Published in proc. of *Neuronimes*
93, pp. 411–424

Representation of Nonlinear Data Structures through Fast VQP Neural Network

Pierre Demartines and Jeanny Hérault

INPG, Labo. TIRF, 46, av. Félix-Viallet
F-38031 Grenoble, France
++33 76 57 45 45

Abstract. The Vector Quantization and Projection neural network (VQP) is a kind of Self-Organizing Map (SOM) where neurons are not fixed on an *a priori* defined discrete lattice, as in Kohonen maps: they find their position in a continuous output projection space through a self-learning algorithm. The main property is therefore the ability to map arbitrary shapes of input distribution, and to project them in a non-linear way, even if the input data sub-manifold is strongly folded. This gives a useful representation of redundant data structures where Principal Components Analysis (PCA) or similar linear techniques fail to reduce to relevant subsets of parameters. Another interesting feature is that, because of a continuous learning scheme, the network remains adaptative, being able to react to an input distribution change. The first presentation of VQP [**?**] was done with a simple projection learning scheme and a classical vector quantization (VQ) algorithm, which suffered from the usual slow rates of convergence of classical algorithms. After a discussion on some VQ algorithms aspects, an optimized version of VQP algorithm is presented.

Keywords: Self-Organizing Maps, Data Analysis, Non-linear Projection, Data Redundancy, High Dimensional Spaces.

Résumé. Le réseau "*Vector Quantization and Projection*" (VQP) est une sorte de carte auto-organisatrice où les neurones ne sont pas fixés sur une grille définie à priori, comme dans les cartes de Kohonen: ils trouvent leur position dans un espace continu de projection par un algorithme d'auto-apprentissage. La propriété principale est la possibilité de représenter des formes arbitraires de distribution d'entrée, et de les projeter de manière non linéaire, même lorsque le sous-espace des données est fortement plié. Ceci fournit une représentation utile de structures de données redondantes là où l'Analyse en Composante Principale (ACP) ou des techniques similaires sont incapables de trouver un sous-ensemble adéquat de paramètres donnant une représentation révélatrice. Un autre aspect intéressant est que, grâce à un apprentissage continu, le réseau reste toujours adaptatif et capable de se réadapter à un changement de distribution d'entrée. Dans sa première forme [5], VQP utilisait une méthode de projection simple et un algorithme de quantification vectorielle classique, et convergeait relativement lentement. Après une discussion sur quelques aspects de la quantification vectorielle, une version optimisée de VQP est présentée.

Mots clés: Cartes auto-organisatrices, analyse de données, projection non linéaire, redondance de données, dimensions élevées.

1 Introduction

The main purpose of Data Analysis is to find hidden relations within a given high-dimensional data set, reducing it to a lower-dimensional space of *underlying* parameters, called the *variety* of the data space.

A well known classical technique is the Principal Component Analysis (PCA), which consists to find p orthogonal axes of largest standard deviation in the n-dimensional data space, and then to project the data set onto the subspace of this basis of p axes. Since it is a linear projection, this technique fails to emphasize strongly non-linear dependance between variables of the data set.

Model fitting is then an alternative technique, whose major drawback is that a relevant parametric model of the data has first to be defined, before to find the correct parameters. This is generally a difficult (not to say impossible) task when the dimension is high. It generally requires human expertise and knowledge of the analyzed phenomenon.

Statisticians are today working on several techniques that generally consist in some kind of particuliar PCA where the metric is locally computed [2]. In fact, these techniques are computationally expensive, and they generally only emphasize the presence of groups within the data set.

In neural networks field, in margin of several techniques belonging to the PCA [13, 17, 16], Kohonen Self-Organizing Maps (SOM) [9] have shown their ability to discover features in an unknown data set, by mapping its submanifold with a neuron grid. This property is discussed in [1] for economic data analysis. Because of the ability of SOM to map distributions with strongly non-linear topology, they overcome PCA methods on this aspects. However, an important drawback is that the projection shape is fixed *a priori* to a rectangular grid. This implies strong distorsions in many cases, and also the well-known problem of *dead units*, typical to most of the competitive learning algorithms, that consists in leaving some units out of the data distribution. Examples of such phenomenons are given in fig. 1.

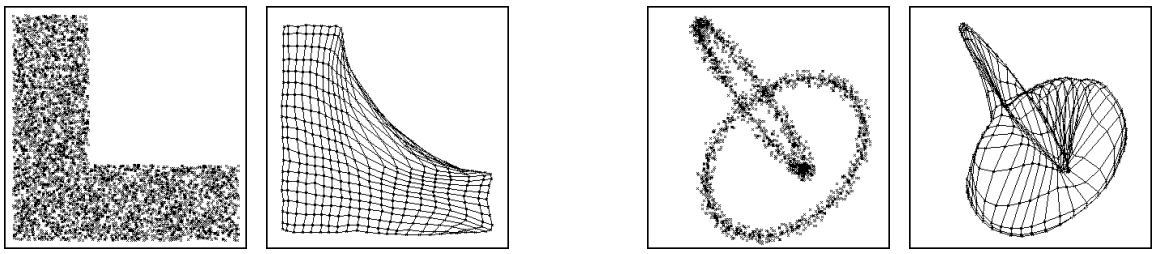


Figure 1: Kohonen Maps imperfections due to the mapping of input data distributions with an *a priori* fixed rectangular grid of neurons. (a) L shaped distribution. (b) distribution within two interleaved rings.

In [5], we introduced a new self-organizing algorithm, called *Vector Quantization and Projection* (VQP), which consists in a kind of SOM where neurons are not fixed on a defined lattice as in Kohonen Maps. In VQP, neurons find their position in the output (or projection) space for whose the only given parameter is the dimension. Thus, the network is able to map any kind of distribution shapes. However, in its original version, the VQP network involved simple algorithms based on winner-take-all schemes that suffered from slow rates of convergence.

We provide here in the following:

1. a presentation of the VQP neural network structure,
2. an overview of some vector quantization algorithms aspects, where benefits of winner-take-most schemes are discussed and taken into account in a efficient VQ algorithm design,

3. an optimized version of the Projection learning rules, which now take into account all units at each time, and,

4. some simple examples which illustrates the VQP algorithm.

2 The VQP network

The VQP function is inspired from the Kohonen maps, whose two main features are:

- The Vector Quantization property, for which the particular structure of the SOM algorithm is interesting because it reduces the number of *dead units* (though not completely avoiding them), that is, the number of units out of the input distribution [7].
- 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.

The VQP network presents both these features, but does not fix any particular *a priori* defined shape, as a rectangular or hexagonal map. The network is composed of two connection layers as shown in figure 2.

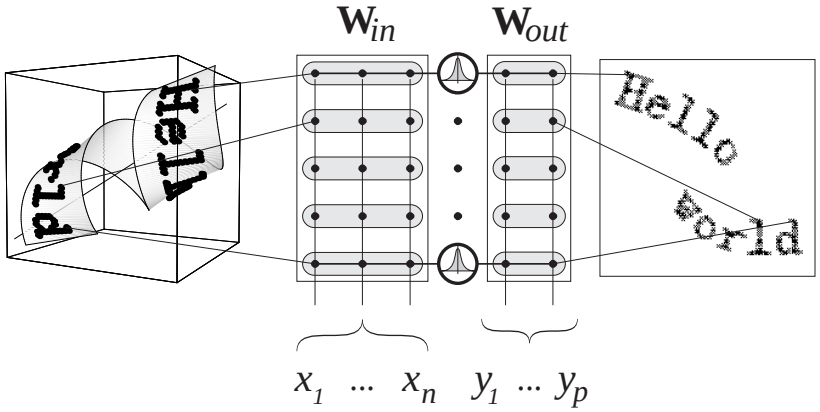


Figure 2: Structure of the VQP network. Here, the input distribution is in 3 dimensions with strongly non-linear relations. The algorithm retrieves the original structure in 2 dimensions, by minimizing the local topology distorsion.

The first layer realizes a vector quantization of the input distribution. The second layer reproduces the local configuration of the first one in a self-organized way. The aim of the network is to project the n-dimensional input vector $\mathbf{x} = [x_1, \ldots, x_n]^T$ to a p-dimensional space, the result being the output vector $\mathbf{y} = [y_1, \ldots, y_p]^T$, 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 (in term of distances conservation) of the input space. Only the dimension p of $\mathbf{y}$ is *a priori* fixed. 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. But, in addition, it points also to a corresponding position $\mathbf{W}_{out,i}$ in the output space, in contrast to Kohonen Maps where this position is the physical location of the neuron in the grid.

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 kernel. Such a kernel could be for example the Gaussian function of the euclidean distance:

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

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

More generally, the kernel should depend of the output dimension p. Thus,

$$a_i = K_p(\|\mathbf{x} - \mathbf{W}_{in,i}\|) \tag{2}$$

Then, the output projection $\mathbf{y}$ is computed as the sum of all output vectors, weighted by the neuron activities (this is a barycentre computation within $\mathbf{W}_{out}$, with neuron activities a_i as weighting factors):

$$\mathbf{y} = \frac{\sum_{i=1}^{N} a_i \mathbf{W}_{out,i}}{\sum_{i=1}^{N} a_i} \tag{3}$$

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 $\mathbf{W}_{out}$.

Let us remark that this projection is continuous, thus input regions between several $\mathbf{W}_{in,i}$ will be projected between the corresponding output vectors $\mathbf{W}_{out,i}$.

Another useful feature of this structure is its ability to perform inverse mapping: therefore, it can be used to retrieve in the input space the corresponding vector of a given output position. This is simply done by exchanging $\mathbf{W}_{in}$, $\mathbf{W}_{out}$, resp. $\mathbf{x}$ and $\mathbf{y}$ in (1) and (3).

The two emerging problems are the learning of vector quantization of input data, then the learning of the projections.

3 Vector Quantization layer

3.1 VQ generalities

Generally, the goal to reach in vector quantizer design is to find a codebook which minimizes the expectation value of the distorsion, measured as the error made by approximating an input vector by a code vector representing it. The most widely used algorithm for vector quantizer design is the Generalized Lloyd Algorithm (GLA) [11], also refered to, in a slightly different form, as "k-means" algorithm. In essence, this algorithm satisfies two necessary conditions to make optimal quantification, i.e. *centroid condition* and *nearest neighbor* condition. The nearest neighbor condition implies that each input vector is represented by the nearest codevector (the codevector at the shortest euclidean distance from the input). The centroid condition is that each codevector has to be the center of gravity of its domination region, defined as the partition of the input space represented by the codevector.

The GLA is the direct implementation of both these conditions: at each iteration, find the partition S_i of the training set X with the nearest neighbor condition (for each codevector $\mathbf{W}_i$, find its domination region (also called *Voronoi* region) $S_i = \{\mathbf{x} \in X \mid \|\mathbf{x} - \mathbf{W}_i\| \leq \|\mathbf{x} - \mathbf{W}_j\| \ \forall j \neq i\}$), then, apply the centroid condition to adjust codevectors ($\mathbf{W}_i \leftarrow E(S_i)$, with $E(S_i)$ the mean of S_i). This is a batch algorithm for which the quadratic distorsion $D(\mathbf{W}) = \sum_i E(\|\mathbf{x} - \mathbf{W}_i\|^2)\Big|_{x \in S_i}$ monotically decreases, thus often falling in local minimum since $D(\mathbf{W})$ is generally not convex [19].

3.2 Winner-take-all algorithms

The neural "on-line" equivalent of the GLA is generally refered to as Competitive Learning (CL). In this algorithm, each iteration consists of a presentation of an input vector to the network, which finds a "winner" neuron w whose weights vector $\mathbf{W}_w$ is the nearest of the input $\mathbf{x}$:

$$w \mid \ \|\mathbf{x} - \mathbf{W}_w\| \leq \|\mathbf{x} - \mathbf{W}_j\| \quad \forall j \neq w \tag{4}$$

then adapt the winner's weights towards the input ($\mathbf{W}_w \leftarrow \mathbf{W}_w + \alpha(\mathbf{x} - \mathbf{W}_w)$).

In the previous version of VQP network [5], the first layer was a slightly modified competitive learning VQ, where the winner was not found by the euclidean minimal distance, but using a maximal activity computed through gaussian kernels. This was not equivalent since each neuron had its own kernel radius, adapted in order to favorize units with low winning rate.

All these algorithms have poor convergence rates, especially for large numbers of units, because each input vector causes the adaptation of only one unit. This is the problem of "winner-take-all" algorithms. Therefore, it seemed desirable to implement an algorithm where more than one unit is adapted at each iteration. The problem is then to properly define a function $G(i, \mathbf{x}) > 0$ for each unit i, which will be used in the learning rule:

$$\mathbf{W}_i \leftarrow \mathbf{W}_i + \alpha_i G(i, \mathbf{x})(\mathbf{x} - \mathbf{W}_i) \quad \forall i \tag{5}$$

This function will be greater for the winner, leading to the concept of "winner-take-most" algorithms described in the following lines.

3.3 Winner-take-most algorithms

In "winner-take-most" algorithms, the winner is moved towards the input vector, but the other units are also moved, with a smaller step-size. However, it is easily understandable that adaptation computed as a continuous function of the distances may cause several units to merge together. In fact, two superposed vectors cannot be separated by such adaptation function, because for each input $\mathbf{x}$, both will be adapted with the same value. Thus, it appears necessary to *unlink* the adaptation factor from the euclidean distance. In competitive learning, this is simply done by setting

$$G(i, \mathbf{x}) = \begin{cases} 1 & \text{if } i = w \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

But, as said before, this is a winner-take-all algorithm, no longer a winner-take-most one.

Moreover, if $G(i, \mathbf{x})$ is defined positive and is only a function of individual distances $\|\mathbf{x} - \mathbf{W}_i\|$ (without a kind of competition or mutual inhibition), all units will merge together on the expectation value of the input, $E(\mathbf{x})$.

Let us observe how some algorithms adapting more than one unit per step unlink adaptation function $G(i, \mathbf{x})$ from the euclidean distance:

3.3.1 Kohonen algorithm

In Kohonen's self-organizing maps [8, 9, 10, 15, 14], once the euclidean winner has been found, the adaptation factor is computed in function of the lateral distance $d(i, w)$ along the grid, where w is the winner index:

$$G(i, \mathbf{x}) = \exp(-d^2(i, w)/\lambda^2) \tag{7}$$

(gaussian neighborhood) where λ is a neighborhood radius decreasing with the time, or:

$$G(i, \mathbf{x}) = \begin{cases} 1 & \text{if } d(i, w) \leq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

(rectangular neighborhood) where λ is also decreasing with the time.

Since in the map (unlike in VQP network) the neurons are fixed on a discrete lattice, two neurons with the same weights vector, though being at the same distance of any input $\mathbf{x}$, can be separated because of their adaptation factor differing through (7) or (8).

3.3.2 Stochastic Relaxation Scheme

A recent VQ algorithm is the Stochastic Relaxation Scheme (SRS), introduced by Yair *et al.* [19]. In this algorithm, all units are updated toward the input vector with a probability computed in order to comply with the Gibbs distribution:

$$P(i) = \frac{\exp\left(-\frac{\|\mathbf{x}-\mathbf{W}_i\|^2}{T}\right)}{\sum_{j=1}^{N}\exp\left(-\frac{\|\mathbf{x}-\mathbf{W}_j\|^2}{T}\right)} \tag{9}$$

where T is a parameter called "temperature", in reference to simulated annealing algorithms. For high temperatures, all units are adapted with about the same probability $\lim_{T\to\infty} P(i) = 1/N$. When the temperature $T \to 0$, $P(i) \to 1$ for the euclidean winner ($i = w$), while $P(i) \to 0 \quad \forall i \neq w$. That is, the algorithm evolves from a stochastic scheme to a simple Competitive Learning, while the temperature decreases.

The adaptation function is then:

$$G(i,\mathbf{x}) = \begin{cases} 1 & \text{with probability } P = P(i) \\ 0 & \text{else} \end{cases} \tag{10}$$

Under certain conditions that are discussed in [20], this scheme is able to avoid local distorsion minima, thus achieving globally optimal quantization. It is however noted to have low convergence rate, since the *transition rate* becomes smaller for low temperatures.

3.3.3 Soft Competition Scheme

In order to solve the problem of low transition rate, the same authors propose in [19] another algorithm, the *Soft Competition Scheme* (SCS), which is a deterministic version of the SRS. There, $G(i,\mathbf{x})$ is no longer a binary random value of probability $P(i)$, but actually takes the value of (9):

$$G(i,\mathbf{x}) = P(i) \tag{11}$$

where $P(i)$ is the expression (9).

Additionnaly, in order to favourize units that have not been moved for a long time, each unit has its own α_i computed as:

$$\alpha_i(n) = \left(1 + \sum_{k=1}^{n-1} G_k(i,\mathbf{x})\right)^{-1}, \tag{12}$$

where k represents all previous steps ($k = \{1,\ldots,n-1\}$), and $G_k(i)$ stands for $G(i,\mathbf{x})$ at iteration k.

For both algorithms (SRS and SCS), we find in (9) how distances and adaptation factors are unlinked: $P(i)$ depends on the selectivity of unit i, rather than on the only euclidean distance. The selectivity of unit i is expressed in term of all distances between input vector and units weights. This implements a kind of competition between units, where all units have approximatively the same value $P(i)$ for high temperatures, but which converges asymptotically to a simple winner-take-all competitive learning (6) when the temperature $T \to 0$.

Unfortunately, since in the SCS $G(i,\mathbf{x})$ is no longer stochastic but continuous with the euclidean distance, units merged together are impossible to separate, on the contrary of the SRS. In practice, it can be observed that, for certain temperature schedules, units do actually merge together and become quite difficult to separate.

Another drawback to the SCS is its slow rate of convergence (slower than the GLA). Moreover, the quality of the resulting quantization depends on the temperature schedule, which is deeply discussed in [19].

Under certain conditions for this temperature schedule, and with a fine tuned method of α_i reinitialization which aims to avoid "cristallization" in local minima, the SCS is able to achieve better quantization (about $1,5dB$ of SNR in mean) than the GLA, however after a longer time.

3.3.4 Neural-gas algorithm

Recently, Martinetz and Schulten [12, 18] have introduced a new VQ algorithm, called "neural-gas". In this algorithm, for each iteration, all units are sorted according to their distance to the input. A "rank of closeness" $k(i)$ is therefore attributed to each unit i, so that:

$$\delta_0 < \delta_1 < \ldots < \delta_k < \delta_{k+1} < \ldots < \delta_{N-1} \tag{13}$$

with:

$$\delta_k = \left\| \mathbf{x} - \mathbf{W}_{k(i)} \right\| \tag{14}$$

The first unit (of rank $k = 0$) is the euclidean winner, while the second unit (rank $k = 1$) comes just after, and so on. The adaptation factor is then computed as:

$$G(i) = \exp\left(-\frac{k(i)}{\lambda}\right) \tag{15}$$

where $k(i)$ is the rank of unit i, and λ a parameter decreasing with the time. The weights of the units are adapted according to (16):

$$\mathbf{W}_i \leftarrow \mathbf{W}_i + \alpha_i G(i, \mathbf{x})(\mathbf{x} - \mathbf{W}_i) \qquad \forall i \tag{16}$$

The algorithm could be seen as a kind of Kohonen Map with monodimensional lattice, where the topology is redefined at each iteration, with the winner at an extremity of the map. Here, the unlinking effect results from the fact that, for *ex æquo* units, ranking is chosen at random.

Parameters α and λ are computed through the same time schedule:

$$r = r_i \left(\frac{r_f}{r_i}\right)^{t/t_{max}} \tag{17}$$

where r_i is the initial value of the parameter, while r_f is the final one, that is, the one got at iteration $t = t_{max}$.

This algorithm is not only very interesting in its concept (because it looks like SOM). It also achieves very homogenous quantization with about the same speed (for relevant parameters α_i, α_f, λ_i, λ_f and t_{max}) than Kohonen Maps.

Unfortunately, it requires relatively large amount of time for the sort operation which has to be done at each iteration, and that may become prohibitive for large networks. Moreover, it does not prevent the previously explained problem of dead units, as it has been observed in simulations with non-convex distribution shapes (such as multimodal ones, for instance).

We will show how to overcome these problems in an optimized version of the neural-gas that we propose in section 3.4.

3.4 Our own VQ algorithm

For this VQ layer, a kind of neural-gas (see section 3.3.4) algorithm has been implemented, but with the following original improvements:

1. Not all of the units are sorted and adapted at each iteration, but only those for which the adaptation function $G(i)$ in (15) will be significant.

2. Dead units are avoided by implementing a kind of *neural fatigue* [4] or *conscience* [6] mechanism that leads units that are out of the input distribution to have a chance to be elected and therefore attracted within the distribution.

3. The time-constrained learning scheme (through the time-dependent parameters schedules (17)) is replaced by a continuous learning scheme, which self-stabilizes when the distribution is stationary, but is able to become highly adaptative again when the distribution is changed.

The result of these improvements is a very fast VQ algorithm, whose convergence depends only on the distribution stationarity, and which maps homogeneously the input distribution.

3.4.1 Limitation of the number of adapted units

Considering (15), one can notice that units of rank $k \gg \lambda$ have negligible adaptation function $G(i) \cong 0$. It is therefore possible to restrict the distances sort for only the K (say, for example, $K = 3\lambda + 1$) first values. In the first coarse adaptation steps, for which λ is large, it does not save much time, but for the next fine adaptation steps (that take in practice the largest part of the process), it avoids a lot of useless computations.

The implemented sort is a tree-sort restricted to the $K \leq N$ first distances. The restriction conditionally applies for each insertion of a new distance δ_k in the tree made while the distances of all units are computed. This kind of sort algorithms are well-known in oriented-graph theory and computer science.

Once sorted, the K selected units are updated according to (18):

$$\mathbf{W}_i \leftarrow \mathbf{W}_i + \alpha_i G(i, \mathbf{x})(\mathbf{x} - \mathbf{W}_i) \qquad \forall i \,|k(i) < K \tag{18}$$

3.4.2 Dead units avoiding

Following [4] or DeSieno [6], we give to never-elected neurons a chance to become eligible. Let us consider the expectation value of the adaptation function of unit i, estimated through a low-pass filter:

$$p_i \leftarrow p_i + \frac{1}{\tau}\left(G(i) - p_i\right) \tag{19}$$

where τ is the time constant for the estimation.

If, for each unit i, the rank $k(i) \in \{0, \ldots, N-1\}$ is equiprobable, then p_i should converge to a constant value $\frac{1}{N}\sum_{k=0}^{N-1} \exp(-k/\lambda)$. Units that are infrequently adapted will have low p_i, while too frequently winning units will have comparatively high values of p_i. In order to favourize fewly sollicited units, the sort operation should be made with respect to weighted distances, thus (13) and (14) should be replaced by:

$$q_0 < q_1 < \ldots < q_k < q_{k+1} < \ldots < q_{N-1} \tag{20}$$

with:

$$q_k = p_i \left\| \mathbf{x} - \mathbf{W}_{k(i)} \right\| \tag{21}$$

Election of units with low winning rate is therefore facilitated, and dead units are no longer observed.

The low-pass time constant τ was typically set to $\tau = 20N$ in our simulations.

3.4.3 Continuous learning scheme

The parameters α and λ computed through (17):

$$r = r_i \left(\frac{r_f}{r_i} \right)^{t/t_{max}} \tag{22}$$

do follow a exponential decreasing predetermined schedule, with initial value r_i at iteration $t = 0$, and final value r_f at iteration $t = t_{max}$. Since this schedule depends only on the time, there is no possibility, after t_{max} iterations, to "come back" to high parameters values, in order to make the network adaptable again when the input distribution changes.

The idea here is to keep the global form of (22), but replacing t/t_{max} by any convenient convergence measure evolving in the same range, that is, in $[0, \ldots, 1]$. Such a measure is obtained by considering for example the whole set of p_i as computed in (19). As previously said, a good mapping with equiprobable Voronoi regions should lead the p_i to have about the same expectation value. The ratio,

$$P = \frac{\min(p_i)}{\max(p_i)} \tag{23}$$

which remains low even if just one unit is not at its right place (for example, if one unit is left out of the distribution), is used as such a measure. For widely spread p_i, typical to bad quantization, P takes a small value. On the other hand, if all p_i are equals, then $P = 1$. Thus, the parameters law becomes:

$$r = r_i \left(\frac{r_f}{r_i} \right)^{P/P_0} \tag{24}$$

where P_0 is a value in the range $[0, \ldots, 1]$, which is the value of P for which the parameter takes its "final" value r_f. In practice, it is almost impossible to obtain a value $P = 1$ according to (23), since it always remains some fluctuations (though of small value) in the p_i's.

A convenient value is for example $P_0 = 0.7$, which is the choice adopted in our simulations.

3.4.4 Parameters summary

The described optimized neural-gas algorithm is robust in regard to the "initial" and "final" parameters values. Following is a set a parameters which have been found to give small convergence times with many input distribution shapes. Moreover, the transition towards a new distribution is surprisingly quick regarding to the good stability obtained once the distribution becomes stationary again:

Parameter designation	symbol	low value	high value	typical
Initial neighb. radius	λ_i	$N/3$	N	$N/2$
Final neighb. radius	λ_f	0.5	2	1
Initial gain	α_i	0.5	1	0.8
Final gain	α_f	0.001	0.05	0.01
Low-pass time constant	τ	$5N$	$200N$	$20N$
Selected units	K	1.5λ	N	$3\lambda + 1$
Expected asymptotic P value	P_0	0.4	1	0.7

4 Projection layer

The projection layer is the second part of the VQP network, which aims to find the output vector through interpolation. Each neuron has to find a relevant position in the output space, trying to copy the topology of the input one.

4.1 "dy, dx" representation

The underlying idea to adapt this layer is inspired by an organization measure we had already defined in [3] for Kohonen networks and we refer here to as "dy, dx" relation. It consists in the representation of the joint distribution of all input weights distances versus the corresponding distances in the ouput space (for Kohonen Maps, the grid indices distances dy along the lattice). This plot, practicable for whatever input and output dimensions, results in a cloud of points whose shape gives a lot of informations. For completely unorganized maps, as shown in fig. 3a or 3c, there is no correlation between input and output distances. This results because output distances (on horizontal axis) have not unique corresponding input distances (on vertical axis), giving a spread cloud of points. In contrast, perfectly regular maps give a "dy, dx" relation which is a straight line of slope a (where a is the weight distance between two consecutive units), as in fig. 3b. Finally, for folded maps (but with local conservation of topology), the relation begins approximatively with a line, but incurves for large grid distances, even coming back to small values for closed (circular or toroidal) maps, as in fig. 3d.

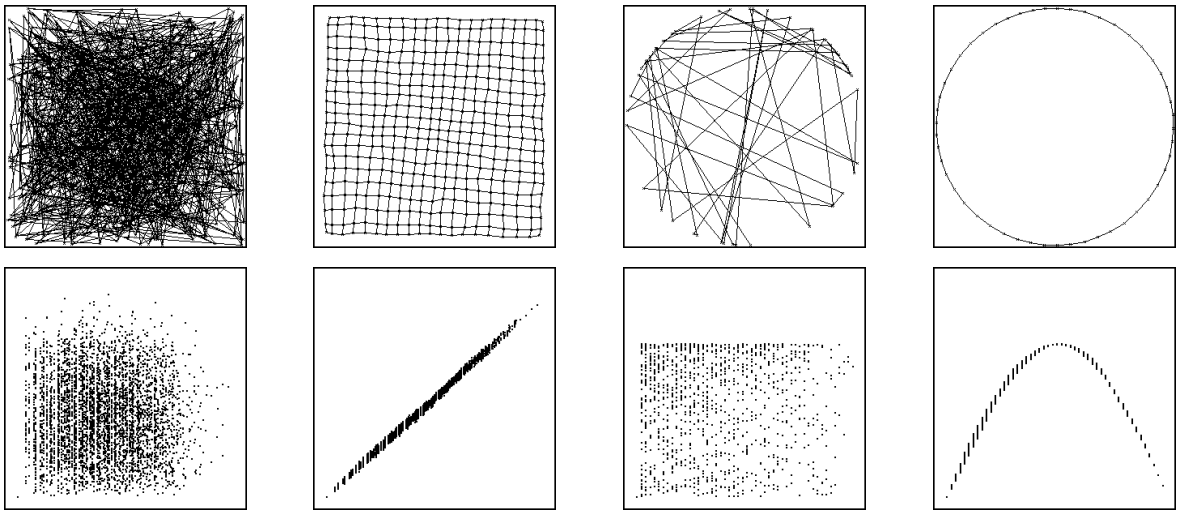


Figure 3: First row: usual weights representation. Second row: "dy, dx" representation. (a) Kohonen map at initialization. (b) After 10000 iterations with rectangular uniform distribution. (c) Kohonen monodimensional map with normalized weights at initialization. (d) The same, organized on a circular distribution.

In practice, this plot does not take into account all the N^2 possible points, which would become heavy for large number N of neurons: only some thousands of points are drawn by selecting randomly couples of neurons.

The use of this relation in VQP, where neurons have a floating position in the output space, is straightforward: the horizontal values dy no longer refer to grid distances, but to the output weight distances between units, while the values dx remain the input weight distances between units.

4.2 Adaptation rules

The goal of VQP is to obtain a "dy, dx" relation which is a thin line, therefore the projection is topologically correct, at least locally. In fact, this task is impossible if the wanted projection need to be non-linear (which is the case when trying to reveal non-linear dependances in the input space). Then, we relax this constraint to be effective not for the whole plot, but only for a restricted range of distances in ouput space. This is done by using a weighting factor $H(dy) = (1 + (dy/dy_0)^q)^{-1}$ that favourizes the adaptation for small output distances, where dy_0 is an output distance cut off and q a strength parameter. Fig. 4 shows $H(dy)$ for a given dy_0 and some exponents q. In the following, "dy, dx" relations will also show in superimposition this weighting factor, whose parameters have been set "by hand" looking at the "dy, dx" distribution for each case.

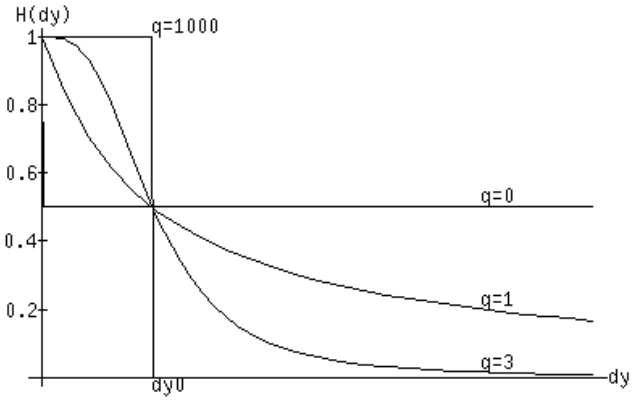


Figure 4: $H(dy)$ for some values of q.

In the previous version of VQP, only two units, the ouput weights of the winner at current iteration ($w(t)$) and of the previous one ($w(t-1)$), were moved closer or farther according to the ratio (25):

$$\beta = \frac{dy - dx}{dx + dy} H(dy) \tag{25}$$

and

$$H(dy) = \frac{1}{1 + (dy/dy_0)^q} \tag{26}$$

where dx is the input weight distance between the winning neurons $w(t)$ and $w(t-1)$, while dy is the corresponding output weight distance.

When the distance between output weights of neurons $w(t-1)$ and $w(t)$ is greater than the correponding distance in the input weights space, β is positive and thus the two neurons output weights are brought closer. On the other hand, when they are too close in the ouput weights space, they are pulled far away.

It has been found out that this scheme was not very efficient for large number of units, for the following reasons: only two units were adapted at each iteration, and moreover, as the

winner index at time t is assumed to be independent from those at time $t-1$, they have very low chance to be neighbour in the output space, and therefore the weighting parameter $H(dy)$ in (25) was small in mean. In other words, we had low chance to pick an interesting couple of neurons.

We generalize here the projection layer rule in a way that implies all units at each iterations:

$$\mathbf{W}_{out,i} \leftarrow \mathbf{W}_{out,i} + \alpha_{out}\beta_{i,w}\left(\mathbf{W}_{out,w} - \mathbf{W}_{out,i}\right) \quad \forall i \tag{27}$$

where α_{out} is an output adaptation factor set to 0.4 ± 0.1 in all our simulations.

$$\beta_{i,w} = \frac{dy_{i,w} - dx_{i,w}}{dx_{i,w} + dy_{i,w}} H(dy_{i,w}) \tag{28}$$

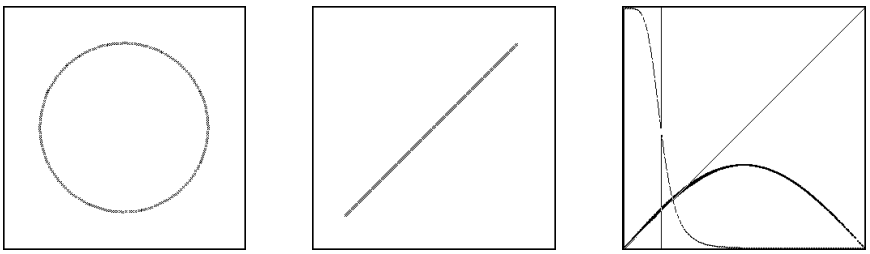
where $dx_{i,w}$ is the input weight distance between neurons i and the winner w, while $dy_{i,w}$ is the corresponding output distance. In practice, an infinitesimal value is added to the denominator $dx_{i,w} + dy_{i,w}$ in order to fix

$$\lim_{dx_{i,w}, dy_{i,w} \to 0} \beta_{i,w} = 0$$

as for example with $i = w$.

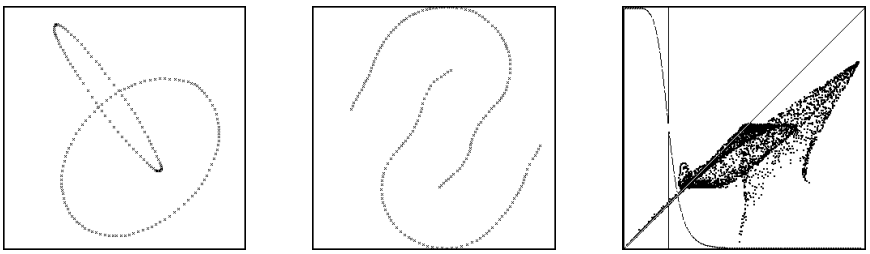
When the neuron i is too distant from the winner neuron w in the output weight space, regarding to the input one, $\beta_{i,w}$ is positive and thus i is brought closer to w through (27). On the other hand, when the neuron i is too close to w, it is pulled far away.

The application of this rule for each neuron at each iteration leads the ouput layer to copy as much as possible the topology of the input one in a much more efficient way than in the previous presentation of VQP algorithm [5]. The speed up is estimated to be of order $O(N^p)$, where N is the number of neurons and p is guessed to be ≥ 3 in practice.



(a) *(b)* *(c)*

Figure 5: Projection of a circular distribution onto one dimension (see text).

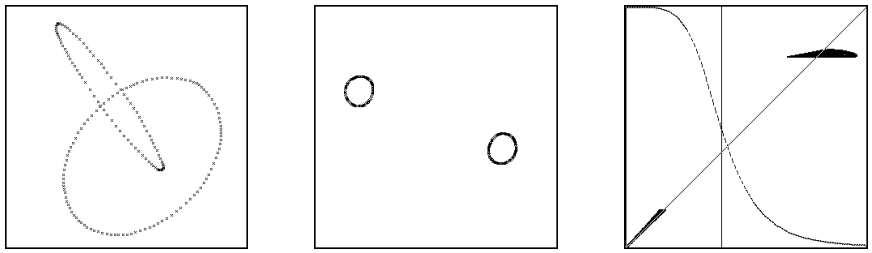


(a) *(b)* *(c)*

Figure 6: Separation of two interleaved rings in three dimensions, onto an output projection of 2 dimensions.

Some examples are given here (figs 5, 6 and 7), with for each one: a) the input weights location, b) the corresponding output weights, and c) the "dy, dx" representation. In fig. 2, the

input distribution was the text "`Hello world`" written onto a $3D$ helicoid, while the output space was fixed to be in two dimensions. In fig. 5, the input was the $2D$ circle of radius 1, while the ouput was fixed to one dimension. Here, the network has cut the circle at an arbitrary place (between two neurons), while the rest of the topology has been conserved, giving the typical "dy, dx" plot of fig. 5 that reminds the one of fig. 3c. Finally, an example with two interleaved rings is given in figs. 6 and 7. In fig. 6, where the input distribution was in three dimensions, each ring in the projection has been broken at the center of the other ring, giving a local respect of topology everywhere, excepted for these two broken points. In fig. 7, we have added supervision by giving a different class label for each ring in a fourth dimension. Therefore, VQP learned to separate the two classes, giving two clusters that correctly represent the rings, and a "dy, dx" representation that indicates the multimodality of the distribution (the two clouds in fig. 7c).



(a) *(b)* *(c)*

Figure 7: Separation of two interleaved rings in four dimensions (the fourth being a class label), onto an output projection of 2 dimensions (see text).

5 Conclusion

The new self-organizing neural network described here, the Vector Quantization and Projection neural network (VQP), is inspired from the well-known Kohonen Maps. However, unlike them, no *a priori* defined structure, such as in a lattice or grid of neurons, is fixed here: the neurons find themselves their position in a continuous output space, trying to copy as much as possible the topology of the input one. It is therefore a method able to represent high dimensional data structures onto lower dimensional spaces, which does not suffer from the drawbacks and limitations usually belonging to the classical techniques (such as Principal Components Analysis), like the restriction to linear projection or prohibitive time wasting.

The first presentation of VQP [5] was only focused on the new concept, and involved simple winner-take-all scheme for both Vector Quantization and Projection layers. A review of several known VQ algorithms and a discussion on the Projection learning part are given in this paper. The optimized version of the algorithm presented here involves the use of an optimized version of the Martinetz and Schulten "neural-gas" algorithm, where the problem of "dead units" is eliminated and which has been transformed to a continuously learning algorithm with very fast learning abilities.

Aknowledgements. This work is partially granted by ELENA ESPRIT Project and by a French ANVAR funding.

References

[1] F. Blayo and P. Demartines. Data analysis: How to compare Kohonen neural networks to other techniques ? In A. Prieto, editor, *International Workshop on Artificial Neural Networks*, volume 540 of *Lecture Notes in Computer Science*, pages 469–476. Springer-Verlag, 1991.

[2] H. Caussinus. Un modèle pour fonder la recherche de projections révélatrices. In *XXIVes Journées de Statistiques*, pages 120–123, 1992.

[3] P. Demartines. Organization measures and representations of Kohonen maps. In J. Hérault, editor, *First IFIP Working Group 10.6 Workshop*, 1992.

[4] P. Demartines and F. Blayo. Kohonen self-organizing maps: Is the normalization necessary ? *Complex Systems*, 6(2):105–123, 1992.

[5] P. Demartines and J. Hérault. Vector quantization and projection neural network. In A. Prieto J. Mira, J. Cabestany, editor, *International Workshop on Artificial Neural Networks*, volume 686 of *Lecture Notes in Computer Science*, pages 328–333. Springer-Verlag, 1993.

[6] D. DeSieno. Adding a conscience to competitive learning. *IEEE International Conference on Neural Networks*, !:117–124, 1988.

[7] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*, volume 1 of *Santa Fe Institute Lecture Notes*. Addison-Wesley Publishing Company, 1991.

[8] T. Kohonen. Self-organization of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[9] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3rd edition, 1989.

[10] T. Kohonen. The self-organizing maps. *Proc. of the IEEE*, 78(9):1464–1480, 1990.

[11] Y. Linde, A. Buzzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Commun.*, COM-28:84–95, 1980.

[12] T. Martinetz and K. Schulten. A neural gas network learns topologies. In T. Kohonen *et al.*, editor, *IEEE International Conference on Artificial Neural Networks, Espoo, Finland*, volume 1, pages 397–407. Elsevier, 1991.

[13] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

[14] H. Ritter, T. Martinetz, and K. Schulten. *Neural computation and self-organizing maps: an introduction*. Addison-Wesley Publishing Company, 1992.

[15] H. Ritter and K. Schulten. On the stationary state of Kohonen self-organizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.

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

[17] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.

[18] J. A. Walter and K. J. Schulten. Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transaction on Neural Networks*, 4(1):86–95, 1993.

[19] E. Yair, K. Zeger, and A. Gersho. Competitive learning and soft competition for vector quantizer design. *IEEE Transactions on Signal Processing*, 40(2):294–309, 1992.

[20] K. Zeger, J. Vaisey, and A. Gersho. Globally optimal vector quantizer design by stochastic relaxation. *IEEE Transactions on Signal Processing*, 40(2):310–322, 1992.