Multiple Growing Cell Structures

Guojian Cheng and Andreas Zell
University of Tübingen
Wilhelm-Schickard-Institut for Computer Science
Department of Computer Architecture
Köstlinstr. 6, D-72074 Tübingen, Germany
{cheng,zell}@informatik.uni-tuebingen.de

Abstract

We present here a variant of B. Fritzke's GCS (Growing Cell Structures) — Multiple GCS (MGCS). In each insertion step of the GCS only one new cell is inserted in the middle of the edge connecting Maximum Resource Vertex (MSV) and the most distant node in the direct topological neighbourhood of MSV. But in our MGCS multiple new cells are inserted at the same time. The first new cell is inserted in the same way as in GCS. Besides that we insert the second new cell between the edge connecting the second MSV and it's most distant direct topological neighbourhood. In the same way we can insert the third and fourth new cells. Our goal is to speed up the convergence of the learning process. Although multiple growing cell mechanism can reduce the required number of learning epochs, it can also lead to an increased network structure. Simulation results on some neural network benchmarks (two-spiral problem, sonar–mine/rock separation problem, glass identification problem, horse colic fate prediction, soybean disease recognition, credit card approval prediction) indicate that, for many data sets, when the number of new cells is within three, the total performance of networks is at best, but that increasing the number of new cells beyond three has very little benefit and sometimes degrades performance.

Keywords: Growing cell structures; Self-organizing maps; Radial basis functions; Supervised learning; Voronoi regions; Neural network benchmarks

1. Introduction

Kohonen's Self-Organizing Maps (SOM) [19] generate mappings from high-dimensional signal spaces to lower dimensional topological structures. Their main features are formation of topology-preserving feature maps and approximation of the input probability distribution. However, SOMs have some disadvantages: their fixed number of neural units makes them impractical for applications where the optimal number of units is not known in advance. Also, a topology of fixed dimensionality results in problems if this dimensionality does not match the dimensionality of the feature manifold. Incremental artificial neural networks grow as they learn, and shrink as they forget. Based on SOM and Hebbian learning, B.

Fritzke introduced an incremental self-organizing network with variable topology, known as Growing Cell Structures (GCS) [14]. The initial topology of GCS is a k-dimensional simplex (a line for $k = 1$, a triangle for $k = 2$, and a tetrahedron for $k = 3, \cdots$). During self-organization new cells are added to the network and superfluous cells are removed. After each modification, the network consists solely of k-dimensional simplices.

The GCS has three main advantages over the SOM: First, the network structure is determined automatically by the input pattern. Second, the network size needs not to be predefined. Third, all parameters of the model are constant. It is, therefore, not necessary to define a decay schedule as for the SOM.

Through a combination of GCS with radial basis functions (RBF) B. Fritzke developed the Supervised GCS (SGCS) [14]. The SGCS has also three main advantages over other traditional supervised neural networks: First, number, diameter and position of RBF units are determined automatically through the growth process, which can be stopped as soon as the network performance is good enough. Second, since positioning of RBF units and supervised training of connection weights is performed in parallel, the current classification error can be used to determine where to insert new RBF units. Third, the networks can generalize well and have a relatively small size.

Since the first presentation of GCS in 1992 [10], many related algorithms of GCS have been developed, such as: dynamic cell structures [5], hierarchical growing cell structures [6], probabilistic growing cell structures [29], a hierarchical network for learning robust models of kinematic chains [22], growing radial basis function networks [1], structurally adaptive intelligent neural tree [26], unsupervised growing neural tree [8], growing multi-dimensional SOM [25].

Also, many application fields of GCS have been explored. For example: finite element mesh generation [3], function approximation [4], efficiently representing the radiosity kernel through learning [2], implementation of SGCS on the CNAPS neurocomputer [27], mobile robot navigation [28], neural information retrieval [32], unsupervised algorithms for learning emergent spatio-temporal correlations [7], pattern classification [13], visualization and clustering of high-dimensional data [11], combinatorial optimization [15], navigation on topologic feature-maps [33], robot motion planning [30], visualizing similarities in high dimensional input spaces [18].

The performance comparisons of GCS with other neural network models have been made by B. Fritzke [12] and F. Hamker [17].

In order to speed up the convergence of the learning process, we present a new variant of the GCS network — the Multiple Growing Cell Structures (MGCS). In each insertion step of GCS only one new cell is inserted in the middle of the edge connecting Maximum Resource Vertex (MRV) and the most distant node in the direct topological neighbourhood of the MRV. But in MGCS two or more new cells are inserted at the same time. The first new cell is inserted in the same way as in GCS. Besides that we insert the second new cell between the edge connecting the second MRV and its most distant node direct topological neighbourhood. With the same mechanism the third and fourth new cells are inserted. Another characteristic of the MGCS is that the insertion point of the new cells is variable. This idea is taken from J. Bruske's DCS (Dynamic Cell Structures) [5]. The exact location of its centre of receptive field is calculated according to the ratio of the resource values. This gives an estimate of the resource values as if the new unit was in the network right from the start.

The organization of the paper is as follows: After a brief description of GCS algorithm in section 2, the MGCS is presented in section 3. The MGCS is then tested with some neural network benchmarks and the results are discussed in section 4. Some conclusions and plans for future research are given in section 5.

2. A Brief Description of the GCS Algorithm

The goal of a GCS network is to create a mapping of vector space $\mathcal{V}$ with a n-dimensional probability distribution $P(\xi)$ onto a discrete k-dimensional topological structure $\mathcal{A}$. This mapping should have the following properties [14]: First, similar input signals are mapped onto topologically close elements of $\mathcal{A}$. Second, topologically close elements in $\mathcal{A}$ should have similar signals being mapped onto them. Third, regions of $\mathcal{V}$ where the probability density of the input vector distribution is higher should be represented by correspondingly more elements in $\mathcal{A}$. The initial topology of the network is a k-dimensional simplex. During self-organization new cells are added to the network and superfluous cells are removed. After each modification, the network consists solely of k-dimensional simplices.

Every cell has a n-dimensional vector $\mathbf{v}_i$ attached, which denotes the position of the cell i in the input vector space $\mathcal{V}$. The adaptation of the synaptic vectors is done as in Kohonen networks. But there are two important differences: First, only the best matching unit (bmu) and its direct topological neighbouring (dtn) cells are adapted. Second, the adaptation strength for bmu and dtn cells (ε_{bmu} and ε_{dtn} resp.) is constant over time. Fig. 1 shows an adaptation step in GCS.

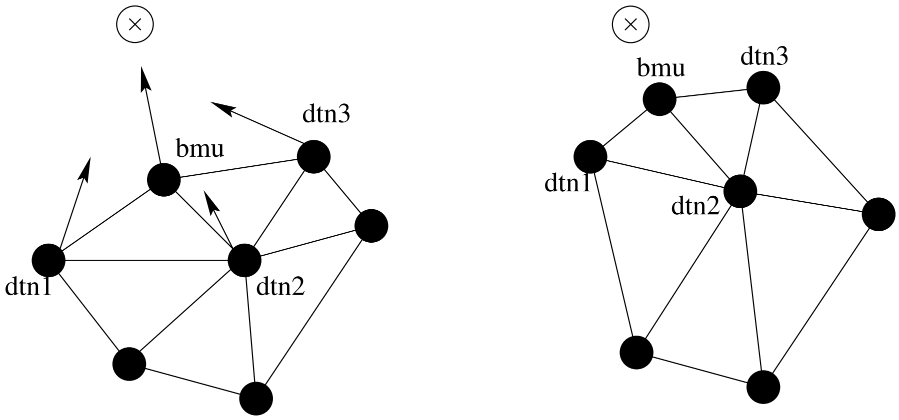

Figure 1. One adaptation step for a two-dimensional GCS. Only the best-matching unit and its direct topological neighbours are adapted. *left*: Occurrence of an input signal. *right*: After adaptation.

Each cell has a counter variable τ_i attached, which contains the number of times it has been best matching unit. After each adaptation all counters are decreased by a fraction α, because more recent signals should count more strongly than old ones. The goal of the network is to construct a structure according to the probability distribution of the input data.

When every cell has the same probability of being best-matching unit for the current input vector ξ, a structure with the synaptic vectors $\mathbf{v}_i$ distributed according to $P(\xi)$ can be obtained. With local signal counters, an estimate of $P(\xi)$ can be computed, namely the relative frequency of input signals received by a certain cell. The relative signal frequency of a cell c is:

$$h_c = \tau_c / \sum_{i \in \mathcal{A}} \tau_i$$

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

After a fixed number (λ) of adaptation steps the vertex (V_{max}) with the highest relative signal frequency is determined by:

$$h_{max} \geq h_c, \qquad (\forall c \in \mathcal{A})$$

Then the vertex with the most distant (V_{mdn}) in the direct topological neighbourhood of V_{max} is determined using:

$$\|\mathbf{v}_{mdn} - \mathbf{v}_{max}\| \geq \|\mathbf{v}_c - \mathbf{v}_{max}\|, \qquad (\forall c \in \mathcal{N}_{max})$$

In the above equations $\mathcal{N}_{max}$ denotes the set of direct topological neighbours of cell V_{max} (V_x represents the cell x in the network or the vertex in the graph and $\mathbf{v}_x$ represents the synaptic vector).

When the unit with the largest output has participated several times in a wrong classification result, a new unit is inserted. This insertion should lead to a higher amount of cells in areas where the classification task seems to be difficult. A new cell V_{new} is inserted between V_{max} and V_{mdn} and connected to the other cells so that the resulting structure consists again only of k-dimensional simplices. Fig. 2 shows the insertion of a new cell. V_{max} represents the cell with the highest signal counter. V_{mdn} is the most distant neighbour of V_{max}. V_{dtn1} and V_{dtn2} are two common dtn cells of V_{max} and V_{mdn}. V_{new} represents the new inserted cell.

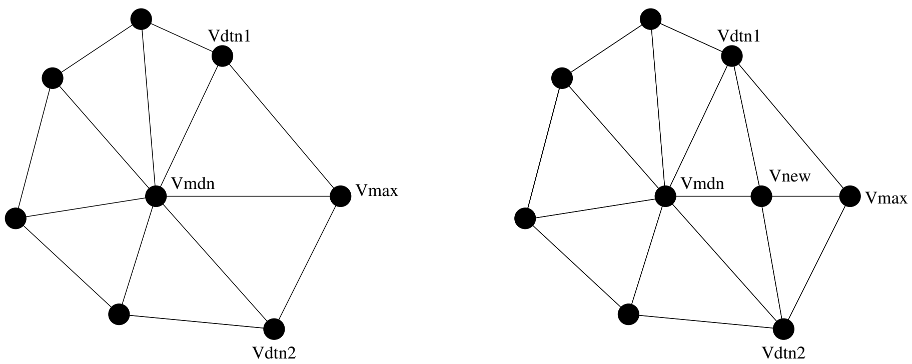

Figure 2. Insertion of a new cell. *left*: Situation before an insertion. The cell V_{max} has received the most input signals. V_{mdn} is the most distant neighbour of V_{max}. *right*: A new cell V_{new} has been inserted and linked with V_{dtn1} and V_{dtn2} which are common direct topological neighbours of V_{mdn} and V_{max}.

For a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ in R^n, the Voronoi region F_i of a particular vector $\mathbf{v}_i$ is defined as the set of all points in R^n for which $\mathbf{v}_i$ is the nearest vector. The synaptic vector $\mathbf{v}_{new}$ of the new cell V_{new} is initialized as the mean of the two previously connected cells:

$$\mathbf{v}_{new} = \frac{1}{2}(\mathbf{v}_{max} + \mathbf{v}_{mdn})$$

The signal counters in this region are redistributed according to the size of the Voronoi regions.

This model has a main characteristic that several adaptation steps (λ) are always followed by a single insertion. There exists a feedback relation between the two types of action [14]:

- Every adaptation step increases the signal counter of the best matching unit and increases thereby 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 the best matching unit for future input signals. We will show later how $||F_c||$ can be effectively approximated for arbitrary dimensions.

In order to model probability distributions which consist of separate regions, superfluous cells are periodically removed. To this purpose the cell with the smallest receptive field is selected and removed.

Growing Cell Structures are based on an unsupervised algorithm, but can be extended to supervised learning, i. e., Supervised GCS (SGCS) [14, 17]. This network combines two families of networks, SOM and radial basis function networks, with the ability to grow during the training process. It consists of three layers, including one hidden layer. The units of the hidden layer have a Gaussian activation function. The weight vector $\mathbf{v}_c$ of each unit defines the centre of the Gaussian function and an additional parameter σ_c defines its size. In addition, the hidden layer forms a topological structure defined through hypertetrahedrons which are a special type of neighbourhood definition of the Kohonen feature map. The neighbourhood also defines the standard deviation of the Gaussian activation function by using the mean distance between a unit and its neighbours. The output of the network is simply computed by a weighted sum of the unit activations in the hidden layer. The output unit with the largest activation gives the classification result.

The training algorithm of the supervised GCS is based on three parts: The first part is concerned with the adaptation of the synaptic weights ($\mathbf{v}$) from the input to the hidden layer. The second part performs insertion and deletion of cells. The classification error occurring for the training data is used to determine where to insert new cells. The third part is the adaptation of the weights ($\mathbf{w}$) from the hidden to the output layer.

The activation of the nodes in the hidden layer is calculated with a Gaussian function. The standard deviation σ_c is defined as the mean length of all edges emanating from the node c.

$$d_c(\xi) = exp(\frac{-||\xi - \mathbf{v}_c||^2}{\sigma_c^2}), \qquad \forall c \in \mathcal{A}$$

The width of the Gaussian kernel is determined by σ_c, which is the mean distance between a unit and its neighbours. It is better if only some fraction of the neighbours is used, so that the kernel becomes small even when only a few neighbours are close to the input. This idea is taken from D. Lowe's VSM (Variable-kernel Similarity Metric) [21]. We introduced a kernel diminution factor β relating the average neighbour distance to σ which is learned as a part of the optimization process:

$$d_c(\xi) = exp(\frac{-||\xi - \mathbf{v}_c||^2}{\beta\sigma_c^2}), \qquad \forall c \in \mathcal{A} \tag{1}$$

The activation of the output layer is calculated by the product of the weights and the activation:

$$\mathbf{o}_j = \sum_{c \in \mathcal{A}} \mathbf{w}_{jc} d_c, \qquad \forall j \in \{1, 2, \ldots, m\} \tag{2}$$

The adaptation of the weights is performed according to the delta rule:

$$\Delta \mathbf{w}_{jc} = \eta(\zeta_j - \mathbf{o}_j)d_c, \qquad \forall j \in \{1, 2, \ldots, m\}, \quad \forall c \in \mathcal{A} \tag{3}$$

The activation of the hidden nodes is not normalized. This noise-saturation dilemma can result in high output activations. If the learning rate η in the delta-rule learning step for output weight $\mathbf{w}_{jc}$ is not chosen appropriately, patterns near class boundaries can lead to high squared errors between the actual and desired output, which causes a severe explosion of the output weights. The GCS algorithm allows the representation of any feature space (e. g., separate feature subspaces).

3. Multiple Growing Cell Structures

3.1. Replace Signal Counter (τ) with Error Counter ($\mathcal{E}$)

The GCS can be used for different means. It is essential that new cells are inserted in such a way that the expected value of a certain error measure or, in other words, resource value (the accumulated resource values plays a nutrition-like role for the cell growing), becomes equal for all neurons [14]. Appropriate resources must have the property that the insertion of a new neuron V_{new} near an existing neuron V_{max} reduces the expected value of the resource of V_{max}.

The quantization error generated by the neurons is generally used as insertion criterion. This is the accumulated squared distance between the reference vector of the neuron and all input signals being mapped onto the neuron.

For unsupervised learning the signal counter of the bmu cell is increased by the squared error:

$$\Delta \mathcal{E}_{bmu} = ||\xi - \mathbf{v}_{bmu}||^2 \tag{4}$$

or

$$\Delta \mathcal{E}_{bmu} = \frac{1}{2}||\xi - \mathbf{v}_{bmu}||^2 \tag{5}$$

By using this measure as insertion criterion, new neurons are inserted not anymore near the neurons getting the most input signals but rather near those input signals which are very different from their reference vectors.

For supervised learning the resource variable of the current bmu cell is updated by adding the overall squared error between actual output $\mathbf{o} = (o_1, \ldots, o_m)$ and desired output $\zeta = (\zeta_1, \ldots, \zeta_m)$ to it:

$$\Delta \mathcal{E}_{bmu} = ||\zeta - \mathbf{o}||^2 \tag{6}$$

For a classification problem, we can use the classification error as resource value. In this case the resource would be updated according to:

$$\Delta \mathcal{E}_{bmu} = \begin{cases} 0 & \text{if } \xi \text{ is classified correctly} \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

This will result in the fact that new cells are only inserted in those regions of the input vector space where misclassifications still occur. In the following adaptation steps the new unit can develop different weights and thereby contribute to better error reduction in this area of the input vector space. Therefore a relative small size of network can be obtained.

Other error measures can also be used as resource value,e. g.
the logarithmic error:

$$\Delta\mathcal{E}_{bmu} = log(1 + \frac{1}{2}\|\zeta - \mathbf{o}\|^2), \tag{8}$$

the mean square error (m is the dimension of the vectors ξ and $\mathbf{v}_{bmu}$):

$$\Delta\mathcal{E}_{bmu} = \frac{1}{2m}\|\zeta - \mathbf{o}\|^2 \tag{9}$$

or the Bernoulli error measure (cross entropy):

$$\Delta\mathcal{E}_{bmu} = -\sum_{j=1}^{m}[\zeta_j log(o_j) + (1 - \zeta_j) log(1 - o_j)] \tag{10}$$

In another paper we will focus on training GCS with different error measures for pattern classification problems.

3.2. The Mechanism of MGCS

For convenience we use MGCS2 to represent a MGCS in which two new cells can be inserted at the same time in each insertion step. MGCS2 could also be called Double GCS (DGCS). In the same way we have MGCS3, MGCS4 and MGCS5. MGCS3 could also be called Triple GCS (TGCS). In the following MGCS2 was used as example to explain the mechanism of MGCS. The cell growing mechanism of MGCS2 is nearly the same as in GCS. The only difference is that in MGCS2 two new cells are inserted simultaneously instead of only one new cell as in GCS. In other words, in each insertion step of GCS only one new cell is inserted in the middle of the edge connecting the Maximum Resource Vertex (MRV) and the most distant node in the direct topological neighbourhood of MRV. But in MGCS2 two new cells are inserted at the same time. The first new cell is inserted in the same way as in GCS. Besides that, another new cell is inserted between the edge connecting the second MRV and the most distant node in its direct topological neighbourhood (see fig. 3). When the second MRV is in the direct topological neighbourhood of the first MRV, we insert only one new cell between first MRV and second MRV.

Each vertex V_c in the current network $\mathcal{A}$ maintains an error counter value $\mathcal{E}_c$ over each organizational process. Whenever an input vector is mapped onto a vertex, the square of the distance between the input vector ξ and the vertex's weight vector $\mathbf{v}_{bmu}$ of bmu cell is added to the error value (see equ. (4), (5)). Large cumulative error values occur at vertices that have too many input vectors mapped onto them. Their weight vectors fail to adequately represent all of the input vectors in that area. Therefore, new nodes are added to the areas with high cumulative error.

Fig. 3 shows the structure of a 2-dimensional MGCS2. V_{max1} is the first MEV, and V_{max2} is the second. V_{max1} has three direct topological neighbours: V_{dtn11}, V_{dtn12} and V_{mdn1}. V_{max2} has also three direct topological neighbours: V_{dtn21}, V_{dtn22} and V_{mdn2}. V_{mdn1} and V_{mdn2} are the most distant nodes from V_{max1} and V_{max2} respectively. Fig. 3 *left* shows the situation before the insertion of new cells. Fig.

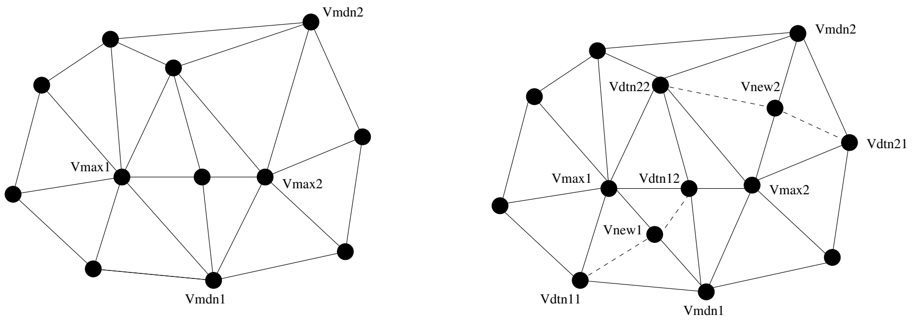

Figure 3. MGCS2: The first MEV is marked with V_{max1}. The second MEV is marked with V_{max2}. V_{max1} has three direct topological neighbours: V_{dtn11}, V_{dtn12} and V_{mdn1}. V_{max2} has also three direct topological neighbours: V_{dtn21}, V_{dtn22} and V_{mdn2}. V_{mdn1} and V_{mdn2} are the most distant nodes from V_{max1} and V_{max2} respectively. *left*: Before the insertion of new cells. *right*: After the insertion of new cells. The first new cell V_{new1} is inserted between V_{max1} and V_{mdn1}. The second new cell V_{new2} is inserted between V_{max2} and V_{mdn2}.

3 *right* shows the situation after the insertion of new cells. The first new cell V_{new1} is inserted between V_{max1} and V_{mdn1}, the second V_{new2} is inserted between V_{max2} and V_{mdn2}.

Suppose $V^{'}_{max2}$ is the MEV of direct topological neighbours of V_{max1}.

If $V^{'}_{max2} = V_{max2}$, i. e., V_{max2} is direct topological neighbour of V_{max1}, then only one new cell V_{new1} is inserted between V_{max1} and V_{max2}. We have:

$$\mathbf{v}_{new1} = \frac{1}{2}(\mathbf{v}_{max1} + \mathbf{v}_{max2}) \tag{11}$$

If $V^{'}_{max2} \neq V_{max2}$, then two new cells, V_{new1} and V_{new2}, are inserted at the same time. We have:

$$\mathbf{v}_{new1} = \frac{1}{2}(\mathbf{v}_{max1} + \mathbf{v}_{mdn1}) \tag{12}$$

$$\mathbf{v}_{new2} = \frac{1}{2}(\mathbf{v}_{max2} + \mathbf{v}_{mdn2}) \tag{13}$$

When insertion factor f_{in} is variable, we call it variable insertion. In this case:

$$\mathbf{v}_{new1} = (1 - f^{'}_{in1})\mathbf{v}_{max1} + f^{'}_{in1}\mathbf{v}_{max2} \tag{14}$$

or:

$$\mathbf{v}_{new1} = (1 - f_{in1})\mathbf{v}_{max1} + f_{in1}\mathbf{v}_{mdn1} \tag{15}$$

$$\mathbf{v}_{new2} = (1 - f_{in2})\mathbf{v}_{max2} + f_{in2}\mathbf{v}_{mdn2} \tag{16}$$

with:

$$f'_{in1}, f_{in1}, f_{in2} \in [0.1, 1.0]$$

or:

$$f'_{in1} = \frac{\mathcal{E}_{max2}}{\mathcal{E}_{max2} + \mathcal{E}_{max1}}, \quad f_{in1} = \frac{\mathcal{E}_{mdn1}}{\mathcal{E}_{mdn1} + \mathcal{E}_{max1}}, \quad f_{in2} = \frac{\mathcal{E}_{mdn2}}{\mathcal{E}_{mdn2} + \mathcal{E}_{max2}}$$

That is to say that the exact location of the centre of receptive field $\mathbf{v}_{new1,2}$ is calculated according to the ratio of the resource values $\mathcal{E}_{mdn1}, \mathcal{E}_{mdn2}$, and $\mathcal{E}_{max1}, \mathcal{E}_{max2}$. In this case, the maximum value of f_{in1} or f_{in2} is $\frac{1}{2}$ and this is just the same as GCS. The new cell V_{new1} is then connected to V_{mdn1} and V_{max1} and their direct topological neighbours: V_{dtn11} and V_{dtn12}. The new cell V_{new2} is then connected to V_{mdn2} and V_{max2} and their direct topological neighbours: V_{dtn21} and V_{dtn22}. Note that when V_{max2} is a cell of direct topological neighbours of V_{max1}, only one cell V_{new1} is inserted between V_{max1} and V_{max2}.

For MGCS3 we simply inserted new cells between the MEVs (V_{max1}, V_{max2} and V_{max3}) and their corresponding most distant neighbourhood ($V_{mdn1}, V_{mdn2} and V_{mdn3}$).

3.2.1 New Resource Value $\mathcal{E}_{new}$ and Output Weight Vector $\mathbf{w}_{new}$

The subscript new of $\mathcal{E}_{new}$ and $\mathbf{w}_{new}$ denotes $new1$ or $new2$. The new resource values and output weight vector of units V_{dtn1}, V_{dtn2} and V_{max} are redistributed according to the change of the Voronoi region. This is because the insertion of V_{new} leads to a new Voronoi region $F_c^{(new)}$ in the input space. At the same time the Voronoi regions of the topological neighbours of V_{new} are diminished. This change is reflected by appropriate redistribution of the error counter $\mathcal{E}_c$ which is computed as

$$\Delta\mathcal{E}_c = f_{va}\mathcal{E}_c, \qquad (\forall c \in \mathcal{N}_{new})$$

with Voronoi field diminution factor:

$$f_{va} = \frac{\|F_c^{(new)}\| - \|F_c^{(old)}\|}{\|F_c^{(old)}\|} \tag{17}$$

whereby $\|F_c\|$ is the n-dimensional volume of F_c. $\mathcal{N}_{new}$ denotes the set of direct topological neighbours of cell V_{new}. The initial value of the new cell is defined as:

$$\mathcal{E}_{new} = -\sum_{c \in \mathcal{N}_{new}} \Delta\mathcal{E}_{new}$$

Whenever a new cell V_{new} is inserted, it gets a new weight vector $\mathbf{w}_{new}$ of weighted connections to the n 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:

$$\Delta\mathbf{w}_c = f_{va}\mathbf{w}_{new}, \qquad (\forall c \in \mathcal{N}_{new}).$$

The initial output weight vector of the new cell is defined as

$$\mathbf{w}_{new} = -\sum_{c \in \mathcal{N}_{new}} \Delta\mathbf{w}_c$$

3.2.2 Computation of the Voronoi Regions

When the dimensionality n of pattern space is greater than 2, the computation of the Voronoi tessellation is very complex. We approximate the volume of the Voronoi region F_c with a t-dimensional hypercube:

$$||F_c|| \approx \tilde{f}_c = (\bar{l}_c)^{\tilde{t}}$$

where $\tilde{t}$ is an estimated value of the true data dimensionality t and $\bar{l}_c$ is the mean length of the edges emanating from cell c.

B. Fritzke [14] has given some general rules for choosing such an estimate $\tilde{t}$:

- Always set $\tilde{t} \leq n$.
- 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.
- Perform a principal component analysis of the data. Then $\tilde{t}$ should be set to the number of principal eigenvalues of the covariance matrix of the data.

It is stressed that GCS is not very sensitive to the choice of $\tilde{t}$ and therefore it is not a critical step. The only case one should avoid is to choose a value of $\tilde{t}$ which is much too high.

3.3 The Complete Algorithm of the 2-dim Supervised MGCS2

Step 1. Initialization

1.1 Initialize cell structure $\mathcal{A}$ with a triangle at random positions in vector space $\mathcal{V} = R^n$.

1.2 Assign λ, ε_{bmu}, ε_{dtn}, η, α, β.

1.3 Create m linear output units and a weighted connection $\mathbf{w}_{jc}$ from each cell $c \in \mathcal{A}$ to each output unit j, $(j \in \{1, 2, \ldots, m\})$.

1.4 Associate every cell (vertex of the triangle) with a Gaussian function.

Step 2. λ-Times Adaptation

2.1 Load I/O-pattern pair $(\xi, \zeta) \in (R^n \times R^m)$ from training data.

2.2 Determine best-matching unit bmu for ξ:

$$bmu(\xi) = arg\ min_{c \in \mathcal{A}} ||\xi - \mathbf{v}_c||$$

2.3 Increase matching for bmu and its direct topological neighbours dtn:

$$\Delta\mathbf{v}_{bmu} = \varepsilon_{bmu}(\xi - \mathbf{v}_{bmu}), \quad \Delta\mathbf{v}_{dtn} = \varepsilon_{dtn}(\xi - \mathbf{v}_{dtn}),$$

2.4 Compute activation d_c for every cell $c \in \mathcal{A}$ with equ. (1).

2.5 Compute the vector **o** $= (o_1, o_2, \dots, o_m)$ of all output unit activations with equ. (2).

2.6 Perform one δ-rule learning step for the weights according to equ. (3).

2.7 Increase resource variable $\mathcal{E}_{bmu}$ through equ. (6) – (10).

2.8 Decrease the error variable of all units:

$$\Delta\mathcal{E}_i = -\alpha\mathcal{E}_i, \qquad \forall i \in \{1, 2, \dots, n\}$$

Step 3. Insertion of new cells

3.1 Determine the cell V_{max1} and V_{max2} with the first and second maximum accumulated error respectively:

$$V_{max1} = arg\ max_{c\in\mathcal{A}} \mathcal{E}_c$$

$$V_{max2} = arg\ max_{c\in\mathcal{A}, c\neq V_{max1}} \mathcal{E}_c$$

3.2 Determine the cell V'_{max2} with the maximum accumulated error in the direct topological neighbourhood of V_{max1}:

$$V'_{max2} = arg\ max_{c\in\mathcal{N}_{max1}} \mathcal{E}_c$$

3.3 Determine the cell V_{mdn1} and V_{mdn2} with the most distant of direct topological neighbourhood of V_{max1} and V_{max2} respectively:

$$V_{mdn1} = arg\ max_{c\in\mathcal{N}_{max1}} \|\mathbf{v}_{max1} - \mathbf{v}_c\|$$

$$V_{mdn2} = arg\ max_{c\in\mathcal{N}_{max2}} \|\mathbf{v}_{max2} - \mathbf{v}_c\|$$

3.4 Cell Insertion

If V'_{max2} = V_{max2} — Single Cell Insertion :

Insert a new cell V_{new1} between V_{max1} and V_{max2} according to equ. (11) or (14).
Connect V_{new1} with V_{max2} and V_{max1}.
Disconnect V_{max2} from V_{max1}.
Connect V_{new1} with V_{dtn11} and V_{dtn12}.

If $V'_{max2} \neq V_{max1}$ — Double Cells Insertion :

Insert a new cell V_{new1} between V_{max1} and V_{mdn1} according to equ. (12) or (15).
Connect V_{new1} with V_{mdn1} and V_{max1}.
Disconnect V_{mdn1} from V_{max1}.
Connect V_{new1} with V_{dtn11} and V_{dtn12}.

Insert another new cell V_{new2} between V_{max2} and V_{mdn2} according to equ. (13) or (16).
Connect V_{new2} with V_{mdn2} and V_{max2}.
Disconnect V_{mdn2} from V_{max2}.
Connect V_{new2} with V_{dtn21} and V_{dtn22}.

3.5 Redistribute the error variables and weight vectors of all direct topological neighbours of new cells by a Voronoi factor f_{va}:

$$\Delta\mathcal{E}_{dtn} = f_{va}\mathcal{E}_{dtn}, \qquad \Delta\mathbf{w}_{dtn} = f_{va}\mathbf{w}_{dtn}$$

where f_{va} is calculated with equ. (17).

3.6 Set the error variable and weight vector of the new unit V_{new}:

$$\mathcal{E}_{new} = -\sum_{c\in\mathcal{N}_{new}} \Delta\mathcal{E}_{new}, \qquad \mathbf{w}_{new} = -\sum_{c\in\mathcal{N}_{new}} \Delta\mathbf{w}_{new}$$

Step 4. Unless the stopping criterion is satisfied, go to Step 2.

4. Supervised Learning Simulation Results

We applied our supervised MGCS algorithm to six neural network learning benchmarks: the two-spiral problem; the sonar–mine/rock separation problem; glass identification problem; horse colic fate prediction; soybean disease recognition; credit card approval prediction. The first two benchmarks are from the CMU database [31], the others are from PROBEN1 [23].

4.1. Two Spirals Problem

The task is to learn to discriminate between two sets of training points which lie on two distinct spirals in the x-y plane (see fig. 4 *left*). These spirals coil three times around the origin and around one another. This appears to be a very difficult task for back-propagation networks and their relatives. Problems like this one, whose inputs are points on the 2-D plane, are interesting because we can display the 2-D receptive field of any unit in the network.

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 $\mathbf{v}_i \in R^2$ and a binary label $(+1, -1)$ indicating to which spiral the point belongs. Obviously the spirals can not be linearly separated. We trained the network with the examples until the learning system produced the correct output for all of them and recorded the required number of epochs and cells. No test set is provided.

In order to solve the two-spiral problem we generated two 2-dimensional MGCS: MGCS2 and MGCS3. Fig. 4 shows the MGCS2 and MGCS3 networks and their corresponding decision regions. Black indicates assignment to the first, white assignment to the second spiral. The classification error on the training set dropped to 0 and the accumulated squared output error served for resource updating, $\Delta\mathcal{E}_{bmu} = \frac{1}{2}||\zeta - \mathbf{o}||^2, \zeta, \mathbf{o} \in \{-1, 1\}$.

It took MGCS2 96 training epochs (CPU: 5.59 sec.) and 138 growing cells until the network achieved zero classification error. In MGCS3 they are 65 epochs (CPU: 4.6 sec.) and 159 cells. For the original SGCS it needed 180 epochs (CPU: about 11.68 sec.) and 145 growing cells. Fig. 5 shows the curve of the mean squared error and classification error vs. learning epochs.

Table 1 gives a detailed comparison between some earlier methods and our approach for the two-spiral problem. As it can be seen the number of epochs and CPU time required by the MGCS is about two times smaller than for original SGCS or DCS-GCS. The performance of MGCS4 is almost the same as

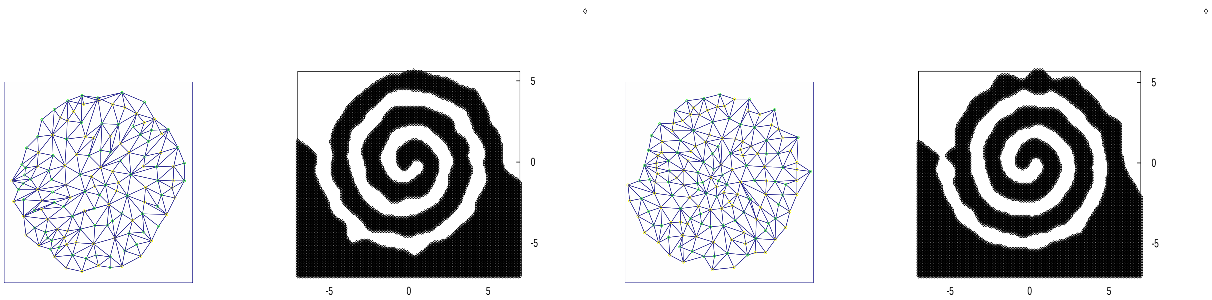

Figure 4. Performance of the MGCS2 and MGCS3 on the two-spiral benchmark. Simulation parameters: $\lambda = 240, \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.00005, \beta = 0.5, \eta = 0.45$, no removal of cells. *left half*: Final MGCS2 network with 138 cells and its decision regions. *right half*: Final MGCS3 network with 159 cells and its decision regions.

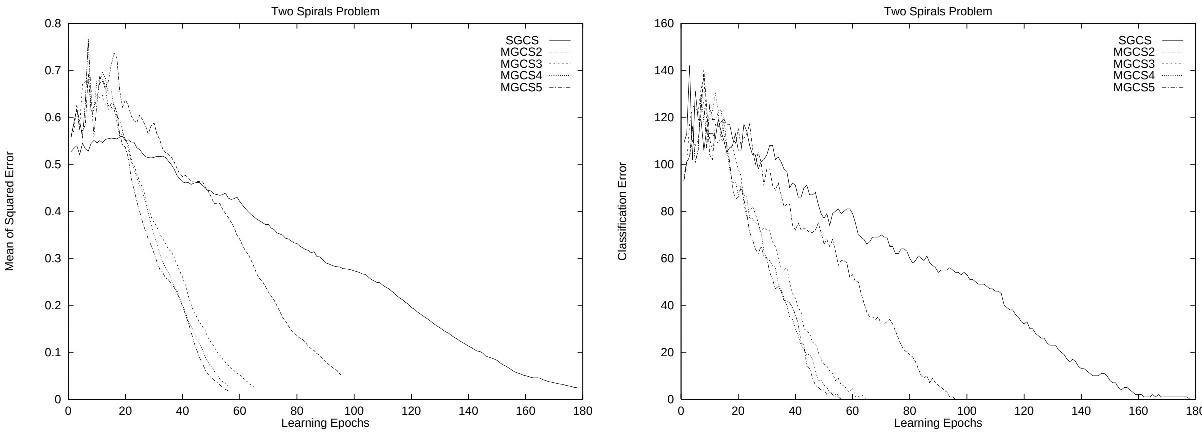

Figure 5. Performace comparison of MGCS and SGCS for the two-spirals problem. *left:* Mean squared error vs. epochs. *right:* Classification error vs. epochs.

of MGCS3. For MGCS5 it would take too many cells and could lead to an oversized network structure. This indicates that increasing the number of new cells beyond three has very little benefit and sometimes degrades performance. Therefore in the following testings we considered only MGCS2 and MGCS3.

4.2. Sonar–mine/rock Separation Problem

The Sonar data set was used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [16]. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The input vector is 60 dimensional and we employed a 1-dimensional output vector. The training and the test set contain 104

Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Reported in
Backpropagation	20000			K.J.Lang [20]
Cross Entropy BP	10000			S.E.Fahlman [9]
Cascade-Correlation	1700			S.E.Fahlman [9]
DCS-GCS	177			J.Bruske [5]
SGCS	180	11.68	145	B.Fritzke [11]
MGCS2	96	5.59	138	this article
MGCS3	65	4.6	159	this article
MGCS4	56	4.52	169	this article
MGCS5	56	6.84	205	this article

Table 1. Epochs for Supervised Learning of Two-Spirals. The first part of the table is taken from J. Bruske [5].

examples each. Gorman and Sejnowski 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 neighbour classifier.

In our simulation the accumulated squared output error served as resource updating: $\Delta\mathcal{E}_{bmu} = ||\zeta - \mathbf{o}||^2, \zeta, \mathbf{o} \in \{-1, 1\}$. We obtained 92.31% correctly classified test samples with only 78 epochs and 147 grown cells in the MGCS2 network. At the same time, the classification error on the training set dropped to 0. For MGCS3 network the accuracy in the testing set reached 95.19% after 106 epochs and 315 cells while the classification error on the training set dropped also to 0. By comparison, after 140 epochs the SGCS reached 90.38% accuracy for test samples with 141 grown cells. At the same time, the classification error on the training set fell to 3.85%. Supervised DCS-GCS of Bruske reached an accuracy of 91.4% in test samples after 95 epochs of training.

Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Training Set CR	Testing Set CR
KNN					82.7%
MLP+BP					90.4%
DCS-GCS	95				91.4%
SGCS	140	30.34	141	96.15%	90.38%
MGCS2	78	17.36	147	100%	92.31%
MGCS3	106	46.95	315	100%	95.19%

Table 2. Comparison of MGCS and SGCS for the sonar–mine/rock separation problem. The first part of the table is taken from J. Bruske [5]. KNN — K-Nearest Neighbour Classifier. CR — Classification Rate.

Table 2 shows the difference between SGCS and MGCS for this problem.

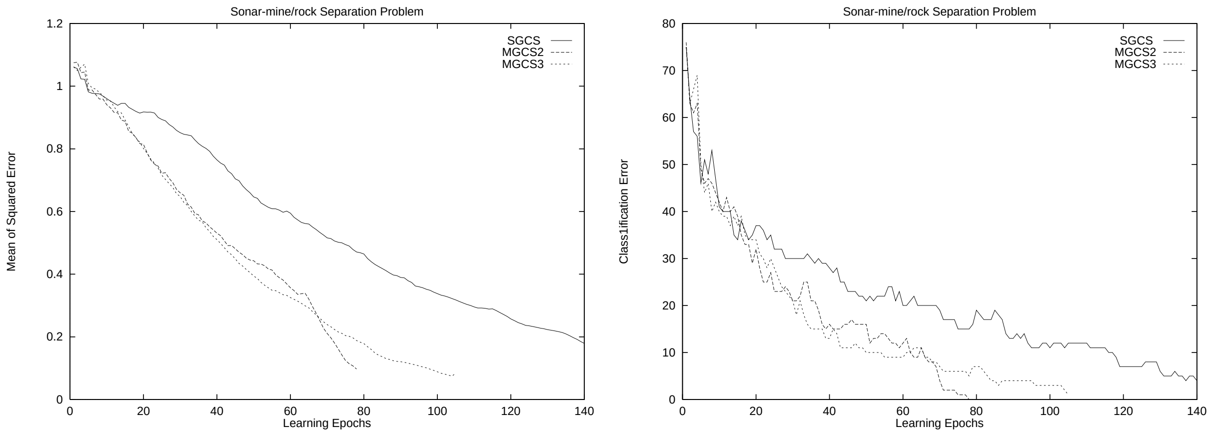

Figure 6. Performace comparison of MGCS and SGCS for the sonar–mine/rock separation problem. Simulation parameters: $\lambda = 104\ (number\ of\ patterns), \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.001, \beta = 0.5, \eta = 0.15$, no removal of cells. *left:* Mean squared error vs. epochs. *right:* Classification error vs. epochs.

Fig. 6 shows the curve of the mean squared error and classification error vs. learning epochs for the sonar–mine/rock separation problem.

4.3. Glass Identification Problem

The benchmarks — glass identification, horse colic fate prediction, soybean disease recognition and credit card approval prediction — are taken from the PROBEN1 benchmark set [23]. For the application of the validation method the datasets were divided into three sets: training set, validation set and test set. The different sets are built from three different partitions of the dataset: the training set consists of 50% of the data in the dataset, the validation set and the test set consist of 25% of dataset each. This partitioning is applied in three different orders to the whole dataset and this leads to three different training, validation and test sets: glass1, glass2 and glass3.

The glass benchmark is based on the description of glass splinters. The description of the glass comprises the chemical analysis (eight different elements) and the refractive index. The classification task is motivated by forensic needs in criminal investigation in order to decide whether the glass is either float processed or not float processed building window, vehicle windows, containers, tableware, or head lamps.

The glass benchmark dataset consists of 9 inputs, 6 outputs and 214 samples. All inputs are continuous, two of them have hardly any correlation with the result. For this simulation, the six classes were coded on 6 output neurons, each class being associated with one neuron. The accumulated squared output error served as resource updating: $\Delta\mathcal{E}_{bmu} = ||\zeta - \mathbf{o}||^2, \zeta, \mathbf{o} \in \{0, 1\}$. The glass dataset with 107 examples used for the training set, 54 examples used for the validation set. 53 examples used for the testing set.

For controlling the training process the early stopping method was used [24]. The criterion can be expressed as: stop training as soon as the error on the validation set is higher than it was the last time it was checked.

Database	Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Training Set CR	Validation Set CR	Testing Set CR
Glass1	SGCS	40	1.79	62	66.36%	68.52%	64.15%
	MGCS2	28	1.21	75	66.36%	62.96%	66.04%
	MGCS3	15	0.65	67	64.49%	64.82%	62.26%
Glass2	SGCS	45	1.86	70	74.77%	72.22%	58.49%
	MGCS2	25	0.96	66	76.64%	64.82%	58.49%
	MGCS3	15	0.63	67	65.42%	55.56%	47.17%
Glass3	SGCS	148	21.25	223	100%	100%	69.81%
	MGCS2	89	11.37	251	100%	100%	66.04%
	MGCS3	60	8.86	268	94.39%	98.15%	60.38%

Table 3. Comparison of MGCS and SGCS for the glass identification problem. Simulation parameters: $\lambda = 107\ (number\ of\ patterns), \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.005, \beta = 1.0, \eta = 0.15$, no removal of cells. CR — Classification Rate.

Table 3 shows the difference between SGCS and MGCS for this problem. The generalization performance is estimated by the error with respect to the testing set. Although the number of cells required by MGCS2 is slightly more than SGCS, the learning epochs and CPU time required by MGCS are much less than SGCS. The generalization performance of MGCS is better than SGCS. For example, classification rate is increased from 64.15% to 66.04% for glass1 dataset. For glass2 dataset, in the same classification rate (58.49%), MGCS2 took only 25 epochs (CPU: 0.96 sec.) while SGCS needed 45 epochs (CPU: 1.86 sec.).

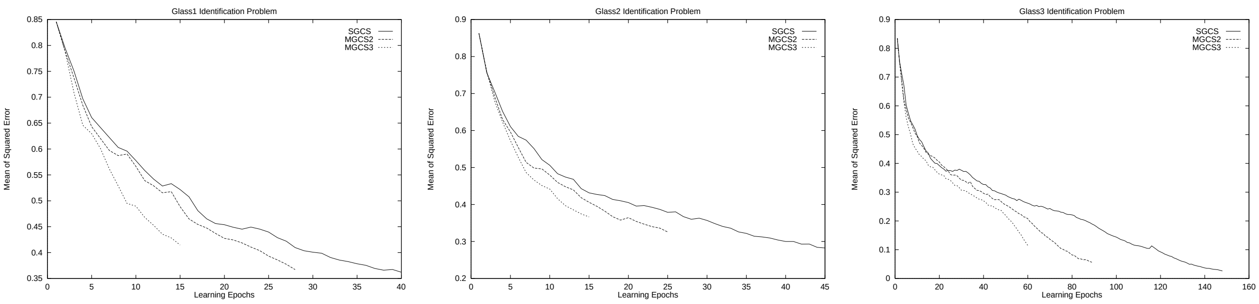

Figure 7. Performace comparison of MGCS and SGCS for the glass identification problem. Mean squared error vs. epochs. *left*: Glass1 dataset. *middle*: Glass2 dataset. *right*: Glass3 dataset.

Fig. 7 shows the curve of the mean squared error and classification error vs. learning epochs for the glass identification problem.

4.4. Horse Colic Problem

This benchmark problem concerns the fate of a horse that has a colic. The results of a veterinary examination of a horse having colic are used to predict whether the horse will survive, will die, or will be euthanized.

The horse benchmark dataset consists of 58 inputs, 3 outputs and 364 samples. In 62% of the examples the horse survived, in 24% it died, and in 14% it was euthanized. This problem has very many missing values (about 30% overall of the original attribute values), which are all represented as missing explicitly using additional inputs. For this simulation, the three classes were coded on 3 outputs neurons, each class being associated with one neuron. The accumulated squared output error served as resource updating: $\Delta\mathcal{E}_{bmu} = \|\zeta - \mathbf{o}\|^2, \zeta, \mathbf{o} \in \{0, 1\}$. The horse dataset with 182 examples used for the training set, 91 examples used for the validation set and the testing set.

Database	Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Training Set CR	Validation Set CR	Testing Set CR
Horse1	SGCS	200	67.39	201	86.81%	50.55%	57.14%
	MGCS2	100	35	188	83.52%	54.95%	56.04%
	MGCS3	70	23.02	208	80.77%	52.75%	53.85%
Horse2	SGCS	240	113.19	241	89.01%	40.66%	52.75%
	MGCS2	120	51.73	234	96.15%	48.35%	53.85%
	MGCS3	80	37.13	238	96.15%	46.15%	49.45%
Horse3	SGCS	175	62.66	177	75.27%	57.14%	43.96%
	MGCS2	85	22.65	158	78.02%	54.95%	42.86%
	MGCS3	60	21.31	178	73.63%	52.74%	46.15%

Table 4. Comparison of MGCS and SGCS for the horse colic problem. Simulation parameters: $\lambda = 182\ (number\ of\ patterns), \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.005, \beta = 1.0, \eta = 0.15$, no removal of cells. CR — Classification Rate.

Table 4 shows the difference between SGCS and MGCS for this problem. It is interesting to note that the number of learning epochs required by MGCS2 is two times samller than the SGCS, while the classification rate is nearly same. Moreover, the number of growing cells required by MGCS2 is slightly smaller than SGCS. The CPU time needed by MGCS is much smaller than SGCS.

Fig. 8 shows the curve of the mean squared error and classification error vs. learning epochs for the horse colic problem.

4.5. Soybean Disease Recognition Problem

This benchmark problem concerns the recognition of 19 different diseases of soybeans. The discrimination is done based on a description of the bean (e.g. whether its size and color are normal) and the plant (e. g. the size of spots on the leafs, whether these spots have a halo, whether plant growth is normal, whether roots are rotted) plus information about the history of the plant's life (e. g. whether changes

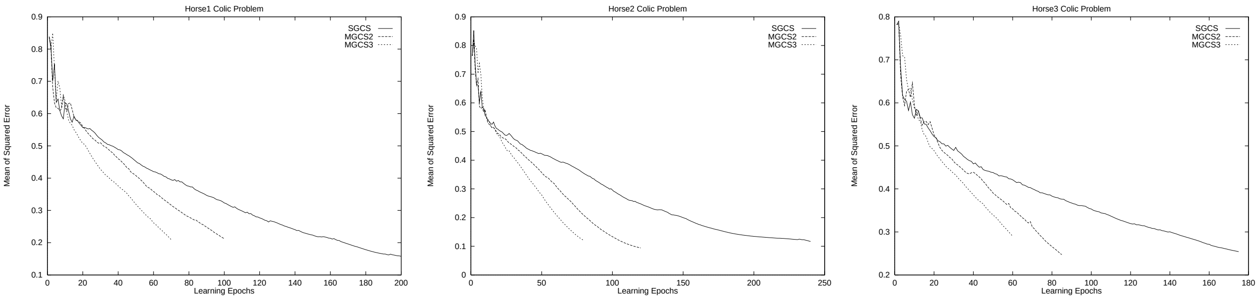

Figure 8. Performace comparison of MGCS and SGCS for the horse colic problem. Mean squared error vs. epochs. *left*: Horse1 dataset. *middle*: Horse2 dataset. *right*: Horse3 dataset.

in crop occurred in the last year or last two years, whether seeds were treated, how the environment temperature is).

The soybean benchmark dataset consists of 82 inputs, 19 outputs and 683 samples. This is the problem with the highest number of classes in the PROBEN1 benchmark set. Most attributes have a significant number of missing values. The problem has been used often in the machine learning literature, although with several different datasets, making comparisons difficult. Most of the past uses use only 15 of the 19 classes, because the other four have only few instances. For this simulation, the 19 classes were coded on 19 outputs neurons, each class being associated with a neuron. The accumulated squared output error served as resource updating: $\Delta\mathcal{E}_{bmu} = ||\zeta - \mathbf{o}||^2, \zeta, \mathbf{o} \in \{0, 1\}$. The soybean dataset with 342 examples used for the training set, 171 examples used for the validation set. 170 examples used for the testing set.

Database	Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Training Set CR	Validation Set CR	Testing Set CR
Soybean1	SGCS	90	56.55	92	92.40%	86.55%	82.35%
	MGCS2	70	55.53	134	94.15%	90.64%	85.88%
	MGCS3	50	50.81	150	94.15%	88.89%	84.71%
Soybean2	SGCS	100	61.73	102	90.06%	88.89%	75.29%
	MGCS2	80	81.2	152	91.52%	88.89%	78.82%
	MGCS3	60	69.24	180	94.74%	88.89%	78.82%
Soybean3	SGCS	110	71.53	112	93.57%	87.72%	82.35%
	MGCS2	70	58.39	125	94.74%	88.89%	84.71%
	MGCS3	40	30.43	120	91.52%	83.63%	82.35%

Table 5. Comparison of MGCS and SGCS for the soybean disease recognition problem. Simulation parameters: $\lambda = 342\ (number\ of\ patterns), \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.005, \beta = 1.0, \eta = 0.1$, no removal of cells. CR — Classification Rate.

Table 5 shows the difference between SGCS and MGCS for this problem. The generalization performance is estimated by the error with respect to the testing set. The generalization performance of MGCS is better than SGCS for this problem. For example, classification rate is increased from 82.35% (SGCS) to 85.88% (MGCS2) and 84.71% (MGCS3) for soybean1 dataset, while the learning epochs are 90, 70 and 50 resp.

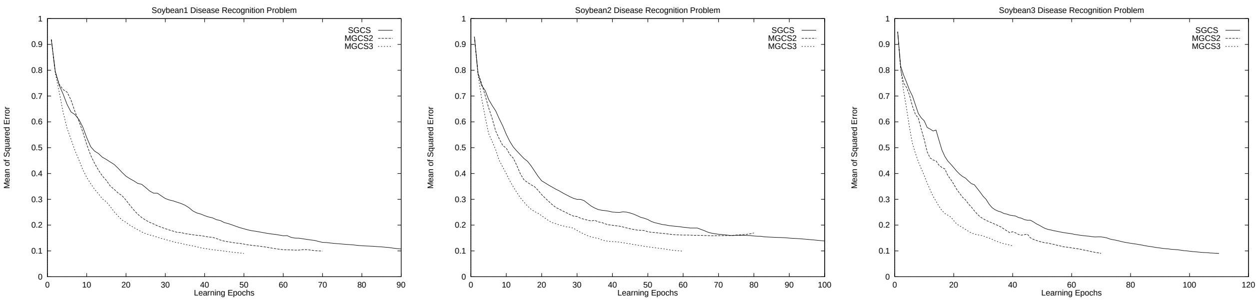

Figure 9. Performace comparison of MGCS and SGCS for the soybean disease recognition problem. Mean squared error vs. epochs. *left*: Soybean1 dataset. *middle*: Soybean2 dataset. *right*: Soybean3 dataset.

Fig. 9 shows the curve of the mean squared error and classification error vs. learning epochs for the soybean disease recognition problem.

4.6. Credit Card Approval Prediction

This problem predicts the approval or non-approval of a credit card to a customer. Each example represents a real credit card application and the output describes whether the bank (or similar institution) granted the credit card or not. The meaning of the individual attributes is unexplained for confidence reasons.

The credit card benchmark dataset consists of 51 inputs, 2 outputs and 690 samples. This dataset has a good mix of attributes: continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values in 5% of the examples. 44% of the examples are positive. For this simulation, the 2 classes were coded on 2 outputs neurons, each class being associated with one neuron. The accumulated squared output error served as resource updating: $\Delta\mathcal{E}_{bmu} = ||\zeta - \mathbf{o}||^2, \zeta, \mathbf{o} \in \{0,1\}$. The card dataset with 345 examples used for the training set, 173 examples used for the validation set. 172 examples used for the testing set.

Table 6 shows the difference between SGCS and MGCS for this problem. As it can be seen that the learning epochs required by MGCS2 and MGCS3 are two and three times smaller than the SGCS, while the classification rate is nearly same. Moreover, the CPU time required by MGCS is also much less than SGCS. For example, the CPU times required by SGCS, MGCS2 and MGCS3 are 33.84, 13.92 and 10.25 seconds for card2 dataset, while the classification rate on testing set are 80.23%, 80.23% and 81.40% resp.

Fig. 10 shows the curve of the mean squared error and classification error vs. learning epochs for the card identification problem.

Database	Algorithm	Epochs	CPU Time (Seconds)	Growing Cells	Training Set CR	Validation Set CR	Testing Set CR
Card1	SGCS	80	22.57	83	87.54%	89.02%	84.30%
	MGCS2	40	10.76	75	86.38%	86.71%	83.72%
	MGCS3	30	10.83	93	85.22%	87.86%	84.88%
Card2	SGCS	100	33.84	103	90.72%	82.66%	80.23%
	MGCS2	45	13.92	85	89.28%	84.97%	80.23%
	MGCS3	30	10.25	93	88.12%	83.82%	81.40%
Card3	SGCS	150	69.86	153	87.54%	86.13%	75.58%
	MGCS2	75	37.44	146	88.12%	86.71%	76.74%
	MGCS3	50	23.12	153	85.51%	86.13%	72.67%

Table 6. Comparison of MGCS and SGCS for the credit card approval prediction problem. Simulation parameters: $\lambda = 345\ (number\ of\ patterns), \varepsilon_{bmu} = 0.1, \varepsilon_{dtn} = 0.006, k = 2, \tilde{t} = 2, \alpha = 0.005, \beta = 1.0, \eta = 0.15$, no removal of cells. CR — Classification Rate.

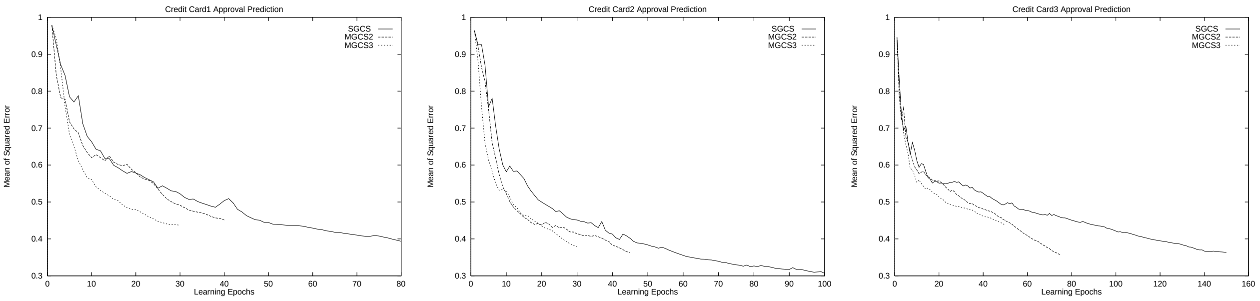

Figure 10. Performace comparison of MGCS and SGCS for the credit card approval prediction problem. Mean squared error vs. epochs. *left*: Credit card1 dataset. *middle*: Credit card2 dataset. *right*: Credit card3 dataset.

5. Conclusions

In this paper we have developed a novel variant of Fritzke's GCS network. The main advantage of GCS is that it can automatically find a network structure and size suitable for a given classification pattern through cell growing. The MGCS algorithm can speed up the convergence of the learning process. The position of cell insertion is calculated according to the ratio of the resource values of two end-nodes. Comparative experimental results on the six benchmark datasets revealed that MGCS is both effective and efficient as a learning algorithm for classification problems by multiple grown cells. The required epochs in MGCS are only nearly half of epochs required by GCS. The number of cells required for MGCS and GCS are nearly equal. MGCS is particularly well-suited for data sets which contain low dimension and less numbers of pattern like the spiral problem. For classification problems with many clusters (e.g. soybean disease discrimination) it also showed a good performance. The MGCS can dynamically produce a self-organizing network with smaller size and better generalization performance and rapid convergence capacity. In order to further reduce the number of growing cells we can set a threshold θ for the second new cell. e. g., when $\mathcal{E}_{max1} - \mathcal{E}_{max2} \leqslant \theta\mathcal{E}_{max1}$ is satisfied, we can only insert the new cell $\mathbf{v}_{new2}$ between $\mathbf{v}_{max2}$ and $\mathbf{v}_{mdn2}$. Our future projects include further improvement of MGCS and its application in the fields of function approximation and time series prediction. Multiple Growing Neural Gas (MGNG) and Growing Cell Network with double winner cells adaptation are also planned.

Acknowledgments

We would like to thank B. Fritzke, M. Kunze, F. Hamker, J. Bruske and L. Prechelt for valuable and constructive discussions about their growing neural network algorithm and neural network benchmark problems. We also wish to thank our colleagues Igor Fischer, Jürgen Wakunda, Fred Rapp and Jutta Huhse for their help in programming and proof reading. The first author also thanks German KAAD (Katholischer Akademischer Ausländer-Dienst) very much for the scholarship.

References

[1] A. G. E. Blanzieri and P. Katemkamp. Growing radial basis function networks. In *Proc. of the 4th Workshop on Learning Robots*, Karlsruhe, Germany, Dec. 1995.

[2] C.-A. Bohn. Efficiently representing the radiosity kernel through learning. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, pages 123–132, New York City, NY, June 1996. Eurographics, Springer Wien. ISBN 3-211-82883-4.

[3] C.-A. Bohn. Finite element mesh generation using growing cell structures networks. In *EANN97: International Conference on Engineering Applications of Neural Networks*, Stockholm, 1997.

[4] C.-A. Bohn. An incremental unsupervised learning scheme for function approximation. In *Proceedings of the 1997 IEEE International Conference on Neural Networks*, pages 1792–1797, Piscataway, NJ, June 1997. IEEE Service Center.

[5] J. Bruske and G. Sommer. Dynamic cell structures. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 497–504. The MIT Press, 1995.

[6] V. Burzevski and C. K. Mohan. Hierarchical growing cell structures. In *ICNN96: Proc. Int'l. Conf. Neural Networks*, Washington D.C., June 1996. Tech. Rep., Syracuse University.

[7] C. K. M. C. Tumuluri and A. Choudhary. Unsupervised algorithms for learning emergent spatio-temporal correlations. In *Technical Report CIS-96-01*, Syracuse University, 1996.

[8] J. Dopazo, H. Wang, and J. M. Carazo. A new type of unsupervised growing neural network for biological sequence classification that adopts the topology of a phylogenetic tree. *Lecture Notes in Computer Science*, 1240:P932, 1997.

[9] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2. Proceedings of the 1989 Conference*, pages 524–532, San Mateo, CA, 1990. Morgan Kaufmann.

[10] B. Fritzke. Growing cell structures – a self-organizing network in k dimensions. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks II*, pages 1051–1056. North-Holland, Amsterdam, 1992.

[11] B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. TR-93-026, International Computer Science Institute, Berkeley, 1993.

[12] B. Fritzke. Kohonen feature maps and growing cell structures – a performance comparison. In L. Giles, S. Hanson, and J. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 123–130. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

Table 7. Table of Symbols

MRV,MEV	Maximum Resource Vertex or Maximum Error Vertex
$\mathcal{A}$	Growing Cell Structures network, also denotes set of cells in the network
$\mathcal{V}$	n-dimensional input vector space
k	dimensionality of the Growing Cell Structures network $\mathcal{A}$
n	dimensionality of $\mathcal{V}$
m	dimension of the output vector space for supervised learning
$\mathbf{v}$	set of all reference vectors for cells in $\mathcal{A}$
$\mathbf{v}_c$	n-dimensional reference (synaptic, weight) vector of cell c
$\mathbf{w}$	set of all output weights
$\mathbf{w}_c$	m-dimensional vector of weights from cell c to the output units
$\mathbf{w}_{jc}$	weighted connection from Gaussian unit c to output unit j
V_{max}	Vertex of maximum error or maximum resource
V_{max1}	Vertex of the first maximum error
V_{max2}	Vertex of the second maximum error
$V^{'}_{max2}$	Vertex of maximum error in the neighbour of V_{max}
V_{mdn}	Vertex of most distance neighbour
V_{dtn}	Vertex of direct topology neighbour
V_{new}	Vertex of new grown cell
$\mathbf{v}_{max}$	weight vector for V_{max} (others: $\mathbf{v}_{mdn}$ vs. V_{mdn}, $\mathbf{v}_{dtn}$ vs. V_{dtn}, $\mathbf{v}_{new}$ vs. V_{new})
$\mathbf{v}_{max1}$	weight vector for V_{max1} (others: $\mathbf{v}_{max2}$ vs. V_{max2}, $\mathbf{v}^{'}_{max2}$ vs. $V^{'}_{max2}$)
h_c	relative signal frequency of cell c
τ_c	signal counter of cell c
$\mathcal{E}_c$	error counter (quantization error, classification error) of cell c
$\mathcal{E}_{max}$	error counter for V_{max} (others: $\mathcal{E}_{mdn}$ vs. V_{mdn}, $\mathcal{E}_{dtn}$ vs. V_{dtn}, $\mathcal{E}_{new}$ vs. V_{new})
λ	number of adaptation steps per insertion
ε_{bmu}	adaptation parameter for best-matching unit
ε_{dtn}	adaptation parameter for direct topology neighbouring cells
η	learning rate for output weights
α	decrease parameter for resource variables
β	Gaussian kernel diminution factor
σ_c	Gaussian function kernel width
$\mathcal{N}_c$	set of direct neighbours of a cell c
$P(\xi)$	probability distribution of input signals
ξ	n-dimensional input signal
ζ	m-dimensional output signal
$d_c(\xi)$	activation of Gaussian unit c
$\Delta y = x$	short-cut for $y^{new} = y^{old} + x$
$\mathbf{o}$	m-dimensional vector of output layer activations
t	true data dimensionality
$\tilde{t}$	estimate for t
$\bar{l}_c$	mean length of the edges emating from cell c
θ	growing threshold for multiple growing cell structure
F_c	Voronoi field of cell c
$\\|F_c\\|$	n-dimensional volume of F_c
f_{va}	Voronoi field diminution factor
f_{in}	cell insertion position factor

[13] B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[14] B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[15] B. Fritzke and P. Wilke. Flexmap – a neural network with linear time and space complexity for the traveling salesman problem. In *Proc. of IJCNN-91*, pages 929–934, Singapore, 1991.

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

[17] F. Hamker and D. Heinke. Implementation and comparison of growing neural gas, growing cell structures and fuzzy artmap. In *Technical report, Schriftenreihe des FG Neuroinformatik der TU Ilmenau*. Report 1/97, ISSN 0945-7518, 1997.

[18] M. Köhle and D. Merkl. Visualizing similarities in high dimensional input spaces with a growing and splitting neural network. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Proceedings of ICANN96, International Conference on Artificial Neural Networks, Bochum, Germany, July 16-19, 1996*, Lecture Notes in Computer Science, vol. 1112, pages 581–586. Springer, Berlin, 1996.

[19] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).

[20] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Proc. of the 1988 Connectionist Summer School*. Morgan Kaufmann, 1988.

[21] D. G. Lowe. Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7(1):72–85, 1995.

[22] E. Mael. A hierarchical network for learning robust models of kinematic chains. *Lecture Notes in Computer Science*, 1112:P617, 1996.

[23] L. Prechelt. PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, Sept. 1994. ftp://ftp.ira.uka.de/pub/papers/techreports/1994/1994-21.ps.Z and ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz.

[24] L. Prechelt. Early stopping — but when? *Lecture Notes in Computer Science*, 1524:P55, 1998.

[25] U. Seiffert and B. Michaelis. Growing 3D-SOMs with 2D-input layer as a classification tool in a motion detection system. *International Journal of Neural Systems*, 8(1):81–9, 1997.

[26] H.-H. Song and S.-W. Lee. A self-organizing neural tree for large-set pattern classification. *Journal of KISS[B] [Software and Applications]*, 24(4):422–31, 1997.

[27] J. Steffens and M. Kunze. Implementation of the supervised growing cell structure on the CNAPS neurocomputer. In F. Fogelman-Soulie and P. Gallinari, editors, *ICANN '95. International Conference on Artificial Neural Networks. Neuronimes '95 Scientific Conference*, volume 2, pages 51–6, Paris, France, 1995.

[28] G. v. Wichert. Selforganizing visual perception for mobile robot navigation. In *1st Euromicro Workshop on Advanced Mobile Robots (EUROBOT'96)*, pages 194–200. IEEE Computer Society Press, 1996.

[29] N. A. Vlassis, A. Dimopoulos, and G. Papakonstantinou. The probabilistic growing cell structures algorithm. *Lecture Notes in Computer Science*, 1327:P649, 1997.

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

[31] M. White and S.Fahlman. Cmu repository of neural network benchmarks. Technical report, The Carnegie Mellon University, 1997.

[32] J. Zavrel. Neural navigation interfaces for information retrieval: are they more than an appealing idea? *Artificial Intelligence Review*, 10(5-6):477–504, 1996.

[33] U. R. Zimmer, C. Fischer, and E. von Puttkamer. Navigation on topologic feature-maps. In *Proc. 3rd Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, pages 131–132, Iizuka, Japan, 1994. Fuzzy Logic Systems Institute.