In J. Mira, J. Cabestany, A. Prieto editors, *International Workshop on Artificial Neural Networks*, volume 686 of *Lecture Notes in Computer Science*, pages 328—333. Springer Verlag, 1993.

Vector Quantization and Projection neural network

P. Demartines & J. Hérault

INPG, Labo. TIRF, 46, av. Félix-Viallet, F-38031 Grenoble, France

Classical data analysis techniques are generally linear. They fail to reduce the dimension of data sets where dependence between observed variables is non-linear. However, for numerous scientific, industrial and economic areas, it should be desirable to obtain a low-dimensional parametric representation of the data set. Model fitting is a way to obtain a usable representation of an observed phenomenon, but it requires expert knowledge about the phenomenon. Moreover, hidden relations between observables could be not revealed. Kohonen maps are shown to be an alternative techniques, able to map even strongly non-linear data sets [1]. Unfortunately, they have an *a priori* fixed shape and neighbourhood structure, thus their use requires some informations about the shape and the dimension of the underlying parameters space. We propose here a new self-organizing neural network, composed of two connections layers. The first one quantizes an input data set, and the second one progressively constructs the projected shape and neighbourhood on an output space of any chosen dimension. We illustrate the algorithm for various applications.

Keywords: Self-Organization, Data Analysis, Non-linear Data Redundancy.

1. Introduction

A problem frequently encountered in many contexts is how to determine a smallest set of independent variables able to describe a redundant multidimensional data set. The space of these underlying parameters is called the variety of the data set.

In margin of many improved techniques (linear regression, canonical analysis, discriminant analysis, model fitting), neural networks have proven certain capabilities in data analysis. Oja proved that a simplified neuron model may act as a principal component analyzer [10]. Cherkassky & Lari-Najafi show how to do non-parametric regression analysis with self-organized neural network [2]. Sanger [12], and Samardzija & Waterland [11] describe neural networks that compute eigenvectors and eigenvalues of data (that is, how to implement a Principal Component Analysis, PCA, in a neuronal way).

The problem of all PCA-like methods (and of the original PCA itself) is that they are only driven by linear dependence between the observed variables. This is a severe limitation since in a number of technical and scientific applications, this dependence is strongly non-linear. In such cases, these methods fail to reduce the number of variables without loosing information. In fact, when statisticians encounter this problem, they generally try to build a model of the observed phenomenon, then they search the proper model parameters with least mean square techniques (fitting). The model is generally not easy (even impossible) to build when the data dimension (the number of observed variables) is high.

Kohonen's Self-Organizing Maps (SOM) [7] have looked very promising, because of their capability to *map* the input data set shape. This property is illustrated in figure 1 (from [1]).

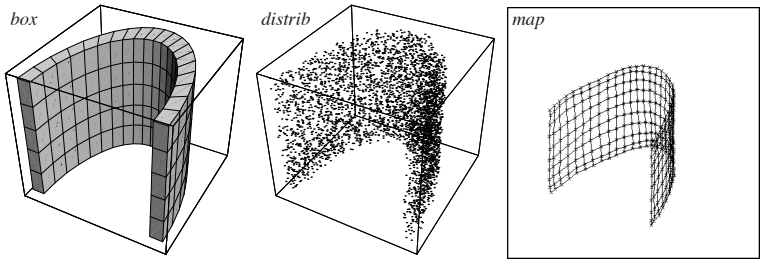

Fig. 1. Placement of the Kohonen grid on a "horseshoe" shape input distribution in a 3D space. This is a good example of non-linear distribution where PCA method fails to reduce the dimension.

Although the SOM seems to overcome the problem of non linear redundancy, it presents a major drawback being the fact that the projection geometry structure is fixed *a priori*. This implies one knows the variety of the data set, in order to choose the correct grid dimension. Moreover, non square shapes are not very well mapped with square maps, as shown in figure 2. For example, the mapping shown on figure 1 was feasible thanks to the fact we knew that the variety dimension was 2, and that the unfolded distribution was about 3 times larger than high, so we took a 2D grid network of 10x30 neurons. On the contrary, in figure 2, where we took a "naive" 15x15 network. Another fact is that the projection made by the original SOM is discrete, as long as only the winner element is considered for one data input (a continuous projection is therefore not available in the output space). In order to overcome all these problems (shape of the input distribution, non-linear projection and continuous projection), we propose a self-organized network where neurons "compute" their position in the projection space in an adaptive manner, and where their activity drives an interpolation algorithm.

2. The VQP network

Considering the function of a Kohonen map, one should focus on two main features:

1) The Vector Quantization property, for which the particular structure of the SOM algorithm is interesting because it reduces the number of dead units, that is, the number of units out of the input distribution [6].

2) The topologically correct projection, where neighbour neurons have neighbour synaptic weights, that is, small distances in the output space (the space of the neuron grid) reflect small distances in the input space.

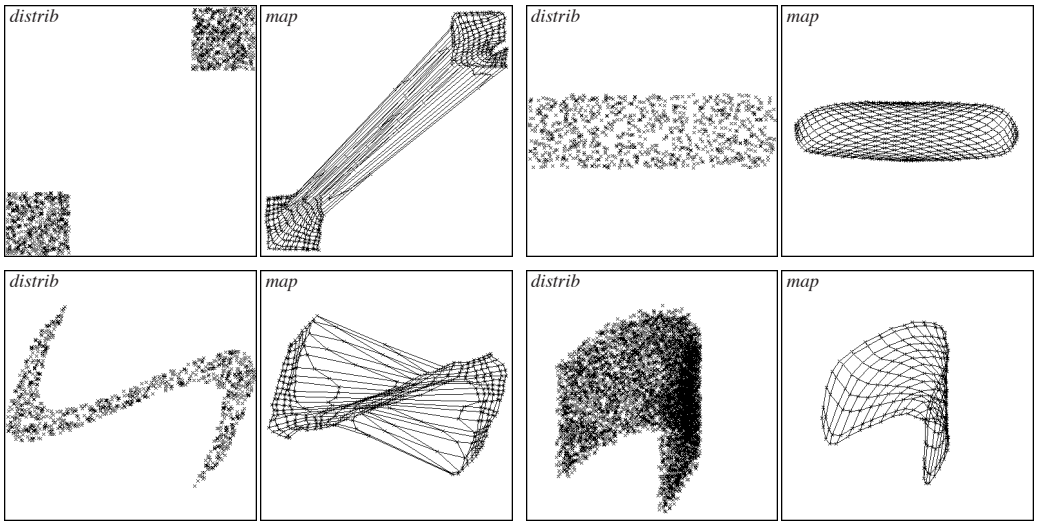

Fig. 2. Some particular distributions and their mapping with a "naive" Kohonen map, that is, where the map is square. This is a choice generally made when no assumption is possible about the distribution shape (for example, when the data set is high-dimensional).

The new self-organized neural network proposed here, Vector Quantization and Projection network, presents these two main features, but does not impose any particular *a priori* defined shape, as a rectangular or hexagonal map. This network is composed of two connection layers as shown in figure 3:

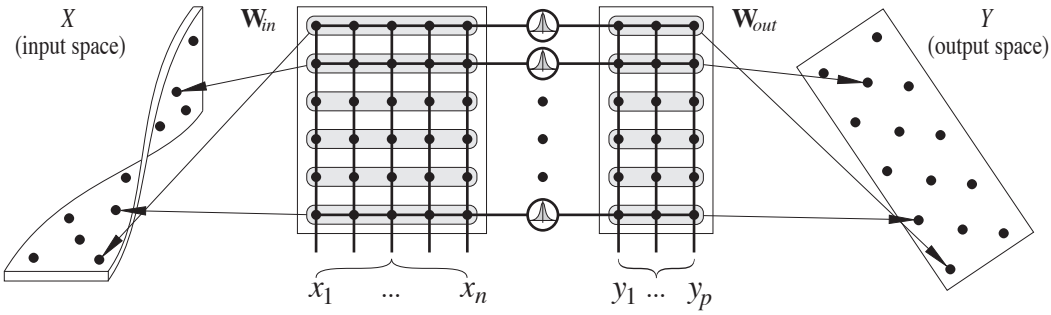

Fig. 3. Structure of the VQP network. The first layer realizes a vector quantization of the input distribution. The second layer reproduces the configuration of the first one in a self-organized way.

The aim of the network is to project the n-dimensional input vector $\mathbf{x}$ $[x_1, ..., x_n]^T$, to a p-dimensional space, the result being the output vector $\mathbf{y}$ $[y_1, ..., y_p]^T$, preserving as much as possible the topological information. The definition of output vector $\mathbf{y}$ is not supervised; the only constraint attached to it is to try to mimic, at least locally, the topology of the input space. The dimension p is the only information fixed (by the user) for $\mathbf{y}$. The first layer realizes a *vector quantization* of the input space, while the second one performs a *projection* towards the output space. As well as in Kohonen maps, where neurons may be considered to point in the input space with their own weight vector, here, each neuron i points to $\mathbf{W}_{in,i}$ in the input space, and also to a corresponding position $\mathbf{W}_{out,i}$ in the output space. When an input vector $\mathbf{x}$ is presented, each neuron i computes an activity a_i reflecting the proximity of $\mathbf{W}_{in,i}$ to the vector $\mathbf{x}$ through a radial basis function (RBF). Such a function is for example the Gaussian function of the euclidean distance:

$$a_i = \exp\left(-\frac{1}{\lambda_i^2} \left\| \mathbf{x} - \mathbf{W}_{in,i} \right\|^2 \right) \qquad (1)$$

where $\|\mathbf{x}\text{-}\mathbf{W}_{in,i}\|$ is the euclidean distance between the input vector $\boldsymbol{x}$ and the input weight $\mathbf{W}_{in,i}$ of neuron i, while λ_i is the influence radius of the neuron i.

Then, the projection is computed as the sum of all output vectors, weighted by the neuron activities (this is an interpolation within the output vectors with Gaussian kernels):

$$\mathbf{y} = \frac{\sum_{i=1}^{N} a_i \mathbf{W}_{out,i}}{\sum_{i=1}^{N} a_i} \qquad (2)$$

where N is the number of neurons. The adapted values through self-learning are the input matrix $\mathbf{W}_{in}$, the influence radius λ_i of each neuron i , and the output matrix $\mathbf{W}_{out}$.

An example of a projection (after adaptation) is given in figure 4:

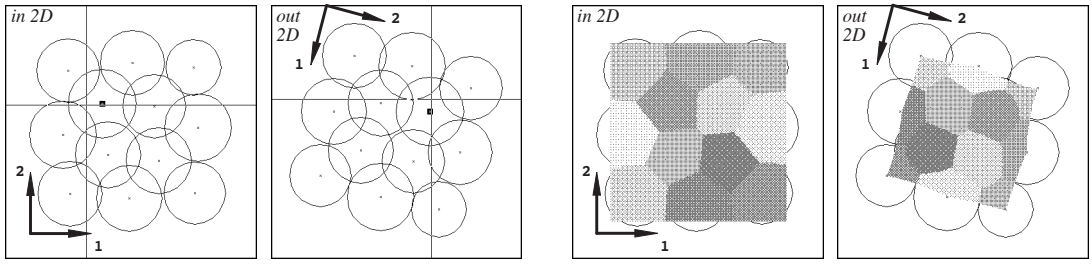

Fig. 4. The network is able to project input vectors into output space. This is shown here after organization on a 2D square distribution for one vector (left side), and for the whole distribution (right side).

The network was trained with a square uniform data set in 2 dimensions, the output space being also in 2 dimensions. In the left part, we show a particular vector presented on input (the cross in the "*in 2D*" display) and its projection (the corresponding cross in the "*out 2D*" display). In the right part, the whole distribution is projected. An interesting feature is that, for any application, the projection can also be used backward, that is, presenting a vector in the output space and computing the corresponding input position. This is simply done by exchanging $\mathbf{W}_{in}$, $\mathbf{W}_{out}$, $\mathbf{x}$ and $\mathbf{y}$ in (1) and (2).

3. Adaptation rules

3.1. First layer

We do not want to focus here on the quantization algorithm. Several algorithms have proven their efficiency from different points of view [6, 9, 14]. Let us just say that we have considered a basic Competitive Learning algorithm [9], where neuron activities are Gaussian kernels. Let us also say some words about the radii adaptation of these kernels. Firstly, the main challenge here is to achieve a correct and smooth projection, even between centroids. This implies kernels with appropriate radii. Secondly, we want that units, initially out of the input distribution (well known as dead units in VQ theory) are, after a while, attracted by the distribution. This implies temporarily growing radii. We will first solve these two problems independently, then combine their solutions into a unique learning law.

After a winner has been chosen according to eq. (3) explained later, a first idea consists in decreasing the sensitivity of units that win too often, giving the other ones a chance to be elected.

$$w \mid \lambda_i' a_i \le \lambda_w' a_w \forall i \qquad (3)$$

$$\lambda_i' \leftarrow \begin{cases} \lambda_i' + \frac{\Delta\lambda}{N} - \Delta\lambda & i = w \\ \lambda_i' + \frac{\Delta\lambda}{N} & i \neq w \end{cases} \qquad (4)$$

$$\mathbf{W}_{in,w} \leftarrow \mathbf{W}_{in,w} + \alpha\left(\mathbf{x} - \mathbf{W}_{in,w}\right) \qquad (5)$$

This is done through eq. (4), at each iteration, by reducing the influence radius of winning unit by a small value $\Delta\lambda$, while all others are increased by $\Delta\lambda/N$ (thus the mean of all radii is left constant). Dead units (never winning) have thus growing radii until they finally win the competitive election process, being then attracted by the distribution (eq. 5). In order to speed up this phenomenon, we consider the expression $\lambda_i a_i$, and not only a_i, to determine the winner w (eq. 3). Without this, dead units would have to increase their influence radius up to a huge value to get a chance to win.

This idea is similar to what is sometimes called "conscience" [5] or "neuronal tiredness" [4]. A secondary interesting effect is that regions of the input space with high density of samples are mapped with a number of neurons with small radii, while regions with low density are mapped with few neurons with big radii.

The other problem, concerning the good interpolation between centroids, is solved as following: consider an input vector being presented to the network, and falling between the input pointing position of some (say, 3 or 4) neurons. Let us remark that almost only these neurons are responsible for the **y** position resulting of eq. (2), because other neurons are too far, thus their activity is negligible. Then, a good empirical constraint to obtain smooth interpolation is to impose the sum of these maximal activities to be roughly 1 in average. This has been found a good approximation of the optimal value (that is, the one which gives the minimal distortion) in one dimension.

$$\lambda_i'' \leftarrow \lambda_i'' \left[1 + a_i \varepsilon \left(1 - \sum_{i=1}^{N} a_i\right)\right] \qquad (4 \text{ bis})$$

Since other neurons activity is near to 0, we may in fact consider the sum of all activities and drive its average value to be approximately 1.

When the sum is too high, the influence regions of neurons pointing near the presented input are supposed to overlap too much. Then, the radii of these units are reduced in function of their contribution in the sum. On the contrary, when the sum is too low, the radii are increased. The weighting by a_i in (4 bis) avoids not concerned kernels to have their radius changed.

We may now consider λ_i' and λ_i'' as a unique value λ_i to simplify the algorithm, as long as the two described effect (dead units suppression and good interpolation) are not antagonistic. To summarize, the first layer adaptation is described by two equations (one for the centroids position, and the other one for their radius), whose dynamics is controlled by three parameters, $\Delta\lambda$, ε, and α. In our simulations, we have taken $\alpha = 10^{-1}$, $\Delta\lambda = 10^{-3}$, and $\varepsilon = 10^{-3}$.

3.2. Second layer

The second layer aims at reflecting the topology of the first one. When 2 units are neighbours in the input space, they should be also neighbours in the output space. On the contrary, if two units are far in the input space, they should be far in the output space.

Shepard and Carroll [13] describe an optimization algorithm which realizes such a projection of N points in n dimensions onto p dimensions. The method is based on the definition of an inverse continuity index κ, and its minimization through a gradient descent algorithm.

$$\kappa = \sum_i \sum_{j \neq i} \frac{dx_{ij}^2}{dy_{ij}^4} \Bigg/ \left[\sum_i \sum_{j \neq i} \frac{1}{dy_{ij}^2}\right]^2 \qquad (6)$$

where dx_{ij} is the distance between points i and j in the input space, and dy_{ij} their corresponding distance in the output space. Smoother functional relations are indicated by smaller values of κ [13].

From a computational point of view, the minimisation of κ is very heavy. The main problem of this method, for a neuronal implementation, is the globality of κ, whose computing involves all inter-neurons distances. This becomes a real problem when the number of representative points is large (i.e. $N > 100$). For this reason, we decided to provide a simpler and more local algorithm: the ideal expected state where distances are conserved is reached simply by maximizing the weight distances correlation between the input and the output spaces. That is, considering the winner at time t ($w(t)$), and the winner at time t-1 ($w(t\text{-}1)$), we compute in (7) their distance in the input space (dx), respectively in output space (dy). Then, the output weights of both neurons $w(t)$ and $w(t\text{-}1)$ are adapted to obtain a better matching between the distances dx and dy (8 and 9).

$$\begin{aligned} \Delta\mathbf{x} &= \mathbf{W}_{in,w(t)} - \mathbf{W}_{in,w(t-1)} \quad , dx = \|\Delta\mathbf{x}\| \\ \Delta\mathbf{y} &= \mathbf{W}_{out,w(t)} - \mathbf{W}_{out,w(t-1)} \quad , dy = \|\Delta\mathbf{y}\| \end{aligned} \tag{7}$$

$$\beta = \frac{dx - dy}{dx + dy} \tag{8}$$

$$\begin{aligned} \mathbf{W}_{out,w(t)} &\leftarrow \mathbf{W}_{out,w(t)} + \frac{\beta}{2}\Delta\mathbf{y} \\ \mathbf{W}_{out,w(t-1)} &\leftarrow \mathbf{W}_{out,w(t-1)} - \frac{\beta}{2}\Delta\mathbf{y} \end{aligned} \tag{9}$$

In (8), β represents a distance adaptation factor. That is, the output distance should be multiplied by β to become more correct: if $dx < dy$, then $\beta < 1$, and if $dx > dy$, then $\beta > 1$. In (9), $\mathbf{W}_{out,w(t)}$ and $\mathbf{W}_{out,w(t\text{-}1)}$ are moved away or brought closer together in function of β. These both cases are illustrated on figure 5. This very simple law unfolds the projected space and drives our projector network toward a state where distances are approximately preserved. Let us remark that, due to the form of (9), the center of gravity of output weights remains obviously constant. On the other hand, the orientation is undefined.

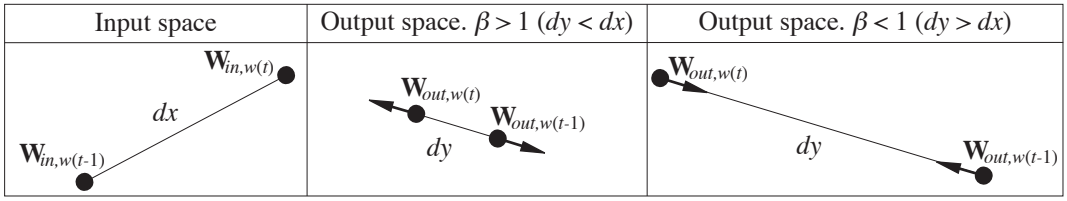

Fig. 5. Adaptation of output weights for winner and last winner in both cases $\beta > 1$ and $\beta < 1$.

In practice, when the dependence in input space is strongly not linear, it is not possible to obtain a correct projection for both short and large distances. In order to unfold the data cloud, it is desirable to give more importance for short distances than for large ones. Then, the projection will be at least locally correct. This is obtained by the adjunction in (8) of a weighting factor monotonically decreasing with the output distance:

$$\beta = \frac{dx - dy}{dx + dy}\left(\frac{1}{1 + \left(dy / dy_0\right)^p}\right) \tag{10}$$

The exponent p controls how quickly the weighting factor has to decrease; in our simulations, p was fixed to 5, in order to ensure a good separation between short and long distances in output space. The value dy_0 is a parameter that give an order of value for dy.

In the complete algorithm, the two layers are adapted independently but at the same time. That means the two described phases (quantization, then projection) are in fact realized simultaneously in one. It has been observed that once the network has found a correct projection for a given distribution, it converges more easily to another one than from a random state.

The algorithm is illustrated by various examples in section 4. Because it is impossible, in high dimensions, to visually assess the quality of the topological correspondence between $\mathbf{W}_{in}$ and $\mathbf{W}_{out}$,, we compute a graphical representation between the input and output space interpoint distances introduced in [13], and defined for Kohonen maps under the name of "δ_i, δ_w relation" in [3]. In this last representation, some couples of units are randomly taken. These couples are represented on a 2D plot by points whose x-coordinates are the output space distances, and the y-coordinates the input space distances. A strictly linear relation between the two distances (i.e. all point are on a line starting from <0,0>) reflects a perfect projection. As shown in [3] for Kohonen maps, folded maps give δ_i, δ_w relation where the input distance no longer grows after a certain output distance. In this paper, the distances are called dx resp. dy, instead of δ_w and δ_i, thus the relation is called here "dy-dx relation" (dy, output distances, are on the horizontal axis).

4. Examples

4.1. 2D → 2D examples

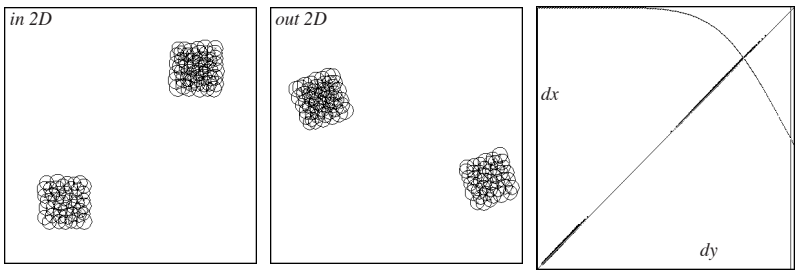

Fig. 6. Here, we show resulting states (after approximately 5000 iterations), given a uniform distribution in two boxes. The dy-dx diagram shows the good matching between output and input distances. As in all following figures, we show on the same graphic the value of dy_0 (the vertical line), and the weighting factor of eq. (10).

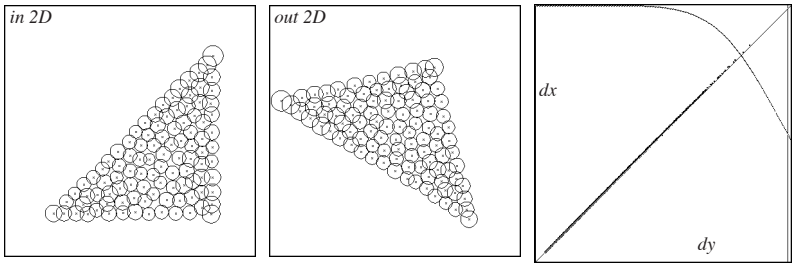

Fig. 7. Same representation as for fig. 6, but with a different input distribution (uniform distribution through a triangular mask).

4.2. 3D → 2D example

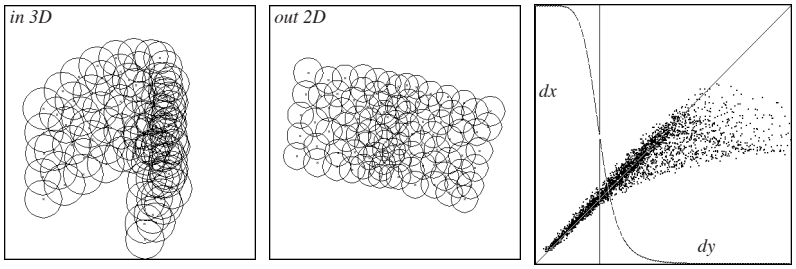

Fig. 8. This is the result obtained on the "horseshoe" shape of figure 1. The network automatically unfolds the distribution, revealing its rectangular shape. The *dy*-*dx* representation show that the projection is strongly non-linear (*d y* distances after the vertical line are not correlated with corresponding *dx* distances, therefore the dots are not on the line *dy*=*dx*).

4.3. Taxonomy (Hierarchical Clustering) of abstract data.

We take here the example given by Kohonen in [8], where abstract data vectors consisting of hypothetical attributes are analyzed to reveal their implicit relations. As shown in figure 11, the VQP algorithm tends to reveal the relations between data and is more easily interpretable than the Self-Organizing Map.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
x_1	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
x_2	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
x_3	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
x_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
x_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

Table 1. (Redrawn from Kohonen [8]). Input data matrix consisting of 32 vectors, each one collecting 5 hypothetical attributes. They are labelled from "A" to "6" for later identification.

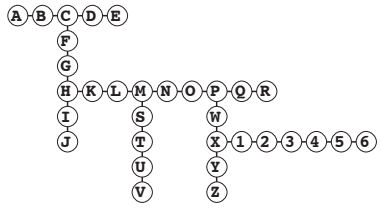

Fig. 9. (Redrawn from Kohonen [8]). Minimal spanning tree (where the most closely similar pairs of items are linked by hand) corresponding to Table 1.

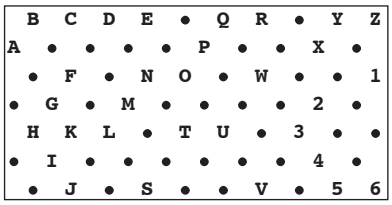

Fig. 10. (Redrawn from Kohonen [8]). Self-organized map of the data matrix of Table 1.

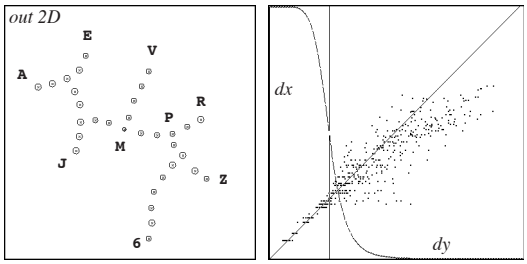

Fig. 11. The projection obtained by the VQP algorithm of the data matrix of Table 1. The output space has been choosed bidimensional. The *dy*-*dx* representation show that the projection is strongly non-linear (*dy* distances after the vertical line are not correlated with corresponding *dx* distances, therefore the dots are not on the line *dy*=*dx*). The VQP algorithm reduces directly the hierarchical clustering in a much more explicit way than the SOM (fig. 10).

4.4 15D → 2D example

We have also used the VQP algorithm to project a particular 15-dimensional distribution onto a plane. The distribution is generated by simulating a system whose purpose is to measure the position of an ultrasonic source on a plane. Several (here $n = 15$) sensors are randomly disposed on the plane and give the distance to the source. We collect all the n distances in a vector, constituting the input distribution. Such a system could learn in a non-supervised way the n to 2 coordinates transform, with no need to know the position of the n sensors. Although the vectors dimension is arbitrarily high (n), the *degree of freedom* of the system is naturally only 2 (because the source is moving on a plane). The VQP output result (not shown here) in two dimensions is the shape of the source position domain (on the plane), for example a square. Of course, such a system could also be implemented in 3 dimensions.

4.4 4D → 2D example

In this example, we show how to separate two classes, consisting here of overlapping toric distributions. These distributions are in 3 dimensions, but we append a class label (0 or 1) magnified by 10 times radius of the torus, in order to achieve the correct separation, giving in fact a 4D input distribution.

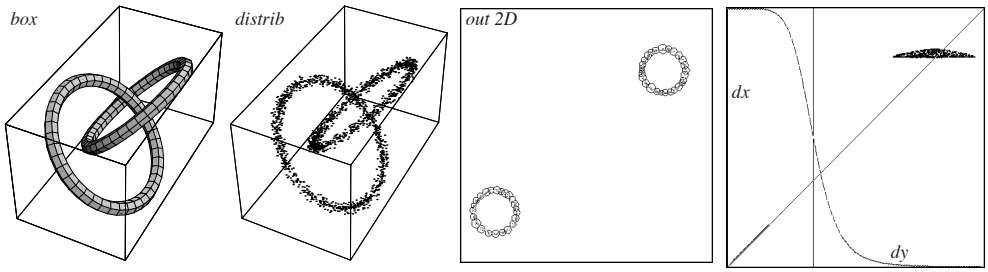

Fig. 11. The two classes are well separated by the algorithm. The *dy-dx* relation is composed of two clouds: one at left of the vertical line ($dy < dy_0$) perfectly linear, corresponding to couples of points of the same class. The other cloud (on the right part of the graphic) is due to couples of points of different classes. For these couples, the projection is strongly non-linear.

5. Conclusion

We have described a new algorithm for data analysis, consisting of a vector quantization of a given distribution (that can be arbitrarily high-dimensional), and its topologically correct projection onto another space of lower dimension. It has be shown some possible applications of this algorithm. We believe that much more applications could be envisaged in various areas:

- Monitoring and control of industrial processes
- Robotics
- Signal (especially speech) processing
- Function approximation

For several applications (especially in robotics), it is interesting to be able to obtain the reverse projection of the data analyzed (that is, to project from the output space to the input one). This can be simply obtained by inverting the respective roles of input and output layer, once the network has been organized.

Further work has to be done to ameliorate the quantization process and to increase the projection fidelity of data itself (not only the centroids). Another hard point is the time wasting for very high dimensional and numerous data vectors (like millions of points in several hundreds dimensions). Evolutive implementation should be envisaged to overcome this problem.

9. References

[1] Blayo F., Demartines P.: *Data analysis: How to compare Kohonen neural networks to other techniques ?* Artificial Neural Networks, International Workshop IWANN'91. A. Prieto Ed. Lecture Notes in Computer Science, Vol. 540, pp. 469-476. Springer-Verlag, 1991.

[2] Cherkassky V., Lari-Najafi H.: *Constrained Topological Mapping for Nonparametric Regression Analysis*. Neural Networks, Vol. 4, pp. 27-40, 1991.

[3] Demartines P.: *Organization measures and representations of the Kohonen maps*. Proc. of the First IFIP Working Group-10.6 Workshop. J. Hérault Ed. Grenoble, 1992.

[4] Demartines P., Blayo F.: *Kohonen Self-Organizing Maps: Is the Normalization Necessary ?* Complex Systems, Vol. 6, No. 2, pp. 105-123, 1992.

[5] DeSieno D.: *Adding a Conscience to Competitive Learning*. Neural Networks, Vol. 1, pp. 117-124, San Diego, 1988.

[6] Hertz J., Krogh A., Palmer R. G.: *Introduction to the Theory of Neural Computation*. Santa Fe Institute Lecture Notes Volume I, Addison-Wesley Publishing Company, 1991.

[7] Kohonen T.: *Self-Organization and Associative Memory (3rd ed.)*. Springer-Verlag, Berlin, 1989.

[8] Kohonen T.: *The Self-Organizing Map*. Proc. of the IEEE, Vol. 78, No. 9, pp. 1464-1480, 1990.

[9] Linde Y., Buzo A., Gray R. M.: *An algorithm for vector quantizer design*. IEEE Trans. Commun., Vol. COM-28, pp. 84-95, 1980.

[10] Oja E.: *A Simplified Neuron Model as a Principal Component Analyzer*. IEEE International Conference on Neural Networks, Vol. 15, pp. 267-273, 1982.

[11] Samardzija N., Waterland R. L.: *A neural network for computing eigenvectors and eigenvalues*. Biological Cybernetics 65, pp. 211-214, 1991.

[12] Sanger T. D..: *Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network*. Neural Networks, Vol. 2, pp. 459-473, 1989.

[13] Shepard R. N., Carroll J. D.: *Parametric Representation of Nonlinear Data Structures*. Proc. of an International Symposium on Multivariate Analysis, pp. 561-592. P. R. Krishnaiah Ed. Academic Press, New-York and London, 1965.

[14] Zeger K., Vaisey J., Gersho A.: *Globally Optimal Vector Quantizer Design by Stochastic Relaxation*. IEEE Transactions on Signal Processing, Vol. 40, No. 2, pp. 310-322, 1992.