A Robust and Flexible model of Hierarchical Self Organizing Maps for Nonstationary Environments

R. Salas\textsuperscript{a,b,*}, S. Moreno\textsuperscript{b}, H. Allende\textsuperscript{b}, C. Moraga\textsuperscript{c,d},

\textsuperscript{a}Universidad de Valparaíso; Departamento de Computación; Valparaíso, Chile.
\textsuperscript{b}Universidad Técnica Federico Santa María; Dept. de Informática; Casilla 110-V; Valparaíso, Chile.
\textsuperscript{c}European Centre for Soft Computing; E-33600 Mieres, Spain.
\textsuperscript{d}University of Dortmund; Department of Computer Science; D-44221 Dortmund, Germany.

Abstract

In this paper we extend the Hierarchical Self Organizing Maps model (HSOM) to address the problem of learning topological drift under non stationary and noisy environments. The new model combines the capabilities of robustness against noise and, at the same time, the flexibility to adapt to the changing environment. We call this model RoFlex-HSOM.

The RoFlex-HSOM model consists in a hierarchical tree structure of growing self organizing maps that adapts its architecture based on the data. The model preserves the topology mapping from the high-dimensional time dependent input space onto a neuron position in a low-dimensional hierarchical output space grid. Furthermore the RoFlex-HSOM algorithm has the plasticity to track and adapt to the topological drift, it gradually forgets (but no catastrophically) previous learned patterns and it is resistant to the presence of noise. We empirically show the capabilities of our model with experimental results using synthetic sequential data sets and the “El Niño” real world data.

Key words: Time-dependent non-stationary environments, Topological Drift, Robust and Flexible Architectures, Hierarchical Self Organizing Maps

* Corresponding author. Departamento de Computación, Universidad de Valparaíso, Av. Gran Bretaña 1091, Playa Ancha, Valparaíso, Chile.

Email addresses: rodrigo.salas@uv.cl (R. Salas), smoreno@inf.utfsm.cl (S. Moreno), hallende@inf.utfsm.cl (H. Allende), moraga@uni-dortmund.de (C. Moraga).
1 Introduction

In most real world applications the environments are non-stationary and the problem may change over time. Consider, for example, the weather prediction problem, where the environment may vary radically with the change of season. Another example is the detection and filtering of spam e-mails, where the descriptions of the classes evolve with time. In this work we deal with the El Niño Southern Oscillation (ENSO) cycles consisting of oceanographic and surface meteorological readings taken from a several buoys positioned throughout the equatorial Pacific.

L. Kuncheva [15] mentions that a model, if intended for real world applications, should be equipped with a mechanism to adapt to the changes in the environment. Furthermore, L. Kuncheva [15], based on the works of [1], [12] and [23], recognizes that the type of changes could be roughly summarized as random noise, random trends (gradual changes), random substitutions (abrupt changes) and systematic trends (“recurring contexts”). These changes may affect both the underlying probability distributions and the classification task. In the machine learning literature the change in the probabilities is known as population drift, otherwise, if the task is changing the problem is termed concept drift.

Unfortunately, several simulations results show that when static models have sequential learning in non-stationary environments, they catastrophically forget the previously learned patterns as was shown in [9], [7] and [12].

G. Widmer [24] remarks that one of the difficult problems in incremental learning is distinguishing between “real” concept drift and slight irregularities that are due to noise in the training data. If the model has the flexibility to react and adapt rapidly to every concept drift will over react to noise by incorrectly interpreting it as concept drift. On the other hand if the model is highly robust to noise will react very slowly and late to the concept drift.

An ideal learner should combine the seemingly incompatible capabilities of robustness against noise and the flexibility of effectively tracking the concept. There exists a clear compromise between flexibility and robustness in the incremental learning systems, as was noted by Grossberg [9] and Widmer [24].

In this paper we extend the Hierarchical Self Organizing Maps model of Rauber et al. [18] to non stationary time-dependent environments by incorporating Robustness and Flexibility to the incremental learning algorithm. We call this extended algorithm RoFlex-HSOM, where Ro stands for Robustness, Flex for Flexibility and HSOM for Hierarchical Self Organizing Maps. This extended model preserves the topology to represent the hierarchical relation of the data in non stationary environments. Furthermore the RoFlex-HSOM
model combines both, the architectural flexibility of adapting to the changing environment and the resistance to irrelevant noise. In addition we have introduced a forgetting operator to the model that will gradually forget (but no catastrophically) previous learned patterns.

The remainder of this paper is organized as follows. In the next section we briefly discuss the difficulty of modeling time-dependent and non stationary environments where we state the concept drift problem and the artificial neural networks Catastrophic Interference problem. In the section 3 we briefly introduce the Self Organizing Maps model. Then, in section 4, we introduce the M-estimators as a robust method for parameter estimation. Our proposal of the RoFlex-HSOM model is stated in section 5. In section 6 we provide some simulation results on synthetic and real data sets. Conclusions and further work are given in the last section.

2 The difficulty of modeling environments which are time-dependent and non stationary

2.1 The concept drift problem

A difficult problem with learning in many real world domains is the non stationary of the environment to be modeled. The task may change over time and an effective learner should be able to track such changes and to quickly adapt to them. In the literature when the change occurs in the concept then it is known as Concept Drift, on the other hand, when the change occurs in the underlying data distribution then it is known as population drift.

A. Tsymbal [22] recognizes three approaches to handling concept drift in the available systems: (1) Instance Selection, (2) Instance weighting, and (3) multiple concept descriptions. In [22] several algorithms that belong to some of these classes are mentioned, we just make a brief description in what follows.

In instance selection, the goal is to select instances relevant to the current concept. The most common technique is to use a sliding window that keeps the most recent data. Examples of this category are the Flora family of algorithms [23].

The instance weighting uses the ability of some learning algorithms to process weighted instances. In [13] a Support Vector Machine is used to process weighted instances but the author shows in his experiments that this approach has not been successful.
In multiple concept descriptions several models are used to learn the concepts. Ensemble and modular algorithms are used as incremental learning that uses some criteria to dynamically create, reactive or eliminate a member of the model. An example is the STAGGER algorithm [20].

Most of the existing models that handle concept drift are for classification tasks and we did not find any work that tracks change in the topology of the environment. In this work we extend the Hierarchical Self Organizing Maps (HSOM) model to handle the population and topological drift of the environment, we use a modular learning (multiple concept description) with a discrete sliding window (instance weighting) approach.

2.2 The catastrophic interference problem

Artificial neural networks with highly distributed memory forget catastrophically when faced with sequential learning tasks in non stationary environments, i.e., the new learned information most often erases the one previously learned. This major weakness is not only cognitively implausible, as human gradually forget, but disastrous for most practical applications. (See [7] and [16] for a review)

Catastrophic interference is a radical manifestation of a more general problem for connectionist models of memory, the so-called stability-plasticity problem (see [9] and [6]). The problem is how to design a system that is simultaneously sensitive to, but not radically disrupted by, a new input. A number of ways have been proposed to avoid the problem of catastrophic interference in connectionist networks (see [5], [7]).

An adequate system must be capable of plasticity in order to learn about significant new events, yet it must also remain stable in response to irrelevant or often repeated events. Thus a primary goal of the present article is to characterize a model capable of self-stabilizing the self-organization of their recognition codes in response to an arbitrary complex environment of input patterns in a way that parsimoniously reconciles the requirements of plasticity, stability and low complexity.

3 Self Organizing Maps

Self Organizing Maps (SOM) model was introduced by T. Kohonen [14] and it is a special kind of artificial neural network with unsupervised learning. The model preserves the topology mapping from the high-dimensional input space.
onto a low-dimensional display. This model and its variants have been very successful in several real applications areas (see [21] for examples).

In this section we briefly introduce the SOM model, for further details please refer to [14]. The Map $\mathcal{M}$ consists of an ordered set of prototypes $\mathbf{m}_r \in \mathbb{R}^d$ with a neighborhood relation between these units forming a grid, where $r$ indexes the location of the prototype in the grid. The most common used lattices are the linear, the rectangular and the hexagonal array of cells. In this paper we will consider a rectangular grid where $\mathbf{r}(\mathbf{m}_r) = (i, j) \in \mathbb{N}^2$ is the vectorial location of the unit $\mathbf{m}_r$ in the grid, where $i$ and $j$ stand for the row and column of the prototype in the rectangular array.

When the data vector $\mathbf{x} \in \mathbb{R}^d$ is presented to the model $\mathcal{M}$, it is projected to a neuron position of the low dimensional grid by searching the best matching unit ($bmu$), i.e., the prototype that is closest to the input, and it is obtained as follows

$$
c(\mathbf{x}) = \arg \min_{\mathbf{m}_r \in \mathcal{M}} \{ \| \mathbf{x} - \mathbf{m}_r \| \} \tag{1}
$$

where $\| \cdot \|$ is some norm, for example the classical Euclidean norm.

The learning process of this model consists in moving the reference vectors towards the current input by adjusting the location of the prototype in the input space. The winning unit and its neighbors adapt to represent the input by applying iteratively the following learning rule:

$$
\mathbf{m}_r(t + 1) = \mathbf{m}_r(t) + h_{c(\mathbf{x})}(r, t)[\mathbf{x}_i - \mathbf{m}_r(t)] \quad \text{for all } \mathbf{m}_r \in \mathcal{M} \text{ and } i = 1..n \tag{2}
$$

The amount the units learn is governed by a neighborhood kernel $h_{c(\mathbf{x})}(r, t)$, which is a decreasing function of the distance between the unit $\mathbf{m}_r$ and the $bmu$ $\mathbf{m}_{c(\mathbf{x})}$ on the map lattice at time $t$. The kernel is usually given by a Gaussian function,

$$
h_{c(\mathbf{x})}(r, t) = \alpha(t) \exp \left( -\frac{\| \mathbf{r}(\mathbf{m}_r) - \mathbf{r}(\mathbf{m}_{c(\mathbf{x})}) \|^2}{2\sigma(t)^2} \right) \tag{3}
$$

where $0 < \alpha(t) < 1$ is the learning rate parameter and $\sigma(t)$ is the neighborhood range. The vector $\mathbf{r}(\mathbf{m}_r)$ and $\mathbf{r}(\mathbf{m}_{c(\mathbf{x})})$ are the vectorial location of the unit $\mathbf{m}_r$ and the $bmu$ $\mathbf{m}_{c(\mathbf{x})}$ in the grid respectively. In practice the neighborhood kernel is chosen to be wide at the beginning of the learning process to guarantee global
ordering of the map, and both its width and height decrease slowly during learning. The learning parameter function \( \alpha(t) \) is a monotonically decreasing function with respect to time, for example this function could be linear \( \alpha(t) = \alpha_0 + (\alpha_f - \alpha_0)t/t_a \) or exponential \( \alpha(t) = \alpha_0(\alpha_f/\alpha_0)^{t/t_a} \), where \( \alpha_0 \) is the initial learning rate (\(< 1\)), \( \alpha_f \) is the final rate (\( \approx 0.01 \)) and \( t_a \) is the maximum number of iteration steps to arrive \( \alpha_f \).

The disadvantage of these models is that the neural designer has to decide in advance the architecture of the SOM that will be used. Unfortunately this is a difficult task due to the high dimensionality of the data. To overcome the architectural design problem several algorithms with an adaptive structure during the training process have been proposed, as for example, we refer to the growing cell structures [8], the Dynamic SOM [2], the GHSOM [18], and the HDGSOM [4].

4 Robust M-Estimators for the Learning Process

The learning process of Artificial Neural Networks models can be seen as a parameter estimation process, and their inference relies on the data [3]. When observations substantially different from the bulk of data exist, they can influence badly the model structure bringing degradation in the estimates. In this work we seek for a robust estimator for the parameters of the RoFlex-HSOM by applying M-estimators introduced by Huber [11].

Let the data set \( \{x_1, \ldots, x_n\} \) consist of an independent and identically distributed (i.i.d.) sample of size \( n \) obtained from the input space \( \mathcal{X} \subseteq \mathbb{R}^d \) of dimension \( d \), i.e., \( x_i \in \mathcal{X} \). An M-estimator \( \hat{\theta}_n^M \) is defined as

\[
\hat{\theta}_n^M = \arg \min\{RL_n(\theta) : \theta \in \Theta\} \quad \text{with} \quad RL_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \rho(x_i, \theta) \tag{4}
\]

where \( \Theta \subseteq \mathbb{R}^D \) is the parametric space, \( RL_n(\theta) \) is a robust functional cost and \( \rho : \mathcal{X} \times \Theta \rightarrow \mathbb{R} \) is the robust function that introduces a bound to the influence of outliers data during the training process. By assuming that the function \( \rho \) is differentiable with respect the parameter \( \theta = (\theta_1, \ldots, \theta_D) \), we obtain the score function \( \psi(x, \theta) = (\psi_1, \ldots, \psi_D)' \) whose components are the partial derivatives \( \psi_j(x, \theta) = \frac{\partial \rho(x, \theta)}{\partial \theta_j} \), \( j = 1 \ldots D \). Then the M-estimator can be defined implicitly as the solution of the system equation,

\[
\frac{1}{n} \sum_{i=1}^{n} \psi_j(x_i, \theta) = 0, \quad j = 1 \ldots D \tag{5}
\]
If the vector parameter \( \theta \) is the estimator of location, with \( d = D \), then we can define the robust function as a function of the difference between the data sample and the parameter \( x - \theta \). Unfortunately, location M-estimators are usually not invariant with respect to scale, which is often a nuisance parameter. To overcome this problem we can define the standardized residual \( z_i \) for each data \( x_i = (x_1, ..., x_d)' \) as \( z_i = (z_1, ..., z_d)' = \hat{S}^{-1/2}(x - \theta) \), where \( \hat{S} \) is the robust estimation of the covariance matrix of the difference \( x - \theta \) and \( \hat{S}^{-1/2} = \sqrt{\hat{S}^{-1}} \) is the square root of the inverse of the covariance matrix\(^1\). Now we can redefine the robust functional cost of equation (4) as a function of the standardized residual \( z_i \) as follows

\[
RL_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \rho \left( \hat{S}^{-1/2}(x - \theta) \right) = \frac{1}{n} \sum_{i=1}^{n} \rho (z_i) \tag{6}
\]

One could compute \( \theta \) and \( \hat{S} \) simultaneously as M-estimator of location and scale respectively. However, simulations have shown the superiority of M-estimators with initial scale estimation given by the scaled version of the median of the absolute deviations from the median (sMAD):

\[
sMAD(x_1, ..x_n) = \frac{1}{\Phi^{-1}(\frac{3}{4})} \text{median}_{i=1..n} \left\{ \left| x_i - \text{median}_{k=1..n}(x_k) \right| \right\} \tag{7}
\]

where \( \Phi^{-1}(p) \) is the inverse of the standard Gaussian accumulative distribution function at the probability \( p \). The constant \( \frac{1}{\Phi^{-1}(\frac{3}{4})} \approx 1.483 \) is needed to make the sMAD scale estimator \( \hat{S} \) Fisher consistent when the data behave as Gaussian distribution. The estimator of the covariance matrix is obtained as:

\[
\hat{S} = [s^2_{ij}]_{i=1..d, j=1..d} \quad \text{where} \quad s^2_{ij} = \begin{cases} 
\{sMAD(x_1(j), ..., x_n(j))\}^2 & \text{if } i = j \\
0 & \text{otherwise} 
\end{cases} \tag{8}
\]

where \( x_l(j), j = 1..d \) is the \( j \)-th component of the vector \( x_l, l = 1..n \). We have assumed independence between the components of the vector \( x_l \).\(^2\)

Some famous examples of M-estimators are the least square method \( \rho^{LS}(z) = z'z \) with the sample mean \( \overline{Z} = \frac{1}{n} z_i \) as the estimator, and, for a given probability

\(^1\) The covariance matrix \( \hat{S} \) is a symmetric matrix and positive definite

\(^2\) Note that me matrix built as equation (8) is a positive definite diagonal matrix and its inverse is given by inverting its diagonal. Furthermore the square root is given by the square root of its elements.
density function $f_\theta(z)$, the choice $\rho^{ML}(z) = -\log f_\theta(z)$ yields the maximum likelihood estimator (MLE). Unfortunately these estimators are non-robust. Example of robust estimators are the least absolute values (LAV) method $\rho^{LAV}(z) = |z|$ with the sample median $\text{median}_{i=1..n}(z_i)$ as the estimator, and the Huber estimator proposed by P. Huber [11] whose robust function $\rho^H(z)$ and score function $\psi^H(z)$ are given by

$$
\rho^H(z) = (\rho^H_1(z_1), \ldots, \rho^H_d(z_d))' \quad \psi^H(z) = (\psi^H_1(z_1), \ldots, \psi^H_d(z_d))'
$$

$$
\rho^H_j(z_j) = \begin{cases} 
-\kappa z_j - \frac{\kappa^2}{2} & \text{if } z_j < -\kappa \\
\frac{1}{2} z_j^2 & \text{if } -\kappa \geq z_j \geq \kappa \\
\kappa z_j - \frac{\kappa^2}{2} & \text{if } z_j > \kappa 
\end{cases} \quad \psi^H_j(z_j) = \begin{cases} 
-\kappa & \text{if } z_j < -\kappa \\
z_j & \text{if } -\kappa \geq z_j \geq \kappa \\
\kappa & \text{if } z_j > \kappa 
\end{cases}
$$

where $\kappa > 0$. Please refer to [10] for other examples of robust functions.

5 The Robust and Flexible Hierarchical Self Organizing Maps model (RoFlex-HSOM)

The RoFlex-HSOM model consists of a hierarchical tree structure of growing self organizing maps that automatically adapts its structure based on the data. The construction of the model was based in previous works of A. Rauber et al. [18], S. Moreno et al. [17] and R. Salas et al. [19]. Figure 1 schematically shows the RoFlex-HSOM model. The model preserves the topology mapping from the high-dimensional input space onto a neuron position in a low-dimensional output space grid. It has the capability of detecting novel data or clusters and creates new maps to learn these patterns avoiding that other maps catastrophically forget. Furthermore the maps with low volume of data can gradually forget by reducing their size and contracting their grid lattice. In addition we incorporate several robust strategies, such as outlier resistant scheme and novel data identification.

The construction and learning process of the RoFlex-HSOM model has two parts. The first part consists in constructing a base structure created to learn the data for the first time. After we have the base structure and more streaming data are arriving, the second part the model learns and adapt to the new environment without starting again from scratches. The description of the algorithm is given in the next subsections.

In what follows we introduce the notation that is explained below. Let $m_0$ be the root neuron of the hierarchical structure. For each layer of the hierarchi-
The hierarchical tree structure of the RoFlex-HSOM model. The root neuron $m_0$ is located at level 0. The growing maps are showed in the successive levels. The input data are projected into this hierarchical structure.

Fig. 1. The hierarchical tree structure of the RoFlex-HSOM model. The root neuron $m_0$ is located at level 0. The growing maps are showed in the successive levels. The input data are projected into this hierarchical structure.

Fig. 1. The hierarchical tree structure of the RoFlex-HSOM model. The root neuron $m_0$ is located at level 0. The growing maps are showed in the successive levels. The input data are projected into this hierarchical structure.

Fig. 1. The hierarchical tree structure of the RoFlex-HSOM model. The root neuron $m_0$ is located at level 0. The growing maps are showed in the successive levels. The input data are projected into this hierarchical structure.

Fig. 1. The hierarchical tree structure of the RoFlex-HSOM model. The root neuron $m_0$ is located at level 0. The growing maps are showed in the successive levels. The input data are projected into this hierarchical structure.
Euclidean norm \( \| \mathbf{x} \| = \sqrt{\mathbf{x}^T \mathbf{x}} \) unless otherwise is stated.

5.1 First Part: Constructing the Base Model

In this section we explain how to construct the base RoFlex-HSOM model \( \mathcal{H}_1 \) at stage 1 when the dataset \( \mathcal{I}_1 \) is presented. To construct the hierarchical model of self organizing maps, we need to explain how to locate the first neuron, how to expand the model hierarchically, how to train and grow the Self Organizing Maps. At the end of this subsection we summarize with algorithms the process of constructing the base model structure.

5.1.1 Locating the root neuron

The construction of the model begins with the location of the root neuron \( m_0 \) as the centroid of the whole data set \( \mathcal{I}_1 \). To accomplish this we robustly estimate the centroid \( m_0 \) by minimizing the robust functional \( RL_n(m_0) = \frac{1}{|\mathcal{I}_1|} \sum_{x_i \in \mathcal{I}_1} \rho \left( \hat{S}_0^{-1/2}(x_i - m_0) \right) \), where \( \hat{S}_0 \) is the robust scale estimator of the covariance matrix. \( \hat{S}_0 \) was estimated with equation (8) and considering the whole data set \( \mathcal{I}_1 \).

After the location of the first neuron, a new square grid with initial size of \( 2 \times 2 \) units is created beneath it. The map could be created randomly or deterministically around the root neuron \( m_0 \). We will consider the prototype \( m_0 \) as the parent of the recently created map.

5.1.2 Robust Learning Algorithm of the Self Organizing Map (RSOM)

In [3] we introduced a robust learning algorithm for the Kohonen’ SOM [14]. The influence of outliers was diminished by introducing a robust score function \( \psi(\cdot) \) in the update rule (2) as follows:

\[
m_r(t+1) = m_r(t) + h_c(x_i)(r,t) \psi \left( \hat{S}_r^{-1/2}[x_i - m_r(t)] \right), \quad m_r \in \mathcal{M} \text{ and } x_i \in \mathcal{C}(\mathcal{M})
\]

(11)

where the neighborhood kernel \( h_c(x)(r,t) \) is given by equation (3) and the bmu \( m_c(x) \) is determined with equation (1). \( \hat{S}_r \) is the robust scale estimator for the neuron \( m_r \), the estimation of \( \hat{S}_r \) is accomplished with equation (8) but the vector set of data \( \left\{ \frac{h_c(x_i)}{\alpha(t)}(x_i - m_r), x_i \in \mathcal{C}(\mathcal{M}) \right\} \) is considered instead.
5.1.3 Quality Measure.

In order to decide if certain map needs to grow or we need to introduce more levels of description to the model, we quantify the quality of adaptation of the prototypes. The quality of adaptation of the prototype \( m_r \in H_T \) is measured with the robust quantization error \( rqe_r \) of the unit. The computation of the \( rqe_r \) value will depend on whether the unit is a leaf \( (m_r \in N_L) \) or an internal node \( (m_r \in N_I) \) as:

\[
rqe_r = \begin{cases} 
\sum_{x_i \in C(m_r)} \rho \left( \hat{S}_x^{-1/2}(x_i - m_r) \right) & \text{if } m_r \in N_L \\
\frac{1}{|M_u|} \sum_{m_i \in M_u} rqe_i & \text{if } m_r \in N_I
\end{cases}
\]  

where, for the case of an internal node \( (m_r \in N_I) \), \( M_u \) is its child map. The robust quantization error of the root neuron \( rqe_0 \) is computed over the whole data set \( \mathcal{I}_T \).

5.1.4 Hierarchical Growth

To decide if a neuron \( m_r \) needs to grow hierarchically we need a criterion of how good the prototype represents the data of its respective Voronoi polygon \( C(m_r) \). The robust quantization error \( rqe_r \) of all leaf units \( m_r \in N_L \) are compared with the \( rqe_0 \) of the root neuron. If the leaf unit satisfies the following hierarchical growth criterion:

\[
rqe_r \geq \tau \cdot rqe_0 \quad \text{and} \quad |C(m_r)| \geq N_{min} \quad 0 < \tau << 1
\]

then, a new map with initial size of \( 2 \times 2 \) units is created randomly or deterministically around the unit \( m_r \). We will consider the prototype \( m_r \) as the parent of the recently created map. The constant \( N_{min} \) is the minimal quantity of data required to create a new map. The equation (13) means that the \( rqe_r \) of the \( m_r \) prototype must be at most \( \tau \cdot 100\% \) of the \( rqe_0 \) of the root unit.

5.1.5 Growth Process

During the growth process the map can extend its grid by adding rows or columns of prototypes inside or outside the lattice. To decide if the map \( \mathcal{M} \) needs to grow in order to reach a desired description of the data, the \( rqe_r \) are computed for all the units \( m_r \) that belong to this map. Then, we determine the maximum error unit \( m_e \) as the prototype with the highest robust quantization error, i.e, \( m_e = \arg \max \{rqe_r | m_r \in \mathcal{M}\} \).
Fig. 2. The internal growth process. A column of prototypes is inserted in between the units $m_e$ and $m_d$.

To decide if the map will introduce new prototypes inside the grid then we compute the map’s robust quantization error as:

$$MRQE = \sum_{x_i \in \mathcal{C}(M)} \rho \left( \hat{S}^{-1/2}(x_i - \bar{m}) \right)$$

where $\bar{m}$ is the center of the map computed with equation (10), and $\hat{S}$ is the scale estimator obtained with equation (8) considering the set $\mathcal{C}(M)$. If the map accomplish this growth criterion

$$\frac{1}{M_m} \sum_{m_r \in M} rqe_r \geq \gamma MRQE, \quad \text{where } 0 < \gamma << 1$$

then the map grows its grid by adding prototypes inside the grid. In the inside growing process, the most dissimilar neighbor $m_d$ to the unit $m_e$ is detected as the farthest neighbor unit, $m_d = \arg \max \{ \| m_r - m_e \| | m_r \in \mathcal{N}_e \}$, where $\mathcal{N}_e$ is the set of neighboring units of the error unit $m_e$. A row or column of units is inserted between $m_e$ and $m_d$ and their model vectors are initialized as the means of their respective neighbors. Figure 2 shows schematically how a column of prototypes are inserted.

To decide if the map will grow by adding prototypes to the outside part of the grid, we compare the robust quantization error of the error unit with respect to all other units of the maps. The map will grow externally if the $rqe_e$ is greater than a factor of the maximum value of the robust quantization errors of the other units, i.e.,

$$rqe_e \geq \beta \max_{m_r \in M, m_r \neq m_e} rqe_r, \quad \text{with } \beta >> 1$$

To accomplish the outside growing process, the neighbor $m_d$ unit to the prototype $m_e$ with the greatest $rqe_d$ is detected. A row (or column) of units is added in the side where the units $m_e$ and $m_d$ are located in the grid, and their model vectors are initialized at half the distance of the neighbor row (or...
Fig. 3. The external growth process. A column of prototypes is added in the side where the units $m_e$ and $m_d$ are located. Column) but in the opposite direction. Figure 3 shows schematically how a column of prototypes are added.

5.1.6 Algorithm for the construction of the base model

In algorithm 1 we state the process for the creation of the base structure. Algorithm 1 calls algorithm 3 to train and adapt the architecture to the data. Algorithm 2 explains how the map $M$ is trained with the data $C(M)$ and how the map grow its grid either internally or externally. Algorithm 3 explains how the model $H_T$ adapts its architecture, decide when to grow hierarchically and call algorithm 2 to train and grow the grids. The second part of the process reuses algorithm 2 and 3 to train the structure at time $T$.

Algorithm 1 Creation of the base structure

1. Locate the root neuron $m_0$ as was explained in section 5.1.1 by using a robust estimator for the centroid of the data set $I_1$. Add the root neuron to the model $H_1 = \{m_0\}$.
2. Create a map $M$ of size $2 \times 2$ prototypes randomly around the root neuron $m_0$.
3. Add the map to the model $H_1 = H_1 \cup M$.
4. Execute algorithm 3 with $T = 1$

Algorithm 2 Training and growing the map $M$ The map $M$ is trained with the data $C(M)$ and grow if it needs a more detailed representation.

repeat
(a) Train the map with the robust learning algorithm explained in section 5.1.2.
(b) Compute the robust quantization error $rqe_r$ of all units $m_r$ that belongs to the map $M$ by using the equation (12).
(c) Compute the map’s mean quantization error (MRQE) with equation (14).
(d) Determine the error unit $m_e$ as $m_e = \arg \max \{rqe_r | m_r \in M\}$.
(e) if $\frac{1}{N} \sum_{m_r \in M} rqe_r \geq \gamma \text{MRQE}$ then the map grows internally as was explained in section 5.1.5, i.e., determine the most dissimilar proto-
type $\mathbf{m}_d$: insert a row or column between $\mathbf{m}_e$ and $\mathbf{m}_d$ and initialize their model vectors as the mean of their respective neighbors.

(f) if $rqe_e \geq \beta \max_{m_r \in M, m_r \neq m_e}rqe_r$ then the map grows externally as was explained in section 5.1.5, i.e., determine the neighbor unit $m_d$ with the highest robust quantization error. A row (or column) of units is added in the side where the units $\mathbf{m}_e$ and $\mathbf{m}_d$ are located and their model vectors are located at half the distance of the neighbor row (or column) but in the opposite direction until no more changes are done in the architecture.

Algorithm 3 Learning and adapting the model $\mathcal{H}_T$ to the data set $\mathcal{I}_T$ at stage $T$

1. Compute the robust quantization error of the root neuron $rqe_0$ using equation (12) with the data set $\mathcal{I}_T$
2. For each map $\mathcal{M}$ that belongs to the model $\mathcal{H}_T$ do
   (a) Consider the data set $\mathcal{C}(\mathcal{M})$.
   (b) Train the map $\mathcal{M}$ by executing algorithm 2 with the data set $\mathcal{C}(\mathcal{M})$.
   (c) repeat
      (i) For each leaf unit $\mathbf{m}_r$ of the map $\mathcal{M}$ do
         (A) Compute the robust quantization error $rqe_r$ with equation (12).
         (B) if $rqe_r \geq \tau r q e_0$ and $|\mathcal{C}(\mathbf{m}_r)| \geq N_{\text{min}}$ then create a map $\mathcal{M}$ of size $2 \times 2$ prototypes randomly around the neuron $\mathbf{m}_r$; train the map $\mathcal{M}$ by executing algorithm 2 with the data set $\mathcal{C}(\mathbf{m}_r)$. Add the map to the model $\mathcal{H}_T = \mathcal{H}_T \cup \mathcal{M}$.
      until the model $\mathcal{H}_T$ is stabilized, i.e., no more units were added.
   (d) Update the internal node $\mathbf{m}_u$ who is the parent of the map $\mathcal{M}$ by computing the mean value of the prototypes of the map by using equation (10), $\mathbf{m}_u = \bar{m}$.

5.2 Second Part: Adapting to changing environments

In this part of the process we need to incorporate additional capabilities to the model to be able to adapt to the topological drift of the environment. The learner will be able to track and adapt to the topological drift, detect novel concepts, recognize and treat recurring contexts and forget outdated knowledge. In what follows, we explain how we introduce these capabilities and at the end of this section we summarize the process of adapting to the changing environment.
5.2.1 Projecting the data to the model

We define the best matching unit \( m_{\eta(x)} \) of the model \( \mathcal{H}_T \) \((bmm)\) to the data \( x \) as the leaf unit:

\[
\mathbf{m}_{\eta(x)} = \arg \min \{ \| \mathbf{x} - \mathbf{m}_r \| | \mathbf{m}_r \in \mathcal{N}_L \} \tag{17}
\]

The sample \( x \) is projected from the high-dimensional input space \( \mathcal{X} \) onto the best matching unit \( m_{\eta(x)} \) of the model. All the samples that do not have a good topological representation with the current model \( \mathcal{H}_T \) are identified and are considered as novel data and we construct the set \( \mathcal{I}_T = \{ x_i \in \mathcal{I}_T | \| \mathbf{x} - \mathbf{m}_{\eta(x)} \| > \epsilon \} \) consisting of all the data whose distance to the model are bigger than some threshold \( \epsilon \).

5.2.2 Freezing and Updating the Internal Units.

The prototypes \( m_r \) that belong to the set \( \mathcal{N}_f \) of internal nodes of the model \( \mathcal{H}_T \) will not adapt when the map learning algorithm explained in section 5.1.2 is executed, i.e., these units are frozen. The prototypes will be updated as the mean values of the neurons belonging to their respective child maps as in equation (10).

5.2.3 Gradually forgetting outdated knowledge.

To achieve better generalization and to keep the model stable, a forgetting strategy is needed. The strategy consists in gradually forgetting by shrinking the lattices towards their respective centroids and, if the grids are sufficiently small, they are contracted by deleting rows or columns of prototypes.

Compute the centroid neurons \( \bar{m} \) with equation (10). The map \( \mathcal{M} \) will forget the old data by applying once the forgetting rule:

\[
\mathbf{m}_r(T) = \mathbf{m}_r(T - 1) + \lambda [\mathbf{m}_r(T - 1) - \bar{m}] \quad \mathbf{m}_r \in \mathcal{M} \tag{18}
\]

where \( \lambda \) is the forgetting rate. \( \mathbf{m}_r(T - 1) \) and \( \mathbf{m}_r(T) \) are the values of the prototypes at the end of the stage \( T - 1 \) and the beginning of stage \( T \) respectively. If the units of the map are too close then the map is contracted. Consider the rectangular grid; the prototype is indexed by \( \mathbf{r}(\mathbf{m}_r) = (i, j) \), where \( i \) and \( j \) are the row and column of the unit in the rectangular array (see section 3). We search for two rows or columns whose neurons are very close. To accomplish this the value \( \nu \) is computed as
Fig. 4. The contraction process. (left) The map before it is contracted. (middle) A column of prototypes is inserted in between the columns e and d. (right) The columns e and d of prototypes are eliminated from the grid.

Contraction Process:

\[
\nu = \arg \min_{e=1..N_r-1, d=1..N_c-1} \left( \frac{1}{N_c} \sum_{j=1}^{N_c} \| \mathbf{m}_{e,j} - \mathbf{m}_{e+1,j} \|, \frac{1}{N_r} \sum_{i=1}^{N_r} \| \mathbf{m}_{i,d} - \mathbf{m}_{i,d+1} \| \right)
\]

where \( N_r \) and \( N_c \) are the number of rows and columns of the map lattice respectively. If the criterion \( \nu < \varepsilon \) for the row (or column) is met and if the prototypes of one of the row (column) \( \nu \) or \( \nu + 1 \) are all leafs then the map is contracted. In the contraction process we insert a row (or a column) of units between \( \nu \) and \( \nu + 1 \) and their model vectors are initialized as the mean of their respective neighbors, and then the rows (or columns) \( \nu \) and \( \nu + 1 \) of prototypes are both eliminated. In figure 4 the column contraction process is shown. The map is contracted iteratively until no other row or column satisfies the criterion. If the map reaches its minimum size of \( 2 \times 2 \) and none of its prototypes have a child map and the distance of the prototypes of both, the row and column, are less than \( \varepsilon \), then the map is eliminated from the model.

5.3 Model capabilities to deal with concept and topological drift

The model with the capability of adapting its architecture based on the data as was explained in section 5.1, gives to the system abilities to deal with novel concepts, topological drift and recurrent context as we will explain below.

When a novel concept appears in the data stream during the stage \( T \) the model \( \mathcal{H}_{T-1} \) will detect the concept during its training at some of its level depending the degree of novelty. The novel concept will increase the robust quantization error of some prototypes of the map \( \mathcal{M} \) at certain level \( u \) of its structure. If the novel concept location is detected in between previous learned concepts, the map will grow internally its grid by introducing prototypes towards this novel concept. If the novel concept is located outside the previous learned concepts, the map will grow externally its grid by introducing prototypes towards this novel concept. Finally, if the concept is strengthening a previous
learned concept, then the prototype, that it is modeling the concept, will grow hierarchically to obtain a more detailed description.

The recurring context occurs when concepts are present in certain stages, and vanish and reappear in future stages. To deal with this phenomenon, note that the model only adapts its leaf prototypes while the internal nodes remain frozen. The internal node will not move from its current location until its child map are contracted and eliminated. For this reason, if the concepts were strong enough to create at least a two level structure, then the model will slowly forget and contract when the concept no longer exists. If the concept vanishes forever or for a very long period of time, the structure will begin to disappear until only one prototype will remain and will move away toward other concept and forgetting the previous learned concept.

Note that if the internal prototypes are not fixed in their location, when the concept disappears, the model will catastrophically forget the concept.

5.4 Algorithm for Adapting to the changing environment

Algorithm 4 Adapting to the changing environment

1. Freeze all internal nodes during the maps training process.
2. Forget the outdated data by shrinking and contracting the grids as was explained in section 5.2.3. Update the model \( H_T \) by extracting all the prototypes that were removed.
3. Execute algorithm 3 with \( T > 1 \)

5.5 Evaluation of the adaptation Quality

To evaluate the quality of adaptation to the data a common measure to compare the algorithms is needed. The following metric based on the mean square quantization error is proposed:

\[
MSQE = \frac{1}{|\mathcal{I}|} \sum_{x_i \in \mathcal{I}} \| x_i - \mathbf{m}_{\eta(x)} \|^2
\]

where \( |\mathcal{I}| \) is the number of data that belongs to the input set \( \mathcal{I} \), and \( \mathbf{m}_{\eta(x)} \) is the best matching unit of the model \( H_T \) to the data \( x \) defined in equation (17)
6 Simulation Results

In this section we empirically show the capabilities of our *RoFlex-HSOM* model proposal compared to the HSOM model and some of its extensions, as they will be described below. In subsection 6.1 and 6.2 we show the performance results for computer generated data and *El Niño* real data set respectively.

For the experiments we evaluated four models. The first model is the basic Hierarchical Self Organizing Maps (*HSOM*), and the other three models were obtained by extending this model. The second model is the Robust HSOM (*Ro_HSOM*) where we added robustness to the basic model. The third model is the Flexible HSOM (*Flex_HSOM*) where we added flexibility to the basic model. And finally we have the Robust and Flexible HSOM (*RoFlex_HSOM*) where we added both, robustness and flexibility.

The models have the following characteristics. Non flexible models, i.e., the *HSOM* and *Ro_HSOM*, have static structures meaning that, at each stage, the previous model is eliminated and constructed again. In this work the static models were trained, for each stage, with the first part of the training algorithm explained in section 5.1. While flexible models, i.e., the *Flex_HSOM* and *RoFlex_HSOM*, have adaptive structure meaning that during the first stage the models were trained with the first part of the training algorithm explained in section 5.1 and, for the following stages, with the second part of the training algorithm explained in section 5.2.

For the non robust models, i.e., the *HSOM* and *Flex_HSOM*, the sample mean estimator $\mathcal{Z} = \frac{1}{n} \sum_{i=1}^{n} z_i$ was used to locate the root neuron; the classical standard deviation $s^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - X)^2$ was used to estimate scales instead of equation (7); and the Least Square score function $\psi^{LS}(z) = z$ was used to train the maps in equation (11). To compute the robust quantization error of equation (12) and the map’s robust quantization error of equation (14) the Least Square function $\rho^{LS}(z) = z'z$ was used.

While for the robust models, i.e., the *Ro_HSOM* and *RoFlex_HSOM*, the sample median estimator $\text{median}_{i=1..n}(z_i)$ was used to locate the root neuron; the sMAD scale estimator given by equation (7) was used to estimate scales; and the Huber score function $\psi^{H}(z)$ of equation (9) was used to train the maps in equation (11). To compute the robust quantization error of equation (12) and the map’s robust quantization error of equation (14) the Huber robust function $\rho^{H}(z)$ given by equation (9) was used.

The data of both, synthetics and real data sets, were separated in training and test sets for each stage. All the models obtained after training at stage $T$ were evaluated with the mean square quantization error (MSQE) (see equa-
tion (19)) with the current test set $\mathcal{T}_{T}^{test}$, the previous test set $\mathcal{T}_{T-1}^{test}$ and the following test set $\mathcal{T}_{T+1}^{test}$ corresponding to the stages $T$, $T-1$ and $T+1$ respectively. The performance of the models with the current test set $\mathcal{T}_{T}^{test}$ will give us information of how well the model did adapt to the current stage. The performance of the models with the previous test set $\mathcal{T}_{T-1}^{test}$ will give us information of how the model forgot previous concepts. The performance of the models with the following test set $\mathcal{T}_{T+1}^{test}$ will give us information of how the model will behave with future concepts.

All the results reported were obtained for each model as the mean value of the metrics computed for 20 runs evaluated at each stage and considering the same sets of data.

### 6.1 Experiment #1: Computer generated data

The synthetic experiment consists in six two dimensional clusters created along the four stages. Let $I_2$ be the identity matrix of size $2 \times 2$. The characteristics for each cluster are:

- Cluster 1: Consists in three little Gaussian clusters created equidistant to the centroid location $[0.9 \ 0.01]'$, each with the covariance matrix of $\Sigma = 0.008^2 I_2$. The cluster is composed of 500 samples for each stage. Starting from the second stage this cluster is vanished and reappears at stage 4 with 50 samples.

- Cluster 2: Consists in three little Gaussian clusters created equidistant to the centroid location $[0.8 \ 0.5]'$, each with the covariance matrix of $\Sigma = 0.008^2 I_2$. The cluster is composed of 500 samples in the first and second stage, 333 samples in the third stage and 166 in the fourth stage.

- Cluster 3: Consists in three little Gaussian clusters created equidistant to the centroid location $[0.6 \ 0.8]'$, each with the covariance matrix of $\Sigma = 0.008^2 I_2$. The cluster begins with no sample at stage 1 and appears at stage 2 with 500 samples. Starting from stage 3 the three little Gaussian clusters are converted in two little Gaussian clusters equidistant to the same centroid, but each with the covariance matrix of $\Sigma = 0.0125^2 I_2$.

- Cluster 4: Consists in three little Gaussian clusters created equidistant to the centroid location $[0.1 \ 0.8]'$, each with the covariance matrix of $\Sigma = 0.008^2 I_2$. The cluster moves, and the centroid is located in $[0.125 \ 0.775]'$ at stage 2, in $[0.15 \ 0.75]'$ at stage 3 and in $[0.1750 \ 0.7250]'$ at stage 4. The cluster is composed of 500 samples for each stage. Starting from stage 2 the three little Gaussian clusters are converted in one big Gaussian clusters with mean equal to the centroid and the covariance matrix is $\Sigma = 0.05^2 I_2$.

- Cluster 5: Consists in one big Gaussian clusters with mean $[0.2 \ 0.1]'$, and covariance matrix of $\Sigma = 0.05^2 I_2$. The cluster is composed of 500 samples.
for each stage. The covariance was increased in the following stages, the covariance matrix is $\Sigma = 0.054^2 I_2$ at stage 2, $\Sigma = 0.0583^2 I_2$ at stage 3 and $\Sigma = 0.063^2 I_2$ at stage 4. Furthermore, starting from stage 3 the one big Gaussian clusters are converted in three little Gaussian clusters equidistant to the previous mean, but each with the covariance matrix of $\Sigma = 0.008^2 I_2$.

- Cluster 6: Consists in three little Gaussian clusters created equidistant to the centroid location $[0.5 0.5]'$, each with the covariance matrix of $\Sigma = 0.008^2 I_2$. The cluster begins with no sample at stage 1 and appears at stage 3 with 1500 samples.

The observational process was obtained by including additive outliers: $Z_k = X_k + V_k U_k$, where $X_k$ are the clusters generated as was previously explained, $V_k$ is zero-one process with $P(V_k \neq 0) = \alpha$, $0 < \alpha \ll 1$, and $U_k$ has distribution $\mathcal{N}(0, \Sigma_{U_k})$ with covariance matrix $|\Sigma_{U_k}| \gg |\Sigma_k|$. The generating process was affected with $\alpha = 0\%$, 1\% and 10\% of outliers and $\Sigma_{U_k} = 10^2 * \Sigma_k$, $k = 1..6$. The generated data sets were divided, for each stage, in training and test sets both of equal size.

For the experiments the growing parameters must be adjusted, i.e., the $\tau$ and $N_{min}$ of equation (13) that control the hierarchical growing process; the $\gamma$ of equation (15) that controls the internal growing process; and the $\beta$ of equation (16) that controls the external growing process. For the robust models we considered the following configuration $\tau = 0.01$, $N_{min} = 15$, $\gamma = 0.06$ and $\beta = 10$; while for the non robust models we used the following configuration $\tau = 0.2$, $N_{min} = 15$, $\gamma = 0.2$ and $\beta = 100$. The latter configuration makes the model simpler than the former, but, unfortunately, with lower values the non robust models become instable in some simulations by growing their maps bigger as we really wanted and consuming a plenty of time.

The forgetting parameters for the flexible models were set at $\lambda = 0.5$ for the forgetting rule of equation (18) and $\varepsilon = 0.01$ for the contraction. At the beginning of each stage the maps will shrink to the half of their current size.

Figure 5 shows the adaptation of the models to the data sets for each stage and considering 1\% of outliers. Each column of graphs of figure 5, from left to right, correspond to the stages $T = 1, 2, 3$ and 4. Each row of graphs of figure 5, from top to bottom, correspond to the models HSOM, Ro_HSOM, Flex_HSOM and RoFlex_HSOM respectively. Note that the HSOM model created a very simple structure and, in most of the cases, did not create a particular map for the clusters that it was modeling; in the second stage, it can be appreciated how the model forgot the cluster 1; and in the last stage the model was not able to model the recurrent cluster 1. The Ro_HSOM model of the second row was able to create maps for each of the cluster; it also forgot the cluster 1 at the second stage; and in the last stage create a map at the third level of its structure to model the recurrent cluster 1; note that when
the clusters changed their topologies, the maps were also changed (see, e.g., the map modeling the cluster 5). The Flex_HSOM model of the third row was able to create maps for each of the cluster; it did not forget the cluster 1 at the second stage; and in the last stage it still remembered the recurrent cluster 1; note that the maps were badly affected by the presence of outliers by locating it far from the clusters of data (see, e.g., the map modeling the cluster 5 at stages 3 and 4) or by overextending the maps of the clusters 3, 4 and 5. Finally, the RoFlex_HSOM model of the fourth row was able to combine the best capabilities of both the Ro_HSOM and Flex_HSOM models; It created maps for each of the cluster; it did not forget the cluster 1 at the second stage although it slowly forgot the cluster; it created a second level map to model the recurrent cluster 1 at stage 4; note that the maps were barely affected by the presence of outliers by over extending the map of the clusters 4. As can be seen in figure 5, the robust models obtained a more detailed description of the data sets and were not badly affected by the outliers. The flexible models did not catastrophically forget previously learned patterns.
Fig. 6. **Number of Neurons of the models:** The graphs show the mean number of neurons obtained with the four models in all four stages and with 0% (left graph), 1% (middle graph) and 10% (right graph) of outliers.

Fig. 7. **The MSQE performance evaluation for the models with the 0% of outliers data set:** The graphs show the mean square quantization error (MSQE) computed for the four models in all four stages $T = 1, \ldots, 4$ and considering the data set with 0% of outliers. Each graph correspond to the evaluation of the model with different test sets: (left graph) the $I_{Test}^T$ test set of the current stage, (middle graph) the $I_{Test}^{T-1}$ test set of the previous stage, and, (right graph) the $I_{Test}^{T+1}$ test set of the next stage.

Figure 6 shows the number of neurons of each model obtained at each training stage. Each graph, from left to right, correspond to the models obtained for the data sets with 0%, 1% and 10% of outliers. Note that the robust models have approximately the same size. The Flex_HSOM model created a bigger structure than the others for the 0% and 1% of outliers cases. The non robust models created a simple structure for higher levels of contamination because they almost did not distinguished the several clusters and they created a big map with low hierarchy.

Figures 7, 8 and 9 show the evaluation of the MSQE, given by equation (19), for the models performance at the test data sets with 0% (figure 7), 1% (figure 8), and, 10% (figure 9) of outliers after each training stage. The graphs of the figures correspond to the MSQE evaluated with the $I_{Test}^T$ (left), $I_{Test}^{T-1}$ (middle) and $I_{Test}^{T+1}$ (right) test data sets. As can be appreciated in the middle graph, the non flexible models have a big amount of error at stage 2 because they forgot the cluster 1 of stage 1. The robust models show better performance in most of the cases even in the non contaminated case; furthermore, they
Fig. 8. The MSQE performance evaluation for the models with the 1\% of outliers data set: The graphs show the mean square quantization error (MSQE) computed for the four models in all four stages $T = 1, \ldots, 4$ and considering the data set with 1\% of outliers. Each graph correspond to the evaluation of the model with different test sets: (left graph) the $T^{Test}_T$ test set of the current stage, (middle graph) the $T^{Test}_{T-1}$ test set of the previous stage, and, (right graph) the $T^{Test}_{T+1}$ test set of the next stage.

Fig. 9. The MSQE performance evaluation for the models with the 10\% of outliers data set: The graphs show the mean square quantization error (MSQE) computed for the four models in all four stages $T = 1, \ldots, 4$ and considering the data set with 10\% of outliers. Each graph correspond to the evaluation of the model with different test sets: (left graph) the $T^{Test}_T$ test set of the current stage, (middle graph) the $T^{Test}_{T-1}$ test set of the previous stage, and, (right graph) the $T^{Test}_{T+1}$ test set of the next stage.

show better behavior in predicting future data. Note that the RoFlex_HSOM model combined both, the good performance of the robust models and the not catastrophically forgetting capability of the flexible models.

If figure 10 the time performances for the training process of the models are shown. The evaluations were made for the data sets with 0\% (left), 1\% (middle) and 10\% (right) of outliers. As can be noted, the Flex_HSOM model shows the worst training time for the 0\% and 1\% of outliers data sets. The robust models needed almost the same time for all the stages and data sets, i.e, they were not much affected by the contamination. The training times of the HSOM show better performance than the robust methods in some cases, probably due to the simpler structure, and worse performance in other cases. Note that the combination of robustness and flexibility in the RoFlex_HSOM
Model reduces the training time cost, and the training algorithm is more stable than the HSOM model.

6.2 Experiment #2: “El Niño” real data

In the real data set experiment we tested the algorithm with the El Niño Data. The data can be obtained from the following site


The El Niño Data are expected to aid in the understanding and prediction of El Niño Southern Oscillation (ENSO) cycles and was collected by the Pacific Marine Environmental Laboratory National Oceanic and Atmospheric Administration. The data set contains oceanographic and surface meteorological readings taken from a several buoys positioned throughout the equatorial Pacific.

The data consist of the following variables: date, latitude, longitude, zonal winds (west < 0, east > 0), meridional winds (south < 0, north > 0), relative humidity, air temperature, sea surface temperature and subsurface temperatures down to a depth of 500 meters. Data taken from the buoys are as early as 1980 for some locations.

The data set was modified by discarding those data with missing values. We kept eight stages corresponding to the first eight years of data collection, from 1980 until 1988. We used 6311 instances of 4 dimensions (meridional winds, relative humidity, sea surface temperature and subsurface temperatures) from whom 4415 were used for training and 1896 for testing. To execute the simulations and to compute the metrics, all the dimensions of the training data set of the first stage were scaled to the unit interval, and with the same scale, the rest of training and test data sets were scaled (Notice that with this scaling...
Fig. 11. “El Niño” real data experiment: The graphs show the mean square quantization error (MSQE) computed for the four models obtained at all eight stages. Each graph correspond to the evaluation of the model with different test sets: (left graph) the $I_{T}^{Test}$ test set of the current stage, (middle graph) the $I_{T-1}^{Test}$ test set of the previous stage, and, (right graph) the $I_{T+1}^{Test}$ test set of the next stage the training and test data will not necessarily fall in the unit interval). We divided the data set according to the years into 8 training data sets composed of 110, 226, 293, 160, 411, 337, 837 and 2041 samples respectively for each year stage; and into 8 test data sets composed of 48, 97, 126, 69, 177, 145, 359 and 875 samples respectively for each year stage.

Figure 11 shows the evaluation of the MSQE, given by equation (19), for the models performance at the test data sets after each training stage. The graphs of the figure correspond to the MSQE evaluated with the $I_{T}^{Test}$ (left), $I_{T-1}^{Test}$ (middle) and $I_{T+1}^{Test}$ (right) test data sets. As can be easily appreciated the robust models outperforms the non robust models in all stages except for the first one, and the former obtained approximately half of the MSQE of the non robust models. Furthermore the robust models have better performance in predicting the past and the future. We can conclude that the behavior of this data set is non Gaussian or is highly contaminated, for this reason is affecting very badly the non robust models. Furthermore, by observing the fourth stage of the middle graph of figure 11 we can conclude that the non robust models obtained at stage 4 were not able to remember and predict the data set of the third stage; and by observing the first, second and fourth stages of the right graph of figure 11 we can conclude that the non robust models were not able to predict the data sets of the second, third and fifth stages with the models obtained at stages 1, 2 and 4 respectively. This last result could signify that the “El Niño” data set either has a topological drift or the contamination is affecting badly the non robust models. We cannot conclude whether the Ro_HSOM or RoFlex_HSOM had a better performance. Further studies will be needed to understand why the non robust models outperforms the robust models at the first stage while for the other stages the opposite occurs, probably this phenomenon is due to some unknown change in the behavior of the “El Niño” data set.
7 Concluding Remarks

In this paper we have postulated a method of how to incorporate Robustness and Flexibility to the Hierarchical Self Organizing Maps model. The extended model was called (RoFlex-HSOM). We had empirically shown that the RoFlex-HSOM had the plasticity to find the structure that suits the best to the data, gradually forgets (but no catastrophically) previous learned patterns, it is robust to the presence of outliers and preserves the topology to represent the hierarchical relation of the data under environments that are time dependent and non stationary. The theory and the method presented in this work can be helpful to incorporate robustness and/or flexibility to other classical models and with this we can extend their capabilities to more complex environments.

In the experimental study we made a comparative analysis of four models with synthetic and real data sets. The data of the synthetic experiment were affected with different degrees of noise. The models used were the HSOM model and three extensions of itself by adding the capabilities of robustness (Ro_HSOM), the flexibility (Flex_HSOM) and both, robustness and flexibility (RoFlex_HSOM). With the synthetic data sets we were able to show that the robust models (Ro_HSOM and RoFlex_HSOM) obtained better performance under contaminated environments and their behavior were more stable during the training process. In addition we showed that the flexible models (Flex_HSOM and RoFlex_HSOM) were able to slowly (and not catastrophically) forget outdated knowledge. Only the RoFlex_HSOM model was successfully in combining the capabilities of robustness against noise and the flexibility of effectively tracking the topological drift.

For the real case, we investigated the El Niño data. The performance of the robust models (Ro_HSOM and RoFlex_HSOM) outperforms the non robust models (HSOM and Flex_HSOM). We suspect that the data either is highly contaminated or has a behavior radically different to the Gaussian distribution, furthermore the data set presented unknown drifts that needs further analysis.

In further studies, the training algorithm should be fine tuned to improve the speed of learning and the quality of adaptation. The RoFlex_HSOM model could be further applied in several other real applications as e.g. self organization of a massive document collection, weather prediction and spam e-mail topology modeling to name a few complex real problems.
Acknowledgments

This work was supported by the following research grants: Fondecyt 1061201, 1040365 and 7050205, DGIP-UTFSM, and, BMBF-CHL 03-Z13 from the German Ministry of Education and Research. The authors like to thank the organizers of IWANN 2005 for inviting us to submit this extended version of our paper [17] to Neurocomputing. We would also thankfully acknowledge the detailed observations and suggestions of the anonymous reviewers.

References


