Clustering with Multilayer Perceptrons and Self-Organized (Hebbian) Learning

Jugurta R. Montalvão Filho, Eduardo O. Freire, and Murilo A. Bezerra Jr.*

Abstract

A new local (Hebbian) learning algorithm for artificial neurons is presented. It is shown that, in spite of its implementation simplicity, this new algorithm, applied to neurons with sigmoidal activation function, performs data clustering by finding valleys of the probability density function (PDF) of the multivariate random variables that model incoming data. Some interesting features of this new algorithm are illustrated by some experiments based on both artificial data and real world data.

Keywords — Hierarchical clustering, Hebbian learning, Deterministic Annealing, PDF valleys.

1 Introduction

Clustering is a self organized process that plays an important role in a wide range of fields, ranging from typical applications, as Pattern Recognition and Signal Compression [3], and Knowledge Discovery in Databases (KDD) [7], to less common ones such as communication channel estimation and/or equalization [8, 2].

In cluster analysis concerned literature, we easily identify two main issues: (a) clustering data into a previously given number of clusters — most popular algorithms for clustering are based on both empirical geometric criterion (e.g., K-mean and vector quantization (VQ)) and probabilistic models, (e.g., Expectation-Maximization (EM))—, and (b) estimating the number of clusters — though there are no completely satisfactory methods for this task [6], among the most performing algorithms, we can find the Duda and Hart‘s test statistic $J_e(2)/J_e(1)$ [3], and the Cubic Clustering Criterion (CCC) [10].

Most popular strategies in both issues are based on clustering multidimensional samples (data) around prototypes, cluster centers, where the association of a sample to its cluster is based on predefined (e.g. Euclidean) distance measure from the cluster center. Accordingly, for estimation of the number of clusters, one rule of thumb is to increase it and to choose a configuration for which the averaged within cluster dispersion decreases more abruptly.

Roughly speaking, classical clustering approaches present some well-known drawbacks, such as sensibility to prototypes initialization and inadequacy when cluster shapes are far from ellipsoidal one, while finding the number of clusters remains a challenging open problem.

It is worth noting that, on the other hand, Multi-Layer Perceptron (MLP) with sigmoidal activation functions became popular thanks to its capabilities to circumvent similar problems in classification context. That is to say that MLP can be quite robust to parameters initialization as well as, in classification tasks, they are suitable to deal with highly non-ellipsoidal class dispersions. However, in spite of its helpful characteristics, MLPs with sigmoidal activation functions are most of the time applied to classification and multi-dimensional regression tasks, and not to clustering tasks.

In this paper, we present a new clustering algorithm, based on MLP with sigmoidal activation function, where self-organized neurons are adapted by a local (Hebbian) rule, derived from a probabilistic approach. Thus, each neuron uses the Kullback-Leibler divergence, together with a stochastic gradient optimization strategy, to fit low density intervals in the Probability Density Function (pdf) that models the Random Variable (RV) source from which data is drawn. Hereafter, it will be referred to as pdf valleys.

Note that, unlike most clustering approaches, the proposed algorithm does not use prototypes around which data is gathered — points of maximum data density —, but it rather looks for regions of low data density, which are, by hypothesis, between-clusters boundary candidates. As a consequence, the new algorithm, hereafter referred to as the MLP-CLUST, provides some interesting properties, including the possibility of estimating the number of clusters and also providing a hierarchical interpretation of the estimated clusters.

In order to present this new algorithm, this paper is organized as follows: Section 2 presents the neural network structure of the proposed algorithm and the cost function applied to the adaptation of each neuron. In Section 3, a learning strategy is proposed. In Section 4, it is explained how to initialize all parameters, while in Section 5, the clustering algorithm itself is briefly presented. Section 6 presents a useful feature of the algorithm: its automatic cluster labeling capability. Finally, In Section 7, computational results are presented. Conclusions and final discussions are provided in Section 8.

*The authors are with the Universidade Federal de Sergipe (UFS), São Cristóvão, CEP. 49100-000. E-mail:jmontalvao@ufs.br, efreire@ufs.br.

2 The MLP-CLUST Cost Function

The proposed algorithm is based on a classical Neural Network (NN) model: the MLP [4]. Nevertheless, to better explain how it works, we first present the clustering capabilities of a Single Layer Perceptron (SLP). Indeed, Section 7 presents an illustration of how such capabilities can be extended to an MLP.

A SLP performs a nonlinear mapping of $\Re^{N_i}$ on the hypercube $]-1,+1[^{N_o}$, given by:

$$\mathbf{y} = \tanh(g(\mathbf{W}\mathbf{x} + \mathbf{b}))$$

where $\mathbf{x} = [x_1\ x_2\ \ldots\ x_{N_i}]^t$ is an N_i-dimensional real valued column vector, representing an input sample, $\mathbf{y} = [y_1\ y_2\ \ldots\ y_{N_o}]^t$ is an N_o-dimensional column vector, g represents the neuron soma gain, always positive, $\tanh(\cdot)$ stands for hyperbolic tangent function, $\mathbf{W}$ is the synaptic weight matrix, and $\mathbf{b}$ is the neuron bias (or threshold) vector, both real-valued.

In order to present the cost function, since it provides a Hebbian learning rule (i.e. a local rule), we may focus on a single neuron, whose output is:

$$y_i(\mathbf{x}) = \tanh(g(\mathbf{w}_i\mathbf{x} + b_i))$$

where $\mathbf{w}_i = [w_{i,1}\ w_{i,2}\ \ldots\ w_{i,N}]$, is the i-th row of $\mathbf{W}$, corresponding to the synaptic weights of the i-th neuron into the layer, and b_i corresponds to its bias.

Assuming that Y_i is a random variable, according to some theoretical reasons presented in the following, we might take, as a suitable cost function, the following expectation:

$$J_i = -E_{Y_i}\{\log(y_i^2 + 1)\} \tag{1}$$

where the addition to one avoids singularity.

Moreover, defining $h(\mathbf{x}) = (y_i^2(\mathbf{x}) + 1)$ and rewriting the cost function as follows:

$$J_i = -\int_{-\infty}^{+\infty} f_{\mathbf{X}}(\mathbf{x}) \log(h(\mathbf{x}))d\mathbf{x} \tag{2}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is the multivariate pdf of the random variable $\mathbf{X}$, we obtain an expression closely related to the Kullback-Leiber (K-L) divergence [4], a well-known tool to compare probability density functions[1].

Note that, although $\int_{-\infty}^{+\infty} h(\mathbf{x})d\mathbf{x} \neq 1$, the function $h(\cdot)$ is strictly positive for all $\mathbf{x} \in \Re^{N_i}$, which allows the comparison between this function and the pdf of $\mathbf{X}$, as follows:

$$D_{f_{\mathbf{X}}(\mathbf{x})\|h(\mathbf{x})} = -\mathcal{H}(\mathbf{X}) - \int_{-\infty}^{+\infty} f_{\mathbf{X}}(\mathbf{x}) \log(h(\mathbf{x}))d\mathbf{x} \tag{3}$$

Since $\mathcal{H}(\mathbf{X})$ corresponds to the entropy of $\mathbf{X}$ (i.e. the NN input), in order to reduce the K-L distance between $f_{\mathbf{X}}(\mathbf{x})$ and $h(\mathbf{x})$, we can only adapt the function $h(\cdot)$, which corresponds to minimize the cost function in Equation 1. Moreover, since $1 \leq h(\mathbf{x}) < 2$, it is easy to show that the integral J_i converges to a real number greater or equal to 0 and lower than $\log(2)$.

An alternative informal explanation to the goal of the chosen cost function can be provided as follows: let neurons parameters be represented by "V" shaped functions — or high-dimensional "hull boat" shaped function —, then each neuron must change its position and concavity width in order to fit the pdf "valleys" or "gaps" – so minimizing Eq. 3. This is specially suitable for finding straight valleys of multivariate density functions.

Hereafter, the expression "neuron valley" is going to stand for the "V" shaped or "hull boat" shaped function of each neuron, as well as the expression "neuron valley bottom", a point, line or hyperplane that corresponds to the locus of the deepest points of a neuron valley. Figure 1 provides an illustration to the concepts conveyed here, for $N = 2$.

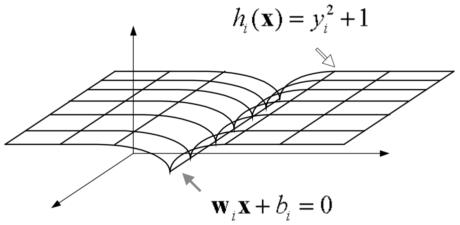

$h_i(\mathbf{x}) = y_i^2 + 1$
$\mathbf{w}_i\mathbf{x} + b_i = 0$

Figure 1: Illustration of the function $h(x)$ of a single neuron.

Note that any algorithm for neuron parameters adaptation based on the minimization of J_i provides a kind of parametric entropy optimization, according to N. Schraudolph [11]. Indeed, in [11] and references therein, we can find a useful discussion about the so-called binary information gain optimization.

3 The Local (Hebbian) Learning

Some strategies for optimizing neurons parameters were tested. However, in this paper, only the simplest online algorithm is presented, where all neuron parameters are adapted, by stochastic gradient descent, for every new incoming data. Accordingly, for each new incoming vector input, the i-th neuron must adapt its j-th weight — corresponding to the synaptic efficiency associated to the scalar input x_j — and bias according to Equations 4 and 5.

$$\begin{cases} w_{i,j}^{k+1} = w_{i,j}^k + \alpha\Delta_i x_j \\ w_{i,j}^{k+1} = w_{i,j}^k / \|w_{i,j}^k\| \end{cases} \tag{4}$$

$$b_i^{k+1} = b_i^k + \alpha\Delta_i \tag{5}$$

where $\Delta_i = \frac{g(y_i(1-y_i^2))}{(y_i^2+1)}$ and α corresponds to the learning rate.

[1] The K-L divergence has some useful properties. Some of them, particularly interesting here are its invariance with respect to: (a) data amplitude scaling and (b) monotonic nonlinear transformation, like the sigmoidal activation functions of popular artificial neuron.

4 Parameters Setup

By assuming that no *a priori* knowledge about the pdf which models the data source is available, in order to reduce the chances of missing a pdf valley, we propose the following weight matrix and bias vector initialization:

(a) $\mathbf{W}$ is initialized with random numbers uniformly distributed between -1 and $+1$;

(b) Each i-th row $\mathbf{w}_i$ of $\mathbf{W}$ is then normalized, i.e. $\|\mathbf{w}_i\| = 1$;

(c) b_i is initialized with random numbers uniformly distributed between 0 and $+1$.

Note that steps (a),(b) and (c), together, are necessary to fit the requirement presented in the Appendix.

5 The MLP-CLUST Algorithm

An important feature of this algorithm is its dependence on parameter g. The greater g is, the bigger the number of pdf valleys seen by the neurons. Equivalently, for high values of g, the cost function (Eq. 1) often presents a high number of local minima.

Indeed, for $g \to +\infty$ the neurons are able to find any valley between any two non-overlapped samples. Otherwise, for $g \to 0$ $(g > 0)$, if samples and neuron weights are finite length vectors, the "V" shaped function $h(\cdot)$ is so stretched that no neuron is able to "find" even a wide valley inside the space spanned by the samples.

These algorithmic aspects suggest a close relationship between our approach and the Deterministic Annealing [9], where g and the temperature parameter of [9] are inversely proportional.

Nevertheless, it is important to highlight one fundamental difference between the Deterministic Annealing (DA) and the proposed algorithm: while the DA approach tries to find a global minimum by gradually reducing the temperature parameter, each neuron of the MLP-CLUST tries to find a local minimum of the cost function. Therefore, although a gradual variation of g is usefull, if it is done from bottom to top (Annealing), all the neurons will converge to the same global minimum, i.e. the same valley, which, obviously, is to be avoided here!

On the other hand, in order to investigate pdf valley on several metric scales, we can set g to a high enough initial value, and then to slowly decrease this parameter to 0, i.e. a heating instead of an annealing strategy.

As a result, the whole MLP-CLUST can be summarized as follows:

1. Raw input data is preprocessed as follows: let $\mathbf{x}$ be an N_i-dimensional real valued column vector, representing an input sample, and let x_i be the i-th element of $\mathbf{x}$. Then each x_i $(1 \leq i \leq N_i)$ is divided by the maximum absolute value among all i-th inputs, from all input samples. Thus, all input vectors are gathered inside a hypercube of unitary edge;

2. g is initialized with a big enough g_0.

   A preliminary result based on the study of the stability points of the cost function J_i, for straight valleys, provided the following helpful inequation: $g_0 \geq 2 \times \text{atanh}(0.486)/GAP_{min}$, where GAP_{min} stands for the width of the narrowest valley we are looking for. Clearly, GAP_{min} is almost always unknown in real-world clustering problems but, even a rough estimation of it can be helpful in setting parameter g_0 (note that, as all input vectors are gathered inside a hypercube of unitary edge, then $GAP_{min} \leq \sqrt{N_i}$, where N_i stands for the dimension of the hypercube.);

3. α is initialized with small positive real;

4. Each neuron's synaptic weights and bias are randomly initialized, according to Section 4;

5. For $n_{ep} = 1$ to N_{ep} epochs:

   - Data samples are randomly presented as inputs;
   - Weights and biases are updated according to Equations 4 and 5;
   - g is linearly decreased by $\Delta_g = g_0/N_{ep}$;
   - Weights of each neuron are normalized.

6 Automatic Binary Labels and Cluster Cardinality

An interesting feature of the MLP-CLUST is the possibility of automatic cluster label generation. It can be provided by the following simple procedure: given an input sample (vector) $\mathbf{x}_m$, it comes form a cluster whose label is given by:

$$\mathbf{L}(\mathbf{x}_m) = \text{sign}(\mathbf{y}_m) \tag{6}$$

where sign$(\cdot)$ stands for the signal function, whose output is a ± 1 valued vector, and $\mathbf{y}_m$ is the corresponding NN output.

Note that each neuron output corresponds to one bipolar bit in the label vector, as well as, in the MLP-CLUST approach, each neuron contributes to between-cluster boundary by providing a single hyperplane. Therefore, if two clusters have labels almost identical, differing by just one label bit, then they (the clusters) are neighbors in the N_i-dimensional feature space, separated by one single hyperplane.

Defining a cluster as any N_i-dimensional subspace, free of neuron hyperplanes, inside which lays at least one input sample, we can estimate the total number of clusters by feeding the neural network with all available samples and counting the number of different labels.

In fact, when the number of neurons is high, such a procedure often produces an also high and meaningless number of clusters. For instance, even if clusters with hundreds of samples are found, a single isolated sample may be considered as being a cluster too.

Note, however, that as the number of neurons increases, even when the gaps between clusters are not clear (noisy case), the number of spurious small clusters is limited, i.e. there is an upper bound proportional to parameter g.

Nevertheless, in order to avoid the (possible) estimation of a high number of too-small clusters, we empirically set that any cluster with cardinality below to 10% of the biggest cluster cardinality is discarded.

It is worth noting that even (geometrically) small clusters may have high cardinality. In fact, small but relevant clusters frequently are related to very concentrated (picked) conditional probability densities (thus providing high-cardinality clusters in sample datasets).

Another important issue concerning cluster cardinality is the dependence of each neuron parameter convergence on the size (cardinality) of clusters in each side of the boundary provided by the neuron. From Equations 4 and 5, it is clear that this dependence is stronger for data points close to the boundary, and it is controlled by parameter g. For instance, note that points far from the boundary provide a Δ_i close to zero. Consequently, as parameter g decreases, the influence of cluster sizes on the boundary positioning also decreases.

7 Practical Results and Illustrations

In this section, some computational results are presented. The first one is based on artificial data, i.e. samples are generated throughout random number generator algorithms, in order to highlight some interesting capabilities of the MLP-CLUST. On the other hand, two experiments based on real data are also reported, providing a more realistic scenario for testing the MLP-CLUST capabilities.

7.1 Non-radial Dispersion Clusters

If more than one layer of neurons is used, the MLP-CLUST is able to find even non-linear between cluster boundaries. In order to provide a simple illustration, Figure 2 shows the result of the non-linear mapping of each layer when a single layer is not enough to provide a between cluster boundary.

Indeed, it is similar to what happens when an MLP is adjusted with the backpropagation algorithm[4], in classification (supervised) tasks. Nevertheless, it is important to highlight that, here, all neurons in both layers are locally adapted with the same Hebbian rule of Equations 4 and 5.

In this illustration, each neuron in the first layer finds a conditional pdf valley. As a result, a single layer finds 4 clusters, but the neuron outputs are not close to ±1, as expected for well-separated clusters.

Finally, by using the first layer output as second layer input, a single neuron is now enough to provide a straight between cluster boundary. Moreover, since the neuron output is quite close to ±1, it can be used as a evidence of the existence of 2 (not 4) clusters, and that there is no need for more layers.

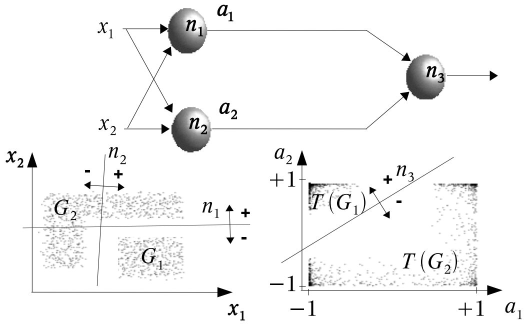


Figure 2: Illustration of the use of more than one layer. $T(\cdot)$ stands for the nonlinear mapping of the first layer, while G_1 and G_2 are labels for clusters 1 and 2, respectively.

Nonetheless, we highlight that learning parameters must be properly tuned in each layer, otherwise the multilayer approach may fail in many ways (e.g. spurious valleys may be generated by unappropriate nonlinear mapping in the first layer). Clearly, it raises many questions that are not going to be addressed here. For a while, we focus our investigation on single layer capabilities.

7.2 Clustering Italian Wines Data

For this experiment we use a wine data set, acquired from the University of California (UCI Machine Learning Repository). This data set consists of 13 continuous-valued features belonging to three physical classes, corresponding to results from chemical analysis of wine produced by three different cultivators from the same region of Italy. This data set contains 178 feature vectors, with 59 in class 1, 71 in class 2, and 48 in class 3.

Note that we have classes, not clusters. However, to compare the MLP-CLUST to others clustering algorithms, we assume that those classes also form clusters on the 13-dimensional feature space, as it was done in [5] and references therein.

As a consequence, after clusters are estimated, we put into correspondence cluster labels and classes labels. Thus, samples whose cluster labels do not correspond to the expected class labels are regarded as "misclassifications".

For this experiment, all sample vectors were mixed up, and we used the following parameters: $\alpha = 10^{-6}$, g was decreased from 2 to 1 for $N_{ep} = 2000$ epochs, with a single layer of 50 neurons.

Figure 3 shows a single run of th MLP-CLUST with the Wine dataset, where a stable configuration of 3 clusters is clearly found for $1.25 < g < 1.75$.

Results presented in Table 1 are averaged and rounded "misclassications". That is, we evaluate the

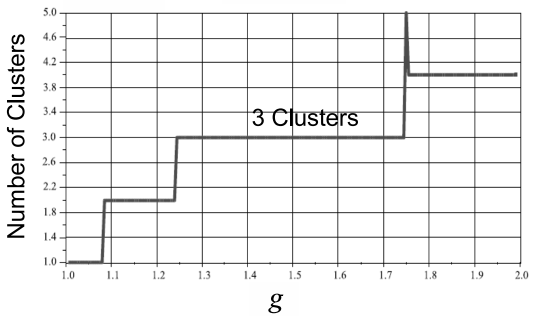


Figure 3: A single run with the Wine dataset (UCI database).

MLP-CLUST performance by computing the number of "misclassications" for $g \approx 1.5$, over 10 independent runs, and then these results are averaged and rounded.

Table 1: Performance Comparison with Italian Wines Data.

Algorithm	"misclassifications"
MLP-CLUST	08
Gustafson-Kessel (GK)	32
Gath and Geva (GG)	49
Fuzzy c-means (FCM)	54
Non-Euclidean FCM(NEFCM)	09

As we can see, the MLP-CLUST outperforms the other algorithms with this specific dataset. It indicates that classes indeed correspond to clusters and that, probably, between-classes separations are well approximated by 13-dimensional hyperplanes in this case.

7.3 Ultrasonic Data

For this experiment with the MLP-CLUST, we have used parameters extracted form echoed ultrasound pulses as input samples [1]. These pulses were obtained from ultrasound sensors placed in front of 6 kinds (classes) of reflectors, 50 cm far form the pulse source. The 6 kinds of reflectors can be summarized as follows: - Convex wall corner, 90° (Class #1);

- Chair leg, circular transversal section (Class #2);

- Table leg, circular transversal section (Class #3);

- Concave wall corner 90° (Class #4);

- Flat wall (Class #5);

- Table leg, squared transversal section (Class #6).

And the pulse parameters to be clustered are: total area under the pulse shape and pulse duration.

Note that the use of only two pulse parameters is a helpful choice in order to provide graphical illustrations. Nevertheless, there is no theoretical limitation to the number of such parameters. Indeed, when the parameters are statistically dependent, the use of higher dimensional spaces (more parameters), can even facilitate the clustering task.

Figure 4 presents the 2-dimensional samples dispersion, labeled according to their classes. It is clear that there exist a strong relationship between pulse classes and clusters, and even classes 4 and 5 can be considered as almost separated radial clusters.

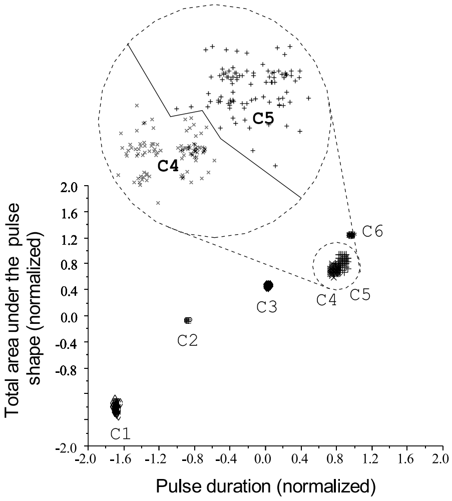


Figure 4: Two-dimensional plot of ultrasonic pulse parameters.

For this experiment with a single layer MLP-CLUST, all samples were mixed up, without their class labels, and we used the following parameters: $\alpha = 10^{-4}$, g was decreased from 10 to 2 for $N_{ep} = 400$ epochs. In order to not overcharge Figure 5 with too many lines, only 100 neurons were used.

Figure 5 presents a single but representative trial with the MLP-CLUST, where subfigures are used to show the "neuron locus" evolution during the run (i.e. each line on subfigures represents a linear boundary provided by one neuron, which corresponds to the locus of points (x_1, x_2) for which the equation $w_{i,1}x_1 + w_{i,2}x_2 + b_i = 0$ holds, where i stands for the neuron index).

Accordingly, the number of clusters, n_G, estimated by the beginning of the algorithm run (from right to left) is high. On the other hand, when "temperature" starts to increase or, equivalently, g decreases, more neighbor samples are clustered together (not necessarily according to the Euclidean distance).

Thus, this temporal evolution of n_G with parameter g tends to be a decreasing one, staying constant for long intervals of g when a stable clustering pattern is found. In other words, when neurons find relevant pdf valleys, they are trapped for longer intervals of g.

Note that, while g is decreased (heating process) the number of estimated clusters, n_G, gradually tends to one (trivial solution were all samples are clustered together).

This gradual variation of n_G with g, which in some sense controls the metric scale of the pdf valley search, suggests a cluster hierarchy.

Figure 6 illustrates a hierarchic interpretation of clusters provided by the MLP-CLUST. This interpre-

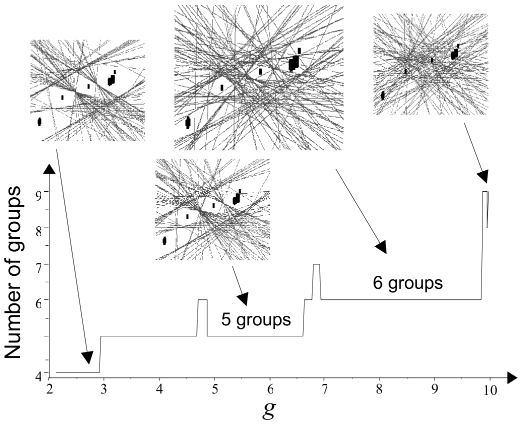


Figure 5: A single run with ultrasonic data.

tation is made clear by observing the cluster labels (see Section 6) and comparing then by Hamming distance measures. As a result, and given the closer relationship between classes and clusters in this particular case, we can also show this hierarchical interpretation by using class labels, as shown in Figure 6.

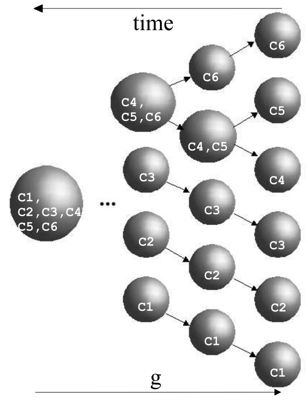


Figure 6: Hierarchic tree suggested by the algorithm.

Moreover, according to the authors of the experiment with ultrasound pulses, pulses from flat wall (Class #5) and pulses from concave wall corner (Class #4) are quite similar, and both are also similar, in a lesser degree, to the pulses from large table lags with squared transversal section (Class #6). Note that this "reasoning´´ is automatically provided by the algorithm output.

When it is compared to the classical K-means, with the same data set, the first remarkable difference is that we must provide the K-means with the number of clusters.

Once it is provided, and referring to as *good results* the clustering output where cluster centers lay close to class centers, there are no warranties that the K-means algorithm is going to converge to this *good result.*

We experimentally found that, for this problem, about 25% of trials with the K-means converged to good results.

On the other hand, the MLP-CLUST provides a flexible probability of correctly estimating the number of clusters and, moreover, to obtain a *good result.* Indeed, the probability P_n of not missing a valley of width Δr depends on both Δr and the number of neurons N_n according to a Binomial distribution, which yields:

$$P_n = 1 - (1 - P_a)^{N_n}$$

where

$$P_a = \frac{\Delta r}{2\pi^N} \int_{-\pi/2}^{\pi/2} \dots \int_{-\pi/2}^{\pi/2} c(\alpha_1, \dots, \alpha_N) d\alpha_1 \dots d\alpha_N \qquad (7)$$

$$c(\alpha_1, \dots, \alpha_N) = \cos\left(\text{atan}\left(\sqrt{\sum_{n=1}^{N} \tan(\alpha_n)^2}\right)\right)$$

$N = N_i - 1$ and N_i is the feature space dimension.

These two formulae are explained in the Appendix. It is interesting to highlight that, with only 100 neurons, in both theoretical and numerical (over 100 independent trials) results, the probability of success on finding all relevant valleys and providing *good results* was about 81 %. Moreover, by increasing the number of neurons we are able to arbitrarily improve this result.

8 Conclusions

A new connectionist algorithm for data clustering was presented: the MLP-CLUST, based on local (Hebbian) adaptation of artificial neurons with sigmoidal activation functions.

Some interesting algorithm characteristics are: (a) its implementation simplicity, where each neuron is adapted every time a new incoming data is provided, and (b) it is rather a probabilistic approach (instead of a geometric one) though, unlike classical probabilistic approach, e.g. the EM algorithm, the MLP-CLUST estimates just partially the data pdf. Indeed, it just looks for pdf valleys, what keeps this new algorithm simpler than the classical EM based clustering approach.

It was also indicated that, with at least two layers of neurons locally adapted — i.e. the same Hebbian algorithm applied to all neurons — it is possible to estimate nonlinear between-clusters boundaries, though we highlight, in Section 7.1, that it raises some important questions concerning the tuning of parameters in both neuron layers, what was not yet studied. On the other hand, it is easy to show that even a single layer of self-organized neurons is able to fit piecewise linear between-cluster boundaries. In other words, it is able to fit even boundaries between non-ellipsoidal clusters. Note that, though we observed this capability in many practical experiments, it also raises many questions concerning algorithm convergence properties for shallow pdf valleys (piecewise liner boundaries usually are "seen" by neurons as shallow marginal pdf valleys), to be addressed in the sequel of this work.

Unlike classical supervised MLP, which suffers from over-dimensioning, the MLP-CLUST has its success probability increased whenever the number of neurons is increased.

Another attracting aspect of the proposed algorithm is that it provides one bipolar label to each estimated cluster, where such labels keep geometric relationship similar to that of clusters: closer labels (in Hamming distance) point to closer clusters (in Euclidean distance).

Finally, a useful tool provided by the bipolar labels, together with the heating strategy, is the possibility of a hierarchical interpretation of more than one stable cluster partitioning.

Important questions to be properly addressed in future works are mainly related to the existence of stable solution and the study of algorithm convergence aspects. For instance, if a continuous spread of input samples comes from a uniform pdf, clearly, it does not present pdf valleys to be fit by the neurons. As a consequence, given that data is mapped into a hypercube of unitary edge, all neurons will diverge toward the outside of this hypercube, and a single cluster is “seen´´. On the other hand, if even a shallow pdf valley does exist (a marginal pdf valley, for instance), it always can trap neurons for high enough values of parameter g. Consequently, the study of how to optimize g in order to accelerate algorithm convergence remains an open issue.

Furthermore, in the sequel of this work, we are going to address some important missing points, such as: (a) the application of the algorithm to an unlimited stochastic data source, such as a communication channel model and, by this way, to evaluate the MLP-CLUSTER performance on channel equalization/identification, according to some theoretical issues presented in [8]; and (b) a study of how parameters g and α influence the algorithm performance, including stability and convergence issues.

Acknowledgments

This work was partially granted by both the *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq) and the *Fundação de Amparo à Pesquisa - Sergipe* (FAP-SE).

Appendix: Analytic Calculation of P_a

Given two clusters whose centers are separated from each other by Δr, P_a corresponds to the probability of a single neuron, initialized as in Section 4, to have its “neuron valley bottom” (see Section 2) crossing the line segment that links the two centers. Here we assume, by simplification, that there is a pdf valley bottom along the line linking the two centers, and that any neuron whose “neuron valley bottom” crosses this line is able to find the valley position and orientation through adaptation.

If many clusters are to be considered, P_a may correspond to a lower bound if Δr is the distance between the two closest centers.

In a 2-dimensional feature space, the probability P_a is analogous to that of a line randomly placed — so that it must intersect the unity circle — crossing the needle (or a piece of wire) of length Δr inside this circle (see Figure 7).

As a consequence, α, the angle between the needle and an axis normal to the above mentioned line, is a random variable with flat pdf $f(\alpha) = 1/\pi$, $-\pi/2 \leq \alpha \leq \pi/2$. And then, given an instance of α, it is easy to see, in Figure 7, that the probability of intersection between needle and line is given by $P(\alpha) = \Delta r \cos(\alpha)/2$. Therefore, the averaged probability over all direction is given by the integral:

$$P_a = \int_{-\pi/2}^{\pi/2} f(\alpha) P(\alpha) d\alpha$$

$$P_a = \frac{\Delta r}{2\pi} \int_{-\pi/2}^{\pi/2} \cos(\alpha) d\alpha = \frac{\Delta r}{\pi} \qquad (8)$$

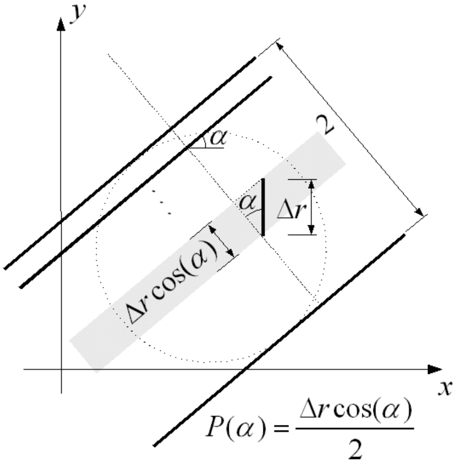


Figure 7: Pa calculation on 2D .

By analogy, in a 3-dimensional feature space, where the line is replaced by a plane $g(x,y) = \tan(\alpha_x)x + \tan(\alpha_y)y + c$ that intersects the unity sphere, depicted in Figure 8, and where α_x and α_y are, respectively, the plane slopes in the corresponding orthogonal direction, it is clear that the maximum slope is the gradient magnitude, given by $\|\nabla g\| = \sqrt{\tan(\alpha_x)^2 + \tan(\alpha_y)^2}$.

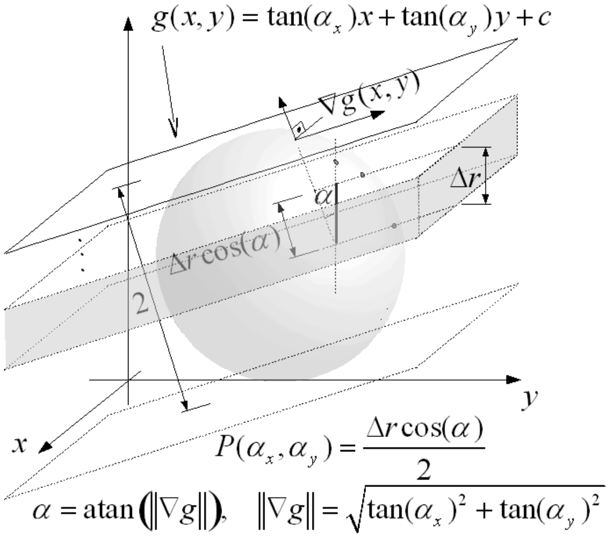
That is, the (maximum) slope angle through the direction pointed by the gradient vector is $\alpha = \text{atan}(\|\nabla g\|)$. Consequently, as it is illustrated in Figure 8, given α_x and α_y, the probability that the plan crosses the needle is $P(\alpha_x, \alpha_y) = \Delta r \cos(\sqrt{\tan(\alpha_x)^2 + \tan(\alpha_y)^2})/2$, and the averaged probability over all directions, in this case, is then:

$$P_a = \int_{-\pi/2}^{\pi/2} \cdots \int_{-\pi/2}^{\pi/2} f(\alpha_x, \alpha_y) P(\alpha_x, \alpha_y) d\alpha_x d\alpha_y$$

where $f(\alpha_x, \alpha_y) = 1/\pi^2, -\pi/2 \leq \alpha_x, \alpha_y \leq \pi/2$ is a joint flat pdf, yielding:

$$P_a = \frac{\Delta r}{2\pi^2} \int_{-\pi/2}^{\pi/2} \cdots \int_{-\pi/2}^{\pi/2} c(\alpha_x, \alpha_y) d\alpha_x d\alpha_y$$

where $c(\alpha_x, \alpha_y) = \cos(\text{atan}\sqrt{\tan(\alpha_x)^2 + \tan(\alpha_y)^2})$.



$g(x,y) = \tan(\alpha_x)x + \tan(\alpha_y)y + c$
$\nabla g(x,y)$
Δr
α
$\Delta r \cos(\alpha)$
2
y
x
$P(\alpha_x, \alpha_y) = \frac{\Delta r \cos(\alpha)}{2}$
$\alpha = \text{atan}(\|\nabla g\|)$, $\|\nabla g\| = \sqrt{\tan(\alpha_x)^2 + \tan(\alpha_y)^2}$

Figure 8: Pa calculation on 3D .

Finally, Equation 7 is a straightforward generalization of the former deduction for feature space dimensions greater than two. Figure 9 shows values of P_a versus feature space dimension from 1 to 60. The rugged lines correspond to estimations of P_a through Monte-Carlo simulation (i.e. 500.000 random neuron initialization and verification whether the "neuron valley bottom" was or not crossing the line linking two centers separated from each other by Δr), while the continuous lines represent the computation of P_a with Equation 7.

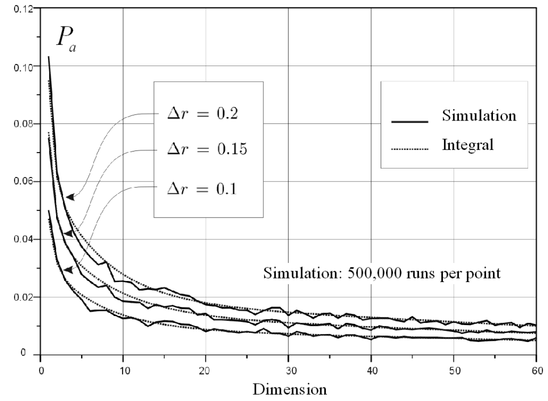


Figure 9: Theoretical and experimental values of Pa versus dimension.

References

[1] A.R. Almeida, E.O. Freire, C.A. Rennó, J.E.S. Vianna, and R.M. Rosi. Neural network recognition of geometric references applied to ultrasound echo signals. *Proceedings of the IEEE 43rd Midwest Symposium on Circuits and Systems - MWSCAS'2000*, 2000.

[2] C.C. Calvacante, J.R. Montalvão, B. Dorizzi, and J.C.M Mota. A neural predictor for blind equalization of digital communication systems: Is it plausible? *IEEE Neural Networks for Signal Processing (NNSP 2000)*, pages 11–13, December 2000.

[3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.

[4] S. Haykin. *Neural Networks, A Comprehensive Foundation.* Prentice-Hall, Englewood Cliffs, USA, 2 edition edition, 1999.

[5] N.B. Karayiannis and M.M. Randolph-Gips. Soft learning vector quantization and clustering algorithms based on non-euclidean norms: Multinorm algorithms. *IEEE Trans. Neural Networks*, 14(1):89–102, January 2003.

[6] G.W. Milligan and M.C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, pages 159–179, 1985.

[7] S. Mitra, S.K. Pal, and P. Mitra. Data mining in soft computating framework: A survey. *IEEE Trans. on Neural Networks*, 13(1):3–14, January 2002.

[8] J. R. Montalvão, B. Dorizzi, and J. C. M. Mota. Channel estimation by symmetrical clustering. *IEEE Trans. on Signal Processing*, 50(6):1459–1469, June 2002.

[9] K. Rose. Deterministic annealing for clustering compression, classification, regression and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239, December 1998.

[10] W.S. Sarle. Cubic clustering criterion. Technical Report A-108, SAS Institute Inc., Cary, NC, 1983.

[11] N. Schraudolph. *Optimization of Entropy with Neural Networks.* PhD thesis, University of California, San Diego, 1995.