A Novel Neural Approach to Inverse Problems with Discontinuities (the GMR Neural Network)

Giansalvo Cirrincione
University of Picardie-Jules Verne
33, rue Saint Leu, 80039 Amiens - France
exin@u-picardie.fr

Maurizio Cirrincione
I.S.S.I.A.-C.N.R. Section of Palermo (former CE.RI.S.E.P.)
Viale delle Scienze snc, 90128 Palermo - Italy
nimzo@cerisep.pa.cnr.it

Chuan Lu
Katholieke Universiteit Leuven, Afd. ESAT SCD (SISTA)
Kasteelpark Arenberg 10, B-3001 Heverlee,Leuven-Belgium
chuan.lu@esat.kuleuven.ac.be

Sabine Van Huffel
Katholieke Universiteit Leuven, Afd. ESAT SCD (SISTA)
Kasteelpark Arenberg 10, B-3001 Heverlee,Leuven-Belgium
Sabine.VanHuffel@esat.kuleuven.ac.be

***Abstract*— The Generalized Mapping Regressor (GMR) Neural Network is able to solve for inverse problems even when multiple solutions are given. In this case, it does not only identify these solutions (even if infinite, e.g. contours), but also specifies to which branch of the underlying mapping it belongs. It is also able to model mapping with discontinuities. The basic idea is the transformation of the mapping problem in a pattern recognition problem in a higher dimensional space (where the function branches are represented by clusters). Training is given by a multiresolution quantization represented by a pool of neurons whose number is determined by the training set. Then, neurons are linked each other by using some kind of Local Principal Component Analysis (LPCA). This phase is the most important and original. Other techniques (e.g. SVM's, mixture-of-experts) could work a priori on the same problems, but are not able to understand automatically when to stop the data quantization. This linking phase can be viewed as a reconstruction phase in which the correct clusters are recovered. The production phase uses a Gaussian kernel interpolation technique. Some examples conclude the paper.**

Keywords—Discontinuity, Incremental, Inverse Problem, Linking, LPCA, Pattern Recognition, Support Vector Clustering

I. INTRODUCTION

This paper deals with the problem of mapping approximation by means of neural networks. In [1] it is proved that a single hidden layer feedforward neural network is able of approximating uniformly any continuous multivariate function to any degree of accuracy, provided a sufficient number of hidden units is given. In [2] it is demonstrated that neural networks with two hidden layers and threshold units can approximate the one-sided inverse of a continuous function. The inversion problem can also be solved by the symmetric counterpropagation [3]. In general, inverse problems cannot be solved efficiently by these techniques. W.r.t. the approximation of discontinuous functions, it is possible to use the mixture-of-experts (ME) which uses a divide-and-conquer strategy, i.e. it adaptively partitions the input space into overlapping regions and allocate different networks to summarize the data located in different regions [4], [5]. The Generalized Mapping Regressor (GMR, [6], [7]) is a neural network able to approximate every mapping also with every kind of discontinuity, and, simultaneously, its inverse, if it exists, or the inverse relation. It also outputs all the solutions (even infinite), their corresponding branches and, if the case, the equilevel surfaces This paper improves GMR by introducing the Local Principal Component Analysis (LPCA) and the Gaussian kernel interpolation in the production phase.

Research of C.L. and S.V.H. supported by FWO: G.0407.02, KU Leuven: GOA-Mefisto 666 and DWTC:IUAP V-22

II. THE ALGORITHM

GMR is mainly an incremental self-organizing neural network. Its architecture is sketched in fig. 1. Its algorithm transforms the mapping problem $f : x \rightarrow y$ into a pattern recognition problem in the augmented space Z represented by vectors $z = [x^T\ y^T]^T$, where T is the transpose operator, which are the inputs of GMR. In this space, the branches of the mapping become clusters which have to be identified. The weights of the first layer are continuous and represent the Z space, while the other ones are discrete (chains between neurons) and represent the mapping branches mapping. The first layer weights are computed (training phase) by a multiresolution quantization phase, the second layer weights are computed (linking phase) by a PCA technique, unlike the original algorithm [6], [7] which used geometrical criteria.

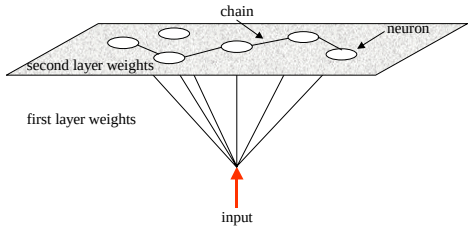


Fig. 1: GMR architecture

A. *Training phase (multiresolution vector quantization)*

The training phase concerns the vector quantization of the *Z* space. This can be obtained by several neural approaches (e.g. SOM, ME, SVC [8]). In the following examples, in order to speed up the computations, the EXIN SNN learning law ([6],[7],[9]) has been implemented, which is unsupervised and incremental. At the presentation of each input of the training set (TS), there are two possibilities: either creation of a new neuron (whose weight vector is equal to the input vector) or adaptation of the weight vector of the closest neuron (in input/weight space). Given a threshold ρ (vigilance threshold), a new neuron is created if the hyperspheres of radius ρ, centred in the already created weight vectors, do not contain the input (and so is unable to represent the input). In order to speed up computations, the weight adaptation is a simple linear combination of the input and the weight (stochastic ML Gaussian mean estimate). The parameter ρ is very important: it determines the resolution of training. Here, after each epoch (presentation of all the TS), it is decreased: in general, two epochs are enough. Pruning strategies have also been used in order to decrease the final number of neurons. The drawback of this approach is the curse of dimensionality: indeed, it simply recovers the input spaces. For higher dimensional spaces, space reduction techniques for data preprocessing are in study. Learning can be divided into two subphases: coarse quantization and fine quantization . The coarse quantization is obtained by using EXIN SNN with a high value for ρ, say ρ_1. Here more than one epoch are accepted. In general, the first epoch defines the number of neurons needed for mapping and the others adapt their weights for a better approximation. The so obtained neurons identify the *objects*, which are compact sets of data in *Z*. The number of epochs can be predefined or determined by a stop criterion which monitors the weight increments and stops if they fall under a certain threshold. The resulting neurons are called *object neurons*. In the second subphase, at first, a preprocessing is required for labelling each neuron with the list of the input data which had the neuron as winner; it can be accomplished by presenting all data to GMR and recording the corresponding winning neurons. At the end, for each neuron a list of the inputs for which it has won is stored. This list represents the *domain* of the object neuron. Every list is considered as the TS for a subsequent secondary *EXIN SNN*. Hence, as many EXIN SNN's as the object neurons are used in parallel in order to quantize each object domain. The secondary learning can have an a priori fixed number of epochs or a stop criterion equal to the one cited above. These intradomain learnings need a threshold lower than ρ_1, say ρ_2, whose value is determined by the required resolution. At the end, the neural network is composed of the neurons generated by the secondary learning phases (*pool of neurons*), labelled as belonging to an object by the corresponding object neuron, which however is not included in the pool. These labelled neurons represent a discrete Voronoi tessellation of the input *Z* space. Note that the proposed quantization technique requires predetermined thresholds which can be easily made adaptive by exploiting the input information. Resuming, the augmented *Z* space is quantized by means of a coarse subphase, a labelling processing requiring a production phase (presentation of the TS to the network) and a fine subphase requiring a parallel implementation. This multiresolution quantization can also be obtained by using other neural networks: what is important is to obtain a pool of neurons and compute its weights w.r.t. to the input (first layer weights, see fig. 1).

B. *Domain setting*

For each neuron, its domain is determined by the data (in *Z* space) of the TS for which it wins (nearest in weight/input space). These domains are recovered by an additional production phase. For each domain (see fig. 2, where the neuron is represented by its weight in *Z* space), the domain radius is computed as the distance between the neuron weight and the farthest data from it. The corresponding hypersphere centred on the weight badly represents the domain, because it does not take into account the directionality of the data (this radius will only be used in the next phases in order to take in account the resolution of the fine quantization).

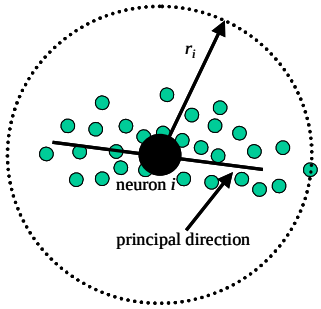


Fig.2 Neuron domain parameters

To overcome this difficulty, by applying PCA to the data of the domain (local PCA in the sense that it is performed for each domain), the principal direction (PD) of data can be computed, which roughly captures the directionality of the domain. PCA is here performed by using PCA EXIN [6], which is an iterative algorithm which outputs the PD. This approach is reminiscent of LPCA [10], but differs from them because the partitioning of data is performed by a neural network and no global criteria are considered. If a domain is composed of only one neuron, the PD is replaced by a tag (impossibility of computing the PD).

C. *Linking and object merging phases*

The training phase is not enough in order to correctly cluster the function branches: all quantization techniques are basically empirical and don't know when to stop clustering. In the limit as many clusters as data can be found. The linking phase is introduced in order to reconstruct the function branches, i.e. linking together the clusters belonging to the same branch. This idea is original and can be applied, more in general, to all the clustering techniques. Basically, this approach tries to *track the shape* of the function branch through its domains. Linking neurons means computing the second layer weights (see fig.1), which are discrete and equal to zero in absence of a link. A link is computed at the presentation of each TS data. Originally [6], [7], this computation was based on the requirement that connected neurons have to approach the direction (called *linking direction*) of the vector connecting the input vector to the weight of the winner neuron (neuron of the domain to which the input belongs) in the Z space. Hence, this linking carried a directional (orientation) information driven by the input data, in order to determine both the position and the shape of the mapping branch (cluster). However, because the TS is noisy, the linking direction may not represent well the branch shape because the input data can be placed everywhere around the neuron. However, this technique is very quick and yields a good performance for small training sets with low noise. In this paper, a novel technique, which exploits the domain PD's, is employed. It is justified by the consideration that the PD's better represent the mapping branches.

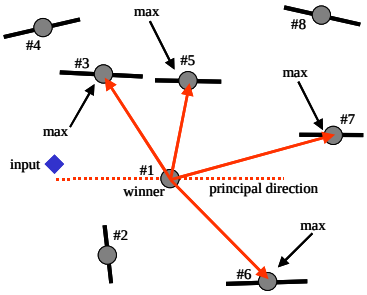


Fig.3: Linking in presence of equivalent maxima

Linking is achieved by one complete presentation of the TS. For each data, the weights are sorted according to the Euclidean distance from the data (see fig. 3). This is the most time consuming part of the algorithm and is here accelerated by means of a branch and bound strategy [11] which saves computations by partitioning the search space (here the partitioning is found by applying the same EXIN SNN's used for training). Hence, the winning (nearest) neuron is determined. This is then linked to another neuron chosen in a subset of the neuron pool (*candidate neurons*). Two criteria have been implemented and compared for determining the subset: the first (δ-BnB) considers only the neurons included in a hypersphere centred in the data and of radius a multiple (a priori defined by a *linking factor*) of the distance between the input and the winner weight vector; the second (k-BnB) considers only the k nearest neighbours of the input. In the latter criterion the value of k has to be defined in advance. Probably it is influenced by the cardinality of the neuron pool, which is a function of the value of the vigilance threshold in the end of the training phase (final resolution). Then, for each candidate, the absolute value of the scalar product between its PD and the winner's PD is evaluated: the winner is linked to the candidate yielding the maximum scalar product (i.e., whose PD is closest in direction to the winner's PD). Linking means increasing of one the corresponding second layer weight (if there were no link, the weight is set to one). However, this kind of linking is not flexible w.r.t. noise. Indeed, if two neurons were linked because they are close and in the same branch, but their PD's slightly differed because of noise, it would be possible that elsewhere another neuron has the same PD as the winning neuron and so it would be incorrectly linked. In this case it is important to define a threshold l which has to be a fraction of the final training resolution, in order to define the *equivalent maxima* (see fig. 3), i.e. neurons whose PD's differ from the winner PD only of $\pm l$. Then the following geometrical criterion is used in order to decide the equivalent maximum which has to be linked to the winning neuron. For each equivalent maximum, compute the absolute scalar product between the winner PD's unit vector and the directed vector from the winner weight to the candidate weight. The neuron yielding the maximum of this quantity is linked (in fig. 3 the winner is linked to the neuron #7). This criterion is justified by the fact that linking results in a *branch tracking* and this branch is in general smooth, so it has to be tracked in a small spatial window, as required by this geometrical approach. If the winning neuron has a tag (instead of the PC direction), the original criterion (i.e. the linking direction) is used, that is, the neuron whose direct vector from the winner weight is closest to the linking direction is chosen for linking. The object merging phase checks if there are links between neurons belonging to different objects. If the second layer weight of the link is positive, the two objects are merged into a unique object and the associated neurons are re-labelled.

D. *Recall phase*

In the recall phase, the input (from now on called x) can be any collection of components of z (the input space is defined as X). Hence the output y is the vector composed of the other elements of z (the output space is defined as Y). All weight vectors are also projected onto X . This projection is easily accomplished by using only the elements of the weight vector whose position indices correspond to the position indices of the input elements in the augmented vector Z. For example, if the first three elements of the vector z are taken as input, the projected weight vectors are composed of only the first three elements of the weight vectors. Unlike the original method, in X space, each neuron is replaced with a Gaussian which represents the neuron domain. Its parameters (mean, covariance) are given by the ML

estimates (the sample mean and sample unbiased covariance). When an input vector x is fed to GMR, the Gaussians are sorted in decreasing order according to their value in x (the value of the Gaussian in x is here considered as a metric). Following this order, each Gaussian is labelled as *level one* if the hypersphere, centred in the mean and whose radius is the domain radius , contains the input (level one test) and *level two* if it is not level one, but is directly linked to a level one Gaussian. This labelling is controlled by the following stop criterion: if the Gaussian is neither classified as level one nor as level two, then the labelling is stopped. All level one Gaussians and level two neurons which are connected each other (even not directly) are considered as belonging to the same mapping branch. Then, for each Gaussian k the complement of the weight of the corresponding domain is defined as t_k. The outputs are associated to the level one Gaussians. For each of these Gaussians, say the i-th, the interpolation phase considers the two Gaussians (either level one or level two) directly linked to it. Call them p_{i-1} and p_{i+1}. The associated output y_i is given by the following kernel interpolation formula:

$$y_i = \frac{t_{i-1} p_{i-1}(x) + t_i p_i(x) + t_{i-1} p_{i+1}(x)}{p_{i-1}(x) + p_i(x) + p_{i+1}(x)} \qquad (1)$$

If one of the two Gaussians does not exist or is neither level one nor level two, its value is set to zero. If the i-th Gaussian has no links, the interpolation is given by:

$$y_i = t_i p_i(x) \qquad (2)$$

No interpolation is required if the value of the i-th Gaussian in x is nearly one (w.r.t. the training resolution). In the end, as a consequence of the interpolation, each level one Gaussian yields an output y. Two different outputs belong to the same branch if they correspond to Gaussians belonging to the same branch. All level one Gaussians and the branches or portions of branches containing only level one Gaussians constitute a discretization of the equilevel hypersurfaces. It is possible to have disjoint equilevel hypersurfaces (branches) for the same object, e.g. the equilevel curves of a saddle in the 3d space for certain section orientations.

III. Simulations for Mapping Approximation

The following examples prove the mapping capabilities of GMR for low dimensional Z spaces. All experiments use a coarse and a fine quantization ($\rho = 0.5$ and 0.2, respectively). The data are noisy and represented in the figures by circles in the input space. The neuron weights after training are represented by crosses, the linking by thin lines and the PD's by bold lines. The Voronoi tessellation of the input space is also shown.

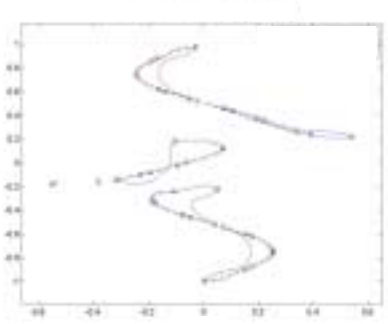


Fig. 4: sent SV's)

The first example deals with the mapping of the following function:

$$x = \frac{1}{4}\left[\sin(2\pi y) + \frac{0.01}{y^2 - 0.04}\right] \qquad (3)$$

The plot in fig. 4 shows the result of the Support Vector Clustering (SVC, [8]) for the best choice of its parameters (see [8]). As for all the other clustering techniques it is not obvious when to stop clustering (here two branches are split into two clusters). The same is true for EXIN SNN. The first two plots in fig. 5 show the results of δ-BnB for the original and the novel linking approach, respectively. Because of the low number of data, some spurious link appears between two branches (lower horizontal asymptote). For the original linking, thirteen objects are found, but, after merging, only three branches are correctly identified (third plot in fig. 5). Fig. 5 also shows the three outputs (diamonds) for $x = 0.2$ (yielded together with the information that two of them belong to the same branch). The level one and two Gaussian means are represented by squares and circles, respectively.

The second example deals with the mapping of the Bernoulli's lemniscate:

$$\left(x^2 + y^2\right)^2 = x^2 - y^2 \qquad (4)$$

and the results are shown in fig. 6. The top plot shows the results obtained by the multi-layer perceptron trained by the backpropagation algorithm (MLP) and with Gaussian mixture outputs (mixture density networks, MDN), which is a generalization [12] of ME [4], [5]. The results don't represent the function at all. Instead, GMR (see the other four plots, which have a different scaling than the top plot) works well. Notice that the novel linking approach works better (see around the origin of axes) than in the previous example. Here k-BnB is used, with $k = 3$. The objects found in the training phase are differently represented. After merging, only one branch is identified. Hence, the outputs are correctly considered as belonging to the same branch.

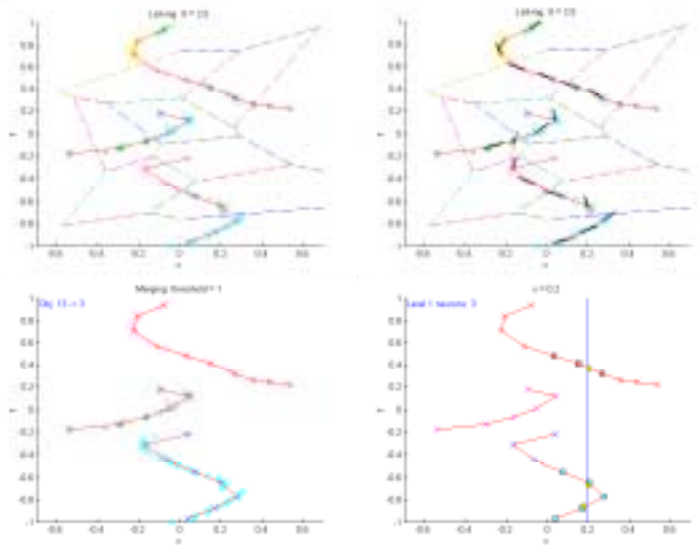


Fig. 5: Example of a multivalued mapping with discontinuities (TS: *98* data, noise std : *0.15*)

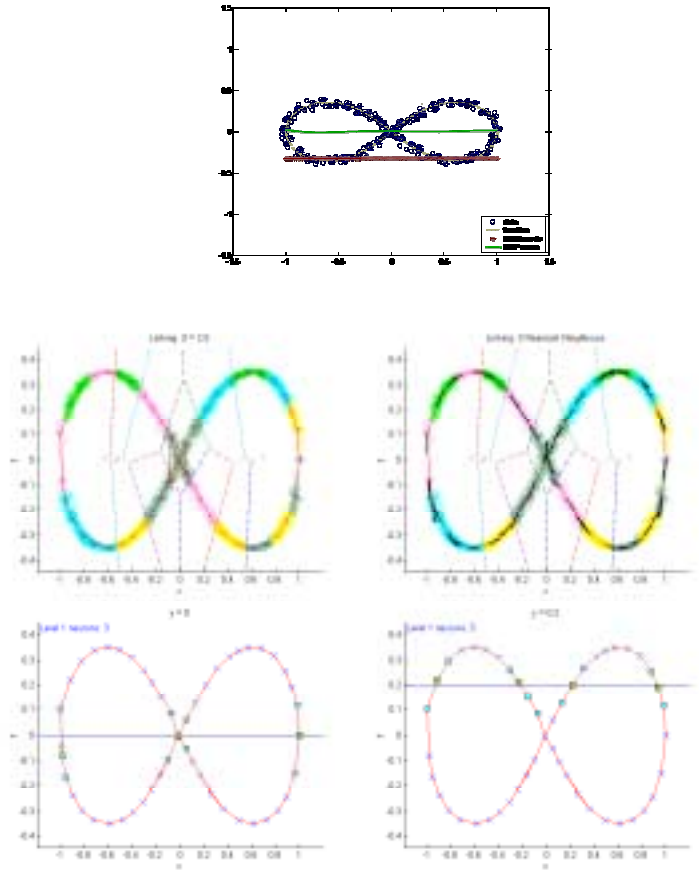


Fig.6: The Bernoulli's lemniscate (TS: *2000* data, noise std : *0.15*) : (top) MLP and MDN; (other plots) GMR (the scaling of the plots is different)

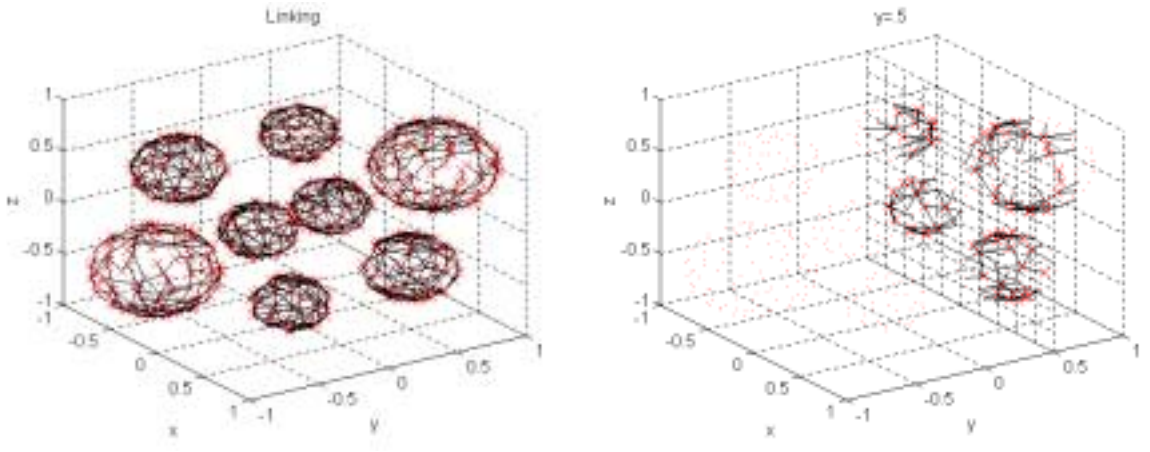


Fig 7. (left) 3d example: data set, neurons and links; (right) Level 1 neurons (crosses) and direct links (only between level 1 neurons) which represent the output discretized contour for *y=0.5* (the links towards the level 2 neurons are also shown).

The third example deals with a 3D mapping given by eight spheres in different positions. The TS is composed of *4000* points selected randomly on the spheres and *2000* points selected randomly outside the spheres (fig. 7 left). *15* neurons have been used for the coarse quantization. After the merging phase, eight objects are correctly identified. This emerges in fig. 7 left which shows the neuron weights and the linking representation. Fig. 76 right shows the level one neurons together with the associated links towards the level one (contour) and level two neurons for a certain value of *y*. In this case, no interpolation has been performed (original GMR algorithm). Notice the discretized contour output by GMR.

IV. Conclusion

Two novel ideas have been established here: the transformation of a function approximation problem into a pattern recognition problem, which allows to find multiple solutions, and the branch reconstruction, which models the branches by tracking the shape of the component clusters. The latter allows to model the discontinuities. As a consequence, a novel neural network, which is incremental, self-organizing and which uses adaptive chains (linking) among neurons has been presented. GMR is an universal approximator which is also able to output infinite solutions by using the links. The originality and force of GMR originates from its open architecture, in the sense that different neural modules can be used for the same task. Indeed, GMR is, above all, a strategy which exploits these two novel ideas. For instance, other clustering techniques, like SVC or ME, can replace EXIN SNN in the training phase. EXIN SNN is a very fast algorithm. In the first example presented before, GMR was eight times faster than SVC. This work, in particular, proposes a novel technique based on PCA for linking, in order to track the mapping branches. The recall phase also uses the Gaussian interpolation. Examples are given for low dimensional Z spaces. For higher dimensional spaces, future work will deal both with the use of nonparametric kernel techniques for training in order to have a smooth continuous approximation of the mapping and with projection techniques for the GMR preprocessing.

REFERENCES

[1] K. Hornik, M. Stinchcombe and H. White, “Multilayer feeforward networks are universal approximators”, Neural Networks, vol.2, n°5, pp. 359-366, 1989.

[2] E.D. Sontag, “Feedback stabilization using two-hidden-layer nets”, IEEE Transactions on Neural Networks, vol.3, n°6, pp.981-990, 1992.

[3] R. Hecht-Nielsen, “Neurocomputing”, Addison-Wesley, 1990.

[4] M.I. Jordan, S.J. Nowlan and G.E. Hinton, “Adaptive mixtures of local experts”, Neural Computation, 3, pp. 79-87, 1991.

[5] M.I. Jordan and R.A. Jacobs, “Hierarchical mixtures-of-experts and the EM algorithm”, Neural Computation, 6, pp. 181-214, 1994.

[6] G. Cirrincione, “Neural structure from motion”, Ph.D. Thesis, LIS INPG, Grenoble (France), 1998.

[7] G. Cirrincione, M. Cirrincione and S. Van Huffel, “Mapping approximation by the GMR neural network”, CSCC 2000, Vouliagmeni (Greece), July 2000, pp. 1811-1818.

[8] A. Ben-Hur, D. Horn, H. Siegelmann and V. Vapnik, “Support Vector Clustering”, Journal of Machine Learning Research 2, pp. 125-137, 2001.

[9] G. Cirrincione and M. Cirrincione, “A Novel Self-Organizing Neural Network for Motion Segmentation”, Applied Intelligence 18 (1), pp. 27-35, January 2003.

[10] P. Meinicke and H. Ritter, “Local PCA learning with resolution-dependent mixtures of Gaussians”, ICANN 99, vol.1 Transactions Industry Applications, vol.1, pp. 497-502, 1999.

[11] K. Fukunaga and P. Narendra, “A branch and bound algorithm for computing k-nearest neighbors”, IEEE Transactions on Computers 24(7), pp. 750-753 ,1975.

[12] C. M. Bishop, “Neural networks for pattern recognition”, Oxford University Press, 1995