Noname manuscript No.
(will be inserted by the editor)

Nonvectorial data clasification with Self-organizing maps using data interpolation v.29

J. Carpio · J.J. Merelo · V. Rivas

Received: date / Accepted: date

Abstract Self-organizing maps (SOM) are widely used to classify vectorial data. However, its methodology is not restricted to metric vector spaces. In this paper we introduces a self-organized map framework to classify any kind of data with only two restrictions: namely, the existence of a distance measure for input elements, and the possibility to interpolate a new element between other two with a certain distance. We formilizes the requiriments to define a distance measure and the interpolation functions and, as a practical example, following this philosophy we implemented a SOM for symbol strings.

Keywords Self-organizing maps; Framework; Clustering; Visualization; Data mining; Nonvectorial data

1 Introduction

[4] The key ingredient of pupular machine learning methods such as suppor vector machines (SVM), learning vector quantization (LVQ), self-organizing maps (SOM), or

J. Carpio
Information Technology Department
University of Huelva
Tel.: +34-959-217658
Fax: +34-959-217658
E-mail: jose.carpio@dti.uhu.es

J. Merelo
Computers Architecture and Technology Department
University of Granada
Tel.: +34-958-243162
Fax: +34-958-243162
E-mail: jmerelo@geneura.ugr.es

V. Rivas
Computer Science Department
University of Jaen
Tel.: +34-953-243162
Fax: +34-953-243162
E-mail: vrivas@ujaen.es

K-nearest neighbor classification is a similarity measure of the data in the input space.... The choice of the metric for these methods is directly connected to the representation of data and it crucially influences the efficiency, accuracy, and generalization ability of the results. Depending on the respective application, different aims need to be fulfilled: the similarity measure should possess the flexibility to capture complexity inherent in the learning task; a powerfull metric allows for sparse models and natural representation of date since complex decision borders

[3] A general framework for unsupervised processing of structured data: Unsupervised learning constitutes an alternative important paradigm for neural networks which has been successfully applied to data mining and visualization (see [10]). Up to now, only few approaches enlarge SOM to sequences: the temporal Kohonen map (TKM)[1] and the recursive SOM (RSOM) [23] and variations thereof. We are aware of only one approach which processess tree structured data and thus alse sequences in an unsupervised way: the SOM for structured data (SOMSD) [22] .These models are trained by Hebbian learning. They have been applied to monitor standard time series or image data, so far. Since a clear measure of the success of unsupervised processing of structured data and a theoretical formulation of the above dynamics and training method are lacking...

SOMSD: Each neuron in the map is equiped with weights (w, r_1, r_2). Hereby, $w \in \mathcal{W}$ is contained in the set of possible labels, r_1 and r_2 are contained in $\mathbb{R}^d$, d denoting the dimensionality of the lattice. They correspond to possible indices of neurons. Given a tree $a(t_1, t_2)$ as input, its distance from neuron n with weights (w, r_1, r_2) is recursively computed by $\alpha \parallel a - w \parallel + \beta \parallel \iota(t_1) - r_1 \parallel + \beta \parallel \iota(t_2) - r_2 \parallel$. Hereby, $\iota(t_1)$ and $\iota(t_2)$ denote the indices of the neurons with smallest distance from t_1 and t_2, respectively.

RSOM: RSOM has originally been proposed for sequences in [22] . However, it can be generalized to tree structures. The neurons are equipped with weights (w, r) where $w \in \mathcal{W}$. $r \in \mathbb{R}^N$ corresponds to the vector of activations of all neurons at the previous time step.

TKM: The temporal Kohonen map consist of a lattecie of neurons where each neuron constitutes a leaky integrator [1]. The approach is defined for sequences over the set $\mathcal{W}$, only. Each neuron is equiped with a weight $w \in \mathcal{W}$. The distance of the sequence $[a_0, ..., a_\iota]$ from neuron n weighted with w is computed by ...

[17]

$$x(n+1) = X(n)^T W \tag{1}$$

$$e(n) = x(n+1) - X(n)^T W \tag{2}$$

$$W' = W + \gamma e(n) X(n) \tag{3}$$

The self-organizing map of autoregressive (AR) models is a 2D lattice of units, indexed in the following by i, with weight vectors W_i each signifying an AR process. According to Kohonen [9], the general procedure in the lattice is:

1. For each input sample, find the best matching unit
2. Modify that unit, and the units in its topological neigborhood, to further improve the match to the present input.

At a given step of the self-organizing algorithm, the inputs to the network, shared by all the units, are the samples in vectors $X(n)$. Each unit tries to predict $x(n+1)$ from Eq. (1) by using its own weigth vector. Since the units are tuned to different AR models, some of the errors (2) will be smallest than the others. The apparent winner (the best matching unit) for each input vector $X(n)$ is the unit with the smallest estimator error.

Once the best-matching unit has been found, that unit and its neighbors are updated according to the *alms* algorithm (3). To define the neigborhood in the 2D lattice, the distance r of two units in the lattice must be defined. Neigborhoods determined by both the Euclidean and the Chebyshev metrics have been tried and they give similar results. When the neigboring units are updated, the scalar adaptation γ of Eq. (3) is multiplied by a factor depending on r to yield

$$g(r) = \gamma(1 - \frac{r}{NE+1}), \tag{4}$$

where r is the distance from the best matching unit to the unit to be updated and parameter NE determines the size of the neigborhood. The size is slowly decreasing in time, e.g., if t denotes the iteration step, then

[11] In the vast majority of SOM applications, the imput data constitute high-dimensional real *feature vectors* $x \in \Re^n$, and the model vectors $m_i \in \Re^n$ are them approximations of the x in somewhat similar sense as the codebook vectors in classical vector quantization are. However, the models need not neccessarily be replica of the input vectors: the may be, e.g., parametric representations of operators that generate sequences of data [17] . On the other hand, there exist means to approximate also non vectorial data, e.g., sets of symbol string can be approximated by “average strings” [6]. In the SOMs that forms similarity graphs of *documents*, the models can still be taken as real vectors that describe collenctions of words in the documents. The models can simply be weighted histograms of the words, but usually some dimensionality reduction of the histogram is carried out, as we shall see next.

[13] The self-organizing maps (SOMs) are usually defined in metric vector spaces. A different idea is organization of *symbol strings* or other nonvectorial representation on a SOM array, whereby the relative locations of the images of the strings on a SOM array, whereby the relative locations of the images of the strings on the SOM ought to reflect, e.g., some distance measure, such as the *Levenshtein distance* (LD) of *feature distance* /FD) between strings (for textbook accounts, cf.). If one tries to apply the SOM algorithm to such entities, the difficulty inmmediatly encountered is that *incremental learning laws cannot be expressed for symbols strings*, which are discrete entities. Neither can a string be regarded as a vector. On of the authors has shown that the SOM philosophy is nonetheless amenable to the construction of ordered similarity diagrams for string variables, if the following ideas are applied:

1. The *batch map* principle [10] is used to define learning as a succession of *conditional averages over subsets of selected strings.*
2. The áveragesóver the strings are computed as generalized *means or mediand* (reference to Kohonen median string) over the string.

SOM for symbol strings has been analized in [13] using batch map principle [10]. The batch map principle is used to define learning as a succession of condicional averages over subsets of selected strings and the averages over strings are computed as generalized means or medians over the strings.

[13] -¿ Initialization of SOM for strings

It is possible to initialize a usual vector-spaces SOM by radom vectorial values. We have been able to obtain organized SOMs for string variables, too, starting with random reference strings. However, it is a great advantage if the initial values are already ordered, even roughly, along the SOM array.

[13] -¿ The batch map for strings

The conventional batch map computing steps [10] of the SOM are applicable to string variables almost as such:

1. For the initial reference strings, take, for instance, samples that are ordered two-dimensionally in the Sammon projection.
2. For each map unit i, collect a list of those sample strings to whoms the reference string of unit i is the nearest reference string.
3. For each map unit i, take for the new reference string the mean or median over the union of the lists that belong to the topological neighborhood set N_i of unit i.
4. Repeat from 2 a sufficient number of times, until the reference strings are no longer changed in further iteractions.

[12] -¿ However, the SOM principle is not restricted to metric vector spaces. It has been pointed out [6] that any set of items, for wich a similarity or distance measure between its elements is defineable, can be mapped on the SOM grid in an ordely fashion. This is made possible by the following principle, which combines the concept of the *generalized median* of a set [**?**,**?**] with the batch computation of the SOM. Assume a fundamental set S of any kind of items $x(i)$ and let $d[x(i), x(j)]$ be some distance measure between $x(i)$ and $x(j)$ and $x(j) \in S$. The generalized median m over S is defined as the item that minimizes the objetive function

$$D = \sum_{x(i) \in S} d[x(i), m]$$

In this work, m is restricted to being an element of S. Notice that if the input samples had been real scalars and the distance measure were the absolute value of their difference, it is easy to show that the generalized median coincides with the aritmetic median. On the other hand, if the input samples where real vectors, if the distance measure were Euclidean, and if the with the smallest sum of the squares of distances from the other items where sought, the generalized median would coincide with the aritmetic mean of the $x(i) \in S$.

PicSOM [16] based on tree structured self-organizing maps (TS-SOMs). Given a set of reference images, PicSOM is able to retrieve another set of images which are similar to the given ones. Each TS-SOM is formed with a different image feature representation like color, texture or shape. The queries are iteratively refined as the system exposes images to the user. After the training phase, each unit of the TS-SOMs contains a model vector which may be regarded as the average of all featured vectors maped to a particular unit. A tree-structured, hierachical representation of all the images in the database is formed. In an ideal situation, there should be one-to-one correspondance between th images and TS-SOM units at the bottom level of each map.

The five separate feature vectos obtained from each image in the database habe to be indexed to facilitate similarity-based queries. For the indexing, similarity-preserving clustering using SOMs (Kohonen book reference) is used. Due to the high dimensionality of the feature vectors and the large size of the image database, complexity is a major issue. The TS-SOM [14, **?**] is a tree-structured vector quantization algorithm that uses SOMs at each of its hierarchical levels. PicSOM may use one or several types of statistical features for image querying. Separate feature vectors can be formed for describing the color, texture, shape, and structure of the images. A separate TS-SOM is then constructed for each feature. These maps are used in parallel to find from the databases those images which are most similar to the images selected by the user. The feature selection is not restricted in any way, and new features can be included as long as the feature vector are of fixed dimensionality and the Euclidean metric can be used to measure distances between them.

The continuous interpolating Self-organizig map (CI-SOM) [**?**] is based on I-SOM. Each neuron stores one corresponding codebook position in the input space and in the output space. The interpolating function passes exactly though these positions (support points). The dimensions of the interpolating function are equal to the dimensions of the map.

[19] -¿ SOM makes a nonlinear projection from a high-dimensional data space (one dimension per variable) on a regular, low-dimensional data (usually 2D) grid of neurons, called units. SOM is further processed using ULtsch method (cite to Ultsch, S.: Kohonen's Self-organizing maps for exploratory data analysis), the Unified distance matrix (U-matrix) wich uses SOM's codevectors (vectors of variables of the problem) as data source and generates a matrix where each component is a distance measure between two adjacent neurons.

In order to apply SOM we need to transform this data set into another one where a distance between each pair of samples can be calculated. In order to do this, we consider a new data set wich contains for each sample a vector with the Hamming distance to each one of the other samples.

[2] This paper proposes a symbolic measure that is used to implement a string self-organizing map based on SOM algorithm. Such measure between two strings is a new string. Computation over strings is performed using a priority relationship among symbols. A complementary operation is defined in order to apply such measure to DNA strands. Finally, an algorithm is proposed in order to be able to implement a string self organizing map. It is important to note that new symbols $a, b \in V \mid \mathcal{O}(a) - \mathcal{O}(b) \mid > 1$, and ther is no symbol c such tah $\mathcal{O}(a) < \mathcal{O}(c) < \mathcal{O}(b)$. That is,

$$(O)^{-1}(k) = \begin{cases} x \in V if \mathcal{O}(x) = k \\ S_k i.o.c. \end{cases}$$

$(O)^{-1}(k) = \{x \in V if \mathcal{O}(x) = k, ski.o.c.with K \in N$

Self-Organizing Maps (SOM) (Tuevo Kohonen [8,10]) are widely used for unsupervised learning, clustering, classification, and visualization of any kind of data. More than 5000 publications [**?**] can be found in the literature in the last years. Many of these works propose significant approaches to deal with vectorial data. The use of linear vector spaces in SOM decrease the complexity thanks to the Euclidean distance and to the arithmetical operations usually implemented as basic instructions in computer processors. These special characteristics of numerical vectors makes easy to implement fast SOM algorithms. However, areas with increasing relevance like image processing, bioinformatics, or speech recognition need to deal with nonvectorial data.

There are two approaches to work in SOM with nonvectorial data. One approach consist on transforming input data to vectorial one. The other approach, is based on SOM algorithm modification to enable it to deal directly with a certain data type. Codify input data as numerical vectors, usually require high efforts for analysis and implementation. Sometimes, this transformation is not possible or reduce certain characteristics of original data. In a recent published work [**?**] we made clustering and visualization of HIV Quasiespecies using SOM. In this paper we have codified original ARN sequences to vectors using *Hamming* distance finding some inconveniences in this process. That experience gided as to work with the second approach.

A good methodology to deal with nonvectorial data directly, is described by T. Kohonen and P. Somervuo in [12]. This method modify original SOM algorithm calculating a *Median Set* as model. Studies over the median for symbol strings in [**?**] demonstrates the NP-Complete complexity. That inconvenience joint with the loss of graduality in learning process motivate us to search for a new methodology that will be nearer to the original SOM simplicity and elegance. In this way, we developed a method that start with SOM for vectorial data algorithm making minimum changes. We have been analized the algorithm in order to identify the steps that modify the models. In original SOM for vectors, the models has same type than input data. Once, we located these crucial steps, we modify it to make it more general. The idea is to provide the needed elements that permit us to work with any type of data. The *model updating process* is one of these steps that "touch" directly the input data. Other operation like calculating neighborhood depends on data coordinates in the grid and not on data itself.

In this paper we propose a framework for SOM that permit us to deal with any kind of data maintaining original vectorial SOM concepts and we formalizes the requirements to implement new modules for any input data type. As example we have implemented the AI::NeuralNet::SOMString Perl module. We present in Sec. **??** some results for strings clustering experiments.

The paper is organized as follows: Sec. **??** contains a brief survey of works related to the application of SOM to nonvectorial data. Sec. 4 introduces our new algorithm based on the use of discrete entities interpolation. Sec. 6 is devoted to describe two experemental examples in order to study how our proposal works with symbol strings and the perfomance of our method method with large data sets.

2 State of the art

Tengo que hablar de trabajos relacionados. Cosas que tengo que contar: - The batch map for strings: "Sammon" projection?

The Self-Organizing Map (SOM) was introduced by Teuvo Kohonen in 1982 (see [8,10] for details).

Learning Vector Quantization (LVQ) supervised learning. No neighborhood are defined around the winner. LVQ can be used to fine tuning the SOM reference vectors for best class separations [13]

To deal with nonvectorial data, there are two approaches. One of them approaches is based on codify input data with real vectors. There are good examples of this method in [5,**?**,20]. Other approach is based on create a map of models using directly original data. Changes in SOM algorithm are made to make it capable to deal directly with non vectorial data. One of these methods is *Median SOM* described originally in [6] and Kohonen and Somervuo in [12] have shown how this algorithm works for clustering a large protein sequence database. We will describe it in section **??** describes this method. We find *Median SOM* examples in [**?**]. We propose a new simple method that no changes on input data are needed. Will be possible to re-use early all the original implementation and need only to include two new functions *integer distance(elementA, elementB)* and *element interpolateNewElement(distance, sourceElement, targetElement).* Section **??** describe this two new functions.

An implementation of multidimensional vectorial SOM developed by T.Kohonen is available in [**?**]. After the seminal Kohonen works, several works have been published that try to enhance the original SOM implementation. Examples of these works are [15,18], and [7] where the authors use evolutionary-learning to make a faster SOM implementation without using a distance function.

3 Self-Organizing Map

The SOM is a non-supervised neural network that tries to imitate the self-organization done in the sensory cortex of the human brain, where neighbouring neurons are activated by similar stimulus. It is usually employed either as a clustering/classification tool or as a method to find unknown relationships among a set of variables that describe a problem.

The main property of the SOM is that it makes a nonlinear projection from a high-dimensional data space (one dimension per variable) on a regular, low-dimensional (usually 2D) grid of neurons (see Figure **??**), and, from the self-organization process, the projection preserves the topologic relations while simultaneously creating a dimensional reduction of the representation space (the transformation is made in a topologically ordered way).

The SOM processes a set of input vectors (samples or patterns), which are composed by variables (features) typifying each sample. It then creates an output topological network where each neuron is also associated to a vector of variables (model vector) which is representative of a group of the input vectors. Note in Figure **??** that each neuron of the network is completely connected to all the nodes (each node is a sample) of the input layer. So, the network represents a feed-forward structure with only one computational layer formed by neurons or model vectors.

There are four main steps in the processing of the SOM. Except the first one, the others are repeated until a stop criterion is met:

- **Initialization of model vectors**. Usually it is made by assigning small random values to their variables, but there are some other possibilities such as an initialization using random input samples.
- **Competitive process**. For each input pattern X, all the neurons (model vectors) V compete using a *similarity function* in order to identify the one most similar or closest to the sample vector. Usually, the similarity function is a distance measure (such as an Euclidean distance). The winner neuron is called the best matching unit (BMU).
- **Cooperative process**. The BMU determines the centre of a topological neighbourhood where those neurons inside it (the model vectors) will be updated to be even more similar to the input pattern. A *neighbourhood function* is used to determine the neurons to consider. If the lattice where the neurons are is rectangular or hexagonal, it is possible to consider as neighbourhood functions rectangles or hexagons with the BMU as centre. However, it is more usual to use a Gaussian function to assure that the farther the neighbour neuron is, the smaller the updating to its associated vector is. In this process, all the neurons within a vicinity cooperate to learn.

- **Learning process**. In this step the variables of the model vectors within the neighbourhood are updated to be closer to those of the input vector. It means making the neuron more similar to the sample. The *learning rule* used to update the vector (V) for every neuron i in the neighbourhood of the BMU is:

  $$V_i^t = V_i^{t-1} + \alpha^t \cdot N_{BMU}^t(i) \cdot (X - V_i^{t-1}) \tag{5}$$

  Where t is the current iteration of the whole process, X is the input vector, N_{BMU} is the neighbourhood function for the BMU, which returns a high value (in [0,1]) if the neuron i is in the neighbourhood and close to the BMU (1 if $i = BMU$), and a small value otherwise (0 if i is not located inside the neighbourhood); and α is the *learning rate* (in (0,1]). Both neighbourhood and learning rate depend on t, since it is usual to decrease the radius of the first one and the value of the second in order to impose a higher updating rate at the beginning of the process and almost none in the final iterations.

The recurrent application of Equation 5 and the update of the neighbourhood function, has the effect of 'moving' the model vectors, V_j from the winning neuron towards the input vector X_i. That is, the model vectors tend to follow the distribution of the input vectors. Consequently, the algorithm leads to a topological arrangement of the characteristic map of the input space, in the sense that adjacent neurons in the network tend to have similar weights vectors.

The SOM is further processed using Ultsch method [**?**], the Unified distance matrix (U-Matrix). It uses SOM's codevectors (vectors of variables of the problem) as data source and generates a matrix where each component is a distance measure between two adjacent neurons.

The U-Matrix shows a lattice where there are one cell per neuron in the map, which is the best matching unit for one (or more) of the input samples, and one more cell between every pair of neurons, which represents the distance between them. This distance is showed using a color code (usually blue means near or little distance, and red far or large distance). Each cell corresponding to one neuron in the map may be labeled with the related tag of the pattern which it represents.

It allows us to visualize any multi-variated dataset in a two-dimensional display, so we can detect topological relations among neurons and infer about the input data structure. High values in the U-matrix represent a frontier region between clusters, and low values represent a high degree of similarities among neurons on that region, i.e. clusters.

Therefore, looking at the output of a SOM and its corresponding U-Matrix, it is possible to recognize some clusters as well as the metric-topological relations of the data items (vectors of variables of the problem) and the outstanding variables.

Although Kohonen's SOMs are not as accurate as other tools at the task of classification, they can be applied to many different types of data, yielding a visualization of natural structures in the data and their relations, as well as the natural groupings that could be among them. In addition, SOMs make easy the estimation of the variables that have more influence on these groupings, via the so-called planes analysis. Other statistical and soft computing tools can also be used for this purpose, but Kohonen's SOMs offer a visual way of doing it which is much more intuitive.

Fig. 1 Left: SOM with 20 neurons (4 rows and 5 columns) and marked neighborhood area with radius 1, centered in winner neuron in a rectangular topology. Right: graphical representation of SOM training process

4 A generalized SOM approach for any kind of data

4.1 The SOM approach for vectorial data

SOM learning process iteratively perfoms the following three steps:

- A certain element of input data is chosen.
- The model of the SOM grid with the best matching with the selected datum is located
- This winner model and its neighborhood is appropriately updated

In this process, the distance measure plays a significant role: the winner model is the *closest* one to the input datum and the update is carried out with the aim of taking the neighborhood *nearer* to the input datum (see Fig. 2). In the case of vectorial data, for each model m_i, the following formula is used to calculate the updated value:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \tag{6}$$

where $m_i(t)$ is the model located at i grid position, $x(t)$ is the selected input datum, $t = 0, 1, 2...$ is the discrete-time coodinate, and $h_{ci}(t)$ is called *neighborhood function.* This *neighborhood function* usually takes a *Gaussian* form:

$$h_{ci}(t) = \alpha(t) \cdot exp\left(-\frac{\| r_c - r_i \|^2}{2\sigma^2(t)}\right) \tag{7}$$

where $\alpha(t)$ is the learning factor in time t, r_c and r_i are the locations of models c and i respectively, and $\sigma(t)$ stands for the Gaussian width.

[include Gaussian function graphic]

The goal of this equation is to compute a new value $m_i(t+1)$ wich is $h_{ci}(t)[x(t)-m_i(t)]$ *nearer* to $x(t)$ and $h_{ci}(t)[x(t) - m_i(t)]$ *farer* from $m_i(t)$. With vectorial data the "+" operator together with the euclidean distance permits us to easily obtain the new value. However, if we are working with nonvectorial data this process has to be appropiately adapted.

4.2 Working with any kind of data

If we want to apply the above described process to nonvectorial data, we need to find a suitable distance measure and an operator that allows us to take the models closer to the selected input datum at each iteration. Let us formalize this idea.

Theorem 1 (Conditions for distance function) *Let D be a data domain (vectorial or not). Let $d : D \times D \longrightarrow R$ be a distance function denoted d(x,y). The distance function must hold the following conditions foreach pairs of elements $x, y \in D$:*

C1) $d(x,y) \geq 0$ *where equality holds if and only if* $x = y$.
C2) $d(x,y) = d(y,x)$
C3) $d(x,y) \leq d(x,z) + d(z,y)$

Theorem 2 (Neighbor Data Set) *Let D be a data domain. Let* $d : D \times D \longrightarrow R$ *be a distance function with the above Theorem restrictions. The Neighbor Data Set of distance* δ *of value x,* $N(x,\delta)$*, is defined as follows:*

$$N(x,\delta) = \{y \in D | d(x,y) = \delta\}$$

That is, the Neighbor Data Set contains all the elements of domain D that are at distance δ from a given datum x. With this definition, we can now proceed to define the operation that allows us to take a model closer to a given element of the domain.

Theorem 3 (Interpolation function) *Let D be a data domain (vectorial or not). let* $d : D \times D \longrightarrow R$ *be a distance function. We say that* $I : D \times D \times R \longrightarrow D$ *is an Interpolation function if the following property holds:*

$$I(x,y,\delta) \in N(x,\delta) \cap N(y, d(x,y) - \delta), \forall \delta \leq d(x,y)$$

That is, the interpolation function, given two elements x and y of the domain and a distance δ, permits us to obtain a new element I(x,y,δ) that is δ farer from x and δ closer to y. In the case of our SOM grid, the interpolation function permits us to generalize equation 6 as follows:

$$m_i(t+1) = I(m_i(t), x(t), h_{ci}(t) \cdot d(m_i(t), x(t)) \qquad (8)$$

This way, with the generalized versions of the + operator and the euclidean distance, we can now apply the above method to any kind of data with the only restriction of having both a suitable distance and interpolation functions.

4.3 The extended algorithm

5 The case of strings

5.1 A distance measure for string data

Hamming Distance Originally this distance measure was defined for binary codes but can be applied to any ordered set of discrete values. The hamming distance is defined only for strings of equal length. The distance consist of the number of different symbols.

```
x = (1,0,1,1,1)
y = (0,1,1,0,1)
dH(x,y)=3

u=(s,a,l,e)
v=(y,a,l,e)
dH(u,v)=1
```

Fig. 2 Updating a SOM with symbol string models using interpolation: $x(t) = ABCD$ and $m_i(t) = A$. The new interpolated valued with, e.g., $h_{ci}(t) = 1$ is $m_i(t+1) = AB$. Only one change that leave $m_i(t+1)$ nearest to $x(t)$. Left: before model updating, Right:After model updating.

Levenshtein or Edit distance The distance between strings A and B is defined as:

$$LD(A, B) = min\{a(i) + b(i) + c(i)\} \tag{9}$$

B is obtained from A by a(i) replacements, b(i) insertions and c(i) deletions of a symbol. There exist an infinite number of combinations for $\{a(i) + b(i) + c(i)\}$, but the minimum can be formed using a dynamic-programming method. The distance measure is more reliable if the editing operations are provided with different weights. There are a weighted version of Levenshtein distance described by T. Okuda et al. in [**?**], with this equation:

$$WLD(A, B) = min\{pa(i) + qb(i) + rc(i)\} \tag{10}$$

where p, q and r may be obtained from the confusion-matrix of the alphabet, as the inverse probabilities for particular types of error to occur. However, this is only usefull for strings of words.

String::Diff algorithm distance This distance measure is based on String::Diff Perl module developed by Kazuhiro Osawa. It is similar to Levenshtein distance, but without using replacements as a basic edit operation.

The distance between strings A and B is defined as:

$$SDD(A, B) = min\{b(i) + c(i)\} \tag{11}$$

B is obtained from A by b(i) insertions and c(i) deletions of a symbol

To compute string distance we use String::Diff Perl module. This is a fast algorithm to obtain strings differences. This is a execution example of String::Diff module wrote in Perl:

```
my $diff = String::Diff::diff_merge("AB", "ABCDE",
   remove_open => "<del>",
   remove_close => "</del>",
   append_open => "<ins>",
   append_close => "</ins>",
);
print "$diff\n";# this is AB<ins>CDE</ins>
```

Arguments *remove_open*, *remove_close*, *append_open* and *append_close* indicates the marks for insert or delete a substring. We have transformed original String::Diff output, to this one:

```
#This is new diff string AB<ins>C</ins><ins>D</ins><ins>E</ins>
```

It is easy to interpolate a new symbol string with a certain distance using this difference string.

5.2 An interpolation function for string data

new_interpolated_model implementation for symbol strings Other important element required to compute new SOM algorithm for nonvectorial data is *new_interpolated_model* function. We have to implement a different function for each data type we want to compute. In case of symbol strings we implemented two different version. One of this is based on Levenshtein algorithm. This version has complexity $O(n^3)$. When string lengths is more than 1000 characters as will be the case of DNA strings, SOM algorithm time to compute increase drastically. This is the reason why we test different distances measures and finally decided to use String:Diff package. This implementation is fast for any string length. Other advantage this measure has got is the possibility to easily interpolate new string using difference string output. This is the pseudocode for *new_interpolated_model* function:

```
 $new\_interpolated\_model(m_i(t), x(t), d(t))$ {
        $diff = distance(m_i(t), x(t))$
        return $calculate\_new\_model(diff, d(t))$
}
```

where $calculate\_new\_model(diff, d(t))$ function, process the difference string returned by $distance(m_i(t), x(t))$ function doing $d(t)$ number of changes. Let the difference string returned by distance function:

`AB<ins>C</ins><ins>D</ins><ins>E</ins>`

make $d(t)$ changes over it, will result the string:

`ABCD<ins>E</ins>`

to obtain the new interpolated string the only thing needed is to eliminate the substring starting at first *"ins"* or *"del"* and finishing at string end. In this example, new interpolated string will be "ABCD". The string have distance 2 from "AB" and is two positions nearer to "ABCDE"

5.3 The approach for string

6 Experimentation

We included in this paper two experiments to illustrate how the algorithm works. First one, are developed using a short symbol string data set in order to easily check clustering capabilities. The second one, is made using a database with 100.000 english word to test how the algorithm works with a large database.

To test the new algorithm presented in Sec. 4 we have implemented a new Perl module called AI::NeuralNet::SOMString. This Perl module is based on a previous implementation developed by Alexander Voischev called of AI::NeuralNet::SOM. One element required to SOM implementation is a distance measure. There are several options to calculate a symbol string distance. We presents some of the most used symbol string distances.

Table 1 Arguments for SOM using short symbol string dataset

Argument	Value
Alpha	0.5
Radius	9.9
Neighborhood function	Gaussian
Alpha function	Linear
Rows	10
Columns	10
Rounds	27

Fig. 3 Symbol string clustering 10x10 SOM after training process. Input data set consist of five strings: CAT, CATTLE, BAT, BATTLE and BATTLEFIELD.

Fig. 4 10x10 SOM umatrix representation. Input data set consist of five strings: CAT, CATTLE, BAT, BATTLE. and BATTLEFIELD. Black represent zero distance. Dark gray represents low distance and light gray represents high distance. This map is calibrated with input values. Each label is located in one of the best matching node.

Input selection An important aspect in SOM experiments is how input are ordered. If we have, e.g., an alphabetically ordered english dictionary as input data set, with radius around 3, we can see the whole map first with words starting with "A", then with words starting with "B" and same with all letters in alphabet. Depending on when we decided to stop the learning process, the map will show a clustering view or other drastically different. Will be better to merge randomly the dataset before use it.

Table 1 represents SOM arguments for the experiment. Note that radius is nearest to SOM array dimension. Several experiments realized with smallest radius values, results in maps with no updated areas. Initial randomly used valued remain after training process. Gaussian Neighborhood function change deeply neurons that are closes to winner. Linear alpha function makes the transitions between clusters gradually. Fig. 3 respresents SOM for strings using a initialization map with string randomly generated with a length no longer than three characters. Fig. 4 represents umatrix for same SOM. Labels is umatrix is the result of map calibration. Taking each input value and locating in the map the nearest value. This map present several models with same distance to each input value. In this case we take one of them randomly.

6.1 **Simple data set experiment**

Data set description This experiment uses same dataset than Panu Somervuo in [21] to easily compare results. Data sets consist on five strings: CAT, CATTLE, BAT, BATTLE and BATTLEFIELD.

6.2 **Large database experiment**

7 Conclusions and further work

Mathematical properties of discrete learning process like convergence, etc.

- Developed methods for winner selection

  Observed differences between continuous data input and discrete data input:

1. **Shorter training process** than training process for real vectors. The number of possible changes are limited by the maximum distance between SOM models and input elements.
2. Depending on items distance and input data set diversity, after training, **SOM will present regions with zero distance**. This happends because possible values in training process are discrete and usually very short. In case of real values, after calculate a new *model* $m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$ it is hard to have exactly the same values. Will be similar to use a discrete numerical space. Several input values will correspond to one output value, e.g., using a round function.
3. **Winner selection**. In case of real data, is difficult to have exactly the same distance after apply adapt function. In case of discrete values with a short range, it is easy to have several winners. In these cases, selection method for winners will be used. We used two methods for winner selection. First one, is a randomly selected winner from winner set. Experiments realized with this *winner selection method* for a large dataset cause no clustering, because each time a different area in map are selected for a certain model. [Sucesive] epochs will modify previous mini area and depending on radius will delete this mini area. Other method selection is to use first funded value. It is faster method and give good enough results. The reason why this method works is because first tends to be located always in same area and clustering is reinforced. When frontier values are selected to be updated, that updating process will not cause the deletion of a bigger cluster than in previous method. In a certain way, second one method (first winner) tends easily to clustering and first one (randomly winner selection) tends to dispersion.
4. In case of real values, it is easy to see map areas where there are no input elements coincidences. Thanks to Euclidean distance, after train process *models* are nearest to the areas where impact more input values. I case of non numerical input values, is most difficult to see that. We have not a Euclidean distance that give as a spacial distance measure for the elements in the map. A way to know how many input values impact in any area we use a graphic with.

References

1. Geoffrey J. Chappell and John G. Taylor. The temporal Kohonen map. *Neural Networks*, 6:441–445, 1993.
2. Luis Fernando de Mingo López, Nuria Blas, and Miguel Díaz. *A String measure with symbol Generation: String self-organizing maps*, chapter A String Measure with Symbols Generation: String Self-Organizing Maps, pages 123–130. Springer Berlin Heidelberg, 2009.
3. B. Hammer, A. Micheli, A. Sperduti, and M. Strickert. A general framework for unsupervised processing of structured data. *Neurocomputing*, 57:3–35, March 2004.
4. Barbara Hammer and Thomas Villmann. Classification using non standard metrics. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 303–316. D-Side Publications, 2005.
5. T. Honkela, S. Kaski, T. Kohonen, and K. Lagus. Self-organizing maps of very large document collections: Justification for the WEBSOM method. In I. Balderjahn, R. Mathar, and M. Schader, editors, *Classification, Data Analysis, and Data Highways*, pages 245–252. Springer, Berlin, 1998.
6. T. Kohonen. Self-organizing maps of symbol strings. Technical Report A42, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland, 1996.

7. T. Kohonen. Fast evolutionary learning with batch-type self-organizing maps. *Neural Processing Letters*, 9:153–62, 1999.
8. Teuvo Kohonen. Self-organizing formation of topologically correct feature maps. *Biol. Cyb.*, 43(1):59–69, 1982.
9. Teuvo Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.
10. Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).
11. Teuvo Kohonen. Self-organization of very large document collections: State of the art. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of ICANN98, the 8th International Conference on Artificial Neural Networks*, volume 1, pages 65–74. Springer, London, 1998.
12. Teuvo K.ohonen and P. Somervuo. How to make large self-organizing maps for nonvectorial data. *Neural Networks*, 15(8-9):945–952, October-November 2002.
13. Teuvo Kohonen and Panu Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1):19–30, 1998.
14. P. Koikkalainen and E. Oja. Self-organizing hierarchical feature maps. In *Proc. IJCNN-90, International Joint Conference on Neural Networks, Washington, DC*, volume II, pages 279–285, Piscataway, NJ, 1990. IEEE Service Center.
15. Pasi Koikkalainen. Fast organization of the Self-Organizing Map. In Abhay Bulsari and Björn Saxén, editors, *Proc. Symp. on Neural Networks in Finland*, pages 51–62, Helsinki, Finland, 1993. Finnish Artificial Intelligence Society.
16. Jorma Laaksonen, Markus Koskela, Sami Laakso, and Erkki Oja. PicSOM—content-based image retrieval with self-organizing maps. *Pattern Recognition Letters*, 21(13–14):1199–1207, Dec 2000.
17. J. Lampinen and E. Oja. Self-organizing maps for spatial and temporal AR models. In Matti Pietikäinen and Juha Röning, editors, *Proc. 6 SCIA, Scand. Conf. on Image Analysis*, pages 120–127, Helsinki, Finland, 1989. Suomen Hahmontunnistustutkimuksen seura r. y.
18. J. Lampinen and E. Oja. Fast computation of Kohonen self-organization. In F. Fogelman-Soulié and J. Herault, editors, *Neurocomputing: Algorithms, Architectures, and Applications, NATO ASI Series F: Computer and Systems Sciences, vol. 68*, pages 65—-74. Springer, Berlin, Heidelberg, 1990.
19. A. Mora, J. Merelo, C. Briones, F. MorÃ¡n, and J. Laredo. *Clustering and Visualizing HIV Quasispecies Using Kohonen's Self-Organizing Maps*, chapter Clustering and Visualizing HIV Quasispecies Using Kohonen's Self-Organizing Maps, pages 940–947. Springer Berlin Heidelberg, 2007.
20. M. Oja, G. O. Sperber, J. Blomberg, and Samuel Kaski. Self-organizing map-based discovery and visualization of human endogenous retroviral sequence groups. *International Journal of Neural Systems*, 15(3):163–179, June 2005.
21. Panu Somervuo. Online algorithm for the self-organizing map of symbol strings. *Neural Networks*, 17(8-9):1231–1239, October-November 2004.
22. Alessandro Sperduti. Neural networks for adaptive processing of structured data. *Lecture Notes in Computer Science*, 2130:5–??, 2001.
23. Thomas Voegtlin. Context quantization and contextual self-organizing maps. In *Proceedings of the International Joint Conference on Neural Networks*, volume 6, pages 20–25, Piscataway, NJ, 2000. Inst des Sciences Cognitives, IEEE.