Syracuse University
SURFACE

College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects

College of Engineering and Computer Science

1996

Hierarchical Growing Cell Structures

Vanco Burzevski
Syracuse University

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

Follow this and additional works at: http://surface.syr.edu/lcsmith_other

Part of the Computer Sciences Commons

Recommended Citation
Burzevski, Vanco and Mohan, Chilukuri K., "Hierarchical Growing Cell Structures" (1996). *College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects.* Paper 22.
http://surface.syr.edu/lcsmith_other/22

This Article is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

Hierarchical Growing Cell Structures

(Revised version of the paper appearing in Proc. IEEE ICNN'96.)

Vanco Burzevski and Chilukuri K. Mohan
2-120 CST, School of CIS, Syracuse University, Syracuse, NY 13244-4100, U.S.A.
315-443-2368/FAX:1122, vanco/mohan@top.cis.syr.edu

ABSTRACT

We propose a hierarchical self-organizing neural network ("HiGS") with adaptive architecture and simple topological organization. This network combines features of Fritzke's *Growing Cell Structures* and traditional hierarchical clustering algorithms. The height and width of the tree structure depend on the user-specified level of error desired, and the weights in upper layers of the network do not change in later phases of the learning algorithm. Parameters such as node deletion rate are adaptively modified by the learning algorithm.

1. Introduction

Connectionist learning systems often face the *stability-plasticity* dilemma [2]. Most unsupervised neural network learning algorithms are stable with respect to their topology and plastic with respect to weight vector adaptations; such is the case in Kohonen's topology-preserving self-organizing map (SOM) [4]. An exception is Fritzke's *Growing Cell Structures* (GCS) network [1], which is much more plastic in that nodes may be inserted into or removed from the network during the learning process. However, extreme plasticity can have unfortunate side-effects since useful accumulated learning experience can be lost. For instance, the deletion of a node can be followed by massive purges in the GCS network, partly because the learning algorithm attempts to maintain a network that can be viewed as a set of hyper-tetrahedra (triangles, for 2D topology).

A little more stability would be useful: this is accomplished using the hierarchical GCS algorithm proposed in this paper, in which weights of upper layer nodes are not modified after sub-trees are spawned. Furthermore, the network automatically adapts the frequency of node deletions invoked, depending on the success or failure of previous attempts to delete nodes. This mechanism eliminates one crucial user-specified parameter of the GCS network, the number of sample presentations after which node deletions are invoked.

Hierarchical clustering algorithms have been known for a long time, e.g., see Jain and Dubes (1988) [3] for a review. Upper layers of the network accomplish high-level clustering using a small number of nodes (often just two or three) at each layer, and children of each node focus attention entirely on the input samples for which the parent is the nearest among nodes at that layer (or *winner*). The traditional clustering algorithm frequently invoked (at each layer) is analogous to simple competitive learning, so that the final results may be far from optimal; better results may be obtainable using a self-organized neural network algorithm in which more than one node is updated when a sample is presented. This motivates our hierarchical GCS algorithm.

Lampinen and Oja [5] have proposed a two-layer network with a hierarchical structure, generalizing Kohonen's SOM. When an input vector is presented to the network, the *best matching*

unit (winner) from the first layer unit is first determined, and its children in the hierarchy are then examined to determine the final winner (a second layer node). As in the case of other hierarchical algorithms, this embeds a useful successive refinement strategy, but network sizes and topology are fixed in each SOM layer, restricting the ease with which the network can be adapted to fit the problem. By comparison, the hierarchical GCS algorithm presented here may have more than two layers, as determined by problem complexity, is more plastic in the sense that the network structure adapts with input sample presentation, but is also more stable in the sense that the higher layer nodes are not adapted in later stages of learning.

Section 2 describes the GCS algorithm. The hierarchical GCS algorithm is described in Section 3. Results are given in Section 4, followed by conclusions.

2. Growing Cell Structures

Fritzke's GCS network has a topology consisting of many hyper-tetrahedra (triangles in the two-dimensional case), whose vertices (nodes) are associated with weight vectors in the input vector space. Each node has a few neighbors, adjacent to it in the topological structure of the network. Periodically, nodes are inserted into and deleted from the GCS network; this occurs at intervals of a fixed number of sample presentations. There are three main aspects of the GCS learning algorithm:

Adaptation: When an input pattern is presented to the GCS network, a competition is conducted, and the weight vectors of each node as well as its immediate topological neighbors are adapted in the direction of the input pattern, albeit with different learning rates. Each node also has an associated *signal counter* τ_j that estimates the number of input patterns for which that node j was the winner. Counter values also decay with time.

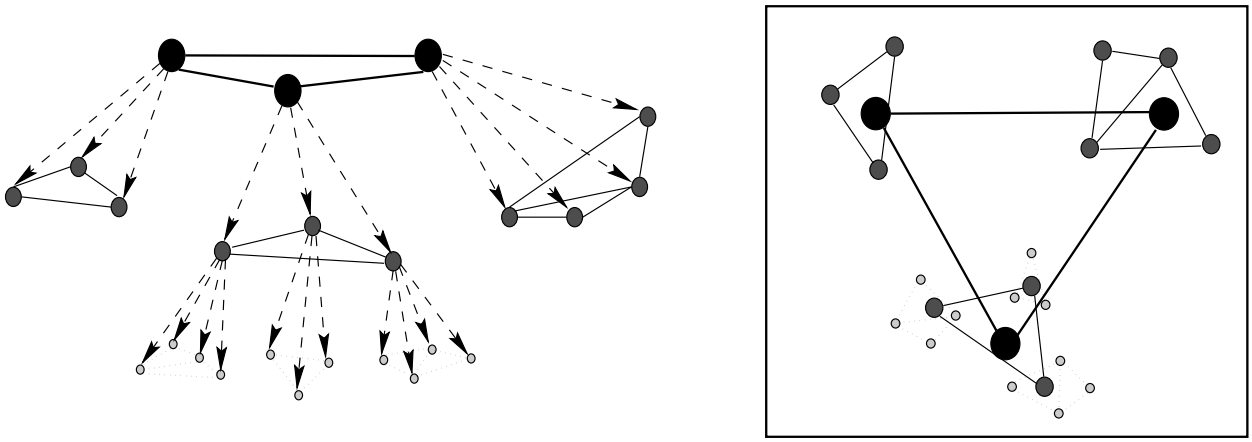
Insertion: A node is inserted halfway between the node with highest signal counter value and its topologically adjacent neighbor which is at the greatest Euclidean distance. Insertion is followed by the establishment of connections to existing nodes, so that the triangular (or hyper-tetrahedral) network structure continues to be maintained. Voronoi regions and signal counters of adjacent nodes are then adjusted as appropriate.

Deletion: Nodes in a GCS with least signal counter value are chosen for deletion. Removal of a node entails removal of the connecting edges, which may leave some nodes "dangling," i.e., not part of any triangle (or hyper-tetrahedron in the general case). This eventuality is unthinkable, hence the dangling nodes are also deleted, possibly making other nodes dangle. Repeated elimination of dangling nodes may result in massive purges of the network, erasing the effect of what has been learned earlier, necessitating that the network must rebuild itself. It is possible for subsequent insertions to result in a similar structure, implying that the network may repeatedly shrink and expand, cycling through similar states. In practice, randomness in input presentations may help the algorithm escape such cycles.

Fritzke [1] has described several examples where the performance of the GCS algorithm is superior to that of the SOM algorithm with non-adaptive topology. The deletion mechanism allows GCS to approximate each region accurately, whereas the SOM tends to place several nodes in between these regions, as a consequence of the fixed topological structure.

3. Hierarchical Growing Cell Structures

The Hierarchical GCS (HiGS) network structure resembles a tree, in which each element is a GCS network, as shown in Figure 1. Every non-root GCS network in the tree corresponds to one node of the parent network. All the components of the training process, i.e., weight adaptations, node insertions, and node deletions, occur only at the lowest levels (leaves) of the network structure.



(a) (b)

Fig. 1: A Hierarchical Growing Cell Structures network: darkly shaded circles indicate nodes at higher layers, and non-arrow lines indicate topological connections in each sub-network. (a) The hierarchy of nodes; arrows go from parents to children. (b) Location of weight vectors of nodes in the data space.

Input samples are repeatedly presented to the network, and weight vectors of nodes adapted as described below. The network training proceeds in iterations, where an iteration is defined in terms of the number of presentations of input samples [1]. To ensure that each network in the HiGS network structure is a winner for a sufficient number of input samples, the number of presentations per iteration is defined as a multiple of the number of nodes in the leaf networks. This ensures that each leaf network gets a chance to adapt sufficiently, thus avoiding excess deletions.

After each iteration, the algorithm either attempts to insert some nodes in the network, or attempts to delete some nodes. Before inserting a node in a sub-network S in the hierarchy, however, the algorithm compares the signal counter values of all the nodes in S; if these values are roughly equal then a process called "splitting" occurs instead of node insertion, with lower level sub-networks being spawned below each node in S. The rest of this section details the exact behavior of HiGS in adapting, inserting, splitting and deleting nodes.

Adaptation: When an input sample is presented to the network, a competition is first conducted among the nodes at the highest layer of the network. There are two cases to be considered:

1. If the winner is a leaf node, its weight vector is adapted in the direction of the input

[1] For a fixed, finite training set, input samples are presented in random order. If the training data is infinite or comes from sampling a steady stream of inputs, there is no need to randomize samples.

sample:

$$\Delta w_{\text{winner}} = (\eta_w)(\text{input sample}), \text{ where } \eta_w \text{ is a learning rate constant.}$$

The weight vectors of nodes topologically adjacent to it (in the same layer) are also adapted in the same direction, using a smaller learning rate:

$$\Delta w_{\text{winner's neighbor}} = (\eta_n)(\text{input sample}), \text{ where } \eta_n < \eta_w \text{ is another learning rate.}$$

2. If the winner is associated with a sub-network, the competition is instead conducted among the nodes of this sub-network. The adaptation algorithm is recursive, with winners at successively lower levels of the hierarchy being determined, until a leaf node is reached. Only leaf nodes can be adapted.

The learning rate of the winner node (η_w) is a user-specified parameter. The learning rate of its topological neighbors (η_n), called the neighborhood learning rate, is an adaptive parameter. The adaptation is on a per-network basis, i.e. different networks in the tree of networks may have different neighborhood learning rates. To be more precise, the neighborhood learning rate depends upon the number of nodes in that particular network. The adaptation rule is chosen so that when the number of nodes in the network is minimal, the neighborhood learning rate is equal to one fifth of the winner's learning rate, and as the network size grows, the neighborhood learning rate approaches the winner's learning rate exponentially.

The equation used to adapt the neighborhood learning rate is:

$$\eta_n = \eta_w/(1 + 4e^{-(N-(I+1))})$$

where I is the input dimensionality, and N is the number of nodes in the network

The rationale behind the choice of an adaptive learning rate is that it is very useful when dealing with an input distribution that consists of multiple disjoint clusters. When the network is small, the nodes are "encouraged" to migrate towards different clusters, but without moving their neighbors too much. As the network gets larger, presumably the nodes have had sufficient time to migrate towards different input clusters. Since new nodes are inserted halfway between already existing nodes, they have a tendency to appear between the input clusters. Hence, the newly inserted nodes will rarely be winners, and will be attracted towards the existing clusters very slowly. Therefore, the probability is low that the newly inserted nodes will move to an input cluster and become winners for some sample presentations. So, if the network is considered for deletion, the newly inserted node will be deleted, and as a side effect, probably cause the deletion of its neighbors as well. So, if the newly inserted nodes are allowed to move more rapidly towards the input clusters, they have a larger probability to migrate to some input cluster before network deletion occurs.

<u>Insertion</u>: Node insertion is attempted after every second iteration. As mentioned earlier, whenever node insertion is attempted, the signal counter values of all nodes are compared. If these values are approximately equal, the HiGS algorithm chooses not to insert a node into the network, but instead performs splitting, generating sub-networks below every node in the network.

Each non-leaf node is associated with an error measure that estimates the quality of the sub-network below it, based on the average Euclidean distance between each sample and the

weight vector of the winner (leaf) node, for cases in which the winner lies in this sub-network. This error is used to determine where insertion occurs: a node is always inserted into the sub-network that caused the largest error.

Insertion is attempted first at the root level in the hierarchy. The node that caused the largest error is determined, and the new node is inserted into the sub-network below this node (if such a sub-network exists). This process is repeated at each level, successively determining the lower level sub-network (with highest error) into which node insertion is to occur. Finally, when a leaf sub-network is reached, nodes are inserted in the same manner as in GCS, i.e. the weight vector associated with the new node is set to the mid-point of weight vectors of the node with the largest signal counter value and its farthest topological neighbor.

To speed up the convergence of the algorithm, nodes can also be inserted into other sub-networks. The algorithm compares the generated error of each sub-network with the largest generated error, and if their ratio exceeds a cut-off point, a node is inserted with the same probability as this ratio (of the errors). This cut-off point is introduced to prevent excessive node insertions in networks that have already generated small enough errors, in order to avoid slowing all the operations over the network. The cut-off point in our experiments is chosen to be 0.8, an empirically determined value that gave satisfactory results.

Splitting: This is a process in which new (lower level) sub-networks are generated. Splitting is invoked as a special case of node insertion in a leaf network.

Splitting occurs when attempting to insert a node in a leaf network, if the signal counters of the nodes in the network have roughly equal values. A network S is split by generating sub-networks of the minimal possible size for each node in the network, and placing each sub-network in the Voronoi region of its parent node. The nodes of a sub-network S' are placed within the Voronoi region of the parent node X, by ensuring that

$$\text{distance}(X, X') < \min[\text{distance}(X, \text{ neighbors of } X \in S)]/2, \text{ for all } X' \text{ in } S'.$$

The similarity criterion (used to determine whether splitting should be conducted) is based on the ratio of the maximum and minimum values of the signal counters of the nodes in the network S. If this ratio is below a certain threshold, the network is split. Our experiments used threshold values of

$$(1.2) * (1.5)^{(\text{number of nodes in sub-network})-(1+\text{ input space dimensionality})}$$

for every network in the HiGS network structure. Satisfactory results were obtained with these empirically determined values.

Deletion: Deletion is attempted only at leaf networks. Nodes with low signal counter values (i.e., nodes that are winners for only a small fraction of input samples) may be deleted after some number of iterations. However, it is difficult to determine *a priori* how often node deletion should be attempted: frequent deletion can lead to instability, and repeated insertion with infrequent deletion can lead to the growth of excessively large networks. Our algorithm adapts and automatically determines the appropriate value for the deletion probability; a different value is associated with each leaf sub-network in the HiGS network structure.

The deletion probability is initially set to its predefined maximum value (chosen to be 1/6). It is then modified depending on the results of the previous deletion. The new value of the deletion probability is a linear function of the number of nodes that have been deleted, such that if no nodes have been deleted, the deletion probability will increase by 20%, and if all nodes have been deleted, then it will decrease by 20%. If all the nodes have been deleted from a network, then a new network with the minimal number of nodes is generated in the Voronoi region of its parent node.

Note that the deletion probability is only the probability with which deletion is attempted; for deletion to succeed, the signal counter values of some nodes in the target leaf network must be sufficiently small.

Termination: The algorithm allows the user to specify a target quantization error. A system quantization error is set to a value below, but close to, the user specified target error. Whenever the error generated by the network is below the system error, our algorithm deletes all spurious nodes. In this case, all leaf networks are subject to deletion, regardless of the deletion probability.

The rationale behind this decision is that spurious nodes in the network do not represent the distribution of the input samples, and as such, need not be allowed to exist in the network.

These deletions may disrupt the whole network and cause the quantization error to increase above the system quantization error. Since the system quantization error is chosen to be below the user-specified target quantization error, the network can still converge. The target error can be adapted, since its value will influence the time needed for the network to converge. If it is too low, the network will require too much time to achieve it. If it is too high, then the final deletion will always raise the error significantly above the desired error criterion, and the training of the network will be repeated indefinitely.

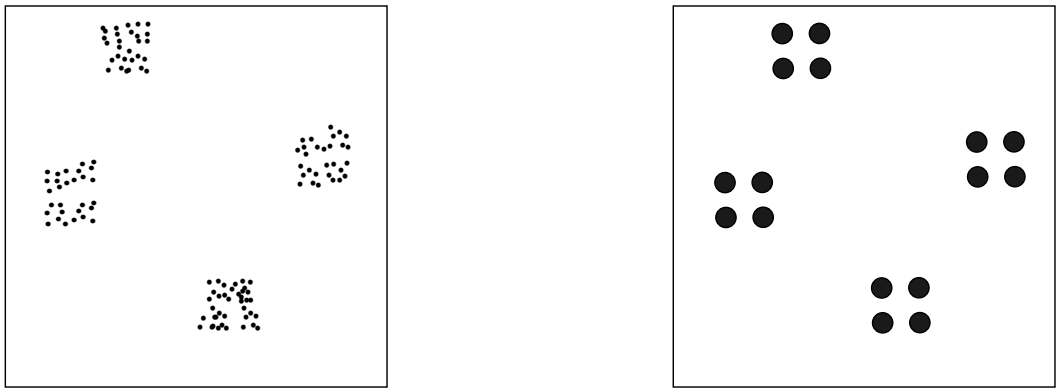
4. Experimental Results

The power of large and non-trivial networks ought to be tested on large and non-trivial problems. Martinetz *et al.* [6] described an interesting problem to illustrate the Neural Gas algorithm. In an instance of this problem, illustrated in Figure 2(a), the input space is filled with a number of disjoint square-shaped clusters, all of which are sufficiently far enough from one another to be clearly distinguishable. All input samples are drawn from one of these squares; within each square, data is uniformly distributed.

We conducted experiments with data (similar to Figure 2(a)) containing various numbers of clusters, ranging from 5 to 15 clusters, with different sizes. In all cases, HiGS succeeded in converging to networks with desired quantization error.

Our experiments compared the HiGS algorithm with Fritzke's GCS, and the Neural Gas algorithm. The deletion rate of the GCS algorithm was set to one deletion every 50 iterations. The Neural Gas algorithm was run with three different numbers of nodes, chosen according to three different criteria: a fixed number of nodes (60), fixed number of nodes per cluster (four), and a fixed number of nodes per area of the input space (one node per 0.01% of the input space area).

Figure 3 describes the instances of the problem and results obtained using various algorithms. Experiments were considered with ten and five clusters in the data, respectively, and with different cluster sizes, with the input space comprising the unit square. All the experiments were conducted 4-5 times, and the results presented are averages over various runs.



(a) (b)

Fig. 2: (a) Example data used to compare algorithms, consisting of several widely distributed clusters. (b) "Desired" positions of weight vectors of nodes, if the number of nodes is four times the number of clusters.

The size of the clusters is given in units of the input space size. All the experiments were run on the same Sun SPARCstation 1 workstation, so that the running times of different experiments are comparable. The times are given in seconds. They may vary when executing the same experiment on different machines, but the ratios of execution times of different experiments should remain the same.

As is evident from Figure 3, HiGS always outperformed GCS, albeit with using a larger number of nodes. The larger number of nodes is expected, because of the nodes in the non-leaf networks in HiGS. The Neural Gas algorithm performs better than HiGS only when the number of nodes is chosen to be a multiple of the area which will be covered by the input samples, *and* the number of nodes is small. In all other cases, HiGS is superior.

5. Conclusion

We have developed a hierarchical neural network (HiGS) that extends Fritzke's Growing Cell Structures network. The algorithm dynamically determines the number of layers in the network, size of each layer, and frequency with which nodes are deleted.

We have demonstrated the success of this algorithm on several instances of the many-square problem used as benchmark by Martinetz *et al.* HiGS, with an adaptive deletion rate, fares better than GCS.

HiGS is computationally efficient, since each sample presentation in HiGS requires time proportional to (average depth of the tree)$\times$(average number of nodes per sub-network), which is less than the total network size. Unlike SOM and Neural Gas algorithms, HiGS does not assume the optimal network size to be known. HiGS does generate more nodes than absolutely necessary, a problem tackled in some of our experiments by trimming leaf nodes of the hierarchy, whenever this can be done without significant degradation of quantization error.

Further experimentation is under way, following preliminary successful results. The algorithm may be modified to allow the higher layer nodes also to be adapted at later stages of the learning algorithm, albeit with smaller learning rates. A supervised extension of the hierarchical GCS

Fig. 3: Comparison of HiGS, GCS, and Neural Gas algorithms

No. of clusters	Cluster size	Algorithm	No. of Nodes	No. of Iterations	No. of Presentations	Time	Frequency of convergence
10	0.02	HiGS	106	91	76084	112.8	100%
10	0.02	GCS	73	91	68408	172.4	100%
10	0.02	NG	60	n/a	24000	369.2	100%
10	0.02	NG	40	n/a	30000	253.0	100%
10	0.05	HiGS	443	138	434673	872.5	100%
10	0.05	GCS	206	229	656855	17250.0	25%
10	0.05	NG	60	n/a	50000	766.4	0%
10	0.05	NG	40	n/a	50000	381.5	0%
10	0.05	NG	250	n/a	24000	>20000	100%
10	0.01	HiGS	51	130	90016	132.0	100%
10	0.01	GCS	14	300	96596	145.1	0%
10	0.01	NG	60	n/a	18000	290.5	100%
10	0.01	NG	40	n/a	24000	203.5	100%
10	0.01	NG	10	n/a	42000	47.5	100%
5	0.02	HiGS	52	43	22044	32.8	100%
5	0.02	GCS	29	37	16904	30.1	100%
5	0.02	NG	60	n/a	12000	184.5	100%
5	0.02	NG	20	n/a	27000	72.2	100%
5	0.05	HiGS	232	81	134188	246.0	100%
5	0.05	GCS	172	185	325412	1927.9	100%
5	0.05	NG	60	n/a	50000	784.6	0%
5	0.05	NG	20	n/a	50000	135.6	0%
5	0.05	NG	125	n/a	30000	1744.2	100%
5	0.01	HiGS	24	67	25908	29.8	100%
5	0.01	GCS	11	300	55544	52.1	0%
5	0.01	NG	60	n/a	5000	76.8	100%
5	0.01	NG	20	n/a	17000	43.9	100%
5	0.01	NG	5	n/a	35750	16.2	100%

algorithm is also envisaged, in which leaf nodes in the hierarchy are analogous to nodes in radial-basis function networks, and convey their outputs to the output nodes equipped with a linear or nonlinear node function; one of the criteria for generating a sub-network is then the supervised learning error to which a leaf node contributes.

References

[1] B. Fritzke, *Growing Cell Structures–A Self-Organizing Network for Unsupervised and Supervised Learning*, Neural Networks, vol. 7, no. 9, pp. 1441-1460, 1994.

[2] S. Grossberg, Studies of Mind and Brain, Reidel, Boston (MA), 1982.

[3] A. K. Jain and R. C. Dubes, Algorithms for Clustering Data, Prentice Hall, Englewood Cliffs (NJ), 1988.

[4] T. Kohonen, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics, vol. 43, pp. 59-69, 1982.

[5] J. Lampinen and E. Oja, *Clustering Properties of Hierarchical Self-Organizing Maps*, Journal of Mathematical Imaging and Vision, vol. 2, pp. 261-271, 1992.

[6] T. M. Martinetz, S. G. Berkovich and K. J. Schulten, *"Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction*, IEEE Transactions on Neural Networks, vol. 4, pp. 558-569, July 1993.