Imposing Tree-based Topologies onto Self Organizing Maps

César A. Astudillo[1⋆] and B. John Oommen[2⋆⋆]

[1] Universidad de Talca, Merced 437 Curicó, Chile. castudil@utalca.cl
[2] School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. oommen@scs.carleton.ca

Summary. The beauty of the Kohonen map is that it has the property of organizing the codebook vectors, which represent the data points, both with respect to the underlying distribution and topologically. This topology is traditionally linear, even though the underlying lattice could be a grid, and this has been used in a variety of applications [19, 25, 28]. The most prominent efforts to render the topology to be structured involves the Evolving Tree (ET) due to Pakkanen *et al.* [26], and the Self-Organizing Tree Maps (SOTM) due to Guan *et al.* [15], among others. In this paper we propose a strategy, the Tree-based Topology-Oriented SOM (TTO-SOM) by which we can impose an *arbitrary*, user-defined, tree-like topology onto the codebooks. Such an imposition enforces a neighborhood phenomenon which is based on the user-defined tree, and consequently renders the so-called bubble of activity to be drastically different from the ones defined in the prior literature. The map learnt as a consequence of training with the TTO-SOM is able to infer *both* the distribution of the data and its structured topology interpreted via the perspective of the user-defined tree. The TTO-SOM also reveals multi-resolution capabilities, which are helpful for representing the original data set with different numbers of points, and this can be obtained without the necessity of recomputing the whole tree. The ability to extract an skeleton, which is a "stick-like" representation of the image in a lower dimensional space, is discussed as well. These properties has been confirmed by our experimental results on a variety of data sets.

⋆ This author is a *Profesor Instructor* at the Departamento de Ciencia de la Computación, with the Universidad de Talca in Chile. The work of this author was done while pursuing his Doctoral studies at the School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. A preliminary version of this paper was presented at the CORES'09, the 2009 Conference on Computer Recognition Systems, Wroclaw, Poland, May 2009. *This talk was a Plenary/Keynote Talk at the Conference.* We record our gratitude to the anonymous Referees of the original version of this paper for their painstaking reviews. The changes that they requested certainly improved the quality of this paper. *Thanks a lot!*

⋆⋆ *Chancellor's Professor* ; *Fellow: IEEE* and *Fellow: IAPR.* This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway. This author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada.

Keywords: Hierarchical SOM, Self Organizing Maps, Kohonen Maps, Topology-Based Self-Organization

1 Introduction

In an increasing number of applications, researchers today encounter situations which require the classification of patterns in large data sets. Central to these methods are approaches which identify the most important *clusters* of the given data in an unsupervised manner. A problem that arises in such situations is to capture the essence of the similarity in the samples, which implies that any given cluster should include data of a "similar" sort, while elements that are dissimilar are assigned to different subsets.

A fundamental approach to solve the clustering problem involves the use of Artificial Neural Networks (ANNs) capable of unsupervised learning. Within this model of computation, a key component is the phenomenon referred as the "competition" between the neurons. From this perspective, it is possible to classify ANNs into two main sub-categories: In the first family only a single output neuron is activated, and this is referred to as *Hard* Competitive Learning. As opposed to this, in the second approach, both the winner and other associated neurons are involved in the process, and this is referred to as *Soft* Competitive Learning.

Another important classification criterion that can be found in such ANNs is the type of topological relationship between the neurons. Some approaches present topologies that are imposed from the start of the training process, and which remain constant until the convergence is attained. Other strategies include topologies that are able to change with time, thus yielding, at the end of the learning process, a topology that represents the underlying data distribution.

One of the most important families of ANNs used to tackle the abovementioned problems is the well known Self-Organizing Map (SOM) [**?**]. Typically, the SOM is trained using (un)supervised learning to produce a neural representation in a space whose dimension is usually smaller than that in which the training samples lie. Further, they seek to preserve the topological properties of the input space.

Although the SOM has demonstrated an ability to solve problems over a wide spectrum, it possesses some fundamental drawbacks. One of these drawbacks is that the user must specify the lattice *a priori*, which has the effect that the user must run the ANN a number of times to obtain a suitable configuration. Other handicaps involve the size of the maps, where a lesser number of neurons often represent the data inaccurately.

The state-of-the-art approaches attempt to render the topology more flexible, so as to represent complicated data distributions in a better way and/or to make the process faster by, for instance, speeding up the task of determining the best matching neuron.

Some approaches try to start with a small lattice [15, 26], while others attempt to grow a SOM grid by adding new rows or columns if the input samples are too concentrated in some areas of the feature space [9]. Others follow a symmetric growing of the original lattice [12]. Alternatively, some strategies add relations (edges) between units as time proceeds, thus not necessarily preserving a grid of neurons. In particular, the literature reports methods that use a tree-shaped arrangement [15]. Also, some researches have attempted to devise methods which "forget" old connections as time goes by [13]. There are strategies that add nodes during training [26], while others use a fixed grid [**?**] or arrange different SOMs in layers [9], and combinations of these principles have also been employed. On the other hand, strategies that try to reduce the time required for finding the winner neuron have also been designed. The related approaches focus on the accuracy of the resulting neurons being the ones to be modified, and on the consequent topology. It is important to remark that no single one of these approaches has been demonstrated to be a clear winner when compared to the other strategies, thus leaving the window of opportunity open for novel ideas that can be used to solve the above-mentioned problems.

The desirable feature of the SOM, namely, to yield prototypes which converge in distribution was a landmark, and the formal results which prove that the stochastic distribution of the prototypes follows the distribution of the underlying data points is available. However, this formal result is true only if the prototypes are linearly arranged, implying that the concept of neighborhood and the bubble of activity are also *linear*. This statement, of course, needs clarification. For example, if the nodes are arranged on a lattice, the neighbors of any node $\langle i, j\rangle$ are the nodes $\{\langle i-1, j\rangle, \langle i+1, j\rangle, \langle i, j-1\rangle, \langle i, j+1\rangle\}$, which, on a more critical inspection, can be seen to be those "linearly" close to $\langle i, j\rangle$. But this leads us to a host of other open issues. The first issue is the following: Is it possible for a user to require the prototypes to follow any arbitrary topology? If this is possible, the implication is significant. First of all, the question of finding the nearest neuron with such a topology (i.e., the winner in the competition) has to be answered. Secondly, the whole issue of how one describes the bubble of activity, and of migrating neighbors of the winner is far more significant. Indeed, the question of what we mean by a neighbor, how the list of neighbors is maintained and what the final convergence is, are unresolved issues. This, indeed, is the goal of this paper.

Our aim is to permit the user to specify any tree-like topology which can be entirely up to his imagination. The reason why we prefer a tree-like topology is to prevent cyclic neighborhood correspondences. Once this topology has been fixed, the concepts of the neighborhood and bubble of activity are specified from *this* perspective. The question is whether the prototypes can ultimately learn the stochastic distribution *and simultaneously* arrange themselves with the topology that mimics the one that the user hypothesized from the outset. We show that this is indeed possible, as demonstrated by a set of rigorous experiments in which our enhanced ANN, the Tree-based Topology

Oriented SOM (TTO-SOM) is able to learn both the distribution and the desired structured topology of the data. Furthermore, a consequence of this is the fact that as the number of neurons is increased, the approximation of the space will be correspondingly superior – both from the perspective of the distribution and of the user-defined topology.

1.1 Contribution of the Paper

The principal contributions of the present work can be summarized as follows:

1. We present a technique by which we can represent data points using prototypes, both with respect to the underlying distribution and *any* user-defined topology.
2. In particular, we demonstrate how the user can impose an arbitrary tree-like topology onto the set of codebook vectors.
3. Since the topology can be fairly arbitrary, we show that the resultant bubble of activity is different from the ones defined in the prior literature, both structurally and conceptually.
4. The map learned as a consequence of the training process is able to infer both the distribution and the structured topology of the data as verified by extensive experiments.
5. The strategy proposed reduces to the traditional 1-dimensional SOM when the tree is a linear sequence of nodes. In other words, the traditional SOM is a special case of the family of ANNs which we propose.

1.2 Organization of the Paper

The organization of the paper is as follows. We first give an overview of the state of the art in Sec. 2. Thereafter in Sec. 3, we present the new variant, namely the TTO-SOM, and the experimental results which demonstrate its power are included in Sec. 4. Finally, Sec. 5 gives some concluding remarks and discussions.

2 Brief Review of the State-of-the-Art

In general, the objective in competitive learning is to place a certain number of *units* (synonymously called codebooks, neurons or prototypes) $\mathcal{C} = \{c_1, c_2, .., c_M\}$ in such a way that they best represent a set of sample points $\mathcal{X} = \{x_1, x_2, .., x_N\}$, $x_i \in I\!R^n$, referred to as the *input data set*. A *reference vector*[3] $v_i \in I\!R^n$ is associated with neuron c_i indicating its position in the feature space. The inter-neuron connections are specified by a (possibly empty)

[3] The dimensionality of these reference vectors is usually smaller than the dimensionality of the sample points.

set $E \subset \mathcal{C} \times \mathcal{C}$ of *neighborhood connections*, which are typically unweighted and symmetric. The generalization to weighted and asymmetric edges is easily achieved, but not of interest in this paper.

The concept of the *direct neighborhood* of any unit c, denoted as N_c, is defined as the set of units such that

$$N_c = \{i \in \mathcal{C} | (c, i) \in E\}. \tag{1}$$

When the ANN receive an input, the closest or *Best Matching Unit* (BMU) is represented by

$$s(x) = \arg\min_{c \in \mathcal{C}} \| x - v_c \|, \tag{2}$$

where $\| \cdot \|$ denotes the appropriate norm (for example, the City-Block or Euclidean distance). Clearly, the notation can be extended to refer to the closest unit $s_1(x)$, the second closest unit $s_2(x)$, and so on.

Observe that the units and their edges together constitute a network, where an unweighted edge represents a relationship between the nodes. These relationships are useful when the units are updated. For example, such updates occur if only $s_1(x)$, the closest unit, is moved towards the input sample x by an update process (hard competitive learning), or additionally other units in the neighborhood of $s_1(x)$ are updated as well, as in soft competitive learning [14].

The SOM [**?**] is a single-layered forward-directed ANN used in clustering and visualization. It allows a mapping from a high-dimensional space to lower dimensions, typically of two or three dimension, so as to enhance visualization. The SOM attempts to concentrate all the information contained in the set of input samples $\mathcal{X}$ within the set $\mathcal{C}$ of representative codebook vectors, which are the constituent vectors. In order to adjust the values of the neurons, the SOM prescribes a specific update process. This process involves information about the input sample x, the closest unit $s_1(x)$, which is the BMU, and a neighborhood function which encapsulates the relative information between the neurons [31].

There are, probably, hundred of publications that attempt to improve or take advantage of the SOM. Applications of the SOM are just as numerous and include the fields of Image and Video Processing, Pattern Recognition (PR), Artificial Intelligence, Engineering, Medicine, etc. [19, 25, 28]. It is fair to state that numerous researchers have attempted to produce enhanced variations of the basic algorithm since its pioneering introduction into the scientific domain.

In attempting to devise enhanced SOM algorithms, researchers have incorporated an ensemble of approaches so as to overcome specific handicaps, for example, that of knowing the size of the map *a priori*. Not knowing the map size can often require experimentation with different sized maps, which, in itself, can be very time consuming. In the case of data clustering, for example, the SOM converges to a network in which nodes stabilize, at sometimes,

sparsely populated areas. In particular, the SOM requires a lot of time to compute the BMU for large maps.

The literature contains enhancements aimed at addressing the problems presented above. The main strategy is to generate a lattice that can dynamically grow. Some approaches attempt to grow a SOM grid by adding new rows or columns if the input samples are too concentrated in some areas of the feature space [9]. Other methods add relations (edges) between the units as time proceeds [15], while others yet attempt to "forget" old connections as time goes by [13]. While there are strategies that add nodes during training [26], others use a grid-based arrangement of the nodes [4], or arrange different SOMs in layers [9], or utilize a hierarchical structure of the neurons [23]. All these strategies possess their unique advantages and disadvantages. The summary of the key components of relevant SOM variants follows[4].

The Growing Cell Structures (GCS) [11], maintains the accumulated error for each neuron, and after a fixed number of iterations, an additional neuron is added between the unit possessing the largest accumulated error and its farthest direct neighbor.

The Neural Gas (NG) [22] generates a ranking of the neurons according to their distance to the stimulus. Each connection between the neurons includes a so-called "age" counter utilized for forgetting old connections. Neurons that are closer to the stimulus generate/rejuvenate their connections. The resulting topology is neither a tree nor a grid, but a graph that is, possibly, not fully connected. In this method, the most expensive task is the one that involves finding the ordering of the elements, requiring $O(N \log N)$ time.

The Growing Neural Gas (GNG) [13] successively adds new units to the network. It keeps the age of the neurons, and forgets connections in the same way as the NG. Additionally, nodes without connections are deleted. The GNG maintains an error for each unit which is further employed for creating new nodes.

The Structure-Adaptive Self-Organizing Map (SASOM) [6] is a SOM-based NN designed for performing pattern recognition by dynamically adapting the structure of the network. The algorithm starts with a small SOM lattice, and trained until convergence is achieved. During the expanding step, the node that should be split is identified and replaced by a small SOM grid. Unless every node represents a unique class, the algorithm trains the current SOM and the process is repeated.

The Growing Hierarchical Self-Organizing Map (GHSOM) [30] is a dynamically growing NN. The GHSOM uses a hierarchical architecture of SOM-like layers. New rows/columns are added to a layer depending on an error measure, and the new SOM-like layers are added if the BMU exceeds certain quantization error criteria.

[4] The discussion of the variants of the SOM has been abridged due to the request of the reviewers.

The Self-Organizing Tree Map (SOTM) [15] is an incrementally grown tree-based SOM which starts with an isolated node. The SOTM creates a new node when the distance between the stimulus and the BMU is greater than a given threshold. As the algorithm is not capable of "forgetting" connections, decisions made in early stages of the training process can strongly determine the shape of the tree.

The Tree-Structured Vector Quantization algorithm (TSVQ) [21] utilizes a tree structure which is fixed in depth and in breath. After a node is trained for a certain amount of time it become "frozen". Frozen units are static neurons, i.e. they become neurons that do not accept training, but allow the training of their direct children. The TSVQ incorporates a BMU search heuristic which is performed in $O(\log n)$ time.

The Evolving Tree (ET) [26] utilizes a tree topology that is allowed to grow. After a certain number of training steps, a node become frozen and then splits, generating a pre-specified number of children, which are further trained. This process is repeated recursively, thus generating a tree. One of the known problems of the ET is that leaves belonging to different branches may move closer to each other after a certain number of iterations. In this way, clusters that are identified at higher levels of the hierarchy of the tree, may not be well represented as one proceeds towards its lower levels.

2.1 Topology Preservation

According to the author of [20], there are three different criteria which can be used to evaluate how good a map is. The first criterion indicates how *continuous* the mapping is, implying that input signals that are close (in the input space) should be mapped to codebooks that are close in the output space as well. A second criterion involves the *resolution* of the mapping. Maps with high resolution possess the additional property that input signals that are distant in the input space should be represented by distant codebooks in the output space. A third criterion imposed on the accuracy of the mapping is aimed to reflect the *probability distribution* of the input set.

There exist a variety of measures for quantifying the quality of the topology preservation [1]. The author of [29] surveys a number of relevant measures for the quality of maps, and these include the Quantization Error, the Topographic Product [3], the Topographic Error [20] and finally the Trustworthiness and Neighborhood Preservation [33].

The ordering of the weights (with respect to the position) of the neurons of the SOM has been proved for unidimensional topologies [7, **?**, 31]. Extending these results to higher dimensional configurations or topologies leads to numerous unresolved problems. First of all, the question of what one means by "ordering" in higher dimensional spaces has to be defined. Further, the issue of the "absorbing" nature of the "ordered state" is open. Budinich, in [5], explains intuitively the problems related to the ordering of neurons in higher

dimensional configurations. Huang *et al* [17] introduce a definition of the ordering and show that even though the position of the codebook vectors of the SOM have been ordered, there is still the possibility that a sequence of stimuli will cause their disarrangement. Some statistical measures of correlation between the measures of the weights and distances of the related positions have been introduced in [2].

This concludes our "brief" survey of the state-of-the-art.

3 The Tree-Based Topology SOM

The Tree-based TOpology SOM (*TTO-SOM*) is a tree-structured SOM which aims to discover the underlying distribution of the input data set $\mathcal{X}$, while also attempting to perceive the topology of $\mathcal{X}$ as viewed through the user's desired perspective. The TTO-SOM works with an imposed topology structure, where the codebook vectors are adjusted using a VQ-like strategy. Besides, by defining a user-preferred neighborhood concept, as a result of the learning process, it also learns the topology and preserves the prescribed relationships between the neurons as per this neighborhood. Thus, the primary consideration is that the concept of neurons being "near each other" is not prescribed by the metric in the space, but rather by the structure of the imposed tree.

3.1 Declaration of the user-defined tree

The topology of the tree arrangement of the neurons plays an important role in the training process of the TTO-SOM. This concept is not new, and a few variations of SOMs that take advantage of this approach have been reported [9, 15, 21, 23, 26, 30].

In general, the TTO-SOM incorporates the SOM with a tree which has an arbitrary number of children. Furthermore, it is assumed that the user has the ability to describe/create such a tree. The user who presents it as an input to the algorithm, utilizes it to reflect the *a priori* knowledge about the *structure* of the data distribution[5] .

The first task to be conducted is that of declaring, as an input, the user-defined tree. We propose that this declaration is done in a recursive manner (see Alg. 1), from which the structure of the tree is fully defined. The input to the algorithm is an array that contains integers specifying the number of children for each node in the tree if the latter is traversed in a Depth-First (DF) manner[6].

The algorithm works in a straightforward manner: The input to the algorithm is a pointer to a node to a tree, which could be the root and

[5] The beauty of such an arrangement is that the data can be represented in multiple ways depending on the specific perspective of the user.

[6] The tree could also easily be traversed in a breadth-first manner as we will see presently.

an associated array. The position i in the array implicitly refers to the i^{th} node of the final tree if traversed in a DF manner. The contents of the array elements in the i^{th} position refer to the number of children that node i has. An example of this is given in Fig. 1 where the input array is $\langle 2, 3, 4, 0, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0\rangle$, and the resulting tree is shown in the figure itself. Observe that the same tree could have also been traversed in a breadth-first manner, in which case, the corresponding array for this tree would be the array $\langle 2, 3, 2, 4, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0\rangle$.

Algorithm 1 `describe-topology`(A,p)

Input:
i) A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n\rangle$, specifying the number of children for each node in the tree.
ii) p, a pointer to the current root of the tree.
Method:

```
 1: if n = 0 then
 2:    return
 3: else
 4:    number-of-children = head(A) {read and cut head of the sequence}
 5:    for i=1 to number-of-children do
 6:       create-node(x)
 7:       add-node(x,p)
 8:       describe-topology(A,x)
 9:    end for
10: end if
```

End Algorithm

3.2 Representation of the Tree

From the immense diversity of possible data structures capable of representing trees, choosing a data structure that best suits the purposes of our problem requires the specification of all the constraints that are to be satisfied. First, each node in the tree could possibly have an arbitrary number of children, limited only by the user's "imagination". Secondly, the form in which the tree is represented must permit fast traversal, specially for the most expensive tasks of the algorithm, namely, the location of the current BMU, and the calculation of the bubble of activity, i.e., the neighborhood of units around it. Fast identification of the parents is required, since our concept of neighborhood involves the traversal of the nodes, not only in the direction of the children of a given node, but also in the direction of its ancestors, as explained in Sec. 3.3, and pictorially described in Fig. 3, which implicitly requires us to maintain a link to quickly reach the parent.

An efficient way to represent the structure of the tree, is to use the Left-most-child, Right-sibling representation. In this approach each node points

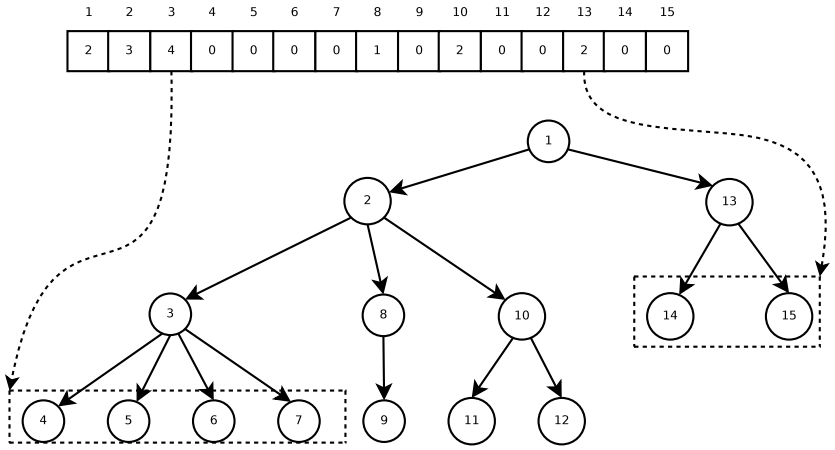


Fig. 1: An example of the description of the original tree topology. The example shows an array containing the number of children for each node in the tree. If the latter is traversed in a DF manner, the index of the array is the position in the DF traversal, and the content of the array is the number of children of the node in question. The corresponding array, if the tree is traversed in a breadth-first manner, is $\langle 2, 3, 2, 4, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$.

only to its left-most child and not to all the direct children, and every node is linked only to its sibling immediately to the right (and not to all the siblings). Simultaneously, we also maintain a pointer from every node to its parent, with the exception of the root node which is pointing to the *NULL* pointer. Interestingly, the Left-most-child, Right-sibling representation stores the data in $O(M)$ space, where M is the number of nodes. Also, this strategy permits the representation of the tree with an arbitrary number of children, to be a *binary tree*.

The implementation of the tree using the "Left-most-child, Right-sibling" is given in Fig. 2 (for the tree of Fig. 1), and will also be represented with the array $\langle 2, 3, 4, 0, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0 \rangle$.

With regard to the data maintained in each node of the tree, we mention that, each node will contain the d-dimensional vector in the feature space migrated as per the position of the input samples and the update rule. Note that in Fig. 2 the details of the vectors in the feature space are omitted in the interest of simplicity

3.3 Neural Distance Between Two Neurons

In the TTO-SOM, the *Neural Distance*, d_N, between two neurons depends on the *number* of unweighted connections that separate them in the user-defined tree, and is defined as the number of edges in the shortest path that connects

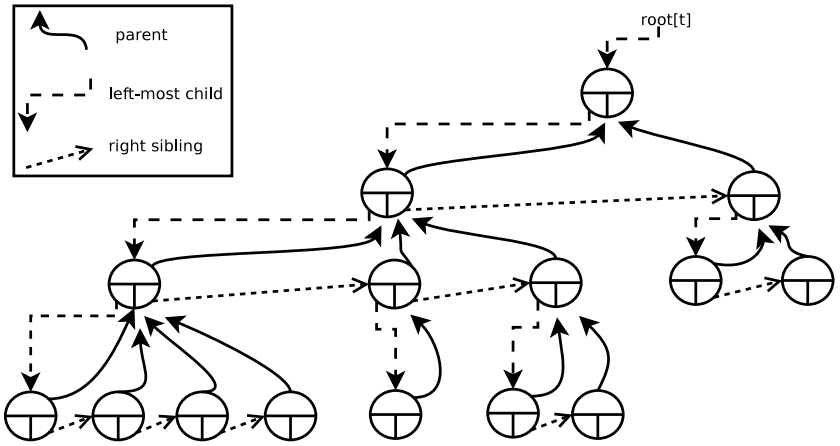


Fig. 2: A suitable way to implement the user-defined tree is by using the Left-most-child, Right-sibling representation, which uses $O(n)$ space. This figure illustrates the Left-most-child, Right-sibling representation for the tree of Fig. 1. For simplicity, the details of the vector in the d-dimensional feature space, associated with each node, are omitted.

the two given nodes. More explicitly, the distance between two nodes in the tree, is defined as the minimum number of edges required to go from one to the other. In the case of trees (since this is the focus of the present work), there is only a single path connecting two nodes, implying the simplicity of enforcing the definition. Additionally it is worth mentioning that this notion of distance is not dependent on whether or not nodes are leaves or not, as in the case of the ET [26], thus permitting the calculation of the distance between *any* pair of nodes in the tree.

More specifically, the distance between a neuron and itself is defined to be zero, and the distance between a given neuron and all its direct children and its parent is unity. This distance is then recursively defined to farther nodes as shown pictorially in Fig. 3a. Clearly, if v_i and v_j are nodes in the tree, $d_N(\cdot,\cdot)$ possesses the identity, non-negativity, symmetry and triangular inequality properties:

$$d_N(v_i, v_j) \geq 0 \tag{3}$$

$$d_N(v_i, v_j) = 0 \qquad \text{if and only if } v_i = v_j \tag{4}$$

$$d_N(v_i, v_j) = d_N(v_j, v_i) \tag{5}$$

$$d_N(v_i, v_k) \leq d_N(v_i, v_j) + d_N(v_j, v_k). \tag{6}$$

The reader should observe in Fig. 3 that nodes at different levels could also be equidistant from any given node. Thus, in the case of Fig. 3a, nodes B, C and D are all at a distance of 2 units away from A, while in Fig. 3b

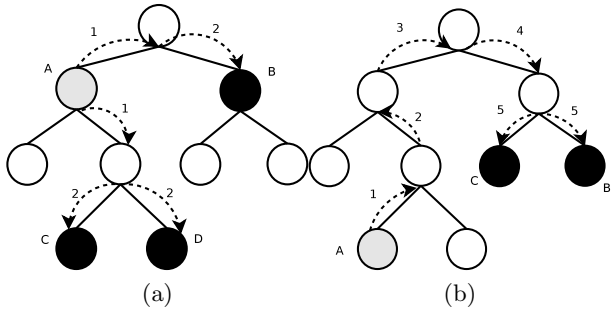


Fig. 3: Fig. 3a shows the neighborhood for the TTO-SOM. Here nodes B, C and D are equidistant to A even though they are at different levels in the tree. Observe that non-leaf nodes may be involved in the calculation. Fig. 3b shows the case for the ET, where nodes B and C are equidistant to A. In the ET, the definition of the distance pertains only to leaf nodes.

nodes B and C are at a distance of 5 units from node A. It should also be mentioned that in the ET, one encounters only distances between the leaves (as in Fig. 3b) and not between internal nodes, as in Fig. 3a.

As in the case of the traditional SOM, the TTO-SOM requires the identification of the BMU, i.e. the closest neuron to a given input signal. This is achieved by invoking the function TTO-SOM_Find_BMU (see Alg. 2). To locate it, the distances, $d_f(\cdot,\cdot)$ are computed in the *feature* space and not in terms of the edges of the user-defined tree. The algorithm first calculates the feature-based distance in the feature space between the input signal, x, and every neuron in the network, and the index of the neuron with the smallest feature-based distance is returned. In the case of a tie, the index of the first-found neuron with minimum distance is selected (although a random tie breaker is also possible). It is important to emphasize that the distance measured between neurons is completely distinct from the feature-based distance between a neuron and the input signal, since the latter is computed based on the coordinates of the neuron and the signal in the feature space, and the distance between neurons is computed in terms of the number of edges to be traversed in the user-defined tree. A common choice for the distance is the Euclidean distance, however, it has been reported that for higher dimensional spaces, this particular definition of distance might be inaccurate [34]. The TTO-SOM attempts to tackle this issue by inheriting the SOM's update rule, which utilizes a distance function but does not necessarily specify one explicitly.

Algorithm 2 `TTO-SOM_Find_BMU`(p,x,v)

Input:
i) p, the node being examined.
ii) x, an input sample.
iii) v, the best matching unit found so far.
Output:
v, the best matching neuron.
Method:
1: **if** p=`NULL` **then**
2: **return** v
3: **else**
4: **if** `getDistance`(x, v) >`getDistance`(x, p) **then**
5: v=p {p is the new best matching unit}
6: **end if**
7: **for all** *child* $\in p$.`getChildren`() **do**
8: v=`TTO-SOM_Find_BMU`(*child*,x,v)
9: **end for**
10: **end if**
11: **return** v
End Algorithm

3.4 The Bubble of Activity

Intricately related to the notion of inter-node distance, is the concept referred as to the "Bubble of Activity" which is the subset of nodes "close" to the unit being currently examined. These nodes are essentially those which are to be moved toward the input signal presented to the network. This concept involves the consideration of a quantity, the so-called *radius*, which determines how big the bubble of activity is, and which therefore has a direct impact on the number of nodes to be considered. The bubble of activity is defined as the subset of nodes within a distance of r away from the node currently examined, and can be formally defined as

$$B(v_i; T, r) = \{v | d_N(v_i, v; T) \leq r\}, \tag{7}$$

where v_i is the node currently being examined, and v is an arbitrary node in the tree T, whose nodes are V. Note that $B(v_i, T, 0) = \{v_i\}$, $B(v_i, T, i) \supseteq B(v_i, T, i-1)$ and $B(v_i, T, |V|) = V$ which generalizes the special case when the tree is a (simple) directed path. Fig. 4 depicts an example of how the number of neurons in the bubble of activity is increased as the radius is increased. In the case of the TTO-SOM the BMU can be *any* node in the tree, and this becomes the center of the bubble of activity. The question of whether or not a neuron should be part of the current bubble, depends on the number of connections that separate the nodes rather than the distance that separate the points in the solution space (for instance, the Euclidean distance).

The concept of neighborhood utilized in the present work is distinct and different from the ones used in other approaches such as the ET [26] or the

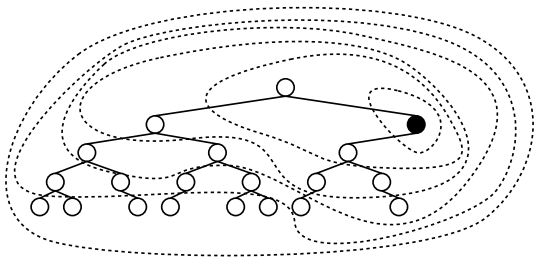


Fig. 4: The figure shows how the bubble of activity includes more nodes as the radius is increased. The currently examined node is given in black.

SOTM [15]. In the case of the TTO-SOM, non-leaf units are considered in the bubble of activity. As an example, in our case, nodes distant by a radius of zero will include, in the bubble of activity, only the node being currently examined, while a radius of unity will consider the subset of all the direct children of the unit being examined as well as the direct parent node. As in any SOM philosophy, the bubble of activity is initially made large enough so as to include all the nodes in the structure, and subsequently, as the learning proceeds, gradually decreased to finally only include the node currently being examined. There are reported works that consider only leaves in the bubble of activity [26] and some of them use the distance of the neurons in the solution space [15].

The function *TTO-SOM_Calculate_Neighborhood* (see Alg. 3) dictates the calculation of the subset of neurons that are part of the neighborhood of the BMU. This computation involves a collection of parameters, including B, the current subset of neurons in the proximity of the neuron being examined, v, the BMU itself, and $r \in \mathbb{N}$ the current radius of the neighborhood. When the function is invoked for the first time, the set B contains only the BMU (which is previously estimated using Alg. 2) marked as the current node, and through a recursive call, B will end up storing the entire set of units within a radius r of the BMU. The tree is recursively traversed for all the direct topological neighbors of the current node, i.e. in the direction of the direct parent and children. Every time a new neuron is identified as part of the neighborhood, it is added to B and a recursive call is made with the radius decremented by one unit[7], marking the recently added neuron as the current node.

3.5 Training the TTO-SOM

The training process of the TTO-SOM involves positioning the neurons which describe the user-defined tree in the feature space so as to capture the distribution and topology of the data points. This process involves a loop of

[7] This fact will ensure that the algorithm reaches the base case when $r = 0$.

Algorithm 3 `TTO-SOM_Calculate_Neighborhood`(B,v,r)

Input:
i) B, the set of the nodes in the bubble of activity identified so far.
ii) v, the node from where the bubble of activity is calculated.
iii) r, the current radius of the bubble of activity.
Output:
The set of nodes in the bubble of activity.
Method:

```
1: if r ≤ 0 then
2:    return
3: else
4:    for all child ∈ v.getChildren() do
5:       if child ∉ B then
6:          B ← B + {child}
7:          TTO-SOM_Calculate_Neighborhood(B,child, r − 1)
8:       end if
9:    end for
10:   parent=v.getParent();
11:   if parent ≠ NULL and parent ∉ B then
12:      B ← B + {parent}
13:      TTO-SOM_Calculate_Neighborhood(B,parent, r − 1)
14:   end if
15: end if
```

End Algorithm

training steps which terminates when the convergence is acceptable to the user. The *Training step* involves requesting an input sample from the dataset, locating the BMU, computing the nodes within the *current* bubble of activity, and migrating those neurons toward the input signal using a SOM-like philosophy.

How the input sample is chosen may vary. In the case of the TTO-SOM, an input sample is drawn at random from the sample set. Other mechanisms may involve the shuffling of the elements in the set, generating an order, and then at each step the samples are chosen as per the pre-established order. Finally, when the last element in the sequence has been drawn, an *epoch* is said to be completed, and optionally a new shuffling takes place.

The TTO-SOM uses the SOM update rule to update the prototypes, and requires the definition of a learning rate α which is a real number between 0 and 1, which, in turn, specifies the fraction by which the BMU will move towards the input presented to the network. The value of the learning rate is initialized to a high value, for example 0.9, and is gradually decreased so as to achieve convergence. At first the updated positions of the BMU and neighbor neurons will be highly influenced by the input sample, but as time goes by, these will tend to move less, and at the end of the process only small changes in the positions of the neurons will be registered.

Alg. 4 gives the details on how the training step is performed. Initially, the index of the BMU is obtained by invoking Alg. 2, and will be the first

neuron to be included in the bubble of activity. The next process involves the calculation of all the neurons in the bubble, which is obtained by calling Alg. 3. Finally, for each neuron in the bubble, an update process take place, which is similar to the SOM update rule.

Algorithm 4 `TTO-SOM_train`(x,p)

Input:
i) x, a sample signal.
ii) p, the pointer to the tree.
Method:
1: $v \leftarrow$ `TTO-SOM_Find_BMU`(p,x,p)
2: $B \leftarrow \{v\}$
3: `TTO-SOM_Calculate_Neighborhood`(B,v,$radius$)
4: **for all** $b \in B$ **do**
5: `update_rule`(b.`getCodebook`(),x)
6: **end for**
End Algorithm

3.6 The Overall Procedure

In what we have covered, we have explained all the independent modules of the TTO-SOM. Using these modules we shall show how all the pieces fit together to compose the overall TTO-SOM algorithm. The main segments of the algorithm also include the initialization of the values and the main training loop.

The input to the algorithm consists of a set of samples in the d-dimensional feature-space, and, additionally, as explained in Sec. 3.1, an array by which the user-defined tree structure can be specified. Observe that this specification contains all the information necessary to fully describe the TTO-SOM structure, such as the number of children for each node in the tree. Furthermore, the algorithm also includes parameters which can be perceived as "tuning knobs". They can be used to adjust the way by which it learns from the input signals. The TTO-SOM requires a schedule for the so-called decay parameters, which is specified in terms of a list, where each item in the list records the value for the learning rate, the radius of the bubble of activity, and the number of learning steps for which the latter two parameters are to be enforced.

The initialization phase constitutes the creation/description of the user-defined tree topology, and the value of the vectors in the feature-space associated with each node within the tree. At the beginning of this phase, the tree must be built as per Alg. 1, implying the specification of the root of the tree. Initially the tree is not defined, and this is done by assigning the parameter p of Alg. 1 to the *NULL* pointer. At this juncture, the starting values of the codebook vectors associated with each node of the tree, are set as well. One common way to accomplish this is by randomly choosing input vectors from

the input sample set, and assigning their values to the codebook vectors. Alternate methods relying on a Principal Component Analysis have also been reported in the literature [18].

The essential phase of the process involves the learning iterations, which is a sequence of steps as per Alg. 4, and are monitored by the schedule of parameters mentioned earlier. Whenever a learning step takes effect, some of the nodes in the tree are moved towards the input sample. The number of nodes moved and the amount by which these neurons are moved, depends on the current parameters, such as the bubble of activity and the learning rate, α. Alg. 5 details the whole process, which terminates when the position of the neurons have converged satisfactorily.

In practice, as we shall see presently, as a result of the execution of Alg. 5, the codebook vectors will be arranged in such a way that they represent the structure and the distribution of the input data. While the "spatial" distribution of the neurons follows the stochastic distribution of the data points, the relationships between the nodes of the TTO-SOM will be as per the user-defined structure. This preserves the topological information in the data points (as in the traditional SOM algorithm), but also the information involving the tree-based relationship between the neurons.

Algorithm 5 `TTO-SOM`(S,A,X)

Input:
i) S, the schedule of parameters
ii) A, the array containing the number of children for each node in the tree
iii) X, the input sample set
Method:
1: $T \leftarrow$ `NULL`
2: `describe-topology`(A,T)
3: `initialize-codebook`(p,X)
4: **for all** $s \in S$ **do**
5: `set-learning-rate`$(s.learning\_rate)$
6: `set-radius`$(s.radius)$
7: **for** $i \leftarrow 1$ to $s.number\_of\_iterations$ **do**
8: $x \leftarrow$ `random-selection`(X)
9: `TTO-SOM_train`(x,T);
10: **end for**
11: **end for**
End Algorithm

4 Experiments and Results

To demonstrate the power of our method, and to obtain a better understanding on how the structure-oriented SOM works, we have implemented and

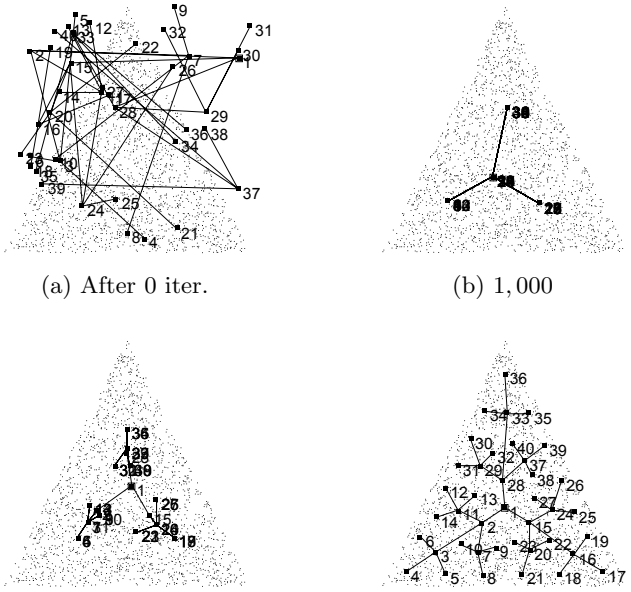
tested it on numerous data sets[8]. First of all, from a pedagogical perspective, the fact that the user is capable of visualizing the resulting tree, is important, because, in this way, a human operator/user will be provided with a quick and intuitive tool for understanding the "structure" of the data. Thus he will be able to comprehend what is happening to the tree that attempts to delineate the structure of the points from the data distribution. This is achieved by virtue of the fact that the position of the nodes belonging to the tree correspond to the value of the codebook vector associated with each node, and which are also manipulated by the cloud of points that are randomly drawn from the input data set. Therefore, to illustrate this philosophy, the experiments reported here, were done in the 2-dimensional feature space. It is important to remark that the capabilities of the algorithm are also applicable for higher dimensional spaces, though, the visualization of the resulting tree will not be as straightforward. Additionally, in all the cases presented here, the input samples were drawn from a probability distribution unknown to the algorithm. While both the distribution and its structure were unknown to the TTO-SOM, the hope is that the latter will be capable of inferring them through the learning process, without human intervention, thus, performing *Unsupervised Learning*. Lastly, the schedule of parameters was specified so as to result in a rather "slow" convergence. This was achieved by defining steady values for the learning rate for a large number of iterations, so that we could understand how the position of the nodes migrated towards their final configuration. We believe that to solve practical problems, the convergence can be accelerated by appropriately choosing the schedule of parameters.

4.1 Learning the Structure

Consider the data generated from a triangular-spaced distribution as per Fig. 5. We assume that the *a priori* knowledge from the user permitted him to define a tree topology with a complete tree of depth 4, and where each node has exactly 3 children. This implies that the user's tree consists of $\sum_{i=0}^{4} 3^i = 1+3+9+27 = 40$ nodes, which is initialized as per the procedure explained in Sec. 3.1. Fig. 5a, depicts the position of the nodes of the tree after a random initialization. Random initialization was used by uniformly drawing points from the unit square. Observe that the original data points do not *lie in the curve*. Our aim here was to show how our algorithm could learn the structure of the data using *arbitrary* (initial and "non-realistic") values for the codebook vectors. The reader must appreciate that the initial random positioning completely obfuscates the tree structure since the input signals conform to a cloud of points within the unit square. However, once the main training loop becomes effective, the codebook vectors get positioned in such a way that they represent the structure of the data distribution, and simultaneously preserve the user-defined topology. This can be clearly seen from Fig.

[8] These data sets have been made available to the public, and can be found at `http://www.scs.carleton.ca/~castudil`.

5b and 5c which are snapshots after 1,000 and 10,000 samples, respectively. At the end of the training process (see Fig. 5d), the complete tree fills in the triangle formed by the cloud of input samples and seems to do it uniformly. The final position of nodes of the tree suggests that the underlying structure of the data distribution corresponds to the triangle. Observe that the root of the tree is placed roughly in the center (i.e. the mean) of the distribution. It is also interesting to note that each of the three main branches of the tree, cover the areas directed towards a vertex of the triangle respectively, and their sub-branches fill in the surrounding space around them.



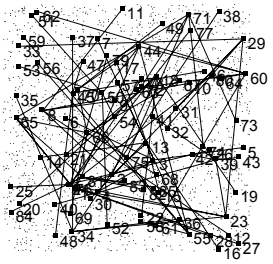
(a) After 0 iter. (b) 1,000

(c) 10,000 (d) 300,000

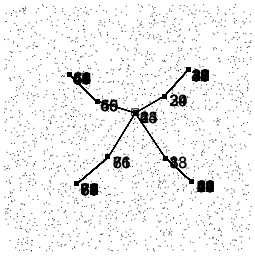
Fig. 5: TTO-SOM-based 3-ary tree topology learnt from a "triangular" distribution.

Another example is shown in Fig. 6, where the data was generated from a "square-shaped" distribution. Analogously, we presuppose that using the information at the disposal to the user, he defines a tree topology with a complete tree of depth 4, and where each node has exactly 4 children. As a consequence, the user's defined tree will be composed by $\sum_{i=0}^{4} 4^i = 1+4+16+64 = 85$ nodes, and is again initialized utilizing the procedure detailed in Sec. 3.1. Fig. 6a, portrays the location of the nodes within the tree after setting their values by randomly selecting points from the unit square. As we can see, the initial position of the codebooks based on the structure of the user-defined

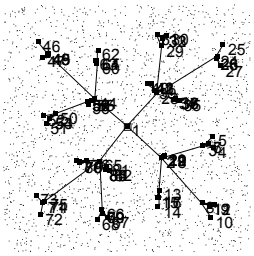
tree is "confusing", because of the random nature of the samples drawn from the unit square distribution. Convergence takes place by repeatedly executing the main training loop, and all the while the codebook vectors are moved until they represent the structure of the data distribution, and at the same time maintain the topology of the tree. This can be clearly seen from Fig. 6b and 6c which display the situation after 10,000 and 100,000 iterations, respectively. Fig. 6d depicts the situation after training. Observe that the position of the nodes appears to be uniform inside the square formed by the cloud of samples. In this case, the position of the nodes implicitly suggests that the underlying structure of the data distribution corresponds to the square. Similar to the triangle example, note that in this case, the root of the tree is placed near the center of mass of the square formed by the cloud of input signals. Also interestingly, the main branches of the tree cover the principal diagonals of the square, and their sub-branches spread through the space around these, respectively.



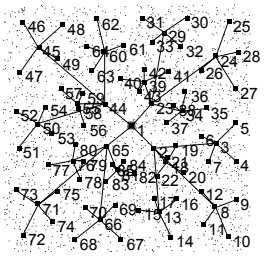
(a) After 0 iter.



(b) 10,000



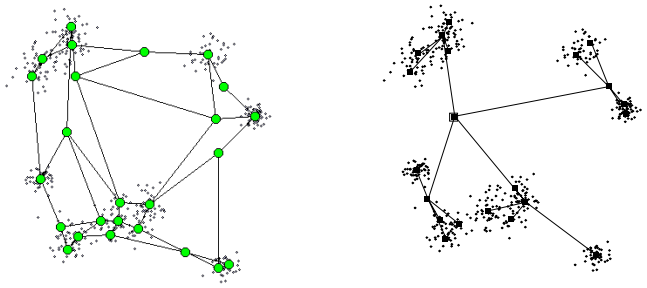
(c) 100,000



(d) 300,000

Fig. 6: TTO-SOM-based 4-ary tree topology learnt from a "square" distribution.

With the aim of comparing our method with the traditional SOM, we considered a discrete data distribution of 500 data points[9]. For this experiment an equivalent number of neurons where employed, The SOM utilized a 5×5 grid, i.e., 25 neurons, while the TTO-SOM employed a complete 4-ary tree of four level, i.e., $1 + 4 + 16 = 21 \approx 25$ neurons. The resulting maps demonstrate one of the known drawbacks of the SOM, namely the convergence of neurons in zero-density areas. Additionally, although the root of the tree trained by the TTO-SOM lays in a zero-density area, it is located roughly in the center of mass of the data distribution, and thus renders meaningful information about the *entire* data set. In our opinion, the TTO-SOM using a lesser number of connections yields a mapping which is more easy for a human being to "read". Note also that the nodes at second level of the tree lie in the centers of mass of the four biggest concentration of data points.



(a) SOM (b) TTO-SOM

Fig. 7: The result of invoking the SOM and TTO-SOM on discrete data distribution using an almost identical number of nodes.

To illustrate the properties of the TTO-SOM by incorporating other structures, we consider the scenario when the topology is unidirectional. Indeed, we obtain very impressive results when the tree structure is the 1-ary tree as seen in Fig. 8. In this case, the user-defined a list (i.e. a 1-ary tree) as the imposed topology. Initially, the codebook vectors were randomly placed as per Fig. 8a. Again, the reader must observe that at the beginning, the linear topology is completely lost due to the randomness of the data points. The migration and location of the codebook vectors after $10,000$ and $100,000$ iterations are given in Fig. 8b and 8c respectively. At the end of the training process the list represents the triangle very effectively, as also reported in [**?**].

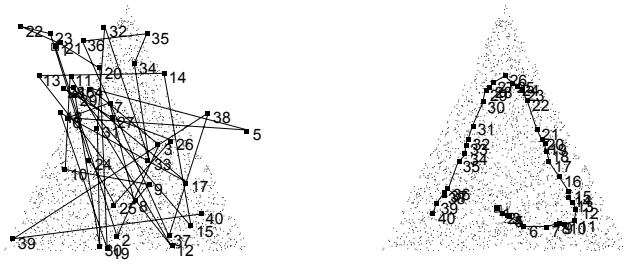
[9] The data set and the SOM training were extracted from the software DemoGNG version 1.5, available at the URL `http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html`.

For the case of using a 1-dimensional tree (i.e., a list) as per Fig. 8, our algorithm is capable of not only representing the whole distribution of the input samples, but also of preserving *this* "tree-like" topology. Indeed, on termination, the indices of the codebook vectors are arranged in an increasing order as seen in Fig. 8d. In this case, the one-dimensional list of neurons is evenly distributed over the triangle, preserving the original properties of the 2-dimensional object and also presenting a shape which reminds the viewer of the so-called Peano curve [27].

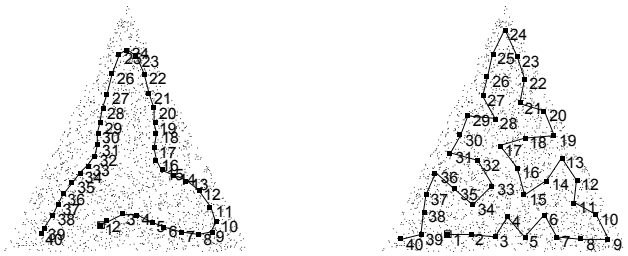
As we can see, this one-dimensional representation is achieved by defining a tree with a single child per node. This example can be effective in distinguishing our method over the ET. If the same data was processed using the ET, the final configuration of the codebook vectors will not represent the input set accurately, even if the topology is preserved. The reason for this is as follows: In the first stage of the algorithm, the ET's root is trained for a certain number of iterations, after which the root will be most likely close to the center of mass of the input data cloud. The ET then "freezes" the location of this unit, which implies that it is no longer trained, and thus *this* codebook vector remains static. It no longer participates in the competitive learning process since our assumption of using a 1-ary tree is in place. After freezing this unit, the ET resorts to a splitting operation. The children are created, and in this particular case only a single child is created and trained. In this phase, the child does not compete with the parent because the latter is static. Consequently, the random samples taken from the input set will again tend to place the child close to the center of mass of the data cloud. As this process is repeated for the subsequent children, all of them will be placed near to the center of the data distribution.

From this perspective, one of the best advantages of the ET is simultaneously a drawback. Freezing the unit after a certain time and then splitting it lends to the algorithm the ability to only train the leaf-nodes. This property is cleverly exploited by the authors of [26] so as to accelerate the search for the BMU in $O(\log n)$ time. On the other hand, the frozen units are excluded from the subsequent competition, and open the possibility of being ill-formed codebook vectors. Briefly put, the 1-ary tree case is an extreme one, and although for the general n-ary tree (for $n > 2$) the ET possesses a good performance, in the linear case our algorithm outperforms the ET.

Fig. 9 presents an example in a higher dimensional space, where the TTO-SOM attempts to learn a unit *sphere*. In this experiment, so as to display the capabilities of the algorithm, we decided to use the same 3-ary tree configuration that was utilized in the triangle-shaped distribution shown in Fig. 5, i.e. a complete 3-ary tree of four levels, thus containing 40 nodes. At the beginning, as shown in Fig. 9a, the initial codebook vectors are randomly distributed within the circumscribed unit cube. As the training proceeds, Fig. 9b depicts the situation where all the nodes converge into a single point which is located roughly in the center of the sphere. Fig. 9c, displays the case when the neurons belonging to lower levels of the tree begin to spread evenly inside



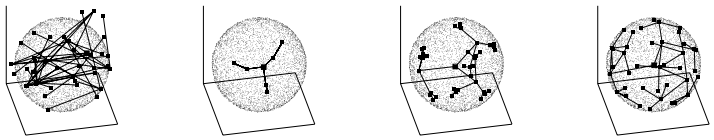
(a) After 0 iter. (b) 10,000



(c) 100,000 (d) 300,000

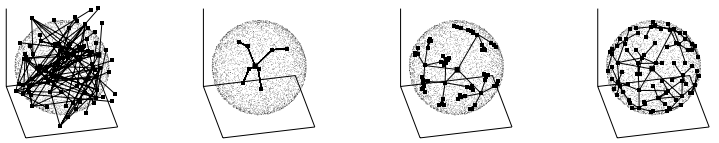
Fig. 8: TTO-SOM-based 1-ary tree (list) topology learnt from a "triangular" distribution

the sphere. The situation after training is shown in Fig. 9d, where the tree expands uniformly in the interior of the sphere, suggesting the spherical shape of the original data manifold. Given the symmetries of the sphere, other tree topologies can also be utilized to learn its structure.



(a) After 0 iter. (b) 10,000 (c) 100,000 (d) 300,000

Fig. 9: The same 3-ary tree utilized for learning the triangle in Fig. 5 now learns the unit sphere in 3-dimensions.



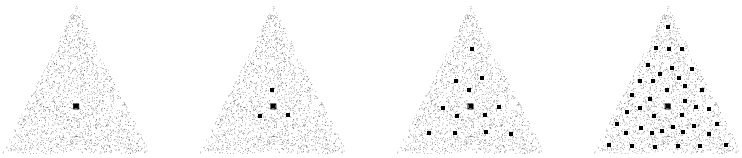
(a) After 0 iter. (b) 10,000 (c) 100,000 (d) 300,000

Fig. 10: The same 4-ary tree utilized for learning the unit square in Figure 6, now learns the unit sphere in 3-dimensions.

4.2 The Hierarchical Representation

Another distinct advantage of the TTO-SOM, which is not possessed by other SOM-based networks, is the fact that it has hologram-like properties. In other words, although the entire tree specified by the user can describe the cloud of data points as per the required resolution, the same cloud can be represented with a more coarse resolution by using a lesser number of points. Thus, if we wanted to represent the distribution using a single point, this can be adequately done by just specifying the root of the tree. A finer level of resolution will include the root and the second level, where these points and their corresponding edges, will represent the distribution and the structure. Increasingly finer degrees of resolution can be obtained by including more levels of the tree. We believe that this is quite a fascinating property.

To clarify this, consider the triangular distribution in Fig. 11, which is the same distribution of Fig. 5. Fig. 11a shows how the cloud can be represented by a single point, i.e. the root. In Fig. 11b it is represented by 4 nodes which are the nodes up to the second level. If we use the user defined tree of four levels, the finest level of resolution will contain all the 40 nodes, as displayed in Fig. 11d.



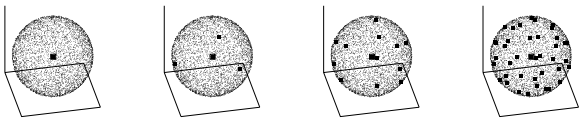
(a) Level 1 (b) Level 2 (c) Level 3 (d) Level 4

Fig. 11: Multi-level resolution of the results shown in Fig. 5.

To demonstrate the analogous effect in 3-dimensions, we similarly, consider the spherical distribution depicted in Fig. 12, which corresponds to the

distribution of Fig. 6. When the cloud is represented by only one point, i.e. the root, a rough approximation of the data distribution can be obtained, as per Fig. 12a, which shows the root of the tree located roughly in the center of the sphere. In Fig. 12b the entire distribution is represented by 4 nodes which corresponds to the nodes in the first and second levels of the tree. If we include nodes up to the fourth level of the tree, the representation at this resolution will contain all the 40 nodes, as displayed in Fig. 12d.

From these results, we believe that the representations obtained for increasing resolutions are both intriguing and remarkable.



(a) Level 1 (b) Level 2 (c) Level 3 (d) Level 4

Fig. 12: Multi-level resolution of the results shown in Fig. 9 for a sphere in 3-dimensions.

4.3 Skeletonization

Intuitively, the objective of skeletonization is to construct a simplified representation of the global shape of an object. In general, such a skeleton is expected to contain much less points than the original image and should be a thinned version of the original shape. According to the authors of [24], skeletonization in the plane is the process by which a 2-dimensional shape is transformed into a 1-dimensional one, similar to a "stick" figure. In this way, skeletonization can be seen as a dimensionality reduction technique that captures local object symmetries and the topological structure of the object. This problem has been widely investigated in the fields of PR and computer vision.

There are different types of algorithms that attempt to solve the skeletonization problem. Traditional skeletonization approaches assume the connectivity of the pixels that constitute the image. Thus, they are inadequate when pixels in the image lack connectivity, as can happen all too often, as a consequence of an inappropriate manipulation of the image, noise, or the intrinsic nature of the data itself. In such cases, traditional methods may not perform well, and advanced techniques that can also inherently process *structure*, are needed.

SOM variations have been used to tackle this situation when points are sparse in the space [8, 32]. In [8] the authors used a GNG-like approach,

while in [32], the authors recommend the use of a Minimum Spanning Tree which is calculated over the *positions* of the codebook vectors, followed by a post-refinement phase that adds and deletes edges.

We now advocate a completely different SOM-like strategy, namely the TTO-SOM. Indeed, as we perceive it, the structure generated by using the TTO-SOM can be viewed as an endo-skeleton of the given data set, in the sense that it conforms to an internal framework that support the "soft parts" of the original object. This skeleton meets at pseudo-joints, which are, in our representation, the nodes of the tree. As the whole structure of these pseudo-bones are dependent on the position of the nodes in the feature space, a single learning iteration of the TTO-SOM is capable of affecting the overall shape of the skeleton, and on convergence, it will self-organize so as to assimilate the fundamental properties of the primary representation.

In this context we propose that the edges can be seen as pseudo-bones. Each pseudo-bone is defined by two codebook vectors in their extremes, and contrary to what happens with real bones, these pseudo-bones have a great measure of flexibility, and also the ability to contract or enlarge as a consequence of the movement of the nodes that define their respective extrema. It is also worth mentioning that the movement of a joint will have the implication of the modification of at least one edge. Thus, when a node is moved, all the edges associated with the children and parent will change accordingly, modifying the shape of the inferred skeleton. The difference between using the SOM-like philosophy [32] and the TTO-SOM lies exactly here. A SOM-like algorithm will change the edges of the skeleton but *only* as the algorithm dictates as per the MST computed over the nodes and their distances in the "real" world; i.e., the feature space. As opposed to this, a TTO-SOM like structure can modify the skeleton as dictated by the particular node in question, but also, *all* the nodes tied to it by the bubble of activity, as dictated by the *user defined tree*, i.e., the link distances.

The reader can appreciate, in Fig. 13, the original silhouette of a rhinoceros, a 2-dimensional person as well as a 3-dimensional person. All three objects were processed by the TTO-SOM using exactly the same tree structure, the same schedule for the parameters, and without any post processing of the edges. From the images at the lower level of Fig. 13 we observe that, even without any specific adaptation, the TTO-SOM is capable of representing the fundamental structure of the three objects in a "1-dimensional" way effectively. The figures at the second level display the neurons without the edges. In this case, it can be seen that our algorithm is also capable of granting an intuitive idea of the original objects by merely looking at the points.

A potentially interesting idea is that of mixing the hierarchical representation of the TTO-SOM presented in Sec. 4.2 with its skeletonization capabilities. We propose that in this case, the user will be able to generate different skeletons with different levels of resolution, which we believe, can be used for managing different levels of resolution at a low computational cost for applications in the fields of geomatics, medicine and video games.

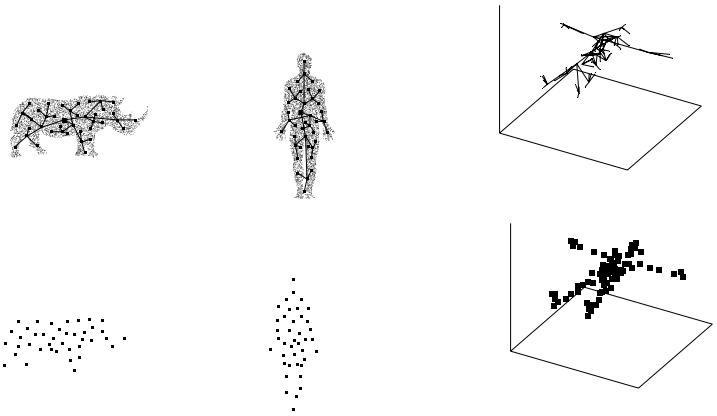


Fig. 13: The result of a skeletonization process for the silhouettes of various shapes using the TTO-SOM, namely, of a rhinoceros, a 2-dimensional person, as well as a 3-dimensional person.

4.4 Clustering and Pattern Recognition Capabilities

Additionally, we have examined the performance of the TTO-SOM in clustering and PR applications for real-world data. The well known Iris dataset[10] was chosen for this purpose. The data set gives the measurements (in centimeters) of the variables which are the sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of the iris family. The species are the Iris Setosa, Versicolor, and Virginica.

Since the TTO-SOM is an unsupervised learning algorithm, in order to show the power of our method, we decided to invoke it using exactly the same configuration employed to learn the triangle and sphere displayed in Figures 5 and 9 respectively, i.e., the same underlying tree topology of a complete 3-ary tree of depth 4. By this we attempt to show examples of how exactly the *same* tree configuration can be utilized to learn the structure from data belonging to the 2-dimensional, 3-dimensional and also 4-dimensional spaces. After executing the TTO-SOM, each of the main branches of the tree were migrated towards the center of mass of the cloud of points in the hyperspace belonging to each of the three categories of flowers, respectively. It is important to mention that the TTO-SOM did not know that the data came from 3 different categories, rather this information was inferred by it. We find this result quite fascinating, indeed!

After convergence, each of the samples were associated to the closest neuron, thus generating a set of clusters that are arranged in a hierarchical man-

[10] The Iris dataset was obtained from The R Project for Statistical Computing, available at the URL `http://www.r-project.org/`

ner. Moreover, if the true labels of the original data set is provided, the TTO-SOM can be further utilized to perform pattern recognition, by identifying each of the neurons as a representative of a particular class. The results of this classification scheme applied to the Iris data set are summarized in Table 1. The information about each category of the Iris families is given in the first row. Analogously, each column considers one of the three main branches of the tree. The table indicates how many elements of a particular kind of Iris were best represented by a particular branch of the tree. For instance, all the data points which were best represented by neurons belonging to Branch1 happened to be of class Iris Setosa. Moreover, all the 50 instances of Iris Setosa found in the data set are represented by Branch1. The last Column of Table 1 summarizes the accuracy of recognizing a particular category of Iris.

	Setosa	**Versicolor**	**Virginica**	**Accuracy**
Branch1	50	-	-	1.00
Branch2	-	48	2	0.96
Branch3	-	2	48	0.96

Table 1: The power of the TTOSOM for learning the characteristics of the Iris dataset. The underlying tree structure was exactly the same one employed for learning the triangle and the spheres of Figures 5 and 9, respectively.

The experimental results shown in Table 1, not only demonstrate the potential capabilities of the TTO-SOM for performing clustering, but also suggest the possibilities of using it for pattern classification. According to [10], there are several reasons for performing pattern classification using an unsupervised approach. Moreover, due to the multi-resolution nature of our proposed strategy, different levels of the tree may be employed for performing increasingly accurate classification tasks. We are currently investigating such a classification strategy.

4.5 Theoretical Analysis

Although the applications of the new method have been demonstrated, we feel that it is appropriate to close this section by mentioning the hurdles that will be encountered in analyzing the asymptotic distribution of the TTO-SOM[11]. In all brevity, we believe that even the tools for such an analysis have not been fully developed, as we explain below.

[11] We are very grateful to an anonymous Referee who pointed us to these references, and for the insight that he provided. We totally concur with him in that it is difficult (if not currently impossible) to mathematically solve the explicit equations for the final distribution and topology for our present solution.

At the outset, we mention that Bauer *et al* [3] explain, in great detail, a tool called the Topographic Product, utilized for the measurement of the topology preservation. These authors also show the power of this tool by applying it on different artificial and real-world data sets, and also compare it with different measures to quantify the topology [2]. Their study concentrates on the traditional SOM, implying that the topologies evaluated were of a "linear" nature, with the consequential extension to 2-dimensions and 3-dimensions by means of grids only. In [16], Haykin mention that the Topographic Product may be employed to compare the quality of different maps, even when these maps possess different dimensionality. However, he also noted that this measurement is only possible when the dimensionality of the topological structure is the same as the dimensionality of the feature space. Further, tree-like topologies were not considered in their study. To be more precise, most of the effort towards determining the concept of topology preservation for dimensions greater than unity are specifically focused on the SOM [2, 3, 5, 7, 17, **?**], and do not define how a tree-like topology should be measured nor how to define the order in topologies which are not grid-based. Thus, we believe that even the tools to analyze the TTO-SOM are currently not available. The experimental results obtained in our paper, suggest that the TTO-SOM is able to train the NN so as to preserve the stimuli. However, in order to quantify the quality of this topology, the matter of defining a concept of ordering on tree-based structure has yet to be resolved. Although this issue is of great interest to us, this rather ambitious task lies beyond the scope of our present manuscript.

5 Conclusions

In the paper we have proposed a schema called the Tree-based Topology-Oriented SOM (TTO-SOM) by which the operator/user is able to impose an *arbitrary*, user-defined, tree-like topology onto the codebook vectors of a SOM. This constraint leads to a neighborhood phenomenon based on the user-defined tree, and, as a result, the so-called bubble of activity becomes radically different from the ones studied in the previous literature. The map learnt as a consequence of training with the TTO-SOM is able to determine *both* the distribution of the data and its structured topology, interpreted through the perspective of the user-defined tree. In addition, we have shown that the TTO-SOM revealed the ability to represent the original data set in multiple levels of granularity, and this was achieved without the necessity of computing the entire tree again. Lastly, we discussed the capability of the TTO-SOM to extract an skeleton, which is a "stick-like" representation of the image in a lower dimension of space. All these properties have been confirmed by numerous experiments on a diversity of data sets.

References

1. E. Arsuaga Uriarte and F. Díaz Martín. Topology preservation in SOM. *International Journal of Applied Mathematics and Computer Sciences*, 1(1):19–22, 2005.
2. H. U. Bauer, M. Herrmann, and T. Villmann. Neural maps and topographic vector quantization. *Neural Networks*, 12(4-5):659 – 676, 1999.
3. H. U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *Neural Networks*, 3(4):570–579, July 1992.
4. J. Blackmore and R. Miikkulainen. Incremental grid growing: Encoding high-dimensional structure into a two-dimensional feature map. In *Proc. ICNN'93, International Conference on Neural Networks*, volume I, pages 450–455, Piscataway, NJ, 1993. IEEE Service Center.
5. M. Budinich. On the ordering conditions for self-organizing maps. *Neural Computation*, 7(2):284–289, 1995.
6. Sung-Bae Cho. Ensemble of structure-adaptive self-organizing maps for high performance classification. *Information Sciences*, 123(1-2):103 – 114, 2000.
7. P. L. Conti and L. De Giovanni. On the mathematical treatment of self organization: extension of some classical results. In *Artificial Neural Networks - ICANN 1991, International Conference*, volume 2, pages 1089–1812, 1991.
8. A. Datta, S. M. Parui, and B. B. Chaudhuri. Skeletal shape extraction from dot patterns by self-organization. *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, 4:80–84 vol.4, Aug 1996.
9. M. Dittenbach, D. Merkl, and A. Rauber. The growing hierarchical self-organizing map. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 6, pages 15–19 vol.6, 2000.
10. R. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.
11. B. Fritzke. Unsupervised clustering with growing cell structures. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume ii, pages 531–536 vol.2, Jul 1991.
12. B. Fritzke. Growing Grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995.
13. B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632, Cambridge MA, 1995. MIT Press.
14. B. Fritzke. Some competitive learning methods. Last accessed on may 15th, 2009. Available on the web at http://citeseer.ist.psu.edu/fritzke97some.html, 1997.
15. L. Guan. Self-organizing trees and forests: A powerful tool in pattern clustering and recognition. In *Image Analysis and Recognition, Third International Conference, ICIAR 2006, Póvoa de Varzim, Portugal, September 18-20, 2006, Proceedings, Part I*, pages I: 1–14, 2006.
16. S. Haykin. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition edition, 2008.
17. G. Huang, H. A. Babri, and H. Li. Ordering of self-organizing maps in multidimensional cases. *Neural Computation*, 10:19–24, 1998.
18. I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

19. Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1981–1997. *Neural Computing Surveys*, 1:102–350, 1998.
20. K. Kiviluoto. Topology preservation in self-organizing maps. In P. IEEE Neural Networks Council, editor, *Proceedings of International Conference on Neural Networks, ICNN'96*, volume 1, pages 294–299, New Jersey, USA, 1996.
21. P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. *IJCNN International Joint Conference on Neural Networks*, 2:279–284, June 1990.
22. M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. In *in Proceedings of International Conference on Articial Neural Networks*, volume I, pages 397–402, North-Holland, Amsterdam, 1991.
23. D. Merkl, S. Hui-He, M. Dittenbach, and A. Rauber. Adaptive hierarchical incremental grid growing: An architecture for high-dimensional data visualization. In *In Proceedings of the 4th Workshop on Self-Organizing Maps, Advances in Self-Organizing Maps*, pages 293–298, 2003.
24. R. L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.
25. Merja Oja, Samuel Kaski, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156, 2003.
26. J. Pakkanen, J. Iivarinen, and E. Oja. The Evolving Tree — a novel self-organizing network for data analysis. *Neural Processing Letters*, 20(3):199–211, December 2004.
27. G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36(1):157–160, 1890.
28. M. Pöllä, T. Honkela, and T. Kohonen. Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. *Neural Computing Surveys*, forthcoming, 2007.
29. G. Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In Ján Paralič, Georg Pölzlbauer, and Andreas Rauber, editors, *Proceedings of the Fifth Workshop on Data Analysis (WDA'04)*, pages 67–82, Sliezsky dom, Vysoké Tatry, Slovakia, June 24–27 2004. Elfa Academic Press.
30. A. Rauber, D. Merkl, and M. Dittenbach. The Growing Hierarchical Self-Organizing Map: exploratory analysis of high-dimensional data. *IEEE Transactions on Neural Networks*, 13(6):1331–1341, 2002.
31. R. Rojas. *Neural networks: a systematic introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1996.
32. R. Singh, V. Cherkassky, and N. Papanikolopoulos. Self-Organizing Maps for the skeletonization of sparse shapes. *Neural Networks, IEEE Transactions on*, 11(1):241–248, Jan 2000.
33. J. Venna and S. Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *ICANN*, volume 2130 of *Lecture Notes in Computer Science*, pages 485–491. Springer, 2001.
34. Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. In *8th International Workshop on Artificial Neural Networks, IWANN 2005*, pages 758–770, 2005.