Pergamon

Neural Networks, Vol. 7, No. 9, pp. 1441–1460, 1994

0893-6080/94 $6.00 + .00

0893-6080(94)E0049-Q

CONTRIBUTED ARTICLE

Growing Cell Structures—A Self-Organizing Network for Unsupervised and Supervised Learning

BERND FRITZKE

Institut für Neuroinformatik

(*Received* 25 *May* 1993; *revised and accepted* 23 *March* 1994)

Abstract—*We present a new self-organizing neural network model that has two variants. The first variant performs unsupervised learning and can be used for data visualization, clustering, and vector quantization. The main advantage over existing approaches (e.g., the Kohonen feature map) is the ability of the model to automatically find a suitable network structure and size. This is achieved through a controlled growth process that also includes occasional removal of units. The second variant of the model is a supervised learning method that results from the combination of the above-mentioned self-organizing network with the radial basis function (RBF) approach. In this model it is possible—in contrast to earlier approaches—to perform the positioning of the RBF units and the supervised training of the weights in parallel. Therefore, the current classification error can be used to determine where to insert new RBF units. This leads to small networks that generalize very well. Results on the two-spirals benchmark and a vowel classification problem are presented that are better than any results previously published.*

Keywords—Self-organization, Incremental learning, Radial basis function, Clustering, Data visualization, Pattern classification, Two-spiral problem, Feature map.

1. INTRODUCTION

Self-organizing neural network models, as proposed by Willshaw and von der Malsburg (1976) and Kohonen (1982), generate mappings from high-dimensional signal spaces to lower-dimensional topological structures. These mappings are able to preserve neighborhood relations in the input data and have the property to represent regions of high signal density on correspondingly large parts of the topological structure. This makes them interesting for applications in various areas ranging from speech recognition (Kohonen, 1988) and data compression (Schweizer et al., 1991) to combinatorial optimization (Favata & Walker, 1991). The fact that similar mappings can be found at various places in the brains of humans and animals indicates that preservation of topology is an important principle at least in natural "signal processing systems."

It has been noted that the predetermined structure and size of Kohonen's model imply limitations on the resulting mappings. Often one realizes only at the end of a simulation that a different shape or number of elements would have been more appropriate. On the other hand, there is in most cases no a priori information available that would allow to choose a suitable network size and shape in advance.

A solution to this dilemma is to determine shape as well as size of the network *during* the simulation in an incremental fashion. This is the main principle of the model presented below. It has a flexible, problem-dependent structure, a variable number of elements, and a k-dimensional topology whereby k can be arbitrarily chosen. Recently it was demonstrated that the new model improves over Kohonen's feature map with respect to various important criteria (Fritzke, 1993a). We acknowledge, however, that the new model owes several ideas to Kohonen's approach and that it is an extension of his work rather than a completely different formalism.

First we outline the network for unsupervised learning and introduce later on the extension of the model to supervised learning.

This work was performed mainly during a stay of the author at the International Computer Science Institute at Berkeley.

Acknowledgements: The author likes to thank Scott Fahlman for the permission to reproduce Figure 20b and for maintaining the CMU Benchmark Collection. Moreover, the two anonymous reviewers deserve thanks for their most helpful comments.

Requests for reprints should be sent to Bernd Fritzke, Institut für Neuroinformatik, Ruhr-Universität Bochum, 44780 Bochum, Germany; E-mail: fritzke@neuroinformatik.ruhr-uni-bochum.de

2. UNSUPERVISED GROWING CELL STRUCTURES

2.1. Problem Definition

Before we describe our network model, it seems appropriate to exactly define the kind of problems the network is supposed to solve. In the first place, we have a number of n-dimensional input signals obeying an unknown probability distribution $P(\xi)$. With $V = R^n$ we denote the vector space the input signals stem from.

Our objective is to generate a mapping from V onto a discrete k-dimensional topological structure A. This mapping should have the following properties:

- Similar input signals are mapped onto topologically close elements of A.
- Topologically close elements in A should have similar signals being mapped onto them.
- Regions of V where the probability density of the input vector distribution is high should be represented by correspondingly many elements in A.

The first two points mean that the mapping should preserve similarity relations in forward and backward direction. If the dimensionality of A is smaller than that of V, a dimensionality reduction is performed. If it is in spite of that possible to preserve the similarity relations, then the complexity of the data is reduced without loss of information. The third point means that we gain some information about the unknown probability density of the input signals.

2.2. Network Architecture

The initial topology of the network A is a k-dimensional simplex. For $k = 1$ this is a line segment, for $k = 2$ a triangle, and for $k = 3$ or higher the structure is denoted tetrahedron or hypertetrahedron. The $(k + 1)$ vertices of the simplex are the *cells* (or neurons). The $(k + 1)k/2$ edges denote topological neighborhood relations. During a self-organization process described further below new cells will be added to the network and superfluous cells will be removed. Every modification of the network, however, is performed such that afterwards the network consists solely of k-dimensional simplices again. Some typical structures for different values of k are shown in Figure 1.

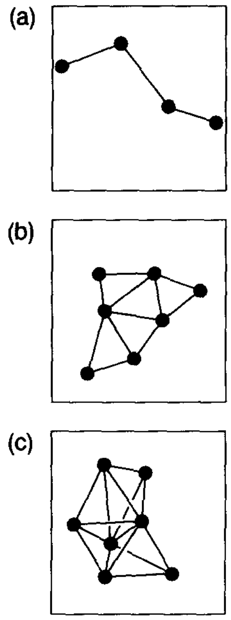

FIGURE 1. Cell structures of different dimensionality k. (a) $k = 1$, (b) $k = 2$, (c) $k = 3$.

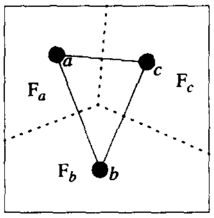

FIGURE 2. Voronoi tessellation generated by a two-dimensional cell structure with the initial triangular topology. The dimension of the input vector space V is also two in this example. Every neuron is projected into V by drawing a circle at the position the reference vector points to. Circles corresponding to topologically neighboring neurons are connected by lines.

We choose hypertetrahedrons for our model because they are of minimal complexity and can, therefore, be easily combined to larger structures. One should also note that the number of vertices of a k-dimensional hypertetrahedron grows only linear with k, whereas, for example, a k-dimensional *hypercube* has an exponentially growing number of vertices (2^k). Therefore, the hypertetrahedron is a good choice even for very high-dimensional networks.

Every cell c has an n-dimensional synaptic vector w_c attached. This vector may be seen as the *position* of c in the input vector space. We denote with w the set of all synaptic vectors w_i, $i \in A$. A mapping ϕ_w from the input vector space V onto the network A can now be defined by mapping every input signal to the cell with the nearest position (or reference vector). More formally we write

$$\phi_w : V \to A, (\xi \in V) \mapsto (\phi_w(\xi) \in A) \qquad (1)$$

with $\phi_w(\xi)$ the so called *best-matching unit* being defined through

$$\|w_{\phi_w(\xi)} - \xi\| = \min_{r \in A} \|w_r - \xi\|. \qquad (2)$$

Thereby $\| \cdot \|$ denotes the Euclidean vector norm. By this V is partitioned into a number of regions $F_i (i \in A)$, each consisting of the locations having a common nearest synaptic vector w_i (see Figure 2). This is known as *Voronoi tessellation,* and the regions are denoted *Voronoi regions.* To simplify some of the following formulas, we assume that our input space V is an arbitrarily large but *finite* subregion of R^n. The consequence of a finite input space is that all Voronoi regions are finite. This is in general not true for those Voronoi regions belonging to a synaptic vector on the convex hull of w.

2.3. Network Dynamics

In principle the adaptation of the synaptic vectors in our model is done as earlier proposed by Kohonen (1982):

1. Determine the best-matching unit for the current input signal.
2. Increase matching at the best-matching unit and its topological neighbors.

In Kohonen's model the strength of the adaptation is decreasing according to a cooling schedule. Moreover, the topological neighborhood inside which significant changes are made is chosen large at the beginning and decreases then, too. Our model follows the same basic strategy. There are, however, two important differences:

- The adaptation strength is constant over time. Specifically, we use constant adaptation parameters ε_b and ε_n for the best-matching unit and the neighboring cells, respectively.
- Only the best-matching unit and its direct topological neighbors are adapted.

These choices eliminate the need to define a cooling schedule for any of the model parameters.

In the following N_c denotes the set of direct topological neighbors of a cell c. Furthermore, we define for every cell c a local counter variable τ_c basically containing the number of input signals for which the cell has been best-matching unit. Because the cells are slightly moving around, more recent signals should be weighted stronger than previous ones. This is achieved by decreasing all counter variables by a certain fraction after each adaptation step. To enable this decay, the signal *counters* must be represented by real-valued variables.

An adaptation step in our model can be formulated as follows[1] (see also Figure 3):

1. Choose an input signal ξ according to the probability distribution $P(\xi)$.
2. Locate the best matching unit $s = \phi_w(\xi)$.
3. Increase matching for s and its direct topological neighbors

$$\Delta w_s = \varepsilon_b(\xi - w_s) \qquad (3)$$

$$\Delta w_c = \varepsilon_n(\xi - w_c) \quad (\text{for all } c \in N_s) \qquad (4)$$

4. Increment the signal counter of s:

$$\Delta\tau_s = 1. \qquad (5)$$

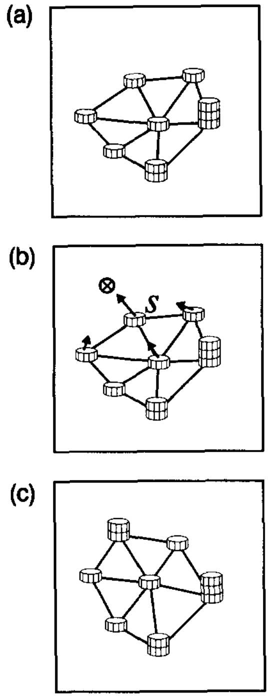

FIGURE 3. One adaptation step for a two-dimensional cell structure. Only the best-matching unit and its direct neighbors are adapted. The columns represent signal counter values. The signal counter of the best-matching unit is incremented. (Here and in the following figures we project the network into the input vector space by drawing every cell at the position the corresponding reference vector points to. This is a useful technique if the input vector space has a dimension ≤3.) (a) Initial situation; (b) occurrence of an input signal; (c) after adaptation.

5. Decrease all signal counters by a fraction α:

$$\Delta\tau_c = -\alpha\tau_c \quad (\text{for all } i \in A).$$

If we choose small values for ε_b and ε_n, then the cells move from their initial random positions to locations with a dynamic equilibrium between the changes in all directions. They do not stop moving completely because the adaptation parameters are not decreased (so this is not stochastic approximation).

Our objective is a structure with the synaptic vectors w_c distributed according to $P(\xi)$. This is achieved when every cell has the same probability of being best-matching unit for the current input vector. We do not know $P(\xi)$ explicitly, but with the local signal counters we can compute an estimate of $P(\xi)$, namely the relative frequency of input signals received by a certain cell.

The *relative signal frequency* of a cell c is

$$h_c = \tau_c / \sum_{j \in A} \tau_j. \qquad (6)$$

Eventually, all cells should have similar relative signal frequencies. A high value of h_c, therefore, indicates a good position to insert a new cell because the new cell is likely to reduce this high value to a certain degree.

[1] Here, and throughout the whole paper, $\Delta x = y$ stands for $x^{\text{new}} = x^{\text{old}} + y$. This is to have a concise notation for incremental changes.

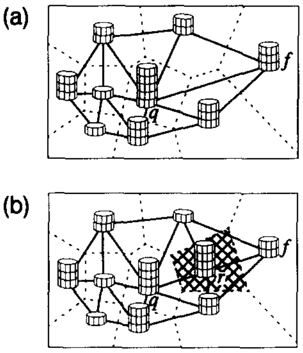

FIGURE 4. Insertion of a new cell. (a) Situation before an insertion. The columns represent signal counter variables. The cell *q* has received the most input signals so far. The grey lines indicate the Voronoi tessellation. (b) A new cell *r* has been inserted and, thus, a new Voronoi region exists now. The signal counter variables are redistributed according to the changes of the Voronoi regions.

Thus, in the following insertions are made on the basis of this criterion.

Always after a fixed number λ of adaptation steps we determine the cell q with the property

$$h_q \geq h_c \quad (\text{for all } c \in A). \tag{7}$$

Then we look for the direct neighbor of q with the largest distance in input space. This is a cell f (see Figure 4a) satisfying

$$\|w_f - w_q\| \geq \|w_c - w_q\| \quad (\text{for all } c \in N_q). \tag{8}$$

We insert a new cell r in between q and f (Figure 4b). This new cell is connected to the other cells in such a way that we have again a structure consisting only of k-dimensional simplices.[2] The synaptic vector of r is initialized as

$$w_r = 0.5(w_q + w_f). \tag{9}$$

The insertion of r leads to a new Voronoi region F_r in the input space. At the same time the Voronoi regions of the topological neighbors of r are diminished. This change is reflected by an according redistribution of the counter variables τ_c. We compute the changes of the signal counters as

$$\Delta\tau_c = \frac{|F_c^{(\text{new})}| - |F_c^{(\text{old})}|}{|F_c^{(\text{old})}|}\,\tau_c \quad (\text{for all } c \in N_r). \tag{10}$$

whereby $|F_c|$ is the n-dimensional volume of F_c. Finally, the initial value of the new cell is defined as

$$\tau_r = -\sum_{c \in N_r} \Delta\tau_c. \tag{11}$$

The redistribution of the counter variables can be seen as ascribing to the new cell as much input signals as it would have received if it had existed since the beginning of the process. In the same way the reduction of the counter variables of its neighbors can be motivated. The basic algorithm for the growing cell structures is shown in Figure 5. A schematic example of the process is shown in Figure 6. The main characteristic of the model is that several adaptation steps are always followed by a single insertion. One can note the following feedback relation between the two types of action:

- Every adaptation step increases the signal counter of the best-matching unit and thereby increases the chance that another cell will be inserted near this cell.
- Insertion near a cell c decreases both the size of its Voronoi field F_c and the value of the signal counter τ_c. The reduction of the Voronoi field makes it less probable that c will be best-matching unit for future input signals.

Our simulations indicate that—under a wide range of parameter settings—the model approaches a state where for every cell i the probability p_i that i is best-matching unit for the next input signal according to $P(\xi)$ is approximately equal. In this case the entropy

$$S = -\sum_{c \in A} p_c \log p_c \tag{12}$$

is approximately maximized and, therefore, the local density of reference vectors gives a good estimate of the unknown probability density of the input vectors. The above-mentioned comparative study indicates that the growing cell structures estimate unknown probability distributions significantly better than Kohonen's feature maps (Fritzke, 1993a).

In Figure 7 some stages of a simulation are depicted. The cell structure grows, guided by the input vectors, and finally finds a suitable structure to model the cloud-shaped distribution. One should note that already in early phases of the simulation the network has basically its final shape only with fewer neurons. This behavior can be described as *fractal growth*, which can be observed frequently in plants (e.g., in ferns). An important property of this kind of change is that we can interrupt the process at any time and still have a well-shaped structure.

```
Start with a $k$-dimensional simplex at random positions in $V = R^n$.
while (desired network size not reached)
    Perform a constant number $\lambda$ of adaptation steps.
    Insert a new cell and distribute the counter variables according to
    eqn. 10 - 11.
```

FIGURE 5. Principal algorithm of growing cell structures.

[2] This can be achieved by, first, connecting r to q, f and to those common neighbors of q and f which are part of a simplex having both q and f are vertices. Second, the original connection between q and f has to be removed.

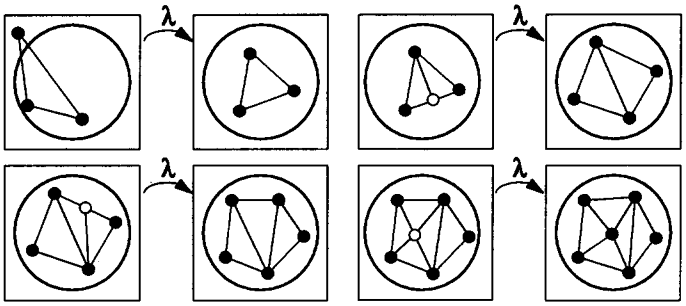

FIGURE 6. A two-dimensional cell structure grows directed by input signals stemming from a uniform distribution in a circle shaped subarea of R^2. The initial structure is a triangle of neurons with randomly initialized reference vectors. The structure is distributed by a constant number λ of input signals. Then a new cell (white circle) is inserted and connected to the other neurons in such a way that again a structure of triangles results. This new structure is distributed again, an other cell is inserted, etc.

Another property of our model that becomes especially evident when viewing computer simulations is that, once a certain number of cells has been created, very little movement of the reference vectors occurs. The main source of change is the insertion of new cells. It is this property that facilitates the extension of the model to a new supervised learning method, as will be demonstrated in Section 3.

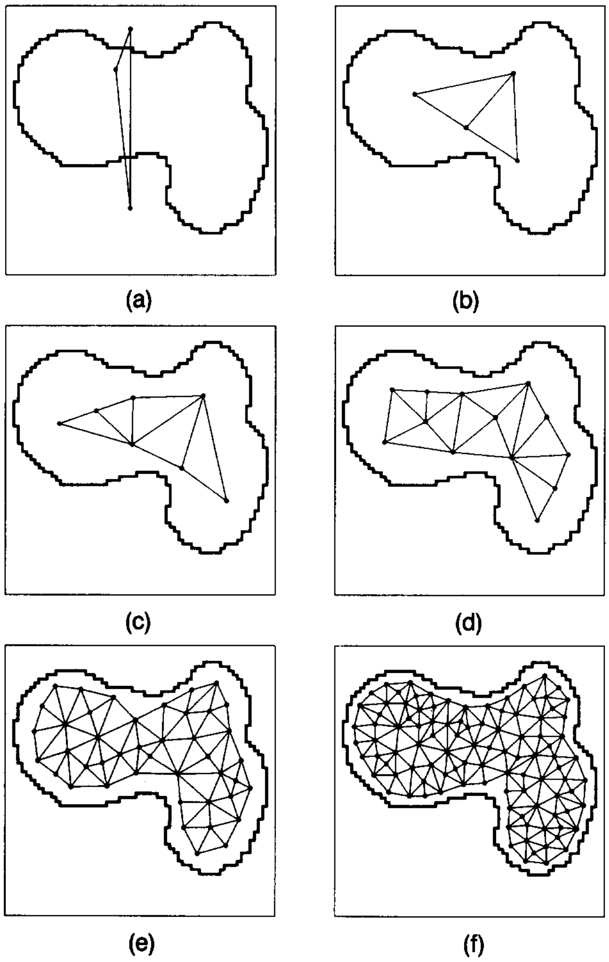
(a) (b) (c) (d) (e) (f)

FIGURE 7. Development of a two-dimensional growing cell structure. The underlying probability distribution $P(\xi)$ is in this case also two-dimensional and is uniform in a cloud-shaped area. Below every subpicture the number of already received input signals (which is the number of adaptation steps) is shown. The employed simulation parameters are $\lambda = 100$, $\varepsilon_b = 0.06$, $\varepsilon_n = 0.002$, $k = 2$, $\alpha = 0.05$. (a) 0 signals, (b) 100 signals, (c) 400 signals, (d) 1000 signals, (e) 4000 signals, (f) 10000 signals.

2.4. Removal of Cells

In some cases, especially if $P(\xi)$ consists of several separate regions of positive probability density, a still better modeling can be achieved by removing *superfluous* cells. A cell can be regarded as superfluous if it has a position (synaptic vector) in a region of V with very low probability density. In general, $P(\xi)$ is unknown, but we can relate the relative signal frequency of a cell

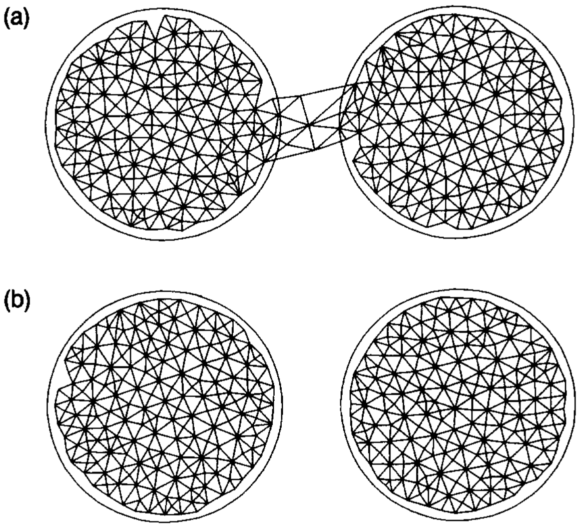
(a)
(b)

FIGURE 8. A growing cell structures network with 400 cells has adapted to the probability distribution of the previous example. Simulation parameters: $\varepsilon_b = 0.06$, $\varepsilon_n = 0.002$, $\lambda = 100$, $\alpha = 0.05$, and $\eta = 0.09$ (for b only). (a) The growth process leads to a well-adapted network structure. Mostly short connections indicate good topology preservation. Few synaptic vectors lie outside the relevant circular areas. (b) By removal of *superfluous* cells substructures can be formed. The positions of the synaptic vectors now indicate a nearly perfect modeling of the probability distribution and there are only short connections.

to the size of its *receptive field* (Voronoi field) to get a local estimate. Specifically, one can note that

$$\tilde{p}_c = h_c / |F_c| \tag{13}$$

is a local estimate of the probability density near w_c. By periodically removing cells with values of $\tilde{p}$ below some threshold η we can model even structured distributions very accurately. Figure 8 shows the simulation results for such a probability distribution without and with removal of neurons.

2.5. Approximation of the Voronoi Regions

The computation of the Voronoi tessellation is very difficult for dimensions $n > 2$. Thus, we replaced the Voronoi region F_c in a first approach by an n-dimensional hypercube with a side length equal to the mean length $\bar{l}_c$ of the edges emanating from c. Instead of $f_c = |F_c|$ we took

$$\tilde{f}_c = (\bar{l}_c)^n, \tag{14}$$

with $\bar{l}_c$ computed by

$$\bar{l}_c = 1/card(N_c) \sum_{i \in N_c} \|w_c - w_i\|. \tag{15}$$

Although the estimation of the probability density is not anymore as accurate as before, it seems to be sufficient to reliably identify superfluous neurons. However, the appropriate value for the threshold depends strongly on the probability distribution. The reason for that is that a probability distribution that is nonzero in a large area of the input vector space has a lower density than a distribution that is concentrated more locally. But if one uses instead of $\tilde{p}$ the normalized value $\hat{p}$, defined as

$$\hat{p} = \tilde{p} \sum_{c \in A} \tilde{f}_c, \tag{16}$$

which is computed by multiplying $\tilde{p}$ with the total volume of all hypercubes, one gets sufficient independence. A threshold value of $\eta = 0.09$ is then appropriate in most cases. The check whether a cell i has a value $\hat{p}_i < \eta$ is performed after each insertion, and the cells fulfilling the condition are removed. The simulation leading to Figure 8b has been performed with this method.

One should note, however, that the choice of an n-dimensional hypercube is only appropriate if the underlying data indeed spans the n-dimensional space. If, on the other hand, the data stems from a lower-dimensional subspace of R^n, it might be better to use a hypercube of that dimensionality.

To illustrate the point, let us consider an extreme example. Assume that our input data is 100-dimensional (which is not uncommon for some real problems), but stems from a two-dimensional submanifold of R^{100} (what we do not know). We might take a two-dimensional network to be able to visualize the data (see Section 2.7). If now for one of our cells the mean edge length shrinks by 5%, then the volume of the corresponding 100-dimensional hypercube collapses to less than 0.6% of its previous size. Obviously, this does not reflect very well the change of the receptive field of the cell. In this case, taking two-dimensional hypercubes would have been more appropriate.

From the above it should be evident that it would be very helpful to know the *true* dimensionality of the data, meaning the smallest dimensionality t, such, that a t-dimensional submanifold of V can be found containing all (or most) input data. Then t-dimensional hypercubes could be used to estimate the size of the Voronoi regions in our model. Unfortunately, it is in general difficult to figure out the value of t, especially because the mentioned submanifold does not have to be linear but could be arbitrarily twisted (e.g., a curved surface in R^3). Therefore, even a principal component analysis of the data does not, in general, reveal their true dimensionality, but gives only (or at least) an upper bound.

As long as there is no simple method to determine the true data dimensionality t, one has to use an estimate $\tilde{t}$ of it. In the following we give some general rules for choosing such an estimate that do work well for all problems we encountered so far:

- Always set $t \leq n$.
- If the different components of the input vectors are known to be stochastically independent from each other use $\tilde{t} = n$.
- If there are known dependencies among the components set $\tilde{t}$to the number of independent variables.
- Always set $\tilde{t}$ smaller or equal to the number of input vectors. This rule applies only to the rather unusual case that the total number of vectors is smaller than their dimensionality n.
- Finally, one can perform a principal component analysis of the data. Then $\tilde{t}$ should be set to the number of large principal eigenvalues of the covariance matrix of the data.

However, in most cases it is not necessary to do the principal component analysis because our method is not very sensitive to the choice of $\tilde{t}$. The only case one should avoid is to choose a value of $\tilde{t}$ that is *much* too high. This can happen only if the data is very high-dimensional and there are strong dependencies among the components. In such a case it can happen that most insertions occur in one region of the structure. This is due to the fact that a newly inserted cell then gets attributed nearly all the signals of its neighbors because the change of their Voronoi fields is overestimated (see above). A simple remedy for this problem is to choose a lower value for $\tilde{t}$ and to perform another simulation.

In conclusion, we choose in the following in each case a value for $\tilde{t}$ and approximate the volume of the Voronoi regions by

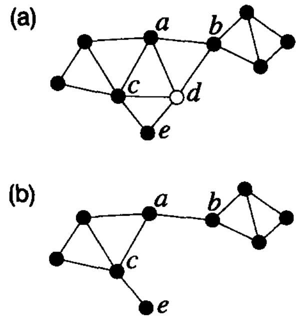

FIGURE 9. Simple heuristics for cell removal can lead to inconsistent structures. (a) Growing Cell Structures. The node *d* is to be removed. This is done by removing the adjacent edges and the node itself. (b) Structure after removal of node *d*. The edges $\overline{ab}$ and $\overline{ce}$ are not part of any triangle anymore. The structure is inconsistent.

$$\tilde{f}_c = (\bar{l}_c)^{\tilde{\iota}} \qquad (17)$$

with $\bar{l}_c$ being the mean edge length [see eqn (15)]. It should be stressed again, however, that the choice of $\tilde{\iota}$ seems not at all to be a critical step. For a given set of data usually many different estimates work well.

2.6. Efficient Manipulation of High-Dimensional Topologies

The implementation of k-dimensional growing cell structures is somewhat more complicated than the implementation of the Kohonen feature map (for which usually a rectangular array of processing units is chosen). Therefore, it seems appropriate to give some hints how this can be done with relatively small effort.

Any implementation of the model must support the two structural update operations:

- Insertion of a neuron.
- Deletion of a neuron.

These operations have to be performed such that the resulting structure consists exclusively of k-dimensional hypertetrahedrons again.

The general structure of the network can be represented as an undirected graph that is a standard data type consisting of nodes and of edges between pairs of nodes.[3] The nodes correspond to neurons and the edges to topological neighborhood relations. Although such a data structure is already sufficient in principle, a considerable search effort is needed to make consistent update operations. The problem is that the removal of a neuron might require that also other neurons and connections are removed to make the structure consistent again. Simple heuristics as, for example, *to remove a node remove all neighboring connections and the node itself,* do not work properly, as is shown in Figure 9. The key idea to solve this problem is to change the level of observation from nodes and connections to hypertetrahedrons. For this purpose we keep track of all the hypertetrahedrons the current network consists of. Technically, a new data type *simplex* is created, an instance of which contains the set of all nodes belonging to a certain hypertetrahedron. Furthermore, with every node we associate the set of those hypertetrahedrons the node is part of. The two update operations can now be formulated as follows:

- Insertion: A new node r is always inserted by splitting an existing edge $\overline{qf}$. The node r has to be connected with q, f, and with all common neighbors of q and f. Also, the hypertetrahedrons have to be updated. Each hypertetrahedron h containing both q and f (in other words, the edge being split) is replaced by two hypertetrahedrons each containing the same set of nodes as h except that q respectively f is replaced by the new node r. Finally, the original edge $\overline{qf}$ is removed. The new hypertetrahedrons have to be inserted in the sets associated with their participating nodes.
- Deletion: To delete a node, it is necessary and sufficient to delete all hypertetrahedrons the node is part of. This is done by removing the hypertetrahedrons from the sets associated with their nodes. Edges for which the respective ending nodes do not share at least one hypertetrahedron are removed. The same is done with nodes having no more edges. This strategy leads to structures with every edge belonging to at least one hypertetrahedron and every node to at least one edge. Therefore, the resulting k-dimensional structures are consistent, that is, contain only k-dimensional hypertetrahedrons.

It is demonstrated in Figure 10 that the problematic example of Figure 9 is now handled correctly.

2.7. Network Visualization for High-Dimensional Input Data

An important property of Kohonen's feature map is the ability to project high-dimensional input data onto a two-dimensional, usually rectangular, grid. This makes a visualization of complex data possible, for example, speech data (Kohonen, Mäkisara, & Saramäki, 1984) or even high-dimensional symbolic descriptions of objects (Ritter & Kohonen, 1989).

The growing cell structures generate less regular networks. In the two-dimensional case the network consists of a number of connected triangles, possibly also of several such networks if removal of cells has been performed. By construction the network is two-dimensional, but it is not obvious how to embed the network into the plane to visualize it. On the other hand, the method of projecting the network into input

[3] Our current implementation of the model is based on LEDA (see Mehlhorn & Näher, 1989), a publicly available library of data types and algorithms. LEDA contains, in particular, a very elaborated data type "graph."

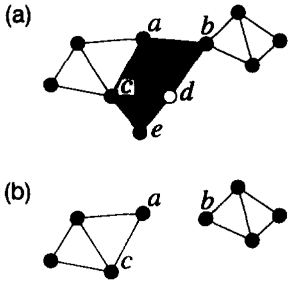

FIGURE 10. Correct removal through introduction of additional structural information. (a) Growing cell structures. The node *d* is to be removed and consequently also those triangles (two-dimensional hypertetrahedrons) in which *d* participates. (b) Structure after removal of *d* and the triangles *d* participated in. The structure consists only of triangles again and is thus consistent.

vector space allows a visualization only for input vector dimensions up to three.

We found, however, a method to embed a k-dimensional network ($k \in \{2, 3\}$) into the k-dimensional space. This makes it possible to visualize networks for arbitrarily high vector dimensions as long as the network dimension is low enough.

Our method employs a simple physical model to construct the k-dimensional embedding during the self-organization process. In the following we assume $k = 2$. The generalization to three dimensions is straightforward.

- Each cell in the network is modeled by a disc made of elastic material.
- The diameter of each disc is d. Therefore, two discs touch if the distance of their centers is equal to d. If the distance gets smaller than d, the discs repel each other according to the occurring elastic deformation.
- Each neighborhood connection is modeled by an elastic string. Two connected, but currently not touching, discs are pulled towards each other.
- All discs are positively electrically charged and repel each other (even if they do not touch).

At the beginning of the self-organization process the three discs are positioned in the plane such that they do not overlap. Each time a new cell is inserted, the position of its corresponding disc is interpolated from the neighbors in the same way the reference vector is interpolated. It may occur that overlaps now exist. Therefore, after every insertion we compute for every disc the sum of forces acting on it and move it accordingly. This is done in an asynchronous manner to avoid oscillation effects.

We did not try to build a physically accurate model. The discs have no associated mass and forces lead to proportionally large motions. For the forces we experimentally determined the following values.

Repelling force f_b of two discs with center distance e:

$$f_b = \begin{cases} 0 & \text{if } 3d < e \\ d/5 & \text{if } 2d < e \leq 3d \\ d/2 & \text{if } d < e \leq 2d \\ d & \text{if } 0 < e \leq d \\ 0 & \text{if } 0 = e \end{cases} . \qquad (18)$$

TABLE 1
Animal Names and Binary Attributes (adapted from Ritter & Kohonen, 1989)

Animal	Dove	Hen	Duck	Goose	Owl	Hawk	Eagle	Fox	Dog	Wolf	Cat	Tiger	Lion	Horse	Zebra	Cow
Is																
Small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
Medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
Big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
Has																
Two legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Four legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
Hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
Mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
Feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Likes to																
Hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
Run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
Fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
Swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

If an attribute applies for an animal the corresponding table entry is 1, otherwise 0.

Attracting force f_n of two connected discs with center distance e:

$$f_n = \begin{cases} 0 & \text{if} \quad e < d \\ (e - d)/2 & (\text{otherwise}) \end{cases}. \tag{19}$$

These two forces have to be balanced against each other. We usually multiplied f_b by 0.2 and f_n by 1.0. In some cases, however, different values might be more appropriate.

An example of the results obtained by the described method is shown in Figure 11. A still larger benefit can be gained by having the embedding when the input data is so high-dimensional that we cannot visualize the network in input vector space anymore. This is in many real applications the case because often we have data consisting of many more than three components.

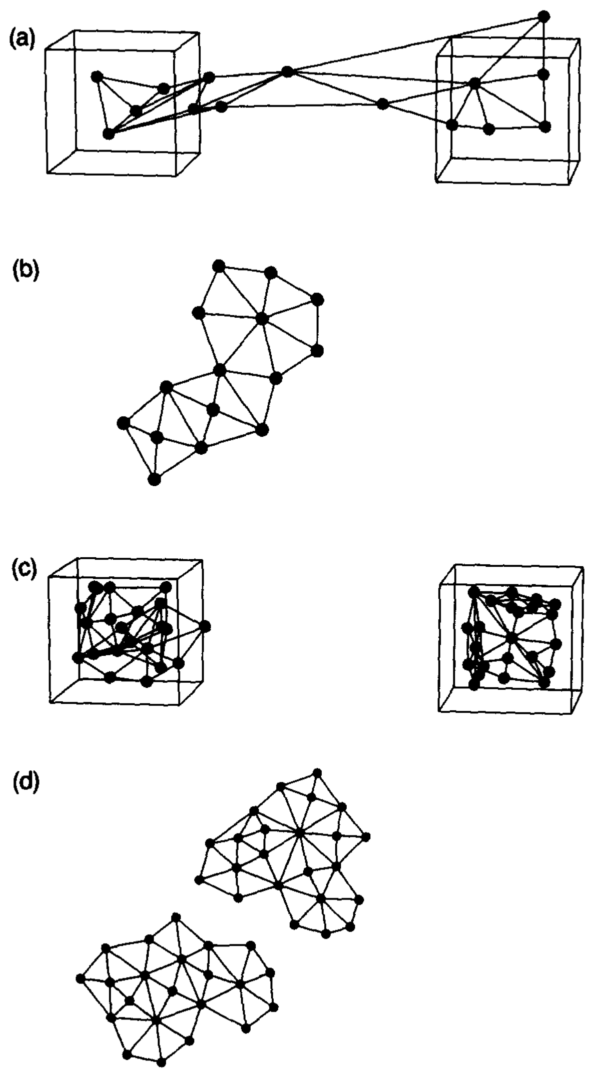

FIGURE 11. Example for the embedding method. The probability distribution is uniform in two separated cubes. The network is two-dimensional. (a) and (b) as well as (c) and (d) show the same state of the simulation, respectively. Through the embedding it is easily possible to detect the splitting of the network, as can be seen. (a) Projection into input vector space. Shortly after the start of the simulation all cells are connected because no deletions took place yet. (b) Embedding into the two-dimensional space. (c) Projection into input vector space. Deletion of superfluous cells has led to two separate structures. (d) Embedding into the two-dimensional space. The two substructures can be recognized easily.

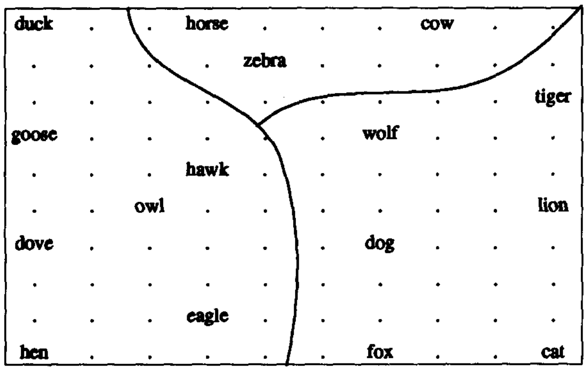

FIGURE 12. Kohonen feature map representing the animal data from Table 1. For every animal the cell is shown that is best-matching unit for the corresponding (feature) vector. Animals with similar properties are represented in neighboring locations of the map, as is shown by the (manually added) partition into three regions (adapted from Ritter & Kohonen, 1989).

Ritter and Kohonen have introduced an illustrative example of high-dimensional data. It consists of the description of 16 animals by binary property lists (Table 1). The thirteen properties together with a 1-out-of-n coding of the name of the animal led to 29-dimensional vectors. These vectors were fed into a two-dimensional Kohonen feature map consisting of 10 × 10 neurons. After the end of the self-organization process it was tested where each of the input vectors was represented on the map. It came out that Kohonen's method had found an interesting projection positioning similar animals generally at neighboring locations on the map. It was, e.g., possible to partition this *semantotopic* map into three connected regions containing all birds, herbivores, and carnivores, respectively (Figure 12).

We tested the growing cell structures with the same data and constructed during the self-organization a two-dimensional embedding of the network with the method just described. Two different stages of a specific simulation are shown in Figure 13. When comparing the results with those of Ritter and Kohonen, the main advantage of our model lies in the fact that it automatically finds meaningful partitions of the data, whereas Ritter and Kohonen had to identify those partitions by themselves.

In general, this technique makes it possible to visualize and cluster high-dimensional data that might be useful in many application areas as, for example, process control or pattern recognition.

2.8. Alternative Insertion Criteria

One goal of our model, as described so far, is to estimate the unknown probability density of the input signals

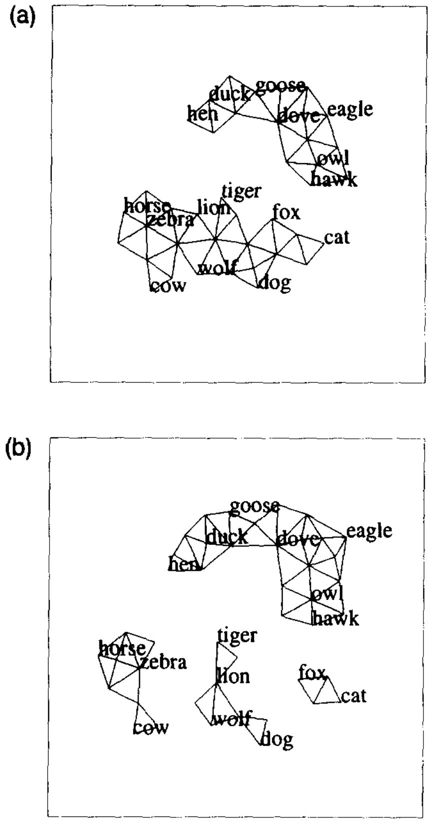

FIGURE 13. Semantotopic growing cell structures. The data used stems from Ritter and Kohonen (see Table 1). The data is ordered as by Kohonen's model but beyond that the ability of the growing cell structures to form substructures makes it possible to partition the data in clusters of mutually similar items. (a) The birds have been divided from the mammals. Among the birds the peaceful ones and the birds of prey are at different positions. Also in the mammal cluster similar animals can generally be found at neighboring positions. (b) The mammal cluster has been split into three other cluster. One contains the large and peaceful animals (horse, zebra, cow), the second contains animals that like to run (tiger, lion, wolf, dog), and the third cluster contains animals which like to hunt, but avoid excessive running (cat, fox).

with the local density of reference vectors in input vector space. This goal would be achieved perfectly if every neuron had the same chance that a randomly drawn input signal was mapped onto it. To approach this goal, we introduced a local signal counter for each neuron and inserted new neurons near existing neurons with high signal counter values.

It has to be pointed out that there is an underlying general principle in this method that can be exploited to achieve quite different goals than estimation of the probability density. The principle is to insert new neurons in such a way that the expected value of a certain error measure, which will be called *resource*[4] in the following, becomes equal for all neurons. Appropriate resources must have the property that the insertion of a new neuron r near an existing neuron q reduces the expected value of the resource of q. Under some additional conditions for the resource, which can be characterized as *well-behavedness* and which are very often fulfilled, we can expect that the strategy of inserting new neurons near neurons with high resource values will lead to the desired result that all neurons have similar expected resource values.

One interesting example of an alternative resource is the quantization error generated by a neuron. This is simply the accumulated squared distance between the reference vector of this neuron and all input signals being mapped onto the neuron. Instead of incrementing the signal counter of the best-matching unit as we did earlier, we change it through

$$\Delta\tau_s = \|\xi - w_s\|^2, \qquad (20)$$

which effectively replaces eqn (5). By using this measure as insertion criterion, new neurons are inserted not anymore near those neurons getting the most input signals but rather near those neurons the input signals of which are very different from their reference vectors. The resulting network structures differ especially for probability distributions with a nonuniform probability density (see, e.g., Figure 14). Recently, this particular insertion criterion was used to develop a new method for vector quantization (see Fritzke, 1993b). For this application the consistency requirements for the structures have been loosened by also allowing separate cells (without any neighbors) to exist. The method is able to generate codebooks of exceptionally good quality.

Another useful example for the resource is discussed in the next section, where we report first results on a new supervised network based on the growing cell structures.

3. EXTENSION TO SUPERVISED LEARNING

3.1. Motivation

Self-organizing networks perform unsupervised learning. Frequently they generate ordered mappings of the input data onto some low-dimensional topological structure. In other cases they are used to partition the input data into subsets (or clusters) such that data items inside one subset are similar but items from different subsets are dissimilar.

In many situations, however, one has given input as well as corresponding output data. The problem is then to learn the underlying relation from a limited number of examples. For the sake of concreteness, let us in the following assume that our data consists of a number of pairs

$$(\xi_i, \zeta_i) \in R^n \times R^m$$

[4] This denotation stems from the idea that the accumulated resource values cause insertion (or growth) and, therefore, play a nutrition-like role for the network.

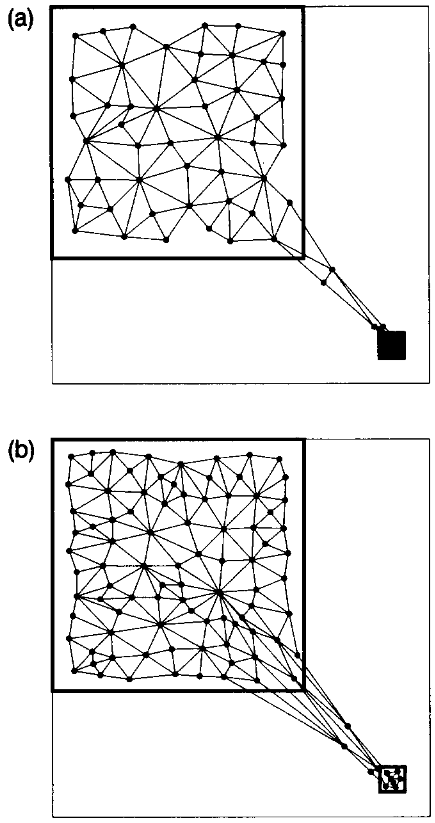

FIGURE 14. Minimization of quantization error. The probability distribution consists of a 10 × 10 field and a 1 × 1 field. Fifty percent of the input signal comes from either of these areas. After letting the networks grow until size 100 the mean square error was determined by 1000 test signals. (a) The original version of the growing cell structures leads to a solution with approximately 50% of the reference vectors in the 10 × 10 field as well as in the 1 × 1 field. The mean square error is 0.00095. (b) The error-minimizing variant of the growing cell structures positions most of the reference vectors in the 10 × 10 field. The mean square error is 0.00054.

whereby ξ_i is the input and ζ_i is the desired output of the ith pair.

Supervised learning methods are in these cases used to train networks to generate the desired output when they are presented with the input part of a specific data pair. Although this is not very useful per se, it is hoped that after finishing the training the network will be able to generate *reasonable* output values for unknown input data. This is often denoted as *generalization.* It is commonplace today that to achieve good generalization the number of free parameters of the network must be kept small. Otherwise there is the danger of *over-fitting,* which denotes a situation where the network still improves on the training data, but already has a decreasing performance on the test data. Typical application areas for supervised learning include pattern classification or function approximation.

In the following we demonstrate how the self-organizing model we presented in this paper can be extended to a supervised learning procedure. The result is a method that resembles the well-known radial basis function network (RBF) but eliminates some serious drawbacks of this approach.

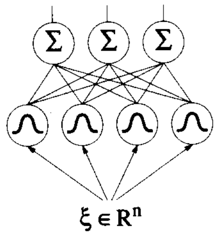

FIGURE 15. Radial basis function network. An n-dimensional input signal ξ is directed to a layer of units with a Gaussian activation function. This layer is via weighted connections linked to the output layer of linear summation units.

3.2. Radial Basis Functions

Radial basis function networks (Moody & Darken, 1989; Poggio & Girosi, 1990) consist of a layer L of units with Gaussian activation functions[5] and an output layer of m linear summation units (Figure 15). We assume again data pairs $(\xi_i, \zeta_i) \in R^n \times R^m$ of input and desired output.

Each Gaussian unit c has an associated vector $w_c \in R^n$ indicating the position of the Gaussian in input vector space and a standard deviation σ_c. For a given input datum ξ the activation of a unit c is described by

$$D_c(\xi) = \frac{f_c(\xi)}{\sum_{i \in L} f_i(\xi)} \tag{21}$$

whereby

$$f_c(\xi) = \exp\left(-\frac{\|\xi - w_c\|^2}{\sigma_c^2}\right). \tag{22}$$

Equation (21) realizes a normalization (proposed by Moody and Darken) such that

$$\sum_{i \in L} D_i(\xi) = 1 \tag{23}$$

always holds. Consequently, every input signal causes in summa the same activation. From the Gaussian units to the output units exists a complete layer of modifiable weights. The overall goal is to set the free parameters of the network such that the output units produce suitable values for given input data. The free parameters in this case are positions and widths of the Gaussians as well as the weights to the output units.

The usual procedure for training such a network consists of two consecutive phases, an unsupervised and a supervised one:

[5] In general, every activation function could be used that is only in a limited and local area of the input vector space considerably different from zero.

1. The Gaussians have to be positioned in the n-dimensional input vector space. Moody and Darken propose the k-means clustering algorithm for this purpose. Moreover, for each Gaussian the standard deviation has to be defined. Moody and Darken report good results for using the distance to the nearest other Gaussian.
2. The layer of modifiable weights has to be trained to produce the desired values at the output units. Commonly the delta rule (also called least mean square rule) is used, but also any conventional method for solving a linear system would do.

Although the described networks are reported to be computationally rather efficient (compared, for example, with back propagation), they have some important drawbacks. First, one has to define the number of Gaussians a priori. This leads to similar problems as the number-of-hidden-units dilemma for multilayer perceptrons because it is very difficult to estimate an appropriate number of units. The second problem stems from the fact that the k-means clustering algorithm positions the Gaussians at those locations in input vector space where many input vectors can be found. In some cases, this might be not at all optimal. Consider a simple classification problem with two classes where most of the data vectors lie in two well-separated clusters, but the remaining vectors of both classes are scattered in several small clusters that are quite close to each other (Figure 16). In this case k-means would position most of the available Gaussians on the two large clusters. A much better choice, however, would be to cover the large clusters with only few Gaussians (having a large standard deviation) and to use the rest to cover the more complicated region containing the small clusters. Generally, relatively more Gaussians should be positioned at those locations where it is difficult to differentiate between the classes. These locations, however, are not known a priori.

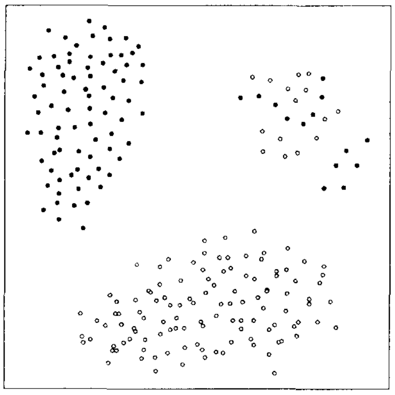

FIGURE 16. Classification problem with two classes: given the shown example, points find a good method to map all points in the square to one or the other class. (Alternatively, one could also consider rejection of points, for which neither class seems to be appropriate.)

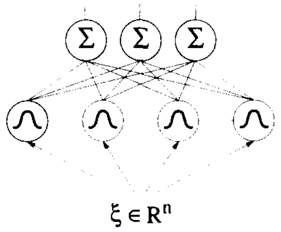

FIGURE 17. Supervised growing cell structures network. In contrast to the conventional radial basis function network there exist topological neighborhood relations (gray arrows) among the Gaussians. They are used to define the radius of the Gaussian as well as to interpolate the position of newly created Gaussians from existing ones.

3.3. Supervised Growing Cell Structures

In a fairly obvious way one can extend the growing cell structures to a supervised radial basis function network (Figure 17):

- For every cell c the reference vector w_c defines the center of a Gaussian activation function.
- The standard deviation σ_c of the Gaussian is defined as the mean length of all edges emanating from c (this is comparable to the heuristic proposed by Moody et al.).
- A number of m linear output units are defined and the Gaussian units are completely connected to them by weighted connections. This can be realized by associating with every cell c an output weight vector $w_c^{\text{out}} = (w_{1c}, w_{2c}, \ldots, w_{mc})$. Thereby, w_{ic} denotes the weight of the connection from cell c to output unit i.

So far this is very similar to a standard RBF network. The difference, however, lies in the training strategy and can be characterized by the following two points:

- Instead of having a two-phase scheme, the self-organization of the RBF layer and the supervised adaptation of the weighted connections are performed in parallel.
- The classification error occurring for the training data is used to determine where to insert new cells (resp. Gaussians).

The parallel training is made possible by the earlier mentioned property of our algorithm that existing weight vectors are moved (changed) only very little. Thus, it makes sense to train the weights to the output units right from the beginning of the growth process.

We have to extend the described algorithm for unsupervised learning accordingly. In particular, we now do one learning step with the delta rule after every adaptation step. As mentioned before, we assume that our data consists of pairs $(\xi_i, \zeta_i) \in R^n \times R^m$ of input vector and desired output vector. We compute the activation D_c of every cell c as

$$D_c(\xi) = \exp\left(-\frac{\|\xi - w_c\|^2}{\sigma_c^2}\right). \qquad (24)$$

We perform no normalization. This has the advantage that outliers do not activate any Gaussian very much and can therefore be identified easily. If we would normalize, on the other hand, input signals that are arbitrarily far away from all Gaussians can also activate them considerably. To support this position one could argue that it is somewhat questionable to *generalize* over patterns that are very different from all patterns seen during training.

The activation of the m output units is computed by

$$o_i = \sum_{c \in A} w_{ic} D_c \quad (\text{for all } i \in \{1, \ldots, m\}). \tag{25}$$

The change of weights (according to the delta rule) is defined by

$$\Delta w_{ic} = \eta(\zeta_i - o_i) D_c \quad (\text{for all } i \in \{1, \ldots, m\}) \quad (\text{for all } c \in A), \tag{26}$$

whereby η is the learning rate.

Finally, we update the resource variable of the current best-matching unit s by adding to it the overall squared error between actual output $o = o_1, \ldots, o_m$ and desired output $\zeta = \zeta_1, \ldots, \zeta_m$:

$$\Delta \tau_s = \| \zeta - o \|^2. \tag{27}$$

This replaces eqn (5), where we incremented the resource variable τ, with eqn (20), where we summed up the quantization error.

If the current task is a classification problem (as opposed to a continuous input/output mapping), we can alternatively use the classification error. In this case the resource would be updated according to

$$\Delta \tau_s = \begin{cases} 0 & \text{if } \xi \text{ is classified correctly} \\ 1 & \text{otherwise.} \end{cases} \tag{28}$$

Networks built with the classification error as insertion criterion still tend to be very small when they start classifying all training examples correctly. This is due to the fact that new cells are only inserted in those regions of the input vector space where misclassifications still occur. On the other hand, learning does practically halt when no misclassifications occur anymore, even if the raw mean square error of the network is still rather large. In some cases this can lead to poor generalization for unknown patterns. Therefore, it seems advisable to use a weighted combination of classification and mean square error. It has to be pointed out, however, that this is merely a matter of fine tuning. From our experience the networks usually generate satisfying mappings in all areas where training vectors are available, no matter which combination of the two kinds of error is used.

Whenever a new cell r is inserted, it gets a vector $w_r^{out} = (w_{1r}, w_{2r}, \ldots, w_{mr})$ of weighted connections to the m output units. Instead of initializing these vectors with zero or random values, they are obtained through a redistribution very similar to that used for the resource variable of the new cell [compare eqns (10) and (11)]:

$$\Delta w_c^{out} = \frac{|F_c^{new}| - |F_c^{old}|}{|F_c^{old}|} w_r^{out} \quad (\text{for all } c \in N_r). \tag{29}$$

whereby $|F_c|$ is the n-dimensional volume of F_c. Finally, the initial output weight vector of the new cell is defined as

$$w_r^{out} = -\sum_{c \in N_r} \Delta w_c^{out}. \tag{30}$$

In doing this redistribution the new cell is given output weights such that it will activate the output units in a way similar to its *mean* neighbor. Because the neighboring Gaussians overlap considerably, the overall output behavior of the network is not changed very much. In future adaptation steps, however, the new unit can develop different weights and contribute to better error reduction in this area of the input vector space. The complete algorithm for supervised growing cell structures is shown in Figure 18.

3.4. Simulation Examples

Example 1: A Simple Classification Problem. We used the described supervised version of the growing cell structures to construct a classifier for the data shown in Figure 16. The network was chosen to be two-dimensional. Because the data had to be classified into two classes, two output units were used. The combined growth and learning process was continued until the MSE for the training data fell below some bound. The resulting network (Figure 19a) was used to map 200 × 200 points inside the square region to either one or the other class (Figure 19b).

One can observe that the size of the triangles and, therefore, the standard deviation of the Gaussians is considerably smaller in the region with the four small clusters (upper right). The reason is that the classification in this area is difficult and, therefore, many classification errors occur during training. This leads to insertions in this area. The resulting decision regions demonstrate that the final network classifies all training vectors correctly. Moreover, it seems to do a rather good job on classifying the other points inside the depicted region.

Example 2: The Two Spirals. A well-known benchmark in the connectionist community is the so-called two-spiral problem. It consists of 194 two-dimensional vec-

[6] This particular behavior is also a characteristic of the original perceptron learning rule introduced by Rosenblatt (1958).

```
Initialize cell structure $A$ with one $k$-dimensional simplex at random positions in $V = R^n$.
Create $m$ linear output units.
Create a weighted connection $w_{ic}$ from each cell $c \in A$ to each output unit $i, (i \in \{1, \dots, m\})$
Associate every cell (vertex of the simplex) with a Gaussian function.
while (classification error not low enough)
    repeat $\lambda$ times
        Choose I/O-pair $(\xi, \zeta) \in (R^n \times R^m)$ from training data
        Determine best-matching unit $s$ for $\xi$.
        Increase matching for $s$ and its direct neighbors.
        Compute activation $D_c$ for every cell $c \in A$ (see eqn. 24).
        Compute the vector $o = (o_1, \dots, o_m)$ of all output unit activations (see eqn. 25).
        Perform one delta-rule learning step for the weights (see eqn. 26)
        Increase resource variable of $s$ through    $\Delta\tau_s = ||\zeta - o||^2$
    Determine cell $q$ with maximum resource value
    Insert a new cell $r$ between $q$ and the direct neighbor $f$ with maximum distance in input vector space
    Redistribute resource values and weight vectors among $r$ and its direct neighbors according to eqn. 10 - 11 and 29 - 30, resp.
```

FIGURE 18. Supervised growing cell structures algorithm.

tors lying on two interlocked spirals that are the classes in this case (Figure 20a). The task is to construct a classifier able to distinguish between the two classes. This benchmark is interesting because, due to the low data dimensionality, it is possible to visualize the decision regions of the network during and after training. Moreover, it seems to be a rather difficult task for typical feedforward networks (e.g., multilayer perceptrons with sigmoidal activation functions). Lang and Witbrock (1989) were unable to solve the problem with a standard multilayer network and had to use additional connections to achieve convergence. Fahlman and Lebiere (1990) used a constructive algorithm called Cascade-Correlation to solve the problem. The resulting decision regions of this network are shown in Figure 20b. One can note that the Cascade-Correlation algorithm is able to learn the training data, but the decision regions show several artifacts. In many cases, points between two training vectors of a specific class are classified as belonging to the other class. This occurs especially in the outer parts of the spirals where the example patterns of one class are further apart from each other than from the representants of the other class. The resulting "cuts" in the spiral can be interpreted as poor generalization. In the absence of other evidence, it seems more natural to assume that those intermediate points belong to the same class. The decision regions produced by the network of Lang and Witbrock look similar.

Baum and Lang (1991) proposed a constructive method and also tested it with the two-spiral problem. Their approach employs an "oracle" that can tell for every point in the plane the desired class. Queries to the oracle are then used to position the hyperplanes corresponding to certain hidden units. An explicit test set of 576 points has been defined consisting of three points between each pair of adjacent same-class training points. Therefore, training and test points together form two spirals with a four times higher point density then the training set alone. For their best model Baum and Lang report an average of 29 errors on the test set.

We generated a two-dimensional growing cell structure to solve the two-spiral problem. The network and the corresponding decision regions are shown in Figure 21. In this case the decision regions form two well-separated spirals with very smooth borders. In fact, the decision regions exhibit a strong similarity to the oracle defined by Baum and Lang. Data points in between training vectors of one class are mapped onto that class and, therefore, the network makes no errors at all on the mentioned test set of Baum and Lang. Even in the

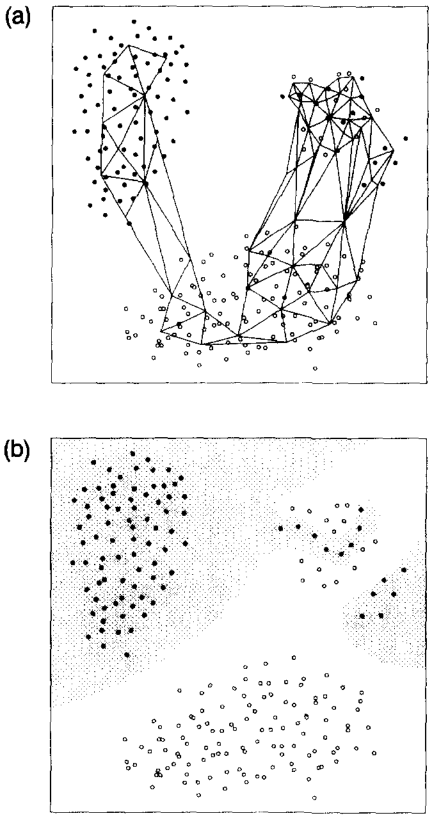

FIGURE 19. Supervised growing cell structures. Network and decision regions for the data shown in Figure 16. Simulation parameters: $\lambda = 240$, $\varepsilon_b = 0.1$, $\varepsilon_n = 0.006$, $k = 2$, $\tilde{t} = 2$, $\alpha = 0.005$, $\eta = 0.15$; no removal of cells. (a) Final network, (b) decision regions.

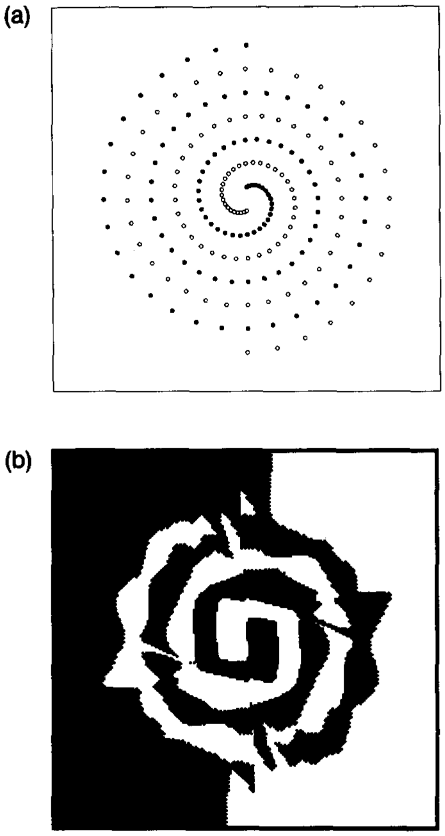

FIGURE 20. Two-spiral problem and learning results of a constructive network. (a) Two-spiral problem, (b) decision regions for Cascade-Correlation (reprinted with permission from Fahlman & Lebiere, 1990).

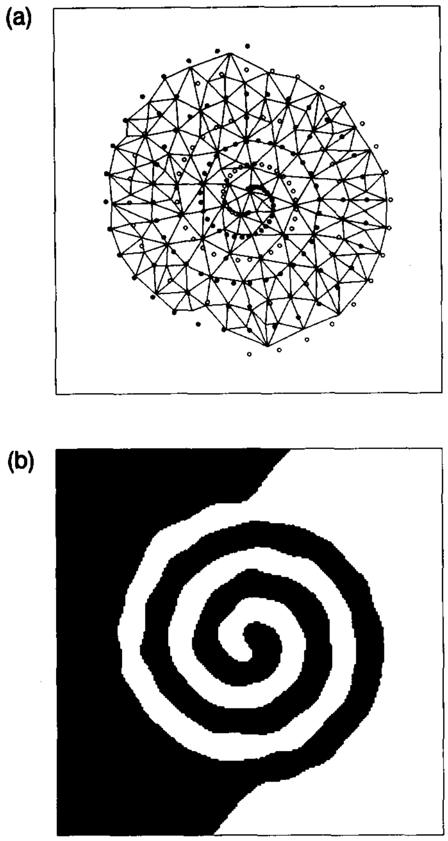

FIGURE 21. Performance of the growing cell structures on the two-spiral benchmark. Simulation parameters: $\lambda = 240$, $\varepsilon_b = 0.1$, $\varepsilon_n = 0.006$, $k = 2$, $\tilde{l} = 2$, $\alpha = 0.005$, $\eta = 0.15$; no removal of cells. (a) Final network with 145 cells, (b) decision regions.

outer regions of the spiral the decision regions follow the example vectors accurately. The local density of cells is rather uniform and does not follow the density of the training vectors, which is higher near the center of the spirals. This is not surprising because near the center fewer units *per training point* are needed to facilitate correct classification.

For every learning method an important practical aspect is the number of pattern presentations necessary to achieve a satisfying performance. In the case of a finite training set, a common measure is the number of cycles through all training patterns, also called *epochs*. We list in Table 2 the number of epochs for the two-spiral problem for some earlier methods and for our approach. As can be seen, the number of epochs required by the new method is about two orders of magnitude smaller than for standard back propagation and nearly one order of magnitude smaller than for Cascade-Correlation.

Example 3*: Speaker Independent Vowel Recognition.* To explicitly investigate the generalization capability of our model, we performed experiments with a vowel recognition problem. The data used was collected by Deterding (1989), who recorded examples of the 11 steady-state vowels of English spoken by 15 speakers for a speaker normalization study. The vowel data (as well as the two-spiral data) is electronically available from the Carnegie-Mellon University connectionist benchmark collection (see Fahlman, 1993).

An ASCII approximation to the International Phonetic Association (I.P.A.) symbol and the word in which the 11 vowel sounds were recorded is given in Table 3. The word was uttered once by each of the 15 speakers, seven of whom were female and eight male.

The speech signals were low pass filtered at 4.7 kHz and then digitized to 12 bits with a 10-kHz sampling

TABLE 2
Training Epochs Necessary for the Two-Spiral Problem

Network Model	Number of Epochs	Reported In
Back propagation	20000	Lang & Witbrock (1989)
Cross entropy BP	10000	Lang & Witbrock (1989)
Cascade-correlation	1700	Fahlman & Lebiere (1990)
Growing cell structures	180	this paper

TABLE 3
Words Used in Recording the Vowels (adapted from Robinson, 1989)

Vowel	Word	Vowel	Word
i:	Heed	O	Hod
I	Hid	C:	Hoard
E	Head	U	Hood
A	Had	u:	Who'd
a:	Hard	3:	Heard
Y	Hud		

rate. Twelfth-order linear predictive analysis was carried out on six 512 sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, giving a 10-dimensional input space. A general introduction to speech processing and an explanation of this technique can be found in e.g., Rabiner and Schafer (1978). Each speaker thus yielded six frames of speech from 11 vowels. This gave 990 frames from the 15 speakers.

Robinson used this data in his thesis (Robinson, 1989) to investigate several types of neural network algorithms. He used 528 frames from four male and four female speakers to train the networks and used the remaining 462 frames from four male and three female speakers for testing the performance.

The classifiers he examined were single-layer perceptrons, multilayer networks with sigmoidal, Gaussian, and quadratic activation functions, a modified Kanerva model, radial basis networks, and also a conventional method, the nearest-neighbor classifier. Due to the limited computational facilities available to Robinson, he did only one run for each of the different architectures. Every run was continued for about 3000 epochs (Robinson, 1993).

To get comparable results, we trained several growing cell structure networks with the same data as Robinson and thereafter used his test data to evaluate the generalization capabilities of the networks. Because the input vector dimension was high-dimensional (10), we used also networks of a somewhat higher dimension than in the previous examples.

The results of Robinson and our results are shown in Table 4. For easier comparison the percentage of correctly classified test patterns is shown graphically in Figure 22. It is evident from the simulations that our approach has the best results of the considered methods. The networks had to be trained only for about 80 epochs, which compares rather well to the other methods. The ratio 3000/80 = 37.5 is also approximately along the lines of our simulations for the two-spiral problem if one compares the number of epochs needed for cross entropy back propagation and for our model (see Table 2).

4. RELATED WORK

In the following we draw some connections to existing models. The list of examples given is not meant to be comprehensive. Rather, we tried to pick out a few models that are so closely related to our work that a comparison is instructive.

4.1. Unsupervised Learning

A number of variations of Kohonen's feature map have been proposed concerning networks with variable topology or variable number of elements. The approach of Jokusch (1990) starts with a usual rectangular grid. The net is incrementally extended by pushing rows or columns of units by one and inserting new units in the created gaps. After a while this can lead to rather complicated structures.

Kangas, Kohonen, and Laaksonen (1990) propose a model where the connectivity is updated periodically to form a minimum spanning tree. An advantage of this approach is that there can be no topological defects like *twisted maps,* which sometimes do occur within Kohonen's original formalism. However, the topology of the input patterns is only captured to a small degree due to the sparse connectivity of this model. Moreover, for high-dimensional data it is hard to visualize the network in a meaningful way.

A very interesting method is the Neural Gas algorithm of Martinetz and Schulten (1991). It starts with no connections and a fixed number of units floating in input vector space like gas particles. Input signals are used to adapt the units and to add connections, which are inserted between the winning unit for a signal and the second winning unit. Moreover, a mechanism for aging and removal of connections is provided. Recently, Martinetz (1993) gave a rigorous definition of *topology preserving feature map* and he could show that his method generates a connectivity structure that is perfectly topology preserving in this sense.

The networks generated by the method of Martinez do look similar in some cases to the networks generated by the growing cell structures. However, there are at least two important differences between both methods. First, neural gas uses a fixed number of units that has to be chosen in advance (in contrast to our model). Finding a suitable number in advance is a very difficult problem. The second difference concerns the dimensionality of the resulting network. The dimensionality of the neural gas networks depends on the local properties of the data. If the data is three-dimensional then so is the network. If part of the data lies in a two-dimensional subspace then in this region a two-dimensional structure is formed. The growing cell structures, on the other hand, have a fixed network dimensionality (e.g., two or three), which is valid throughout the whole network and is chosen at the beginning. This leads, of

TABLE 4
Test Results on Vowel Recognition Problem

Classifier	Number of Hidden Units	Correctly Classified	Percent Correct
Single-layer perceptron	—	154	33
Multilayer perceptron	88	234	51
Multilayer perceptron	22	206	45
Multilayer perceptron	11	203	44
Modified Kanerva Model	528	231	50
Modified Kanerva Model	88	197	43
Radial basis function	528	247	53
Radial basis function	88	220	48
Gaussian node network	528	252	55
Gaussian node network	88	247	53
Gaussian node network	22	250	54
Gaussian node network	11	211	47
Square node network	88	253	55
Square node network	22	236	51
Square node network	11	217	50
Nearest neighbor	—	260	56
Three-dimensional GCS	154	309	67
Three-dimensional GCS	165	285	62
Three-dimensional GCS	158	282	61
Five-dimensional GCS	135	306	66
Five-dimensional GCS	196	307	66

The table shows the network size, the number of correctly classified test patterns (out of 462), and the corresponding percentages. The single layer perceptron shows the results reported by Robinson in his thesis (Robinson, 1989). He got the best classification rate for the nearest neighbor method. The 3-D and 5-D GCS shows the result of several nets generated by the growing cell structures method. All of them have a higher rate of correctly classified test patterns than the nearest neighbor method (and all the other models examined by Robinson). We tried networks of dimensionality three and five. The parameter $\tilde{t}$ was set equal to the network dimension in each case. The second run with a five-dimensional network was continued very long to see whether over-training effects could be produced, which was not the case in that simulation. Also, the different choices for $\tilde{t}$ did not seem to influence the outcome of the algorithm very much.

course, to a topology preservation that might be far from being optimal in some cases. An advantage of a fixed dimensionality, however, is that the network can be visualized if it is low-dimensional, no matter how high-dimensional the input data is (see Figure 13). Thus, high-dimensional data can be visualized in (sometimes) interesting ways with our model.

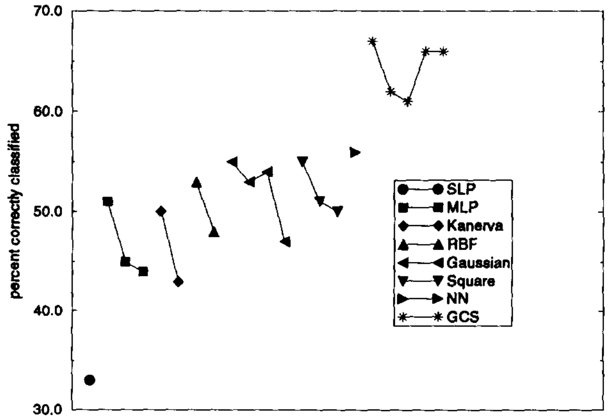

FIGURE 22. Percentage of correctly classified test patterns for the vowel recognition problem.

Other models allow a variable number of elements, but have predefined principal structure (e.g., rectangular array), namely the interpolative algorithm of Rodrigues and Almeida (1990). For this approach a considerable speed-up compared with the original formalism is reported. However, the problem of having to choose the shape of the network in advance, persists.

The learning expectation method introduced by Xu (1990) starts with one rectangular grid and adds, one-by-one, more identical rectangular grids if the data cannot be captured by the existing ones. This leads to a covering of the relevant parts of the input space with a number of identical rectangular networks. Depending on the input data, this might be very inappropriate, because there is no good way to model small and large clusters at the same time with this method.

A proposal to use random structures stems from Ritter (1991). He shows an example where a random structure is used in a quite appealing way to represent related and more distant concepts. However, in general it is hard to predict how good a random structure will be for a given problem and probably still harder to arrange the structure such that a suitable modeling of the data is indeed achieved.

An interesting approach with a network growing on a grid has been introduced by Blackmore and Miikkulainen (1992). A rectangular neighborhood is used, as in Kohonen's model, but single connections are allowed to be missing to better model the shape of the underlying data distribution. Because the network is restricted to a grid it can be easily visualized even for high-dimensional data vectors.

4.2. Supervised Learning

There exists some relevant related work also for the supervised variant of the growing cell structures. In the following we mention some methods that are based on networks of localized units and that allow a change in the number of free parameters of the network.

The work probably most similar to our model stems from Platt (1991). He proposes a resource-allocating network for function estimation. His network consists of radial basis units and he adds a new unit whenever a data item is sufficiently far from all existing centers *and* is mapped sufficiently poorly. In this case a new center is allocated and is set up such that the current data item is mapped correctly. The main difference between Platt's approach and the growing cell structures is that Platt inserts a new unit on the basis of *one* poorly mapped pattern whereas in the growing cell structures error information is always accumulated over a number of patterns before a new unit is inserted. The advantage of the latter strategy is that noisy data does not lead to a huge number of inserted units, which can easily happen with Platt's model. Moreover, in Platt's model there is a resolution parameter that specifies the smallest distance two radial basis function units can have and that effectively makes it impossible to model functions appropriately that vary on a smaller scale.

Hakala and Eckmiller (1993) propose a method that is very similar to Platt's approach. They also allocate nodes on the basis of the observed error for single examples. In contrast to Platt, however, who does not change the parameters of existing units, Hakala and Eckmiller reduce the diameter of neighboring units when a new unit is inserted. This leads to a controlled overlap between adjacent units. This again makes the learning task easier for the delta rule, easier because there are less conflicts arising from units that overlap to a large extent. A similar mechanism is realized in the growing cell structures: here the diameter of each local unit is determined by the mean distance to its topological neighbors. Therefore, the insertion of a new unit reduces the diameters or regions of influence of its neighbors.

Other related work has been done by Bonnlander and Mozer (1993). They arrange localized units on a rectangular grid. They do not vary the number of units but have found another way to change the number of free parameters of the network. At the beginning very strong constraints are imposed on the units, allowing neighboring units to have only slightly varying parameters. During the training these restrictions are slowly loosened up so that the number of free parameters is increased. However, there is an upper limit for the number of free parameters that depends on the initially chosen (and thereafter unchanged) size of the network.

5. DISCUSSION

In the first part of the paper we introduced a new self-organizing network model. It has the following advantages over existing models:

- The network structure is determined automatically from the input data.
- There is no need to choose the network size in advance. Instead, the growth process can be continued until a performance criterion is met.
- All parameters of the model are constant. Therefore, it is not necessary to define a decay schedule as in other models.
- The insertion of new units can be influenced such that the generated network estimates the probability density of the input signals, minimizes the quantization error, or pursues still other goals.
- Because the final structure depends on the input data, it can be used for data visualization and for clustering. In contrast, most other models have a fixed structure that does not provide any information of that kind.

In the second part of the paper we developed a combination of the self-organizing network with the radial basis function (RBF) approach. It provides a number of improvements over current network models with localized receptive fields (and also some other models):

- Number, diameter, and position of RBF units are determined automatically through a growth process that can be stopped as soon as the network performs well enough.
- Because positioning of RBF units and supervised training of connection weights is performed in parallel, the current classification error can be used to determine the locations of new RBF units. Previous approaches did often rely on clustering algorithms, which in general fail to find good positions for the RBF units with respect to classification accuracy.
- The networks are relatively small and generalize very well.
- The necessary number of training epochs seems to be one to two orders of magnitude smaller than for other approaches.

Although the results obtained to date are very promising, it is necessary to investigate the performance of the network for larger problems than the ones presented here. Furthermore, it would be an improvement if one could find ways to automatically choose some of those parameters that still have to be set by the user. An interesting goal would be a model with no parameters except the properties of the desired classifier. This goal is, of course, still very distant but the proposed methods might be a step in the right direction.

REFERENCES

Baum, E. B. & Lang, K. E. (1991). Constructing hidden units using examples and queries. In R. P. Lippmann, J. E. Moody, & D. S. Touretzky (eds.), *Advances in neural information processing systems 3* (pp. 904-910). San Mateo, CA: Morgan Kaufmann Publishers.

Blackmore, J. & Miikkulainen, R. (1992). Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. University of Texas at Austin, TR AI92-192, Austin, TX.

Bonnlander, B. V. & Mozer, M. C. (1993). Metamorphosis networks: An alternative to constructive models. In L. Giles, S. Hanson, & J. Cowan (eds.), *Advances in neural information processing systems 5* (pp. 131-138). San Mateo, CA: Morgan Kaufmann Publishers.

Deterding, D. H. (1989). *Speaker normalisation for automatic speech recognition,* Ph.D. thesis, University of Cambridge.

Fahlman, S. E. (1993). *CMU benchmark collection for neural net learning algorithms.* Carnegie Mellon University, School of Computer Science [machine-readable data repository], Pittsburgh.

Fahlman, S. E. & Lebiere, C. (1990). The Cascade-Correlation learning architecture. in D. S. Touretzky (ed.), *Advances in neural information processing systems 2* (pp. 524–532). San Mateo, CA: Morgan Kaufmann Publishers.

Favata, F. & Walker, R. (1991). A study of the application of Kohonen-type neural networks to the travelling salesman problem. *Biological Cybernetics,* **64,** 463–468.

Fritzke, B. (1993a). Kohonen feature maps and growing cell structures—a performance comparison. In L. Giles, S. Hanson, & J. Cowan (eds.), *Advances in neural information processing systems* 5 (pp. 123-130). San Mateo, CA: Morgan Kaufmann Publishers.

Fritzke, B. (1993b). Vector quantization with a growing and splitting elastic net. *ICANN'93: International Conference on Artificial Neural Networks,* Amsterdam, pp. 580–585.

Hakala, J. & Eckmiller, R. (1993). Node allocation and topographic encoding NATEnet for inverse kinematics of *ICANN'93: International Conference on Artificial Neural Networks,* Amsterdam, pp. 309-312.

Jokusch, S. (1990). A neural network which adapts its structure to a given set of patterns. In R. Eckmiller, G. Hartmann, & G. Hauske (eds.), *Parallel processing in neural systems and computers* (pp. 169-172) Amsterdam: Elsevier Science Publishers B.V.

Kangas, J. A., Kohonen, T., & Laaksonen, T. (1990). Variants of self-organizing maps. *IEEE Transactions on Neural Networks,* **1,** 93–99.

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

Kohonen, T. (1988). The neural phonetic typewriter. *IEEE Computer,* **21,** 11-22.

Kohonen, T., Mäkisara, K., & Saramäki, T. (1984). Phonotopic maps—insightful representation of phonological features for speech recognition. *Proceeding 7th International Conference on Pattern Recognition,* Montreal, pp. 182-185.

Lang, K. J. & Witbrock, M. J. (1989). Learning to tell two spirals apart. in D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School* (pp. 52-59). San Mateo, CA: Morgan Kaufmann.

Martinetz, T. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps. *ICANN'93: International Conference on Artificial Neural Networks,* Amsterdam, pp. 427–434.

Martinetz, T. M. & Schulten, K. J. (1991). A "neural-gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, & J. Kangas (eds.), *Artificial neural networks* (pp. 397–402). Amsterdam: North-Holland.

Mehlhorn, K. & Näher, S. (1989). LEDA, a library of efficient data types and algorithms. Universität des Saarlandes, Fachbereich Informatik, TR A 04/89, Saarbrücken.

Moody, J. & Darken, C. (1989). Learning with localized receptive fields. In D. Touretzky, G. Hinton, & T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School* (pp. 133-143). San Mateo, CA: Morgan Kaufmann.

Platt, J. C. (1991). A resource-allocating network for function interpolation. *Neural Computation,* **3,** 213-225.

Poggio, T. & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks, *Science,* **247,** 978-982.

Rabiner, L. R. & Schafer, R. W. (1978). *Digital processing of speech signals.* Englewood Cliffs, NJ: Prentice Hall.

Ritter, H. J. (1991). Learning with the self-organizing map, In T. Kohonen, K. Mäkisara, O. Simula & J. Kangas (eds.), *Artificial neural networks* (pp. 379-384). Amsterdam: North-Holland.

Ritter, H. J. & Kohonen, T. (1989). Self-organizing semantic maps. *Biological Cybernetics,* **61,** 241-254.

Robinson, A. J. (1989). *Dynamic error propagation networks.* PhD thesis, Cambridge University.

Robinson, A. J. (1993). (personal communication).

Rodrigues, J. S. & Almeida, L. B. (1990). Improving the learning speed in topological maps of patterns. *Proceedings of INNC,* Paris, pp. 813-816.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review,* **65,** 386–408.

Schweizer, L., Parladori, G., Sicuranza, G. L. & Marsi, S. (1991). A fully neural approach to image compression. In T. Kohonen, K. Mäkisara, O. Simula, & J. Kangas, (eds.), *Artificial neural networks* (pp. 815–820). Amsterdam: North-Holland.

Willshaw, D. J. & von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London B,* **194,** 431-445.

Xu, L. (1990). Adding learning expectation into the learning procedure of self-organizing maps. *International Journal of Neural Systems,* **1,** 269-283.

NOMENCLATURE

A	Growing cell structures network, also denotes set of cells in the network
k	dimensionality of the growing cell structures network A
V	n-dimensional input vector space
n	dimensionality of V
w_c	n-dimensional reference (synaptic, weight) vector of cell c
w	set of all reference vectors for cells in A
ϕ_w	mapping $V \rightarrow A$
λ	adaptation steps per insertion
ε_b	adaptation parameter for best-matching unit
ε_n	adaptation parameter for neighboring cells
η	threshold for cell removal
$P(\xi)$	probability distribution of input signals
α	decrease parameter for resource variables
N_c	set of direct neighbors of a cell c
F_c	Voronoi field of cell c
t	*true* data dimensionality

$\tilde{t}$	estimate for t
$\vert F_c \vert$	n-dimensional volume of F_c
ξ	n-dimensional input signal
ζ	m-dimensional output signal
m	dimension of the output vector space for supervised learning
(ξ, ζ)	I/O-pair (for supervised learning)
o	m-dimensional vector of output unit activations
τ_c	resource variable of cell c (can contains, e.g., signals, quantization error, classification error)
L	layer of Gaussian units in RBF networks
$D_c(\xi)$	activation of Gaussian unit c
h_c	relative signal frequency of cell c
$\tilde{p}_c$	estimate of the probability density near w_c
$\hat{p}_c$	estimate of the normalized probability density near w_c
$\Delta x = y$	short-cut for $x^{\text{new}} = x^{\text{old}} + y$
w_c^{out}	m-dimensional vector of weights from cell c to the output units
w_{ic}	weighted connection from Gaussian unit c to output unit i
$\Vert \cdot \Vert$	Euclidean vector norm