Interpreting Topology Preserving Networks

Jürgen Rahmel[†] and Thomas Villmann[‡]

[†]Centre for Learning Systems and Applications
[‡]Institute for Industrial Mathematics
University of Kaiserslautern
PO Box 3049, 67653 Kaiserslautern, Germany
e-mail:*rahmel@informatik.uni-kl.de*

Abstract

In this report, we first propose a dichotomy of topology preserving network models based on the degree to which the structure of a network is determined by the given task. We then look closer at one of those groups and investigate the information that is contained in the graph structure of a topology preserving neural network. The task we have in mind is the usage of the network's topology for the retrieval of nearest neighbors of a neuron or a query, as it is of importance, e.g., in medical diagnosis systems. In general considerations, we propose certain properties of the structure and formulate the respective expectable results of network interpretation. From the results we conclude that both topology preservation as well as neuron distribution are highly influential for the network semantics. After a short survey on hierarchical models for data analysis, we propose a new network model that fits both needs. This so called SplitNet model dynamically constructs a hierarchically structured network that provides interpretability by neuron distribution, network topology and hierarchy of the network layers. We present empirical results for this new model and demonstrate its application in the medical domain of nerve lesion diagnosis. Further, we explain a view how the interpretation of the hierarchy in models like SplitNet can be understood in the context of integration of symbolic and connectionist learning.

1 Introduction and Motivation

Topology preserving neural networks are a widely ranged type of neural vector quantizers which are used for example in data visualization, feature extraction, principle component analysis, image processing, classification tasks, robotics and other. Since the early works from Willshaw and v.d. Malsburg [WdM76] and Amari [Ama80] several models were introduced; the most known of them are developed by Kohonen [Koh84], Luttrell [Lut88], Linsker [Lin89] and Martinetz [MS94]. In the present paper the authors focus onto the discussion of the properties and the interpretation of the Self–Organizing Map (SOM) of Kohonen [Koh95] and some of its relations to the Topology Representing Network (TRN) of Martinetz.

1.1 Basic Models

Both networks, the SOM and the TRN, map data vectors ω of a data set $V \subseteq \mathbb{R}^k$ onto a set A of neurons i which is formally written as

$$\Psi_{V \to A} : V \to A \ . \tag{1.1}$$

For each neuron there is an associated pointer $\mathbf{w}_i \in \mathbb{R}^k$ which all form the set $\mathbf{w} = \{\mathbf{w}_i\}_{i \in A}$ of weight vectors. The mapping description of (1.1) is a winner take all rule, i.e. a stimulus vector $\omega \in V$ is mapped onto that neuron $s \in A$ the pointer $\mathbf{w}_s$ of which is closest to ω,

$$\Psi_{V \to A} : \omega \mapsto s(\omega) = \underset{i \in A}{argmin} \|\omega - \mathbf{w}_i\| \ , \tag{1.2}$$

called 'winner' neuron or best matching unit (*bmu*). The set

$$R_i = \left\{ \omega \in \mathbb{R}^d \mid s(\omega) = \underset{i \in A}{argmin} \|\omega - \mathbf{w}_i\| \right\} \tag{1.3}$$

forms the receptive field of the neuron i and $\tilde{R}_i = R_i \cap V$ is called the masked receptive field. However, in the SOM the neurons are arranged *a priori* on a fixed grid[1] whereas in the TRN the topological structure of A is a object of learning itself. This is realized by different dynamics for the adaptation of the weight vectors during the learning process: a stimulus ω is randomly presented to the network according to a certain data distribution $P(V)$. In the case of the SOM we have as the adaptation rule

$$\triangle \mathbf{w}_i = \epsilon h_{SOM}(\omega, \sigma)(\omega - \mathbf{w}_i) \tag{1.4}$$

whereby the neighborhood function is usually of Gaussian shape: $h_{SOM}(\omega, \sigma) = \exp\left(-\frac{\|i - s(\omega)\|_A^2}{2\sigma^2}\right)$ evaluated in the output space A. For the TRN we get a similar rule

$$\triangle \mathbf{w}_i = \epsilon \exp\left(-\frac{k_i(\omega, \mathbf{w})}{\lambda}\right)(\omega - \mathbf{w}_i) \ . \tag{1.5}$$

However, here the neighborhood function $h_{TRN}(\omega, \lambda) = \exp\left(-\frac{k_i(\omega, \mathbf{w})}{\lambda}\right)$ is computed in the input space V with $k_i(\omega, \mathbf{w})$ gives the number of pointers $\mathbf{w}_j$ for which the relation $\|\omega - \mathbf{w}_j\| \leq \|\omega - \mathbf{w}_i\|$ is valid [MBS93]. The parameter ϵ plays the role of a learning parameter whereas σ and λ describe the range of neighborhood in the SOM and TRN, respectively.

Several variants of these algorithms have been established in the recent years. An expansive overview is given in [Koh95]. Here we only remark that the structure of the network can be an objective of learning itself, i.e. in addition to the learning dynamic of the weight vector one has a learning strategy for the adaptation of the number of neurons and the topological structure of the network. For the SOM we refer to [Fri93a], [Häm95] and [BV96], for the TRN it is published

[1] Usually the grid A is chosen as a d_A–dimensional hypercube and then one has $i = (i_1, \ldots, i_{d_A})$. Yet, other ordered arrangements are also admissible [KKL90]. If it is not specified otherwise we assume the rectangular arrangement.

in [Fri95]. Section 3.4 introduces a network model that in addition constructs a *hierarchy* in the network during the training phase, thus providing a means of gaining structured knowledge.

Depending on the specification of the given task and the kind of topology preserving model that is used, we can distinguish two different approaches to the application of such network models. On one hand, the network interpretation might be prespecified, so a particular network topology is fixed for the training. On the other hand, the development of the structure of the network might have more degrees of freedom. Then, suitable principles and algorithms are needed to extract knowledge from the trained network. The following subsections will further explain those views and demonstrate the position of this report in the framework.

1.2 Prespecified network interpretation

If a network is to be applied for a task, where the required interpretation of the training result is given by the user, the degrees of freedom for the network are quite restricted. Examples for this setting are:

- Visualization of satellite data in colour plots $\Rightarrow$ mapping of high-dimensional data onto a color cube (e.g. [GS93])
- Learning of control data for a robot arm $\Rightarrow$ mapping onto a 3-d network (e.g. [RMS91])
- Visualization of data features $\Rightarrow$ projection of principal components onto a 2-d network
- Quantization of the input signal (e.g. [NY88])

This list could be further extended and examples of these approaches are numerous. The point we want to make about these approaches to the application of topology preservation is that the nature of these applications restrict the degrees of freedom for the definition and training of network models. The design choices are confined with respect to:

- Network topology, type of lattice
- Network dimension
- Number of neurons
- Choice of similarity measure
- Choice of training samples

The question that is to be answered when using this appraoch is: how well is the specified interpretability achieved for the trained network? In other words, it has to be checked, how the constraints given above lead to a satisfying training result. While the network's topology, dimension and often the number of neurons is mostly determined by the task, there is a range of choice for the similarity measure for the training vectors in input space. The image registration procedure in [HRW96] or the similarity based clustering [Rah95] are applications with a specially defined similarity measure for training vectors, in the former there is also a selection of training samples by indentification of regions of interest. Ways of evaluating the training result are for example:

- Visual inspection by domain experts and comparison with network results (applicable in restricted areas only)
- measurement of resulting performance, i.e. classification, control or reconstruction accuracy
- Measurement of topology preservation

1.3 Interpreting a trained network

In contrast to the preceding observations, the growing interest in dynamical and hierarchical network models implies the need to interpret a network structure that developed during the training procedure instead of being fixed at the beginning. The Neural-Gas algorithm learns positions and topological connections of a given number of reference vectors while the Competitive Hebbian Learning rule [Mar93] uses prepositioned neurons to add the topology preserving connections, possibly masked by the given input distribution. The Growing Cell Structures (GCS) [Fri93b] use simplices with a fixed dimension for vector quantization and dimension reduction with a variable number of neurons. The SplitNet model (cf. Sec. 3.4) dynamically adapts the number of neurons while including topological connections to the network structure and it builds a hierarchical structure over the neuron set.

All these models have in common that a training algorithm forms a network topology, which is not fixed in advance and that has to be interpreted in order to extract the trained knowledge. In general, we observe the following degrees of freedom:

- Network size, i.e. number of neurons
- Position of neurons, distribution of weight vectors
- Network dimension
- Network topology, connections between neurons
- Hierarchy of neuron sets

The dynamic models self-develop with respect to these aspects and the interpretation component has to combine external knowledge (about the principal properties of a training algorithm and the expected results) and internal knowledge (stored in the actually trained network) to make best use of the information given through the training data. For the extraction of knowledge from the network structure, we can note various open questions including:

- What is the semantics of the neuron distribution?
- What is expressed by the network topology? How can it be used?

- How good is the topology preservation? How does this influence the interpretation of the topology?
- What is the semantics of the hierarchical structure?

This report is aimed to provide a step towards the answers of the above questions. Further insight might help to develop new training algorithms or to improve existing ones. If, for example, the connection between the neuron distribution and the training data distribution is fully understood and controllable, we might construct training algorithms that influence the neuron positions corresponding to various criteria like, e.g., misclassification rates or given cost functions. The gain would be the representation of different concepts by a single distance measure and the modified neuron distribution. For example, incorporating misclassification costs in the distance measure between vectors of different classes moves class borders towards class centers with higher costs (the class is proposed, only if the similarity to class prototypes is large enough). These modified decision regions can be approximated by a modified neuron distribution together with an unmodified (e.g. euclidean) distance measure. Thus, local estimates of neuron distribution may replace global knowledge of misclassification costs and different distance measures, a method that might be helpful regarding the scaling problems of many of the learning algorithms, both connectionist and symbolic.

An important tool for the measurement of topology preservation was developed in [VDHM94], [VDHM96]. We will use it to evaluate the construction of topology and take it as a basis for the discussion on the interpretation of the network's topology. We will demonstrate how the connections in the network support retrieval tasks, a class of problems that occur in many different fields of applications.

Another aspect is the meaning and utility of the hierarchy, as it is developed by the SplitNet model. It has to be investigated, how this approach can be compared to other hierarchical or tree structured methods and what benefits for training and explanation tasks are to be expected by adding this conceptual dimension to a network.

1.4 Contents of this report

This report will not deal with the problems of convergence of any particular network training algorithm. We will not go into the questions of the existence and uniqueness of the solution for the stochastic processes involved or the time and space complexity for the cited algorithms. The aim of this report is to show the possibilities of network interpretation when neuron distribution and topology preservation are degrees of freedom in the network training.

Section 2 will investigate the measurability of topology preservation and the efficiency of retrieval algorithms with various degrees of search depths. We introduce a heuristic search algorithm that improves the retrieval by orders of magnitude. In Sec. 3, we discuss hierarchical and tree structured models for data analysis and introduce a new neural network model called SplitNet. We show the basics of its training algorithm and present simulation results. The description of an application for medical diagnosis tasks finishes this section. The interpretation of hierarchical structures is discussed in Sec. 4, where we again look at the models described in the preceding section and compare them with respect to the information content in the hierarchical structure.

2 Interpretation of Topology

The scenario we will deal with throughout this section is the m-nearest-neighbor approach to classification. The problems are (i) to find the number and positions of neurons that are useful and efficient for the given data and (ii) to retrieve a set of m nearest neighbors for a presented query q that is to be classified. This number m is not necessarily known a priori. The need to increase the number of visited neighbors may arise with the results of an already conducted analysis. Consider for example medical diagnosis (see Sec. 5), where the search for nearest neighbors of a query is needed to determine the class of the query. If the class assignment is unique, only very few neighbors are sufficient to ensure the correct result. If, on the other hand, inspection of the first neighbors leads to a tie between two or more possible classes, a decision cannot be made. In this case there is a need to visit more neighbors until the suggestion of the network is clear. Thus, in order to be able to continue search instead of starting from scratch each time m is augmented, it is desirable to develop incremental retrieval algorithms.

After introducing some notations, we present a systematic approach of interpreting a given network topology with respect to retrieval problems. Disregarding the construction of a graph G in this chapter, we will postulate some properties for this graph and investigate the possible retrieval algorithms. This will clarify the requirements a network model has to fullfil in order to be applicable for the retrieval task. In Sec. 3.4, we then propose a new hierarchical network model that fits those needs.

2.1 Terms and notations

For the formal description of the network structure and its properties, we transform a given network into a graph $\mathcal{G} = (\mathcal{V}; E)$, where the set $\mathcal{V} = \{v_1, \dots, v_n\}$ of nodes is given by positions $\mathbf{w}_1, \dots, \mathbf{w}_n$ of the neurons $N_1, \dots, N_n$ of the network. The topology of the network is represented by the set E of edges, where $e_{ij} \in E$ iff N_i and N_j are direct neighbors in the network topology. By $E_i := \{e_{ij} \in E\}$ we denote the set of edges starting at v_i and $\mathcal{V}_i := \{v_j \mid e_{ij} \in E_i\}$ is the set of direct neighbors of a node v_i. As indicated above, knowledge is contained in the topology, representing spatial relations of the data points. We denote the distance between two points x and y in data space by $d(x, y)$, assuming $d(.,.)$ to be a distance measure defined according to the needs of the domain. For visualization purposes, $d(.,.)$ will be the euclidean distance in the figures below.

The problem for a dataset $D = \{d_1, \dots, d_N\}$ with records $d_i \in \mathbb{R}^k$ is now to find correct and complete retrieval algorithms for any given graph G and a query $q \in \mathbb{R}^k$. The query might be a newly encountered problem that is to be solved with the help of the nearest neighbors stored so far in the network. We now consider the retrieval result for a query q. The nodes $v_{j_1}, \dots, v_{j_m}$ retrieved as first m neighbors for the query are renamed to $l_1, \dots, l_m$, being members of the retrieval list $L \subset \mathcal{V}$. Such a list $L = (l_1, \dots, l_m)$ with $\{l_1, \dots, l_m\} \subseteq \mathcal{V}$ of nearest neighbors to the query is said to be *correct*, if $d(q, l_i) \leq d(q, l_j)$ holds for all $j > i$. The list L is said to be *complete*, if there exists no $v_j \notin L$ with $d(q, v_j) < d(q, l_i)$ for any $l_i \in L$. A retrieval algorithm is said to be correct (complete) if the retrieved list L is correct (complete). The correctness is only a matter of sorting the list, but for incremental retrieval algorithms, i.e., where the m+1-st neighbor is determined with the m previously found list elements, completeness at every step m

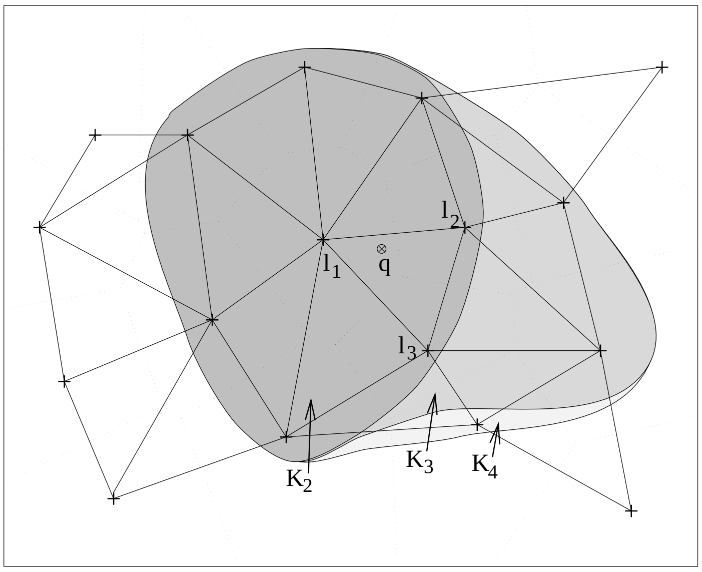


Figure 1: A sample data set, perfect topology preservation and the candidate sets constructed by algorithm RETRIEVE$_{cp}$

implies the correctness of the retrieval.

2.2 Complete storage of data

In this section, we assume a complete storage of data records in the graph $\mathcal{G}$, i.e., $\mathcal{V} = D$. Thus, for each data vector in the training set, there exists a neuron that uniquely represents this vector. This is a highly idealized assumption, because in real world applications there will nearly always be too much data to be stored exhaustively in easy accessible memory structures, but we start from this point in order to clarify the optimal situation and to be able to evaluate the additional retrieval effort that has to be done when weaker properties are considered.

2.2.1 Perfect topology preservation

For the graph structure a perfect topology preservation, which means that $e_{ij} \in E \Leftrightarrow R_i \cap R_j \neq \emptyset$ where R_i denotes the Voronoi region of node v_i (cf. equation (1.3) for the definition of the Voronoi region). Thus, the graph corresponds to the so called Delaunay triangulation of the set $\mathcal{V}$ with respect to the used distance measure.

Figure 1 shows a sample data set of two dimensional vectors (crosses) and the set of edges (lines). Given a query q, retrieval of the m nearest neighbors of q (which then constitute the list L) can be done as follows. First, determine the nearest node and let it be l_1, the first element in the list L. This can be done very fast by using an external search tree [Kel91] or by using an inherent hierarchical structure of the network (cf. Sec. 3.4).

With the best neighbor known, build the candidate set K for the next nearest neighbor, consisting of all direct neighbors of elements in L. Choose the node in K which is closest to the query q. Repeat this process until $|L| = m$, where $|.|$ denotes the cardinality of the list. The step that creates the candidate set explicitly uses the perfect topology preservation of the graph $\mathcal{G}$. By this, the algorithm also uses the information that is contained in the distance measure that is needed to create the Voronoi regions.

This retrieval algorithm RETRIEVE$_{cp}$ can be formalized as follows[2] :

1. Let s := 1.
2. Determine l_1, let $L_s := (l_1)$.
3. Let $K_{s+1} := (\bigcup_{\{i|l_i \in L_s\}} \mathcal{V}_i) \setminus L_s$.
4. Choose $v_p \in K_{s+1}$ with $d(v_p, q) \leq d(v_j, q)$ for all $v_j \in K_{s+1}$.
5. Let $l_{s+1} := v_p$, $L_{s+1} := (L_s, l_{s+1})$.
6. If $s + 1 = m$, then return L_{s+1}, STOP.
7. Let $s := s + 1$, goto step 3.

This algorithm is inherently of incremental nature. The elements of the list L and the current candidate set are used for determining the next nearest neighbor. For any number m the following statement holds:

Theorem 2.1
The retrieval algorithm RETRIEVE$_{cp}$ is correct and complete.

The proof of this and the following theorems can be found in Appendix A. The computational effort of this retrieval algorithm is in the worst case, even for the highly idealized assumptions of this section, quite high. Let c be the average number of edges leading from a node to its neighbors. For the search of m nearest neighbors to a query we have to determine the best neighbor, e.g. by a tree search procedure, and then successively grow the candidate set. Thus, at step s, the number of visited nodes is estimated by

$$logN + c + 2c + \cdots + (s-1)c = logN + c\sum_{i=1}^{s-1} i = logN + \frac{1}{2}(s^2 - s)c. \tag{2.1}$$

Now, since the nodes in the surrounding of a point tend to have common neighbors, this estimation is likely to be unrealistic in the average case. Figure 2 shows the worst case with an average connectivity $c = 5$ compared to the real number of visited nodes for a set of 400 points randomly drawn from a square, when the 40 nearest neighbors were determined incrementally. It can be seen, that the number of necessary distance calculations grows almost linearly, with an incline of less than 2.

As stated at the beginning of this section, the postulated graph properties correspond to the construction of the Voronoi tesselation or equivalently the Delaunay triangulation of the given data space. As this is computationally intractable for realistic data dimension ($k > 20$), we will now investigate weaker properties.

[2]The index cp means complete storage, perfect topology preservation

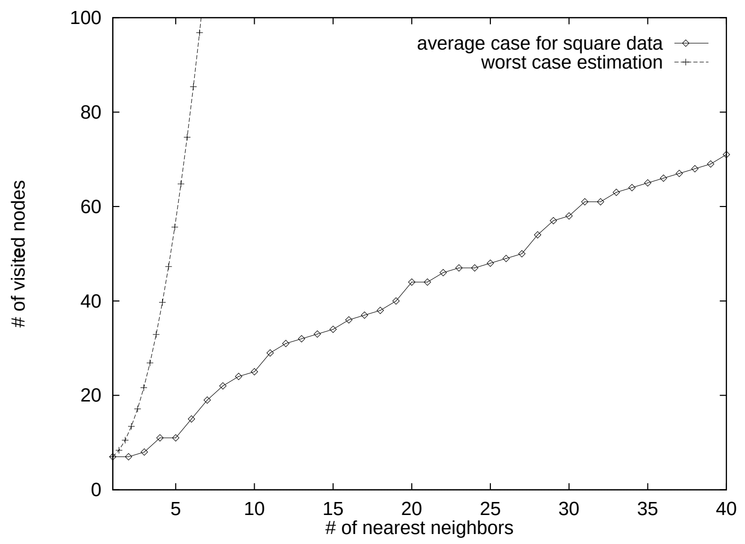


Figure 2: Worst case and real values for the algorithm RETRIEVE$_{cp}$

2.2.2 Imperfect topology preservation, a first approach

Imperfect topology preservation means that the set E of edges in $\mathcal{G}$ now is only a subset of the complete Delaunay triangulation. The positions in the graph, where a node v_i is not connected to node v_j even though $R_i \cap R_j \neq \emptyset$ holds, are called *topological defects*. They strongly influence the complexity of the retrieval process. In the course of this report, we will concentrate on only one direction of the mapping process considered for the measurement of topology preservation, namely the mapping from input vectors onto the network topology. A rigorous discussion of general case can be found in [VDHM96].

For measuring those defects, we use a modification of the topographic function Φ described in [VDM94]. A function $f_i(t)$ counts the number of nodes v_j, the Voronoi regions of which are adjacent to the region of v_i but which are exactly t steps away from v_i in the graph $\mathcal{G}$:

$$f_i(t) = |\{j \mid \|v_i - v_j\|_{\mathcal{G}} = t\,;\; R_i \cap R_j \neq \emptyset\}|. \tag{2.2}$$

As long as $\mathcal{G}$ consists of exactly one connected component, all neighboring nodes are reachable by traversing edges in $\mathcal{G}$. Now Φ is defined as

$$\Phi(t) = \sum_i \sum_{l \geq t} f_i(l). \tag{2.3}$$

The largest $t^+ > 0$, for which $\Phi(t^+) \neq 0$ still holds, determines the size of the largest topological defect in $\mathcal{G}$. This t^+ controls the efficiency of the retrieval of the nearest neighbors. For the graph in figure 3 we have $t^+ = 2$. By $\mathcal{V}_i^t := \{v_j \mid \|v_i - v_j\|_{\mathcal{G}} \leq t\}$ we denote the set of nodes, that are reachable from v_i with at most t steps in $\mathcal{G}$.

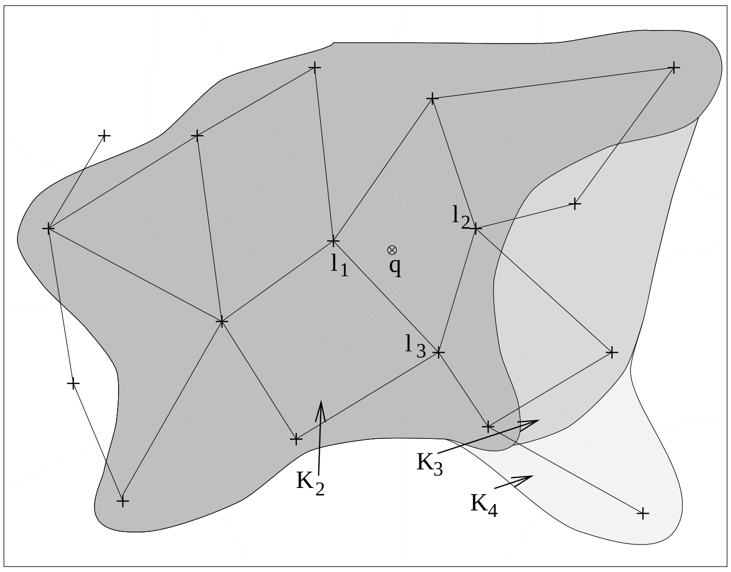


Figure 3: A sample data set, imperfect topology preservation and the candidate sets constructed by algorithm $\textsc{Retrieve}_{ci}$

The retrieval algorithm $\textsc{Retrieve}_{ci}$ for the imperfect topology preservation is very similar to $\textsc{Retrieve}_{cp}$. Only the construction of the candidate set K differs, because now t^+ steps have to be considered to include all neighboring nodes. Thus, we replace this step in $\textsc{Retrieve}_{cp}$ for $\textsc{Retrieve}_{ci}$ by:

3'. Let $K_{s+1} := (\bigcup_{\{i|v_i \in L_s\}} \mathcal{V}_i^{t^+}) \setminus L_s$.

Also for this algorithm we can state:

Theorem 2.2
$\textsc{Retrieve}_{ci}$ is correct and complete.

The problem is the efficiency of $\textsc{Retrieve}_{ci}$, because in the worst case estimation, the number of visited neighbors of a node grows exponentially (c^{t^+} for an average node connectivity $c = \overline{|E_i|}$). For the number of node visits, we can adapt equation (2.1) and get:

$$log N + c^{t^+} + 2c^{t^+} + \cdots + (s-1)c^{t^+} = log N + c^{t^+} \sum_{i=1}^{s-1} i = log N + \frac{1}{2}(s^2 - s)c^{t^+}. \tag{2.4}$$

As in the case of perfect topology preservation, the real number of visited nodes during the retrieval of nearest neighbors is expected to be much lower than this worst case estimation. Again, there is a kind of saturation of the candidate set and its size grows slower than the formula above indicates. Figure 4 shows sample curves for different values of t^+ compared to the worst case.

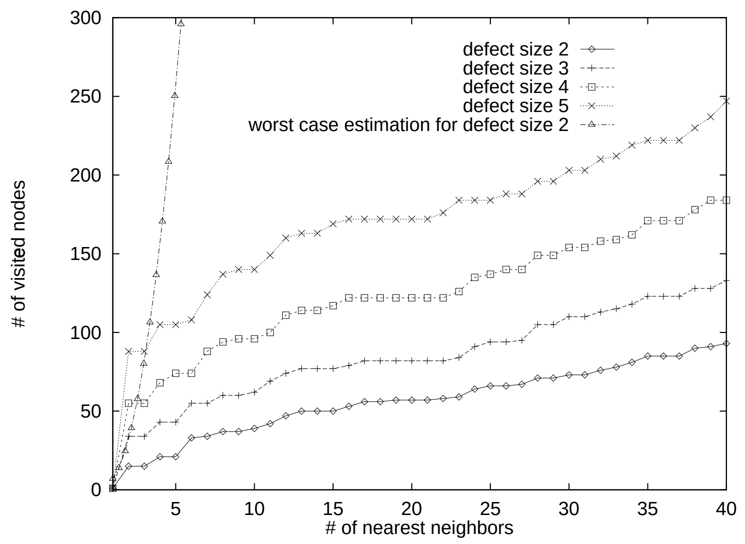


Figure 4: Worst case and real values for the algorithm RETRIEVE$_{ci}$ with different values of the defect measure t^+

It can be observed from the figure that for larger numbers of nearest neighbors to be retrieved, the incremental search effort rises almost linearly. The drawback is the fact that for higher values of t^+ there is a large increase at the beginning, i.e. relatively many nodes have to be visited in order to find the first few nearest neighbors. The retrieval effort for the first three neighbors with $t^+ = 5$ is nearly 6 times as high as for $t^+ = 2$.

For the Kohonen network we can have quite large values for t^+, compared to the overall number of nodes, if a dimension reduction takes place during the training (cf. section 3.4.6). Thus we conclude that an important point for the applicability of a topology preserving network model for retrieval purposes is the ability of the model to keep the topological defects in a very restricted range. As for most of the applications the inherent structure of the data space is not known a priori, this fact implies that the topology of the network must be constructable and changeable during the training process.

In order to tackle this problem from the theoretical side and to increase the efficiency for imperfectly topology preserving networks we developed a search heuristic that uses precalculated lengths of the edges in the graph to exclude certain nodes from the distance calculation during the retrieval phase.

2.2.3 Imperfect topology preservation, an optimized approach

Extending the naive and exhaustive approach described in the last section, we now introduce an improved version of the retrieval algorithm that is able to avoid nearly all of the node visits in the far neighborhood of the list elements. The advantage at retrieval time has to be paid for by additional effort during generation time of the network but the extra work has to be done only once and can then be used for any retrieval operation in the network structure.

So far, the edges in the graph were used only for determination of the topologic neighbors of nodes in the graph $\mathcal{G}$. Now, we will associate some information with those edges. After the training of the network is finished, we run once through the edge set and compute the length of every edge. These additional distance calculations (and the additional strorage space for this length information) are the only effort we have to make. Now, the optimized algorithm can use these distances together with the distances already computed for some of the nodes to replace possibly expensive distance calculations for nodes (representing complex cases) by inexpensive addition and comparison of real numbers.

Figure 5 explains the use of the edge lengths in the optimized algorithm. On the left, it shows a little section of a network structure. Suppose that the first three neighbors to the query q were already found ($L = (l_1, l_2, l_3)$) and the distances $d(q,1), \ldots, d(q,5)$ are already known. If the fourth neighbor is to be determined and t^+ is supposed to be greater than 2, the exhaustive algorithm RETRIEVE$_{ci}$ would have calculated the distances $d(q,6)$ and $d(q,7)$. For the example $c = d(q,6)$, we demonstrate, that this calculation is superfluous.

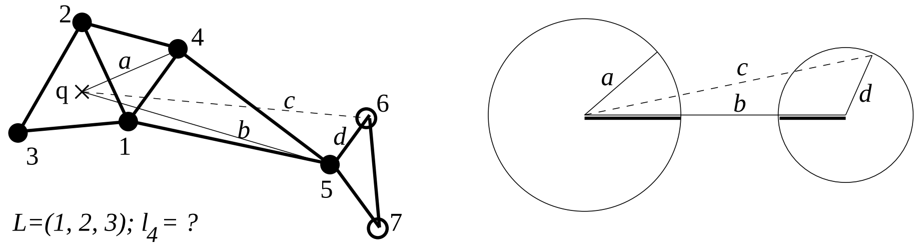


Figure 5: The heuristic of the optimized retrieval algorithm is explained for the determination of the fourth nearest neighbor (see text). Left: a section of the network topology. Right: distance calculation for c can be suspended, because it cannot be shorter than a.

The heuristic applied in this case is the preference of short distances between list elements and candidates. Thus, the edge length also determines the search order through the set of neighbors of a node. For the situation in Fig. 5 this means that node 5 and its neighbors are checked after node 4, which is closer to the list elements than node 5. Now, when the algorithm comes to check node 5 and its neighbors, the currently known best distance for the fourth neighbor is $a = d(q,4)$. The base distance $b = d(q,5)$ for this case is already known and greater than a. Node 6 is a possible candidate, but we can use the already known distances to decide that c is not to be calculated now. As we have $a < b - d$ and $c \geq b - d$ we can conclude $a < c$ without knowing the actual value of $c = d(q,6)$. If further neighbors of node 6 are to be considered, we can use $b_{new} = b - d$ as the new base distance. This strategy saves many of the useless distance calculations in the farther neighborhood of the list elements. The exhaustive search still ensures the correctness and completeness of the retrieval, but Fig. 6 shows, how the number of distance computations is reduced by the optimization. In the figure, we compare the case of $t^+ = 2$ with $t^+ = 5$, both are shown with and without the heuristic approach. The computational effort for the weaker topology preservation is reduced to nearly the rate for the stronger one. Even for the critical case of retrieving only a few nearest neighbors, the optimized algorithm shows its power of saving computation effort.

We can formalize the optimized algorithm RETRIEVE$_{ci,opt}$ as follows:

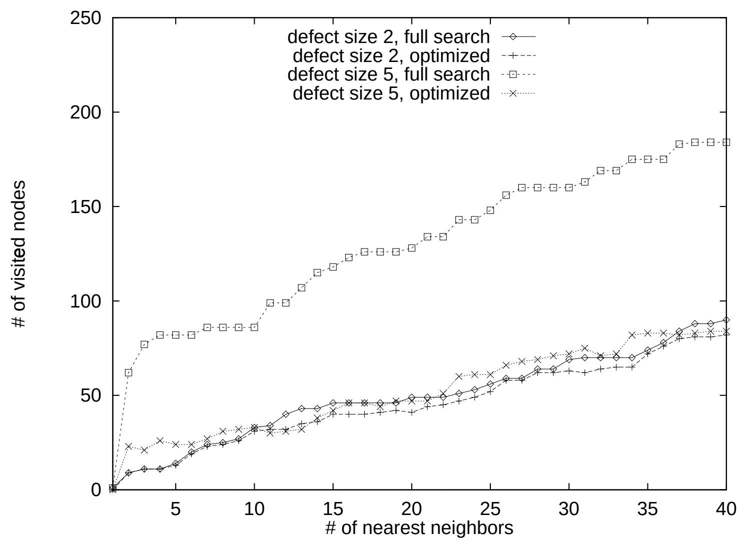


Figure 6: Comparison of the exhaustive and the optimized approach to retrieval with two different values for the defect measure t^+

1. Let $s := 1$.
2. Determine l_1, let $L_s := (l_1)$.
3. Let $d_{min} := \infty$. Let $l_{next} := nil$.
4. For all $l_i \in L_s$:
   - Let $d_{base} := d(q, l_i)$.
   - $checkNeighbors(l_i, t^+, d_{base})$
5. Let $L_{s+1} := (L_s, l_{next})$.
6. If $s + 1 = m$, then return L_{s+1}, STOP.
7. Let $s := s + 1$, goto step 3.

Like the algorithms before, at step s RETRIEVE$_{ci,opt}$ runs through a set of candidates to determine the $s + 1$-st nearest neighbor. But now, instead of collecting an increasing set of candidates to select the next list element from, the algorithm stores information in the nodes themselves and keeps track of the best choice. Starting from each of the list elements retrieved so far, a search with depth t^+ is executed to judge the mere possibility of a node to be a successful candidate.

We rather informally present now the procedure *checkNeighbors* that is called recursively to run through all neighboring nodes of a list element with the required search depth t^+, marking a node as rejected at step s, if the distance calculation can be suspended for the above explained reasons. Otherwise, the distance to the query is to be determined and is stored in the node.

In both cases, the search continues at the currently examined node until the needed depth is reached.

Algorithm $checkNeighbors(currentNode, depth, baseDistance)$:

```
1. If depth = 0, then return.
2. Let $K_{sort}$ := sorted list of neighbors for currentNode (with respect to the edge lengths).
3. For all $K_i \in K_{sort}/L_s$
    • Let $newBase := baseDistance - d(currentNode, K_i)$.
    • If $K_i$ was marked rejected during loop $s$ of RETRIEVE$_{ci,opt}$
      then:
        $checkNeighbors(K_i, depth - 1, newBase)$.
      else:
        If $newBase < d_{min}$
        then:
          Compute $d(K_i, query)$, if not already known.
          If $d(K_i, query) < d_{min}$
          then:
            Let $d_{min} := d(K_i, query)$, $l_{next} := K_i$.
          $checkNeighbors(K_i, depth - 1, d(K_i, query))$.
        else:
          Mark currentNode as rejected in loop $s$.
          $checkNeighbors(K_i, depth - 1, newBase)$.
```

From the explanations given above it can be seen the heuristic of choosing near neighbors of list elements first in the course of the retrieval will affect the efficiency of the procedure, but not the correctness of completeness. Depending on the selection order RETRIEVE$_{ci,opt}$ will compute for more or less nodes their distance to the query, but it will never suspend the computation for a real candidate.

Theorem 2.3
RETRIEVE$_{ci,opt}$ is correct and complete.

2.3 Incomplete storage of data

This section deals with the incomplete storage of data points, a situation that will be most appropriate for application purposes, since storage capacity is limited and generalization will naturally occur. Now we have for the graph $\mathcal{G}$ that $|\mathcal{V}| < |D|$, which means that in general several data records d_i are associated with a graph node v_j. By $D_{v_j} := \{d_i \mid d(d_i, v_j) \leq d(d_i, v_p)\}$ for all $p \neq j$ we denote this set of data records mapped onto a certain node.

Figure 7 shows a sample data set (small crosses) and a distributon of graph nodes (large crosses). For now, no assumptions are made about the topology, so we omitted the graph edges

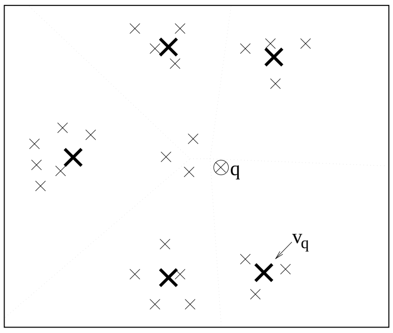


Figure 7: A sample data set with incomplete storage of data vectors

in the figure. The query q falls into the Voronoi region of the node labelled v_q, whereas the nearest data vectors remain unvisited. So q would at first be associated with the data records in D_{v_q}, which are not the ones closest to q. In fact, no algorithm can guarantee to traverse any given topology (even the perfect one) with a predefined number of steps and construct a complete list L of nearest data vectors.

Thus, in order to minimize the risk of error the distribution of the graph nodes is not arbitrary. Therefore, let us consider the *reconstruction error*

$$E_p = \int_V \left\| \mathbf{w}_{s(\omega)} - \omega \right\|^p P(\omega)\, d\omega \tag{2.5}$$

where $P(V)$ is again the distribution of the data set $V \subseteq \mathbb{R}^k$ as introduced above. A vector quantization algorithm which minimizes E_p distributes its weight vecor set $\mathbf{w} = \{\mathbf{w}_i\}_{i \in A}$ according to

$$P(V) \propto P(\mathbf{w})^{\alpha} \tag{2.6}$$

where α is the *magnification factor.* It is coupled to p–value via

$$\alpha = \frac{k}{k+p} \tag{2.7}$$

[Zad82]. Furthermore, let be Π_i denote the probability that the neuron i is the best matching unit *bmu.* The *entropy* H is then defined as

$$H = -\sum_{i \in A} \Pi_i \cdot \log \Pi_i \ , \tag{2.8}$$

and H becomes maximal if $\alpha = 1$ holds [Bra90]. However, for the SOM in the case of mapping of a one–dimensional input space onto a chain of neurons one gets $\alpha_{SOFM} = \frac{2}{3}$ [RS86][3]. For the TRN one finds in general $\alpha_{TRN} = \frac{k}{k+2}$ [MBS93], i.e. α_{TRN} only depends from the dimensionality of the data set V embedded in the $\mathbb{R}^k$. These results yield that the entropy does not become maximal and, on the other hand, the SOFM minimizes the (somewhat strange) reconstruction error $E_{\frac{1}{2}}$ and the TRN the more usual E_2. A neural vector quantizer the goal of which is to maximize H

[3]this result is valid also for higher–dimensional cases which separate

is the above mentioned network of LINSKER [Lin89]. Yet, recently in [HBD95, BDH96] a variant of the original SOM algorithm is published which allows a control of the magnification factor α_{SOFM} of the SOM. It is based on a localized learning parameter ϵ which now is defined as

$$\langle \epsilon_i \rangle = \epsilon_0 P(\mathbf{w}_i)^m \tag{2.9}$$

where the brackets $\langle \ldots \rangle$ denote the average in time. The learning rule (1.4) now is written as

$$\triangle \mathbf{w}_i = \epsilon_{s(\omega)} h_\sigma (i, \omega, \mathbf{w}) (\omega - \mathbf{w}_i) \tag{2.10}$$

which leads to the new magnification factor

$$\tilde{\alpha}_{SOM} = \alpha_{SOM} \cdot (m + 1) \ . \tag{2.11}$$

Hence, one is able to control the magnification factor $\tilde{\alpha}_{SOM}$ via the parameter m and, as the consequence, also the reconstruction error E_p from (2.5).

Additionally, since unsupervised clustering uses no class information for the training cases, minimizing an error measure like the reconstruction error is the best that can be done with the training data. If we have access to class membership information for the data vectors, we can use this additional information to control the training process of the network and minimize different criteria, that consider, e.g. the classification error or the costs of misclassification as well. The insertion rules of a dynamic network model like the SplitNet model described in Sec. 3.4 may be chosen in a way that the selected criterion in optimized by the growing network.

3 Hierarchical Models

In this section, we first give a quick survey of existing hierarchical models for data analysis, both from statistics and the neural networks field. Then, we present a new architecture, the SplitNet model, that integrates the ideas and goals of both, the non-hierarchical topology preserving maps and the hierarchically organized decision methods.

3.1 Hierarchical Cluster Analysis

The hierarchical cluster analysis, either the divisive or the agglomerative methods, are methods that progressively split or link clusters of data. The result of the method can be visualized as a dendrogramm, which is a two-dimensional tree structure that shows the order of linkage (for the agglomerative case) and the distance or similarity at which this linkage of clusters was performed. Thus, this method is able to display the clustering of data but it is not possible to reason about the real spatial relationship of the observed pattern. There is no similarity information other than the one for linked clusters. Similar statements are true of course for divisive methods. So the hierarchical clustering methods are useful tools for a preliminary analysis of the data, but they do not provide additional ways for explaining the classification result or reason on alternative solutions based on neighborhood observations.

3.2 Classification and decision trees

The classification trees [BFOS84] are a tree-structured representation of a classifier consisting of decision nodes, the nonterminal nodes in the tree, and class assignment nodes, the terminal nodes. In the decision nodes, the solution space is separated into two subspaces by testing a linear combination of attributes against a threshold. Thus, on a path through decision nodes on a branch of the tree, a sequence of hyperplanes will classify a given input vector into the subspace belonging to one of the terminal nodes. The orientation of the hyperplanes is determined separately for each decision node and chosen to optimize an a posteriori citerion like the misclassification cost. This computationally expensive strategy is looking for an optimal division of the current training set but may not be adaptable for incremental learning tasks. Additionally, like for the clustering methods, the topological information of neighboring subspaces is lost during the sequential splitting by hyperplanes.

Decision trees [Qui86] and the dynamic version C4.5 [Qui93] are symbolic learning methods that construct a tree structure by considering an information gain criterion. According to this criterion, at each level of the subtrees, an attribute and a partition value are selected, in order to form a decision at this branch. Using the entropy criterion yields trees with shortest average path length (i.e. the number of decision necessary to assign a class) but as in the other methods decribed above, the successive seperation of the input space by orthogonal hyperplanes destroys neighborhood relationships between the leaves of the tree, a fact that becomes important in application with uncertain or fuzzy data.

3.3 Hierarchical Neural Models and Tree Networks

In the past, several hierarchical connectionist models have been introduced that inspired our work, but differ in many aspects from the appraoch proposed in Chapter 3.4. An important point for simulation of the SOM is the search for the best matching unit (*bmu*), i.e. the neuron that matches best the current input vector according to some distance or similarity measure. The distance measure is usually chosen to be the euclidean distance as in equation (1.2). However, other distance measures are also applicable. The Probing Algorithm [LO89] uses the topological neighborhood of the map to look for better units than the current candidate neuron that was determined in a fixed amount of time by e.g. a projection method.

Tree structures obviously accelerate the search. The use of dynamic k-d-trees for efficient *bmu* search in a regular map of fixed size is proposed in [Kel91]. Another way to benefit from tree like structures is a hierarchical structured SOM itself. In [KO90] and [Koi94] the hierarchical network TS-SOM is introduced, that has a SOM on each level of the hierarchy, with increasing number of neurons in each subsequent level. The intermediate SOMs are fixed and the *bmu* search involves the above mentioned Probing Algorithm to avoid the error accumulation from the smaller parent SOMs to the lowest child SOM. However, the effect of the Probing Algorithm is limited by the (in general) non optimal topology preservation of the maps and this problem is not tackled by the TS-SOM.

There exist a number of neural network models that are called to be tree structured and which, considering the network topology of nodes and propagating links, are indeed trees. The construction of these neural trees however is not done in a hierarchical way by a neural training mechanism.

Tree-structured MLP networks like the Perceptron Trees [Utg88] are quite comparable to decision trees (see below) as they use an attribute test at each decision node. But unlike in decision trees, now the terminal nodes are not class assignment nodes, but Linear Treshold Units (LTU), which divide the current subspace by a hyperplane. Thus, the terminal nodes require an additional test and then provide a class assignment. A classification along a path in the Perceptron Tree corresponds to a division of the solution space by hyperplanes perpendicular to the attribute dimensions and only for the last test contained in the terminal node, arbitrary orientation of the decision plane is allowed.

A model based on a competitive tree structured network is proposed in [FJW$^+$91], [LFJ92]. Those binary Neural Trees have a predefined size and structure and may be trained with variants of the competitive learning scheme. Input data is forwarded through the tree and at each level takes the path with the winning neuron. The weight adaptation affects the complete subtree of the first winning neuron. Thus, again we have a subsequential division of the data space by hyperplanes, but for the Neural Trees, the orientation of the planes may change during weight adaptation. The classification structure is more flexible then e.g. but the need to predefine the network structure is a drawback.

3.4 The SplitNet Model: Hierarchical Clustering and Classification

3.4.1 Related work

The Learning Vector Quantization (LVQ) method [Koh90] as well as the k-means clustering method (see e.g. [TG74]) place a number of reference vectors into a set of data records and iteratively minimize a certain quantization error criterion. The drawback of both methods is the a priori setting of the number of reference vectors and the loss of topologic information during the adaptation of positions. The SOM maintains topologic relationships but, as mentioned before, due to the reduction of the data space to a predefined dimension it is unable to keep the topologic defects small (cf. Sec. 3.4.6).

The Growing Cell Structures (GCS) [Fri93b] are a dynamic vector quantization model. Different criteria, e.g. the quantization error, determine the insertion position of a new neuron. Removal strategies yield an adaptive quantizer that is superior to the original SOM, but the GCS model also uses an a priori specified dimensionality (by the choice of simplices) and still involves computations that are performed for all the neurons in the network. The Neural Gas algorithm [MS91] also approximates the distribution of input data and constructs topology preserving connections between its neurons. In [Mar93] another non-hierarchical model for topology approximation is presented. This Competitive Hebbian Learning assumes a given distribution of reference vectors and learns (in the limes) the Delaunay triangulation of these vectors.

The Growing Neural Gas algorithm [Fri95] is a dynamic variant of the Neural Gas, based on the simplices of the original GCS that is free of parameters varying in time. A growing self-organizing feature map constructing hypercubical structures is described in [BV96]. This model is able to add a new dimension to the structure of the network, if topological defects occur during training.

3.4.2 The SplitNet model

The SplitNet model introduces a new aspect compared to the existing models described above. During training, it develops a hierarchical structure in the networks architecture, while still being topology preserving on the *flat*, i.e. lowest, neuron level. Therefore, all considerations of Sec. 2 concerning the retrieval using the networks topological connections still hold in this context. In addition, the network creates a hierarchy of sets of neurons that can be interpreted similar to a decision or classification tree (Sec. 3 will deal with this topic).

Definition 3.1
Let $I = \{i_1, \dots, i_n\}$ a deliberate set and let $\mathcal{P}_*(I)$ the power set of I without the empty set. Then, a system of sets $\mathcal{H} \subset \mathcal{P}_*(I)$ is called a *hierarchy* of I, if for two different sets B, C from $\mathcal{H}$ exactly one of the following three alternatives holds:

$$\text{(i) } B \cap C = \emptyset \quad \text{or} \quad \text{(ii) } B \subset C \quad \text{or} \quad \text{(iii) } C \subset B.$$

The SplitNet model constructs such a hierarchy over its set of neurons by a combination of divisive methods. The splitting of a neuron set occurs for one of the following reasons:

- topological defects,
- clustering aspects (outliers) or
- deletion of neurons.

Each splitting of a neuron set grows the hierarchy at the leaf node representing the splitted set. Supposed the distance measure is given, the only ways of learning are adjusting the neuron positions, creating topological connections as well as changing the number of neurons. Growing neural networks start training with a low number of neurons and grow by repeatedly adding neurons to the network structure. This insertion of neurons can be controlled by, e.g.:

- a vector quantization criterion,
- misclassification rates or
- misclassification costs.

The vector quantization criterion applies for unsupervised learning, where no class information is available. SplitNet uses this criterion to approximate the input distribution by the distribution of neurons. Both the other criteria can be used for supervised learning only. The goal is to analyze the class distribution over the represented data vectors, a task that needs additional considerations developed in part in the statistical classification field (see Sec. 3.4.7 for an outlook on these issues).

As a model of unsupervised learning SplitNet provides an efficient quantization of data distributions of unknown dimensionality. It approximates the distribution by a growing number of one-dimensional Kohonen chains that are linked in a topology preserving manner while keeping topological defects small. This is achieved by repeatedly growing the existing chains, detecting topological defects, resolving them by splitting the chain, augmenting the tree structure and removing superfluous neurons. In the following we describe the principles of these tasks.

3.4.3 The tree structure

To faciliate the discussion, we introduce a formal notation for the resulting tree structure, that represents the constructed hierarchy $\mathcal{H}$. The nodes of the tree are described by a tuple $T_{ij} = (C, S, P, M, s, p, e)$, where i is the level of the tree and j numbers the nodes at the same level. The indices for the elements of the tuple are omitted if they are unambiguous from the context. C stands for the Kohonen chain, that is actually represented by node T. C is empty, if the corresponding chain was splitted. In that case, pointers to the nodes of the resulting chains are included in the set S of sons of T. Thus, $C = \emptyset \Longleftrightarrow S \neq \emptyset$ holds. P represents the father node of T_{ij}. It is needed for propagating values like errors or centroids to the parent nodes of the actual chains. The set M links C to other chains adjacent in the topology of the input space. The values s and p contain the size (represented number of neurons) and the center of centroid of a node, respectively. They are defined recursively by

$$s_{ij} := \begin{cases} |C_{ij}| & \text{if } C_{ij} \neq \emptyset \\ \sum_{\{k|C_{i+1,k} \in S_{ij}\}} s_{i+1,k} & \text{else} \end{cases} \quad \textit{and} \tag{3.1}$$

$$p_{ij} := \begin{cases} 1/s_{ij} \sum_{N_k \in C_{ij}} \mathbf{w}_k & \text{if } C_{ij} \neq \emptyset \\ \dfrac{\sum_{\{k|C_{i+1,k} \in S_{ij}\}} s_{i+1,k} p_{i+1,k}}{\sum_{\{k|C_{i+1,k} \in S_{ij}\}} s_{i+1,k}} & \text{else} \end{cases} \quad . \tag{3.2}$$

Equation (3.1) counts the neurons of C_{ij} if this chain is not empty, else it sums up the sizes of subsequent chains. Similarly, (3.2) determines the centroid of a node representing one or more chains. The error related insertion variable e will be described in Sec. 3.4.5.

3.4.4 Creating the tree structure

One of the main tasks of the model presented here is to improve the topology preservation with respect to the hierarchical arrangement of the substructures. This is achieved by detecting the topological defects in the chains of the network and resolving them by splitting a folded chain into two adjacent ones, each of which covers a part of the subspace of the former unsplitted chain. Other splitting reasons are the detection of outliers in the data by observing variances in local quantization estimates and the deletion of neurons at positions where the data probability is zero. In the following, we present those splitting reasons in detail.

Topological Defects. As described in Sec. 2.2.2, topological defects are locations where close input vectors are mapped to neurons that are not neighbors in the network topology. For the one-dimensional Kohonen chain, we therefore get a topological defect, if the chain bends into the input space and the ends of the chain come close to each other.

Figure 8a) shows a chain C_{11} folded into the two-dimensional input space $[0; 1]^2$. For each neuron N_k the Voronoi region R_k (cf. Equ. (1.3))is given by the dotted lines. It can be seen that e.g., the neurons N_1 and N_7 are adjacent in input space but not in the topology of the chain. Connecting them by an edge would form a cycle of topological neighbors both in input space and

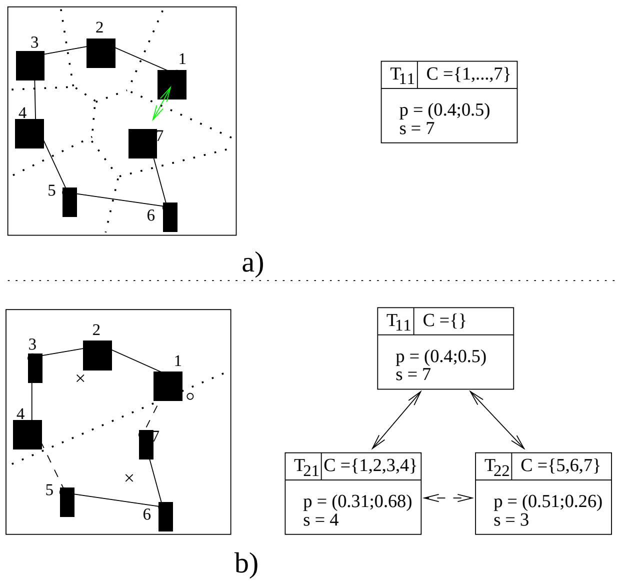


Figure 8: Resolving a topological defect by splitting the chain and adding meta-edges (see text)

in the chain. During training, always after a predefined number of training steps, these cycles are determined by the following suboptimal procedure.

Let C^1 and C^2 denote the set of neurons in the first and second half of a chain, respectively. For pairs (c_i^1, c_j^2) of neurons with $c_i^1 \in C^1$ and $c_i^2 \in C^2$ we calculate the midpoint $c^{mid} = \frac{1}{2}(c_i^1 + c_j^2)$ and present it as input vector to the network (in test mode, without weight adaptations, of course). If the first and second winner for this point c^{mid} are just the points c_i^1 and c_i^2, we have detected a topological defect. This procedure is suboptimal, because even if one or two other neurons win for this input, c_i^1 and c_i^2 may nevertheless have adjacent Voronoi regions and thus be neighbors, forming a topological defect. Though, this heuristic works, because topological defects are mostly occurring for several adjacent neurons and if one is overlooked by the procedure, the adjacent one is likely to be found. The main advantage of this heuristic is avoiding costly calculation of the correct Voronoi regions.

If topological defects are found, they are resolved by splitting the cycle in the middle between the neurons of the defect, thus creating two new chains at this particular defect. Those new chains C_{21} and C_{22} continue learning with their own increased neighborhood radii and are added to the network tree in the set S of sons of T_{11}, while C_{11} is now empty. Both, the edge that is removed by the splitting and the edge that would have formed the cycle are added as *meta-edges* (dashed lines) to the sets M_{21} and M_{22} of the topologically adjacent chains.

The resulting chains and their centroids p (crosses) are depicted in Fig. 8b). The new tree structure is shown on the right. The old chain C_{11} is empty, but T_{11} still represents the neurons from the chains below.

The SplitNet model contains another type of meta-edges that is based on centroids of intermediate levels of a larger tree and improves the topology preservation even more. Again after some predefined number of training steps, for a certain level of the tree, the centroids are examined with respect to their neighborhood relations (with a procedure similar to the one detecting defects). Neighboring centroids are connected by meta-edges and those meta-edges sink down in the tree until they reach the neuron level, providing additional topologic information that might be available only in higher levels of the tree.

The centroids are used to traverse the tree in order to find the best matching unit (*bmu*). As in other hierarchical networks, e.g. the TS-SOM [Koi94], the search might be misleaded by the upper levels. Here, the critical point is the decision according to the centroids. This may lead to suboptimal results (consider the input marked by a circle in Fig. 8b)), and a procedure like the Probing Algorithm [LO89] can be used to find the real *bmu*. Now, as the topology preservation is nearly optimized by the structuring of the network, the Probing Algorithm does not fail because of local minima as it often is the case in the original SOM. With a strictly local search (just following the topologic connectivity), all relevant nodes are found in the close neighborhood of the first guess. As only the chain of the *bmu* is updated, the computations in the network during training and structure learning only affect a very localized group of neurons and tree nodes per time step, the largest part of the net remains unchanged.

Splitting for outliers. An observation often made through our experiments with various input distributions was the fact that outliers in the data are represented by one or several members of a chain, whereas the rest of this chain approximated a (part of a) different cluster. In order to capture this conceptually, we let the network apply a graph-theoratical criterion formerly introduced for cluster detection with minimal spanning trees [Zah71]. Inside a chain, the edge lengths between the neurons are compared with each other. If an edge length is significantly larger than the average edge length in that chain, this edge can be interpreted as a bridge, connecting two independent regions in the input space. Thus, splitting is done at that edge, replacing it by a meta-edge and thus increasing the corresponding subtree.

Deletion of neurons. Below, we describe reasons for deleting a neuron in a chain. If this neuron is at the end of the chain, it simply is discarded. If on the other hand this neuron is inside a chain, deletion of this neuron also invokes a split at that location, connecting the former neighbors of the deleted neuron by a meta-edge and thus growing the tree at this level.

3.4.5 Shrinking and growing of chains

Because of the above splitting process it may happen that there exist neurons in places that belong to the subspace of another chain. Those neurons are no longer useful for the function of the net. If the lateral connections are not strong enough to pull them back into their homeground, they should be removed. In order to discover those neurons, each neuron N_j has a counter b_j that registers how long this unit has not been *bmu* although the chain of N_j also included the *bmu*:

$$b_j := \begin{cases} 1 & \text{if } N_j = bmu \\ b_j - \Delta b_C & \text{if } N_j \neq bmu \text{ and } N_j \in C(bmu) \\ b_j & \text{else} \end{cases} \tag{3.3}$$

After splitting, the counters of all neurons in the affected chains are reset to $b_j = 1$. The constant Δb_C can be chosen differently for each chain and is set to the reciprocal of a small multiple of the chain size. Neurons with $b_j = 0$ are removed from the chain. This removal criterion is similar to the approach presented in [Fri93b], but it is modified to be strictly locally computable. Only the neurons in the winning chain are updating their counters, thus no global calculations are involved.

After repeatedly splitting chains, the neuron distribution in the input space may not reflect the training data properly and may not be optimal for quantization or classification tasks. A mechanism for adding neurons to the network according to some suitable insertion criteria is needed. Incremental tasks require a criterion for spontaneous insertion in order to cope with new relevant data subspaces. In this report, the discussion is limited to statistical insertion of neurons according to a quantization error measure.

In order to achieve an optimal reduction of the network quantization error $E_{VQ} := \sum_i ||\mathbf{x}_i - \mathbf{w}_{bmu}||^2 p(\mathbf{x}_i)$, a new neuron should be inserted at a location, where the actual quantization error is high. The contribution of a chain to E_{VQ} is estimated with the help of the errors of its neurons. At each step of the training process, the *bmu* adapts its error variable E_{bmu} by an equation that is similar to the Kohonen learning rule:

$$E_{bmu}^{t+1} := E_{bmu}^t + \alpha\,(d_{bmu}^2 - E_{bmu}^t), \tag{3.4}$$

where $d_{bmu} = ||\mathbf{x}_i - \mathbf{w}_{bmu}||$ and $\mathbf{x}_i$ is the current input vector. The learning rate can be set to a small constant or can be reduced by a cooling scheme involving the error variance in the chain. When the movement of the neurons stops after some training time, the expected value $\langle d_j^2 \rangle$ equals the quantization error $E_{VQ,j}$ of neuron N_j

$$\langle d_j^2 \rangle = \sum_{d_j^2} d_j^2\, p(d_j^2) = \sum_{d_j^2} (d_j^2 \sum_{\{x_i | d_j^2 = ||w_j - x_i||^2\}} p(x_i)) = \sum_{\{x_i | d_j^2 < d_k^2, \forall k \neq j\}} d_j^2\, p(x_i) = E_{VQ,j}, \tag{3.5}$$

thus also $\langle E_j \rangle = E_{VQ,j}$ holds. Now we estimate the relative error of a chain C by

$$E_C^{rel} := \frac{h_C \sum_{j \in C} E_j}{s_C}, \tag{3.6}$$

where h_C stands for the number of times the chain C contained the *bmu*. When the neuron N_{new} is inserted into the chain, its position is set to the point $\mathbf{w}_{new} = \frac{1}{2}(\mathbf{w}_i + \mathbf{w}_j)$ halfway between its new neighbors N_i and N_j. The error variable E_{new} is initialized with $(\frac{1}{3}E_i + \frac{1}{3}E_j)$ and E_i and E_j are reduced by one third each, approximating that the new neuron had been there before. Thus, the estimated overall error of the chain still is the same right after insertion, but the chain has grown by the newly inserted neuron, reducing the relative error of the chain.

To determine the appropriate chain for insertion, we observe the reduction $\Delta E_{VQ} := E_{VQ}^{old} - E_{VQ}^{new}$ of error caused by the insertion. We have

$$\Delta E_{VQ} = \frac{h_C \sum_{j \in C_{old}} E_j}{s_C} - \frac{h_C \sum_{j \in C_{new}} E_j}{s_C + 1} \overset{*}{=} h_C \sum_{j \in C_{old}} E_j (\frac{1}{s} - \frac{1}{1+s}) \tag{3.7}$$

where the step marked by $\overset{*}{=}$ is possible, because at insertion time $\sum_{j \in C_{old}} E_j = \sum_{j \in C_{new}} E_j$ holds. Then, we get

$$\Delta E_{VQ} = h_C \sum_{j \in C_{old}} E_j \frac{1}{s(s+1)} = \frac{E_C^{rel}}{s+1} \tag{3.8}$$

Thus, the reduction of error is maximal, if the new neuron is inserted into the chain C, for which

$$e_{insert} := \frac{E_C^{rel}}{s_C + 1} \tag{3.9}$$

is maximal. Thus, again only local computations are necessary to evaluate the insertion criterion and a fast tree search determines the chain with the highest e_{insert}. As in the case of splitting, an increase of the neighborhood radius allows a rearrangement of neurons after insertion.

3.4.6 Empirical Results

The topology preservation as well as quantization error and retrieval speed were compared to the original SOM, where both networks contained $N = 40$ neurons (SplitNet started with 10 neurons and grew up to 40). The input vectors where chosen at random from the unit square $[0; 1]^2$.

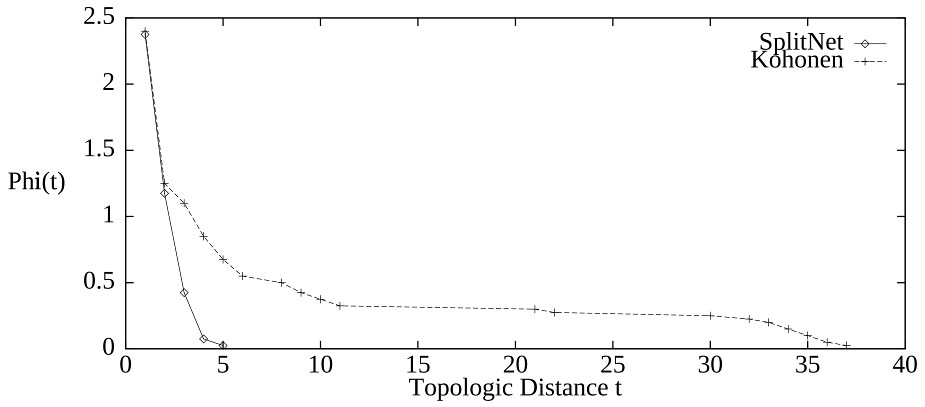


Figure 9: Topology preservation of SplitNet and a regular Kohonen chain

Figure 9 shows a comparison of the values of the topographic function Φ in (2.3) for the Kohonen chain and the SplitNet model. Note the large sizes of topologic defects for the regular Kohonen chain. The important point for SplitNet is the disappearance of $\Phi(t)$ for small t ($t^+ = 5$ in this example). That means that a moderate amount of search, e.g. by a kind of Probing Algorithm, will suffice to guarantee that all topologic relevant nodes in the network structure are encountered. For the Kohonen chain, $t^+ = 37$ holds, thus large portions of the structure are unreachable by short range search in the network topology although those neurons are located quite near in problem space.

These results show the utility of the SplitNet model for tasks that need to retrieve a number of similar neighbors to a given query. With low values of t^+, the retrieval algorithms of Sec. 2 become applicable and thus SplitNet is a model that complies with the requirements of e.g. diagnosis applications in the medical domain.

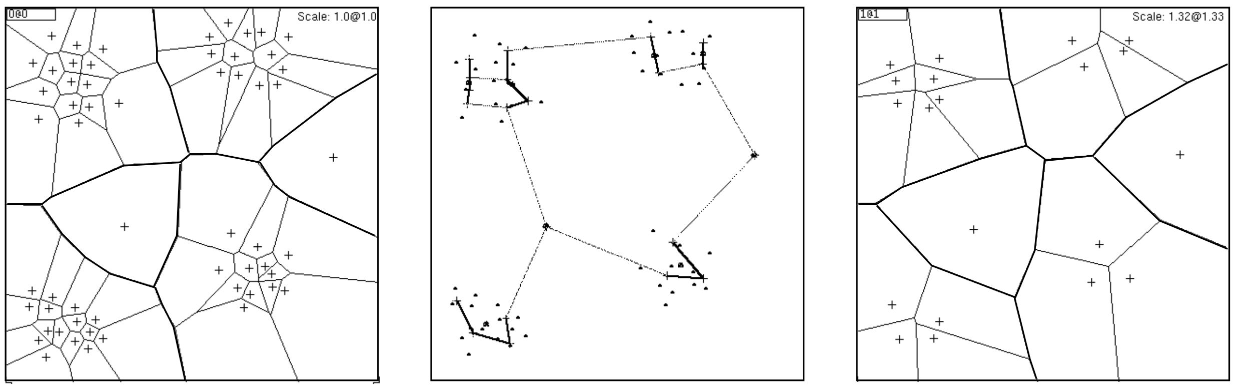


Figure 10: A sample data set (left), the interconnected SplitNet chains (middle) and Voronoi diagram with decision regions (right)

The quantization results of both networks are quite comparable with a slight advantage for the SplitNet model in the simulations done. However, the simulation and retrieval times of the SplitNet model were at least one order of magnitude faster than those of the regular Kohonen chain and this advantage can be expected to increase with more complex applications.

Figure 10 depicts the behaviour of SplitNet for a sample data set (left). SplitNet develops a structure of interconnected chains (middle) and quickly approximates the decision regions of the data records as they appear in the Voronoi diagram (left and right, bold lines).

The real value of a model can only be estimated when it is applied to real world data or even better when it is evaluated in a real world application environment. We currently establish the application of SplitNet for medical diagnosis and monitoring tasks, some details of which are described in Sec. 5.

3.4.7 Future extensions

Right now, the hierarchical structure of the network is constructued by a purely divisive algorithm. We defined several splitting criteria and divide a neuron set, i.e. a chain, into subsets, the sub chains. Especially when we consider the fact that the network is growing from a very small initial configuration we can conclude that early splits might become obsolete in later stages of the training. Despite a correct representation of the training data by neuron distribution and topological connections, the inherent grouping by the chains of the network might be improveable by some agglomerative methods. This would possibly reduce the number of chains in favour of a better comprehensibility and interpretability of the hierarchy in applications like the one described in Sec. 5.

In order to incorporate these ideas, it might be necessary to abandon the chain based architecture and develop a more general variant of the model. For the growth process of the network we then try to apply insertion criteria based on misclassification rates and cost functions. By this, we can control the training of the network by more information than contained in the distribution of training vectors. Additional knowledge of misclassification costs will improve the performance of the network in cost sensitive applications.

4 Interpretation of Hierarchy

In this section, we outline the semantics of the hierarchy in the models described in the previous section. The different meanings of tree traversal in those models as well as the utility of topology interpretation (cf. Sec. 2) demand efforts on the integration of different approaches on a common basis. This is one objective of our current research.

4.1 Hierarchical Cluster Analysis

The dendrogram constructed by Hierarchical Cluster Analysis is a hierarchy on the set of data vectors (cf. Definition 3.1) that can be drawn as a binary tree. Figure 11 shows an example for arbitrarily chosen data vectors $a, \ldots, f$. On top level, the root of the tree represents the whole data set whereas at each lower level a branching represents a split of a set into subsets or linkage of to sets into a larger set (depending on the nature of the construction method). As the tree levels are related to the distance at which the split or linkage occurred, a tree traversal can provide information on the distance of data subsets occurring on the selected path.

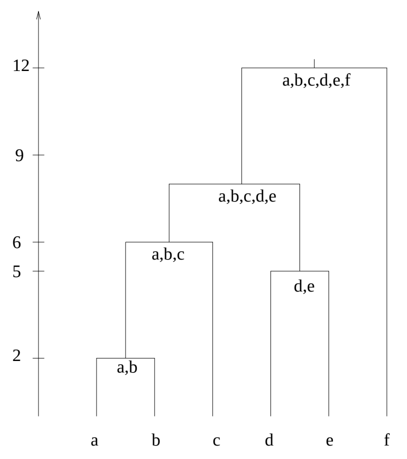


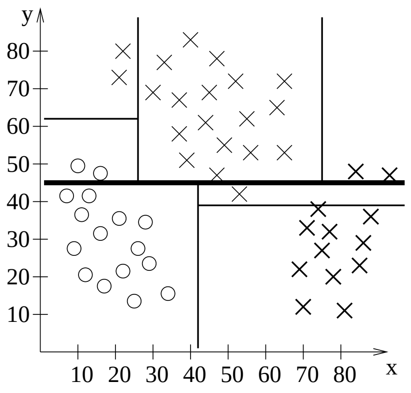
Figure 11: An example of a dendrogram as it is produced by Hierarchical Cluster Analysis

If we assume an agglomerative method for the following discussion, we can read from the dendrogram the order of linkage. The most similar vectors are a and b and they are combined to $\{a, b\}$ at distance 2. Next, d and e are linked to $\{d, e\}$ at distance 5, then $\{a, b\}$ and c are linked to $\{a, b, c\}$ and so on. What we do not know from the dendrogram is, e.g., the relative position of single data vectors or groups of vectors to each other. Vector f is linked last to the other vectors and thus is quite far off the group $\{a, b, c, d, e\}$, but in which direction? There is no information on that contained in the dendrogram. Additionally, the dendrogram itself provides no decision information for the processing of new vectors to be classified. It does not produce or represent decisions regions of any kind.

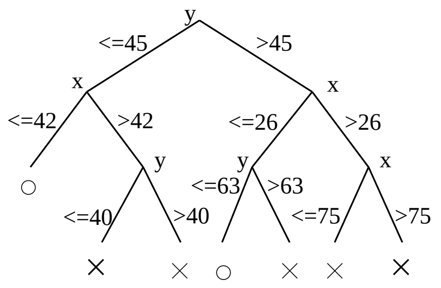
Thus, the Hierarchical Cluster Analysis is a useful means of displaying the structure of a data set as a preliminary data analysis, but it seems to be inadequate for tasks that require processing and decision capabilities.

4.2 Classification and decision trees

As described in Sec. 3.2, both the classification and the decision trees construct a tree that represents a sequence of hyperplanes. Those hyperplanes successively subdivide the data space into regions and subregions according to the respective criterion. Figure 12 gives an idea of the kind of division of the data space by a decision tree method that selects attributes and then determines a suitable attribute value for the positioning of the hyperplane.



a)



b)

Figure 12: An example data set for three classes, a possible division of the data space by hyperplanes and its representation in a decision tree

First, the whole data set is divided into two subspaces, those subspaces are divided again and so on. The resulting structure is a binary tree with decisions at each branch. The leaves contain only data vectors of one single class. Thus, for a new data vector the tree traversal according to the node decisions yields a classification result. This result is gained with the lowest misclassification rate for the classification tree and with the least average number of decisions for the decision tree. Nevertheless, the path has to be followed down to the leaf, because intermediate stages cannot assign a class membership to the query. And once a leaf is reached there is, like in the Hierarchical Cluster Analysis, no information on similar data vectors in neighboring subspaces. One can see from Fig. 12b) that each class has representants in the left subtree as well as in the right subtree, albeit each class consists of only one cluster. The similar vectors of the same class are far away in the tree, only reachable by extensive backtracking.

A classification of a query is completely based on the strict path decisions inside the tree nodes. Uncertain or vague information cannot be processed in the tree, due to the lack of neighborhood information. Thus, the strengths of the classification and decision trees are the deterministic construction of tree structures that minimize a given criterion (see above), but the methods are not suitable for the retrieval of similar vectors or the processing of fuzzy information.

4.3 Hierarchical Neural Models and Tree Networks

For the Perceptron Trees described in Sec. 3.3 many of the above statements regarding the decision trees hold. The only difference is that the leaf of the Perceptron Tree is also a decision node and provides a decision of class membership only after the test. The orientation of the dividing hyperplanes is flexible and trainable, but again the sequence of tests along a path in the tree consists of crisp decisions and neighborhood relationships of the original data are lost. Like for the classification and decision trees the sequence of space divisions is determined during construction according to the class information of the training set.

The Neural Tree model mentioned in Sec. 3.3 is an unsupervised method and produces a more data driven organization of the tree structure. The positioning of the given neurons is done by the competitive learning rule, thus each neuron represents a cluster or a part of a cluster that is covered by several neurons. The lack of class information during training leads to a more cluster oriented division of the data space and thus the model is rather a quantization than a classification tool. Like in the methods analyzed above, there are no interconnections between neurons at the same level of the tree, so again the missing of information on similar vectors limits the applicability of the Neural Trees for real world tasks that involve retrieval and inspection of similar cases.

4.4 The SplitNet Model

In the above methods, the hierarchy plays the essential role in functionality and interpretability. The sequence of decisions according to the constructed set of hyperplanes yields the classification result for a given query and the construction method determines the efficiency of the application. But all the models have the drawback that they cannot compare their result to alternative possibilities or simply retrieve a (possibly incrementally growing) set of similar cases for a query. In the SplitNet model, we integrate the retrieval approach described in Sec. 2 that is crucial for efficient k-nearest-neighbor class assignment and evaluation of alternative solutions with the tree structured decision methods, that effectively divide the relevant data space. The advantage of this integration is obvious: the k-nearest-neighbor approach is suitable for unsupervised training methods and the tree structure is a very efficient method for locating best matches to a given query.

The hierarchy in the SplitNet model serves, similar to the structure of the Neural Tree, as an efficient access structure for fast determination of the neuron the weight vector of which is most similar to a given query. Again, the path decision during traversal of the hierarchy is made with respect to hyperplanes. These hyperplanes are not represented explicitly like in the decision nodes of the classification and decision trees, but in each node implicitly defined by the positions of all (possibly more than 2) successor nodes. Thus, they can have arbitrary orientation and are controlled by the distribution of the data vectors. As demonstrated in Fig. 10, page 25, the location of neurons and consequently the orientation of hyperplanes preserve the natural decision regions as they are observed with the specified distance measure.

The hierarchy of SplitNet is only one part of network structure that is constructed during training, the topology of the network being the other, but the hierarchy itself can be of great importance for the visual inspection of the principal structure of a data set. In the next section, we show how the hierarchical structure, printed as a two-dimensional tree, is used for inspection of the training result and interpretation of neuron contents with respect to a medical diagnosis and monitoring problem.

5 Application: Medical Diagnosis

5.1 Goals of the Research

Together with researchers from the Bad Neustadt Hand Center we are working on new methods for the diagnosis of Ulnar nerve lesions. While assessing ulnar sensory function is easy, objective and quantitative analysis of motor funtion is quite difficult. Especially improvement of fine motor activities are difficult to quantify. Motor dysfunction of the ulnar nerve is described based on personal experience and is paraphrased as 'clawing' or 'rolling' (the static and dynamic dysfunction, respectively). Until now there is no convenient measurement system to distinguish finger movement patterns. We tried to establish a system for measurement and classification of finger movement patterns in ulnar nerve palsy. From the medical point of view, there are a number of goals a new and successful diagnosis method should aim at:

- Diagnosis of less severe cases (as for example entrapment or compression neuropathies)
- Analysis of different stages of static and dynamic dysfunctions
- Detection of combination of static and dynamic dysfunction
- Potential discovery of new functional loss pattern
- Judgement of success after surgical reconstruction either of a nerve or of motoric function

5.2 Methods for data generation

Finger movement is controlled by several muscles. The intrinsic muscles innervated by the ulnar nerve basically coordinate the movement of the three finger joints. There are many ways of moving the fingers. We used finger movement from full extension to maximal fist closure back to full extension (movement cycle) as a model, because all three joints are involved to achieve maximum range of motion.

The examination of possible disturbances of the motoricity must be easily and quickly executable in vivo, so the angles in the three joints of the finger (see figure 13) are measured by an ultrasound device. The patients are asked to open and close their fist several times and the result of measurement is the course of the angles depending on time (figure 14). No particular properties can be read from these original curves, so they have to be processed further. A first approach reconstructs the movement of the finger and, at special positions of the fingertip, writes out current angle data. These data sets characterize the motion of the finger and are used for the training of the network. Right now, this pattern set is generated by an expert who analyzes the steps of the finger movement by hand. Current efforts are concerned with automatic and more powerful methods of data generation and feature extraction, but the currently available data is sufficient to demonstrate the principle of the expected analysis.

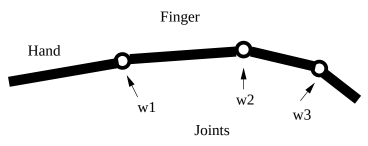


Figure 13: Finger joints and their angles during flexure

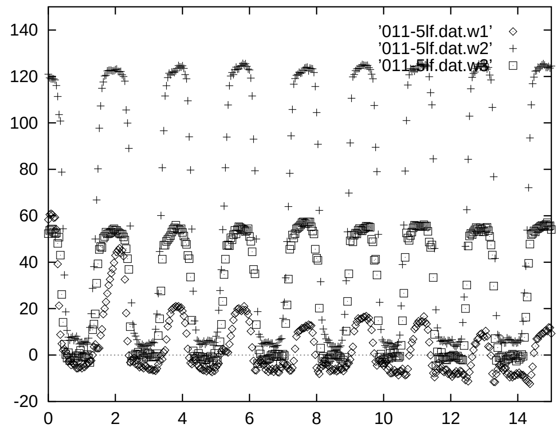


Figure 14: The course of joint angles during fist opening and closing

5.3 First empirical results

We now present the results of applying the SplitNet model to the data generated by the method mentioned in the last paragraph. The data sets are classified but the classes are not used in the training process. They only serve for visualization of the results. The patients with an Ulnar nerve lesion are coded with three letters while healthy subjects of a control group are represented by three digits. Figure 15 shows a resulting tree structure for a small data set. The subject codes are mapped to the tree node that contains the winner neuron for a pattern of this subject.

The tree structure serves for a fast classification of a new generated pattern and assigns it to a leaf box that contains similar pattern. In addition to this decision tree like behaviour (which is of course not entropy based in this example), the structure can be examined further, considering the meta-edges in the topology of the network (dotted lines in the figure). Together with distance information of these edges, it is possible to interpret the structure with respect to size and proximity of clusters as well as the level in the tree on wich a separation occurred.

In our little example here, we note that subject **010** was separated first from the rest. It was supposed to be a healthy person but this result indicates the need for a further medical examination. In the second level, we have four sons of decision box 2, two of which are leaf nodes that contain subjects with nerve lesions. On the next level, the other two boxes split the remaining set into several clusters, that are linked amongst themselves. They contain the pattern of the healthy subjects and two of the nerve lesion group. This shows an advantage of network structure: the distance information of the meta-edges can be used to see that the pattern for subject **szi** are somewhere between the lesion and the control group and subject **akk** falls back into the control group. These facts can be supported by the medical explanation for these cases. Subjects **szi** and **akk** are patients where the nerve is currently recovering quite well from a cut and motoricity was judged by medical experts to be normal. The boxes with subjects **cav**, **rap** and **zit** separate two cases of static dysfunction from one case of dynamic dysfunction. Thus we succeeded in our first steps towards the goals given by medical research (see chapter 5.1).

An additional advantage that arises from the direct coding in networks like SplitNet is the

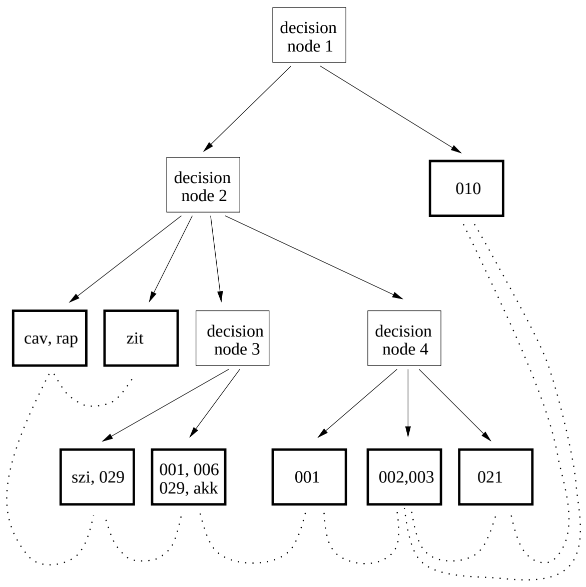


Figure 15: A tree structure for a restricted data set (see text for explanation)

fact that the weights of the neurons are retransferable into the form and semantics of the original data. In our example this means that from a weight vector, we can reconstruct the steps of finger movement that would have generated this weight vector as training pattern. Thus, each cluster or even decision node in the tree might associated with the characteristic (generalized) movement of the represented neurons that is easily interpretable by the medical experts. This might help in identifying and describing new forms of motoric dysfunctions - another one of the medical goals. Figure 16 shows an example of the retransformation of information from the neuron level to the visualization of finger movements. This tree was generated for a new data set containing 9 steps per movement cycle and with more patient records available [HRKB96]. The text-only version of this tree is shown in Fig. 17. The nodes contain besides some network specific data that is used for identification purposes only the mapped subject codes like in Fig. 15. Those two versions of a tree can be used for interpreting the training result by the medical experts.

From the beginning the tree trunk #1 splits into two branches depending on layer #2 and #3. Branch #3 contains mainly the physiological movement while #2 contains mainly pathological ones. Further processing of branch #2 results in 6 leaves, four of them are of special interest. Leaf #6 contains movements where in full extension $w1$-joints are hyperextended whereas $w2$- and $w3$-joints never reach full extension (clawing). In contrast to this, leaf #14 represents mainly dynamic dysfunction. Here, $w2$- and $w3$-joints flex before $w1$-joint flexion thus reassembling the rolling of a carpet. Leaves #11 and #15 present different forms of combined clawing and rolling.

The leaves depending on branch #3 mainly contain physiological movement data. Among these data are only a few data of patients with ulnar nerve lesion, e.g. data of (**iyi**) a young girl with partial lesion of the ulnar nerve and minor changes in movement. Leaf #4 and #18 are of special interest. They represent patients after Zancolli-lasso-plasty, a surgical reconstruction of motoric function. Especially the movements of leaf #18 show exclusively $w1$-flexion initiating the flexion curve and late $w2$ and $w3$ during fist closure. This movement simulates an intrinsic-plus at the beginning.

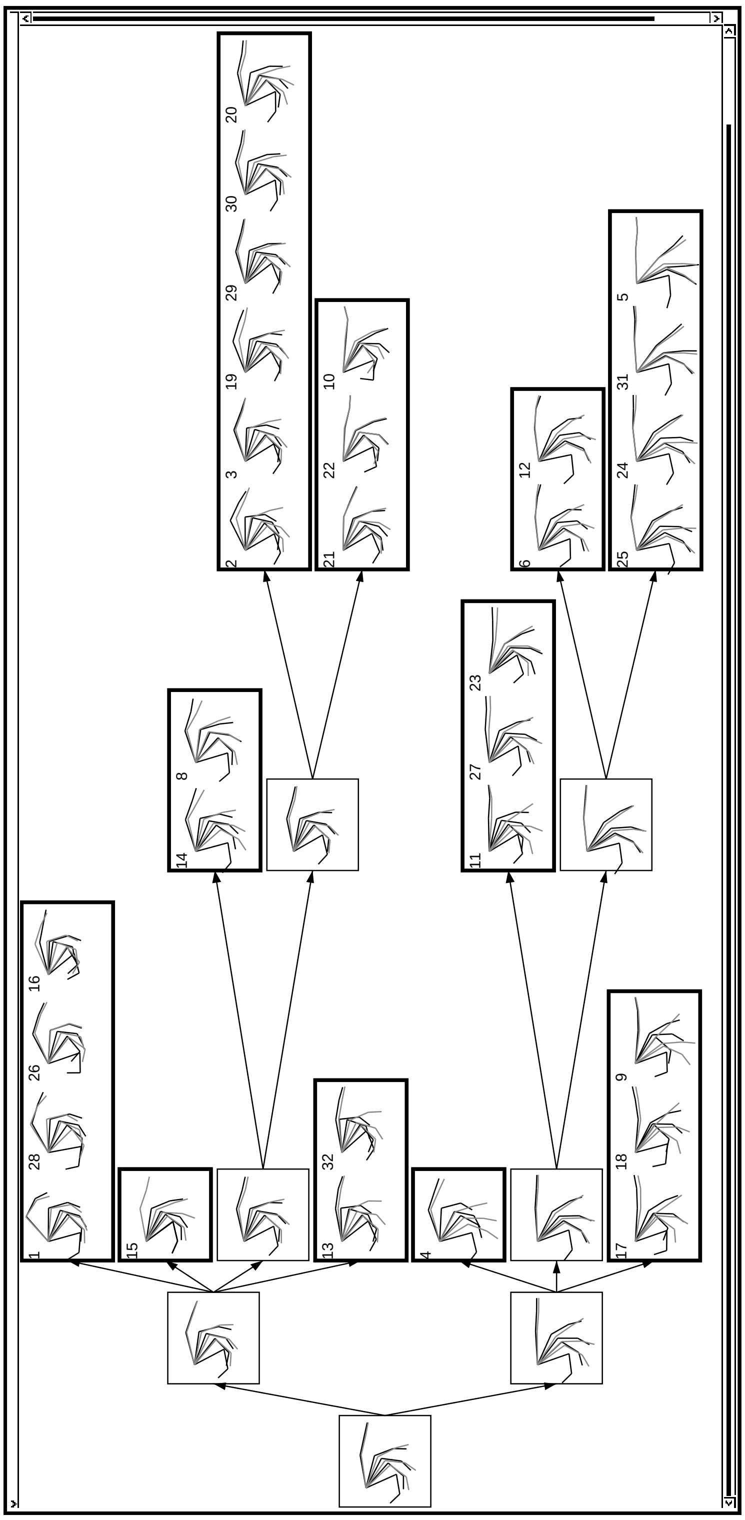


Figure 16: Finger movement pattern as a visualization for neuron content. Each pattern shows nine steps of one movement cycle, five steps for the closing of the fist (black) and four steps for the opening (gray)

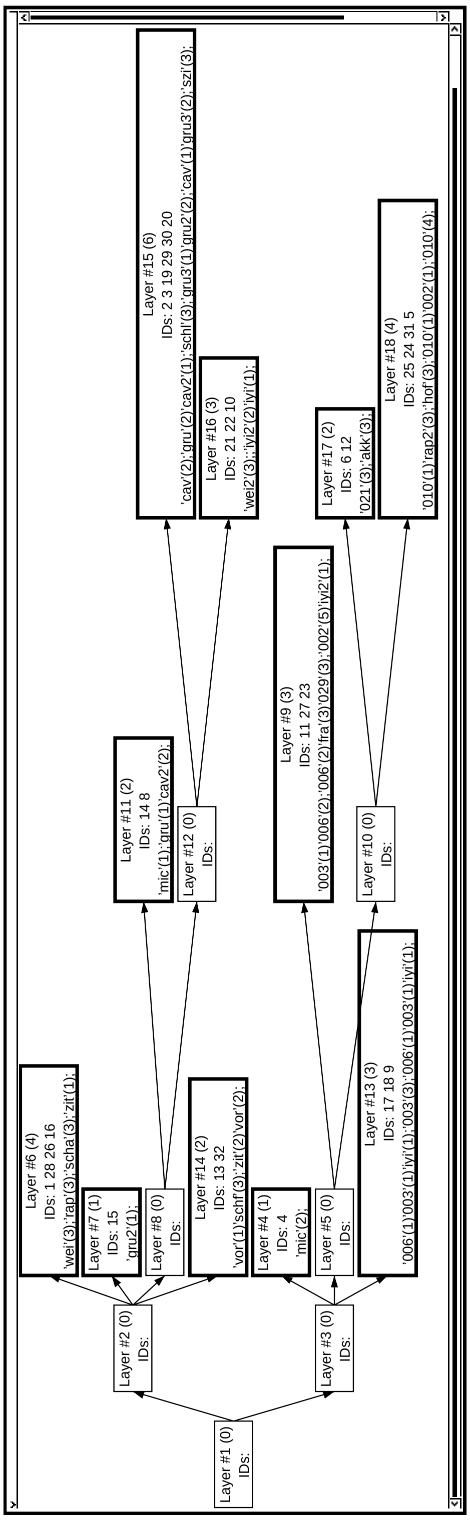


Figure 17: Text version of the previous tree. The leaf nodes contain identification numbers for the chains and neurons of the network and the subject codes of the patients mapped onto the representing neurons

5.4 Discussion

Motor changes caused by ulnar nerve lesion are difficult to analyze. Joint angle measurements and grip strength evaluation can only describe a static state of the underlying pathological process. Real time measurement of motion is an well established procedure [AC91]. Usually they are used in biomechanic laboratory rather than in clinical kinematic to obtain kinematic data as rotation angles or velocity. Such data can be analysed by common statistical methods. In contrast there are no investigations of movement pattern.

Our system is the first to apply pattern recognition by a neural net approach. In this sense we are trying to build an expert system which can distinguish between normal and pathological movement. Looking at our results we can conclude that the SplitNet model is principially able to fulfill this task. In our study normal and pathological movements are already split at the level of Branch #2 and #3. But beside these 'simple' expert task SplitNet is able to do more. Most Neural Networks are able to learn data by training, but usually the distribution of knowledge in the network is not accessible. Training SplitNet with input data which are in fact an image of the real movement, we can extract an image from each neuron after training. This allows the interpretation of the learning processes which have happened. Thus interpretation of the images enhances our knowledge about the movement pattern. For example we recognize that most of the Zancolli-lasso-plasty are seperated, because they present a special type of movement pattern, which resembles intrinsic-plus movement. At the moment we have data of only a small number of patients. So we do not know, if we portray the whole spectrum of ulnar nerve dysfunction. More data have to be recorded to decide how many different types of changes in movement pattern there are. Data from non-traumatic lesions as compressions neuropathies have to be gathered. The aim is to build up a neural net containing all types of normal and pathological movement. Then we are able to classify all ulnar nerve lesions by recording finger movement and arranging the movement pattern by the neural net at a typical location in the tree as well as having them connected by the topology of the network and thus being able to quickly access all relevant movement pattern of requested similarity.

A Proofs of the Theorems

Theorem 2.1 The retrieval algorithm $\textsc{Retrieve}_{cp}$ is correct and complete.

Proof: As $\textsc{Retrieve}_{cp}$ is an incremental retrieval algorithm, it is sufficient to prove the completeness of each list L_{s+1} that is constructed in step 5 of the algorithm. If L_{s+1} is complete at any step s, then $\textsc{Retrieve}_{cp}$ is also correct.

We prove the completeness of the algorithm by induction on s. Let L_s denote the list of s nearest neighbors to the query q.

Hypothesis:
The node v_{N_1} is nearest neighbor to query $q \Rightarrow L_1 = (v_{N_1})$ is complete.

Induction step:
Let L_s be complete. For the determination of the next nearest neighbor the candidate set

$K_{N_{s+1}} = (\bigcup_{\{k|l_k \in L_s\}} V_k) \setminus L_t$ is built (cf. step 3 of the algorithm). Then, $\exists v_r \in K_{N_{s+1}}$ with $d(v_r, q) \leq d(v_j, q)$ for all $v_j \in K_{N_{s+1}}$ and $L_{s+1} := (L_s, v_r)$ is complete. Thus, the candidate for the $(s+1)$-st position really is in the candidate set $K_{N_{s+1}}$.

Proof:
We prove the step by contradiction. Assume $\exists v_r$ for which $L_{s+1} := (L_s, v_r)$ is complete, but $v_r \notin K_{N_{s+1}}$ holds.
Then we have $d(v_r, q) \leq d(v_j, q)$ for all $v_j \in K_{N_{s+1}}$. $(*)$

Let R_{v_r} be the Voronoi-Region belonging to node v_r and $R_L = \bigcup_{v_i \in L_s} R_{v_i}$ be the union of all regions for nodes in the list. From the assumption, we have $R_{v_r} \cap R_{L_s} = \emptyset$. Define $\mathbf{d} = (\mathbf{v}_r - \mathbf{q})$ to be the vector starting at q in the direction of v_r. Then there exists a $\lambda_0 > 0$, such that for $p_0 = \mathbf{q} + \lambda_0 \mathbf{d}$ holds: $p_0 \notin R_{L_t} \wedge p_0 \notin R_{v_r}$, i.e. p_0 is neither in the voronoi region of v_r, nor in one of the regions of nodes belonging to the list. Thus, p_0 is in the Voronoi region of a newly encountered node v_s, $p_0 \in R_{v_s}$, and $d(v_r, p_0) > d(v_s, p_0)$ holds. With lemma A.1 (see below) we can conclude $d(v_r, q) > d(v_s, q)$.

Case 1: $v_s \in K_{N_{s+1}} \Rightarrow$ Contradiction to $(*)$.
Case 2: $v_s \notin K_{N_{s+1}} \Rightarrow L_{s+1}$ is not complete, contradiction to the assumption.

Thus, we have $L_{s+1} := (L_s, v_r)$ is complete, with $v_r \in K_{N_{s+1}}$. □

Lemma A.1
Let points p, q, r and s be given, for which the following holds: $p \in \overline{qr}$ and $d(s,p) < d(r,p)$. Then we conclude $d(s,q) < d(r,q)$.

Proof: Figure 18 illustrates the course of the proof.

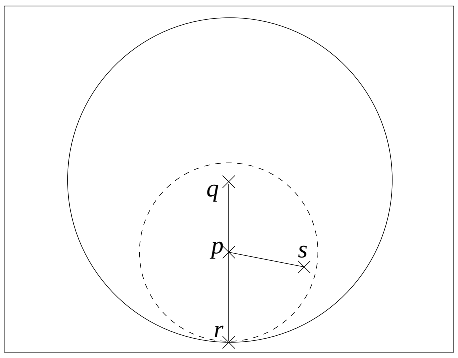


Figure 18: The points p, r, s and their relation to query q

The bigger circle has the center q and radius $d(q,r)$, the smaller one is drawn around p with radius $d(p,r)$.

The assumption is $d(s,p) < d(r,p)$. $(*)$

Thus, s is properly inside the dashed circle.

Now, suppose that $d(s,q) \geq d(r,q) \Rightarrow s$ lies on or outside the large circle $\Rightarrow$ Contradiction to $(*)$.

Therefore, we have $d(s,q) < d(r,q)$. $\square$

Theorem 2.2 RETRIEVE$_{ci}$ is correct and complete.

Proof: It remains to show that step 3 of the algorithm RETRIEVE$_{ci}$ includes all the nodes into the candidate set that are neighbors to elements of the current list L_s according to the Delaunay graph. As the definition of the maximal size t^+ of topological defects just uses the Delaunay neighbors of nodes, a breadth-first search with depth t^+ ensures the completeness of $K_{N_{s+1}}$. Now the correctness and completeness of RETRIEVE$_{ci}$ is shown as in the proof for Theorem 2.1.

$\square$

Theorem 2.3 RETRIEVE$_{ci,opt}$ is correct and complete.

Proof: As described in the explanations for Fig. 6, the optimized algorithm suspends the computation of a distance if and only if the triangle inequality ensures that the unknown distance (c in the example of Sec. 2.2.3) cannot be smaller than the distance of another already known distance (a). Thus, the heuristic in the algorithm RETRIEVE$_{ci,opt}$ only affects the efficiency but not the correctness and completeness. $\square$

B Acknowledgement

This research is supported by the Graduate College Technomathematics at the University of Kaiserslautern. The authors wish to thank Christian Blum for implementing the medical application system and especially Dr. Peter Hahn for inviting us to collaborate in a field of current medical research.

References

[AC91] K. N. An and E. Y. S. Chao. Kinematic analysis. In K. N. An, R. A. Berger, and W. P. Cooney III, editors, *Biomechanics of the Wrist Joint*, pages 23–36. Springer-Verlag, New York, 1991.

[Ama80] S.-I. Amari. Topographic organization of nerve fields. *Bulletin of Mathematical Biology*, 42:339–364, 1980.

[BDH96] H.-U. Bauer, R. Der, and M. Herrmann. Controlling the magnification factor of self–organizing feature maps. *Neural Computation*, 8(4):to appear, 1996.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees*. Belmont, CA, Wadsworth, 1984.

[Bra90] R. Brause. Optimal information distribution and performance in neighbourhood-conserving maps for robot control. In *Proc.2nd Int. IEEE Conference on Tools for Artificial Intelligence*, pages 451–456, Los Alamitos, CA, 1990. IEEE Comput. Soc. Press.

[BV96] H.U. Bauer and Th. Villmann. Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 1996. to appear.

[FJW$^+$91] L. Fang, A. Jennings, W. X. Wen, K. Li, and T. Li. Unsupervised learning for neural trees. In *Proceedings of the IJCNN*, 1991.

[Fri93a] B. Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. Technical Report TR-93-026, Int. Computer Science Institute, Berkeley, CA, 1993.

[Fri93b] B. Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learning. Technical Report TR-93-026, ICSI, 1993.

[Fri95] B. Fritzke. A growing neural gas network learns topologies. In Tesauro G., Touretzky D.S., and Leen T.K., editors, *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

[GS93] M.H. Gross and F. Seibert. Visualization of multidimensional image datasets using a neural network. *Visual Computer*, 10:145–159, 1993.

[Häm95] A. Hämäläinen. Using genetic algorithm in self-organizing map design. In *Proceedings of the ICANNGA'95, Arles, France*, 1995.

[HBD95] M. Herrmann, H.-U. Bauer, and R. Der. Optimal magnification factors in self-organizing feature maps. In F. Fogelman-Soulié and P. Gallinari, editors, *Proc. ICANN'95, Int. Conf. on Artificial Neural Networks*, volume I, pages 75–80, Nanterre, France, 1995. EC2.

[HRKB96] P. Hahn, J. Rahmel, B. Krapohl, and C. Blum. Classification of Movement Pattern - The Ulnar Nerve Palsy. *Journal of Hand Surgery*, 1996. submitted.

[HRW96] St. Huwer, J. Rahmel, and A.v. Wangenheim. A new image registration method for local deformations. *Pattern Recognition Letters*, 1996. to appear.

[Kel91] M. Kelly. Self-organizing map training using dynamic k-d trees. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*. Elsevier, 1991.

[KKL90] Jari A. Kangas, Teuvo K. Kohonen, and Jorma T. Laaksonen. Variants of Self-Organizing Maps. *IEEE Trans. Neural Networks*, 1(1):93–99, 1990.

[KO90] P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. In *Proc. of the IJCNN*, 1990.

[Koh84] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.

[Koh90] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[Koh95] Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, Heidelberg, 1995.

[Koi94] P. Koikkalainen. Progress with the tree-structured self-organizing map. In A. Cohn, editor, *Proc. of the ECAI*, 1994.

[LFJ92] T. Li, L. Fang, and A. Jennings. Structurally adaptive self-organizing neural trees. In *Proc. of the ICNN*, 1992.

[Lin89] R. Linsker. How to generate maps by maximizing the mutual information between input and output signals. *Neural Computation*, 1:402–411, 1989.

[LO89] J. Lampinnen and E. Oja. Fast self-organization by the probing algorithm. In *Proc. of the IJCNN*, Washington, 1989.

[Lut88] S. P. Luttrell. Self-organizing multilayer topographic mappings. In *Proc. ICNN'88, Int. Conf. on Neural Networks*, volume I, pages 93–100, Piscataway, NJ, 1988. IEEE Service Center.

[Mar93] Th. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *Proc. of the ICANN*, Amsterdam, 1993.

[MBS93] Thomas M. Martinetz, Stanislav G. Berkovich, and Klaus J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Trans. on Neural Networks*, 4(4):558–569, 1993.

[MS91] K. Martinetz and K. Schulten. A Neural-Gas Network Learns Topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 397–402, 1991.

[MS94] Thomas Martinetz and Klaus Schulten. Topology representing networks. *Neural Networks*, 7(2), 1994. (in press).

[NY88] N.M. Nasrabadi and Feng Y. Vector Quantization of Images based Upon the Kohonen Self-Organizing Feature Maps. In *IEEE Int. Conf on Neural Networks*, San Diego, 1988.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufman, 1993.

[Rah95] J. Rahmel. Similarity-based self-organized clustering. In *EPIA 95: Workshop Fuzzy Logic and Neural Networks in Engineering*, 1995.

[Rah96a] J. Rahmel. On the Role of Topology for Neural Network Interpretation. In W. Wahlster, editor, *Proc. of the ECAI*, 1996.

[Rah96b] J. Rahmel. SplitNet: A Dynamic Hierarchical Network Model. In *Proc. of the AAAI*, 1996.

[Rah96c] J. Rahmel. SplitNet: Learning of Hierarchical Kohonen Chains. In *Proc. of the ICNN '96*, Washington, 1996.

[RH96] J. Rahmel and P. Hahn. A Tree-Structured Approach to Medical Diagnosis Tasks. In F. Kurfess, editor, *ECAI-Workshop Neural Networks and Structured Knowledge*, 1996. to appear.

[RMS91] H. Ritter, Th. Martinetz, and K. Schulten. *Neural Networks.* Addison Wesley, 1991.

[RS86] H. Ritter and K. Schulten. On the stationary state of Kohonen's self-organizing sensory mapping. *Biol. Cyb.*, 54:99–106, 1986.

[TG74] J. T. Tou and R. C. Gonzales. *Pattern Recognition Principles.* Addison-Wesley, 1974.

[Utg88] P. Utgoff. Perceptron trees: A case study in hybrid concept representations. In *Proc. of the Nat. Conf. on AI*, pages 601–606, St. Paul, MN, 1988.

[VDHM94] Th. Villmann, R. Der, M. Herrmann, and Th. Martinetz. Topology preservation in self-organizing feature maps: General definition and efficient measurement. In B. Reusch, editor, *Fuzzy Logik, Theorie und Praxis.* Springer, 1994.

[VDHM96] Th. Villmann, R. Der, M. Herrmann, and Th. Martinetz. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, 1996. To appear.

[VDM94] Th. Villmann, R. Der, and Th. Martinetz. A new quantitative measure of topology preservation in Kohonen's feature maps. In *Proc. of the ICNN*, pages 645–648, 1994.

[WdM76] D. J. Willshaw and C. Von der Malsburg. How patterned neural connections can be set up by self–organization. *Proceedings of the Royal Society of London, Series B*, 194:431–445, 1976.

[Zad82] P. L. Zador. Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Transaction on Information Theory*, (28):149–159, 1982.

[Zah71] C. T. Zahn. Graph-theoretical methods for detection and describing gestalt clusters. *IEEE Transactions on Computers*, C-20(1):68–86, January 1971.