Utilizing the Topology Preserving Property of Self-Organizing Maps

an overview

Peter van der Putten[1]

[1] Dept. of Computer Science, Utrecht University

Abstract

The Kohonen Self Organizing Map (SOM) Algorithm is a popular unsupervised learning technique. It shares with classical Nearest Neighbor approaches the ability to produce an appropriate codebook vector on the presentation of an input vector. Moreover, the Kohonen Algorithm defines an ordering on the codebook vectors that reflects the structure of the input space. We claim that this ordering property is almost never utilized in SOM studies. In this paper we give some examples of research in which the ordering property is used, ranging from optimization, clustering to sequential processing applications. Finally, we outline some promising future utilities of the ordering property.

Contents

Chapter 1

Preface

Kohonen's Self Organizing Map (SOM) has become one of the most popular paradigms in unsupervised neural network learning. Originally designed to mimic the computational properties of maps in the brain, Kohonen maps are now applied in a vast research area, ranging from technological applications to neuropsychological and neurobiological simulations. What is it that makes this class of neural networks so popular?

One of the main reasons is that SOMs can give an attractive representation of the input space: not only are the units allocated according to the probability distribution described by the input vectors, there is also a topological ordering on the units that reflects the structure of the input space.

Thus, for problems that have a clear spatial structure, e.g. routing problems, cell placement, visuo-motor coordination, function approximation etc. , application of the Kohonen SOM is logical and straightforward.

But, as earlier mentioned, for other (not directly spatial) purposes such as clustering, pattern recognition and sequential processing SOMs are also popular. However, it is surprising to see that most of the time the property of the SOM to reflect the probability distribution of the input is used, but not so often the topological ordering property of the SOM is exploited!

This paper identifies research actually exploits the topology preserving property of Kohonen SOMs. The first chapter begins with some examples of neural SOMs; suggestions about the biological and evolutionary function of SOMs are presented, and the Kohonen SOM Algorithm is reviewed. The next chapter is concerned with methodological aspects of the survey, and a collection of relevant research is presented. Finally the presented research is evaluated and some promising directions for future research are proposed.

Acknowledgements This unpublished report reflects part of the work done to obtain a master's degree in Cognitive Science & Artificial Intelligence. I would like to thank both my supervisors from the Computer Science department, dr. Joost N. Kok and drs. Sebastiaan Haring, for providing me with valuable criticism, support and guidence.

Chapter 2

Introduction to the SOM

In this chapter the Kohonen SOM is introduced and its properties are examined.

One of the first purposes of the SOM Algorithm was to give an account for the automatic formation of brain maps. Therefore, first is explained what these brain maps are and what the biological utility could be. Readers not interested in biological plausibility can safely skip this first section. The next section discusses the SOM Algorithm in its most usual form (as it is used by most of the studies presented in the next chapter). Moreover, the so called local form of the SOM Algorithm is reviewed and the topology preserving property of SOMs is examined.

2.1 Topological maps in the brain

This section discusses topological maps in the brain. First some examples of brain maps are described; next, biological advantages of such ordered representations will be reviewed.

One of the most simple examples of brain maps is 'the somatosensory homunculus'. When a person receives tactile stimulation, certain parts of his cortex (the brain's outer cell layers) will be stimulated. A map of the involved cortex can be drawn with the different pieces of cortex labeled according to the part of the body they represent (see figure 2.2). A three dimensional interpretation of this emergent map can be seen in figure 2.1. One could say that the two-dimensional somatosensory cortex is cut and folded in a three-dimensional 'little man', a homunculus. Two points should be noted:

- All the body parts are in the right place, in other words the topology of the input is preserved

- Body parts with a lot of receptors (hands, lips) are exaggerated, in other words the probability distribution of the input is respected.

These properties are called respectively

- the topology preserving property (TPP) and
- the probability respecting property (PRP).

An analogous map for motor function has also been detected on the cortex [Kolb90].

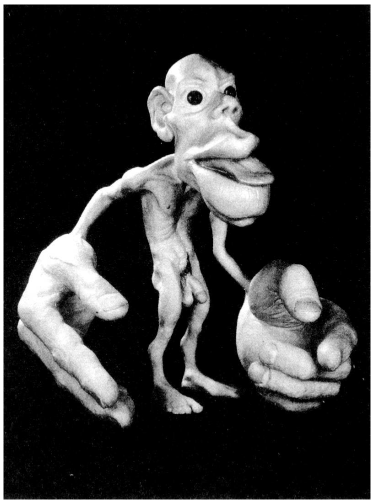

Figure 2.1: Three dimensional interpretation of human somatosensory cortex. [Kolb90], p. 219.

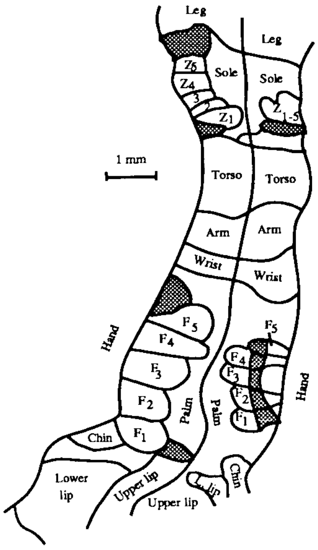

Figure 2.2: Map of a part of the body surface in the somatosensory cortex of a monkey. From [Ritter92], p. 21.

The operation of these maps could be explained by an almost one-to-one mapping from receptors (resp. effectors) to cortex neurons. However, things get more complicated when the map represents signals ordered to more abstract dimensions. Some sort of abstraction or computation must have been carried out: these maps are computational [Knudsen87]. Examples are maps in the visual area that code color, line orientation, movement direction, or ocular dominance by the possible locations on the map. Examples from the auditory area are maps that code pitch, amplitude spectrum, interaural delay, interaural intensity difference or sound source location.

Moreover, there is suggestive evidence for the existence of higher level maps that map multi-modal signals according to highly abstract dimensions. Examples are maps of objective space, faces and facial expressions and semantics of words.

However, maps are not so easy to detect as it may seem now: the topography of maps may be convoluted or curved, the size may be very small, different maps may be nested and neurons may be tuned for other (irrelevant) signals. Most important, topology is often not so well preserved as the homunculi would want us to believe... ([Knudsen87], [Kohonen89a,90a], [Ritter89], [Miikkulainen93]).

As SOMs really exist in the brain, they must have evolutionary purpose or —less dramatic— be an advantageous processing component for the brain. The same goes for the topology preserving property of SOMs. What could those advantages be?

First, because maps are topologically ordered, rather straightforward and economical schemes of connectivity can be used. This is important for both processing and genetic encoding.

¿From a processing point of view, ordering minimizes the amount of nerve tissue needed to associate brain areas that represent correlated signals. And the brain may use neural maps as a general frame of representation; it can exploit the similar form of representation to combine multi-modal information for further processing [Knudsen87]. Also the brain often employs short-range diffusion of neurotransmitters and neuromodulators; for these mechanisms some sort of localization or ordering seems preferable [Ritter89].

There is also a developmental aspect: the Kohonen Algorithm shows that the assumption of an ordering combined with very general initial connectivity and simple development rules can yield a complex set of processors, adaptively tuned to the environment. This suggests that a SOM can be an efficient 'code' for genetic encoding.

Second, the SOM proposes a solution to a fundamental question that troubled researchers ever since the invention of neuroscience: Is processing localized or distributed in the brain? The SOM adds a nice perspective to this debate: responses of neural networks are localized, but are developed and computed by the total activity of the network (see section 2.2.1). SOMs may thus be the ideal blend of a localist (almost synonym with efficient) and a distributed (almost synonym with fault-tolerant) approach.

Note that preservation of topological structure may seem a nice property; but if it is not put to use it is senseless. In other words, the mere existence of ordered maps is only part of the solution; the question still remains what kind of mechanisms take advantage of such an ordering, *whether it comes from a map or directly from a receptor surface.*

The brain is not a machine that preserves structure ad infinitum, but a system that has to interpret and act.

2.2 The Kohonen Algorithm

This section introduces the Kohonen Algorithm for SOMs as a simple nearest neighbor classifier; next, a more local and biologically plausible form of the SOM is discussed and the topology preserving property of SOMs is examined.

2.2.1 The Simplified Algorithm

The Kohonen algorithm originated from an iterative clustering method called k-means clustering and can in principle be seen as a vector quantizer.

A number of N-dimensional input vectors is coded by a smaller number of N-dimensional codebook vectors (also: reference vectors, weight vectors). Every input vector is associated with its nearest neighbor in the codebook. The codebook thus partitions the input space into a so called Voronoi tessellation (figure 2.2.1) reflecting the probability density of the input (the probability respecting property (PRP)).

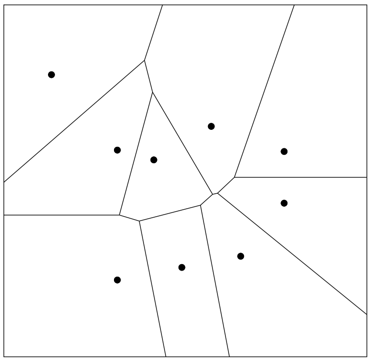

Figure 2.3: Voronoi tesselation of the input space by the codebook vectors.

Codebook formation starts with a random codebook. Then, at each presentation of an input vector, the codebook vector that resembles the input vector most to some given metric (the winner) is moved into the direction of the input vector. This step is iterated up to a given norm (e.g. number of steps, maximum fault).

The Kohonen algorithm, however, does not only generate a codebook that approximates the probability distribution of the input, but also orders the codebook vectors so that the topology of the input space is reflected (the topology preserving property (TPP)); in other words codebook vectors that are near in input space, are also near in the ordering. It does so by labeling the codebook vectors with a position, and also moving reference vectors that are near to a winner (according to their position) in the direction of the input vector.

We will now review the prototypical structure of a Kohonen SOM, and the similarity matching and the updating steps more detail.

Structure A Kohonen SOM usually consists of a 2-dimensional lattice of units (positions, neurons) $i \in I$; with each unit a codebook vector $\vec{m}_i$ is associated. An input vector $\vec{x}$ is broadcast to all units for similarity matching.

Similarity Matching This step can be described as follows: Find the unit c (the winner, best matching unit (bmu)) that minimizes the distance $d(\vec{x}, \vec{m}_c)$ with d a distance metric. For example let $d(\vec{x}, \vec{y})$ be the Hamming distance, the number of positions for which the elements of $\vec{x}$ differ with the elements of $\vec{y}$; or let $d(\vec{x}, \vec{y})$ be $\vec{x} \cdot \vec{y}$, the inner product metric (in this case it constrains $\vec{x}$ and $\vec{y}$ to be normalized); or finally let $d(\vec{x}, \vec{y})$ be $||\vec{x} - \vec{y}||$ (this is the most commonly used metric, the Euclidian distance).

Updating Not only the bmu is allowed to update its codebook vector, also its neighboring units are. The amount of updating can be made dependent on the distance from the bmu. An example of prototypical updating functions are:

$$\vec{m}_i(t+1) = \begin{cases} \vec{m}_i(t) + \alpha(\vec{x}(t) - \vec{m}_i(t)) & \text{if } i \in \mathcal{N}_c \\ \vec{m}_i(t) & \text{otherwise} \end{cases}$$

with α a learning parameter and $\mathcal{N}_c$ a neighborhood function that defines which units belong to the neighborhood. Both the neighborhood function and the learning parameter decrease linearly over time.

The neighborhood is prototypically of rectangular or hexagonal form; it defines the connectivity between units in the lattice (figure 2.2.1). Including this neighborhood enforces respecting local topology, and since the topological relation is transitive, also global topology.

This concludes the review of the general and prototypical form of the Kohonen Algorithm. The rest of this chapter will be concerned with the local interpretation of the Kohonen Algorithm and the topology preserving property of SOMs. [Kohonen82,89a,90a]

2.2.2 The Local Algorithm

The Kohonen Algorithm was originally designed to model the ability of the brain to form neural maps. However, the algorithm described in the previous section

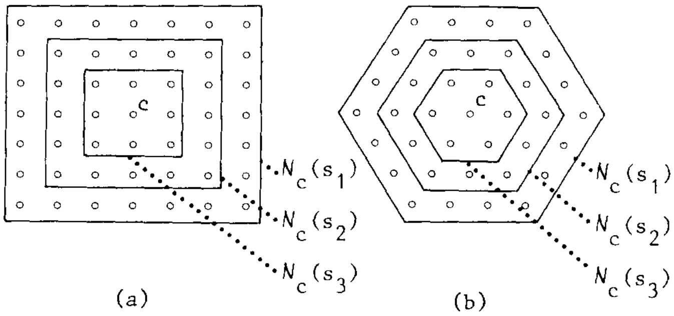

Figure 2.4: Neighborhood relations: every unit is connected to eight (a) or six (b) neighbors. From: [Kohonen89a], p. 132.

differs with 'neurobiological' computation in that it is a global algorithm: In the similarity matching step there must be a central processor to decide which of all the codebook vectors matches best with the input; and in the updating step, in adapting the weights of each unit it is of importance to know which unit is the bmu. The processing that takes place in the brain is mainly performed locally: the 'local memory' of a neuron (or functional group of neurons) only consists of (i) the signals and weights on its external input connections and (ii) the signals and weights on the connections with its neighbors.

The ordering on the lattice that is defined by the TPP can be put to use to rewrite the simplified algorithm in a completely local form. Next, we will give an intuitive account how.

Input can exist in two forms: First, the input can be given as a form of value-unit encoding: the meaning of a signal is coded by a place on a (receptor)-surface. The units of the SOM are totally connected to the receptor surface; the codebook vectors now represent weights on the connections between receptor and SOM. This approach is examined in [Ritter92], Chapter 7. Second, the input can just supposed to be a (low)- dimensional vector that is broadcast to all the units.

Each unit unit can now independently compute its initial activation. The (long range) lateral connections between units are supposed to be inhibitory (short range excitatory connections are included for direct neighbors), so that higher activated units suppress the activation of their neighbors. Thus the initial activation caused by the input will (hopefully) result in a localized response, a so called 'activity bubble' around the bmu. The bubble corresponds to the previously introduced neighborhood concept, and should be shrinking over time.

Note that this happens without 'knowledge' at the units which is the bmu; the ordering on the lattice defined by the TPP is utilized to yield a bmu without external supervision.

After settling of the activations, Hebbian learning causes the units to change its weights. The external input weight (codebook vector) is changed according to its own activity level and the input signal; the lateral weights are changed according to the unit's own activity and the activity of its neighbors. A local approach of the Kohonen algorithm as sketched above, including self-developing lateral weights and shrinking activity bubbles is described in [Sirosh93], [Miikkulainen91].

This interpretation of the Kohonen Algorithm is computationally expensive; applications that use a more or less local form of the Kohonen algorithm usually postulate a global supervisor to determine the bmu. The activation and weight adaptation is then performed locally, functionally dependent on the distance to the bmu. The distance function that models the influence of the lateral weights is usually a so called 'Mexican Hat' function (figure 2.2.2), or 'Difference of Gaussians'. Note that the bmu must be known to all units and that the DOG function simulates both excitatory and inhibitory weights. Then the prototypical updating rules have a form like:

$$\vec{m}_i(t+1) = \vec{m}_i(t) + \alpha h_{ic}(\vec{x}(t) - \vec{m}_i(t))$$

with c =bmu, h =distance function.

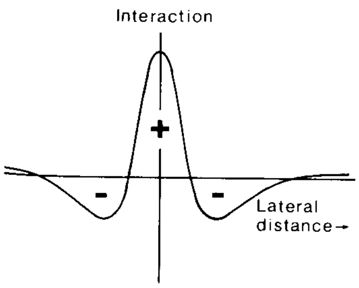

Figure 2.5: Mexican Hat function or Difference of Gaussians modeling lateral connections between units.

We saw that the ordering on the codebook vectors defined by the TPP can yield a local clustering method that has many biologically inspired and perhaps biologically plausible properties. This is not only of interest for biological simulations. Local algorithms are very efficiently implementable in parallel hardware, compared to global algorithms. SOM applications have already been implemented on for example Hillis' Connection Machine (e.g. [Ritter92], p. 120). As parallel architecture becomes cheaper, this will be a very important advantage of the Kohonen Algorithm.

2.2.3 Do Kohonen Maps Really Preserve Topology?

In section 2.2.1 it was argued that by allowing units neighboring to the winner move their codebook vectors into the direction of the input local topology was preserved, and given the transitivity of the topology relation global topology was also preserved. This proposition needs some refinement: topology can be violated, and the question is which topology is represented.

Topology is violated if close regions in input space correspond to distant locations on the map. This type of violation can occur when mapping a more dimensional input space on a lower dimensional map.

Let us consider the following example (after [Kohonen89a], [Miikkulainen93b]): we try to map a uniform distribution of input vectors from a rectangular two-dimensional input space on a one-dimensional (linear) SOM. Because the SOM tries to preserve the probability distribution of the input space, it develops an approximation similar to a space filling curve, a so called Peano curve (figure 2.2.3). As you can see, the regions A and B are close in input space, but correspond to distant units on the SOM.

Topology is also violated if close locations on the map correspond to distant regions in input space. This type of violation occurs because the SOM tries to allocate all its units to the input data.

For example, a SOM that represents high-dimensional taxonomic data displays a minimal spanning tree (MST) of the data squeezed into the map (figure 2.8). As one can see, the units A and B are close on the map, but they represent data that is distant in the formal MST representation (figure 2.7).

This problem can also be seen as a too big discrepancy between input and map topology. See figure 2.2.3 for another example.

Finally topology can be violated because convergence has not completed: the map is 'twisted'. During training, a SOM can reach some sort of a sub-optimum solution that preserves much of the local topology, but violates global topology. An example can be seen in figure 2.10: a SOM with rectangular lattice tries to map a uniform distribution of two-dimensional input vectors from a rectangular input space; but the SOM is 'twisted' and thus violates global topology. Fortunately, in this case it is easy to see that the SOM has not reached an optimal

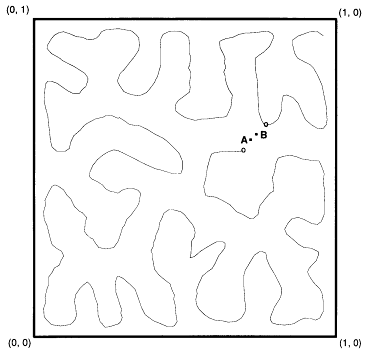

Figure 2.6: SOM with one-dimesional lattice tries to map two-dimensional input space (codebook vectors are drawn in input space and connected according to the lattice). Points A and B are close in input space, but are mapped on distant units in the lattice. From: [Miikkulainen93], p. 108.

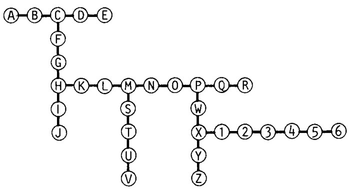

Figure 2.7: Minimal Spanning Tree of taxonomic data. From: [Kohonen89a], p. 141.

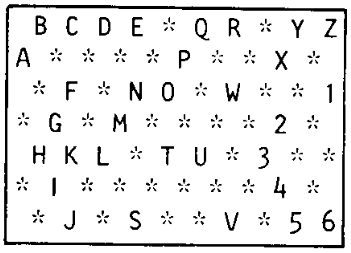

Figure 2.8: SOM mapping taxonomic data (units are labeled according to the class they respond best to; intermediate units are not labeled). From: [Kohonen89a], p. 142.

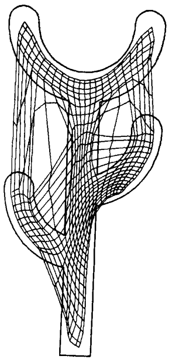

Figure 2.9: Two-dimensional SOM mapping 'cactus-shape' input distribution. Some codebook vectors are in regions outside the 'cactus' and some codebook vectors that are close in the lattice are distant on the 'cactus'. From: [Kohonen89a], p. 137.

solution.

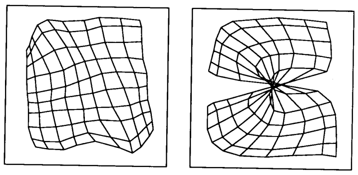

Figure 2.10: Examples of different stages of convergence in the training of a SOM. The left example achieved a good mapping, but the right example is stuck in a sub-optimal stage: global topology preservation is violated. From: [Ritter92].

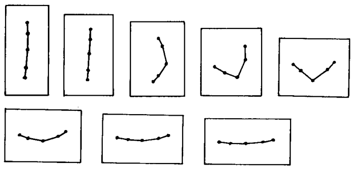

Figure 2.11: One- dimensional SOM trying to map 2-dimensional input space. Note the difference in orientation as the shape of the input space changes. From: [Kohonen89a], p. 156.

But then which topologies are represented by the SOM? The two properties of SOMs (probability distribution and topology preservation) cause it to try to map the topology of the input space along the axe(s) with the highest variance. Figure 2.11 can make this intuitively clear: as the orientation of the input space changes, so does the orientation of the axe which topology is mapped.

The axes may vary locally within the input space. This ensures that the preferred mapping has to be non-linear. The Kohonen Algorithm is ideally suited to generate such a mapping [Ritter92].

Is it possible to give a formal account of topology preservation? In the examples above, topology preservation is examined intuitively. Ordering proofs exist for the case of SOMs with one dimensional lattice structure (e.g. [Kohonen89a], for an overview [Kangas94]). However there is no general consensus yet on a formal mathematical, c.q. qualitative or quantitative account of topology preservation in the case of more dimensional SOMs. For an example, see e.g. [Martinetz93] (qualitative account of perfect preservation using so called masked Voronoi regions), [Zrehen93], [Bauer92] (both quantitative accounts of preservation).

Chapter 3

Overview of Research Utilizing the Topology Preserving Property

What research exploits the topology preserving property of SOMs, and in what ways? In this chapter we try to answer this central question by reviewing a various number of relevant SOM studies.

Before turning to the actual examples we will first define what we actually mean with 'exploiting' or 'utilizing' the TPP and give an indication of the literature sources for the survey; next the we discuss the relevance of the survey and outline the structure of the chapter.

The Kohonen Algorithm assumes units neighboring the bmu to be similar, for the codebook is ordered; thus the neighborhood of the winner is also allowed to move in the direction of the input vector, resulting in a more ordered codebook and may be better and faster codebook formation.

But what about the topological ordering of the codebook? Is it just an by- product of the Kohonen Algorithm, to take for granted? Or is it a useful property, one that lends itself to further exploitation?

In this survey every utilization of the topology preserving property other than employed in section 2.2.1 is defined as exploiting the topology preserving property.

We searched the following sources (among others) for research matching this definition:

- Proceedings of the ICANN'90 Paris, ICANN'91 Espoo, ICANN'93 Amsterdam (International Conference on Artificial Neural Networks)
- Proceedings of the IJCNN'89 Washington D.C., IJCNN'90 San Diego,

IJCNN'90 Washington D.C., IJCNN '91 Seattle, IJCNN '91 Singapore, IJCNN'92 Baltimore (International Joint Conference on Neural Networks)

- Proceedings of the ICNN'93 San Francisco (International Conference on Neural Networks)
- Neural Networks 1990, 1991, 1992
- Biological Cybernetics vol. 58-68, 1988-1993
- IEEE Transactions on Neural Networks 1990, 1991, 1992, 1993

We note that due to the limited character of this report the collection of presented research is not to be supposed exhaustive, but illustrative.

The relevance of researching SOM studies that utilize the TPP is connected with the following question: why use the Kohonen algorithm for codebook formation? There can be several reasons to select a SOM to perform a certain task.

Quantitative reasons[1] to prefer a SOM to related algorithms could be its faster and better convergence,[2] or the easiness to express the algorithm in a local 'neural' form as flows of activations between interconnected units. In the context of this paper, we are not interested in these sort of quantitative motivations. It will be clear that much debate is possible over these matters.

The qualitative motivations to use a SOM are twofold. The SOM can be used as a simple nearest neighbor classifier: on the representation of an input the most representative codebook vector is selected. These codebook vectors thus map the probability distribution of the input. As should be very clear by now, what makes the SOM special is the ordering defined on the units by the TPP. This could be an obvious motivation to use a SOM. Therefore we are interested whether and in what wat way the TPP is utilized.

The reviewed research is grouped by the task it performs and the stage in which utilization of the TPP take place: the training or processing phase. Unlike most other neural network models the Kohonen Algorithm in principle does processing and training simultaneously (in the prototypical Kohonen model processing can be compared to similarity matching and training to codebook vector updating). But in normal application of the SOM the map is first trained to form a proper codebook, training is eventually stopped and then the codebook is used to perform the processing task it was designed for. So this yielded the following grouping:

[1] A quantitative motivation for preferring a certain algorithm concerns properties on which that algorithm is claimed to perform better than others; a qualitative motivation for choosing a certain algorithm concerns properties that others don't have.

[2] For empirical evidence and a mathematical treatment on the influence of neighborhood cooperation on convergence, see [Ritter92].

- Section 3.1 is concerned with research that takes advantage of the TPP in the training stage or tries to improve SOM operation in general. The next sections will mainly be concerned with research that exploits the TPP for different processing purposes.
- Section 3.2 on optimization recognizes that the PRP (trying to cover all inputs) and the TPP (trying to keep neighboring codebook vectors related) can be seen as two opposed forces trying to reach an optimal solution. Discussed subjects are the Travelling Salesman Problem, Motion Planning, Regression and Efficient Chip design.
- Section 3.3 is concerned with the classical task for SOMs, Vector Quantization and Clustering. Discussed subjects are o.a. Reducing Transmission Error, Data Visualization and Hierarchical Clustering.
- The last section (section 3.4) discusses research that takes interest in the relation between input vectors, or in other terms is concerned with interpreting groups of input vectors. Processing sequences, for example. Subjects are Speech Processing, Contextual Processing, Semantotopical Maps and Compositional Visual Processing.

In every section, general and background information is alternated with specific references to SOM studies.

3.1 Improved Learning

As described earlier, the function of the prototypical SOM can be described in two phases: in the training stage it is used for codebook formation and in the processing stage the formed codebook is used for processing (note that the algorithm does not necessarily make this difference). This section will mainly be concerned with research that improves or speeds up convergence by exploiting the TPP during the training stage. The other sections will be concerned with models that exploit the TPP in the processing stage.

(i) Improving recognition accuracy The TPP enables easy location of codebook vectors that are near to the bmu. This information can be exploited for example to improve recognition accuracy.

References
In [Maggioni91] the recognition accuracy of a trained SOM for 3-D pose estimation is greatly enhanced by interpolating between the units. The SOM has to determine the angles with which a certain 3-D line drawing has been rotated given the coordinates of the 2 dimensional projections of the corner points. During training the

possible coordinates are learned to the SOM; moreover the selected winner units are used to update codebook vectors for the rotational parameters. During processing, information about projected corner points is used to select the initial rotational angles. Given that neighboring units code similar rotations (TPP) linear interpolation is employed to produce improved rotational angles. This can be seen as a type of final adaptation. In input space imaginary lines are drawn between the winning coordinate codebook vector and coordinate codebook vectors of its direct neighboring units. The input vector is orthogonally projected on those lines and the best resulting interpolated codebook vector is selected. Now according to its displacement of the original coordinate codebook vector, a better approximation of the rotational codebook vector is calculated (see figure 3.1). Maggioni et al. report a reduction of the average error by 84%.

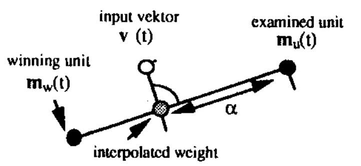

Figure 3.1: Calculation of an error measure for the winning codebook vector by interpolating between winning codebook vector, input vector and lattice neighbors. From: [Maggioni91].

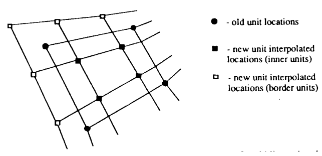

Figure 3.2: Calculating a new, larger codebook by interpolating between the old codebook vectors. From: [Rodrigues90].

(ii) Dynamical Lattice Structures A more interesting utilization of the TPP is seen in SOMs with dynamical lattice structure. These models solve a classical and important problem in vector coding and clustering: How to know—a priori—how many codebook vectors suffice. Moreover they solve a prototypical SOM problem: What is the optimal lattice structure for perfect topology preservation. Because the SOM can be seen as a labeled and ordered codebook, the TPP can be exploited to decide efficiently which reference vectors to delete or insert, or in other words where to remove or delete units.

References

In [Rodrigues90] a codebook is successively built up by interpolating between codebook vectors alternated by longer stages of normal SOM formation. The main objective was to speed up training. In the simulation example the initial codebook consists of a 3×3 lattice; it is trained for 1000 iterations and then the number of codebook vectors is doubled using interpolation rules (see figure 3.2). This procedure can be repeated a few times until convergence. Note that the new unit's codebook vectors are calculated with respect to the neighboring unit's codebook vectors (TPP is used). The authors remark that neighborhood size can remain constant because the lattice is growing itself.

The approach presented in [Fritzke91a/b/c] does not only try to determine the appropriate number of codebook vectors, but also tries to construct optimal lattice structure. The SOM Algorithm is therefore extended with both unit deletion and insertion. The lattice is initialized to a triangle of units. Then the SOM is trained for a certain number of steps: every step, the bmu has to add its distance to the input vector to its own attached error value. The bmu that collected the largest error value over this period (the so called black sheep unit) defines an area of comparatively low 'unit density'. So a new unit is created between the black sheep unit and the most distant of its direct neighbors; the new codebook vector is placed precisely in between the codebook vectors of its neighboring units (see figure 3.1). Note that contrary to the previous case, the codebook need not be recalculated in total; units are added step by step, with use of the TPP. The author also added a mechanism of unit deletion that did not make use of the TPP.

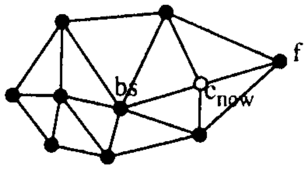

Figure 3.3: Interpolation of a new unit between 'black sheep unit' (unit that collected largest error value over a certain period of time) and the most distant of its neighbors. From: [Fritzke91b].

The incremental grid growing algorithm in [Blackmore93] is presented as an improvement of Fritzke's approach. The lattice is not an arbitrarily connected graph with but a regular 2-dimensional grid. This lattice structure should provide an always drawable representation of discontinuities, non-convexities (compare section 2.2.3), large clusters (represented by more than one codebook vector), etc. before labeling has taken place. Initial lattice structure starts with 2×2 units. Every iteration consists of three steps. First, normal SOM training is performed to adapt the codebook to the input distribution. Then at the lattice boundaries units are added in underrepresented areas, depending on some collected error measure. The positions of the new codebook vectors are calculated w.r.t. the codebook vectors of neighboring units. Finally inappropriate lattice connections are broken and desirable connections are formed by comparing codebook vector differences of (possibly) neighboring units to predefined thresholds. Note that both with adding units and changing connections the codebook is assumed to be ordered. The authors achieved favorable results from simulations; see figure 3.4 and figure 3.5 for two examples. The main objective of the authors was detecting structure of complex clusters in high-dimensional datasets. The algorithm is thus particularly fit for data visualization (see section 3.3). However, the question still remains how to interpret the lattice structure automatically... (see also note in section 2.1).

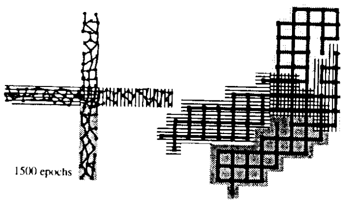

Figure 3.4: SOM grid mapping non-convex input space. From [Blackmore93].

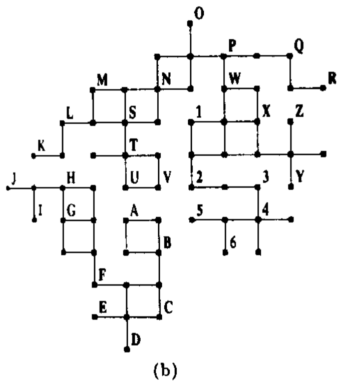

Figure 3.5: SOM grid mapping taxonomic data (see also section 2.2.3). Note that the grid structure directly corresponds with the MST representation of figure 2.7. From: [Blackmore93].

(iii) Genetic Algorithms and SOMs A more different application of the SOM and its TPP is the combination with other adaptive learning techniques, such as Genetic Algorithms (GA's).

The operation of a GA is inspired by the process of Natural Selection in biology. The GA postulates a population of programs that each code a (procedural) solution to a certain problem. A cross-over operation is defined that arbitrarily combines two individual programs; programs that code a better solution with respect to a pre-defined fitness measure have a larger probability to 'mate'. Often also some sort of random mutation process is performed to include noise.

GA's are already used for determining optimal structure of neural networks; on the other side a SOM can be used for setting a framework for Genetic Algorithms...

> **References**
> In [Toth93] such a combination of SOMs and GA's is discussed. Let us assume that the GA has to find a solution that is continuously dependent on some external variables. A SOM can be used to partition this space of external variables; units that are neighbors now represent similar situations (TPP), and because of the continuous dependence, similar solutions. The prototypical GA can now be extended to a so called GA with migration: successful individuals (representing programs to a solution) are allowed to 'mate' both with individuals in the same situation and individuals in neighboring situations. For instance, the author uses this combination of a SOM and a GA to learn a simulated robot arm to catch a falling ping pong ball without bouncing on the bat. The possible dropping heights were learned to a SOM; GA with migration was used to learn the optimal robot bat displacements over time. Final GA performance was far better with migration.

(iv) Fast bmu location In section 2.2.1 the similarity matching stage was described: which codebook vector corresponds best to the input vector. In the purely local SOM, the winner is selected by inner product calculation and lateral inhibition. However, as long as distributed hardware is not commonplace, the purely local form of the SOM will not be used. There must be a central process that finds a best matching unit. In his textbook Kohonen did not suggest what algorithm to use in a SOM implementation. The most obvious candidate is of course exhaustive search. But the ordering of the SOM may be exploited to yield a less accurate, but faster algorithm...

> **References**
> In [Lampinen89] the authors present the so called Probing Algorithm that 'travels' over the lattice of the SOM to find a well matching unit; especially in case of high dimensional SOMs with a small number of codebook vectors it performs better than traditional algorithms. It works like this: Try to find (with any algorithm) an initial matching

unit within a predetermined small number of steps. Then examine the lattice neighbors of the current matching unit; select the best matching unit and repeat this step. Terminate if none of the neighbors matches better. Note that it is obvious that this algorithm works on a trained SOM. For an untrained SOM the authors advise to disable the probing algorithm for the first few steps. But the errors the probing algorithm makes will be consistent in such manner that it will not disturb self-organization and may even facilitate it.

3.2 Optimization

The SOM mechanism can be described as trying to optimize between covering the probability distribution of the input vectors and keeping codebook vectors linked to neighboring units as similar as possible. This suggests that the SOM may be used for finding sub-optimum solutions for optimization problems.

(i) TSP An example is the Travelling Salesman Problem, the problem of finding a shortest path given that a number of positions (cities) must be visited. The cities are coded as positions in 2-dimensional input space, the lattice is one-dimensional and circular and codebook vectors are initialized to a ring-shaped distribution in input space. Now when the SOM is allowed to self-organize it can be seen that gradually codebook vectors move to the cities and at the same time, due to the TPP, paths between neighboring cities will be as short as possible (see figure 3.2).

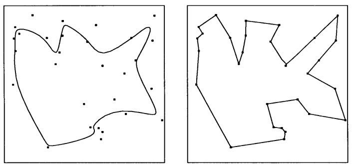

Figure 3.6: Using a one-dimensional SOM to solve the Travelling Salesman problem. From: [Ritter92].

References
In their mathematical treatment of the SOM algorithm ([Ritter88]) Ritter and Schulten were one of the first to apply the SOM Algorithm to the TSP (30 cities). Major progress was then reported

(1000 cities) in the classical paper by Angeniol et al. ([Angeniol88]) by introducing a modified version of the Kohonen SOM that included mechanisms for incremental unit creation and deletion. A similar 'evolutionary' approach is studied in [Fritzke91c]. With a constant number of units, Favata et al. reached reasonable results for problem spaces up to 10000 cities, and furthermore predicted linear complexity for their algorithm ([Favata91]; the same claim was also made by Fritzke). Ghaziri applied the SOM to related problems as the Multiple Travelling Salesman Problem and the Vehicle Routing Problem [Ghaziri91]. Finally Aarts et al. give a short review of 'neural' applications to the TSP and report favorable results on a problem of 11849 cities ([Aarts93]).

(ii) Efficient chip design An instance of the TSP is the problem how to design a chip efficiently.

References
In [Zhang91] an adapted SOM is used to solve the task of efficient chip design: How to place a certain number of modules with certain connectivity on a minimal surface with minimal circuitry length. The units of the SOM represent the modules to be placed: the lattice structure correspond to the desired connectivity. A specially designed 'layer function' analogous to the neighborhood function of section 2.2.2 is used. Input consists of the positions of empty slots on the chip. The TPP now enforces that connected modules are placed as near as possible and circuitry length is reduced. The probability distribution of the input can be adapted to fit certain design styles or avoid existing modules. In simulations the authors found that their algorithm was over four times faster than a classical algorithm (simulated annealing). For related applications see [Jiang92], [Hemani90].

(iii) Motion Planning A related application comes from the field of robot motion planning: by mapping all possible connected places where a robot can be on a two dimensional lattice, a map of objective space will be built; given a goal this map can then be used to find a shortest path.

References
In [Najand92] the authors mainly present an approach to build such a map; their algorithm is an adaptation of the modified Kohonen Algorithm of Kangas et al. [Kangas90] that uses a Minimal Spanning Tree topology. Another approach to learn topologies to a neural network is described by Martinetz et al. in [Martinetz91] with their 'Neural Gas' method: neighborhood relations between units are not

prescribed by a lattice, but dependent on the distance between its codebook vectors. In the definition in this paper it is discutable that both these papers profit from the TPP; here, the TPP is viewed as a tool to utilize or make assumptions about codebook vectors you do not have direct access to. In [Vleugels93] a SOM inspired by Fritzke's model of growing cell structure is learned to classify both legal and illegal positions to a SOM; then a deterministic procedure is used to search a path for a given start and finish.

The same procedure can be followed for learning a robot arm a space with corresponding joint angles. Given a goal in space, the robot arm can perform a smooth sequence of movements to reach the target, even with an arm with redundant degrees of freedom. The continuity of the SOM mapping enforced by the TPP ensures this ability. Even more, it is possible to include system parameters that describe eye position, head position, gripper location etc.

References

In [Saxon90] a simple model of this type is presented; possible positions with corresponding arm configurations are learned to a single 2d SOM. Obstacles avoidance is learned by 'shutting off' units that represent illegal positions. Given a start and goal, spreading activation over the legal units is suggested as a mechanism for path planning.

[Graf88] describes a classical application ('Neuroplanner') to the task of reaching an object that is focussed upon by two cameras. The Neuroplanner architecture consists of three maps: a fixed map quantizing object space and obstacles (object map), a SOM quantizing camera angles (eye map) and another SOM quantizing arm configurations (arm map). It operates as follows: The current camera configuration activates a unit on the eye map; through intra map associative connections a number of units in the arm map are activated as possible goal configurations. The current arm configuration also activates an arm map unit as the present state. Finally object map units representing obstacles label (also via intra map connections) some arm map units as impossible configurations. A classical path planning algorithm can now be used to produce an appropriate and smooth sequence of arm configurations. The intra map connections are trained in a correlation learning stage, after independent training of the SOMs.

[Graf89] describes an extension of Neuroplanner with a movable 'head' platform for the camera's; the task for the robot is now to look at its hand, which is achieved by path planning in the eye map.

There is no reason to restrict this type of application to spatial tasks. For instance, in process control (see also section 3.3) you could quantize 'sensor space' to map the undesirable process states, 'action space' to map the possible actions to direct the process and 'quality space' to map the quality of the output. Path planning on the quality map would steer the process [Graf89]. So, path planning combines well with SOMs mapping more abstract spaces.

(iv) Regression The TSP can be seen as related to regression problems: both ask for solutions that cover the input in a continuous fashion. But for regression problems, not all 'cities' have to be covered, and the resulting trajectory must not be a ring, but rather 'functional'.

References
In [Cherkassky90] a SOM is applied to the problem of non-parametric regression analysis. Non-parametric methods make few assumptions about the regression surface. The only assumption made by the authors about the regression surface is that it is a piecewise linear function: it consists of hyperplanes that join at so-called knots. A SOM can now be used to to yield the knots (due to the probability respecting property) and the ordering that is defined on them (due to the TPP). To prevent that the algorithm yields non-functional solutions, the original Kohonen Algorithm is adapted to a so-called Constrained Topological Mapping algorithm: Given a set of samples (X_i, Y_i) from N-dimensional space a $N-1$ dimensional SOM is used to find an approximation function Y (Y is function of independent variables $X = x_1 \ldots x_{N-1}$). The map is initialized so that the codebook vectors (of dimension N) are already correctly ordered in the subspace of independent variables. In the similarity matching step, the codebook vector and input are compared in the $N-1$ dimensional subspace of independent variables X. In the updating step, codebook vectors are adapted according to the distance in the N dimensional space. Note that the SOM associates a unique bmu with every sample position X. In combination with the (global) topology preserved by the TPP this probably results in a functional mapping. In simulations the authors found evidence for the claim that given a certain accuracy, the number of data points needed increased approximately linearly with the dimensionality of the problem; standard statistical techniques need exponential increase. However with this method, computational costs increase exponentially. In [Cherkassky91] the authors compared CTM (with Probing Algorithm incorporated) to Backprop and Projection Pursuit (conventional approach). They claimed CTM was superior both for prediction and speed.

An engineering application of finding a piecewise line to approximate a point distribution is described in [Gemello91]: a SOM is used for contour detection in videophone images. Detecting the visible contour of a person's face etc. is important because only the information inside this contour is worth transmitting. A pre-processing module is used to yield a cloud of points describing the contour. A one dimensional SOM can now be used to approximate the points. The line it approximates is not necessarily a function, so regular Kohonen learning can be used. The SOM is initialized to the bottom of the image; the first resp. the last unit are clamped to resp. the left lower and the right lower corner. For the first presentation of the image, the SOM takes a lot of time to converge. But because of the temporal continuity of the video signal, convergence is very fast for following frames.

3.3 Vector Quantization and Clustering

The most usual application of SOMs is vector quantization and clustering. However, taking advantage of the TPP is not very often seen.

(i) Vector quantization The prototypical task a SOM performs is vector quantization (see section 2.2.1): a smaller set of codebook vectors represents a larger set of input vectors. Vector quantization techniques are mainly used for signal transmission purposes: quantizing and thus reduction of the signal means efficient transmission (for research explicitly modeling time and sequential processing (see section 3.4). The advantage of the SOM are twofold: due to the probability respecting property signals are assigned to the correct clusters (with more clusters for ranges with frequent occurring signals); but a SOM is special in that these clusters are ordered according to the topology of the original signal (due to the TPP). If noise occurs, the error in transmission will not drastically disturb the decoding process.

References
Actually, Kohonen predicts that the most important future application of the SOM will be in telecommunication ([Kohonen90a]). Since his research group is leading in the area of SOMs this will be to some degree a self- fulfilling prophecy. In [Kohonen90b] an adaptive discrete signal detector based on a SOM is described. In ideal signal transmission, a quantized signal can only take discrete values (see figure 3.3 left). But due to transmission failures, the received values may be distorted (figure 3.3 right). A SOM can be used to solve this problem. Kohonen et al. propose a one dimensional SOM (or eventually a two dimensional one) to code the levels of quantization. The codebook vectors are one dimensional and their initial configuration may be random. The probability respecting property of the

SOM forces it to adaptively choose the right levels to code the signal. Moreover, the TPP ensures that small transmission errors will be adaptively corrected. Noise will affect the signal levels, but not the structure of the signal. Thus the codebook vectors will adapt itself to the changing values, but the topological relation (linear ordering relation) stays the same. Thus topology of the original signal is preserved. Kohonen et al. report favorable results on simulations carried out with one- as well as two-dimensional SOMs. As long as distortions weren't too big, zooming, rotating or shifting the signal didn't influence signal detection. Adaptation was most reliable for deformations that caused changes topologically similar to all levels (or in 2-d, grid points).

In another application a SOM was used for reduction of bit errors in transmission of speech ([Bradburn89]). Note that a direct coding of a signal by a binary number may involve large errors, for example when a 'high' bit is flipped. Contrary to the previous case, the SOM was not one but eight dimensional. A standard set of quantization levels (so-called μ-255) is learned to the network (a unit cube); neighboring nodes (nodes that differ only one bit in position on the lattice) now represent similar quantization values. After learning, all codebook vectors are clamped to the μ-quantization values they represent best. Now the coefficients of the SOM are a robust code to represent quantization levels because the effect of a small bit error is averaged and minimized. In experiments, the author found that the number of bit substitutions that could be tolerated without unacceptable degradation of a speech signal could be doubled.

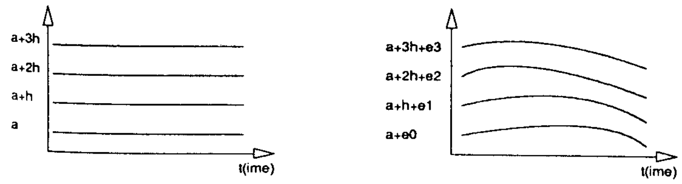

Figure 3.7: The levels on which a signal is quantized (left) may be distorted over time (right). From: [Kohonen90b].

(ii) Clustering The task that is mainly performed by vector quantization is a sort of clustering: try to find a reduced representation of a signal that preserves as much information as possible. Clustering however, is a concept that is often used in a much broader context: the input does not necessarily have to be a signal, and the reason to form clusters may be more than plain data reduction.

For example, the ordering of the TPP may be exploited for sorting high-dimensional information in lower dimensions for database purposes.

References
In [Baruah91] CAD images of objects are coded to 12-segment input vectors and learned to a SOM. Now appropriate clusters are formed; moreover the ordering of the SOM (due to the TPP) facilitates browsing through the objects. The 'continuity of representation' due to the TPP may result in units that don't represent any objects. This indicates this unit's neighbors represent very different objects.

SOM clustering is particularly fit for the task of visualizing high dimensional data. The utilization of the TPP may now seem rather trivial; the ordering is just displayed. However, it can be of great utility for an expert, say a doctor, an operator, a scientist or perhaps an expert system, to have access to a reduced representation of complex high dimensional information in which the orderings are naturally reproduced.

References
An example is visualizing data for the purpose of process monitoring and modeling in chemistry. In [Tryba91] a SOM is learned with data collected from a distillation column. Both the parameters to be optimized (the separated materials) as the other parameters (such as temperature an input mixture) constitute the components of the input vectors. To achieve sufficient topology preservation, the number of codebook vectors must be considerably larger that the number of collected process states. For every component the trained SOM can be displayed with its units labeled by the value of the component. The process dynamics can be visualized as a trajectory of winning units on the SOM (figure 3.3). With this visual information an expert operator can determine process history, stability of the present process state, desirable process states and parameters that will probably hold in future process states. For automatic process control the authors suggest that the SOM may be combined with classical path searching techniques to lead to an optimal process state. Undesirable or dangerous process states could be avoided by shutting off regions on the map.

In [Ultsch93] a technique described as a 'U-Matrix Method' is introduced to facilitate interpretation of a trained SOM. The simplest U-Matrix method calculates for each unit the total distance to the codebook vectors of it's neighboring units (so TPP is used). These

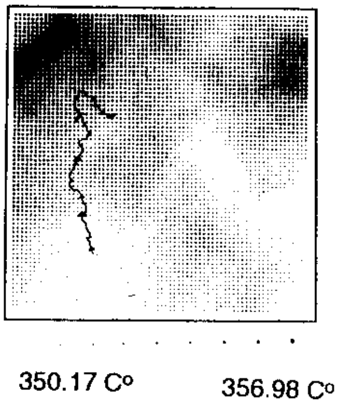

Figure 3.8: Process dynamics visualized as a trajectory on a SOM, with units labeled to temperature. From: [Tryba91].

values can be displayed as a 3-D hill surface on top of the SOM. Hills correspond to regions in which process states change quickly; valleys correspond to regions with similar and maybe stable process states. In this way, large grain clusters that consist of multiple units can easily be visualized. These descriptions yielded by the U-Matrix Method were also used to automate part of the knowledge acquisition stage in designing an expert system. The author claims that this approach was succesful for process control as well as other applications.

The responses of a SOM may be clustered by a smaller second map. Because the ordering of the responses is preserved, it can be claimed that the TPP is utilized (though the task performed is just double application of the normal SOM algorithm). The question still remains however, what the final utility of the ordering should be...

References
In [Lampinen92] a so-called Hierarchical SOM is described: a small SOM learns the output (positions of bmu's in the lattice) of a larger SOM. The authors claim that in principle any arbitrarily complex cluster could be learned by the SOM (for example non-convex clusters as in the two-spiral problem). Note that the unsupervised character of SOM learning prevents the a priori specification of the

classes to cluster. Moreover, they claim HSOM learning outperforms normal SOM learning, and classical algorithms like k-means clustering and ISODATA w.r.t. the shape of the cluster formed.

3.4 Compositional and Sequential Processing

Clustering and vector quantization algorithms learn to code discrete input vectors (or vectors drawn from a certain probability distribution) independent of their ordering to a certain cluster. But sometimes it is the relation between the input vectors we are interested in. This relation can for instance be a time ordering. Because SOMs give an ordered representation of the inputs, they may be ideally suited to map such relations. The many examples in this section will illustrate this claim.

(i) Selecting subsets Because a SOM codebook is ordered with respect to some familiarity measure (due to the TPP) selection of meaningful subsets of the codebook is easy to implement.

References
In [Kangas92a] an adapted SOM (based on Kohonen's Hypermap architecture [Kohonen91]) is used to code the (continuous) temporal development of input vectors on locations in the map. First the context of an input vector is presented (e.g. the preceding signal). A so-called pre-active area is defined to be for instance the neighborhood of the best matching unit. Then the current input vector is presented to the pre-active area. So the context and the TPP are used to select an appropriate subset of the codebook on which final selection can take place. Kangas sees a lot of advantages of a pre-active area: like the other relations temporal relations are naturally coded by places on the map; different signal sets can be connected to different overlapping parts of a map, and the preactive area can be used to move over it; several independent active areas can exist simultaneously. In summary, the TPP allows context-dependent selection of bmu candidates from (possibly multi signal space) SOM codebooks.

The Trace Feature Map (TFM, [Miikkulainen92/93]) is a SOM that makes use of local interactions and the TPP to implement an episodic associative memory. Excitatory connections are used to lay traces to units that represent previously stored items. The starting point for a TFM is a trained SOM. During storage, the output of a unit over time must only depend on the external input (no lateral interactions). The map responds at an input vector with developing an activity bubble around the unit whose reference vector resembles the input vector most. Now a trace is created by updating the lateral weights of the active units (activation bigger than a preset threshold): a connection to a less activated unit becomes more inhibitory while a connection to a more activated unit becomes excitatory, both proportional to the presynaptic activity level (figure 3.9). During retrieval, the output of a unit over time must both depend on external activation as well as lateral interactions coding the traces. A stored vector can now be retrieved by presenting only an approximation: because the map is ordered the center of the initial activity bubble will be somewhere near the unit representing the target trace. The excitatory lateral connections will then pull the activity towards the target (figure 3.10) If the cue vector is too far from the target, the activity will be turned off by the inhibitory connections. The next step is again the initial response. This oscillation indicates that no appropriate trace is found. Opposed to the prototypical Kohonen Algorithm the TFM is an efficient and robust model to learn different subsets of a codebook. By storing different traces on different places on the map it tries to minimize interference. This combination should be the advantage of the TFM.

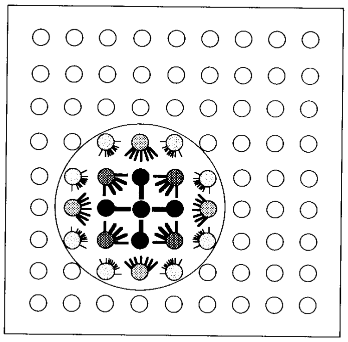

Figure 3.9: Storing a trace on a Trace Feature Map. From: [Miikkulainen93].

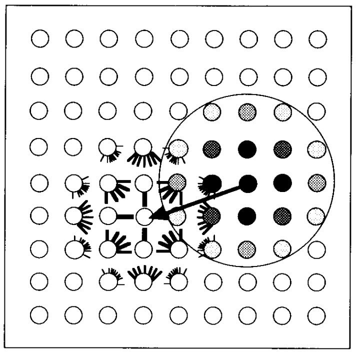

Figure 3.10: Retrieving a trace from a Trace Feature Map. From: [Mikkulainen93].

(ii) Semantotopic Maps In section 2.1 was claimed that there is some evidence for the existence of brain maps that code meanings of words, so- called 'Semantotopic Maps'. But there is a problem when trying to derive such maps automatically. Words are symbols; symbol labels tell nothing about their meaning, so merely presenting those labels to a SOM will not work. A solution is to include context, for the meaning of words is partly defined by their surrounding words. Strictly speaking of course, this ordering obtained with hard effort must be put to use in some way to qualify the semantotopic map as utilizing the TPP; up to now there have only been some hints how this may be.

References The classical paper on the semantotopic map is [Ritter89] (see [Ritter90] for a shorter version). The authors describe two methods for building a semantotopic map: training a SOM with input that consists of a symbolic part plus pre- defined attributes and training a SOM with a symbolic part plus context words surrounding the word to be learned. We will restrict our description to the second method. Input is constructed from automatically generated, semantically and grammatically correct three word sentences, such as "Mary likes Jim", "dog drinks fast" and "Bob buys meat". The symbolic part is included to allow look up without context; it consists—as a true symbol label—of a meaningless random code vector. The context part consists of the predecessor plus the postprocessor word label or only of the predecessor word label (for a simulation example see figure 3.4). Note that words are grouped according their semantical and syntactical categories, and on different levels too. Of course the question still remains how to take advantage of the resultant ordering. In related research that implements the semantotopical map [Scholtes92] a hint is given that disambiguation of a word can be solved by scanning the neighborhood of winning units.

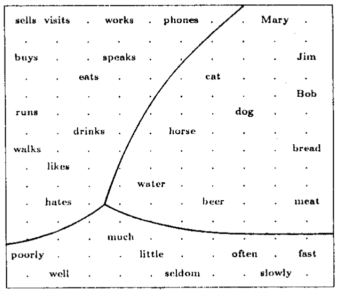

Figure 3.11: 'Semantotopic map' mapping the meaning of words by training a SOM with words plus their context. From: [Ritter90].

(iii) Processing sequences as trajectories Another way to deal with the (temporal) relationships between input vectors is to process the sequence of winning units that code the signal. This is how Kohonen's famous 'phonetic typewriter' ([Kohonen89b]) works: a speech signal is presented to a SOM to build up a so-called phoneme map that codes all possible phonemes; the trajectories of bmu's over time are processed to yield valid phoneme sequences. However, just interpreting those trajectories as sequences does utilize topology but does not implicate utilization of the ordering of the SOM. Only when is considered that the trajectory is not just a sequence of winning nodes, but a graph on the lattice with nodes on specific locations and edges with specific length, the TPP may be explicitly utilized. This can be advantageous for segmenting the sequence in meaningful parts or matching the actual trajectory with a target trajectory.

References

In [Behme93] a hierarchical model for segmenting speech is presented. In the first layer a standard SOM clusters parts of speech (not necessarily phonemes) into a phoneme map. The resulting trajectory is divided in short and long jumps. If the path stays within a short range the TPP suggests that a single segment is coded. On the contrary, a long jump codes a segment boundary. The long and short jumps are distinguished by defining a critical jump distance (an upper limit for a short jump) and a minimum segment length. A neural model for segment detection would consist of a number of units corresponding to all possible jumps, all weighted with the length of the jump they model and input to a single thresholded unit that can give an end-of- segment signal. If such a signal is given, a segment is classified.

In [Kangas92b],[Torkkola91] a speech processor is described that classifies phoneme sequences by feeding mean or median positions of trajectories to a Backprop classifier. Note that accepting these positions as meaningful inputs, the assumption is made that the codebook is ordered. The starting point is a coordinate sequence that results from presenting a (sampled) speech signal to a trained phoneme map. To include time context information feature vectors are produced by sliding a multi-resolution window over the coordinate sequence: every component of the feature vector consists of the median or mean point of 16 trajectories (coordinate sequences, windows) of different scope, centered around the present time-step. If the center of this feature vector is the best matching coordinate for 75 % of the whole duration of a certain phoneme, this feature vector is accepted as a training vector for that phoneme and used to train the BP- classifier. The authors found in simulations that the classical phonetic typewriter method still outperforms this approach.

In [Naylor92] a model is presented to recognize a small set of keywords with a SOM. The topology preserving property is exploited when a trajectory of winning nodes is compared to the nearest trajectory using a measure for the topological distortion. The SOM is used as a complement to a conventional wordspotter; it has to make the final recognition decision. For each keyword, an individual SOM is trained in such a way that the word is represented by a left-to-right trajectory through the output grid over the course of a word utterance. The horizontal axis then represents time normalized over word length; the vertical axis represents clusters that model the speech state at that moment (figure 3.12). Beginning and ending of a word are learned to only a one column wide region of the map; the mid part is learned to a wider region. This is to model the greater expected temporal variability at that points. Naylor calls this map a Temporally Constrained Self Organizing Feature Map (TSOFM). Incoming speech is processed by the conventional wordspotter and putative words are matched with the trained TSOFMs, using the same temporal constraints imposed during training. A recognition decision is based on a word distortion measure determined by the total distortion and dividing by the number of regions used for a word. A word likelyhood score is computed by subtracting from the word distortion measure a distortion measure computed by presenting the putative keyword to a SOM trained to all the keywords without temporal constraints.

(iv) Normalization and SOM combination Vector quantization as described in section 3.3 can be seen a sort of (scalar) normalization. Mapping of SOM outputs by a higher level SOM as described in section 3.3 can be used to combine information of several SOMs. In combination with techniques described in the previous paragraph this can lead to speech processor that makes optimal use of the TPP.

References
An example of such an approach is described in [De Haan92]. SOMs are used for data normalization and information combination in a speaker independent digit recognition system. From the input signal 2 sets of information were generated: vectors consisting of the output of digital filters on seventeen frequencies and vectors consisting of zero-crossing rate plus log rms energy. All the components of the vectors were learned to one-dimensional SOMs. The output of these and the other SOMs consists of the position vector of the bmu. In this manner normalization of the input is carried out. The TPP ensures that the normalized code reflects the magnitude of the signal.

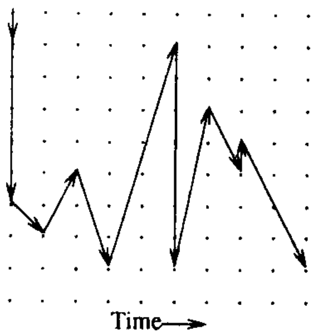

Figure 3.12: SOM mapping speech states (vertical) and time (horizontal). From: [Naylor92].

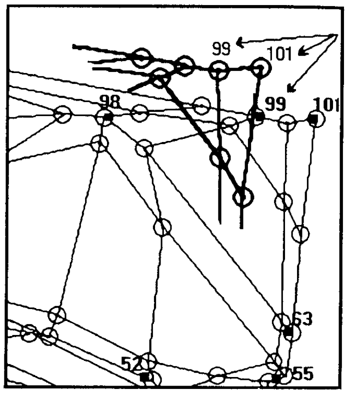

Figure 3.13: Classification of electrophoresis images with a SOM. Both the SOM trained with reference information and a SOM trained with the test information are drawn. The similar overall configuration of the lattices in input space is exploited to overcome local distortions.

The two normalized sets were learned to two different SOMs; the outputs of these SOMs (i.e. the locations of the bmu's) were then combined in a single input vector and learned to a third SOM.

(v) Compositional processing As stated earlier, we may be interested in other than temporal relations between input vectors. For example, a set of pixels or spots can be related in some sort of global organization. This global organization is reflected in the ordering present in the map; so the TPP can be exploited to overcome local distortions.

References
In [Menard90] the Kohonen SOM is used for recognition of proteins by interpreting electrophoresis images. Electrophoresis is a technique that yields a set of spots of varying intensities (representing proteins) on a two dimensional test gel. For various reasons the interpretation of these gels by comparing them to pre-labeled reference images may be difficult: the same biological sample will never produce identical images and small changes in the original sample can produce large overall distortions. The SOM, as an ordered representation, can be used to overcome certain local distortions. Test and reference gel images are pre-processed to result in a set spot coordinates. These coordinates are learned to two different SOMs, one for the test image and one for an appropriate reference image. Now a matching algorithm is used that tries to match a collection of neighboring units from both maps: For every spot in the test image, the bmu is determined. From the reference map a collection of five (labeled) bmu is determined by Euclidean distance (so without use of the TPP). For any of these bmu's a topological cost is calculated by taking information in neighboring units into account: if the test unit has neighboring units in same directions (north, south, west, east) as the reference unit, then topological cost is minimized; otherwise it is penalized. For an example, see figure 3.13. The upper middle node is associated with protein label 99, although it is closer to the protein label 101 unit in the reference image. With this approach the authors claimed that there was 10% improvement on recognition performance compared to normal SOM or nearest neighbor approach.

Chapter 4

Discussion

In this chapter we briefly discuss advantages of the TPP by making a short remark on the biological plausibility of the SOM and evaluating the reviewed research.

Let us first make a general remark. We think that that the amount of literature we researched is representative for studies using SOMs and sufficiently large.[1] However, we found that surprisingly few of these studies utilize the TPP. So a conclusion can be that SOM studies in general do not utilize the TPP.

The TPP is the main qualitative distinction with other nearest neighbor classifiers (e.g. classical clustering algorithms such as ISODATA, K-means Clustering, and neural techniques such as ART. So why use a Kohonen SOM, if the TPP is not put to use?

The only quantitative motivation for using a SOM we want to comment on is its (presumed) biological plausibility. Because SOMs are biologically inspired they may share just those desirable properties that make brain maps such efficient information processors (see section 2.1). In other words, SOMs exist in the brain, so why don't use them for engineering purposes.

However we think that the SOM is successful in the brain just because of those properties that computer architecture does not share with neural hardware. For example, for practical purposes brain processing must be confined to small areas; computers in general do not have these kind of restrictions because all memory is addressable (that is, all lattice connections are possible). Furthermore, contrary to the brain, in computers it is easy to make a copy of the original input to restore its ordering relations; for example in image processing the original image can be retained to attach the classifications made by further

[1]For an up to date collection of references to SOM studies get the file "1012 studies of the Self-Organizing Map (SOM) and Learning Vector Quantization (LVQ)" from ftp-site cochlea.hut.fi.

processing. On the other hand the Kohonen algorithm can be rewritten in a purely local form (section 2.2.2). This is not only interesting from a viewpoint of biological plausibility, it will also prove to be a major advantage when cheap parallel architecture comes available.

The reviewed research shows that the SOM has a lot of potential, but its properties can be exploited more. Especially when concepts from the previous chapter are put into combination, the TPP can turn into a true advantage. We will try to show this by making some small remarks on every section.

Amongst other things section 3.1 shows that the TPP can be utilized to improve performance in an ad hoc manner, see for instance the Probing Algorithm. Of course, this is more an implementational aspect of qualitative behaviour of the SOM. Dynamical lattices like the Incremental Growing Grid are typical examples of SOMs that prove useful when put into combination with other SOMs that suppose an ordered codebook, for example SOMs for data visualisation or path planning.

Which leads us to section 3.2. This section shows that coding and monitoring optimization problem as the TSP and related problems is very straightforward in SOMs. So when computing speed is less important than man machine interaction, SOMs can prove to be very efficient.[2] Furthermore, object avoidance (or more abstract: dangerous zone avoidance in any space) is very easy to implement in SOMs.

The most trivial utilization of the TPP from the next section (section 3.3) may perhaps turn out to be the SOMs greatest commercial success: data and process visualization. The SOM, possibly enhanced with methods like U-Matrix maximizes possible interaction between professional or layman. Note that U-Matrix and similar methods perform the same task as SOMs with dynamical lattices; the only difference is that it is applied on a trained codebook, instead of being executed iteratively during training.

Finally, exploiting the TPP for sequential and compositional processing (section 3.4) is for sure scientifically very interesting. The ordering of the codebook can easily be exploited to arrive at more robust representations of sequences or other compositional inputs. It is obvious that utilizing the ordering in such manner can also directly profit of U-Matrix methods or dynamical lattices.

[2]Note that we do not want to make any claim on the computational efficiency of SOMs for optimization problems.

Chapter 5

Conclusion

A more critical approach to using a Kohonen SOM is to be preferred. Often the TPP is acknowledged and described as a useful property in SOM studies but is not utilized in the end.

On the other hand, the TPP allows easy monitoring and visualization of its processes. Moreover there is still and there will be much debate about the qualitative reasons for using a SOM; for example it may just turn out to be a *fast* algorithm…

Anyhow its present popularity predicts good prospects for the SOM.

Chapter 6

References

Aarts E.H.L., Stehouwer H.P. , the Travelling Salesman Problem and the Self-organizing Map, proc. ICANN-93 Amsterdam, pp. 950-955, 1993

Anderson J.A. Neurocomputing: foundations of research, MIT Press 1988

Aleksander I. , Neural Computing Architectures: the design of brain- like machines, North Oxford Academic 1989

Angeniol B., de la Croix Vaubois G., le Texier J.Y. , Self-Organizing Feature Maps and the Travelling Salesman Problem, Neural Networks 1, pp. 289-293, 1988

Blackmore J., Miikkulainen R. , Incremental Grid Growing: Encoding High-Dimensional Structure into a Two-Dimensional Feature Map, proc. ICNN-93 San Francisco, pp. 450-455, 1993

Baruah A.B., Atlas L.E., Holden A.D.C. , Kohonen's Feature Maps Applied to Ordered Clustering Applications, proc. IJCNN-91 Singapore, pp. 597-599, 1991

Bauer H., Pawelczik K. , Quantifying the neighborhood preservation of Self-Organizing Maps, IEEE Trans. Neural Networks, 1992

Behme H., Brandt W.D., Strube H.W. , Speech Recognition by Hierarchical Segment Classification, proc. ICANN-93 Amsterdam, pp. 416-419, 1993

Bradburn D.S. , Reducing Transmission Error Effects Using a Self- Organizing Network, proc. IJCNN-89 Washington D.C., pp. II/531-537, 1989

Cherkassky V., Lari-Najafi H. , Self-Organizing Neural Network for Non-Parametric Regression Analysis, Proc. INNC-90 Paris, pp. 370-374, 1990

Cherkassky V., Lee Y., Lari-Najafi H. , Self-Organizing Network for Regression: Efficient Implementation and Comparative Evaluation, proc. IJCNN-91 Seattle, pp. I/79-84, 1991

De Haan G.R., Egecioglu O. , Feature Maps for Input Normalization and Feature Integration in a Speaker Independent Isolated Digit Recognition System, proc. IJCNN-92 Baltimore, pp. III/677-682, 1992

Favata F., Walker R. , A study of the application of Kohonen-type neural networks to the Travelling Salesman Problem, Biol. Cybern. 64, pp. 463-468, 1991

Fritzke B. , Let it Grow - Self-Organizing Feature Maps with Problem Dependent Cell Structure, ICANN-91 Espoo, pp. 403-408, 1991a

Fritzke B. , Unsupervised Clustering with Growing Cell Structures, proc. IJCNN-91 Seattle, pp. II/531-536, 1991b

Fritzke B. , Wilke P. FLEXMAP - A Neural Network for the Travelling Salesman Problem With Linear Time and Space Complexity, proc. IJCNN-91 Singapore, pp. 929-934, 1991c

Gemello R., Lettera C., Mana F., Masera L. , Self Organizing Feature Maps for Contour Detection in Videophone Images, proc. ICANN-91 Espoo, pp. 1305-1308, 1991

Ghaziri H. El , Solving Routing Problems by a Self-Organizing Map, proc. ICANN-91 Espoo, pp. 829-834, 1991

Graf D.H., Lalonde W.R. , A Neural Controller for Collission Free Movement of General Robot Manipulators, proc. ICNN-88 San Diego, 1988

Graf D.H., Lalonde W.R. , Neuroplanners for Hand/Eye Coordination], proc. IJCNN-89 Wash. D.C., pp. II/543-548, 1989

Hemani, Postula , Cell Placement by Self-Organization, in: Neural Networks vol. 3, pp. 377-383, 1990

Kangas J., Kohonen T., Laaksonen J., Simula O., Venta O. , Variants of Self-Organizing Maps, IEEE Trans. Neural Networks, p.93-99, 1990

Kangas J. , Temporal Knowledge in Locations of Activations in a Self-Organizing Map, proc. ICANN-92, pp. 117-120, 1992a

Kangas J., Torkkola K., Kokkonen M. , Using SOMs as Feature Extractors for Speech Recognition, proc. ICASSP-92 San Francisco, 1992b

Kangas J. , On the Analysis of Pattern Sequences by Self-Organizing Maps, Dr. Tech. thesis, Helsinki University of Technology, 1994

Knudsen E.I., du Lac S. , Esterly S.D. Computational maps in the brain, Annual Review of Neuroscience 10:41-65, 1987; Reprinted in [Anderson88]

Kohonen T. , Self-organized Formation of Topologically Correct Feature Maps, Biological Cybernetics 43, pp. 59-69, 1982

Kohonen T. , Self-Organization and Associative Memory, 3d ed. (1st ed. 1984), Springer Berlin-Heidelberg-New York 1989a

Kohonen T. , Speech Recognition based on Topology-preserving Neural Maps, in: [Aleksander89], 1989b

Kohonen T. , The Self-Organizing Map, proc. IEEE, vol. 78-9, September 1990a

Kohonen T., Raivo K., Simula O., Venta O., Henriksson J. , An Adaptive Discrete Signal Detector based on Self-Organizing Maps, proc. IJCNN-90 Washington D.C., pp. II/249-252, 1990b

Kohonen T. , The Hypermap Architecture, proc. ICANN-91 Espoo, pp. II/1357-1360, 1991

Kolb B., Whishaw I. , Fundamentals of Human Neuropsychology, Freeman New York 1990

Lampinen J., Oja E. , Fast Self-Organisation by the Probing Algorithm, IJCNN-89 Washington D.C., pp. II/503-507, 1989

Lampinen J., Oja E. , Clustering Properties of Hierarchical Self- Organizing Maps, Journal of Mathematical Imaging and Vision 2, pp. 261-272, 1992

Maggioni C., Wirtz B. , A Neural Net Approach to 3-D Pose Estimation, proc. ICANN-91 Espoo, pp. 75-80, 1991

Martinetz T., Schulten, K. , A 'Neural-Gas' Network Learns Topologies, proc. ICANN-91 Espoo, pp. 397-401, 1991

Martinetz T. , Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps, proc. ICANN-93 Amsterdam, pp. 427-434, 1993

Menard F, Fogelman Soulie F. , Application of the Topological Maps Algorithm to the Recognition of Bi-dimensional Electrophoresis Images, proc. INNC-90 Paris, pp. 99-102, 1990

Miikkulainen, R. , Self-Organizing Process Based on Lateral Inhibition and Synaptic Resource Distribution, ICANN-91 Espoo, pp. I/415-420, 1991

Miikkulainen, R. , Trace Feature Map: a model of episodic associative memory, Biol. Cybern. 66, pp. 273-282, 1992

Miikkulainen, R. , Subsymbolic Natural Language Processing, MIT Press Cambridge-London, 1993

Najand S., Lo Z-P., Bavarian B. , Using Kohonen Topology Preserving Mapping Network for Learning the minimal environment representation, proc. IJCNN-92 Baltimore II/87-93, 1992

Naylor J.A., Rossen M.L. , Neural Network Word/False Alarm Discriminators for Improved Keyword Spotting, proc. IJCNN-92 Baltimore, pp. II/296-300, 1992

Ritter, H., Schulten K. , Convergence properties of Kohonen's topology preserving maps: fluctuations, stability and dimension selection, Biol. Cybern. 60, pp. 59-71, 1988

Ritter H., Kohonen T. , Self-organizing Semantic Maps, Biol. Cybern. 61, pp. 241-254, 1989

Ritter H., Kohonen T. , Learning Semantotopic Maps from Context, proc. IJCNN-90 Wash. D.C., pp. II/543-548, 1990

Ritter H., Martinetz T, Schulten K. , Neural Computation and Self- Organizing Maps, Addison-Wesley Reading Mass., 1992

Rodrigues J.S., Almeida L.B. , Improving the Learning Speed in Topological Maps of Patterns, proc. INNC-90 Paris, pp. 813-816, 1990

Saxon J.B., Mukerjee A. , Learning the Motion Map of a Robot Arm with Neural Networks, proc. IJCNN-90 San Diego, pp. II/777-782, 1990

Scholtes J.C. , Kohonen Feature Maps in Natural Language Processing, TR CL-91-01 dept. of Computational Linguistics, University of Amsterdam, 1992

Sirosh J., Miikkulainen R. , How Lateral Interaction Develops in a Self-Organizing Feature Map, ICNN-93 San Francisco, pp. 1360-1365, 1993

Torkkola K., Kokkonen, M. , Using the Topology-Preserving Properties of SOFMs in Speech Recognition, proc. ICASSP-91, pp. 261-264, 1991

Toth G.J., Lorincz A. , Genetic Algorithm with Migration on Topology Conserving Maps, proc. ICANN-93 Amsterdam, ppp. 605-608, 1993

Tryba V., Goser K. , Self-Organizing Feature Maps for Process Control in Chemistry, proc. ICANN-91 Espoo, pp. I/847-852, 1991

Ultsch A. , Self Organized Feature Maps for Monitoring and Knowledge Acquisition of a Chemical Process, proc. ICANN-93 Amsterdam, pp. 864-867, 1993

Vleugels J.M., Kok J.N., Overmars M.H. , A Self-Organizing Neural Network for Robot Motion Planning, proc. ICANN-93 Amsterdam, pp. 281-284, 1993

Zhang C-H., Mlynski D. , Neural Somatotopical Mapping for VLSI Placement Organization, IJCNN-91 Singapore, pp. 862-868, 1991

Zrehen S. , Analyzing Kohonen Maps with Geometry, proc. ICANN- 93 Amsterdam, pp. 609-612, 1993