlap at a higher resolution, there still may be a first map code that is nearer to the input than the nearest neighbor in the second map.
For BSOM training the ordering property is not needed. For each input we select the two nearest neighbors in the first codebook and compare the class labels. If these labels are different, the input is sent through to the second map. After training the second map is labelled by majority voting; during recognition BSOM also produces the code to the nearest neighbor found in the union of both codebooks.

4.3 Related Work

The Kohonen LVQ1-3 algorithms are related to BSOM and ESOM because they perform exactly the opposite operation: codes are shifted towards class centers, away from class boundaries. This can be a matter of scale: to avoid the pitfall of overgeneralization it could be better to code a class with a few codes in the center instead of coding class decision boundaries with a lot of codes.
An example of using competing SOMs to map disjunctive parts of input space is described in [Cheng, 1992]; an example of multiple SOMs coding exactly the same areas at different resolutions is the HSOM that we discussed earlier [Lampinen and Oja, 1992].

4.4 Experiments and Results

We performed some experiments with BorderSom and EdgeSom to test their classification ability. To achieve realistic results all experiments were performed on real world problems. Three medical diagnosis problems were taken from the Proben1 benchmark collection: Pima Indians diabetes data, Cleveland heart disease data and Wisconsin breast cancer data. Additional experiments were run on the so called Vowel Recognition Problem and LandSat Sattelite Image data. For descriptions and references of these dataset see appendix A. We divided all available data in training sets, validation sets and test set, generally in the ratios 2:1:1. Only the training sets were used to train the networks. The validation set was used to test the generalization capability of the networks on new data. The best network found was finally tested on the test set. For each medical diagnosis problem three different permutations of the original data were used to check the effect of different divisions.

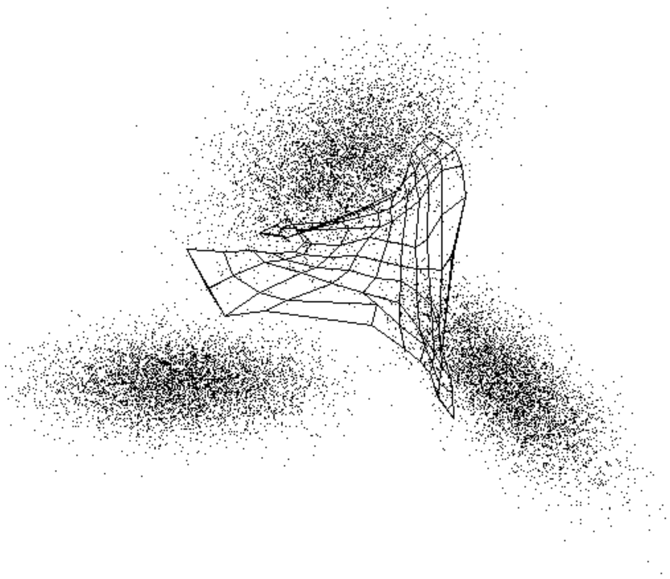


Figure 4.3: EdgeSom coding of three Gaussian input distributions in two dimensional pattern space. First and second had 10×10 ordering; second SOM (first SOM not shown) mainly codes areas of class overlap.

	NN	SOM	BSOM	ESOM
diabetes1	27.1	26.0	27.1	25.0
diabetes2	38.0	31.8	32.8	27.6
diabetes3	28.6	29.2	32.8	25.5
diabetes	31.2	29.0	30.9	26.0
cancer1	4.0	2.3	1.7	1.7
cancer2	4.0	4.6	4.0	4.0
cancer3	4.0	2.9	2.9	1.7
cancer	4.0	3.3	2.9	2.5
heartc1	3.0	34.7	26.7	29.3
heartc2	12.0	9.3	9.3	9.3
heartc3	22.7	21.3	20.0	21.3
heart	12.6	21.8	18.7	20.0

Table 4.1: Test set classification errors (%) for the Nearest Neighbor, SOM, BSOM and ESOM algorithms on benchmarks from the Proben1 benchmark collection. For explanation about the benchmarks used see appendix A.

	NN	SOM	BSOM	ESOM
vowel	41.1	50.6	52.0	49.9
landsat	10.6	14.5	14.8	12.6

Table 4.2: Test set classification errors (%) for the Nearest Neighbor, SOM, BSOM and ESOM algorithms on the vowel and `landsat` problem. For explanation about the benchmarks used see appendix A.

Tests were run for different SOM, BSOM and ESOM architectures. The output of such a run was the test set classification error (% test samples classified wrong) on the architecture with lowest validation set error. By selecting an architecture in this way we hope to find the network that generalizes best on new (test) data. SOM architectures used were 5×5, 7×7, 10×10, 15×15 and 100×1. BSOM and ESOM architectures were $5 \times 5 - 5 \times 5$, $7 \times 7 - 7 \times 7$, $13 \times 1 - 6 \times 6$ and $51 \times 1 - 7 \times 7$, all with rectangular neighborhood relations (8 neighbors). In ESOM we reduced the number of codes that qualify to pass inputs through by only looking at codes directly left and above the winning code for class difference.
All SOM, BSOM and ESOM training was divided into an ordering and a finetuning phase. In the ordering phase, both neighborhood size δ and learning speed α decreased linearly, δ from half of maximum lattice size to 0 and α from 0.1 to 0. In the finetune phase neighborhood was 0 and learning speed decreased from 0.02 to 0. The ordering phase took 2000 input presentations, finetuning took 30000 input presentations. In the BSOM and ESOM experiments we first ordered SOM one, then we ordered SOM two, finetuned SOM two and then finetuned SOM one. To reduce training time we used a block neighborhood function.

We hoped that the BorderSOM and EdgeSOM algorithms would score better than the normal SOM algorithm, to compensate for the increased computational complexity. See table 4.4 and table 4.4 for the results we got on the experiments.

4.5 Discussion

We would like to discuss some of the results of the experiments.
Firstly, the experiments show huge differences between the different permutations of `heartc`, `cancer` and `diabetes`. We also see that BSOM and ESOM score below half of NN error for cancer1.

This would not have been possible if the experimentally found error would have approximated real NN error. These are indications that we deal with small and highly unrepresentative training, test and validation sets. We have to keep this in mind when assessing the real practical value of a certain classification result, since different divisions of data can lead to large differences.
Secondly, we see that ESOM generally outperforms SOM, which confirms our expectations. The outcomes for BSOM are close to the SOM results. The difference in performance may be caused by the fact that the first SOM in ESOM is a 'broader' filter than the first SOM in BSOM: in general more inputs will be passed to the second map and it will occupy a larger portion of pattern space. Still, the results of ESOM compared to SOM and the computationally simple Nearest Neighbor are not always better. Also the ESOM approach is complex and problem dependent. In problems with a lot of different classes, practically all inputs will be sent through to the second map, which almost reduces ESOM to a SOM with the same size. Moreover, SOMs optimize clustering criteria and not classification criteria. For classification it is better to combine SOMs with supervised learning architectures. We shall do this in the next chapters.

The interesting result remains that both by coding either class boundaries (ESOM) or class centers (LVQ) positive results can be achieved.

Chapter 5

Constructing Radial Basis Function Networks

5.1 Introduction

The SOM can be used to detect the structure of pattern space. We can utilize this information for optimal initialization of Radial Basis Function (RBF) Networks [Moody and Darken, 1989, Hertz et al., 1991]. Before turning to our application we will first discuss Radial Basis Functions, feedforward networks and radial basis function networks.
Radial Basis Functions are functions that have their maximum at a locally constrained region of input space. This activation peak has a certain position m and a certain scale σ. It can for example be a Gaussian (figure 5.1), but can also be a non-differentiable function like a block function or pyramidal function.
We can combine RBFs with a feedforward network to create RBF networks (see figure 5.2). We will first turn to the normal feedforward network.
The most simple feedforward network consists of two layers, an input layer L_1 of size n and an output layer L_2 of size m. Each input unit i corresponds to an input pattern feature, every output unit j corresponds to a certain class. Both layers are totally connected with weights w_{ij} determining the strength of the link. An input x is coded as activations on L_1; the activation o_j of each output unit j is determined by calculating a weighted sum over all input unit activations and applying a transfer function on the result:

$$o_j = f\left(\sum_{i=1}^{n} x_i w_{ij}\right)$$

with L_1 input layer of size n, o_j output on unit j of layer L_2, w_{ij} weight on connection between input unit i and output unit j. A common transfer function f is the step function: $f(x) = 0$ if $x < t$, $f(x) = 1$ if $x \geq t$ for some threshold t. In this setup every output unit j defines a hyperplane

$$\sum_{i=1}^{n} x_i w_{ij} = t$$

in pattern space $\mathcal{R}^n$, with $o_j = 0$ for all inputs at one side of the hyperplane and $o_j = 1$ for inputs lying at the other side of the hyperplane.
To achieve optimal classification, these hyperplanes should coincide with the class decision boundaries. To find the optimal set of weights we can train the network with the delta rule:

$$\Delta w_{ij} = \alpha(\xi_j - o_j)x_i \quad \forall j \in L_2, \forall i \in L_1$$

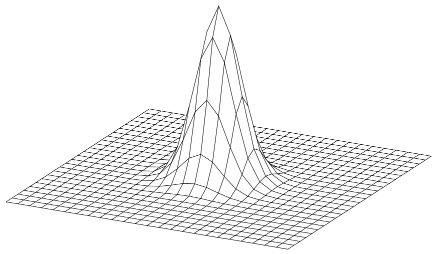


Figure 5.1: Radial Basis Function: Gaussian $A_i = e^{-\|x-m_i\|^2/\sigma^2}$ with $\sigma = 1, m = (0,0)$.

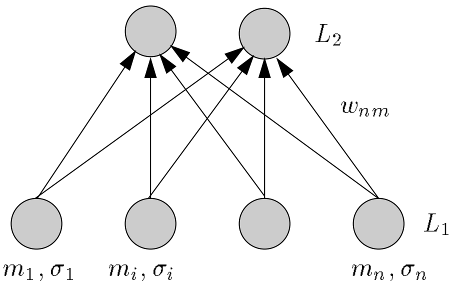


Figure 5.2: RBF network: every input unit i corresponds to a RBF with position m_i, scale σ_i; every output unit j corresponds to class j.

with Δw_{ij} weight change on connection from input unit i to output unit j, o_j output on unit j of layer L_2, ξ_j target output (1 for class number j, zero for the rest of L_2) and learning parameter α [Rumelhart and McClelland, 1986].
A RBF network is an extension of the feedforward network and differs only in the input layer. With every node in the input layer a RBF A_i with position m_i and scale σ_i is associated. These radial basis function should pave the input manifold with overlapping receptive fields of different scale. Instead of coding the input as a distributed activation pattern on L_1 we apply each A_i to x to compute the activation for each unit $i \in L_1$. Activation propagation and weight training is the same as in the normal feedforward network. Prototypically, the positions of the input layer RBFs are calculated using any algorithm that yields a codebook, e.g. k-means clustering or the SOM. The scale of a RBF is usually set to the distance to the nearest other code.

5.2 RBF Initialization with SOMs

There is a close relation between a set of RBFs and a SOM: both are designed to be a robust and regular representation of the input manifold, which occupies only part of pattern space at varying densities. The code values in a SOM and the positions of the RBFs indicate in which parts of pattern space input is located. In its neural network version a SOM boils down to a set of RBF's with constant fixed scale (neighborhood size). However, the varying densities of the input manifold are represented in RBFs by varying scales. In the SOM density of the input manifold is not explicitly represented, but is implicitly present in the ordering information: the ordering property ensures that pattern space distance to lattice neighbors corresponds to the distance to the real nearest neighbors in pattern space; the clustering property then ensures that small distances correspond to relatively dense input. So ordering information and RBF scales are related.
Therefore in our application we use a SOM to initialize the parameters of the RBF network. Basicly, the codebook is used to set the positions; the class labels on the codebook determine the initial values of the weights. Furthermore we use the codebook and the ordering to set the initial values of the scales.
We start off with a SOM of codebook size n and train it globally on the input manifold. We label the codebook. Then we construct a RBF network with n input units and m output units, one for each class. Positions of input unit RBF's are set to code vector values. Then for each input unit we set all the weights on the outgoing connections to zero, except for the connection to the output

unit that corresponds to the class label the codebook predicts; this connection is set to one. The initial RBF network will now predict the same class label as the SOM. We use the ordering L to define the scales: the scale of unit i is set to the topological error (definition 2.3.3) of code i, the mean distance to its direct neighbors in L. Because this is an average scale we think this gives a more accurate scale than simply the distance to the first nearest neighbor. For the BorderSom and EdgeSom algorithms of previous section the procedure followed was similar. For each unit in the second SOMs we defined the scale as the mean of the topological error and the distance to the nearest neighbor in the first SOM.
Finally the weights on the feedforward networks are trained with the delta rule.

5.3 Related Work

The basic RBF algorithm has been described in [Moody and Darken, 1989]; the interest in RBF networks has been increasing ever since. The only other study we found that sets the scales to topological error is a combination of RBF and the growing cell structures network by Fritzke [Fritzke, 1994]. We did not find any study that experimentally compared this initialization of scale to other settings.

5.4 Experiments and Results

We ran some experiments to test our way of constructing RBF networks and explicitly test our assumption that topological error yields a better scale than nearest neighbor.
To achieve realistic results all experiments were performed on the same real world datasets as in chapter 4: `heartc, diabetes, cancer, vowel` and `landsat`. We used the same divisions in train, validate and test set and the same permutations of the Proben1 medical diagnosis data.
The same setup for finding the optimal architecture as in chapter 4 was repeated: tests were run for different SOM+RBF, BSOM+RBF and ESOM+RBF architectures. The test error for the architecture with lowest validation error was the output of the test. The SOM architectures used were 7×7, 10×10 and 15×15. BSOM and ESOM architectures were $7 \times 7 - 7 \times 7$, $12 \times 12 - 12 \times 12$ and $13 \times 1 - 6 \times 6$ all with rectangular ordering relations (8 neighbors). The RBF network was constructed as described before. To test our assumption that topological error is a more useful scale than simply the distance to the nearest neighbor we also constructed this kind of RBF network (NNRBF).
All SOM, BSOM and ESOM training was divided in an ordering and a finetuning phase. In the ordering phase, both neighborhood and learning speed decreased linearly, neighborhood from half of maximum lattice size to 0 and learning speed from 0.1 to 0. In the finetune phase neighborhood was 0 and learning speed decreased from 0.02 to 0. The ordering phase took 2000 input presentations, finetuning took 30000 input presentations. In the BSOM and ESOM experiments we first ordered SOM one, then ordered SOM two, finetuned SOM two and then finetuned SOM one. To speed up training we used a block neighborhood function.
The radial basis function we used for experiments was (see also figure 5.1):

$$A_i = e^{\frac{-\|x-m_i\|^2}{\sigma^2}}$$

Following [Fritzke, 1994] we did not perform any normalization on the RBFs. We used a linear transfer function; the output of the RBF network was defined as the index j of the output unit with maximum activation. Feedforward weights were trained with the delta rule with learning parameter α fixed at 0.05. We trained for 30000 input presentations (not epochs!) which is short when compared e.g. to backpropagation and checked every 250 steps if validation set classification error was lower. The output of a single architecture run was the test error for the iteration when validation error was lowest.
We hoped for the following results: All RBF algorithms should score better than normal SOMs. All RBF architectures initialized to topological error (RBF, BRBF, ERBF) should outperform

	NN	SOM	NNRBF	RBF	BRBF	ERBF
diabetes1	27.1	26.0	26.6	24.0	25.5	24.0
diabetes2	38.0	31.8	30.2	28.1	29.2	30.7
diabetes3	28.6	29.2	29.7	27.1	26.0	28.1
diabetes	31.2	29.0	28.8	26.4	26.9	27.6
cancer1	4.0	2.3	2.9	1.2	2.3	2.3
cancer2	4.0	4.6	6.9	4.0	5.2	3.4
cancer3	4.0	2.9	3.4	3.4	4.0	4.6
cancer	4.0	3.3	4.4	2.9	3.8	3.4
heartc1	3.0	34.7	21.3	20.0	24.0	24.0
heartc2	12.0	9.3	8.0	6.7	8.0	5.3
heartc3	22.7	21.3	18.6	16.0	16.0	17.3
heart	12.6	21.8	16.0	14.2	16.0	15.5

Table 5.1: Test set classification errors (%) for Nearest Neighbor, SOM and RBF initialized with NN, SOM, BSOM & ESOM on datasets from the Proben1 benchmark collection. For explanation about the benchmarks used see appendix A.

	NN	SOM	NNRBF	RBF	BRBF	ERBF
vowel	41.1	50.6	37.7	33.3	35.1	42.9
landsat	10.6	14.5	14.7	12.8	13.1	13.1

Table 5.2: Test set classification errors (%) for the Nearest Neighbor, SOM and RBF initialized with NN, SOM, BSOM & ESOM resp. on the twospiral, vowel and `landsat` problem. For explanation about the benchmarks used see appendix A.

RBF initialized with only the nearest neighbor (NNRBF). Finally RBF networks based on BSOM and ESOM (BRBF, ERBF) should give better results than RBF.
The actual results are listed in table 5.4 and table 5.4.

5.5 Discussion

Firstly we want to make the same general remark as in the previous section about the reliability of test results on small training sets like the Proben1 medical diagnosis sets. Again, test results are very different for different permutations and divisions of the data sets. This is another indication for the unreliability of test results on small data sets, if the division into train, validation and test set is not *exactly* the same.
Secondly, we want to stress a couple of results relevant to this section. The foremost result is that in *all* cases but one our RBF network initialized with topological error (RBF) significantly outperformes exactly the same network initialized with just the nearest neighbor (NNRBF). Only in `cancer3` the NNRBF network reaches the level of accuracy of RBF!
The overall result of RBF compared to a normal SOM is very good as well, in all cases but `cancer3` RBF outperforms SOM.
Finally we see that in almost all cases BRBF and ERBF score worse than RBF. The extra computational complexity of these architectures does not pay off.

The only difference between NNRBF and RBF was that RBF utilizes the ordering L to set the scales. By running tests on a considerable number of real world data sets we found consistent empirical support for our claim that setting scales to topological error improves performance. So L can convey important information.

Chapter 6

Combining SOMs with Backpropagation Networks

6.1 Introduction

A feedforward network trained with backpropagation ([Rumelhart and McClelland, 1986], see section 6.4 for a description) is the most widely used neural network for classification. So for classification tasks it may be interesting to combine SOMs and backprop networks. We use a SOM as a pre-processor to perform quantization and dimension reduction on the training data and then the output of the SOM becomes the input for the feedforward network.
In our setup the input to the feedforward network includes information regarding ordering L. In our experiments we investigate whether the backprop network takes this ordering information into account.

6.2 Transforming the Pattern Space with a SOM

There are many ways to use a SOM as a preprocessor for feedforward networks. Basicly these algorithms differ in the way the output of the SOM is represented as input for the feed forward network.
In one type of solution every SOM code corresponds to an input unit in the feed forward network, just like the RBF networks described in chapter 5. However, if we permute the ordering in the first layer, nothing changes really: the units in the next layer compute a weighted sum over the input activations; this sum concerns the same input units because the layers are totally connected, and summation is a commutative operation. Without any additional information like the scales from chapter 5, ordering information is lost without being put to use.
So for the algorithms of this chapter we considered an input representation in which the ordering information is preserved: given a certain input every feedforward input unit codes a coordinate of lattice position l_c belonging to the nearest neighbor(s) m_c. So now lattice space $\mathcal{Z}^E$ becomes the pattern space for the feedforward network.
We constructed three different approaches (see figure 6.2):

- **BMUBP** Given a certain input x the input y to the feed forward network will be the lattice position $l_c \in \mathcal{Z}^E$ of nearest neighbor code m_c. This means that the input layer will have E units.

- **BMUSBP** Let $l_{c'}$ be the index of the second nearest neighbor $m_{c'}$ (nearest neighbor in C minus m_c). Then y will be the concatenation of l_c and $l_{c'}$; the input layer will have $2E$ units.

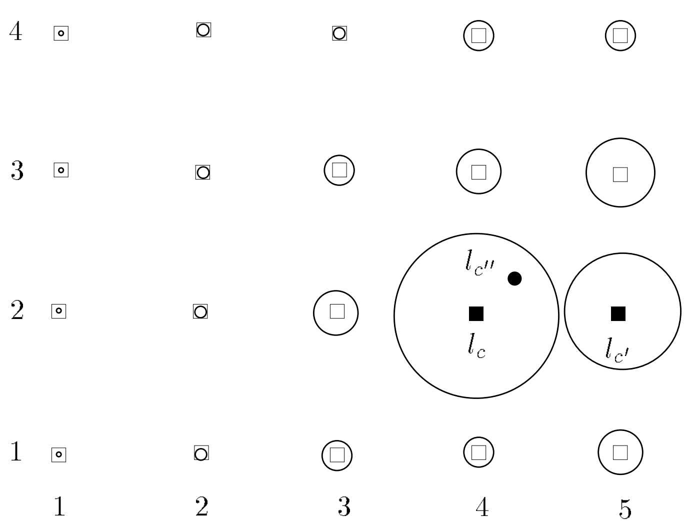


Figure 6.1: Different versions of SOM preprocesing. Examples of input to feedforward network for BMUBP, BMUSBP and DBMUBP. SOM is displayed in lattice space and activated by a certain input; radius circles indicates level of activation. The input to the feedforward network could be $l_c = [4, 2]$ for BMUBP, $l_{c'} = [4, 2, 5, 2]$ for BMUSBP and $l'_c = [4.35, 2.25]$ for DBMUBP.

- **DBMUBP** The feedforward network is trained with a continuous index l'_c instead of the usual index l_c. This should give a more accurate description of the position in $\mathcal{Z}^E$ where the gravity point of activation is located. See 6.4 for details of calculation of l'_c.

We propose the following procedure. First train a SOM on the training data. We transform both train and test set into patterns in lattice space using the rules described above. Then a feedforward network is trained and tested on this transformed data only.
This setup is a kind of special purpose algorithm: it will primarily have advantages for large datasets of high dimension. The input layer size will not be equal to the dimensionality of the the input, but it will consist of a constant low number of units instead. Since feedforward layers are totally connected, this heavily reduces the number of weights. Our primary goal in this case is not to reduce classification error, but to improve the convergence rate. And perhaps when trying to find a solution in high dimensional weight space we might get stuck in a local optimum earlier or suffer from overgeneralization. In this case our algorithm would even find a better solution. See our experiments for what we found in practice.

6.3 Related work

An early algorithm that combines a SOM with a feedforward network was the Counterpropagation network [Hecht-Nielsen, 1988, Hertz et al., 1991]. The first two layers are used to code an input manifold and a output manifold. Associative connections between the two layers are learned with the delta rule; the layers are fully connected with each other. Algorithms in which the second layer is connected partially to parts of the first layer are [Iwata et al., 1990] and [Hsieh and Chen, 1993]. In some studies the lattice position of the nearest neighbor was sent to a post-processor, just as in our BMUBP algorithm. In the HSOM discussed earlier, the lattice positions are clustered by a second SOM [Lampinen and Oja, 1992]; processing strings of lattice positions is done in various speech recognition applications [Torkkola and Kokkonen, 1991, Kangas, 1994].
An algorithm related to DBMUBP is described in [Maggioni and Wirtz, 1991]. They use a supervised SOM to predict the angle of an object, given the two dimensional projections of vertex

points. The exact angle is found by interpolating between activations of the nearest neighbor and its direct neighbor in L which is nearest in $\mathcal{R}^D$.

6.4 Experiments and Results

We ran experiments to test the performance of our algorithms in terms of convergence rate and classification error.
The experiments were performed on the same real world datasets as in chapter 4: `heartc`, `diabetes`, `cancer`, `vowel` and `landsat`. We used the same divisions in train and test set and the same permutations of the Proben1 medical diagnosis data.
First a SOM was trained on the train set. SOM topology was restricted to a 10×10 array with rectangular neighborhood relations (8 neighbors). We used a block neighborhood function with neighborhood size δ and learning parameter α both linearly decreasing. The ordering phase took 1000 input presentations (initial $\delta = 5$, $\alpha = 0.1$) and the finetune phase took 5000 input presentations (initial $\delta = 0$, $\alpha = 0.02$).
Then we transformed our train and test patterns to lattice positions: each pattern is transformed to the lattice position of the nearest neighbor for this pattern (BMUBP) or the two nearest neighbors(BMUSBP). Calculation of the continuous lattice position l'_c for DBMUBP was more complex.
We defined l'_c as the center of gravity of activation in lattice space. The activation A_i of each code m_i was calculated with the radial basis function from the previous chapter; scales were set to topological error (mean distance to lattice neighbors, definition 2.3.3). The choice for this variable scale instead of the constant scale used in SOM training was confirmed in initial experiments. Then each coordinate l'_{cj} of l'_c is determined by linear interpolation over the A_i:

$$l'_{cj} = \frac{\sum_{i=1}^{n} A_i l_{ij}}{\sum_{i=1}^{n} A_i}$$

The transformed train and test sets were used to train and evaluate the feedforward networks. For comparison we also trained a feedforward network on the original data.
The feedforward networks were trained with the momentum backpropagation rule, a generalization of the delta rule for multilayer feedforward networks. This generalized delta rule has the same form as the delta rule from chapter 5:

$$\Delta w_{ij}^t = \alpha \delta_j o_i + \beta \Delta w_{ij}^{t-1}$$

with Δw_{ij}^t weight change on connection unit i to j, α learning speed, o_i activation on unit i, δ_j error signal on unit j and $\beta \Delta w_{ij}^{t-1}$ a momentum term. The momentum term adds the weight change from the previous time step to stabilize learning. For an output unit j computation of error signal δ_j is similar to the delta rule:

$$\delta_j = (\xi_j - o_j) f'(net_j)$$

with ξ_j target output on j, $net_j = \sum o_i w_{ij}$ incoming activation on j and f' derivative of transfer function f. For a hidden unit i calculation of error signal δ_i is less obvious, because we have no target output available. We compute δ_i in terms of error signals δ_j of the next layer units to which it connects and the weights w_{ij} on these connections:

$$\delta_i = f'(net_j) \sum_{j=1 \ldots m} \delta_j w_{ij}$$

with m units in the output layer. This way all eror signals can be propagated back through an arbitrary number of layers, so that weight adaptation can take place.
For all architectures including standard backpropagation we used feedforward networks with one input, hidden and output layer, with a hidden layer of 10 units. For each class there was one

	BP	BMU+BP	BMUs+BP	dBMU+BP
diabetes1	23.5	30.2	25.5	29.2
diabetes2	25.0	26.6	29.2	33.3
diabetes3	22.4	27.1	22.9	27.1
diabetes	24	28	26	30
cancer1	1.15	1.15	1.15	1.72
cancer2	3.45	4.02	3.45	3.45
cancer3	4.02	2.87	2.87	2.87
cancer	2.9	2.7	2.5	2.7
heartc1	16.0	26.7	24.0	28.0
heartc2	2.67	6.67	8.00	10.7
heartc3	13.3	13.3	14.7	16.0
heart	11	16	16	18

Table 6.1: Proben1 test set classification errors (%) for standard backprop and backprop with one winner, two winners or an interpolated winner. For explanation about the benchmarks used see appendix A.

output unit. The classification decision made by the feedforward network was defined as the class label belonging to the output unit with the highest activation.
The networks were trained for 5000 epochs with learn parameter 0.1 and momentum 0.9; an epoch corresponds to one single sweep through all patterns x_i in train set X. For simplicity we did not use a stop criterion; output of a certain run was the lowest test classification error we obtained. The train and test classification error and root mean square error was calculated every 10 epochs. The overall results we hoped for were:

- Reasonable classification results for BMUBP, BMUSBP, DBMUBP compared to standard backpropagation for high dimensional classification problems.
- Better performance for BMUSBP and DBMUBP compared to BMUBP for any problem.
- Improved convergence rate for BMUBP, BMUSBP, DBMUBP compared to standard backpropagation for high dimensional problems.

First we applied our algorithms to the relatively low dimensional medical diagnosis problems `heartc, diabetes` and `cancer` (see table 6.1) and the `vowel` problem (see table 6.2). First note that again the different divisions of the `heartc` and `cancer` data leads to considerably different results, although the union of the train and test sets contains exactly the same patterns. This unpredictability of these real world datasets also occurred in the previous experiments. We see that for the `cancer` problem our algorithms (BMUBP, BMUSBP, DBMUBP) outperformed normal backpropagation (BP). For the other algorithms preprocessing with a SOM resulted in considerable increase in classification error, for example for the BMUBP algorithm there was a relative increase in classification error of 17% (`diabetes`) and 45% (`heartc`).
Table 6.1 also shows that there are as many runs in which BMUBP outperforms BMUSBP and DBMUBP than there are runs in which BMUSBP and DBMUBP score better or equal than BMUBP.
So for these relatively small and low dimensional data sets classification error increased in general and there was no significant improvement in classification accuracy when comparing BMUSBP and DBMUBP.

Note however that our algorithms were specifically designed to yield favourable results on large datasets of high dimension. An example of such a training set is `landsat`: it contains 4000 train patterns and 2000 test patterns each of which consists of 36 continuously valued features.

	BP	BMU+BP	BMUs+BP	dBMU+BP
vowel	32.5	49.4	53.25	48.9
landsat	37.7	18.5	16.5	17.4

Table 6.2: Test set classification errors (%) for standard backprop and backprop with one winner, two winners or an interpolated winner. on the `vowel` and `landsat` problem. For explanation about the benchmarks used see appendix A.

Measured in final classification results our algorithms (BMUBP, BMUSBP, DBMUBP) all outperformed standard backpropagation (see table 6.2). Since we just wanted our algorithms to approximate standard BP error this is beyond expectation.
To be able to compare the convergence rate of the different algorithms we had to make a couple of assumptions.
Firstly we have to define how we should measure the time scale against which the classification error is plotted. The number of iterations or epochs is not a useful measure because complexity of training depends largely on network architecture. So we chose to take the number of connection updates as the time scale. A connection update directly corresponds to a weight change. Given a feedforward network with layer sizes k, l, m, dataset size N and epoch T the current number of connection updates U will be:

$$U = (k * l + l * m) * T * N$$

Secondly, we assumed computation time for the SOM codebook was negligible. In practice unsupervised learning is very fast when compared to supervised learning. For a rough indication compare the time scale of figure 2.7 multiplied by codebook size ($10 * 10$ in this case) with the time scale in figure 6.2.
In figure 6.2 we compared classification error over time for normal BP and BMUBP (see upcoming paragraph for an explanation of PBMUBP). If we compare BMUBP test error (third line from below) and BP test error (fourth line from below) we see that the BMUBP algorithm outperforms BP. We also see that BMUBP train and test error follow each others course, whereas BP train and test error take very different values. This indicates that the better results from BMUBP are not caused because BP is stuck in a local optimum on the error surface, but BMUBP performs useful generalization on the train data. Comparing convergence rate for test errors makes no sense, but if we compare train errors we see that faster convergence only is achieved in the first 100 epochs of BP training (2.5e+09 connection updates roughly corresponds to 1500 BP epochs) which is still a considerable amount of computer time with datasets of this size and dimension.
We compared our different versions of SOM preprocessing in figure 6.3. The extra ordering information in BMUSBP and DBMUBP clearly leads to better results. The feedforward network was able to detect this extra information in the transformed data.
There is only one snake in the grass. The only information the feedforward network received were lattice positions l_i. Does this mean that this post-processor utilized the ordering?
According to our definition some algorithm utilizes the ordering if it just refers to lattice positions l_i (see section 2.1). So in this case the ordering property was utilized. The question still remains though whether the algorithm was able extract ordering information about pattern space from lattice space information alone, and whether this extra information made a difference. Did we really need the ordering property to hold to reach the same level of performance?
The answer is no: it is theoretically still possible that it would have made no difference to the feedforward network if the indices were symbolic labels belonging to some Voronoi tiles in pattern space without a certain metric or even ordinal relation defined on these labels. The input is then produced as if it were nominal, not ordinal or continuous, in other words an input (x_1, x_2) is as different from $(x_1 + 1, x_2 + 1)$ as from $(x_1 + 10000, x_2 - 10000)$.
For example, assume we have a $x * y$ SOM and a feedforward network with n hidden units. Each hidden unit then corresponds to a hyperplane in lattice space; with $n \geq (x - 1) + (y - 1)$ hidden

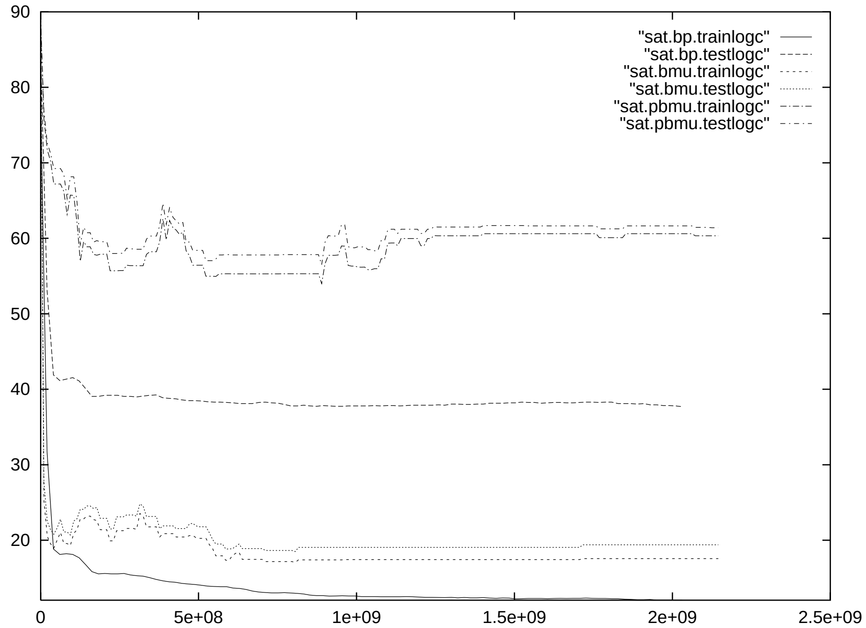


Figure 6.2: Classification error on `landsat` training and test set for normal backpropagation (BP), backpropagation with nearest neighbor as input (BMUBP), and backpropagation with one winner as input, ordering L permutated (PBMUBP). X-axis represents number of connection updates. For this problem, permutating SOM ordering severely damages classification accuracy; SOM preprocessing leads to improved classification accuracy compared to normal backpropagation.

units it is possible to identify each individual SOM code, so no ordering is needed. Depending on the nature of the problem we may need even less hidden units to separate the classes in lattice space. [1]
To exclude this possibility we constructed a 'nominal' or 'permutated' variant of our BMUBP algorithm, PBMUBP. PBMUBP differs from BMUBP only in one extra step we perform right after SOM training. We construct a bijective function p which maps a lattice position l_i to a random position l_j in lattice space $\mathcal{Z}^E$. A new SOM is then constructed by transforming the lattice positions with p:

$$l_i^{new} = p(l_i)$$

for all lattice positions l_i. So now ordering L does not make any sense any more and the feedforward network can only learn them as unrelated nominal labels. We then continued feedforward training in exactly the same way as for BMUBP.
We plotted the performance of PBMUBP in figure 6.2. We see that both PBMUBP train and test error is well above the level of BMUBP error. This explicitly shows that at least for this dataset the feedforward network utilizes ordering information to improve performance!

6.5 Discussion

The experiments have shown that preprocessing with a SOM leads to unpredictable results on low dimensional datasets. It is possible that classification improves compared to normal backpropagation but a severe increase in error can also occur.

[1] Assume we have a three class problem with $l(l_0) = \theta_0, l(l_1) = \theta_1, l(l_i) = \theta_2, i > 2$ and a SOM with linear ordering. A hidden layer of 4 units will suffice to separate classes in lattice space.

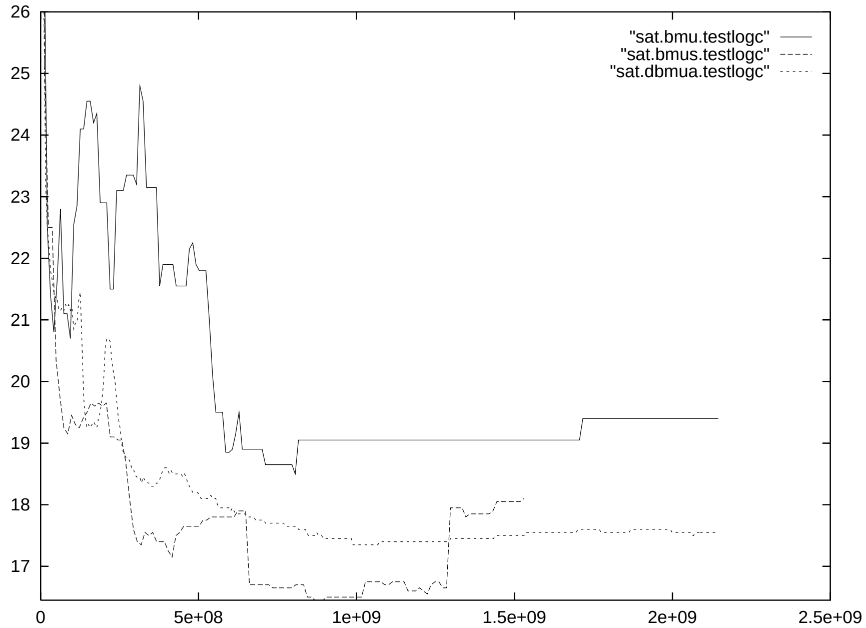


Figure 6.3: Classification error on `landsat` test set for backpropagation with nearest neighbor as input (BMUBP), and backpropagation with two nearest neighbors as input (BMUSBP), and backpropagation with continuous nearest neighbor position as input (DBMUBP). X-axis represents number of connection updates. Presenting additional ordering to the postprocessor leads to a decrease in classification error.

When we apply SOM preprocessing to high dimensional problems we can profit optimally from the decrease in the number of weights. Our experiments showed improved convergence rate on a small timescale and improved generalization.
The result that is of more relevance to the theme of our paper is that we have shown that permutating the SOM (in other words, throwing the ordering away) severely worsens classification results. Furthermore we found for our high-dimensional problem that adding extra ordering information improves performance.

Chapter 7

Discussion

In our discussion of the SOM algorithm we identified two different sources of related work: classical pattern recognition (clustering & dimension reduction) and neurobiological modelling. Let's delve a bit more into these fields to shed some light on the relevance of ordering.
Firstly, we turn to neurobiological modelling. We would like to stress that we think that this is just one of the applications for neural networks (if you want to see a SOM as a neural network at all), and nothing more. To our opinion biological plausibility is only of value for neurobiological modelling; for other applications it is totally irrelevant. On the other hand nothing keeps us from using mechanisms found in this field if this helps us to improve performance in our own applications.
Regarding the subject of ordering, it is surprising to see that the focus within neurobiological modelling has been entirely on simulation of formation of ordered brain structures. Very few work has been done on modelling mechanisms that really need an ordered structure as input, and not just to build yet another ordered representation. Candidate mechanisms are local operations like lateral inhibition, lateral facilitation, synchronisation [Knudsen et al., 1987] and the more global 'smart mechanisms' [van de Grind, 1990].
The second field of inspiration for the SOM consisted of classical clustering and dimension reduction techniques which were mainly developed outside the field of neural networks. Dimension reduction and ordering are useful when there is user interaction: if high dimensional, highly non-linear data is visualized, it is easier to interpret and act on the data. The same goes for the so-called hierarchical clustering algorithms. Though these algorithms were designed to speed up clustering, the generic relation between clusters that evolves can be useful for user interpretation.
So regarding the utility of ordering we claim that little spinoff is to be expected from the field of neurobiological modelling and that the purpose of ordering in classical pattern recognition is mainly constrained to user interaction.

Let's get back to our approach to the central theme, the classification experiments.
When we compare our algorithms (ESOM, RBF, BMUBP) to their counterparts (SOM, NNRBF, PBMUBP) we see that the use of ordering information generally leads to improved performance. Let's discuss our algorithms one by one.
The ESOM algorithm outperforms the SOM in 9 sets and has a similar performance in the other 2. So we have consistent positive results for these datasets. Note however that for the task ESOM performs the ordering property is not a necessary property. The BSOM algorithm (which performs worse) works by the same principles, but does not look nearest neighbors up in the lattice but directly in pattern space.
Also our RBF algorithm generally outperforms its counterpart NNRBF, now in 10 of the 11 cases. We have shown that SOM ordering and RBF scales both code the density of the input manifold, so that makes SOMs and RBFs related. Note that again ordering was not a necessary property. We could have set the scales directly with information from pattern space.
Finally, BMUBP had faster convergence than BP on the high-dimensional `landsat` data as we

expected. It also achieved lower classification error, which was beyond our expectations. The variants which included extra ordering information (BMUSBP, DBMUBP) outperformed BMUBP on this data as well. Furthermore we showed that though ordering information was not necessary information, its ommitance resulted in considerable increase of error (PBMUBP). Just like in the previous two experiments, we showed ordering could be relevant.
On the other hand, we have to interpret these results with apt reserve. By using only real world problems we tried to approximate real world performance of our algorithms as close as possible. We have the opinion that artificial problems often contain regularities which will never arise in reality and which will bias an experiment. A disadvantage of using real world data sets on the other hand is that these sets are usually small and unrepresentative. This can lead to very different results for different divisions in training, validation and test sets.
Another constraint on the value of our positive results is set by some methodological assumptions we made. For simplicity and following previous work from other authors on the same datasets we only performed 3-fold cross-validation on the Proben1 data and no cross-validation on the `vowel` and `landsat` data. Furthermore we did use a separate test set but no validation set in the backpropagation experiments because of constraints in the backpropagation source code used.
We have already interpreted the results specificly to investigate whether using ordering had a positive effect. If we interpret the results to find the algorithm that performed best on all datasets we do not find a definite winner. Generally speaking, RBF outperforms ESOM and ESOM outperforms SOM. Though it is dangerous to compare the backpropagation experiments to the other algorithms because of differences in training methodology there is an indication that backpropagation performs well on the medical diagnosis data. Surprisingly, the simple nearest neighbor algorithm, which is trivial to implement, performs very good on the `heart` and `landsat` datasets (and on these two only). Hence one may conclude that we should always give simple nearest neighbor a try.

Generally speaking, our experiments have shown several *possibilities* to utilize the ordering property. On the other hand ordering property was not always a necessary property. We might as well get our information directly from pattern space. Furthermore, using the ordering did not supply us with an algorithm which generally outperforms all the other (nearest neighbor) algorithms; this would not have been a realistic aim for research either.
We conclude that the ordering property contains useful information. This information is usually thrown away, which seems to be a waste. But as long as there are no clear mechanisms found that necessarily need the ordering property to hold it makes no sense to refer to this property as an explicit reason for using a SOM instead of another algorithm. Until then we can only justify our choice of the SOM algorithm by referring to clustering or classification performance.

Chapter 8

Conclusion

When we compare our algorithms (ESOM, RBF, BMUBP) to their equivalents that do not use ordering explicitly (SOM, NNRBF, PBMUBP) we see that there is improved performance. For these algorithms ordering helps to improve classification performance. So in these cases order makes sense.

On the other hand we showed ordering was not always a necessary property in these experiments. Moreover, if we look at other SOM applications in a non neural way we see that ordering is generally not used explicitly. And we should not be using SOMs just because an ordered representation looks nice on paper.

The frequent claims that a SOM is used because it is an *ordered* representation must be judged with reserve. The exact advantages of SOM ordering in normal SOM training or in explicit use are still too unclear.

Appendix A

Benchmarks

We merely used real world data to test our algorithms, because we believe it is always possible to construct artificial data that matches a certain algorithm. The following set of benchmarks was used:

- Datasets from the Proben1 benchmark collection [Prechelt, 1994]:
  - `heartc` Cleveland Heart database: decide if one of four main vessels is reduced in diameter by approximately 50%. Features describe sex, smoking and drinking habits etc. [Detrano et al., 1989]
  - `diabetes` Pima Indians Diabetes database: diagnose diabetes of Pima Indian based on personal information and medical examinations. [Smith et al., 1988]
  - `cancer` Wisconsin breast cancer database: classify tumor as benign or malignant based on microscopic examination of cells. [Mangasarian and Wolberg, 1990]
- `vowel` Vowel recognition problem [Robinson, 1994]: distinguish 11 different vowel phonemes. Feature vectors describe 10 different frequencies for the vowels in: heed, hid, head, had, hard, hud, hod, hoard, hood, who'd en heard [Robinson, 1994].
- `landsat` Segmentation/classification of land surface areas in a Landsat sattelite image [Michie et al., 1994]. Input patterns consist of 4 spectral values for each pixel in a 3×3 neighborhood. This database was in use in the European StatLog project, which involves comparing the performances of machine learning, statistical, and neural network algorithms on data sets from real-world industrial areas.

For statistics see table A. Both the Vowel Recognition Benchmark and the Proben1 benchmark are available via the neural bench repository at CMU (ftp.cs.cmu.edu, /afs/cd/project/connect/bench/). The Landsat dataset can be found in the UCI Repository Of Machine Learning Databases and Domain Theories (ftp.ics.uci.edu:pub/machine-learning-databases).

	twospiral	vowel	heart	diabetes	cancer	landsat
binary features	0	0	18	0	0	0
continuous features	2	10	6	8	9	36
nominal features	0	0	11	0	0	0
dimension	2	10	35	8	9	36
number of classes	2	11	2	2	2	6
train set size	800	528	152	384	350	4000
val,test set size	800	231	76	192	175	435,2000

Table A.1: Statistics on datasets used.

Appendix B

The MOvieS Simulator

To study the behaviour of the various algorithms discussed in this paper we designed a C++ class library and a graphical user interface for the Silicon Graphics IRIX environment. Based on these elements we built simulators for the SOM variants described in this paper. Next will discuss then class library and the interface in more detail.

B.1 MOvieS class library

The class library is built up hierarchically around the central class `codebook` (see figure B.1). This is to express that a SOM basicly is nothing else than a codebook in pattern space, just like familiar algorithms such as LVQ or simple nearest neighbour.
The most important member functions of `codebook` handle finding the k nearest neighbors and attaching class labels to the codebook by majority voting. Furthermore there are functions to compute quantization and classification error and functions to dump a codebook on screen or in Maple format.
The main member function for the derived classes – adapt – defines for each algorithm its characteristic way of forming the codebook. Obviously this function is empty for the `nearest neighbor` object, because the codebook is set to the patterns in the train set. Each of the derived classes has its own additional specific member functions, e.g. `som` possesses functions to draw ordering L in lattice or pattern space and a function to compute topological error.
The `rbf` object takes a `som` (or a `bsom, esom`) for construction. Apart from the usual error and train functions it possesses a special function that resets the Gaussians to the first nearest neighbor. Finally we defined two auxiliary objects: `vector` is a double valued vector with all basic mathematical operations and `input` can contain a dataset with functions to read, write and generate patterns, get and set labels etc.

B.2 MOvieS graphical user interface

Based on the MovieS class library , FORMS GUI library and Silicon Graphics GL (graphics library) we built a graphical user interface for the Silicon Graphics Irix platform.On the screendump of our SOM simulator (figure B.2) the following areas and objects are of interest:

- **Viewport** The Viewport is a window on pattern space or lattice space. In this example a SOM maps a 'cross' of two planes lying perpendicular to each other. The pattern space in which the data is projected is always three dimensional: if the data is of a higher dimension we just draw the first three dimensions; if data is of lower dimension it is projected on a plane or a line. With the Viewport Buttons directly to the lower right corner we can rotate in three dimensions, zoom in or out and change the size of label cubes.

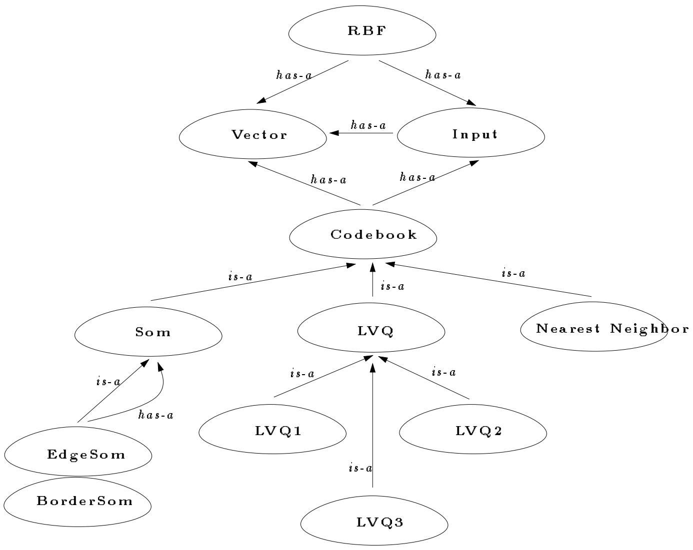


Figure B.1: MOvieS Class Hierarchy

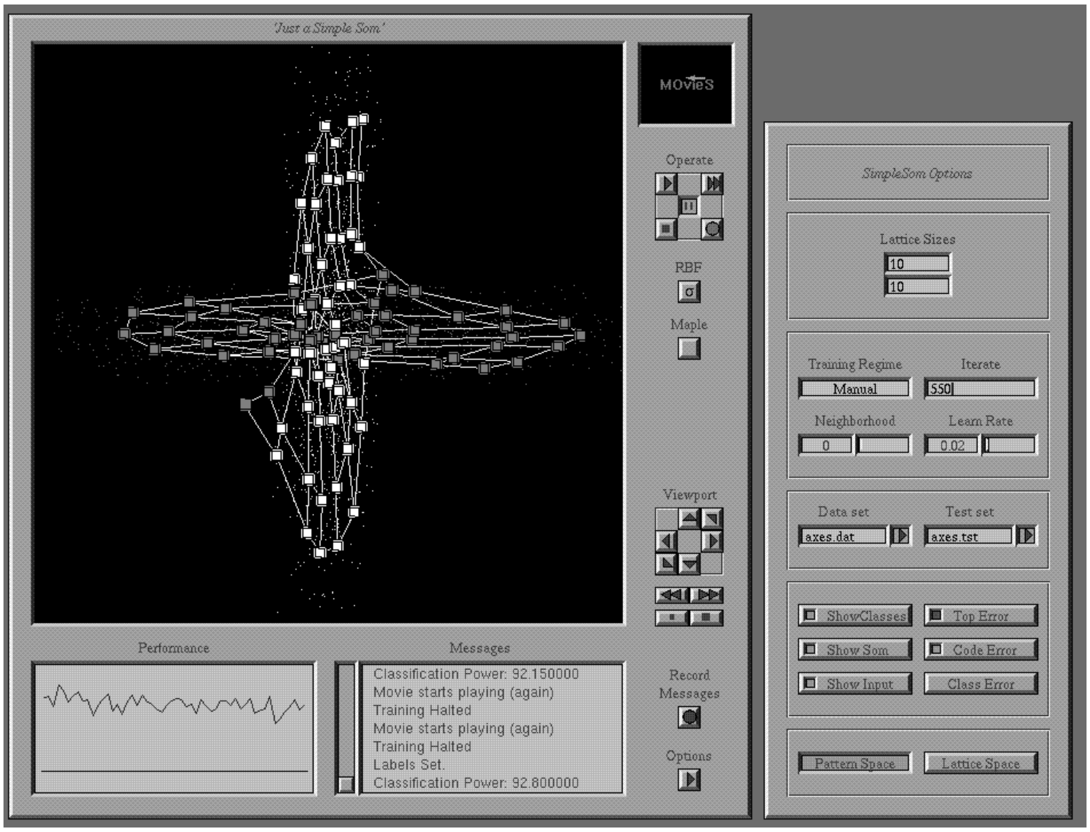


Figure B.2: An example of the MOvieS Graphical Interface: SOM Simulator

- **Operate Buttons** Directly to the upper right corner of the viewport the Operate Buttons are located. These buttons serve to stop, start, fastforward (no graphical output) or pause training and to quit the application. Below these buttons we see a button to construct a RBF network based on the present state of the SOM. A window will appear which displays train, validate and test classification errors. The Maple Button will dump the SOM and the train set in a Maple file.

- **Message and Performance Window** Directly below the lower right corner of the Viewport we see the Message Window in which information about the datasets, state of training, errors etc. is displayed. This log can be saved to a file with the Record Messages Button. In the Performance Window left to the Message Window error graphs are displayed.

- **Option Pad** Parameters specific to the simulator and extra display toggles are shown on the Option Pad.

  In this example we can set the dimensions of SOM topology and select data sets for training and testing. The training parameters can be set manually with the Neighborhood and Learn Rate Sliders. We can also choose a linear or quadratic training regime with the Training Regime Button; the Neighborhood and Learn Rate Sliders will then be controlled by the simulator.

  The other group of buttons controls which information we want to display in the Viewport ('show' buttons) or the Performance – Message Window ('error' buttons). The Pattern Space – Lattice Space Buttons toggle between pattern space and lattice space display in the Viewport.

Using these GUI objects we built simulators for some of the algorithms presented in this paper, viz. for SOM, SOM+RBF, ESOM and also for Learning Vector Quantization (LVQ). These simulators were primarily built to study whether the algorithms showed correct behavior.

Bibliography

[Amari, 1980] Amari, S.-I. (1980). Topographic organization of nerve fields. *Bulletin of Mathematical Biology*, 42:339–364. Reprinted in [Anderson et al., 1990].

[Anderson et al., 1990] Anderson, J., Pellionisz, A., and Rosenfeld, E., editors (1990). *Neurocomputing 2*. MIT Press, Cambridge, Mass.

[Angèniol et al., 1988] Angèniol, B., Vaubois, G. D. L. C., and Texier, J. Y. L. (1988). Self-organizing feature maps and the Travelling Salesman Problem. *Neural Networks*, 1(4):289–293.

[Budinich and Taylor, 1995] Budinich, M. and Taylor, J. G. (1995). On the Ordering Conditions for Self-Organizing Maps. *Neural Computation*, 7(2).

[Cheng, 1992] Cheng, Y. (1992). Clustering with competing self-organizing maps. In *Proc. IJCNN'92, Int. Joint Conf. on Neural Networks*, volume IV, pages 785–790, Piscataway, NJ. IEEE Service Center.

[Cherkassky and Lari-Najafi, 1990] Cherkassky, V. and Lari-Najafi, H. (1990). Self-organizing neural network for nonparametric regression analysis. In *Proc. INNC'90, Int. Neural Network Conf.*, volume I, pages 370–374, Dordrecht, Netherlands. Kluwer.

[Cottrell and Fort, 1987] Cottrell, M. and Fort, J. (1987). étude d'un process d'auto-organisation. *Ann. Inst. Henri Poincaré (Probabilités et Statistiques)*, 23:295–311.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. In *IEEE Transactions on Information Theory*, volume IT-13, pages 21–27. Reprinted in [Anderson et al., 1990].

[Detrano et al., 1989] Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., and Schmid, J. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–310.

[Durbin and Mitchison, 1990] Durbin, R. and Mitchison, G. (1990). A dimension reduction framework for understanding cortical maps. *Nature*, 343:644–647.

[Durbin and Willshaw, 1987] Durbin, R. and Willshaw, D. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691.

[Fritzke, 1994] Fritzke, B. (1994). Growing cell structures — a self-organzing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460.

[Goodhill et al., 1995] Goodhill, G. J., Finch, S., and Sejnowski, T. J. (1995). Quantifying neighborhood preservation in topographic mappings. Technical Report INC-9505, Institute for Neural Computation Report Series.

[Goodhill et al., 1996] Goodhill, G. J., Finch, S., and Sejnowski, T. J. (1996). Optimizing cortical mappings. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Neural Information Processing Systems*. MIT Press, Cambridge MA.

[Hecht-Nielsen, 1988] Hecht-Nielsen, R. (1988). Applications of the counterpropagation network. *Neural Networks*, 1:131–139.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the theory of neural computation.* Addison Wessley.

[Hsieh and Chen, 1993] Hsieh, K.-R. and Chen, W.-T. (1993). A neural network model which combines unsupervised and supervised learning. *IEEE Trans. Neural Networks*, 4(2):357–360.

[Iwata et al., 1990] Iwata, A., Tohma, T., Matsuo, H., and Suzumura, N. (1990). A large scale neural network 'CombNET' and its application to Chinese character recognition. In *INNC'90, Int. Neural Network Conf.*, volume I, pages 83–86, Dordrecht, Netherlands. Kluwer.

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

[Knudsen et al., 1987] Knudsen, E. I., du Lac, S., and Esterly, S. D. (1987). Computational maps in the brain. *Annual Review of Neuroscience*, 10:41–65.

[Kohonen, 1982a] Kohonen, T. (1982a). Analysis of a simple self-organizing process. *Biological Cybernetics*, 44:135–140.

[Kohonen, 1982b] Kohonen, T. (1982b). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69.

[Kohonen, 1989] Kohonen, T. (1989). *Self-Organization and Associative Memory.* Springer Verlag, Berlin–Heidelberg–New York, third edition.

[Kohonen, 1990] Kohonen, T. (1990). The Self-Organizing Map. *Proceedings of the IEEE*, 78(9):1464–1480.

[Kohonen, 1995] Kohonen, T. (1995). *Self-Organizing Maps.* Springer, Berlin, Heidelberg.

[Kohonen et al., 1990] Kohonen, T., Raivio, K., Simula, O., Ventä, O., and Henriksson, J. (1990). An adaptive discrete-signal detector based on Self-Organizing Maps. In *Proc. IJCNN-90-WASH-DC, Int. Joint Conf. on Neural Networks*, volume II, pages 249–252.

[Lampinen and Oja, 1992] Lampinen, J. and Oja, E. (1992). Clustering properties of hierarchical self-organizing maps. *J. Mathematical Imaging and Vision*, 2(2-3):261–272.

[Linde et al., 1980] Linde, Y., Buzo, A., and Gray, R. (1980). *IEEE Transactions Communication*, 28(84).

[Loyd, 1957] Loyd, S. (1957). Least squres quantization in pcm. Technical report, Bell Labs. Eventually published in IEEE transactions on information Theory, IT-28, 1982, 129–137.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In LeCam, L. and J, N., editors, *Mathematics, Statistics and Probability: 5^{th} Berkely Sumposium*, volume 1, pages 281–297, Berkely, California. University of California Press.

[Maggioni and Wirtz, 1991] Maggioni, C. and Wirtz, B. (1991). A neural net approach to 3d pose estimation. In *Proc. ICANN'91, Int. Conf. on Artificial Neural Networks*, pages 75–80.

[Mangasarian and Wolberg, 1990] Mangasarian, O. L. and Wolberg, W. H. (1990). Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18.

[Martinetz, 1993] Martinetz, T. (1993). Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps. In *Proceedings ICANN-93 Amsterdam*, pages 427–434.

[Martinetz and Schulten, 1994] Martinetz, T. and Schulten, K. (1994). Topology Representing Networks. *Neural Networks*, 7(3):507–522.

[Michie et al., 1994] Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine learning, Neural and Statistical Classification.* Ellis Horwood Series In Artificial Intelligence.

[Moody and Darken, 1989] Moody, J. and Darken, C. (1989). Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–249.

[Murtagh and Hernández-Pajares, 1995] Murtagh, F. and Hernández-Pajares, M. (1995). The Kohonen self-organizing map method: An assessment. *Journal of Classification*, 12. (in press).

[Prechelt, 1994] Prechelt, L. (1994). PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany. Anonymous FTP: /pub/papers/techreports/1994/1994-21.ps.Z on ftp.ira.uka.de.

[Ritter et al., 1992] Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps.* Addison–Wesley, Reading Mass.

[Robinson, 1994] Robinson, A. (1994). *Dynamic error propagating networks.* PhD thesis, Cambridge University.

[Rumelhart and McClelland, 1986] Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: explorations in the microstructure of cognition.* MIT Press, Cambridge, Mass.

[Sammon, 1969] Sammon, J. (1969). A non-linear algorithm for data structure analysis. *IEEE Transactions Communication*, pages 401–409.

[Shepard, 1962] Shepard, R. (1962). The analysis of proximities: multi dimensional scaling with an unknown distance function. *Psychometrika*, 27:125–140, 219–246.

[Sirosh and Miikkulainen, 1994] Sirosh, J. and Miikkulainen, R. (1994). Cooperative self-organization of afferent and lateral connections in cortical maps. *Biological Cybernetics*, 71:65–78.

[Smith et al., 1988] Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., and Johannes, R. S. (1988). Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press.

[Torkkola and Kokkonen, 1991] Torkkola, K. and Kokkonen, M. (1991). Using the topology-preserving properties of SOFMs in speech recognition. In *Proc. ICASSP-91, Int. Conf. on Acoustics, Speech and Signal Processing*, volume I, pages 261–264, Piscataway, NJ. IEEE Service Center.

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

[van de Grind, 1990] van de Grind, W. (1990). Smart mechanisms for the visual evaluation and control of self-motion. In Warren, R. and Wertheim, A., editors, *Perception & Control of Self-Motion.* Lawrence Erlbaum Ass., Hillsdale New Jersey.

[van der Putten, 1994] van der Putten, P. (1994). Utilizing the Topology Preserving Property of Self-Organizing Maps–an overview. Unpublished paper, Dept. of Computer Science, Utrecht University.

[Vleugels et al., 1993] Vleugels, J. M., Kok, J. N., and Overmars, M. H. (1993). A self-organizing neural network for robot motion planning. In Gielen, S. and Kappen, B., editors, *Proc. ICANN'93, Int. Conf. on Artificial Neural Networks*, pages 281–284, London, UK. Springer.

[Whillshaw and von der Malsburg, 1976] Whillshaw, D. and von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization. In *Proceedings of the Royal Society of London, B*, volume 194, pages 431–445. Reprinted in [Anderson et al., 1990].

[Zhang and Mlynski, 1991] Zhang, C.-X. and Mlynski, D. A. (1991). Neural somatotopical mapping for VLSI placement optimization. In *Proc. IJCNN-91-Singapore, Int. Joint Conf. on Neural Networks*, pages 863–868, Piscataway, NJ. IEEE Service Center.