# SOME REMARKS ON THE SELF-ORGANIZING FEATURE MAPS

**TRAN DUC MINH**

*Visiting Researcher at Graduadte School of Information, Production and Systems, Waseda University, Japan*

**LE HAI KHOI**

*Institute of Information Technology – VAST, Ha Noi, Viet Nam*

## Abstract

Recent development of Self-Organizing Feature Maps (hereafter SOFMs) algorithm becomes more and more interesting in many fields such as: pattern recognition, clustering, and function approximation, data- and web-mining. This paper will cover issues about the SOFM its self as well as some of its variances like Neural Gas, Growing Neural Gas, Growing Cell Structures, etc and they are all the kind of unsupervised competitive learning algorithms.

This paper will be organized into 5 parts. After the introduction to the SOFM algorithm and some discussion on the motivations that led to the origin is given in part 1, in part 2, the goodness as well as the weakness of this model is discussed. In part 3, the details about the variances most noticeable are discussed in part 2. Exploring the variances of the original SOFM algorithm mentioned in that part will do it. Part 4 gives some discussion on the general of the applications of this model in author's opinions. A conclusion in part 5 is given to conclude this paper.

## 1. Introduction

In this paper, we are investigating a kind of neural network called Self-Organizing Maps (SOMs or Self-Organizing Feature Maps, SOFMs). This kind of network operates based on *competitive learning* in which the output neurons will compete

to be activated or fired. The neuron that won the competition by having the minimum value at some measurement method (e.g. Euclidean distance) is called *winning neuron*. It is some times called *winner-takes-all* neuron.
In an SOM, the neurons are arranged into the nodes of a lattice that is shown in the *Figure 1*.

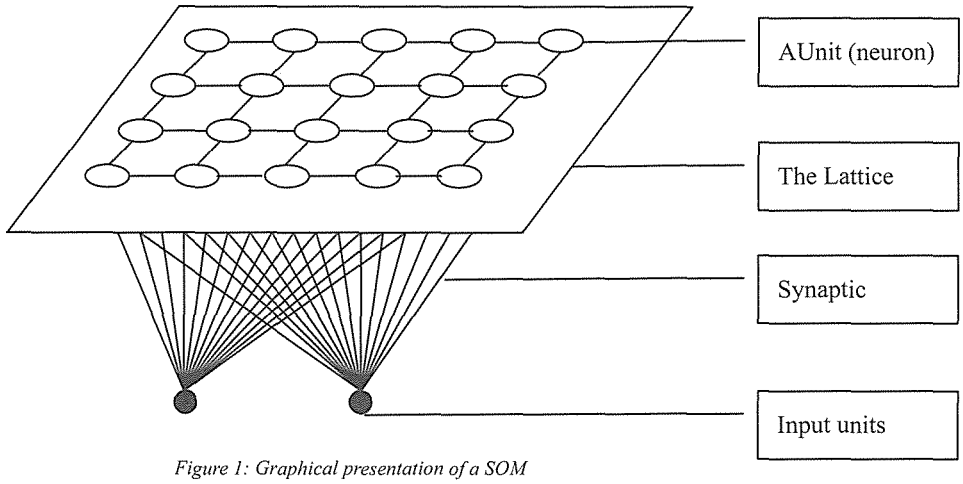


*Figure 1: Graphical presentation of a SOM*

The lattice is usually one or two-dimensional. Higher dimensional maps are possible but not as common because it is limited by the visualization ability and also the applications of this kind of network. Moreover, this kind of network is designed mainly based on the inspiration of an interesting feature of the human brain's cerebral cortex. The training process will make the neurons becomes selectively tuned to various input patterns or classes of input patterns, known as stimuli. A distinct feature of human brain inspires the development of SOMs as a neural model. This is because the *cerebral cortex in the human brain maps different sensory inputs onto corresponding areas of the cerebral cortex in an orderly fashion* [19].

In the literature, we can find 2 basically different feature-mapping models based on the neurobiological inspiration. D.J Willshaw and C. von der Malsburg [26] proposed the first model and the second was the pioneer research work of Teuvo

Kohonen [25]. The later is more general than the former in the characteristic of data compression. Because it can be used to map the distribution of the input data of arbitrary dimension into lower dimensional outputs, thus can be used in data compression. Hereafter, we will mainly discuss about the Kohonen's model that is involved in many researches until recently.

The main goal of the SOFM is to transform an input signal pattern of arbitrary dimension into a lower (1, 2 or 3 dimensions) dimensional discrete map and to perform this transformation adaptively in a topological orderly fashion.

The SOFM algorithm starts by randomly initializing the synaptic weights in the network. That is, no prior order is imposed on the network in the initialization operation. After that, there are 3 major activities involved in the formation of the SOM. They are the followings:

1. Competition
2. Cooperation
3. Adaptation of synaptic weights.

### 1.1. Competition

For each input pattern, the neurons in the network will compute their respective values of a discriminate function based on the input. This function provides the basis for competition among the neurons. The neuron with the smallest value of the discriminate function (usually in Euclidean space: Euclidean distance) is declared as the winning neuron.

To understand which is the distance measurement used in that discriminate function, we will briefly describe it below.
Assume that $m$ is the dimension of the INPUT FEATURES (or input space) and $v$ denotes an input pattern vector., i.e. $v = [v_1, v_2, \ldots, v_m]^T$. The synaptic weight vector has the same dimension as the input space. Let the synaptic weight vector of neuron $i$ denotes by $w_i = [w_{i1}, w_{i2}, \ldots, w_{im}]^T$. To find the best match of the input vector $v$ with the synaptic weight vector $w_i$, compare the distance of the input vector with all the synaptic weight vectors for $i = 1, \ldots N$, where N is the total

number of neurons, and select the smallest. The selection of the *best matching* or *winning* neuron $k$ as,

$$k = \arg\min_i \quad \|v - w_i\|, where \quad i = 1,...,N$$

### 1.2. Cooperation

The *winning* neuron $k$ determines the spatial location of a topological neighborhood of excited neurons, i.e. determines all the neurons that in the "neighborhood area" of the *winning* neuron. According to neurobiological evidence (lateral inhibition or some times called *on-center-off-surround*), a neuron that is firing tends to excite the neurons in its neighborhood more than those far away from it. This observation leads the SOFM algorithm to define the topological neighborhood around the *winning* neuron $k$ as explained below.

Let $d_{ik}$ be the lateral distance between the *winning* neuron $k$ and the excited neuron $i$. We can define the topological neighborhood $h_{ik}$ as a uni-modal function with the following two requirements:

1) It is symmetric about the maximum point defined by $d_{ik} = 0$.
2) Its amplitude decreases monotonically to zero with increasing lateral distance $d_{ik}$. A typical choice of $h_{ik}$ that satisfies this requirement is the Gaussian function:

$$h_{ik} = \frac{-d_{ik}^2}{2\delta^2},$$

where $\delta$ is the width of the topological neighborhood. When using SOFM, some times $\delta$ is shrinked with time. An exponential decay function is a popular choice for this. The $\delta$ reduction scheme ensures that the map actually approaches a neighborhood preserving the final structure, assumed that such a structure exists. If the topology of the output space does not match that of the data manifold, neighborhood violations is inevitable.

An example of such a reduction scheme is provide below:

$$\delta(n) = \delta_0 \exp\left(-\frac{n}{\tau_1}\right), \quad n = 0,1,2,3$$

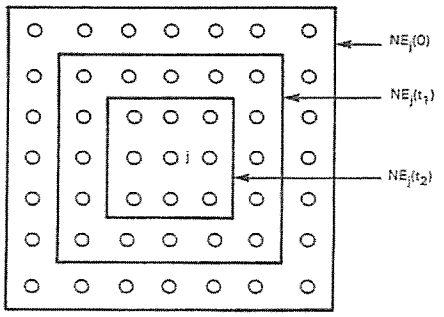


*Figure 2*: A sample reduction scheme of δ

where $n$ indicates the number of sequence, $\delta_0$ is the beginning value of $\delta$, and $\tau_1$ is the time parameter used to reduce the value of the width of the topological neighborhood.

Or another simple method used to decrease the $\delta$ is just multiple it by a value that is less than 1, say, 0.9. These two methods are similar to the simulated annealing scheme with the following note:

The **spread** of the neighborhood function should initially include all neurons for any winning neuron and during the ordering phase should be slowly reduced to eventually include only a few neurons in the winner's neighborhood. During the convergence phase, the neighborhood function should include only the winning neuron.

Example of a 2-D Gaussian neighborhood function for a 40 30 neuronal lattice

is given in *Figure 3*.

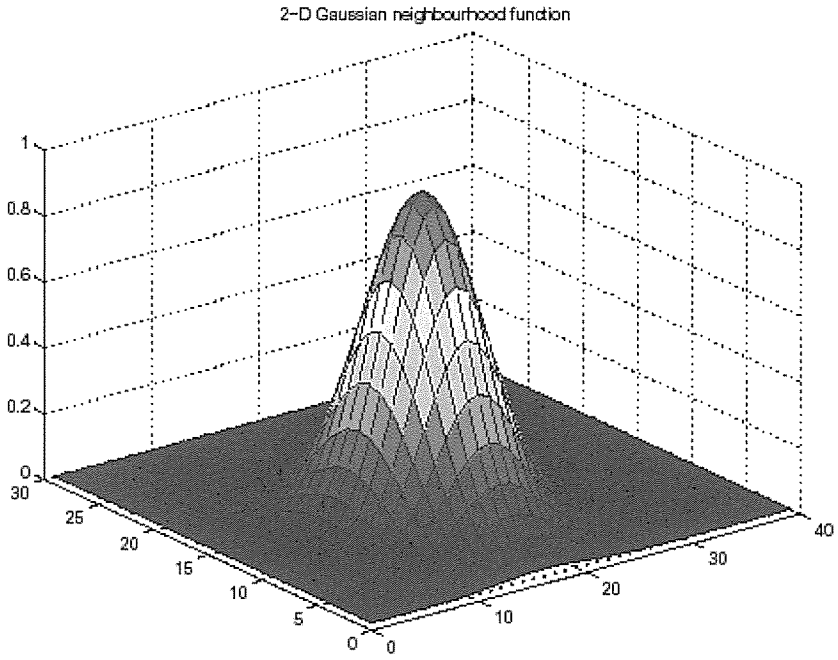


*Figure 3: 2-D Gaussian neighborhood function*

### 1.3. Adaptation of synaptic weights

For the network to be self-organizing, the synaptic weight vector $w_i$ of neuron $i$ is required to change in relation to the input vector $v$. This is a kind of un-supervised learning, so we can use a modified version of Hebbian learning by including a *forgetting term* $g(y_i)$ that is used to keep the weight from growing large, where $g(y_i)$ is some positive scalar function of the response of the network $y_i$. So, we can define the change to the weight vector as:

$$\Delta \mathbf{w}_i = \eta y_i \mathbf{v} - g(y_i)\mathbf{w}_i, \text{ where } \eta \text{ is the learning rate.}$$

Further more, we can define $g(y_i)$ as a linear function of $y_i$, say, $g(y_i) = \eta y_i$ and setting $y_i = h_{ik}$. So we can simplify above equation to the following:

$$\Delta \mathbf{w}_i = \eta h_{ik}(\mathbf{v} - \mathbf{w}_i).$$

Using discrete time formalism, we can derive the final formula for the weight update procedure:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta(n)\, h_{ik}(n)\, [\mathbf{v}(n) - \mathbf{w}_i(n)],$$

where $n$ denotes the times step updating the weight.

This above equation will be applied to all the neurons that lie inside the neighborhood of the *winning* neuron (other neurons are also updated but the weight change is zero caused by the value of the neighborhood function). Upon repeated presenting training data, the synaptic weight tends to follow the distribution of the input data due to neighborhood updating, which causes the adaptation of the weight vectors to the input. The neighborhood updating also makes the weights between neighbor neurons have the similar values to each other [11]. Therefore, the SOFM algorithm leads to a topological ordering of the feature map in the input space in the sense that neurons that are adjacent in the lattice will tend to have similar synaptic weight vectors. The SOFM basic algorithm is presented in Table 1.

**Table 1: Basic SOFM Algorithm**

**Step 1**: Choose random value for the weight vectors $\boldsymbol{w}_i(0)$. It is recommended that all the vectors should be different to each other.

**Step 2**: Select an input pattern $v$ from the input space randomly

**Step 3**: Find the *best matching* or *winning* neuron $k$ at time step $n$ by using the Euclidean minimum-distance criterion:

$$k = \arg \min_i \|v(n) - w_i(n)\|, where \quad i = 1,..., N$$

**Step 4**: Update the synaptic weight vectors of all the neurons using:

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(\mathrm{n}) + \eta(n)\, h_{ik}(n)\, [\mathbf{v}(n) - \mathbf{w}_i(n)]$$

where both $\eta(n)$ and $h_{ik}(n)$ are varied dynamically through time. For example:

$$\delta(n) = \delta_0 \exp(-\frac{n}{\tau_1}) \quad \text{and} \quad \eta(n) = \eta_0 \exp(-\frac{n}{\tau_2}) \quad \text{or so.}$$

**Step 5**: Repeat Step 1 – Step 4 until there is no noticeable changes in the feature map are observed.

The adaptation of the weight vector may be decomposed into two phases

i) Self-organizing or ordering phase: topological ordering of the weight vectors. This phase occurs at early in the beginning when the step size $\eta(n)$ and neighborhood size $h_{ik}(n)$ are large. Some times it is called coarse search since at the beginning, the algorithm will try to spread the affect of activated neuron to

most of the neurons.

ii) Convergence phase: after ordering, for accurate statistical quantification of the input space. This phase will happen when the step size $\eta(n)$ and neighborhood size $h_{ik}(n)$ are reduced to a small value. This can be taken as refinement search.

## 2. Desire and the weakness properties of SOMs

### 2.1. Desire Properties

Once the SOFM algorithm has converged, the feature map computed by the algorithm, displays important statistical characteristics of the inputs. If we denote the spatially continuous input space by $X$ and the topology of which is defined by the metric relationship of the vectors $\mathbf{x} \in X$. Let $A$ denotes a spatially discrete output space, the topology of which is endowed by arranging a set of neurons as the computation nodes of a lattice. Let $\Phi$ denotes a nonlinear transformation called a feature map, which maps the input space $X$ onto the output space $A$, as shown by: $\Phi : X \rightarrow A$.

Given an input vector $\mathbf{x}$, the SOFM algorithm proceeds by first identifying a best matching or winning neuron $i(\mathbf{x})$ in the output space $A$, in accordance with the feature map $\Phi$. The synaptic weight vector $\mathbf{w}_i$ then maybe viewed as a pointer for that neuron into the input space $X$.

In detail, the SOFM algorithm has some important properties as described below:

#### 2.1.1. Approximation of the input space

The Self-Organizing Feature Map $\Phi$, represented by the set of synaptic weight vectors $\{w_i \mid i = 1,2,\ldots, N\}$, in the output space $A$, provides a good approximation to the input space $X$.

The SOFM algorithm's basic aim is to store a large set of input vectors by finding a smaller set of prototypes, so as to provide "good" approximation to the original input space. It is in fact a vector quantization algorithm, which provides us the approximation on the input space $X$.

#### 2.1.2. Topological Ordering

The feature map $\Phi$ computed by the SOFM algorithm is topologically ordered in

the sense that the spatial location of a neuron in the lattice corresponds to a particular domain or feature of input patterns.

This property is a direct consequence of the update of synaptic weight that forces the synaptic weight vectors of the winning neuron to move toward the input vector. It also has the effect of moving the weight vectors of neighbor neurons of winning neuron. The overall aim of the algorithm thus is stated below:

*Approximate the input space X by* pointers *or* prototypes *in the form of synaptic weight vectors* $\mathbf{w}_i$ *in such a way that the feature map* $\Phi$ *provides a faithful representation of the important feature that characterize the input vectors in X.*

The feature map $\Phi$ is usually displayed in the input space $\boldsymbol{X}$. For example, if the input space has 3 dimensions, then the feature map is displayed in the 3-D space respectively. The synaptic weight vectors are presented by dots and the pointers of neighboring neurons are connected with lines in accordance with the topology of the lattice. In particular, we observed that the algorithm (after converged) captures the underlying topology of the uniform distribution at the input.

The topological ordering property of the SOFM algorithm, coupled with its computational tractability, makes it a valuable tool for the simulation of computational maps in the brain. Indeed, the SOFMs are perhaps the simplest model that can account for the adaptive formation of such topographic representations (Ritter et al., 1992).

### 2.1.3. Density Matching

The feature map $\Phi$ reflects variations in the statistics of the input distribution: regions in the input space $\boldsymbol{X}$ from which sample vectors are drawn with a high probability of occurrence are mapped onto larger domains of the output space $\boldsymbol{A}$, and therefore with better resolution than regions in $\boldsymbol{X}$ from which sample vectors are drawn with lower probability of occurrence.

As a general rule, the feature map computed by the SOFM algorithm tends to

over-represent regions of low input density and under-represent regions of high input density. One way to improve the density-matching property of the SOFM algorithm is to add heuristic to the algorithm, which force the distribution computed by the algorithm to match the input distribution more closely. Another way is to use the information-theoretic approach for the computation of the SOFM.

## 2.2. Problems associated with basic SOM algorithm

Although there is many applications of SOFM can be recognized in the literature such as: data analysis, data- and web-mining, function approximation, discovering similarities in data, etc, the SOFM algorithm has some limitations. For example, in the Kohonen's model, the neighborhood relations between neurons have to be defined in advance. Also the topology of the input space has to match the topology of the output space, which is to be represented. That is the property of neighborhood preservation, which distinguishes self-organizing maps from other neural network paradigms, depends on the choice of the output space map topology. However, in real world data sets, the proper dimensionality required by the input space is usually not known a priori, yet the output grid of the lattice has to be specified prior to learning. To tackle this problem, one can use an advanced learning scheme, which adapts not only the weight vectors of the neurons, but also the topology of the output space itself. Some samples of such algorithms include topology representing network [20], the growing cell structure algorithm [2], SPA neural tree algorithm [16] and the growing hypercubical output space algorithm [1].

In addition, the dynamics of the SOFM algorithm cannot be described as a stochastic gradient descent on *any* energy function. The only solution for this problem currently, is to describe the dynamics of the algorithm as a set of energy functions, one for each weight vector [10].

There are also other problems that unsuitable for the Kohonen net as below:

- Kohonen nets work by clustering
- Nearby data points are expected to behave similarly, e.g. have similar outputs.
- Parity-like problems such as the XOR do not have this property. There would be

unstable for solution by a Kohonen net.

## 3. Variance of basic SOM algorithm

We can see that basic SOFM algorithm defines a non-parametric regression solution to a class of vector quantization problems and in that sense, it does not need any modification because this model that was initially proposed by Kohonen for this task. However, there exist other problems that SOFM can be applied in various ways, as an example, pattern classification tasks rather than just using it as a vector quantization machine.

There seems to exist a number of ways to define the matching of an input occurrence with the internal representation (e.g. different metrics) and even the neighborhood of a neuron can be defined in many ways without affect the performance of the network [10]. Regarding the definitions of neighborhood, authors suggested that the formulation of $h_{ik}$ should be depended on the intermediate results.

In the following sections, we will investigate briefly about some well-known variances of the Kohonen SOFM algorithm.

### 3.1. Batch Update: LBG

The LBG (or generalized Lloyd) algorithm ([28]; [29]; [30]) works by repeatedly moving all reference vectors to the arithmetic mean of their Voronoi sets. The theoretical foundation for this is that it can be shown ([31]) that a *necessary* condition for a set of reference vectors $\{\boldsymbol{w}_x \mid x \in A\}$ to minimize the distortion error

$$\text{¦}(\text{Å}D,A) = \frac{1}{|D|}\sum_{x\in A}\sum_{\varepsilon\in R_x}\left\| \hat{I}-\boldsymbol{w}_x \right\|^2$$

is that each reference vector $\boldsymbol{w}_x$ fulfills the *centroid condition*. In the case of a finite set of input signals and the use of the Euclidean distance measure the centroid condition reduces to

$$\mathbf{w}_x = \frac{1}{|\boldsymbol{R}_x|} \sum_{\xi \in \boldsymbol{R}_x} \hat{\boldsymbol{I}}$$

whereby $\boldsymbol{R}_x$ is the Voronoi set of unit $x$.

The complete LBG algorithm is the following:

1. Initialize the set $A$ to contain $N$ ($N << M$) units $x_i$

   $A = \{x_1, x_2, ..., x_N\}$

   with reference vectors $\mathbf{w}_{x_i} \in \mathbb{R}^n$ chosen randomly (but mutually different) from the finite data set $\boldsymbol{D}$.

2. Compute for each unit x $\in A$ its Voronoi set $\boldsymbol{R}_x$.
3. Move the reference vector of each unit to the mean of its Voronoi set:

$$\mathbf{w}_x = \frac{1}{|\mathbf{R}_x|} \sum_{\xi \in \mathbf{R}_x} \hat{\boldsymbol{I}}$$

4. If in step 3 any of the $\mathbf{w}_x$ did change, continue with step 2.
5. Return the current set of reference vectors.

The steps 2 and 3 together form a so-called *Lloyd iteration*, which is guaranteed to decrease the distortion error or leave it at least unchanged. LBG is guaranteed to converge in a finite number of Lloyd iterations to a local minimum of the distortion error function.

An extension of LBG, called LBG-U [8], is often able to improve on the local minima found by LBG. LBG-U performs non-local moves of single reference vectors which do not contribute much to error reduction (and are, therefore, not *useful*, thus the "U" in LBG-U) to locations where large quantization error does occur. Thereafter, normal LBG is used to find the nearest local minimum of the distortion error function. This is iterated as long as the LBG-generated local minima improve. LBG-U requires a finite data set, too, and is guaranteed to converge in a finite number of steps.

### 3.1. Growing Self-Organizing Map (GSOM)

Bauer et al. [1] proposed a grow algorithm called Growing Self-Organizing Map (GSOM), which can adapt both output space topology as well as the weight vectors. GSOM starts with 2 neurons configuration, learning using basic SOFM

algorithm, adds neurons to the output space according to a criterion, learns again and keeps on repeating the above operations until a pre-specified maximum number of neurons is reached. The growth can be achieved either by adding neurons in one of the directions which is already spanned by the output space or by adding a new dimension which is decided on the basis of the fluctuations within the masked Voronoi cells of the neurons. In this model, the author decomposes the re-construction error (i.e. $v$ - $w_i$) along the different directions, which result from projecting back the output space onto the input space. This reconstruction error is used as the criterion in the growth algorithm to add neurons in the direction, which has on average the largest error amplitude. The GSOM algorithm restricts the output space structure to the shape of a general hypercube with the overall dimensionality of the grid and its extension along the different directions being subject to adaptation. Details of the algorithm can be found on [32] where the authors modify the original GSOM algorithm and use it with spread factor to control the growth process applied in knowledge discovery.

### 3.2 Clustering algorithms (K-means/VQ)

A common self-organizing principle is to search for clusters in the input data distribution. In this context the data is not coded in terms of projections, as in the principal component case, but in terms of radial basis functions. The idea is to find the set of center locations $\mathbf{w}_k$, or cluster centers, that best describe the data set. This is done by minimizing, e.g. the reconstruction error:

$$E = \frac{1}{2} \sum_{n=1..N} \sum_{k=1..K} \Lambda_k [x(n)] \left\| x(n) - w_k \right\|^2 \qquad (+)$$

subjects to the constraint that K << N (K can be either fixed or adaptive).
Here $\Lambda_k$[x(n)] is a membership function that takes the values

$$\Lambda_k [x(n)] = \begin{cases} 1 & if \left\| x(n) - w_k \right\| < \left\| x(n) - w_j \right\| \quad \forall j \neq k \\ 0 & otherwise \end{cases}$$

Gradient descent on (+) leads to the updating equation

$$w_k (n+1) = w_k (n) + \eta \sum_{n=1}^{N} \Lambda_k [x(n)][x(n) - w_k]$$

which is the K-means algorithm. It is also sometimes called vector quantization

(VQ).

The K-means algorithm (actually the on-line form of it) is listed in Table 2. For this algorithm to really converge, it is necessary to let the learning rate (or step length) $\eta$ decrease with time.

Once all the center locations, or codebook vectors, $\mathbf{w}_k$ have converged to stable locations, then we can use them to code the data. The simplest coding would be to replace the observation $\mathbf{x}(n)$ with (the index of) its closest matching code-book vector. Another, slightly more complicated, is to replace $\mathbf{x}(n)$ with the indices of the $L$ closest code-book vectors, and the distance to them. This information can then be used to "triangulate" and thereby reproduce the observation with a higher precision than if only the winning unit is used. For more details on k-means algorithm, refer to [3].

**TABLE 2: *K*-means Algorithm**

1. Start with initial values for $\mathbf{w}_k$, $k = 1, \ldots K$, e.g. random.
2. Repeat until the reconstruction error defined in (+) is below some pre-specified value, or until the weight changes are very small.
   2.1 Present a pattern $\mathbf{x}(n)$.
   2.2 Find the center location $\mathbf{w}_k$ that lies closest to the presented pattern. That is, the weight vector $\mathbf{w}_k$ that fulfills the condition: $\|\mathbf{x}(n)-\mathbf{w}_k\| < \|\mathbf{x}(n)-\mathbf{w}_j\| \; \forall \; j \neq k$. Denote this center location as the "winner".

### 3.3. Neural Gas Algorithm

Martinetz et al proposed the Neural Gas (NG) network algorithm for vector quantization, prediction and topology representation. This algorithm has some advantage as compare with other algorithms: 1) converges quickly to a low distortion errors, 2) reachs a distortion error lower than resulting from K-means clustering, maximum-entropy clustering and Kohonen's SOFM, 3) obeys a gradient descent on an energy surface. Similar to SOFM algorithm, NG uses a *soft-max* adaptation rule (i.e. not only update the winning neuron, but also affects all the neurons denpending on their proximity to the input signal). This is mainly to generate the topographic map and also to aviod confinement to local minima

during the adaptation process.

In the NG algorithm, the synaptic weights are adapted without any fixed topological arrangement of the neural units within the network. Instead, it utilizes a *neighborhood-ranking* scheme for the synaptic weight vectors for a given data vector. The synaptic weight changes are not determined by the relative distances between the units within a topologically prestructure lattice, but by the relative distances between the units within input space: hence the name *Neural Gas* network.

Information about the arrangement of the receptive fields within the input space is implicitly given by a set of distortions, $D_x = \{||\boldsymbol{x} - \boldsymbol{w}_k||, k = 1,2, \ldots, N\}$, associate with each $\mathbf{x}$, where $N$ is the total number of units in the network[21]. Each time, an input signal $\mathbf{x}$ is presented to the network, the ordering of the elements of the set $D_x$ is necessary to determine the adjustment of the synaptic weight $\mathbf{w}_k$. This ordering has a time complexity of *O(NlogN)* in its sequential implementation. The resulting adaptation rule can be described as a *winner-takes-most* instead of *winner-takes-all* rule.

The neural gas algorithm is an approach to avoid the assumption of the topology of the SOM, and let the algorithm discover the topology itself. However, the algorithm uses a fixed number of units.
The neighborhood function is

$$\Lambda_{jj^*} = \exp\left[\frac{-k_j}{\Delta}\right]$$

where $k_j$ is the ranking that node $j$ has in relation to the input signal $\mathbf{x}$.
The winning node (i.e. the node closest to $\mathbf{x}$) is given the ranking $k_{j^*} = 0$. The second closest node is given the rank 1, and so on. The weights $\mathbf{w}_j$ are then updated according to:

$$\mathbf{w}_j = \mathbf{w}_j + \eta \exp[-k_j/\Delta](\mathbf{x}(n) - \mathbf{w}_j).$$

The topology of the map is constructed using a connectivity matrix $\boldsymbol{C}$ with the elements 0 or 1

$$C_{jk} = \begin{cases} 1 & \text{if nodes } j \text{ and } k \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases}$$

The connection elements $C_{jk}$ are then given an age, i.e. the ages for all elements C increase by one every iteration by the algorithm. If the age for a connection is above some threshold, then that connection is set ot 0. The connection element $C_{jj*}$ connecting the winning node with the second closest node is set to 1 at every iteration, and the age for the connection is set to zero.

The algorithm is described in the Table 3.

**TABLE 3: Self-organizing "neuron gas"**

1. Choose how many cluster centers to use.
2. Initiate all $\mathbf{w}_{ij}$ randomly. Set $C_{jk} = 0$ for all $j$ and $k$. Set the age of all connections to $\tau_{jk} = 0$.
3. Repeat until all $\mathbf{w}_j$ and $C_{jk}$ have converged (i.e. when they do not change much anymore):
   - 3.1 Select a random input pattern $\mathbf{x}(n)$ from the data set X.
   - 3.2 Find the winning node $\mathbf{y}_{j*}(n)$ and give it the rank $k_{j*} = 0$. Rank the remaining nodes by $k_j = 1, 2, 3, \ldots$ depending on how close they are to the input pattern.
   - 3.3 Update the weights according to:

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta \exp[-k_j/\Delta][\mathbf{x}(n) - \mathbf{w}_j(t)].$$

   You must let the step length $\eta \rightarrow 0$ as the algorithm converges. It is also common to let the neighborhood function's $\Delta \rightarrow 0$.
   - 3.4 Update the ages $\tau_{jk} = \tau_{jk} + 1$ for all $j$ and $k$. Set the connection $C_{jj*} = 1$ between the winning node and the second closest node, also set the age $\tau_{jj*} = 0$.

### 3.4. Competitive Hebbian Learning

This method (Martinetz and Schulten, 1991 [22]; Martinetz, 1993 [23]) is usually not used on its own but in conjunction with other methods. It is, however, instructive to study *competitive Hebbian learning* on its own. The method does not change reference vectors at all (which could be interpreted as having a zero

learning rate). It only generates a number of neighborhood edges between the units of the network. It was proved by Martinetz (1993) [23] that the so generated graph is optimally topology-preserving in a very general sense. In particular each edge of this graph belongs to the Delaunay triangulation corresponding to the given set of reference vectors. The complete *competitive Hebbian learning* algorithm is the following:

**TABLE 4: Competitive Hebbian Learning**

1. Initialize the set $A$ to contain $N$ units $\mathbf{x}_i$:

$$A = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

with reference vectors $w_{x_i} \in R^n$ chosen randomly. Initialize the connection set $C$, $C \subset A \times A$, to the empty set:

$$C = \varnothing$$

2. Generate at random an input signal $\mathbf{x}$.
3. Determine units $\mathbf{s}_1$ and $\mathbf{s}_2$ ($\mathbf{s}_1$, $\mathbf{s}_2 \in A$) such that

$$s_1 = \arg\min_{c \in A} \|x - w_c\|$$

and

$$s_2 = \arg\min_{c \in A \setminus \{s_1\}} \|x - w_c\|$$

4. If a connection between $\mathbf{s}_1$ and $\mathbf{s}_2$ does not exist already, create it:

$$C = C \cup \{(\mathbf{s}_1, \mathbf{s}_2)\}$$

5. Continue with step 2 unless the maximum number of signals is reached.

### 3.5. Neural Gas plus Competitive Hebbian Learning

This method [19,20] is a straight-forward superposition of *neural gas* and *competitive Hebbian learning*. It is sometimes denoted as "topology-representing networks" [20]. This term, however, is rather general and would apply also to the *growing neural gas* model described later.

At each adaptation step a connection between the winner and the second-nearest unit is created (this is *competitive Hebbian learning*). Since the reference vectors are adapted according to the *neural gas* method a mechanism is needed to remove edges which are not valid anymore. This is done by a local edge aging mechanism.

The complete *neural gas with competitive Hebbian learning* algorithm is in the following Table:

**TABLE 5: Neural Gas plus Competitive Hebbian Learning algorithm**

1. Initialize the set $A$ to contain $N$ units $\mathbf{x}_i$:

$$A = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$$

with reference vectors $w_{x_i} \in R^n$ chosen randomly.

Initialize the connection set $C$, $C \subset A \, X \, A$, to the empty set:

$$C = \varnothing$$

Initialize the time parameter $t$:

$$t = 0$$

2. Generate at random an input signal $\mathbf{x}$.

3. Order all elements of $A$ according to their distance to $\mathbf{x}$, i.e., find the sequence of indices $(i_0, i_1, \ldots, i_{N-1})$ such that $w_{i_0}$ is the reference vector closest to $\mathbf{x}$, $w_{i_1}$ is the reference vector second-closest to $\mathbf{x}$ and $w_{i_k}, k = 0,..N-1$ is the reference vector such that $k$ vectors $w_j$ exist with $\|x - w_j\| < \|x - w_k\|$. Following Martinetz et al. (1993) [24] we denote with $k_i(\mathbf{x}, A)$ the number $k$ associated with $w_i$.

4. Adapt the reference vectors according to

$$\Delta \mathbf{w}_i = \varepsilon(t).h_\lambda(k_i(\mathbf{x}, A)).(\mathbf{x} - \mathbf{w}_i)$$

with the following time-dependencies:

$$\lambda(t) = \lambda_i \left(\frac{\lambda_f}{\lambda_i}\right)^{\frac{t}{t_{\max}}},$$

$$\varepsilon(t) = \varepsilon_i \left(\frac{\varepsilon_f}{\varepsilon_i}\right)^{\frac{t}{t_{\max}}},$$

$$h_\lambda(k) = \exp\left(-\frac{k}{\lambda(t)}\right).$$

5. If a connection between $\mathbf{i}_0$ and $\mathbf{i}_1$ does not exist already, create it:

$$\boldsymbol{C} = \boldsymbol{C} \cup \{(\mathbf{i}_0, \mathbf{i}_1)\}$$

Set the age of the connection between $\mathbf{i}_0$ and $\mathbf{i}_1$ to zero ("refresh" the edge):

$$age_{(i_0,i_1)} = 0$$

6. Increment the age of all edges emanating from $\mathbf{i}_0$:

$$age_{(i_0,i)} = age_{(i_0,i)} + 1 \quad (\forall i \in N_{i_0})$$

Thereby, $N_x$ is the set of direct topological neighbors of $x$.

7. Remove edges with an age larger than the maximal age $T(t)$ whereby

$$T(t) = T_i \left( \frac{T_f}{T_i} \right)^{t/t_{\max}}.$$

8. Increase the time parameter $t$:

$$t = t + 1$$

9. If $t < t_{max}$ continue with step 2.

For the time-dependent parameters suitable initial values $(\lambda_i, \varepsilon_i, T_i)$ and final values $(\lambda_f, \varepsilon_f, T_f)$ have to be chosen.

### 3.6. Growing Neural Gas

This method ([5], [6]) is different from the previously described models since the number of units is changed (mostly increased) during the self-organization process. The growth mechanism from the earlier proposed *growing cell structures* ([2]) and the topology generation of *competitive Hebbian learning* ([21]) are combined to a new model. Starting with very few units new units are inserted successively. To determine where to insert new units, local error measures are gathered during the adaptation process. Each new unit is inserted near the unit which has accumulated most error. The complete *growing neural gas* algorithm is the following:

**TABLE 6: Growing Neural Gas Algorithm**

1. Initialize the set $\boldsymbol{A}$ to contain two units $\mathbf{x}_1$ and $\mathbf{x}_2$

$$\boldsymbol{A} = \{ \mathbf{x}_1, \mathbf{x}_2 \}$$

with reference vectors chosen randomly.

Initialize the connection set $C$, $C \subset A \, X \, A$, to the empty set:

$$C = \varnothing$$

2. Generate at random an input signal **x**.

3. Determine the winner $\mathbf{s}_1$ and the second-nearest unit $\mathbf{s}_2$ ($\mathbf{s}_1$, $\mathbf{s}_2 \in A$) such that

$$s_1 = \arg\min_{c \in A} \|x - w_c\|$$

and

$$s_2 = \arg\min_{c \in A \setminus \{s_1\}} \|x - w_c\|$$

4. If a connection between $\mathbf{s}_1$ and $\mathbf{s}_2$ does not exist already, create it:

$$C = C \cup \{(\mathbf{s}_1, \mathbf{s}_2)\}$$

Set the age of the connection between $\mathbf{s}_1$ and $\mathbf{s}_2$ to zero ("refresh" the edge):

$$age_{(s_1, s_2)} = 0$$

5. Add the squared distance between the input signal and the winner to a local error variable:

$$\Delta \mathbf{E}_{s_1} = \left\| \mathbf{x} - \mathbf{w}_{s_1} \right\|^2$$

6. Adapt the reference vectors of the winner and its direct topological neighbors by fractions $\varepsilon_b$ and $\varepsilon_n$, respectively, of the total distance to the input signal:

$$\Delta \mathbf{w}_{s_1} = \varepsilon_b (\mathbf{x} - \mathbf{w}_{s_1})$$
$$\Delta \mathbf{w}_i = \varepsilon_n (\mathbf{x} - \mathbf{w}_i) \quad (\forall i \in N_{s_1})$$

Thereby $N_{s_1}$ is the set of direct topological neighbors of $s_1$.

7. Increment the age of all edges emanating from $s_1$:

$$age_{(s_1, i)} = age_{(s_1, i)} + 1 \quad (\forall i \in N_{s_1})$$

8. Remove edges with an age larger than $a_{max}$. If this results in units having no more emanating edges, remove those units as well.

9. If the number of input signals generated so far is an integer multiple of a parameter $\lambda$, insert a new unit as follows:

- Determine the unit $q$ with the maximum accumulated error:

$$q = \arg\max_{x \in A} \mathbf{E}_x$$

- Determine among the neighbors of $q$ the unit $f$ with the maximum accumulated error:

$$f = \arg\max_{x \in Nq} \mathbf{E}_x$$

- Add a new unit $r$ to the network and interpolate its reference vector from $q$ and $f$.

$$A = A \cup \{r\}, \qquad \mathbf{w}_r = (\mathbf{w}_q + \mathbf{w}_f) / 2$$

- Insert edges connecting the new unit $r$ with units $q$ and $f$, and remove the original edge between $q$ and $f$:

$$C = C \cup \{(r, q), (r, f)\}, \quad C = C \setminus \{(q, f)\}$$

- Decrease the error variables of $q$ and $f$ by a fraction $\alpha$:

$$\Delta\mathbf{E}_q = -\alpha\mathbf{E}_q, \qquad \Delta\mathbf{E}_f = -\alpha\mathbf{E}_f,$$

- Interpolate the error variable of $r$ from $q$ and $f$:

$$\mathbf{E}_r = (\mathbf{E}_q + \mathbf{E}_f) / 2$$

10. Decrease the error variables of all units:

$$\Delta\mathbf{E}_x = -\beta\mathbf{E}_x \quad (\forall x \in A)$$

11. If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled continue with step 2.

### 3.7. Growing Cell Structures

This model [2] is rather similar to the *growing neural gas* model. The main difference is that the network topology is constrained to consist of $k$-dimensional simplices whereby $k$ is some positive integer chosen in advance. The basic building block and also the initial configuration of each network is a $k$-dimensional simplex. This is, e.g., a line for k=1, a triangle for k=2, and a tetrahedron for k=3.

For a given network configuration a number of adaptation steps are used to update the reference vectors of the nodes and to gather local error information at each node.

This error information is used to decide where to insert new nodes. A new node is always inserted by splitting the *longest edge* emanating from the node $q$ with maximum accumulated error. In doing this, additional edges are inserted such that the resulting structure consists exclusively of $k$-dimensional simplices again.

The *growing cell structures* learning procedure is described in the following

Table:

**TABLE 7: Growing Cell Structures Algorithm**

1. Choose a network dimensionality $k$.

Initialize the set $A$ to contain $k$+1 units $x_i$

$$A = \{\mathbf{x}_1, \ldots, \mathbf{x}_{k+1}\}$$

with reference vectors $w_{x_i} \in R^n$ chosen randomly.

Initialize the connection set $C$, $C \subset A \; X \; A$, such that each unit is connected to each other unit, i.e., such that the network has the topology of a $k$-dimensional simplex.

2. Generate at random an input signal $\mathbf{x}$.

3. Determine the winner $s$

$$s(x) = \arg\min_{c \in A} \|x - w_c\|$$

4. Add the squared distance between the input signal and the winner unit $s$ to a local error variable$\mathbf{E}_s$:

$$\Delta \mathbf{E}_s = \|\mathbf{x} - \mathbf{w}_s\|^2$$

5. Adapt the reference vectors of $s$ and its direct topological neighbors towards $\mathbf{x}$ by fractions $\varepsilon_b$ and $\varepsilon_n$, respectively, of the total distance:

$$\Delta \mathbf{w}_s = \varepsilon_b \, (\mathbf{x} - \mathbf{w}_s)$$

$$\Delta \mathbf{w}_i = \varepsilon_n \, (\mathbf{x} - \mathbf{w}_i) \; (\forall i \in N_s)$$

Thereby, we denote with $N_s$ the set of direct topological neighbors of $s$.

6. If the number of input signals generated so far is an integer multiple of a parameter $\lambda$, insert a new unit as follows:

- Determine the unit $q$ with the maximum accumulated error:

$$q = \arg\max_{c \in A} \mathbf{E}_c$$

- Insert a new unit $r$ by splitting the longest edge emanating from $q$, say an edge leading to a unit $f$. Insert the connections $(q,r)$ and $(r,f)$ and remove the original connection $(q,f)$. To re-build the structure such that it again consists only of $k$-dimensional simplices, the new unit $r$ is also connected with all common neighbors of $q$ and $f$, i.e., with all units in the set $N_q \cap N_f$.
- Interpolate the reference vector of $r$ from the reference vectors of $q$ and $f$:

$$\mathbf{w}_r = (\mathbf{w}_q + \mathbf{w}_f) / 2$$

- Decrease the error variables of all neighbors of $r$ by a fraction which depends on the number of neighbors of $r$:

$$\Delta \mathbf{E}_i = -\frac{\alpha}{|N_r|}\mathbf{E}_i \quad (\forall i \in N_r)$$

- Set the error variable of the new unit $r$ to the mean value of its neighbors:

$$\mathbf{E}_r = \frac{1}{|N_r|}\sum_{i \in N_r}\mathbf{E}_i$$

7. Decrease the error variables of all units:

$$\Delta \mathbf{E}_c = -\beta \mathbf{E}_c \qquad (\forall \boldsymbol{c} \in \boldsymbol{A})$$

8. If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled continue with step 2.

### 3.8. Growing Grid

*Growing grid* is another incremental network. The basic principles used also in *growing cell structures* and *growing neural gas* are applied with some modifications to a rectangular grid. Alternatively, *growing grid* can be seen as an incremental variant of the *self-organizing feature map*.

The model has two distinct phases, a *growth phase* and a *fine-tuning phase*. During the growth phase a rectangular network is built up starting from a minimal size by inserting complete rows and columns until the desired size is reached or until a performance criterion is met. Only constant parameters are used in this phase. In the fine-tuning phase the size of the network is not changed anymore and a decaying learning rate is used to find good final values for the reference vectors.

As for the self-organizing map, the network structure is a two-dimensional grid ($\boldsymbol{a}_{ij}$). This grid is initially set to 2x2 structure. Again, the distance on the grid is used to determine how strongly a unit $\tau = \boldsymbol{a}_{km}$ is adapted when the unit $\boldsymbol{s} = \boldsymbol{a}_{ij}$ is the winner. The distance measure used is the $\mathbf{L}_1$-norm

$$d_1(r, s) = |i - k| + |j - m| \qquad \text{for } r = \boldsymbol{a}_{km} \text{ and } \boldsymbol{s} = \boldsymbol{a}_{ij}$$

Also the function used to determine the adaptation strength for a unit $r$ given that $s$ is the winner is the same as for the *self-organizing feature map*:

$$h_{rs} = \exp\left(-\frac{d_1(r,s)^2}{2\sigma^2}\right).$$

The width parameter $\sigma$, however, remains constant throughout the whole simulation. It is chosen relatively small compared to the values usually used at the beginning for the *self-organizing feature map*. One can note that as the *growing grid* network grows, the *fraction* of all units which is adapted together with the winner decreases. This is also the case in the *self-organizing feature map* but is achieved there with a constant network size and a decreasing neighborhood width. The complete *growing grid* algorithm is the following:

**TABLE 8: Growing Grid Algorithm**

*Growth Phase*

1. Set the initial network width and height:

$$N_1 = 2, N_2 = 2$$

Initialize the set $A$ to contain $N = N_1 \, . \, N_2$ units $\mathbf{x}_i$

$$A = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$$

with reference vectors $w_{x_i} \in R^n$ chosen randomly..

Initialize the connection set $C$, $C \subset A \, X \, A$, to form a rectangular $N_1 \, x \, N_2$ grid.
Initialize the time parameter $t$:

$$t = 0$$

2. Generate at random an input signal $\mathbf{x}$.

3. Determine the winner $s(\mathbf{x}) = s$:

$$s(x) = \arg\min_{c \in A} \|\mathbf{x} - \mathbf{w}_c\|$$

4. Increase a local counter variable of the winner:

$$r_s = r_s + 1$$

5. Increase the time parameter $t$:

$$t = t + 1$$

6. Adapt each unit $r$ according to

$$\Delta \mathbf{w}_r = \varepsilon(t) h_{rs}(\mathbf{x} - \mathbf{w}_r)$$

whereby

$$\varepsilon(t) = \varepsilon_0$$

7. If the number of input signals generated for the current network size reaches a multiple $\lambda_g$ of this network size, i.e., if

$$\lambda_g \, . \, N_1 \, . \, N_2 = t$$

then do the following:

o Determine the unit q with the largest value of $\tau$:

$$q = \arg\max_{c \in A} \tau_c.$$

o Determine the direct neighbor $f$ of $q$ with the most distant reference vector:

$$f = \arg\max_{c \in Nq} \|\mathbf{w}_q - \mathbf{w}_c\| .$$

o Depending on the relative position of $q$ and $f$ continue with one of the two following cases:

Case 1: $q$ and $f$ are in the same *row* of the grid, i.e.

$$q = \boldsymbol{a}_{i,j} \text{ and } (f = \boldsymbol{a}_{i,j} \text{ or } f = \boldsymbol{a}_{i,j-1})$$

Do the following:

Insert a new column with $N_1$ units between the columns of $q$ and $f$.

Interpolate the reference vectors of the new units from the reference vectors of their respective direct neigbors in the same row.

Adjust the variable for the number of columns:

$$N_2 = N_2 + 1.$$

Case 2: $q$ and $f$ are in the same *column* of the grid, i.e.

$$q = \boldsymbol{a}_{i,j} \text{ and } (f = \boldsymbol{a}_{i+1,j} \text{ or } f = \boldsymbol{a}_{i-1,j})$$

Do the following:

Insert a new row with $N_2$ units between the rows of $q$ and $f$.

Interpolate the reference vectors of the new units from the reference vectors of their respective direct neigbors in the same columns.

Adjust the variable for the number of rows:

$$N_1 = N_1 + 1.$$

o reset all local counter values:

$$r_c = 0 \qquad (\forall c \in A)$$

o reset the time parameter:

$$t = 0.$$

8. If the desired network size is not yet achieved, i.e. if

$$N_1 \cdot N_2 < N_{min},$$

then continue with step 2.

*Fine-tuning Phase*

9. Generate at random an input signal $\mathbf{x}$.

10. Determine the winner $\boldsymbol{s}(\mathbf{x}) = \boldsymbol{s}$:

$$s(x) = \arg\min_{c \in A} \|\mathbf{x} - \mathbf{w}_c\|$$

11. Adapt each unit $r$ according to

$$\Delta \mathbf{w}_r = \varepsilon(t) h_{rs} (\mathbf{x} - \mathbf{w}_r)$$

whereby

$$\varepsilon(t) = \varepsilon_0 \, (\varepsilon_1 / \varepsilon_0)^{t/t_{max}}$$

with

$$t_{max} = \lambda_f \, . \, N_1 \, . \, N_2$$

12. If $t < t_{max}$ continue with step 9.

### 3.9. Other variances

Blackmore and Miikkulainen (1992) [13] let a irregular network grow on positions in the plane which are restricted to lie on a two-dimensional grid. Rodrigues and Almeida (1990) [14] increased the speed of the normal *self-organizing feature map* by developing an interpolation method which symmetrically increases the number of units in the network by interpolation. Their method is reported to give a considerable speed-up but is not able to choose, e.g., different dimensions for width and height of the grid as the approach of Bauer and Villmann (1995) [1] or the *growing grid.* Further approaches have been proposed, e.g. by Jokusch (1990) [20] and Xu (1990) [17].

Several other models without a fixed network dimensionality are known. DeSieno (1988) proposed a method [9] where frequent winners get a "bad conscience" for winning so often and, therefore, add a penalty term to the distance from the input signal. This leads eventually to a situation where each unit wins approximately equally often (entropy maximization).

Kangas et al. (1990) [12] proposed to use the minimum spanning tree among the units as neighborhood topology to eliminate the *a priori* choice for a topology in some models.

Another interesting algorithm was Structure Parameter Adaptive (SPA) neural tree. It was proposed by Li et al [13]. Tree structure classifiers have been widely used in pattern recognition tasks. It has been shown excellent results in the literature. The SPA neural tree can get adapt to a changing environment parametrically and structurally. In this architecture, no structural constraints are specified for the

neurons within the same level. That is, neurons in the same level are not ordered as a one or two dimensional array. The SPA neural tree begins with an empty structure and neurons are added to the tree when the error rate exceeds a threshold. Some neurons are deleted if they remains inactive for a long period. It uses a vigliance factor to controll the creation of new neurons and a threshold factor to controll the splitting of the neurons into more neurons. An operational measure is used to control the deletion of neurons from the tree. In the SPA neural tree architecture, the neurons of a sub-tree have similar synaptic weight vectors, which reflect that its architecture can be used as a hierachical classifier.

## 4. Applications

We can see the ability of mapping the arbitrary input space into a lower dimensional output space in orderly fashion of SOFMs makes it the powerful tool on image processing and signal processing. However, we can also use the SOFMs for other applications such as: time-series prediction, functions approximation, and so on. The process to apply SOFMs in image processing is clear since we can use 2-D-2 model (i.e. 2-dimensional input space and form a 2-Dimensional output space). In function approximation and time-series prediction, to apply SOFMs, we simply use the "pair" (input, output) to form the stimulus to the network. Still, the most important is the data and the ability of memory.

One of the earliest papers about the time-series prediction using a modification of SOFM is the work of T. Martinez [24]. The model was called Neural Gas (NG). In the NG algorithm, the synaptic weights are adapted without any fixed topological arrangement of the neural units within the network. Instead, it utilizes a *neighborhood-ranking* scheme for the synaptic weight vectors for a given data vector. The neighborhood ranking scheme and aging technique for the units that are activated makes the Neural Gas works more efficiently than a standard SOFM. Time-series prediction task can be taken as chaotic model building. Many approaches have been developed for chaotic model building. Traditionally, they have been categorized into local and global models. A global model is applicable to all neighborhoods, while a local model varies from neighborhood to neighborhood in the phase space. Most of the works trying to predict the future

using SOFM followed the local approach since the adaptation rule is only done within a neighborhood of activated unit.

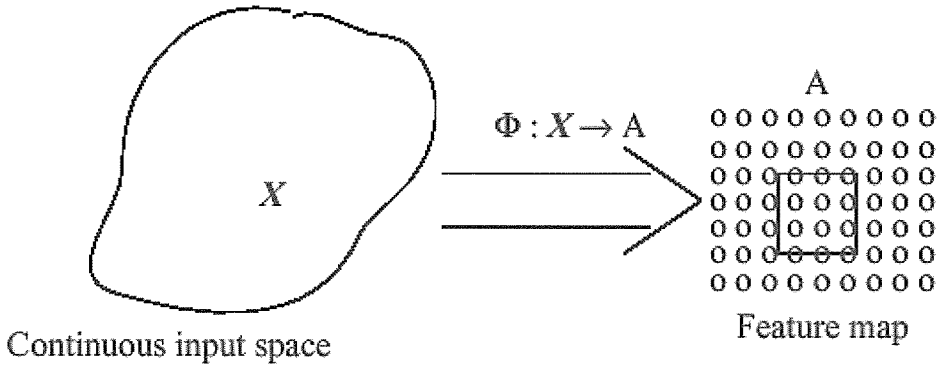


*Figure 5*: Feature Mapping

The SOFM has very interesting properties for time series modelling. Let Φ, ***X***, ***A*** denote the SOFM mapping, input sample space and the discrete output space respectively. When the network converges to its final stable state following a successful learning process, it displays four major remarkable properties:

1. The SOFM map Φ is a good approximation to the input space ***X***. This property is important since it provide a compact representation of the given input space.
2. The feature map Φ naturally forms a topologically ordered output space such that the spatial location of a neuron in the lattice corresponds to a particular domain in input space. The advantage of this feature is that it can simplify local modeling of the input signal ***X*** embedded in the ***A*** space.
3. The feature map Φ embodies a statistical law. In other words, the input with more frequent occurrence occupies a larger output domain of the output space ***A***. This property helps to make the SOFM an optimum codebook of the given input space.
4. A space dimension reduction is attained via the feature map Φ. That is, the continuous input space is mapped to a discrete output space with lower dimension. This property makes the simple architecture of codebook representation feasible.

The straightforward way to take advantage of the above properties for time series modeling is to create a SOFM from the input signal. Since such feature map Φ provides a faithful topologically organized output of the input vectors $\mathbf{x} \in \boldsymbol{X}$,

the local model fitting can then be performed over the compact codebook domain $A$. The proposed non-linear modelling scenario follows three steps: a. Reconstruction of the state space from the input signal; b. Embedding the state space in the neural field; c. Estimation of the locally linear predictors.

* *Reconstruction of the state space from the training signal.* Following the approach by Takens, a sequence of N+1 dimensional state vectors $[\mathbf{x}(n)^T, x(n+\tau)]^T$ is created from the given training time series, where $\mathbf{x}(n) = [x(n-(N-1)\tau), x(n-(N-2)\tau), \ldots, x(n)]$ and $\tau$ is the appropriate time delay where and D the dimension of the underlying dynamical process.

* *Embedding the state space in the neural field.* This step is accomplished via the Kohonen learning process. With each vector-scalar pair $[\mathbf{x}(n), x(n+1)]$ presented as the input to the network, the learning process of Kohonen feature mapping algorithm adaptively discretizes the continuous input space $\mathbf{X} \subset \mathbf{R}^{N+1}$ into a set of $\mathbf{K}$ disjoint cells $A$ to construct the mapping $\mathbf{\Phi}$. This process continues until the learning rate decreases close to zero and the neighborhood function covers about one output unit. After learning, a neural field representation $A$ of the input space $X$ via the constructed mapping relationship $\Phi$ is formed in terms of a set of disjoint units topologically organized in the output space.

* *Estimation of the locally linear predictors.* For each element its local linear predictor in terms of $[\mathbf{a}_i^T, b_i]$ is estimated based on which is a set of $L$ elements in the neighborhood of $u_i$ including $u_i$ itself. One example of $\alpha_i \in A$ is shown in Fig. 4. Each element $u_i$, has a corresponding weight vector $[\mathbf{w}_i^T, w_{i(N+1)}]$, where $\mathbf{w}_i^T = [w_{i(1)}, w_{i(1)}, \ldots, w_{i(N)}]$. The local prediction model $[\mathbf{a}_i^T, b_i]$ is fitted in the least-square sense to the set of weights in $\alpha_i$, i.e.

$$w_{j(N+1)} = b + \mathbf{a}^T \mathbf{w}_j$$

To ensure a stable solution of the above equations, $\alpha_i$ must have more than N+1 elements. Thereafter for each output unit $u_i$ there corresponds a unique linearly local model function $\tilde{f}(\mathbf{a}_i(^o), b_i(^o))$ in terms of the vector-scalar parameter pair $[\mathbf{a}_i, b_i]$.

The global dynamics of the given process can be described by the set of all the constructed local models pieced together. For an input state vector $\mathbf{x}(n)=[x(n-N+1), x(n-N+2),..., x(n)]^T$, the matched prototype element is found

based on the SOFM competition among all elements in $\boldsymbol{A}$. The predicted value x(n+1) is obtained by evaluating

$$\tilde{f}(\mathbf{a}(^{o}), b(^{o})) \qquad \text{at} \qquad \mathbf{x}(n)=[x(n\text{-}N+1), x(n\text{-}N+2),..., x(n)]^{T},$$

$$x(n+1) = \tilde{f}(\mathbf{a}(u_i o), b(u_i o), \mathbf{x}(n)) = b(u_i o) + \mathbf{a}^{T}(u_i o)\, \mathbf{x}(n)$$

In a similar manner, a K-step prediction x($n$+K) based on x(n) can also be obtained by iterative prediction, i.e. feeding the output back to the input,

$$x(n+K) = \tilde{f}_K(\tilde{f}_{K-1}(\ldots \tilde{f}_1(\mathbf{a}(u_i o), b(u_i o), \mathbf{x}(n))))$$

where $\tilde{f}_j = \tilde{f}(a(u_i o), b(u_i o), x(n))$ is the prediction function at step $j$. That is, the first prediction generates a new state, which is used to find the new local model function. Evaluation of the new local model function at the new state produces in turn a new prediction until the final K-step prediction. Compared with the direct prediction, this recursive prediction has the advantage of higher accuracy.

Application in function approximation can be found on [15, 18] where the authors used standard and a modification of SOFM algorithm. The approach is to use the pairs of (**x, y**) as input for the network to form the distribution for the pairs. However, the functions approximation application is somewhat similar to the time-series prediction application where the later stands for the more general case.

Another important application of SOFMs is to use it as a tool for knowledge discovering in Databases. One can find a very well known application that was developed by group of researchers led by Teuvo Kohonen called WEBSOM [34]. This application used SOFM algorithm as a tool to explore knowledge from a very large data on the Web. Another application can be found in [32], where the SOFM is modified as a dynamic growing grid structure with controlled by spread factor. It is shown that SOFM can be used as a tool for knowledge discovery in database.

## 5. Concluding remarks

Neural network is a very special model for computation. It has many remarkable abilities; especially in solving the problems that we know it has a solution existed but requires many efforts to solve in normal way (i.e. using mathematical tools or statistic mechanisms). However, it has a main drawback when compare with other

methods. It is the un-cleared reason why it came to the solution. It works as a black box and we still cannot explain well why neural network come to the solution. Many efforts tried to combine the neural network with other methods forming the hybrid systems to over come this problem.

One of the research directions to modify the standard SOFM toward accelerating the process to form the feature map [33] is to combine several techniques: K-means algorithm to select the size of the feature map to be formed, cluster centers from a data set; a heuristic assignment strategy is employed to organize the selected data points into an neural array so as to form an initial feature map; if the initial map is not good enough, then it will be fine-tuned by the traditional Kohonen self-organizing feature map (SOM) algorithm under a fast cooling regime in the third stage. By using the combination of the three techniques, the three-stage method, a topologically ordered feature map would be formed very quickly instead of requiring a huge amount of iterations to fine-tune the weights toward the density distribution of the data points, which usually happened in the conventional SOM algorithm. This is one of many efforts to modify the SOFM for other purposes using heuristic modifications.

In this paper, however, we just outline the SOFMs algorithm and its variances. This will give more details about current situation of SOFM. One should note that each variance of the SOFM algorithm might have good performance in some problems but not all the others. However, when we deal with a problem, one should consider carefully the feasible solutions, since SOFM and its modifications are not the global solution for a real world problem. As noted earlier, it is not suitable for:

- Kohonen nets work by clustering
- Nearby data points are expected to behave similarly, e.g. have similar outputs.
- Parity-like problems such as the XOR do not have this property. There would be unstable for solution by a Kohonen network.

When we have to solve such the kind of problems, consider using other kinds of networks such as: feed-forward neural networks with back-propagation learning or Boltzmann Machine or Hopfield network or even the traditional methods which do not have such limitations.

REFERENCE

[1] Bauer, H., Villmann, T., Growing a hypercubical output space in a self-organizing feature map, *IEEE Transactions on Neural Networks*, Vol. 8(2), pp218-226, 1997.

[2] B. Fritzke. Growing Cell Structures – A Self-Organizing Network for Unsupervised and Supervised learning. *Neural Network*, Vol. 7, No. 9, pp 1441-1460, 1994.

[3] B. Fritzke, Samples of Competitive learning algorithms implementation, URL: http://www.sund.de/netze/applets/gng/full/GNG_0.html

[4] B. Fritzke, "Come competitive learning method", URL: http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/t.html, Draft from April 5, 1997.

[5] B. Fritzke. "Fast learning with incremental RBF networks.", *Neural Processing Letters*, 1(1):-5, 1994b.
URL:
http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.incremental_rbf.ps.gz

[6] 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. MIT Press, Cambridge MA, 1995a.
URL:
http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.nips94.ps.gz

[7] B. Fritzke. "Incremental learning of local linear mappings.", In F. Fogelman and P. Gallinari, editors, *ICANN'95: International Conference on Artificial Neural Networks*, pages 217-222, Paris, France, 1995b. EC2 & Cie.
URL:
http://www.neuroinformatik.ruhr-uni-bochum.de/ini/PEOPLE/fritzke/papers/fritzke.icann95.ps.gz

[8] B. Fritzke. "The LBG-U method for vector quantization - an improvement over LBG inspired from neural networks.", *Neural Processing Letters*, 5(1), 1997.
URL:
ftp://ftp.neuroinformatik.ruhr-uni-bochum.de/pub/manuscripts/IRINI/irini97-01/irini97-01.ps.gz

[9] D. DeSieno. "Adding a conscience to competitive learning.", *IEEE International Conference on Neural Networks*, volume 1, pages 117-124, New York, 1988. (San Diego 1988) IEEE.
[10] E. Erwin, K. Obermayer, and K. Schulten. Self-Organizing Maps: Ordering, Convergence properties and Energy Functions. *Biological Cybernetics*, 67:47-55, 1992.
[11] M.T. Hagan, H. Demuth and M. Beale, "Neural Network Design", PWS Publishing, 1996.
[12] J. A. Kangas, T. Kohonen, and T. Laaksonen., "Variants of self-organizing maps.", *IEEE Transactions on Neural Networks*, 1(1):-99, 1990.
[13] J. Blackmore and R. Miikkulainen. "Incremental grid growing: encoding high-dimensional structure into a two-dimensional feature map", TR AI92-192, University of Texas at Austin, Austin, TX, 1992.
[14] J. S. Rodrigues and L. B. Almeida., "Improving the learning speed in topological maps of patterns.", *Proceedings of INNC*, pages 813-816, Paris, 1990.
[15] J. Walter, H. J. Ritter, and K. J. Schulten. "Non-linear prediction with self-organizing maps.", *International Joint Conference on Neural Networks*, pages I.589-594, San Diego, 1990.
[16] Li, T., Y.Y. Tang, and L.Y. Fang, *A structure parameter adaptive (SPA) neural tree for the recognition of large character set*,Vol. 28, No. 3, pp. 315-329, 1995.
[17] L. Xu. ,"Adding learning expectation into the learning procedure of self-organizing maps.", *Int. Journal of Neural Systems*, 1(3):-283, 1990.
[18] M. Aupetit, et al., "C-SOM: A Continuous Self-Organizing Map For Function Approximation", *Proc. of Intelligent System and Control (ISC99)*, Santa-Barbar, October 1999.
[19] S. Haykin, "Neural Networks: A Comprehensive Foundation", Prentice Hall, New Jersey, 2nd Edition, 1999.
[20] S. Jokusch. "A neural network which adapts its structure to a given set of patterns", In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 169-172. Elsevier Science Publishers B.V., 1990.
[21] T. Martinetz and K. Schulten, A Neural Gas Learns Topologies, *Artificial Neural Networks*, pp 397-402. Noth Holland, Amsterdam, 1991.

[22] T. Martinetz and K. Schulten, Topology Representing Network, *Neural Networks*, Vol. 7 No. 3, pp 507-522. 1994.

[23] T. M. Martinetz. "Competitive Hebbian learning rule forms perfectly topology preserving maps.", *ICANN'93: International Conference on Artificial Neural Networks*, pages 427-434, Amsterdam, 1993. Springer.

[24] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-gas network for vector quantization and its application to time-series prediction.", *IEEE Transactions on Neural Networks*, 4(4):-569, 1993.

[25] T. Kohonen. "Self-organized formation of topologically correct feature maps.", *Biological Cybernetics*, 43:-69, 1982.

[26] D. J. Willshaw and C. von der Malsburg., "How patterned neural connections can be set up by self-organization.", *Proceedings of the Royal Society London*, volume B194, pages 431-445, 1976.

[27] Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997, URL: http://www.cs.rhul.ac.uk/NCS/vol1_4.ps.gz

[28] E. Forgy., "Cluster analysis of multivariate data: efficiency vs. interpretanility of classifications.", *Biometrics*, 21:, 1965. abstract.

[29] Y. Linde, A. Buzo, and R. M. Gray., "An algorithm for vector quantizer design.", *IEEE Transactions on Communication*, COM-28:-95, 1980.

[30] S. P. Lloyd., "Least squares quantization in pcm.", *Technical note*, Bell Laboratories, 1957., published in 1982 in *IEEE Transactions on Information Theory*.

[31] R. M. Gray., "Vector Quantization and Signal Compression.", *Kluwer Academic Press*, 1992.

[32] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan, "Dynamic Self-Organizing Maps with Controlled Growth for Knowledge Discovery", *IEEE TRANSACTIONS ON NEURAL NETWORKS*, VOL. 11, NO. 3, pp 601-614, MAY 2000.

[33] Mu-Chun Su and Hsiao-Te Chang, "Fast Self-Organizing Feature Map Algorithm", *IEEE TRANSACTIONS ON NEURAL NETWORKS*, VOL. 11, NO. 3, pp 721-733, MAY 2000.

[34] K. Lagus, "Text mining with the WEBSOM", PhD dissertation, Helsinki University of Technology, 2000.