Dynamic Cell Structure Learns Perfectly Topology Preserving Map

Jörg Bruske
Gerald Sommer
Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität Kiel, Preusserstraße 1-9,
D-24105 Kiel, Germany

Dynamic cell structures (DCS) represent a family of artificial neural architectures suited both for _unsupervised_ and _supervised_ learning. They belong to the recently (Martinetz 1994) introduced class of _topology representing networks_ (TRN) that build _perfectly topology preserving feature maps_. DCS employ a modified _Kohonen learning rule_ in conjunction with _competitive Hebbian learning_. The Kohonen type learning rule serves to adjust the synaptic weight vectors while Hebbian learning establishes a dynamic _lateral connection structure_ between the units reflecting the topology of the feature manifold. In case of supervised learning, i.e., function approximation, each neural unit implements a _radial basis function_, and an additional layer of linear output units adjusts according to a _delta-rule_. DCS is the first RBF-based approximation scheme attempting to concurrently learn and utilize a perfectly topology preserving map for improved performance. Simulations on a selection of CMU-Benchmarks indicate that the DCS idea applied to the _growing cell structure_ algorithm (Fritzke 1993c) leads to an efficient and elegant algorithm that can beat conventional models on similar tasks.

1 Introduction

Kohonen's self-organizing feature maps (SOM) (Kohonen 1987), besides backpropagation networks, are now the most popular and successful types of artificial neural networks (ANN). This is impressively demonstrated by a constantly growing list of references to SOM-related research and applications available from Helsinki University of Technology containing about 500 entries.[1]

SOMs are used for adaptive vector quantization, clustering, and dimensionality reduction, and can be extended to associative memories

[1]Via anonymous FTP from cochlea.hut.fi (130.233.168.48).

like sensorimotor maps simply by adding an output to each neural unit (Ritter *et al.* 1991). Their main features are

- formation of "topology preserving" feature maps, i.e., mapping similar input signals to neighbored neural units (and vice versa) and
- approximation of the input probability distribution, i.e., the number of neural units responding to a certain subset of the input space is proportional to the probability of a stimulus to come from this subspace.

However, it has long been noticed that Kohonen maps have several drawbacks when used for tasks different from visualization of high dimensional data. Mainly these are

1. a fixed number of neural units, making them impractical for applications where the optimal number of units is not known in advance (but only, say, some accuracy parameters),
2. a topology of fixed dimensionality, resulting in problems if this dimensionality does not match the dimensionality of the feature manifold (in this case one cannot claim topology preservation),
3. that classes/clusters have to be separated by hand, whereas an automatic separation is clearly desirable, and
4. unoptimal behavior if, as in the case of sensorimotor maps, one is interested in optimizing the output and not so much in approximating the input density (there may be less interesting regions of high input density but important regions of low input density).

All these problems are topics of ongoing vivid research. In particular, Fritzke's growing cell structures (GCS) (Fritzke 1992, 1993), represent a computationally inexpensive neural algorithm with a variable number of neural units that elegantly combines the merits of RBF networks with an SOM-like topology preserving neighborhood relation between units. A local error measure serves to allocate new units that, since the neighborhood relation between units is known, can be placed in between neighbored units. However, although the topology of GCS is much more flexible than that of the Kohonen map, the problem of fixed topology dimension remains. Further, GCS cause problems when cells are to be deleted.[2]

There have been numerous attempts (e.g., Sebestyen 1962; Hart 1968; Reilly *et al.* 1982; Specht 1990) to realize variable sized clustering and RBF networks (Platt 1991a; Hakala and Eckmiller 1993). In the latter two, units are inserted depending on the overall performance of the net, the center of their receptive field and output being set to the current training input and output. Unlike Kohonen's feature maps these typical RBF networks do not utilize a neighborhood relation between units, nor do

[2]This is due to the lateral connection structure between cells, which has to form k-dimensional simplices.

they learn one. We also point out the close relation of all these algorithms to techniques of case based reasoning in symbolic AI and techniques for fuzzy rule generation.

The missing attention to problem (2) turned out to be a problem of missing definition: There has been no rigorous definition of "topology preserving feature mapping" up to the article of Martinetz (1993). A rather intuitive and imprecise notion of "topology preservation" prevailed. Having established this definition Martinetz was able to show that a very simple competitive Hebbian learning rule can learn perfectly topology preserving feature maps if the neural units are "dense." Martinetz (1992) demonstrates that his *neural gas* algorithm enriched by his competitive Hebbian learning rule has the potential to learn perfectly topology preserving mappings. However, the neural gas does not further exploit this information and continues to recompute the k nearest-neighbors of the best matching unit on every presentation of a new stimulus. The very recommendable contribution (Martinetz and Schulten 1994) summarizes these ideas and outlines the relevance of perfect topology preservation for practical applications.

The authors' DCS appear as a natural consequence: Combining the merits of locally tuned processing units with Martinetz's idea of perfectly topology learning we obtain SOM like ANNs that concurrently attempt to learn and utilize a perfectly topology preserving map for an improved training and approximation performance of RBF networks. DCS promise to solve the problem of perfect topology preservation and support automatic cluster separation. Compared to Martinetz's neural gas algorithm, DCS avoid computational burdens by utilizing the lateral connection structure (topology) learned so far.

The particular instance of a DCS algorithm presented in this paper is the *DCS-GCS* algorithm, which rests on Fritzke's GCS. We have chosen GCS because of its increasing popularity and because of encouraging results on a selection of CMU Benchmarks (Fritzke 1993c) to which DCS-GCS can be readily compared. This comparison indicates that DCS-GCS compares well to GCS while beating most conventional algorithms on similar tasks. Unlike GCS, however, DCS-GCS does not use a priori information about the topology of the feature manifold but learns a perfectly topology preserving map and is easier to implement. Similar simulations by Fritzke have confirmed our results (Fritzke, personal communications on ICANN'94).

2 Foundations

In this section we want to recapitulate the definitions and theorems of Martinetz concerning the formation of perfectly topology preserving maps and outline the most important features of Fritzke's GCS. We also indicate how their work is extended and synthesized to GCS-DCS.

2.1 Perfectly Topology Preserving Maps. In the following, let

- G be a graph (network) with vertices (neural units) i, $1 \leq i \leq N$ and edges (lateral connections) between them weighted by C_{ij}, its adjacency matrix[3] (weight matrix).
- $M \subseteq \Re^D$ be a given manifold of features $v \in M$,
- $S = \{w_1, \dots, w_N\}$ be a set of pointers (synaptic weight vectors) $w_i \in M$, each of which is attached to a vertex i of G,
- $V_i = \{v \in \Re^D \mid (\|v - w_i\| \leq \|v - w_j\|, 1 \leq j \leq N)\}$ the Voronoi polyhedron belonging to $w_i \in M$, and
- $V_i^{(m)} = V_i \cap M$ the masked Voronoi polyhedron of V_i, $1 \leq i \leq N$.

Definition 1. Two points w_i, $w_j \in S$ are adjacent on M if their masked Voronoi polyhedra $V_i^{(M)}, V_j^{(M)}$ are adjacent (have some boundary points in common), i.e., $V_i^{(M)} \cap V_j^{(M)} \neq \emptyset$.

Definition 2. The graph G forms a perfectly topology preserving map of M, if pointers w_i, w_j, which are adjacent on M, belong to vertices i, j which are adjacent in G ($C_{ij} \neq 0$), and vice versa.

Definition 3. The induced Delaunay triangulation $D_S^{(M)}$ of S, given M, is defined by the graph, which connects two points w_i, w_j iff their masked Voronoi polyhedra $V_i^{(M)}, V_j^{(M)}$ are adjacent, i.e., $(C_{ij} \neq 0) \Leftrightarrow (V_i^{(M)} \cap V_j^{(M)} \neq \emptyset)$.

Definition 4. The set $S = \{w_1, \dots, w_N\}$ is dense on M if for each $v \in M$ the triangle $\Delta(v, w_{i_0}, w_{i_1})$ formed by the point w_{i_0}, which is closest to v, the point w_{i_1}, which is second closest to v, and v itself lie completely on M, i.e., $\Delta(v, w_{i_0}, w_{i_1}) \subseteq M$ is valid.

We are now able to quote Martinetz's central theorem

Theorem 1 (Martinetz 1993). *If the distribution of pointers $w_i \in S$ is dense on M then the edges (lateral connections) i–j formed by the competitive Hebb rule*

$$\Delta C_{ij} = \begin{cases} y_i \cdot y_j; & y_i \cdot y_j \geq y_k \cdot y_l \; \forall (1 \leq k, l \leq N) \\ 0; & \text{otherwise} \end{cases} \tag{2.1}$$

define a graph (network) G that corresponds to the induced Delaunay triangulation $D_S^{(M)}$ of S and, hence, forms a perfectly topology preserving map of M.

Here, $y_i = R(\|v - w_i\|)$ is the activation of the ith unit with w_i as the center of its receptive field on presentation of stimulus v. The mapping $R(\cdot)$: $\Re \rightarrow [0, 1]$ must be a continuously monotonically decreasing function. Martinetz (1994) coins the term topology representing network (TRN) for networks that use equation 2.1 for topology learning.

[3]The adjacency matrix A of a graph G normally is defined by $a_{ij} = 1$ if node i is connected with node j, and $a_{ij} = 0$ otherwise. However, our adjacency matrix C is defined by $0 < c_{ij} \leq 1$ if node i is connected with node j, and $c_{ij} = 0$ otherwise. The c_{ij} may be interpreted as the certainty that i is connected with j.

Of course, when dealing with realistic data from an unknown feature manifold M, we cannot decide whether a given (learned) set $S \subseteq M$ is dense. Instead, only a (possibly small) set of training data $T \subseteq M$ is available, from which a set of points $S(T)$ has to be constructed such that $D_S^{(M)} = D_{S(T)}^{(M)}$ for some dense set $S \subseteq M$. Moreover, we are often interested in smallest dense sets $S \subseteq M$ because these result in the highest data reduction. A third problem arises if the pattern distribution $P(v)$ is not stationary. In this case the neighborhood relation may change with time and thus lateral connections may have to be removed (forgotten): The same problem appears with dynamic data sets S and T.

A straightforward solution to the last problem is to introduce a forgetting constant α, $0 < \alpha < 1$, such that $C_{ij}(t+1) = \alpha C_{ij}(t)$. In DCS we started experiments with the competitive Hebbian learning rule

$$C_{ij}(t+1) = \begin{cases} 1; & y_i \cdot y_j \geq y_k \cdot y_l \ \forall (1 \leq k, l \leq N) \\ 0; & C_{ij}(t) < \theta \\ \alpha C_{ij}(t); & \text{otherwise} \end{cases} \tag{2.2}$$

where θ, $0 < \theta < 1$, serves as a threshold for deleting lateral connections.

For off-line learning with a training set T of fixed size $|T|$,

$$\alpha = \sqrt[|T|]{\theta} \tag{2.3}$$

is likely to be a good choice because once $S(T)$ is dense on M one further epoch of training will yield the induced Delaunay triangulation $D_{S(T)}^{(M)}$. For on-line learning the optimal choice of α will depend on $P(v)$ and the error distribution $P(\Delta v)$ (which most often are unknown). We also conducted experiments with an alternative to equation 2.2, where

$$C_{ij}(t+1) = \left\{ \max\{y_i \cdot y_j, C_{ij}(t)\}; \quad y_i \cdot y_j \geq y_k \cdot y_l \ \forall (1 \leq k, l \leq N) \right. \tag{2.4}$$

Equation 2.4 offers the advantage that the induced connection strength between two units peaks for stimuli lying exactly in between these units. It can be expected to be less sensitive to noise and to perform better if S is not dense. Indeed, best results on the tested Benchmarks have been obtained using equation 2.4.

Martinetz (1992) points out that for reasons of efficiency instead of decreasing all connections one can decrease the connections to the best matching unit only, and that these methods are equivalent if each unit has equal probability of being the best match. Moreover, this method offers an additional advantage for on-line learning situations where equation 2.2 or 2.4 may lead to a total decay of the connection structure in regions of the input space which have not been visited for a longer time.

2.2 Growing Cell Structures and Resources. In Fritzke's GCS, the network is initialized with a k-dimensional simplex of $N = k + 1$ neural units and $(k+1) \cdot k/2$ lateral connections (edges). Growing of the network is performed such that after insertion of a new unit the network consists solely of k-dimensional simplices again. Thus, like Kohonen's SOM, GCS can learn a perfectly topology preserving map only if k meets the actual dimension of the feature manifold. Assuming that the lateral connections do reflect the adjacency of units the connections serve to define a neighborhood for a Kohonen-like adaptation of the synaptic vectors w_i and guide the insertion of new units. Insertion happens incrementally and does not necessitate a retraining of the network. The principle is to insert new neurons in such a way that the expected value of a certain local error measure, which Fritzke calls the resource, becomes equal for all neurons. For instance, the number of times a neuron wins the competition, the sum of distances to stimuli for which the neuron wins or the sum of errors in the neuron's output can all serve as a resource and dramatically change the behavior of GCS. Using different error measures and guiding insertion by the lateral connections contributes much to the success of GCS. For more details about GCS the reader is referred to Fritzke (1993c).

DCS-GCS works much like GCS with one essential difference: The topology of the graph G (lateral connection scheme between the neural units) is not of a predefined and fixed dimensionality k but rather is learned on-line (during training) according to 2.4. This not only decreases overhead (Fritzke has to handle sophisticated data structures to maintain the k-dimensional simplex structure after insertion/deletion of units) but offers the possibility of learning real (perfectly) topology preserving feature mappings. Since the isomorphic representation of the topology of the feature manifold M in the lateral connection structure is central to performance, DCS-GCS can be expected to outperform GCS (if k is not constant over M or is not known in advance).

Note that if a DCS algorithm has actually learned a perfectly topology preserving mapping, cluster analysis becomes extremely simple: Clusters that are bounded by regions of $P(v) = 0$ can be identified simply by a connected component analysis. However, without prior knowledge about the feature manifold M it is, in principle, impossible to check for perfect topology preservation or the density of S. Noise in the input data may render perfect topology learning even more difficult. So what can perfect topology learning be used for? The answer is that for every set S of reference vectors perfect topology learning yields maximum topology preservation[4] with respect to this set. So in this sense the learned connection structure C is the best estimate for a topology preserving neighborhood relation if no a priori knowledge of the dimensionality k of M is available. Consequently, this is the case where it should be used for Kohonen-like adaptations of the reference vectors and interpolations between the outputs of neighbored units—the principle of DCS. Con-

[4]If topology preservation is measured by the topographic function as defined in Villmann *et al.* (1994).

nected components with respect to C may well serve as an initialization for postprocessing by hierarchical cluster algorithms.

Admittedly, if k is known in advance (a priori knowledge) then SOM-like algorithms that utilize k can be advantageous, especially if training data are sparse.

3 Unsupervised DCS-GCS

In this section we present our algorithm for unsupervised learning DCS-GCS. Simulations serve to illustrate the dynamics.

The unsupervised DCS-GCS algorithm can be obtained from Figure 3, the supervised version, by dropping procedures *calcOutput*(y, v, bmu, σ) and *deltaRule*(y, bmu, u, η) and neglecting the training outputs u. It starts with initializing the network (graph) to two neural units (vertices) $n1$ and $n2$. Their weight vectors w_1, w_1 (centers of receptive fields) are set to points $v_1, v_2 \in M$, which are drawn from M according to $P(v)$ in procedure *getNextExample(&v,* TRAIN). In procedure *enforceConnection*$(n1, n2, 1.0)$ they are connected by a lateral connection of weight $C_{12} = C_{21} = 1$. Note that lateral connections in DCS are always bidirectional and have symmetric weights.

Now the algorithm enters its outer loop, which is repeated until *stoppingCriterion*( ) is fulfilled. This stopping criterion could, for instance, be a test whether the quantization error has already dropped below a predefined accuracy.

The inner loop is repeated λ times. In off-line learning λ can be set to the number of examples in the training set T. In this case, the inner loop just represents an epoch of training.

Within the inner loop, the algorithm first draws an input stimulus $v \in M$ from M according to $P(v)$ by calling *getNextExample(&v,* TRAIN) and then proceeds to calculate the two neural units, which weight vectors are first and second closest to v (by *calcTwoClosest(&bmu, &second, v*)].

$$\begin{aligned} \|w_{bmu} - v\| &\leq \|w_i - v\|,\ (1 \leq i \leq N), \\ \|w_{second} - v\| &\leq \|w_i - v\|,\ (1 \leq i \neq bmu \leq N) \end{aligned} \tag{3.1}$$

In the next step, the lateral connections between the neural units are modified according to equation 2.4, a competitive Hebbian learning rule. It is implemented by the procedure *competitive Hebb*$(bmu, second, \alpha, \theta)$. As already mentioned, it is a good idea to set $\alpha = \sqrt[\lambda]{\theta}$ in off-line learning.

Procedure *restrictedKohonen*$(bmu, v, \varepsilon_B, \varepsilon_N)$ adjusts the weight vectors w_i of the best matching unit and its neighbors in a Kohonen-like fashion:

$$\begin{aligned} \Delta w_{bmu} &= \varepsilon_B (v - w_{bmu}) \qquad \text{and} \\ \Delta w_j &= \varepsilon_{Nh}(v - w_j), (k = bmu) \wedge [j \in Nh(bmu)] \end{aligned} \tag{3.2}$$

where the neighborhood $Nh(j)$ of a unit j is defined by

$$Nh(j) = \{i \mid (C_{ji} \neq 0, 1 \leq i \leq N)\} \tag{3.3}$$

The inner loop ends with updating the resource value of the best matching unit. The resource of a neuron is a local error measure attached to each neural unit. As pointed out in Section 2.2, one can choose alternative update functions corresponding to different error measures. For our experiments (Section 3.1) we used the accumulated squared distance to the stimulus, i.e., $\Delta\tau_{bmu} = \|v - w_{bmu}\|^2$.

The outer loop now proceeds by adding a new neural unit r to the network (*addNewNeuron*( )). This unit is located in between the unit l with largest resource value and its neighbor n with second largest resource value.[5]

$$\begin{aligned} \tau_l &\geq \tau_i, \qquad (1 \leq i \leq N) \qquad \text{and} \\ \tau_n &\geq \tau_i, \qquad [1 \leq i \neq l \leq N, n \in Nh(l)] \end{aligned} \tag{3.4}$$

The exact location of its center of receptive field w_r is calculated according to the ratio of the resource values τ_l, τ_n, and the resource values of units n and l are redistributed among r, n, and l:

$$\gamma = \tau_n/(\tau_n + \tau_l), \qquad \Delta\tau_l = \frac{1}{2}(1-\gamma)\tau_l \qquad \text{and } \Delta\tau_n = \frac{1}{2}\gamma\tau_n \tag{3.5}$$

$$\begin{aligned} w_r &= w_l + \gamma(w_n - w_l), \qquad \tau_r = \Delta\tau_n + \Delta\tau_l, \tau_l = \tau_l - \Delta\tau_l, \text{ and} \\ \tau_n &= \tau_n - \Delta\tau_n \end{aligned} \tag{3.6}$$

This gives an estimate of the resource values if the new unit had been in the network from the start. Finally the lateral connections are changed:

$$C_{rl} = C_{lr} = 1, \qquad C_{nr} = C_{rn} = 1 \text{ and } C_{nl} = C_{ln} = 0 \tag{3.7}$$

connecting unit r to unit l and disconnecting n and l.

This heuristic guided by the emerging lateral connection structure and the resource values promises insertion of new units at good initial positions. It is responsible for the better performance of DCS-GCS compared to algorithms that do not exploit the neighborhood relation between existing units.

[5]Fritzke inserts new units at a slightly different location, using not the neighbor with second largest resource but the most distant neighbor.

The outer loop closes by decrementing the resource values of all units [in procedure *decrement-ResourceValues*(β)]:

$$\tau_i(t+1) = \beta\tau_i(t), \qquad 1 \le i \le N \tag{3.8}$$

where $0 < \beta < 1$ is a constant. This last step just avoids overflow of the resource variables. For off-line simulations, $\beta = 0$ is the natural choice.

3.1 Unsupervised Simulation Results. Before turning to the results of two simulations of unsupervised DCS-GCS on artificial data, we want to draw the reader's attention to the kind of data preprocessing necessary to obtain satisfying results with GCS and DCS-GCS. First, due to the insertion strategy, GCS like networks have difficulties unfolding if the starting units are very close to each other. Maximally distant data points are best suited for initialization. Second, because learning constants are usually high and will not be "frozen," the algorithms are very sensitive to the order of data presentation. Therefore, we strongly recommend choosing a random order presentation to prevent erratic oscillations.

In our first example, the training set T consists of 2000 examples drawn from $[0,100] \times [0,100] \subset \Re^2$ according to

$$P(v) = \begin{cases} 1/4; & v \in [10,40] \times [10,40] \\ 1/4; & v \in [60,90] \times [10,40] \\ 1/4; & v \in [60,90] \times [60,90] \\ 1/4; & v \in \{p \mid (40 \le p_x = p_y \le 60)\} \end{cases} \tag{3.9}$$

Thus our feature manifold M consists of three squares, two of them connected by a line. The development of our unsupervised DCS-GCS network is depicted in Figure 1, with the initial situation of only two units shown in the upper left. Examples are represented by small dots, the centers of receptive fields by small circles, and the lateral connections by lines connecting the circles. From left to right the network is examined after 0, 9, and 31 epochs of training (i.e., after insertion of 2, 11, and 33 neural units).

After 31 epochs the network has built a perfectly topology preserving map of M, the lateral connection structure nicely reflecting the shape of M: Where M is two-dimensional the lateral connection structure is two-dimensional, and it is one-dimensional where M is one-dimensional. Note that a connected component analysis could recognize that the upper right square is separated from the rest of M. The parameters for this simulation were $\varepsilon_B = 0.1$, $\varepsilon_N = 0.006$, $\beta = 0$, and $\alpha = \sqrt[|T|]{\theta}$. The accumulated squared distance to stimuli served as the resource.

The quantization error $E_q = \sum_{v \in T} \|v - w_{bmu(v)}\|^2$ dropped from 100% (3 units) to 3% (33 units).

The second simulation deals with the two-spirals benchmark. Data were obtained by running the program "two-spirals" (provided by CMU) with parameters 5 (density) and 6.5 (spiral radius) resulting in a training

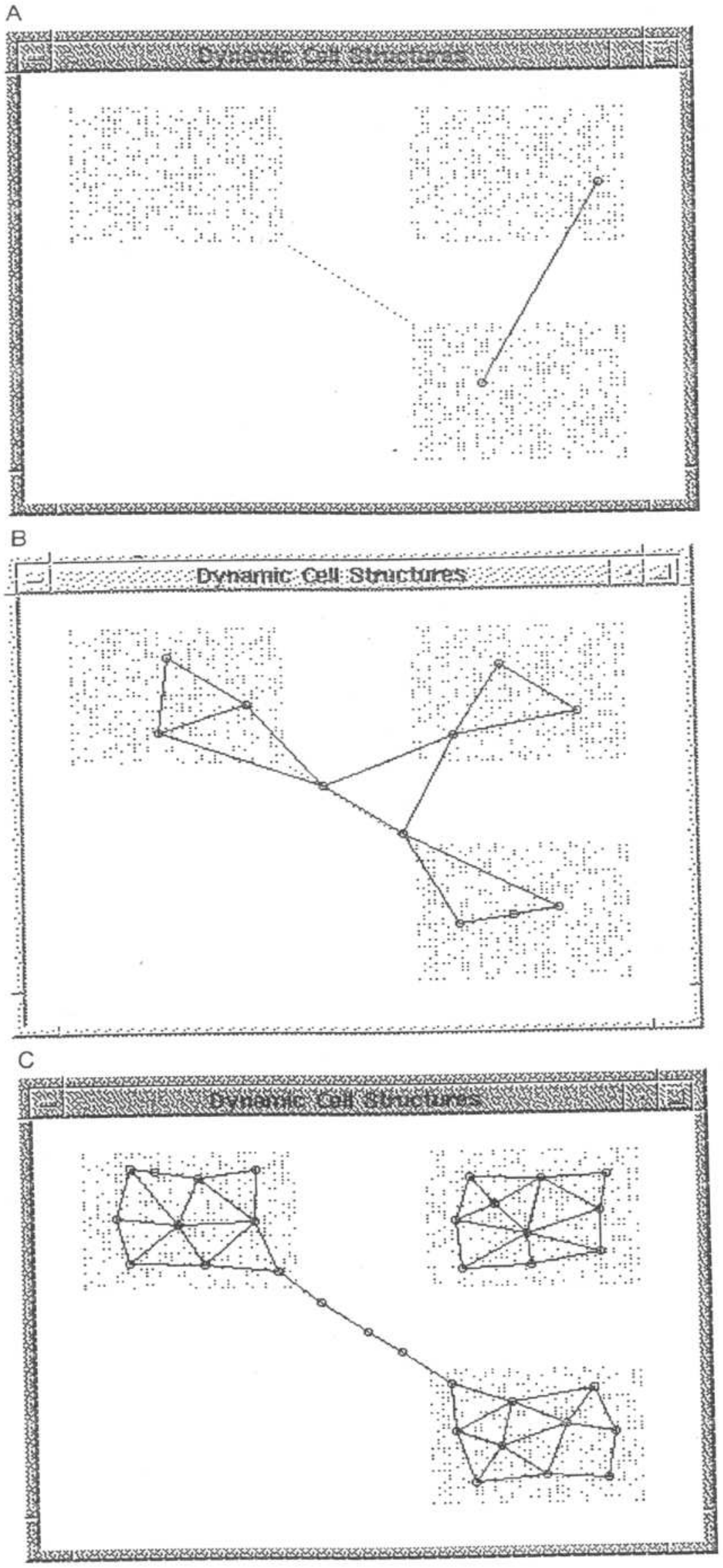

Figure 1: Unsupervised DCS-GCS on artificial data.

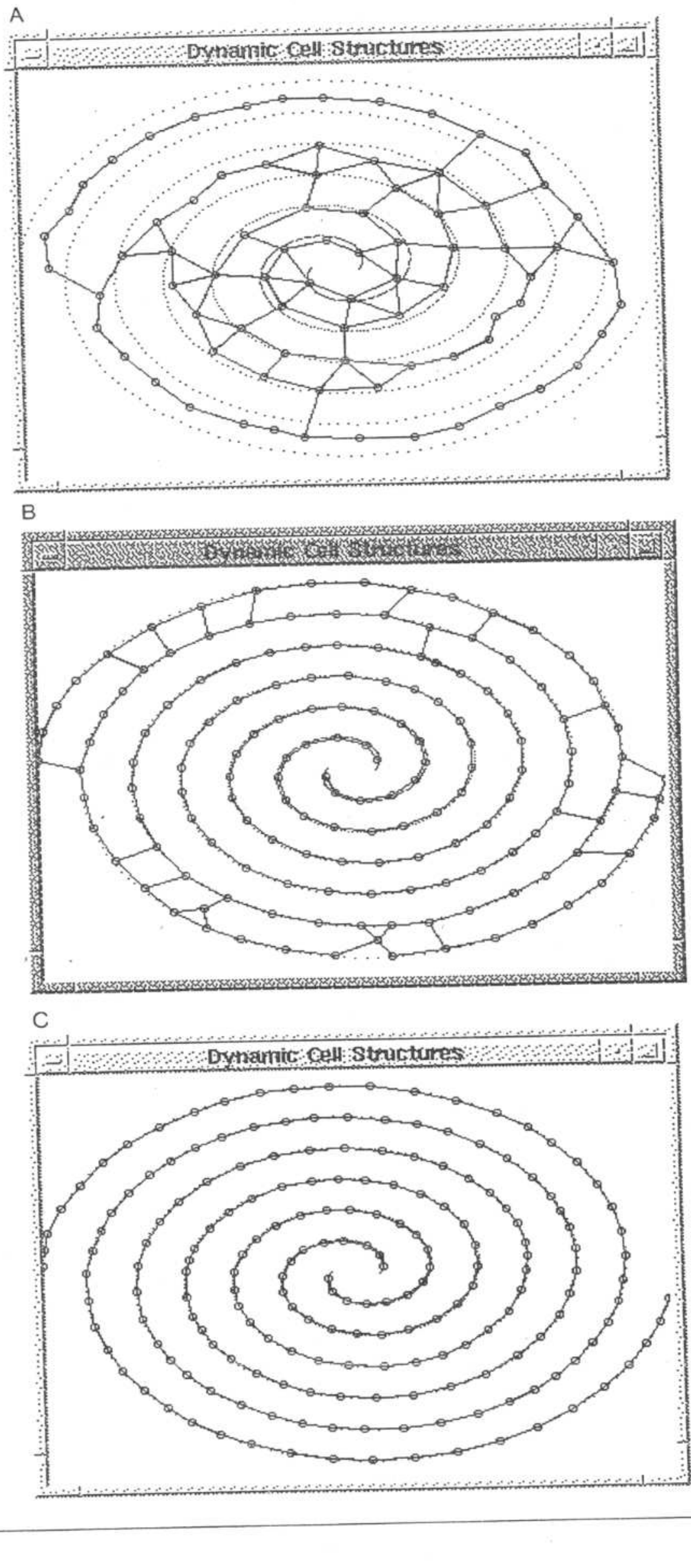

Figure 2: Unsupervised learning of two spirals.

```
void DCS-GCSalgorithm()
{ float $\varepsilon_B$, $\varepsilon_N$, $\eta$, $\alpha$, $\beta$, $\sigma$, $\theta$;
  InputVector $v$;
  OutputVector $u$, $y$;
  neuronP $n1$, $n2$, $bmu$, $second$;

  getNextExample(&$v$,&$u$,TRAIN);
  $n1$ = insertNewNeuron($v, u$);
  getNextExample(&$v$,&$u$,TRAIN);
  $n2$ = insertNewNeuron($v, u$);
  enforceConnection($n1$, $n2$, 1.0);

  do{
    for ( $\lambda$ times){
      getNextExample(&$v$, &$u$, TRAIN,);
      calculateTwoClosest(&$bmu$, &$second$, $v$);
      competitiveHebb($bmu$, $second$, $\alpha$, $\theta$);
      restrictedKohonen($bmu$, $v$, $\varepsilon_B$, $\varepsilon_N$);
      calculateOutput($y$, $v$, $bmu$, $\sigma$);
      deltaRule($y$, $bmu$, $u$, $\eta$);
      updateResource($bmu$, $v$, $y$, $u$);
    }
    if (stoppingCriterion())break;
    addNewNeuron();
    decrementResourceValues($\beta$);
  }loop;
}
```

Figure 3: The supervised DCS-GCS algorithm.

set T of 962 examples. The data represent two distinct spirals in the x-y plane. Unsupervised *DCS-GCS* at work is shown in Figure 2, after insertion of 80, 154, and, finally, 196 units. With 196 units a perfectly topology preserving map of M has emerged, and the two spirals are clearly separated. Note that the algorithm has learned the separation in a totally unsupervised manner, i.e., not using the labels of the data points (which are provided by CMU for supervised learning). Parameters and the type of resource are the same as in the previous simulation.

4 Supervised DCS-GCS

The algorithm for supervised DCS-GCS (see Fig. 3) differs from the unsupervised version in just two lines of code: the calls to procedure *calcOutput*(y, v, bmu, σ) for calculating the output vector y of the network and procedure *deltaRule*(y, bmu, u, η) for adjusting the output vectors o_i, $(1 \leq i \leq N)$ according to the teaching output vector u.

It works very similarly to its unsupervised version except

- when a neuron n_i is inserted by *insertNewNeuron*(v, u) an output vector o_i will be attached to it with $o_i = u$. If it is added by *addNewNeuron*( ) its output vector is initialized by $o_i = o_l + \gamma(o_n - o_l)$

- the output y of the network is calculated as a weighted sum of the best matching unit's output vector o_{bmu} and the output vectors of its neighbors o_i, $i \in Nh(bmu)$,

$$y = \sum_{i \in \{bmu \cup Nh(bmu)\}} a_i o_i \tag{3.10}$$

  where a_i is the activation of neuron i on stimulus v. We used activation functions

$$a_i = \frac{1}{\sigma \|v - w_i\|^2 + 1} \tag{3.11}$$

  with σ, $\sigma > 0$ representing the size of the receptive fields. In our simulations, the sizes of receptive fields have been equal for all units.

- adaptation of output vectors by *deltaRule*(y, bmu, u, η): A simple delta-rule is employed to adjust the output vectors of the best matching unit and its neighbors:

$$\Delta o_j = \eta a_j (u - y), \qquad j \in bmu \cup Nh(bmu) \tag{3.12}$$

Most important, the approximation (classification) error can be used for resource updating. This idea of Fritzke leads to insertion of new units in regions where the approximation error is worst, thus promising to outperform algorithms that do not employ such a criterion for insertion. In our simulations we used the accumulated squared distance of calculated and teaching output

$$\Delta \tau_{bmu} = \|y - u\|^2 \tag{3.13}$$

4.1 Variations on DCS-GCS. In this section we want to discuss some variations on DCS-GCS. While having been tested in our Benchmark simulations but not significantly affecting performance, it may be useful to reconsider them in other applications.

4.1.1 Normalized Radial Basis Functions. Normalized radial basis functions have often been reported to result in better interpolation characteristics. Simply change equations 3.10 and 3.12 accordingly.

4.1.2 Variable Sized and Formed Receptive Fields. In general, one might benefit from variable sized and formed receptive fields instead of fixed σ. Using local covariance matrix estimation, not only the topology of the network but also the form of receptive fields can adapt to the topology of the feature manifold M. However, these modified activation functions can no longer be used for perfect topology learning that has to be done separately.

4.1.3 Error Based Adaptation of Reference Vectors. As stressed by Fritzke, one of the key ideas of GCS is that the distribution of units generally should not depend on the input probability distribution $P(v)$ but should reflect an even distribution of resource values among the units. In GCS, this idea is supported only by the insertion strategy, whereas the Kohonen type adaptation still depends on $P(v)$ only. Thus the larger λ the more the distribution of units will be determined by $P(v)$. A first improvement could be to use the usual stochastic gradient with respect to the output error for updating the weights of the bmu and its topological neighbors. This gradient based adaptation (which is also consistent with the delta rule) would then be responsible for output error minimization, while the insertion process tries to evenly distribute resource values. Alternative ideas for error-weighted adaptation aiming at an even distribution of errors in K-means type algorithms can be found in Chinrungrueng and Sequin (1993).

4.2 Supervised Simulation Results. We applied our supervised DCS-GCS algorithm to three CMU benchmarks, the supervised two-spiral problem, the speaker independent vowel recognition problem, and the sonar mine/rock separation problem. The first two problems have also been used by Fritzke to test his GCS, so that we have some indication of the performance of DCS-GCS relative to GCS on these problems.

4.2.1 General Method of Simulation. DCS-GCS like any other algorithm using a stochastic gradient (sample by sample) following update rule is sensitive to the order of sample presentation. Moreover, in our simulations the order of sample presentation also determined the two starting units. We therefore repeated our simulations with 20 different random orders of sample presentations[6] and will subsequently report the statistics of these runs. These are e_{min} and n_{min}, the minimal classification error and number of neural units for this error, e_{mean} and n_{mean}, the mean classification error and number of units, and σ_e and σ_n, the standard deviation in classification error and number of units.

The second point that needs to be discussed is the choice of an adequate stopping criterion. Only the two spirals provide a concrete objective: The training error has to be zero. Consequently, this objective together with the (self-imposed) constraint, that the number of units should not exceed the number of training samples, defines the stopping criterion.

Things are different with the vowel recognition and the sonar classification benchmark. Here, one has to be as good as possible but the regulations neither bound classification performance nor number of units. Furthermore, it is well known (Robinson 1989) that the minimum for the training set does not coincide with the minimum for the test set. We

[6]Using just 20 successive "seeds" for the random generator used to mix the training sets.

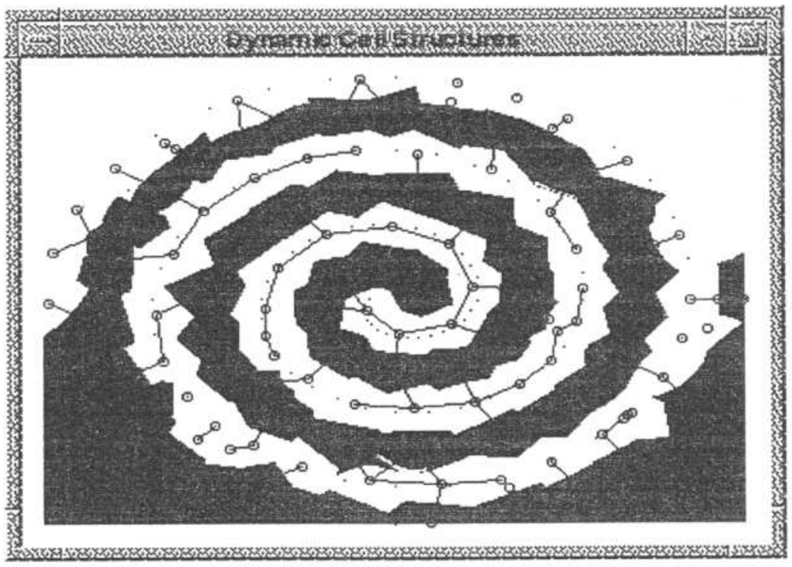

Figure 4: Supervised learning of two spirals.

therefore did not define an explicit stopping criterion but by cross-validation recorded the performance of the network (classification error and number of units) on the test set after each epoch of training. The result of each simulation was then set to the best result thus obtained. A run was terminated if the number of units exceeded the number of training samples.

4.2.2 The Two Spirals Problem. Let us first turn to the supervised version of the two-spiral problem already introduced in the previous section. The training set for benchmarking had to be produced by running the "two spirals" program with parameters 1 (density) and 6.5 (radius), producing 194 examples, each consisting of an input vector $v \in \Re^2$ and a binary label indicating to which spiral the point belongs. Obviously the spirals cannot be linearly separated. The task is to train the examples until the learning system can produce the correct output for all of them and to record the time. No test set is provided. While this task is trivial for algorithms doing essentially table-lookup, it is a very hard task for MLPs with sigmoidal activation functions. GCS and DCS-GCS are somewhere in between, using locally tuned units but not directly placing them on data points.

The decision regions learned after 135 epochs of supervised DCS-GCS training are depicted in Figure 4. Black indicates assignment to the first and white assignment to the second spiral. The network and the examples are overlaid. The classification error on the training set (measured in accordance with the CMU regulations) dropped to 0%. Parameters are

Table 1: DCS-GCS Classification Results on Two Spirals Problem.

Training set performance						Test set performance		
e_{min}	n_{min}	e_{mean}	n_{mean}	σ_e	σ_n	e_{min}	e_{mean}	σ_e
0.0%	135	0.36%	163	0.43%	14	0.7%	1.5%	0.7%

Table 2: Epochs for Supervised Learning of Two Spirals.

Network model	Number of epochs	Reported in
Backpropagation	20000	Lang and Witbrock (1989)
Cross entropy BP	10000	Fahlman and Lebiere (1990)
Cascade-correlation	1700	Fahlman and Lebiere (1990)
GCS	180	Fritzke 1993
DCS-GCS	135	This article

$\varepsilon_B = 0.2$, $\varepsilon_N = 0.012$, $\beta = 0$, $\alpha = \sqrt[|T|]{\theta}$, $\eta = 0.3$, $\sigma = 2.0$, and the accumulated squared output error served for resource updating, $\Delta\tau_{bmu} = ||y-u||^2$, $y, u \in \{-1, 1\}$. The statistics for this parameter set are presented in Table 1. In 10 of 20 runs the training set performance dropped to zero before utilizing the maximum number of 194 units. Among the other runs, maximally three training samples have been misclassified. The difference in classification reflects the dependency on the order of sample presentation. The performance on the test set is given for reasons of completeness; it is not required by the benchmark.

Supervised spiral learning nicely demonstrates properties of GCS and DCS-GCS: The distribution of units does not reflect the input probability density (which is highest in the center and continuously decreasing toward the periphery) but by trying to equalize resource values is relatively dense at the periphery. This is not surprising, since due to the decreasing probability density classification is most difficult at the periphery. The "unfolding problem" already mentioned in Section 3.1 further contributes to spatially decreasing classification performance (reference vectors at the center have experienced more adaptation steps than those at the periphery). On the other hand, topology preservation is rather bad due to the sparse data.

For comparison we list results obtained by Lang and Witbrok (1989), Fahlman and Lebiere (1990), and Fritzke (1993c) in Table 2.

4.2.3 The Speaker Independent Vowel Recognition Problem. The data for the speaker independent vowel recognition problem comprises a training set of 582 examples and a test set of 462 examples. The input vector is 10-dimensional, $v \in [0, 1]^{10}$, and we used an 11-dimensional output vector

Table 3: DCS-GCS Classification Results on Speaker Independent Vowel Recognition.

Test set performance						Training set performance		
e_{min}	n_{min}	e_{mean}	n_{mean}	σ_e	σ_n	e_{min}	e_{mean}	σ_e
35%	108	40%	97	2%	32	0.5%	7%	5%

Table 4: Speaker Independent Vowel Recognition.

Classifier	Hidden units	Percent correct
Single layer perceptron		33
Multilayer perceptron	88	51
Modified Kanerva model	528	50
Radial basis function	528	53
Gaussian node network	528	55
Square node network	88	55
Nearest-neighbor		56
3D GCS	158, 165, 154	61, 62, 67
5D GCS	135, 196	66, 66
DCS-GCS	108	65

u with a 1 in the jth position indicating the presence of the jth vowel and -1 in all the other positions. For details about the preprocessing steps yielding these input vectors the interested reader is referred to the thesis of Robinson (1989). With $\varepsilon_B = 0.05$, $\varepsilon_N = 0.006$, $\beta = 0$, $\alpha = \sqrt[|T|]{\theta}$, $\eta = 0.075$, $\sigma = 2.0$ and the same resource as in the previous simulation we obtained a peak performance of 65% correctly classified test samples with 108 neural units. The statistics for this parameter set are presented in Table 3. For comparison, Table 4 shows results obtained by others. The upper part of the table was published by Robinson, reporting final performance figures after about 3000 trials,[7] the lower in Fritzke (1992a), reporting peak performances of some 3D and 5D GCS for particular (unpublished) parameter sets and orders of presentation.

The figures indicate that DCS-GCS beats the conventional methods on this problem with respect to average peak classification performance and qualitatively compares to GCS (peaks above the 60% margin). Since for single simulation runs the fluctuations can easily wipe out any difference between methods, and reporting best results may be considered a questionable method, we do not regard the gap in peak performance between DCS-GCS and GCS as statistically significant. Note that DCS-GCS does

[7]Robinson reports a peak performance of about 54% for most models.

Table 5: DCS-GCS Classification Results on Sonar Target Recognition.

Test set performance						Training set performance		
e_{min}	n_{min}	e_{mean}	n_{mean}	σ_e	σ_n	e_{min}	e_{mean}	σ_e
5%	88	8%	85	2%	12	0%	2%	3%

not rely on a prespecified connection structure (but learns it by means of its easy-to-implement competitive Hebb rule).

4.2.4 The Sonar Target Classification Problem. Our last simulation concerns a data set used by Gorman and Sejnowski (1988) in their study on classification of sonar data. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Our input vector is 60-dimensional, $v \in [0,1]^{60}$, and we employ a 2D output vector $u \in \{-1,1\}^2$. The training and the test set contain 104 examples each.

Gorman and Sejnowski (1988) report best results of 90.4% correctly classified test examples for a standard BP network with 12 hidden units and 82.7% for a nearest-neighbor classifier. Supervised DCS-GCS reaches a peak performance of 95% correctly classified test examples after only 88 epochs of training. Parameters were $\varepsilon_B = 0.2$, $\varepsilon_N = 0.006$, $\beta = 0$, $\alpha = \sqrt[|T|]{\theta}$, $\eta = 0.3$, $\sigma = 0.5$, and the squared output error served as the resource. The statistics for this parameter set are presented in Table 5.

4.3 Complexity of DCS. We restrict our complexity analysis to the time a DCS algorithm needs to process a single stimulus (including response calculation and adaptation).

Here, the main argument in favor of DCS is that the topologically nearest neighbors of the best matching unit can be found in linear time by exploiting the induced Delaunay triangulation. Searching for the best matching unit can obviously be accomplished in linear time,[8] too. Hence, if connection updates (equations 2.2 or 2.4) are restricted to the best matching unit and its neighbors, the serial time complexity for processing a single stimulus is $O(N)$. Yet for planar manifolds it is well known (Preparata and Shamos 1985) that the number of edges of the Delaunay triangulation is $O(N)$, implying linear time complexity even if all connections are updated on each stimulus. The number of edges of the induced Delaunay triangulation also determines the space complexity of DCS. Clearly, $O(N^2)$ is an upper bound (and we are not aware of lower upper bounds except for the planar case).

[8]In parallel implementations the best matching unit can be found in constant time, as has been pointed out in Martinetz (1992).

Note that the serial time complexity of the neural gas with k nearest-neighbors is $\Omega(N)$, approaching $O(N \log N)$ for $k \rightarrow N$.

5 Conclusion

We introduced the idea of RBF networks that concurrently learn and utilize perfectly topology preserving feature maps for adaptation and interpolation. This family of ANNs, which we termed dynamic cell structures, offers conceptual advantage compared to classical Kohonen-type SOMs since the emerging lateral connection structure maximally preserves topology. We discussed the DCS-GCS algorithm as an instance of DCS. Compared to its ancestor GCS of Fritzke, this algorithm elegantly avoids computational overhead for handling sophisticated data structures. Having linear (serial) worst time complexity, DCS may also be considered as an improvement of Martinetz's neural gas idea. The simulations on CMU-Benchmarks indicate that DCS indeed has practical relevance for classification and approximation.

Thus encouraged, we look forward to applying DCS at various sites in our active computer vision project, including image compression by dynamic vector quantization, sensorimotor maps for the oculomotor system, and hand–eye coordination, cartography, and associative memories.

Acknowledgments

The authors would like to thank their reviewers for numerous hints and for pointing out some earlier works on growing clustering and RBF networks. We also feel indebted to our colleagues, in particular Konstantinos Daniilidis and Josef Pauli, for fruitful discussions and stylistic improvements on this article.

References

Chinrungrueng, Ch., and Sequin, C. H. 1993. Adaptive K-means algorithm with error-weighted deviation measure, *Proc. IJCNN 93* 626–631.

Fahlman, S. E. 1993. *CMU Benchmark Collection for Neural Net Learning Algorithms*. Carnegie Mellon Univ., School of Computer Science, machine-readable data repository, Pittsburgh.

Fahlman, S. E., and Lebiere, C. 1990. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pp. 524–534. Morgan Kaufmann, San Mateo, CA.

Fritzke, B. 1991. Unsupervised clustering with growing cell structures. *Proc. IJCNN 91* 531–536.

Fritzke, B. 1992. Growing cell structures—a self organizing network in k dimensions. In *Artificial Neural Networks 2*, pp. 1051–1056. North-Holland, Amsterdam.

Fritzke, B. 1993a. Kohonen feature maps and growing cell structures—a performance comparison. In *Advances in NIPS*, Vol. 5, pp. 123–130. Morgan Kaufmann, San Mateo, CA.

Fritzke, B. 1993b. Vector quantization with a growing and splitting elastic net. *Proc. ICANN 93* 580–585.

Fritzke, B. 1993c. *Growing cell structures—a self organizing network for unsupervised and supervised training*. Tech. Rep., tr-93-026, ICSI Berkeley.

Fritzke, B. 1995. Growing cell structures—a self-organizing network for unsupervised and supervised training. *Neural Networks* **7**(9), 1441–1460.

Gorman, R. B., and Sejnowski, T. J. 1988. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks* **1**, 75–89.

Hakala, J., and Wernteg, H. W. 1992. Node allocation and topographical encoding (NATE) for inverse kinematics of a redundant robot arm. *ICANN'92*, 615–618.

Hakala, J., and Eckmiller, R. 1993. Node allocation and topographical encoding (NATE) for inverse kinematics of a 6-DOF robot arm. *ICANN'93*, 309–312.

Hart, P. E. 1968. The condensed nearest neighbor rule. *IEEE Transact. Inform. Theory* **IT-4**, 515–516.

Kohonen, T. 1987. Adaptive, associative, and self-organizing functions in neural computing. *Appl. Optics* **26**, 4910–4918.

Lang, K. J., and Witbrock, M. J. 1989. Learning to tell two spirals apart. In *Proceedings of the1988 Connectionist Models Summer School*, pp. 52–59. Morgan Kaufmann, San Mateo, CA.

Martinetz, T. 1992. Selbstorganisierende neuronale Netzwerke zur Bewegungssteuerung. Dissertation, DIFKI-Verlag.

Martinetz, T. 1993. Competitive Hebbian learning rule forms perfectly topology preserving maps. *Proc. ICANN 93*, 426–438.

Martinetz, T., and Schulten, K. 1994. Topology representing networks. *Neural Networks* **7**(3), 505–522.

Moody, J., and Darken, C. J. 1989. Fast learning in networks of locally-tuned processing units. *Neural Comp.* **1**(2), 281–294.

Platt, J. 1991a. A resource-allocating network for function interpolation. *Neural Comp.* **2**, 213–225.

Platt, J. 1991b. Learning by combining memorization and gradient descent. *NIPS'91* 714–721.

Preparata, F. P., and Shamos, M. I. 1985. *Computational Geometry—An Introduction*. Springer-Verlag, Berlin.

Reilly, D., Cooper, L., and Elbaum, C. 1982. A neural model for category learning. *Biol. Cybern.* **45**, 35–41.

Ritter, H., Martinetz, T., and Schulten, K. 1991. *Neuronale Netze*. Addison-Wesley, Reading, MA.

Robinson, A. J. 1989. Dynamic error propagation networks. Cambridge Univ., Ph.D. thesis.

Sebestyen, G. S. 1962. Pattern recognition by an adaptive process of sample set construction. *IRE Transact. Inform. Theory* **IT-8**, 82–91.

Specht, D. F. 1990. Probabilistic neural networks. *Neural Networks* **3**, 109–118.