2

SOMs for machine learning

Iren Valova, Derek Beaton and Daniel MacLean
University of Massachusetts Dartmouth
USA

1. Introduction

In this chapter we offer a survey of self-organizing feature maps with emphasis on recent advances, and more specifically, on growing architectures. Several of the methods are developed by the authors and offer unique combination of theoretical fundamentals and neural network architectures. Included in this survey of dynamic architectures, will also be examples of application domains, usage and resources for learners and researchers alike, to pursue their interest in SOMs.

The primary reason for pursuing this branch of machine learning, is that these techniques are unsupervised – requiring no a priori knowledge or trainer. As such, SOMs lend themselves readily to difficult problem domains in machine learning, such as clustering, pattern identification and recognition and feature extraction. SOMs utilize competitive neural network learning algorithms introduced by Kohonen in the early 1980's. SOMs maintain the features (in terms of vectors) of the input space the network is observing. This chapter, as work emphasizing dynamic architectures, will be incomplete without presenting the significant achievements in SOMs including the work of Fritzke and his growing architectures.

To exemplify more modern approaches we present state-of-the art developments in SOMs. These approaches include parallelization (ParaSOM – as developed by the authors), incremental learning (ESOINN), connection reorganization (TurSOM – as developed by the authors), and function space organization (mnSOM). Additionally, we introduce some methods of analyzing SOMs. These include methods for measuring the quality of SOMs with respect to input, neighbors and map size. We also present techniques of posterior recognition, clustering and input feature significance. In summary, this chapter presents a modern gamut of self-organizing neural networks, and measurement and analysis techniques.

2. Overview of competitive learning

2.1 Unsupervised and competitive learning

Very broadly defined, neural networks learn by example and mimic human brain in its decision or object identification capabilities. The concept of artificial neural networks (ANN) is based on two different views of the human brain activity, both of which rely on the functionality of a single neuron. The neurons are perceived as adding devices, which react,

or fire, once the incoming signals sum reaches a threshold level. Fig.1 illustrates the functionality of a single neuron, which receives signals from other neurons it is connected to via weighted synapses. Upon reaching the firing level, the neuron will broadcast a signal to the units connected to its output.

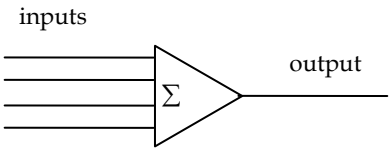

Fig. 1. Artificial neuron functionality

Returning to the two major types of ANN, one view generally relies on the individual neuron and its ability to respond, or fire, given sufficient stimulus. The topology of neurons and the connections among them is not the goal of this type of ANN, but rather the output produced by the respective neuron.
The second view banks on the neurons functioning as a team. As such, it takes into account the concept of map formed by neuron positions, much like the visual cortex map, producing a two dimensional image of the perceived visual field. This type of ANN produces a topology of neurons, connected by weighted synapses and features the natural grouping of the input data (Fig.2). This translates into input density map and necessitates the development of evaluation procedures on the formed clusters for the purpose of identifying or matching patterns in the data.

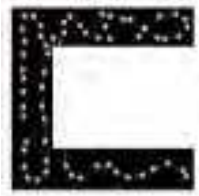

Fig. 2. The black area denotes input space distribution, where the neurons have organized to cover that input topology as a team

The taxonomy of learning methods and algorithms for ANN is multifaceted and includes many hierarchical classifications (Fig.3). In this chapter we are concerned with unsupervised learning that is also competitive. Learning in ANN is the process of connection weight adjustment, which, in turn guides the neuron to a better position in terms of input data configuration.
In the case of supervised learning, the weight adjustment will be guided by the teaching signal and the penalty/reward of the error in the ANN response. Unsupervised learning methods do not benefit from teacher signal guidance. The neurons compete to match the input as closely as possible, usually based on Euclidean distance. The neuron closest to the considered input exemplar is the winner taking it all, i.e. adjusting its weight to improve its position and thus move closer to the input.

We are describing the extreme case of competition, i.e. winner-take-all. Depending on the learning algorithm and the ANN application, while the winning neuron will be selected, a neighbourhood of influence may be established, whereby the winning neuron neighbours will also move in the same direction albeit at a lesser distance.

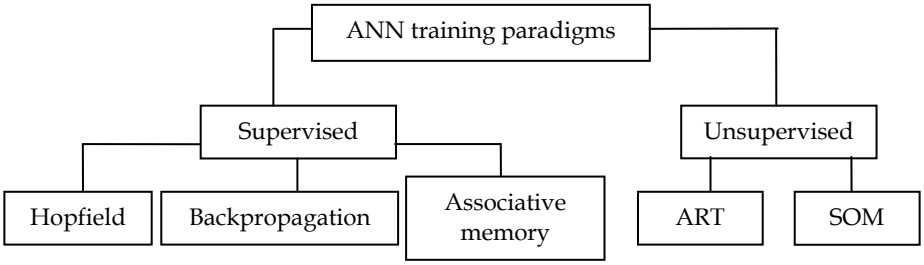

Fig. 3. Taxonomy of ANN learning methods

Unsupervised learning (UL) is generally based on competition. UL seeks to map the grouping or patterns in the input data. This can be accomplished either by neurons resonating with the input exemplar (e.g. adaptive resonance theory) or by neurons winning the distance from the input exemplars competition. It must be noted that there are ANN models that learn in unsupervised manner, but are not based on competition. Among those, the principal component network should be mentioned here as a prelude to later sections in this chapter.

2.2 Kohonen's SOM

The brains of higher animals are organized by function, e.g. the visual cortex processes the information received through the optical nerve from the eyes, the somatosensory cortex maps the touch information from the surface of the body, etc. Inspired by the mapping abilities of the brain, the self-organizing feature map (SOM) was introduced in early 1980's by Teuvo Kohonen (Kohonen, 1995). SOMs are used to topologically represent the features of the input based on similarity usually measured by Euclidean distance. SOMs are useful tools in solving visualization and pattern recognition tasks as they map a higher dimension input space into a one- or two-dimensional structure. SOMs are initialized usually randomly (Fig.4b), in a topology with fixed number of neurons, that can be ordered in a chain (i.e. each neuron has at most two neighbors) or in a two-dimensional grid of rectangular (Fig.4c) or hexagonal nature, where the neurons have at most four neighbors.

Fig. 4. Input space: a) distribution; b) with randomly initialized neurons; c) two-dimensional rectangular grid

Before a brief overview of the SOM algorithm, let us take the reader through the concept of Kohonen's ANN. The input space of Fig.4a is used along with the random initialization in Fig.4b. As every input vector (in this case a two-dimensional *x, y* representation of the location of each dot comprising the black area in Fig.4a) is presented to the network, the closest neuron responds as a winner of the Euclidean distance-based competition and updates its weight vector to be closer to the input just analyzed. So do its neighbors dependent on the neighborhood radius, which can be reduced as the time progresses. The rate of reduction is determined by the learning rate. As inputs are presented in a random order, the neurons move closer to the input vectors they can represent and, eventually, the movement of neurons becomes negligible. Usually, that is when the map is considered to have converged. This process is illustrated in Fig.5a for one-dimensional SOM and Fig.5b for rectangular SOM.

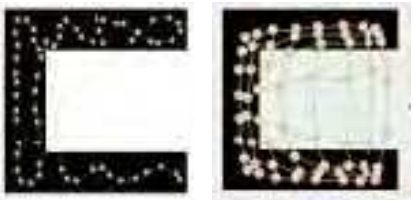

Fig. 5. Converged SOM: a) one-dimensional topology; b) two-dimensional topology

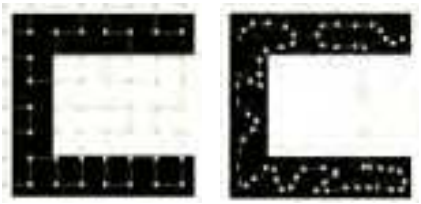

Fig. 6. Hilbert curve initialization approach: a) initialized network; b) resulting map

While the classic SOM features random initialization of the weight vectors, the authors have demonstrated the advantages of one-dimensional SOM initialization based on a Hilbert curve (Buer, 2006; Valova, Beaton, & MacLean, 2008a) with neurons positioned in a chain following the space-filling curve (Fig.6a). Kohonen posited that one-dimensional SOM converge in a shape resembling Peano curves. The authors followed this observation and utilized the idea in the initialization process to speed up convergence and ensure linearity of the network. Fig.6b demonstrates the resulting map. It is obvious that the map is not tangled, and the neurons that are physical neighbors also represent topologically close input vectors unlike the map on Fig.5a, which is tangled and topologically close neighbors do not always share physical proximity.

The algorithm can now be formalized. Each neuron in the network has a weight or reference vector $\xi = [x_1, x_2, ..., x_m]$ where x_i is an individual attribute of ξ. The neurons are gradually organized over an n-dimensional input space V^n. Each input $\vartheta = [\mu_1, \mu_2, ..., \mu_n] \in V^n$, where μ_i is an attribute in ϑ, has the same number of attributes as the weight vector ξ of each neuron in the network.

Once the network is initialized, the input is presented sequentially to the network. The best-matching unit in the network is determined by comparing each neuron to the current input based on Euclidean distance, with the winner being the neuron closest to the input.

$$c = \arg_i \min\{\xi_i - \mu_i\} \tag{1}$$

where c is the best-matching unit.
The winning neuron and all neurons falling within the neighborhood radius for an iteration i update their reference vectors to reflect the attraction to ϑ. To change the reference vector of a neuron, the following equation (which is a very common SOM learning equation) is used:

$$\xi(t+1) = \xi(t) + h(t)\alpha(t)(\vartheta - \xi) \tag{2}$$

where t is the current iteration and α (t) is a monotonically decreasing learning rate, and h(t) is the neighborhood function.

2.3 Visualizing a SOM

In low dimensional patterns, such as one- or two-dimensional, the input and the SOM can be visualized by using positions of pixels. However, when scaling to three, or five dimensions, pixels can be used for dimensions that represent two-dimensional space, but the remaining one or three attributes in this case would be represented by gray scale or RGB, respectively. This, however, implies that the visualization of the data, and the SOM can be expressed in x, y and color values.
When dealing with complex data that is not directly visualized, or even very high dimensional data (e.g. greater than 10) visualization becomes an issue. One of the methods used for showing the lattice structure style network for high dimensional data, is to use a dimensionality reduction or dimensionality mapping technique. One of the simplest, and most well known is Sammon's mapping (Eq.3) (Sammon 1969).

$$\left(\sum_{(i,j)\in A^2}^{i\neq j} d_A(i,j) \right)^{-1} \sum_{(i,j)\in A^2}^{i\neq j} \frac{(d_A(i,j) - d_V(w_i,w_j))^2}{d_A(i,j)} \tag{3}$$

This mapping measure effectively takes the distances between high dimensional objects, e.g. neurons, and allows them to be plotted in two-dimensions, so the researcher may visually inspect the geometric relations between neurons. Often, when the number of inputs is very high, or patterns are non-distinct and complex, the visualization does not include input data, rather, just the neurons.
To form the lattice structure correctly, the connections between known neighbors should be illustrated. The neurons are often recorded in one- or two-dimensional arrays, to allow the physical neighbors of each neuron to be recorded.

3. Growing Self-organizing Algorithms

3.1 Fritzke's growing SOM variants

Self-organizing Maps, as introduced by Kohonen, are static sized network. The obvious disadvantage to the predetermined number of neurons is that number is either not high enough to adequately map the input space or too high, thus leaving many neurons underutilized. SOM being trained without supervision, is not expected to know the input space characteristics apriori. Hence, a topology which allows the addition/removal of neurons, is a logical step in the development of self-organization. Fritzke introduced three architectures in the 1990's - growing grid (GG), growing cells (GC), and growing neural gas (GNG) (Fritzke, 1992, 1993a, b, c, 1994, 1995). All three start with minimal number of neurons and add neurons as needed. The need is based on an age parameter, while an error parameter determines which neuron will be joined by a new neighbor (GC, GNG) or a new set of neighbors (GG).

In the case of GC, once a neuron with the highest error value is selected, its longest connection (or edge) is replaced by two edges and a neuron is added to facilitate their connection (Fig.7).

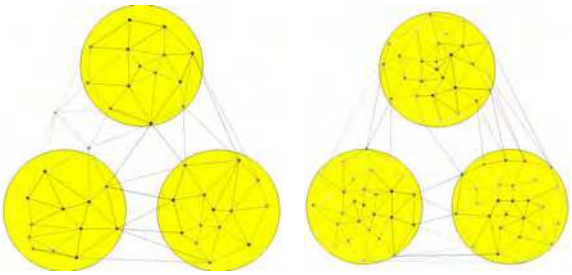

Fig. 7. Growing cells at: a) 3333 iterations; b) 15000 iterations

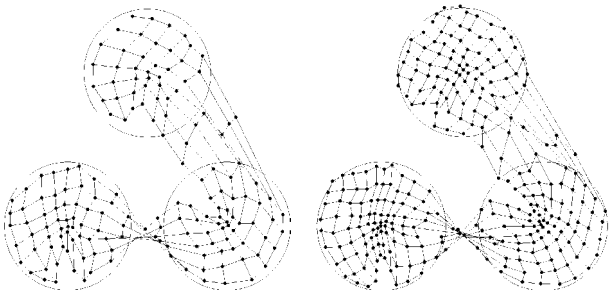

Fig. 8. Growing grid at: a) 33333 iterations; b) 100000 iterations

GG also utilizes an age parameter for the neurons. Every time a neuron wins, its age is incremented. At intervals, the neuron with the most wins is slated to receive new neighbors and the furthest direct topological neighbor from the selected neuron is found. GG utilizes a rectangular grid topology, which is maintained during growth. If these two neurons are in the same row of neurons, a column will be added between the two – thus affecting neurons

in other rows. If the selected neurons are in the same column, then a new row will be added between them – thus affecting neurons in other columns (Fig. 8).
In the third Fritzke contribution - the GNG - a similar concept to GC is observed. However, the connections between the neurons are assigned the age parameter. Therefore, GNG adds and removes connections, based on neuron selection. GNG is also capable of attaining configurations with multiple networks (Fig. 9).
One of the major drawbacks to GNG, as well as other growing methods, is that the number of nodes are ever increasing. This is, in part, compensated for by an extension to GNG, Growing neural gas with utility (GNG-U), which features removal of neurons based on the criterion of low probability density in the underlying input space "beneath" the neuron. The downfalls to these methods are than they do not exhibit incremental learning – a problem that is discussed in a later section on ESOINN.

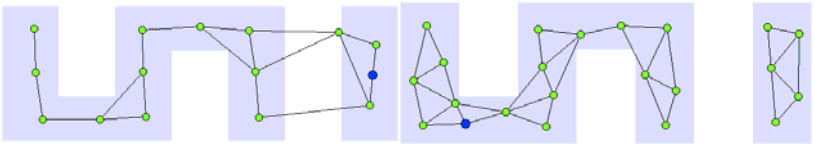

Fig. 9. Provided by (DemoGNG); At 7500 iterations (left) the map is starting to take the form of the input; At 12000 iterations (right) the map is a much better topological representation of the given input.

Fritzke's growing methods can be explored by the reader interactively at the DemoGNG website (DemoGNG), provided by the Institut für Neuroinformatik.

3.2 ParaSOM

The ParaSOM is architecture developed by the authors (Valova, Szer, Gueorguieva, & Buer 2005), which shares several similar attributes with the classic SOM and the growing architectures developed by Fritzke. However, it improves the effectiveness of the network and allows for different approaches to cluster analysis and classification, which will be demonstrated in later sections. The unique characteristics of the architecture include: parallel input processing, the adoption of cover region to manage the influence area of a neuron, and network growth, which is inspired by the GC architecture.
The ParaSOM is designed to process the entire input space in parallel. The classic SOM presents the network with one input at a time and determines the winner to move closer to that input. With the ParaSOM, however, the entire input space is presented to the network at the same time. Therefore, multiple neurons can adapt to nearby inputs independently of other neurons. This trait helps the network in recognizing patterns that it has already learned. For instance, imagine the network adapted itself to a particular pattern A, and another pattern B, that is very similar to A, is presented to the network. Because the neurons process the input space independently of each other, the ones that are already covering the pattern (most of them, since A is very similar to B) will not move. As a result, adapting to B is much faster, as the network only needs to learn the small differences between A and B.
ParaSOM features the following parameters that are unique to the architecture. The network produces a cover matrix, which represents the approximation of an input pattern made by

the network in its current state. The cover matrix calculation is based on the cover region, which is an attribute of every neuron. This is the area surrounding the point in space where the neuron exists and is considered to be the region of space that the neuron is covering. The move vector is another neuron attribute, which indicates the amount a neuron should move and in which direction is calculated, and added to the neuron's reference (weight) vector. The age parameter, which represents for how long a neuron has been well or poorly positioned is closely related to the inertness of a neuron. The inertness is the measure of how effectively a neuron covers the input space. Both, age and inertness determine when and where the network should grow or contract.
The ParaSOM in action is illustrated in Fig.10. The network is initialized with a minimal number of neurons, which have large cover regions (denoted by the large circles in Fig.10a). As the input space is shown to the network, the move vectors are calculated along with the other parameters gauging the performance of individual neurons (i.e. age, inertness, cover region) and the neurons are moved, new ones are added or some are removed in order to achieve the final comprehensive coverage illustrated in Fig.10c.

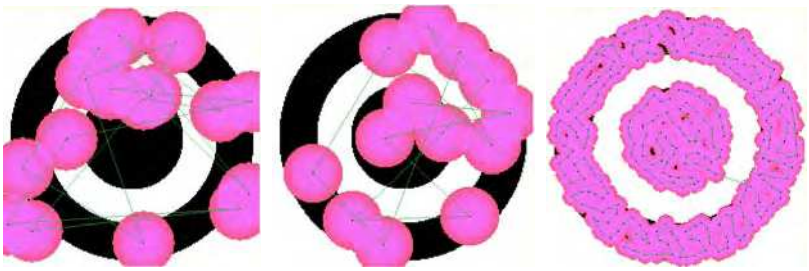

Fig. 10. ParaSOM in action: a) randomly initialized with a predetermined number of neurons; b) the network position at iteration 25; c) the network at iteration 500

The formalization of the algorithm will begin with the cover matrix which consists of subtracting a set of cover regions from the input space. Each such region is associated with a single neuron, and represents the contribution of the neuron to the total approximation. The job of the cover region is to weaken the signals of the inputs that are being covered, where the weaker the signal, the better the neuron is covering that region. Each cover region also maintains a radius which decreases in each epoch. The cover matrix is formulated as

$$C = V^m - \sum_{i=1}^{s} \theta_i \tag{4}$$

with the cover region calculated as

$$\theta_i(x_j) = \begin{cases} f_{m_i}(x), if \left\| x_j - m_i \right\| < threshold \\ 0, \quad else \end{cases}, \ \{1 \le j \le k\} \tag{5}$$

where the modified Gaussian cover function is defined as

$$f_{m_i}(\mathrm{x})=\exp\left[-\frac{\left[\left(\mu_1-x_1\right)^2+\cdots+\left(\mu_n-x_n\right)^2\right]^2}{\lambda}\right] \tag{6}$$

for radius λ and μ being an attribute of the input vector.
The cover function is utilized in the computation of a cover value, which indicates how well the neuron is covering the inputs and is calculated as

$$c_i = C_i \cdot f_{m_i} \tag{7}$$

where the local cover matrix is represented by

$$C_i = \left(C+\theta_i\right) \tag{8}$$

The inertness, is an indicator as to whether the neuron should be moved, removed, or is in a place where new neurons ought to be added nearby. The lower the inertness the better the coverage and hence position of the neuron. The inertness is given by

$$\varphi_i = c_i / c_{\max} \tag{9}$$

where the cover value max is calculated by

$$c_{\max} = \int_{V^m} f_{m_i}(x)dx \tag{10}$$

The network utilizes the inertness to determine whether and where to grow/shrink. A high inertness indicates that the neuron is well positioned and should not move (or move very little), while a low inertness indicates poor positioning and greater neuron movement. Inertness is one of two components that dictate network growth, with age being the second. Each neuron has an age attribute that. When a neuron is well positioned, as determined by a high-inertness threshold, its age is incremented. The same is true with a poorly positioned neuron having its age decremented based on a low-inertness threshold. When a neuron is well (or poorly) positioned for a sufficient period of time, it becomes a candidate to have a neuron added as a neighbor, or to be removed from the network, respectively.
Finally, the move vector, which indicates the amount a neuron should move and in which direction is calculated, and added to the neuron's reference vector. The attractions also affect immediate neighbors of the neuron but to a lesser degree where the amount of movement is proportional to the distance between the neuron and neighbor. The move vector $v_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ consists of components $v_{ik} = \int_{V^m} C^i \cdot f_{m_i}(x_k)dx$.

The authors have explored the effect a Hilbert initialization has on ParaSOM. As with the classic SOM, this network is also positively influenced by this mode of initialization. Fig.11 shows some results. Fig.11a features the network at iteration 0, using the same input topology as Fig.10. Fig.11b illustrates the intermediate result at iteration 25, and Fig.11c illustrates the final converged state of ParaSOM at iteration 500, same as the iteration in

Fig.10c. The last point is made to focus the reader attention to the tangled state of the randomly initialized network in Fig.10c. The Hilbert initialization, as the same iteration, features untangled, well-organized network.

Fig. 11. ParaSOM in action: a) Hilbert initialized with a predetermined number of neurons; b) the network position at iteration 25; c) the network at iteration 500

Other investigations with ParaSOM include parallelization (Hammond, MacLean, & Valova, 2006) via message-passing interface (MPI) among 4 worker and 1 director machines (Valova et al., 2009), controlling the parameters of ParaSOM with genetic algorithms (MacLean & Valova, 2007), and more recently, testing and network adjustment for multidimensional input.

3.3 ESOINN

The Enhanced Self-Organizing Incremental Neural Network (ESOINN) (Furao, Ogura, & Hasegawa 2007) represent growing architectures, which are partially inspired by GNG and GNG-U. According to the authors of ESOINN, it addresses the stability-plasticity dilemma (Carpenter & Grossberg 1988), by providing the ability to retain knowledge of patterns it has already learned (stability), while still being able to adapt to, and learn, new patterns that it is yet to be exposed to (plasticity). ESOINN identifies clusters during execution by joining together subclusters that form within a larger cluster. Thus, overlap in multiple clusters can be identified and effectively separated.

Typically with growing architectures, the network grows by adding neurons in sufficiently dense area of the input space. In ESOINN, neurons are added when the current input is adequately distant from the closest neuron. A new neuron is added to the network containing the same (not similar) reference vector as the input.

ESOINN decides on adding a neuron based on similarity threshold. It is basically a dynamic distance measure calculated by the distance of the neuron's neighbors, or, if no neighbors are available, all other neurons in the network. When an input is presented to the network, the first and second winners, or best matching unit (BMU) and second matching unit (2BMU), are determined. The network then decides if a connection between the winner and second winner should be created, if one does not already exist.

In ESOINN, knowledge of the neuron density in a given area of the input space is critical to performing tasks such as creating connections between neurons and detecting overlap in clusters. By being able to measure the density, the network can better determine whether a particular section of the input space is part of a single cluster or of an overlapped section. After detection of overlapped areas, connections between neurons of different subclasses are

removed. This separates the subclasses belonging to different composite classes. This process is performed at regular intervals, where the number of inputs presented to the network is evenly divisible by predetermined integer value.

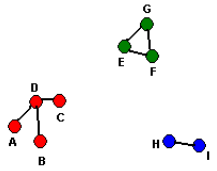

Fig. 12. Neurons in {A, B, C, D} are all connected by paths and therefore are in the same cluster. The same is true with {E, F, G}, and {H, I}. Conversely, A and E have no path to each other, and therefore are not in the same class

Connections in ESOINN are used to identify subclasses. To aide in this identification, they are created and removed from neurons as new data is presented to the network. When a connection is removed between two neurons, a boundary is identified between the different classes that each neuron is a part of. The paths created by connections are also the way that neurons are classified at the end of ESOINN execution. Any given neuron *i*, and all other neurons that are connected to *i* by a path, are considered to be in the same class. Neurons that cannot be connected by a path are said to be in different classes (Fig.12).
Connections in ESOINN are created when there is no existing connection between a winner and second winner. In this case, the newly created connection has an age attribute that is set to zero. If a connection already exists between the winner and second winner, the age of that connection is reset to zero. In either scenario, the ages of all existing connections between the winner and its neighbors are increased by one (except the connection between it and the second winner). Deletion of connections occurs when the ESOINN algorithm determines that the current winner and second winner are in different subclasses and those subclasses should not be merged.
ESOINN adds neurons because they represent noisy input which is likely to be distant from relevant patterns. As a result, the input will be outside of the similarity threshold of the winner and second winner, and a new neuron is created. These neurons are undesirable because they are generally placed in low-density areas and can skew the cluster identification. ESOINN removes neurons with two or fewer neighbors utilizing average density value. When a neuron is deleted, all connections associated with it are also removed. This process also occurs after predetermined number of inputs has been presented.
The connection removal and addition features of ESOINN make it very powerful at finding distinct patterns in a wide array of problem domains. ESOINN is a major step forward in unsupervised learning. Since ESOINN addresses the stability-plasticity dilemma (continued learning with no forgetting), it is an algorithm that can be used for varying types of data sets, including overlapping Gaussian distributions.

3.4 TurSOM

TurSOM (the amalgamation of Turing and SOM) is a new variant of the Kohonen Self-organizing Map, introduced by the authors (Beaton, 2008; Beaton, Valova, & MacLean, 2009a, b, c). TurSOM's primary contribution is the elimination of post-processing techniques

for clustering neurons. Its features are inspired in part by Turing's work on unorganized machines (Turing, 1948). Turing's unorganized machines (TUM) represent early connectionist networks, meant to model the (re)organization capability of the human cortex. In Kohonen's SOM algorithm, the neurons are the components of self-organization, whereas with Turing's idea, the connections also fulfil that role. In TurSOM, we capitalize on both methods of self-organization.

While the neurons of TurSOM adhere to the same learning rules and criteria of the standard SOM, the major differentiating feature of TurSOM is the ability to reorganize connections between neurons. Reorganization includes the removal, addition, or exchanging of connections between neurons. These novelties make TurSOM capable of identifying unique regions of input space (clustering) during execution (on-the-fly), as demonstrated in Fig.13. The clustering behavior is achieved by allowing separate networks to simultaneously execute in a single input space. As TurSOM progresses, connections may be removed, or exchanged – causing a network to split into two networks, and two into three or four, and so on. Additionally, when separate networks get closer to one another they may join to form a single network.

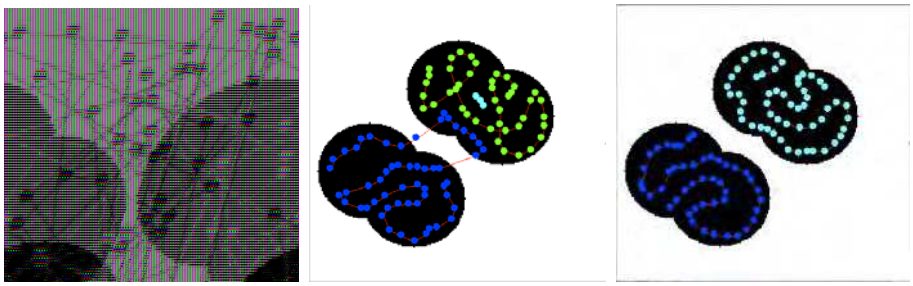

Fig. 13. TurSOM on connection reorganization: a) TurSOM at initialization; b) TurSOM at 250 iterations – exemplary of TurSOM reorganizing connections; c) TurSOM at 350 iterations – exemplary of TurSOM identifying unique patterns

In order for TurSOM to achieve the behavior it exhibits, several new mechanisms are introduced to the Kohonen SOM.

In SOM algorithms, there is a neuron learning rate. The point of the neuron learning rate is to decrease the movement of winning neurons (and subsequently their neighbors) as time progresses. As a SOM adapts to input, it should require less drastic organization, i.e., smaller movements.

Similarly, TurSOM introduces a connection learning rate (CLR), which regulates the reorganization of connections as TurSOM progresses. The CLR is a global value controlling the maximum allowable distance between two neurons. If the distance between any two neurons exceeds the CLR, they must disconnect. CLR is computed as follows:

$$CLR = Q_3 + (i \times (Q_3 - Q_1)) \tag{11}$$

The CLR formula is derived from the upper outlier formula from box-plots (a statistical technique of measuring distribution by analyzing four quartiles). In CLR, the $_x$ in Q_x represents which quartile it is, and *i,* is an incrementing value as time progresses. The data

being measured (for the quartiles), is the length of all connections available in the current input space. The CLR is instrumental to the reorganization process in TurSOM as it effectively decides which connections are unnecessary.

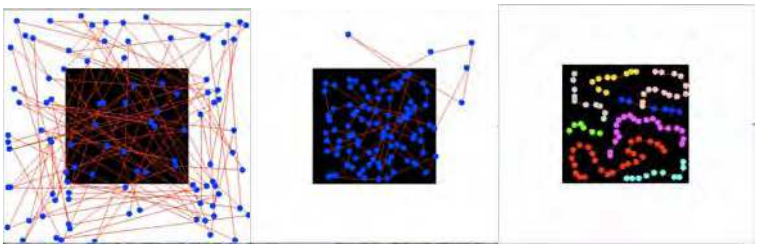

Fig. 14. CLR in TurSOM: a) the square pattern with random initialization; b) the first 50 iterations of TurSOM, where its behavior is the same as a SOM; c) CLR has been active for 100 iterations and a rapid, and sudden reorganization of connections is evident

Fig.14a, b and c demonstrates a simple solid pattern for the first 150 iterations of TurSOM. The CLR determines which connections will be eliminated. The connections that are not considered optimal are removed and as evident by the figure, the removed connections were negatively impacting the network.

So far, we have described how TurSOM may separate into different networks, but we have not addressed how two networks can rejoin into one. The neuron responsibility radius (NRR), inspired by ParaSOM's neuron cover region (addressed in section 3.2), becomes active in TurSOM when two neurons disconnect from one another. However, there is one requirement for networks that disconnect – they must be of size three or greater. Empirically, it has been shown (in TurSOM) that networks smaller than three (i.e. 2 or a single neuron) become "pushed aside" for other neurons that are active in a network. A neuron with an active radius still has one neighbor.

The neuron responsibility radius, is effectively a "feeler", actively searching for other "free" neurons with similar features. To calculate the NRR, the following formulae are used when the dimensionality of input space is even:

$$r_e = \left[\rho / e \right]^{1/\delta} \tag{12}$$

$$\mathrm{e} = \left(\frac{\delta}{2}! \right)^{-1} \times \pi^{\frac{\delta}{2}} \tag{13}$$

If the dimensionality is odd:

$$r_o = \left[\rho / o \right]^{1/\delta} \tag{14}$$

$$\mathrm{o} = \left(\frac{2^{\delta} \times \frac{\delta - 1}{2}!}{\delta!} \right) \times \pi^{\frac{\delta - 1}{2}} \tag{15}$$

where δ represents the number of dimensions, and ρ represents the number of inputs a neuron is responsible for. ρ is calculated by dividing the number of neurons, by the number of inputs. To follow along with the example provided in the previous section on connection learning rate, the following Fig.15) demonstrate the remaining iterations of TurSOM, where the effects of the NRR are seen.

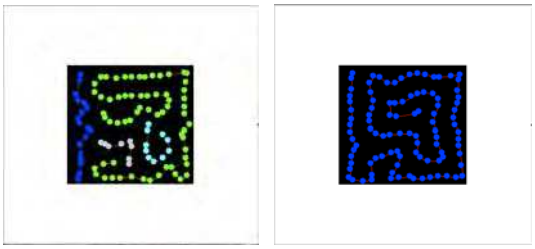

Fig. 15. The effects of NRR: a) demonstrates the reconnection process, which is governed by the NRR; b) The single pattern in an optimal mapping, where the Peano-like curve of the network must be noted

As demonstrated in Fig.15, the NRR determines the reconnection process, and works in cooperation with the CLR (the disconnection process). TurSOM also provides for a neuron to sense nearby neurons that are *better suited* to be connected neighbors than current neighbors. Simply stated, this is a check that neurons perform by knowing the distance to their neighbors, and knowing of other neurons in the network that are nearby. This process is considered to be a part of the reorganization process.

Similar to Frtizke's growing grid algorithm, TurSOM has a growth mechanism. TurSOM begins as a one-dimensional chain, which upon *convergence*, will spontaneously grow (SG) to two-dimensional grids. The term *convergence* is used loosely here to mean a network reaching a fairly stable representation of the input where further computation would not benefit the network significantly. During the spontaneous growth phase, connection reorganization (which implies network splitting and rejoining) is turned off. Presumably, at this point, the one-dimensional networks have settled to satisfactory positions, and do not require further adaptation. The growing process is demonstrated in Fig.16. Fig.16a illustrates the input pattern. The converged one-dimensional SOM is shown in Fig.16b. Finally, the SG is demonstrated in Fig.16c, where it is evident that each one-dimensional network grows independently.

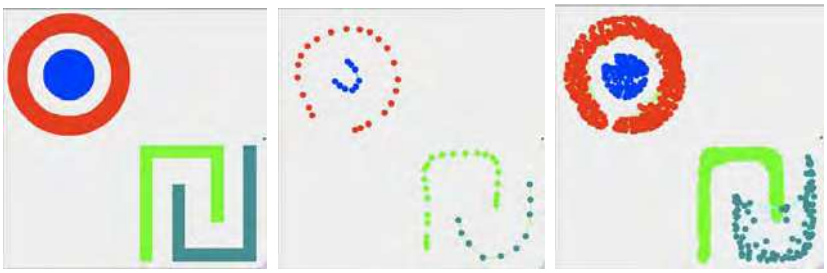

Fig. 16. TurSOM in action: a) Input space with 4 distinct patterns, which are five-dimensional data (X,Y, R, G, B); b) TurSOM in one-dimensional form mapping each of the distinct patterns; c) TurSOM with SG for a better representation of the patterns

TurSOM's growing capabilities are an instrumental part facilitating the performance of the network. Often times, one-dimensional networks do not represent the underlying data well enough. Two-dimensional networks have a better representation, or, a better *resolution*.
Finally, the TurSOM algorithm can be summarized in the following steps:

a) Select Input
b) Find best-matching unit (BMU)
c) Update BMU and BMU's neighbors
    1) Record the distances between all connected neighbors
d) Check lengths of all connections (Step c.1)
    1) If connection is too large
        - Disconnect neurons
        - Update Connection Learning Rate
        - Activate Neuron Responsibility Radius
e) Check neuron physical locations
    1) If neuron A is a neighbor of B, but not C (which is a neighbor of B), but A is closer to C than B, switch connections - thereby changing neighbors
f) Check neuron responsibility radius for proximity to other neurons
    1) Reconnect neurons that have overlapping NRR
g) If TurSOM has reached convergence
    1) Spontaneous Growth

3.5 Modular Network Self-organizing Map

While not a growing architecture, a very recent SOM architecture called the modular network Self-Organizing Map (mnSOM) (Tokunagaa & Furukawa, 2009) is mentioned here. This architecture is a hybrid approach to neural network computing. The mnSOM architecture consists of a lattice structure like that of a typical SOM. Additionally, the neurons in the SOM behave in a similar self-organizing fashion. However, each neuron is composed of or "filled with" a feed-forward network, such as a multi-layer perceptron (MLP).
The major difference between SOMs and feed-forward networks, is that SOMs learn the *topology* or structure of data. Feed-forward architectures learn *functions* about input.
The effective outcome of this network is that it *self-organizes function space*. That is to say, when presented with various types of input patterns where functional knowledge might be very important, mnSOM is able to topologically order functions based on similarity.

4. Methods for SOM analysis

Self-organizing maps are powerful analytical tools. Visualization is often employed to analyze the resulting topological map. However, sometimes networks do not represent optimal mappings. This can skew the understanding, or even representation of the data that is supposed to be visualized. In this section we provide methods of analyzing the *quality* of a SOM network. Commonly, these techniques are used *post-execution*, in order to analyze how well the SOM *converged* to the given data. A survey of SOM quality measures can be found in (Pölzlbauer 2004).

4.1 Quantization Error

Quantization error (QE) is a simple measure used in other fields, including clustering and vector quantization as a technique for verifying that inputs are with their proper (or best suited) clusters. As SOMs perform clustering, QE can be utilized. However, one major draw back is that QE does not address the *quality of organization* of the network. Rather, it measures neuron placement to inputs.

Quantization error is measured by computing the average distance from inputs to their appropriate outputs. One point to note about this measure, is that when the number of neurons is decreased or increased for the same input space, the value acquired by quantization error is increased or decreased respectively. Effectively, more neurons mean a smaller error, and vice versa for less neurons. The QE is calculated by computing the average distance from inputs to their associated neuron. Short pseudocode is given below:

```
uniquely number all neurons
for each input
        find best-matching unit (BMU); aka neuron
        array[BMU#][1] = array[BMU#][1] + distance from input to BMU
        array[BMU#][2] = array[BMU#][2] + 1;
end
for each neuron as x
        error[x] = array[x][2] / array[x][1]
end
```

4.2 Topographic Error

Topographic error (TE) measures the *quality of organization* of SOMs, and provides information of *how well organized neurons are with respect to other neurons*. This measure is used to see if neurons are correctly identified as topological neighbors, with respect to inputs.

Conceptually, TE is attempting to give an idea as to how *twisted*, or *tangled* a SOM network is. An example of a 2-dimensional pattern, with a 1-dimensional map is shown in Fig.17. Topographic error is represented as a value between 0 and 1, where 0 indicates no topographic error, *therefore, no tangling*, and 1 would indicate maximum topographic error or *complete tangling*.

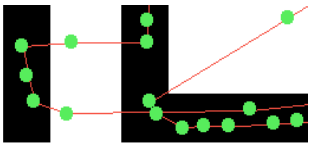

Fig. 17. This pattern shows (in the bottom right) a 1-dimensional network that intersects, or crosses connections. Effectively, this network is tangled, as there are *more appropriate neighbors* for some of the neurons.

Topographic error is computed as follows:

```
error = 0
for each data sample
        find best-matching unit (BMU)
        find second best-matching unit (2BMU)
```

```
            if BMU is not a lattice neighbor of 2BMU
                  error = error + 1;
            end
end
error = error / number of neurons;
```

4.3 Topographic Product

Topographic product (TP), introduced by (Bauer & Pawelzik, 1992), is a measure to indicate if a SOM is too large or too small for the input it is representing. TP measures the *suitability* and *size appropriateness* of a map, for a given set of input. Two primary variables are utilized for the computation of TP, Q_x and P_x. Q_x is a ratio of distances found in input and output spaces, and P_x is a multiplicative normalization of it's respective Q_x value. Below are the initial steps for TP:

Step 1: For the weight (in input space) (**w**) of neuron j ($\mathbf{w}_j$), find the k^{th}:

a. Closest data in input space, as distance d_1^V

b. Closest neuron in output space, as distance d_2^V

Step 2: For the neuron j, find the k^{th}

a. Closest neuron in output space, as distance d_1^A

b. Closest data in input space, as distance d_2^A

Step 3: Create two ratios:

$Q_1(j,k) = d_1^V / d_2^V$

$Q_2(j,k) = d_1^A / d_2^A$, where k represents an iterative value.

When using k in the SOM, the iteration occurs through all other neurons besides j (steps 1a and 2a). Similarly, when calculating Q_1, the iteration occurs through all inputs, excluding $\mathbf{w}_j$ if $\mathbf{w}_j$ is equal to one of the inputs (steps 1b and 2b).

These two values, Q_1 and Q_2, optimally would be equal to 1, if and only if neighbors are correctly preserved and suitably organized. However, Bauer and Pawelzik point out that this is far too sensitive. A normalization of Q_1 and Q_2 is required, via Π function (pseudo code provided):

```
Px(j,k) = 1;
for each neighbor of a neuron j, represented by k
      Px(j,k) = Px(j,k) * Qx(j,k)
end
Px(j,k) = power(Px(j,k), (1/k))
```

x of P_x, and Q_x are either 1 or 2, defined from the previous steps. At this point, $P_1 \geq 1$, and $P_2 \leq 1$, as P_1 is a value created from all the data in input space, based on the weights of all neurons. If there is a 1-to-1 mapping of neurons to input, then this value should be 1. Additionally, P_2 will be less than or equal to one, because it is a representation of neighboring neurons in the output space. This occurs because the denominator of Q_2 comes from input space distances and the numerator comes from neurons distances.

However, having two numbers to explain a mapping is not desirable, so Bauer and Pawelzik introduce P_3 (provided below in pseudo code):

```
P3(j,k) = 1;
```

for each neighbor of a neuron j, as k
$P_3(j,k) = P_3(j,k) * (P_1(j,k) * P_2(j,k))$
end
$P_3(j,k) = power(P_3(j,k), (1/2k))$

P_3 is normalized. The relationship of P_1 and P_2 is *inverse*, thereby giving way to these rules:
1: $P_3 > 1$ means the map is too large, $P_1 > (1/P_2)$
2: $P_3 < 1$ means the map is too small, $P_1 < (1/P_2)$
The final step in topographic product is computing an average of the values already obtained, when using all neurons in a SOM:

P = 0;
for each neuron, as j
for each neighbor as k
$P = P + log(P_3(j,k))$
end
end
P = P/(N(N-1)) // where N is the number of neurons*

All values of P that deviate from 1, should be of concern. The same rules apply to P as do P_3, concerning deviation from 1. The formulaic view of P_3 is provided by Eqs. (16) and (17).

$$\mathrm{P}_3 = \left(\sum_{l=1}^{k} \frac{d^V\left(w_j, w_{n_l^A(j)}\right)}{d^V\left(w_j, w_{n_l^V(j)}\right)} \cdot \frac{d^A\left(j, n_l^A(j)\right)}{d^A\left(j, n_l^V(j)\right)} \right)^{1/2k} \tag{16}$$

$$P = \frac{1}{N(N-1)} \sum_{j=1}^{N} \sum_{k=1}^{N-1} \log(P3(j,k)) \tag{17}$$

4.4 Other Measures

We have presented three measures of SOM that evaluate fundamental aspects of SOM quality, namely, correct neuron to input positioning (QE), network organization quality (TE), and suitability of map size (TP).
However, there are several measures beyond these that attempt to combine these fundamental aspects, or measure other characteristics of SOMs. Some of these measures include Trustworthiness and Neighborhood Preservation (Venna & Kaski, 2001), which aim to measure data projection relations, by comparing input data relations to output data relations; and Topographic Function (Villmann et al., 2007), a measure which accounts for network structure, and neuron placement.

5. Pattern identification and clustering

Over the years many methods of analyzing the patterns of the neurons of SOMs have been introduced. One of the simplest methods is the gray scale clustering presented by Kohonen in his book, on the poverty map data set (Kohonen, 1995). Kohonen's example colors distances between nodes a shade of light gray, if the nodes are close, or dark gray, if the nodes are far. However, visual methods leave interpretation up to the reader. In this section

we present two methods of analyzing and identifying patterns exhibited by the neuron configurations of SOMs. These are methods for post-convergence analysis. When a SOM converges, it is not always necessary to perform any post-processing techniques, especially in lower dimensionality. At the time of convergence, what we do know is that each neuron has found a suitable region of space, where it is representing a given amount of inputs. Exactly what inputs is not always clear unless another technique is used (one technique to map inputs to neurons is QE). Additionally, there may be a relationship that exists between neurons. This section will explain methods of measuring similarity and the relationships between neurons.

5.1 PCA

Principal components analysis (PCA) is a well-established statistical technique, used in a variety of fields on high-dimensional data. The primary goals of PCA are dimensionality reduction and explanation of covariance (or correlation) in variables. Effectively, PCA provides linearly separable groups, or clusters within high-dimensional data along a given dimension (variable). Additionally, the principal components computed by PCA can identify which variables to focus on, i.e. which variables account for the most variance. Variables are determined to be unnecessary when they do not explain much variance. For a detailed explanation on PCA and how to implement it, please see (Smith 2002, Martinez, & Martinez 2005).

PCA can be used as a pre- (Kirt, Vainik & Võhandu, 2007; Sommer & Golz, 2001) and post- (Kumar, Rai & Kumar 2005; Lee & Singh, 2004) processor for SOM. Additionally, a SOM has been created to combine the capabilities of both PCA and SOM (López-Rubio, Muñoz-Pérez, Gómez-Ruiz, 2004).

When analyzing a SOM for potential clusters, understanding the relationship among neurons usually presents great challenge. This analysis can become difficult when analyzing a converged map when there are very few (small network) or very many (large network) neurons. Additionally, it may be more useful to ignore certain variables prior to executing a SOM on a data set. This is where PCA becomes a very useful tool.

It is important to note that PCA is a linearly separable unsupervised technique. Effectively, a vector is drawn from the origin to a point in space and it is determined that the groups to one side and the other are significantly distinct (based on a given variable or dimension). SOM on the other hand, is non-linear, and each neuron can be thought of as a centroid in the k-means clustering algorithm (MacQueen, 1967). Neurons become responsible for the input that they are closest to, which may be a spheroid, or even a non-uniform shape.

In the case PCA is performed *prior to* executing a SOM on a data set, it will be determined which variables, or dimensions, are most important for a SOM, and now the neurons in a SOM will have less weights than the original data set. In case PCA is performed *after* a SOM has executed, the method will determine which variables in the weights of the SOMs are most important. This will help explain which neurons are more similar than others, by contrast to other methods like distance measures and coloring schemes. In summary, PCA helps eliminate attributes that are largely unnecessary.

5.2 ParaSOM modifications

The ParaSOM architecture takes a unique approach to performing cluster identification. It relies heavily on the features of the network and the behavior it exhibits because of those features (Valova, MacLean & Beaton, 2008b).

When the network is well-adapted and near the end of execution, the cover regions of the neurons are generally small and covering their respective sections of the input space precisely. Therefore, in dense regions the neurons should be plentiful and in very close proximity. A key property of the network at convergence is that the distances between intra-cluster neurons will likely be much smaller than the distance between inter-cluster neurons. This is the central concept of the cluster identification algorithm that ParaSOM takes advantage of.

Once convergence takes place, the network will perform clustering in two phases. The first phase utilizes statistical analysis to initially identify clusters. The second phase employs single and complete linkage to combine any clusters that may have been separated, but are in close enough proximity to be considered a single cluster.

In order to determine the minimal distance between neurons in different clusters we make use of the mean of the distances between neighboring neurons, $\bar{x}$, as well as their standard deviation, σ. The standard deviation is used to determine how far away from the mean is considered acceptable in order for a neighbor to be labeled in the same cluster as the neuron in question.

The overwhelming majority of the connections between neighbors will be within clusters. Therefore, the number of standard deviations away from the mean connection distance that a certain majority of these connections is within will be a good indicator of an adequate distance threshold.

To discover the initial clusters, the mean of the distances between neighbors is determined through iteration on all neurons. Following that, the standard deviation of the distances between neighboring neurons is computed via Eq. (18).

$$\sigma = \sqrt{\sum_{i=1}^{n} \left(\bar{x} - d_i\right)^2} \tag{18}$$

where d_i is the distance between a pair of neighboring neurons.

Further, determine how many neuron pairs lie within $\bar{x} + m$ standard deviations, for each $m \in \{1 \ldots n\}$. Based on some threshold α, where 0.0<= α <=1.0, determine the minimum distance $\bar{x} + m\sigma$ that the percentage of neurons specified by α are within. This distance will become the initial cutoff threshold κ. Finally, iterate through the neurons one final time to determine where new clusters are formed. When a neuron's distance to a neighbor exceeds κ, add the neighbor to new cluster.

The method used to determine κ is based on Chebyshev's theorem (Sternstein, 1994), which states that at least $1 - 1/p^2$ of the values in a set of data lie between p standard deviations of the mean. This theorem is applicable to generalizations that are valid for any set of data. However, the behavior of the ParaSOM requires heuristic modifications to the theorem. We modify the original theorem to obey the following principle:

$$\kappa = \bar{x} + p \cdot \sigma \tag{19}$$

where p satisfies the minimum $p \cdot \sigma$ from $\bar{x}$ that the percentage of neurons specified by α lie within. Since the ParaSOM provides a narrow normal distribution of distances between neighbors, the determination of κ for it to be fine-tuned provides effective adaptation to any input space distribution.

There are situations where separating clusters based on distances between neighbors leads to undesired results. Ideally, the cutoff threshold κ should be adequate to accurately determine cluster membership. However, a number of factors that influence the location of neurons can cause erroneous decisions by the method described above. There are cases where the distance between neighboring neurons is outside the cutoff threshold because of the insertion of neurons between them (Fig.18).

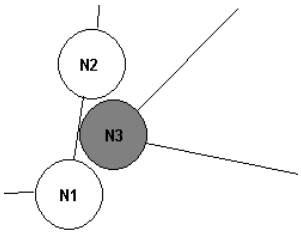

Fig. 18. Neurons N1 and N2 are neighbors, but their distance exceeds the cutoff threshold because N3 is between them.

The cluster identification algorithm takes this into account and will recombine clusters that are originally determined to be disjoint, but have sufficient similarity to merge. This cluster recombination is facilitated by two techniques used in tandem: single and complete linkage. Single linkage is a technique that links together clusters based on the minimal distance between any one element in one cluster and any one element in a second cluster. Single linkage recombination sometimes has a tendency to chain together clusters and produce potentially undesirable results. Being aware of this tendency, ParaSOM's recombination algorithm uses a threshold value β to make sure the linkage distance between clusters does not exceed this value. The β threshold determination also relies on the ParaSOMs tendency to create an abundance of tightly packed neurons in dense input regions.

Looking again at Fig.18, let us assume that the distance separating N1 and N2 is greater than κ. Therefore, according to the initial clustering, N1 and N2 are in different clusters. Let us also assume that N2 and N3 are in the same cluster. Because N1, N2, and N3 are all in such close proximity, it makes sense that they should be in the same cluster. When single linkage occurs, the distance between cluster containing N1 and the cluster containing N2 (or N3) will be well within the β threshold, and these clusters will be combined.

Since the ParaSOM covers space as effectively as possible, in a mature network adaptation of the input space the distance between neighboring clusters that should be combined will be very close to 2*r*, where *r* is the cover region radius. As a result, any reasonable β value will recombine these two clusters. For the ParaSOM, β is set to $\bar{x} + \sigma$.

The second aspect of the cluster recombination process is using complete linkage to rejoin stray clusters that were mishandled by the initial clustering and not accounted for by single linkage. Complete linkage is utilized by taking the maximum distance, or dissimilarity, between elements of two clusters. Complete linkage is employed to join small clusters and clusters containing only a single neuron. These are cases where the clusters may have been too distant for single linkage to recombine them. Generally, when complete linkage is used, there is a chance that the quality of clusters may degrade if the farthest neighbors that join two clusters together happen to be closer to other clusters than they are to their own. However, such an event will not be the case. Since complete linkage measures cluster distance based on the furthest neighbors, the threshold that determines if clusters are recombined is more lenient than β. For complete linkage, we use a threshold of κ, which was introduced previously.

Following the methodology description, we provide two examples of the pattern identification capabilities of ParaSOM. Figs.19 and 20 take the network through its progress in adjusting to the input topology and proceeding to evaluate the number of distinct regions the input space features. The pattern in Fig.19 is clearly requiring one class, however due to the two very distant shoulders, presents problems for most unsupervised clustering methods. Not so for ParaSOM, which recognizes and configures the final result into one cluster. The pattern in Fig.20 presents the difficulty of the two clusters being very close together. Within 600 iterations and the above described algorithm, ParaSOM clearly identifies the correct number of clusters.

Fig. 19. Identification of patterns with ParaSOM: a) network at 100 iterations; b) network state at 400 iterations; c) 600 iterations; d) inspite of the input distribution, the single object is identified

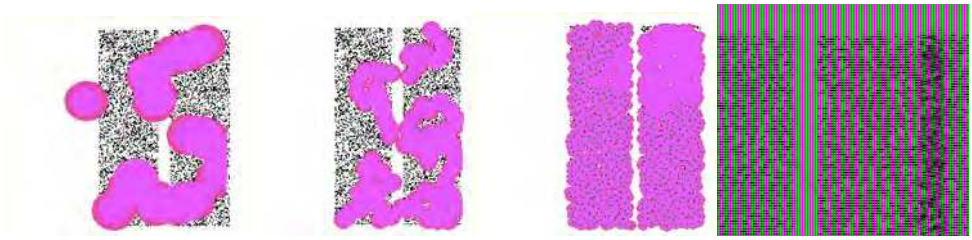

Fig. 20. Identification of patterns with ParaSOM: a) network at 100 iterations; b) network state at 300 iterations; c) 600 iterations; d) the two distinct regions are recognized

6. Conclusions

In summary, we present a number of SOM algorithms, with a primary focus on growing architectures. The goal of this chapter is to introduce, in chronological order, the

development of growing SOM-based techniques, featuring various variants based on Kohonen and Fritzke algorithms. The authors present their two major contributions - ParaSOM and TurSOM along with the ESOINN network which features incremental learning and cluster identification. We also present methods for SOM analysis which we have used in our research to facilitate the comparison between the methods we develop and the state-of-the-art algorithms presented by other researchers.
As a point of future work, we are continuing the development of both ParaSOM and TurSOM for various applications and further improve the algorithms in terms of complexity of performance and implementation.

7. References

Bauer, H. U. and Pawelzik, K. R., (1992) Quantifying the neighborhood preservation of self-organizing feature maps, *IEEE Trans. on Neural Networks*, vol. 3, no. 4, July 1992.

Beaton, D. (2008). *Bridging Turing unorganized machines and self-organizing maps for cognitive replication*. Master's Thesis, University of Massachusetts Dartmouth.

Beaton, D., Valova, I., & MacLean, D. (2009a). TurSOM: a Turing inspired self-organizing map, *International Joint Conference on Neural Networks*.

Beaton, D., Valova, I., & MacLean, D. (2009b). Growing Mechanisms and Cluster Identification with TurSOM, *International Joint Conference on Neural Networks*.

Beaton D., Valova, I., & MacLean, D. (2009c). The Use of TurSOM for Color Image Segmentation. *IEEE Conference on Systems, Man and Cybernetics.*

Buer, A. (2006). *Initialization of self-organizing maps with self-similar curves*. Master's Thesis, University of Massachusetts Dartmouth.

Carpenter, G. A., & Grossberg, S., (1998) The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network, *Computer*, vol. 21, no. 3, pp. 77-88.

Carpenter, G.A., & Grossberg, S. (1990) ART 3: hierarchical search using chemical transmitters in self-organizing pattern recognition architecture, *Neural Networks*, 3:129-152.

Chien-Sing, Lee & Singh, Y.P., (2004) Student modeling using principal component analysis of SOM clusters, *IEEE International Conference on Advanced Learning Technologies*, 30:480-484.

Fritzke, B. (1992). Growing cell structures -- a self-organizing network in k dimensions. In I. Aleksander, & J. Taylor (Eds.), *Artificial neural networks II*. North-Holland, Amsterdam.

Fritzke, B. (1993a). Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7, 1441-1460.

Fritzke, B. (1993b). Supervised learning with growing cell structures. In *Proceedings of Neural Information Processing Systems*, pp. 255-262.

Fritzke, B. (1993c). Kohonen feature maps and growing cell structures – a performance comparison. In *Proceedings of Neural Information Processing Systems,* pp. 123-130.

Fritzke, B. (1994). A growing neural gas network learns topologies. In *Proceedings of Neural Information Processing Systems,* pp.625-632.

Fritzke, B. (1995). Growing grid – a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, *2*, 5: 9-13.

Furao, S., Ogura, T., & Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, 20, 8:893-903.

Furao, S., Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, 19, 1:90-106.

Hammond, J., MacLean, D., & Valova, I., (2006) A Parallel Implementation of a Growing SOM Promoting Independent Neural Networks over Distributed Input Space, *International Joint Conference on Neural Networks (IJCNN)*,1937 - 1944.

Institut für Neuroinformatik from DemoGNG website: http://www.neuroinformatik.ruhr-uni-bochum.de/VDM/research/gsn/DemoGNG/GNG.html

Kirt, T., Vainik, E., Võhandu, L., (2007). A method for comparing self-organizing maps: case studies of banking and linguistic data. In: *Local Proceedings of the 11th East-European Conference on Advances in Databases and Information Systems,* 107 - 115.

Kohonen, T. (1995) *Self-Organizing Maps*, Springer, Berlin, Heidelberg, New York.

Kumar, D., Rai, C.S., Kumar, S., (2005) Face Recognition using Self-Organizing Map and Principal Component Analysis, *Neural Networks and Brain, 2005. ICNN&B '05. International Conference on*, pp.1469-1473.

López-Rubio, E., Muñoz-Pérez, J. & Gómez-Ruiz, J.A., (2004) A principal components analysis self-organizing map, *Neural Networks*, 17:261-270.

MacLean, D., (2007) Clustering and classification for a parallel self-organizing map. Master's Thesis, University of Massachusetts Dartmouth.

MacLean, D., & Valova, I. (2007) Parallel Growing SOM Monitored by Genetic Algorithm, *International Joint Conference on Neural Networks (IJCNN)*, 1697-1702.

MacQueen, J., (1967) Some methods for classification and analysis of multivariate observations, *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, 281-297.

Martinez, W.L, & Martinez, A.R., (2005). *Exploratory Data Analysis with MATLAB*. Chapman & Hall/CRC Press, Boca Raton.

Polani, D. (2002). Measures for the organization of self-organizing maps. In U. Seiffert, & L. C. Jain (Eds.), *Self-organizing neural networks,* Physica Verlag.

Pölzlbauer, G.,(2004) Survey and comparison of quality measures for self-organizing maps.

In Ján Paralic, Georg Pölzlbauer, and Andreas Rauber, editors, *Proceedings of the Fifth Workshop on Data Analysis,* 67-82, Elfa Academic Press.

Sammon, J.W. Jr., (1969) *A nonlinear mapping for data structure analysis*, IEEE Transactions on Computation, C-18, 401–409.

Smith, L (2002) A tutorial on Principal Components Analysis. Website: www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf

Sommer, D., & Golz, M. (2001). Clustering of EEG-Segments Using Hierarchical Agglomerative Methods and Self-Organizing Maps. In *Proceedings of the international Conference on Artificial Neural Networks*. 642-649.

Sternstein, M., (1994). *Statistics*. Barron's Educational Series, Inc., Hauppage, N.Y.

Tokunaga, K. & Furukawa, T., (2009) Modular network SOM, *Neural Networks*, 22:82-90.

Turing, A.M. (1948) 'Intelligent Machinery'. National Physical Laboratory Report. *Collected Works of A.M. Turing: Mechanical Intelligence.* Ince, D.C. (ed.) (1992), North Holland, Amsterdam.

Valova, I., Szer, D., Gueorguieva, N., & Buer, A. (2005). A parallel growing architecture for self-organizing maps with unsupervised learning. *Neurocomputing*, *68C*, 177-195.

Valova, I., Beaton, D., & MacLean, D. (2008a) Role of initialization in SOM networks - study of self-similar curve topologies, In *Proceedings of International Conference on Artificial Neural Networks in Engineering* (pp. 681-688).

Valova, I., MacLean, D., & Beaton, D. (2008b) Identification of Patterns via Region-Growing Parallel SOM Neural Network, *International Conference on Machine Learning and Applications (ICMLA)*, 853 - 858.

Valova, I., Beaton, D., MacLean, D., Hammond, J., & Allen, J. (2009) *NIPSOM: Parallel Architecture and Implementation of a Growing SOM,* The Computer Journal, Oxford.

Venna, J., Kaski, S., (2001) Neighborhood preservation in nonlinear projection methods: An experimental study. *Lecture Notes in Computer Science*, 2130:485–491.

Villmann, T. et al., (1997) Topology Preservation in Self-Organizing Feature Maps: Exact Definition and Measurement; *IEEE Transactions on Neural Networks*, vol 8. no. 2, pp 256-266.

Machine Learning

Edited by Yagang Zhang

ISBN 978-953-307-033-9
Hard cover, 438 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Machine learning techniques have the potential of alleviating the complexity of knowledge acquisition. This book presents today's state and development tendencies of machine learning. It is a multi-author book. Taking into account the large amount of knowledge about machine learning and practice presented in the book, it is divided into three major parts: Introduction, Machine Learning Theory and Applications. Part I focuses on the introduction to machine learning. The author also attempts to promote a new design of thinking machines and development philosophy. Considering the growing complexity and serious difficulties of information processing in machine learning, in Part II of the book, the theoretical foundations of machine learning are considered, and they mainly include self-organizing maps (SOMs), clustering, artificial neural networks, nonlinear control, fuzzy system and knowledge-based system (KBS). Part III contains selected applications of various machine learning approaches, from flight delays, network intrusion, immune system, ship design to CT and RNA target prediction. The book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821