New Algorithm for Neural Network Data Discrimination Applied to Markarian 421 High Energy Gamma Rays

Luc Macot

A Thesis

in

The Department

of

Physics

Presented in Partial Fulfilment of the Requirements for the
Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

April 1998

0-612-40302-5

ABSTRACT

New Algorithm for Neural Network Data Discrimination Applied to Markarian 421 High Energy Gamma Rays

Luc Macot, Ph.D.
Concordia University, 1998

A new neural network learning algorithm, called the Umbrella Algorithm, is developed and analysed. Its generalization, which does not exhibit over-specialisation, is observed in the EXOR problem and in an artificial data discrimination (Toy Data) problem. The learning time is found to be about 1/15 of backpropagation learning time for the parity problem. The algorithm is applied to cosmic high energy gamma ray detection and is further used for determining the spectral index of the Markarian 421 source.

STATEMENT OF ORIGINALITY

In this thesis,

1. I invented, developed and analyzed the Umbrella Algorithm.

2. I applied the Umbrella Algorithm for the first time to a Toy Data problem, the Standard Parity problem and the EXOR problem.

3. I invented a very practical method for constructing a learning set from raw data using a Bayesian-like separator technique.

4. I used the Umbrella Algorithm with the above learning set to construct a neural network architecture.

5. I used the above architecture to discriminate between gammas and hadrons in Markarian 421, producing roughly 3 times the number of photons found by others, to a high excess in sigma.

6. I obtained, from the above, a fuller curve than previously found for the high-energy end of the spectral index function, enabling a potentially more careful spectral analysis.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Barry Frank. His interest, encouragement and patience have been a major motivation throughout the work and have been greatly appreciated.

I would like to express my appreciation to the Whipple collaboration for the data provided.

I would like to thank Jacques Lafortune for fruitfull discussions in the early developments of the neural network software.

I would like to thank Monique Macot for her patience and understanding throughout the research years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND DEFINITIONS

x, y, z: neuron states of the x, y, and z layers respectively.

ω: output cell state.

i, i′, ι, ι', j, j′, k, m: neuron indices.

d_x, d_y, d_z: dimensions of the x, y and z layers respectively.

κ, κ': class example indices (in exponents).

α, α': class indices (they take values 0 and 1).

$\Gamma^{\alpha}, \Gamma^{\alpha'}$: set of values of κ, κ' for classes α and α' numbering $N^{\alpha}, N^{\alpha'}$ examples in classes α and α' respectively.

$\left\{x_i^{\kappa}\right\}$: set of neuron states; i runs over the neuron indices of the layer and κ runs over the examples of the α class.

u,v: any real-valued variables.

S(u): output function of the neuron, also called signal function and activation function; S is for sigmoid here.

c_j: sigmoid constant of the neuron j.

θ_j: sigmoid threshold of the neuron j.

$R_x^{\alpha\alpha'}$: similarity measure of the vectors of α and α' in the x layer.

$\langle y_j \rangle_\alpha$: average, over the α class, of the j neuron state value in the y layer.

β, γ: learning rates.

U: umbrella function; it carries the index of the layer.

W: monitoring umbrella function

f, g: feedback control parameters.

C_{ij}: linear coupling from the i cell of a layer to the j cell of the next layer.

$D_{ii'j}$: non-linear coupling from the product of the i and i' cells of a layer to the j cell of the next layer.

p: number of the current pass in a learning process

Δ: correction, applied from pass p to pass p+1, to the architecture variable it prefixes.

$\vec{v}$: a vector of unknown class.

E: Error (function): number of input configurations from a learning set or a test set that are assigned to the wrong class.

δ: Kronecker delta.

L W M D Az F2 F3 Al: Length, Width, Miss, Distance, Azimuth width, Fraction 2, Fraction 3, Alpha; Hillas parameters.

S: Size, total number of photomultiplier impulses.

σ: excess in sigma.

Q: Quality factor, in astrophysics data discrimination.

N(S): Number of events having a size larger than S.

I - INTRODUCTION AND DEFINITIONS

For the last few decades, research and achievements about neural systems have been growing exponentially (Hubel 1979). Although biology is the central science, the challenge is broad enough to require intervention from all fields of knowledge and research. Almost any discipline has something to say about the brain, from psychology to computer science and physics (see for instance the special issues of Sc. Am., Sept. 1979 and 1992). Physics has been involved, through technology, in the development of artificial computing and artificial memory, beginning in the nineteenth century with the works of Jacquart and Babbage (Morrison and Morrison 1961), more recently using magnetic tapes, flip flops, etc. Those successes led to the ambitious project of modeling biological memory via physical systems, by holography for instance (Willshaw 1981) or higher processes like pattern recognition by moments (of inertia) or their ratios (Alt 1962).

More recently statistical physics techniques were used to construct memory models having biological-like behaviours (Smolensky and Riley 1984, Cotterill 1986, Smolensky 1987, Cooper 1988, Sompolinsky 1988). They propose a broad conceptual framework concerned about stability conditions, order-disorder transitions, fixed points, etc., that are useful for the understanding of information storage and macroscopic brain dynamics (Hopfield 1982, Kirkpatrick et al. 1983, Lukashin 1987).

Since the Perceptron days (Rosenblatt 1962, and Minsky-Papert 1969) it is well established that neural networks (see Section II-1) require intermediate layers between their input and output, the so-called hidden layers, or internal representation layers, in order to reproduce various interesting behaviours. Existence theorems about the structure of these intermediate layers have been proven recently (Kolmogorov, in Hecht-Nielsen 1989 and Leshno et al 1995); but the problem of the so-called hidden layer has not been settled yet (and may never be). The now classical approach to the supervised learning problem, which amounts to the construction of the hidden layer(s), is to use the discrepancy between the actual output of the network and the desired output to modify the architecture variables to reduce that discrepancy or error function. Backpropagation is the archetypal example of that type of approach. Many variations and improvements have been proposed to circumvent the so called local minima problems (see, for example Hinton 1989b). Presented here, is a new approach to the hidden layer construction that does not use the traditional error function; the approach is to try to construct an internal representation as conformal to the problem as possible.

The typical problem addressed here is a standard inverse problem: given the caracteristics of an individual event the system, ideally, will be able to identify to which population the event belongs. It is a classical data discrimination problem that finds many applications in physics and in science in general. The inverse problem and the data discrimination problem already have some solutions in statistics and in filtering. Since the biological system easily outperforms the sophisticated fast fourier transform, for example in the well-known cocktail effect or in the 3D stereo images (Hankinson 1994), it may be interesting to try to reproduce, artificially, such behaviours and exploit them in an inverse problem context.

I-1 Comparison of exact and biological memories

Figure I-1 presents typical behaviours of memories. Although there are some measures for a very simple artificial neural network in Appendix II, it is only a conceptual illustration, and does not correspond to actual measurements for biological systems. The horizontal axis is the number of input configurations that have to be stored and the vertical axis, to the same scale, is the number of input configurations that are actually stored, or memorized, in the device. The dotted line represents the behaviour of an exact memory like a computer or an ideal tape. The rule is: whatever has been stored once can be recovered exactly; there is a one-to-one mapping between the input configuration and the encoding kept by the memory. There is no loss of information, the only treatment being an appropriate encoding. The solid curve is an illustration of the required behaviour of biological memories. For a small number of examples, the biological system usually performs below the exact memory level; but when the number of examples reaches some critical value, the biological memory outperforms the exact one.

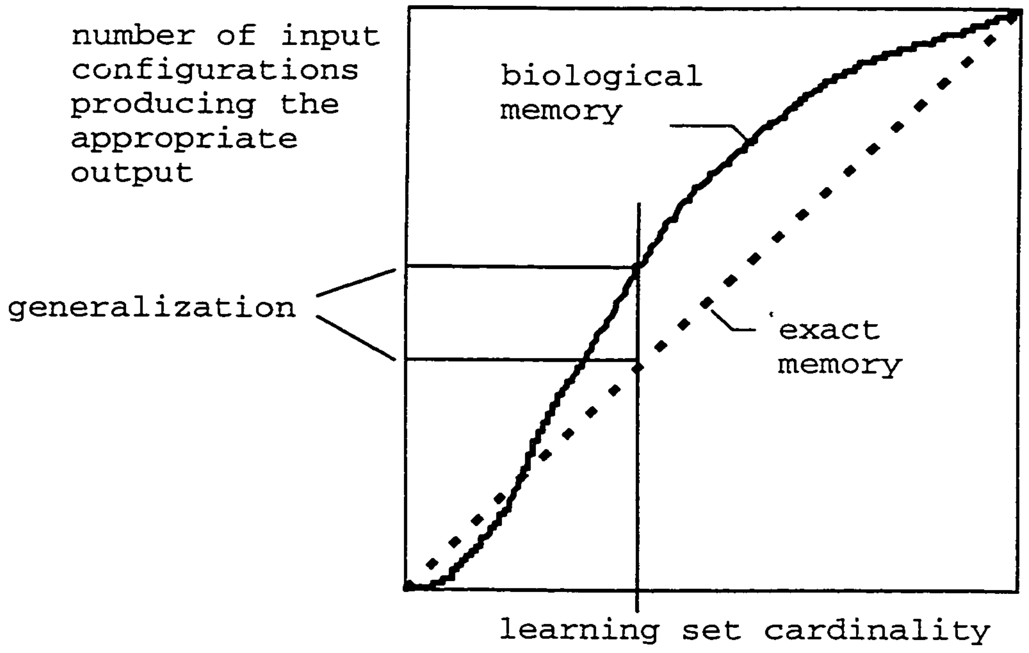


Figure I-1 Exact and biological memory storage.

That is because the storage is an internal representation rather than a one-to-one encoding and therefore some treatment of information is required. Since the number of recognized input configurations is larger than the stored ("cognized") ones, the mapping from input to internal representation must be many-to-one and there has to be some information loss in this process. It is this property of biological memory which can be used to filter or discriminate data in physics problems; an application in astrophysics is discussed in Chapter V.

I-2 General Definitions

An artificial neural network has two modes of operation: i) the learning mode, during which the architecture is built up or modified in some appropriate way so as to achieve the desired goal and ii) the normal operation mode, that is, the performance by the neural network of the task which it has learned.

Learning is a process, the essential ingredients of which, in our context, are i) the neurons, which are seen as nodes in some network, ii) the architecture which defines the linkage of the neurons to each other; iii) the set of rules required to build up the architecture (the "learning algorithm") and iv) the learning set: that is, a certain number of input configurations which are to be taken as examples of the problem to be learned by the network. During the first mode of operation, the learning itself, the network is exposed to a certain number of input configurations, the learning set. Initially the neural network starts in a randomly defined state and does nothing interesting; its inputs and outputs are not related in any particular way. It is the purpose of the learning process to make the system evolve toward the desired function or to map from the inputs to the outputs by adjusting the relevant architecture variables.

The crucial difference, in recognition behaviour, between the exact and biological memories is called generalization; it is as if the biological system actually knew more than whatever was learned (the strict information content of the learning set). Biological systems not only learn but they also automatically generalize. That generalization is a vital property of the learning ability of biological neural networks.

There are many conceptual models in the Psychology literature (for example: the Pandemonium in Selfridge 1959, Lindsay and Norman 1977; the Memory surface in de Bono 1969; and a general discussion about schemata in Rumelhart and McClelland 1986) that try to describe or analyse such learning-generalization abilities; they are certainly useful, but they remain on the macroscopic level; i.e., they do not lead naturally to operational neural network architectures.

There are also many neural network architecture candidates in the current literature (see Rumelhart and

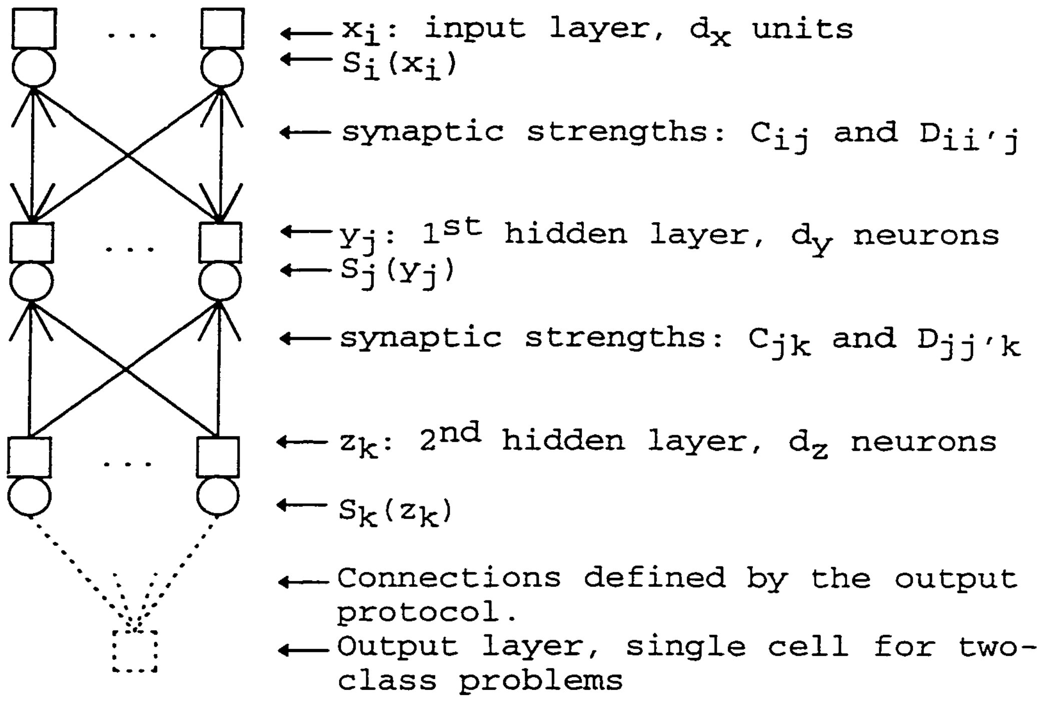
McClelland 1986, Aderson & Rosenfeld 1989, Hinton 1989, Wasserman 1989, Kohonen 1989, Aleksander & Morton 1990, Kosko 1992 Haykin 1994) that possess learning and generalization abilities. It seems that supervised error learning devices have a tendency to over-specialise. Their asymptotic convergent behaviours tend to exact memories (Hecht-Nielsen 1989, Haykin 1994, Blayo et al 1996), and after a certain number of learning steps the generalization diminishes. One of the crucial differences between that kind of learning and ours is that the generalization in our device cannot diminish, which makes it more appropriate for applications like data discrimination.

Chapter II presents the description of the learning algorithm and the description of its typical behaviour; Chapter III is the program outlines; Chapter IV presents the simulation experiments and Chapter V presents an astrophysics application.

II - UMBRELLA ALGORITHM

II-1 Architecture definitions

Let x_i be the inputs, y_j the first hidden layer and z_k the second hidden layer. Only one and two hidden layers will be discussed, figure II-1.



x_i: input layer, d_x units
$S_i(x_i)$
synaptic strengths: C_{ij} and $D_{ii'j}$
y_j: 1st hidden layer, d_y neurons
$S_j(y_j)$
synaptic strengths: C_{jk} and $D_{jj'k}$
z_k: 2nd hidden layer, d_z neurons
$S_k(z_k)$
Connections defined by the output protocol.
Output layer, single cell for two-class problems

Figure II-1 Neural network architecture

The real valued neurons states are then given by:

$$y_j = \sum_{i=1}^{d_x} C_{ij} S_i(x_i) + \sum_{i=1}^{d_x} \sum_{i'=i}^{d_x} D_{ii'j} S_i(x_i) S_{i'}(x_{i'})$$

$$z_k = \sum_{j=1}^{d_y} C_{jk} S_j(y_j) + \sum_{j=1}^{d_y} \sum_{j'=j}^{d_y} D_{jj'k} S_j(y_j) S_{j'}(y_{j'})$$

where: d_x, d_y and d_z are the dimensions, i.e. the number of cells, in the input and the first and second hidden layers, respectively.

The S(u) are the neuron signal output functions. The centered **sigmoid** functions is used here:

$$S(u) = \frac{1 - e^{-c(u-\theta)}}{1 + e^{-c(u-\theta)}}$$

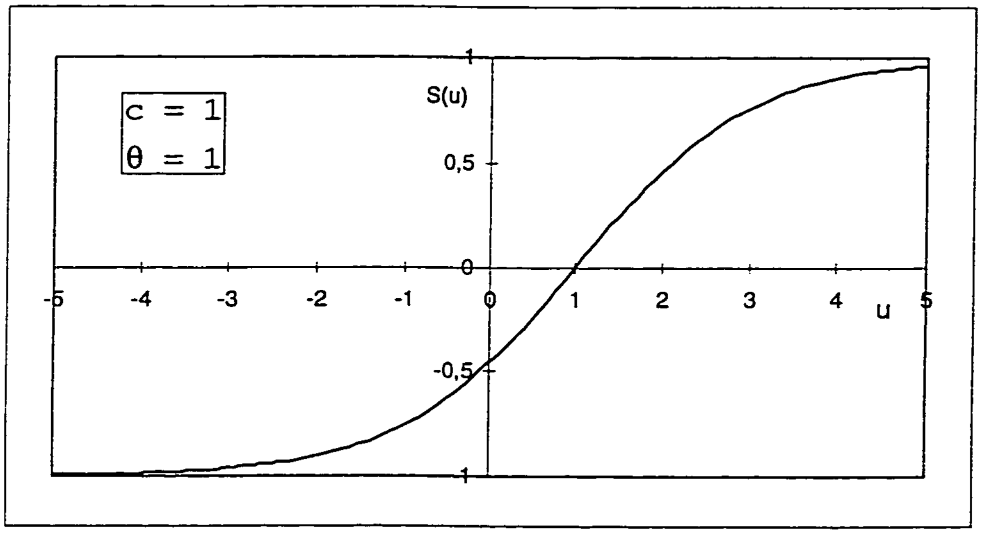


Figure II-2 Vertically centered sigmoid function

c is the sigmoid constant, it fixes the slope of the function at the root: $\left.\frac{\partial S}{\partial u}\right|_{u=\theta} = \frac{c}{2}$; and θ is the threshold. They both are neuron dependent and can be learned by the algorithm. This is not biologically realistic (Appendix I and Dowlings 1992).

The C_{ij}'s and C_{jk}'s are the linear combination coefficients; the $D_{ii'j}$'s and $D_{jj'k}$'s are the non-linear couplings in the system; they are both learned.

II-2 Algorithm description

Learning set: Let $\{\mathbf{x}^{\kappa}\}$ and $\{\mathbf{x}^{\kappa'}\}$ be input vector examples representing the two classes Γ^{α} and $\Gamma^{\alpha'}$; κ and κ' are indices running from 1 to N^{α} and $N^{\alpha'}$ respectively. The sets $\{\mathbf{x}^{\kappa}\}$ and $\{\mathbf{x}^{\kappa'}\}$ constitute the learning set.

Input Layer: The function of the input layer is

1) to distribute the signals to the first operating layer and

2) to center the problem; this is done by setting the sigmoid thresholds of the input layer to the cell-averages of the learning set:

$$\theta_i = \frac{1}{2}\left[\frac{1}{N^{\alpha}} \sum_{\kappa\in\Gamma^{\alpha}} x_i^{\kappa} + \frac{1}{N^{\alpha'}} \sum_{\kappa'\in\Gamma^{\alpha'}} x_i^{\kappa'}\right]$$

Output layer: The function of the output layer is to decode the internal representation in order to eventually produce the desired output. The output layer is connected to the internal representation by a procedure that called the output protocol (Figure II-3) and is described below.

Internal representation layers: Some of my preliminary studies (Appendix II) as well as other research (Grossman 1988, Frean 1990) have shown that the quality of learning and generalization depends on the internal representation.

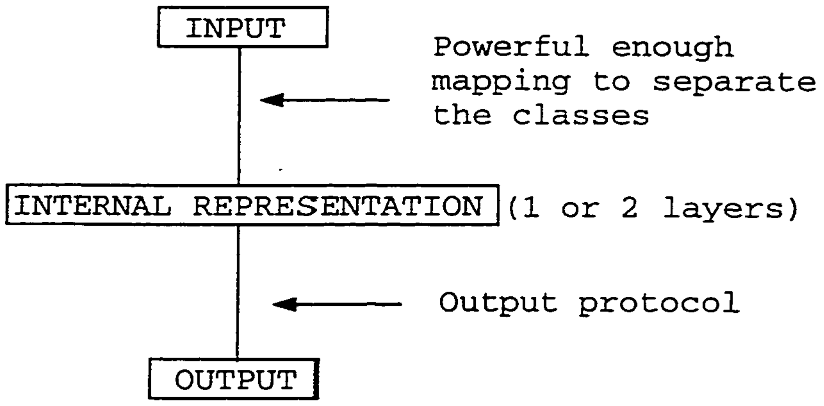


Figure II-3 The three types of layers

An appropriate internal representation would be one for which input configurations belonging to the same class are mapped to (or have) the same internal representation and the different classes correspond to as different as possible internal representation vectors. The learning problem is then to optimize the internal representation in that sense. The internal representation must prepare, encode, simplify the problem occurring at the input layer in such a way that the output layer can actually solve the problem. The hidden layer must extract the significant features that characterise the class of the input as well as the significant features that discriminate among the different classes. One classical approach is to use some predefined distance function and distance value to be used for classification purposes: all internal representation configurations within the chosen distance of an internal representation of a well-known class are declared to belong to that class. The present approach consists in adapting the distance value: to increase the distance between internal representation vectors that must be discriminated, and diminish the distance between the internal representation vectors belonging to the same class.

The main feature of the umbrella algorithm is to induce the system to construct internal representations $\{\mathbf{y}^{\kappa}\}$, $\{\mathbf{y}^{\kappa'}\}$, $\{\mathbf{z}^{\kappa}\}$... such that all the vectors (of the same layer) from the same class are as similar as possible while vectors from different classes are as distinct as possible. Ideally all the vectors of the same class will be mapped to the same single internal representation vector (in some layer), while the different class vectors will point as far apart as possible.

For that purpose **similarity measures** $R_Y^{\alpha\alpha'}$ are defined, for two sets of vectors $\{\mathbf{y}^{\kappa}\}$ and $\{\mathbf{y}^{\kappa'}\}$:

$$R_Y^{\alpha\alpha'} \equiv \frac{1}{d_Y} \sum_{j=1}^{d_Y} \left(\frac{1}{N^{\alpha}} \sum_{\kappa=1}^{N^{\alpha}} s_j(y_j^{\kappa}) \, \frac{1}{N^{\alpha'}} \sum_{\kappa'=1}^{N^{\alpha'}} s_j(y_j^{\kappa'}) \right)$$

where the y_j^{κ} are the components of $\mathbf{y}^{\kappa}$; the upper indices label the class and the lower indices indicate the components, or the neurons. The $R_y^{\alpha\alpha'}$ are average dot products measuring the correlations i) within a class when $\alpha = \alpha'$ and ii) from class-to-class when $\alpha \neq \alpha'$. The similarity measures are defined for each layer by replacing y by the appropriate layer index. With the standardization factor $\frac{1}{d_y}$ the similarity measures are independent of the dimensions of the internal representation layer and one can use them to follow the evolution of classes' overlapping as a function of the depth of the layers when they do not have the same number of cells. In Figure II-4, the region of input space between the vectors $x^{k=1}$ and $x^{k'=1}$ is an example of overlapping.

To discuss the mathematical properties of the similarity measures, define the class sigmoid average:

$$\langle y_j \rangle_\alpha \equiv \frac{1}{N^\alpha} \sum_{\kappa \in \Gamma^\alpha} s_j(y_j^\kappa)$$

as the average, over the class, of the *j-th* component and $\langle \vec{y} \rangle_\alpha$ as the average class vector in the *y-layer;* then:

$$R_y^{\alpha\alpha'} = \frac{1}{d_y} \langle \vec{y} \rangle_\alpha \cdot \langle \vec{y} \rangle_{\alpha'}$$

The asymptotic behaviour of the similarity measures is: $R_y^{\alpha\alpha'} = 1$ when $\alpha = \alpha'$ and $R_y^{\alpha\alpha'} = -1$ when $\alpha \neq \alpha'$. That will happen when $\langle\vec{y}\rangle_\alpha \cdot \langle\vec{y}\rangle_{\alpha'} = d_y$ for $\alpha = \alpha'$ and $\langle\vec{y}\rangle_\alpha \cdot \langle\vec{y}\rangle_{\alpha'} = -d_y$ for $\alpha \neq \alpha'$. The three last condition can be fulfilled only when

(1) all the components of the class average vectors are +1 or -1 and

(2) whenever a component of a class average vector is +1, the same component of the other class average vector is -1. That is, for the components to be ±1, all the sigmoids must be saturated: their argument must be "large" in absolute value. And since the components $\langle y_j \rangle_\alpha$ of the $\langle\vec{y}\rangle_\alpha$ vectors are averages of values bounded below by -1 and above by +1, one must have: $y_j^\kappa = \pm 1 \; \forall \, \kappa \in \Gamma^\alpha$ simultaneously with $y_j^{\kappa'} = \mp 1 \; \forall \, \kappa' \in \Gamma^{\alpha'}$; that is, the mathematical realization of: "all the vectors of the same class are mapped to a single internal representation vector and the two internal representation class vectors are as far apart as possible", see Figure II-4 (Frank 1995, Macot 1996).

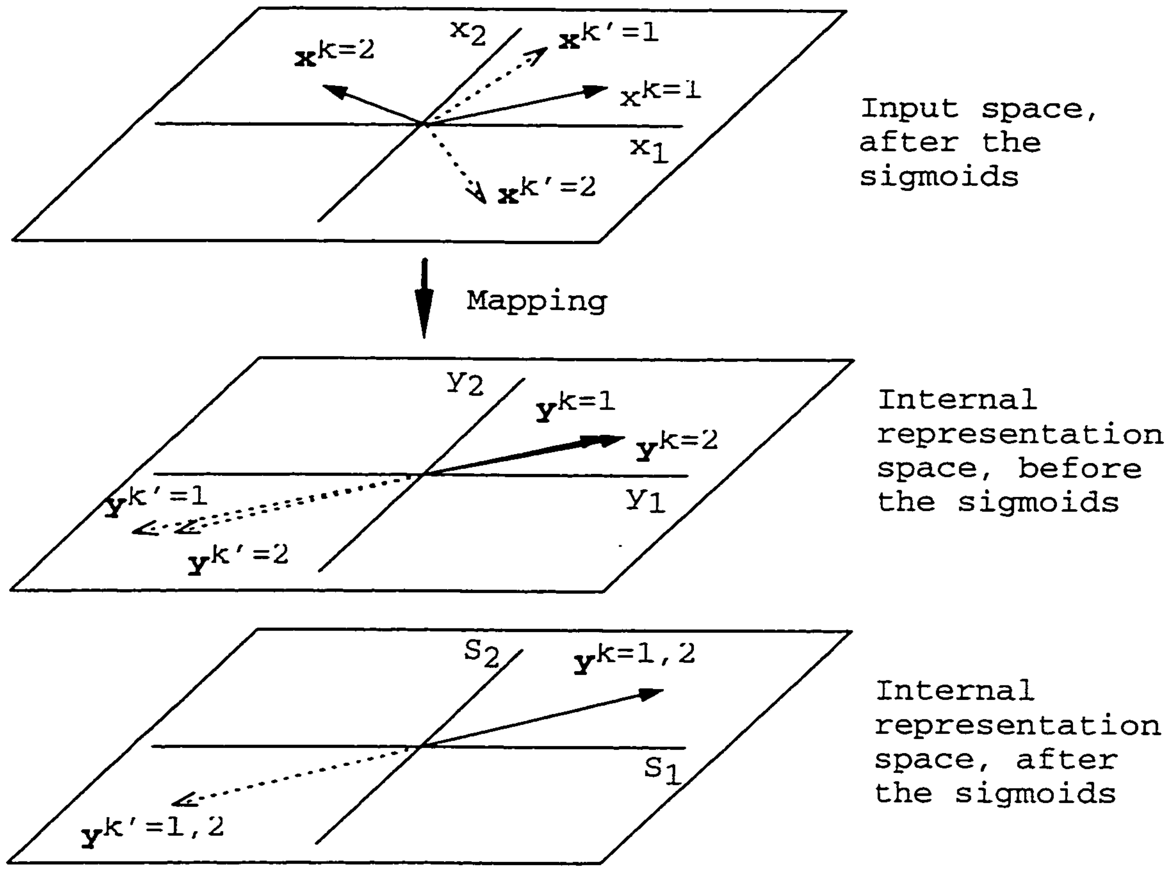


Figure II-4 Input to internal representation mapping

The mapping separates the classes, and the sigmoids compress the vectors toward the classes averages. This is the image at the origin of the name **umbrella**, as if the vectors were two open umbrellas joined at their tips with their spokes initially intertwined; the algorithm closes them and simultaneously moves the handles away from each other.

The similarity measures depend on the neuron parameters $\{c_j\}$ and $\{\theta_j\}$ and, through the mapping, on the couplings $\{C_{ij}\}$, for the linear dependence of the neuron states of the internal representation on the inputs; and $\{D_{ii'j}\}$, for the non-linear part of the mapping function. The purpose of the learning algorithm is then to adapt these four sets (per layer) of adjustable parameters to achieve the smallest possible value (ideally -1) for $R_y^{\alpha\alpha'}$ when $\alpha \neq \alpha'$, and the largest possible values (ideally +1) for $R_y^{\alpha\alpha'}$ when $\alpha = \alpha'$. The argument applies as well for any following layers.

The similarity measures are used to further define the layer umbrella function for the y-layer

$$U_y \equiv \sum_{\alpha} \sum_{\alpha' \geq \alpha} \left[\beta\delta_{\alpha\alpha'} + \gamma(1 - \delta_{\alpha\alpha'})\right] R_y^{\alpha\alpha'}$$

where $\beta > 0$ and $\gamma < 0$ are the learning rates for the within-class and class-to-class correlation measures respectively. The umbrella functions for the other layers are similarly defined by replacing the y-index by the appropriate one.

The conditions on the similarity measures amounts to maximization of the umbrella function. When there are only two classes, named 0 and 1, as in all the simulations and application problems discussed in the present research, the umbrella function becomes:

$$U_Y = \beta (R_Y^{00} + R_Y^{11}) + \gamma R_Y^{01}$$

With the (arbitrary) choice $\beta = 1$ and $\gamma = -2$, the umbrella function becomes the "monitoring function":

$$W_Y = (R_Y^{00} + R_Y^{11}) - 2 R_Y^{01}.$$

In terms of the R's this becomes:

$$W_Y = \frac{1}{d_Y}\left[\left\|\langle\vec{y}\rangle_0\right\|^2 + \left\|\langle\vec{y}\rangle_1\right\|^2 - 2\langle\vec{y}\rangle_0 \cdot \langle\vec{y}\rangle_1\right] = \frac{1}{d_Y}\left[\langle\vec{y}\rangle_0 - \langle\vec{y}\rangle_1\right]^2$$

and is a distance measure between the two class average vectors.

In the context of many layers, there is a W-function defined for each layer. E.g., with one input layer, two internal representation layers and one output layer one has:

$$W_X = \frac{1}{d_X}\left[\langle\vec{x}\rangle_0 - \langle\vec{x}\rangle_1\right]^2$$

$$W_Y = \frac{1}{d_Y}\left[\langle\vec{y}\rangle_0 - \langle\vec{y}\rangle_1\right]^2$$

$$W_Z = \frac{1}{d_Z}\left[\langle\vec{z}\rangle_0 - \langle\vec{z}\rangle_1\right]^2$$

$$W_\omega = \frac{1}{d_\omega}\left[\langle\vec{\omega}\rangle_0 - \langle\vec{\omega}\rangle_1\right]^2$$

where ω stand for "output" and the d's for the dimensions of their respective layers. The purpose of the algorithm is to map the input vectors into other vectors in successive layers:

$$\left\{\mathbf{x}^{K}\right\} \rightarrow \left\{\mathbf{y}^{K}\right\} \rightarrow \left\{\mathbf{z}^{K}\right\} \rightarrow \left\{\omega^{K}\right\}$$

and $\left\{\mathbf{x}^{K'}\right\} \rightarrow \left\{\mathbf{y}^{K'}\right\} \rightarrow \left\{\mathbf{z}^{K'}\right\} \rightarrow \left\{\omega^{K'}\right\}$

in such a way as to have: $W_x \leq W_y \leq W_z \leq W_\omega \leq 4$.

That is: the asymptotic values of the W-functions are 0, when all the vectors of all the classes are mapped into a single vector in the internal representation; and 4 when all vectors of the same class are mapped into a single class vector AND the two class average vectors point in opposite directions.

This can be accomplished provided that:

(1) the vectors are properly normalized or compressed for the inequalities to be meaningful.

(2) the mapping is powerful enough to separate the classes (cf. the one dimensional problem in the remarks and justifications section).

The normalized approach was explored by Kohonen (Kohonen 1989) albeit with a completely different algorithm. The sigmoid compression is used here. The normalization approach may lose crucial information provided by the norm of the vectors, in certain problems; with compression replacing normalization that information is not completely lost.

The umbrella algorithm is used to determine the architecture variables for all the layers from the input to the last internal representation layer. The connections from the last internal representation layer to the output layer are determined by the output protocol defined below. The learning then consists in finding the four sets of values (per layer, that is): $\{c_j\}$, $\{\theta_j\}$, $\{C_{ij}\}$ and $\{D_{ii'j}\}$ such that the umbrella function reaches its largest possible value.

Since the problem cannot be solved analytically, due to the complexity of the equations (θ's and c's occur in exponents) and the large number of unknowns, the maximization is obtained using the gradients of the umbrella functions, one per layer, to iteratively adjust all the architecture variables. The corrections are made in the direction of increasing umbrella functions. The corrections are made synchronously on all the variables, at each **pass**. In one pass the algorithm goes through all the examples of the learning set, calculates all the neuron states (except the output that does NOT participate in the learning), calculates the gradient of the umbrella functions and applies the corrections to the architecture variables. Letting p denote the current pass number in the learning process, the so-called learning rules are:
for the input layer:

$$c_i\big|_{p+1} = c_i\big|_p + \Delta c_i\big|_p \quad \text{where} \qquad \Delta c_i\big|_p = \left.\frac{\partial U_x}{\partial c_i}\right|_p$$

$$\theta_i\big|_{p+1} = \theta_i\big|_p + \Delta\theta_i\big|_p \quad \text{where} \quad \Delta\theta_i\big|_p = \left.\frac{\partial U_x}{\partial \theta_i}\right|_p$$

for the first internal representation layer:

$$c_j\big|_{p+1} = c_j\big|_p + \Delta c_j\big|_p \quad \text{where} \quad \Delta c_j\big|_p = \left.\frac{\partial U_y}{\partial c_j}\right|_p$$

$$\theta_j\big|_{p+1} = \theta_j\big|_p + \Delta\theta_j\big|_p \quad \text{where} \quad \Delta\theta_j\big|_p = \left.\frac{\partial U_y}{\partial \theta_j}\right|_p$$

$$C_{ij}\big|_{p+1} = C_{ij}\big|_p + \Delta C_{ij}\big|_p \quad \text{where} \quad \Delta C_{ij}\big|_p = \left.\frac{\partial U_y}{\partial C_{ij}}\right|_p$$

$$D_{ii'j}\big|_{p+1} = D_{ii'j}\big|_p + \Delta D_{ii'j}\big|_p \quad \text{where} \quad \Delta D_{ii'j}\big|_p = \left.\frac{\partial U_y}{\partial D_{ii'j}}\right|_p$$

and, for the second internal representation layer:

$$c_k\big|_{p+1} = c_k\big|_p + \Delta c_k\big|_p \quad \text{where} \quad \Delta c_k\big|_p = \left.\frac{\partial U_z}{\partial c_k}\right|_p$$

$$\theta_k\big|_{p+1} = \theta_k\big|_p + \Delta\theta_k\big|_p \quad \text{where} \quad \Delta\theta_k\big|_p = \left.\frac{\partial U_z}{\partial \theta_k}\right|_p$$

$$C_{jk}\big|_{p+1} = C_{jk}\big|_p + \Delta C_{jk}\big|_p \quad \text{where} \quad \Delta C_{jk}\big|_p = \left.\frac{\partial U_z}{\partial C_{jk}}\right|_p$$

$$D_{jj'k}\big|_{p+1} = D_{jj'k}\big|_p + \Delta D_{jj'k}\big|_p \quad \text{where} \quad \Delta D_{jj'k}\big|_p = \left.\frac{\partial U_z}{\partial D_{jj'k}}\right|_p$$

The detailed derivative formulas are given in Appendix IV.

The above assumes that the architecture variables are independent of each other which is not necessarily the case.

Moreover the U_z function, also, depends on the sigmoid parameters of the y-layer (the c_j and θ_j) as well as on the couplings between the x-layer and the y-layer; consequently the first order corrections for the architecture variables might better be taken as:

$$\Delta c_j = g\,\frac{\partial U_y}{\partial c_j} + f\,\frac{\partial U_z}{\partial c_j}$$

$$\Delta \theta_j = g\,\frac{\partial U_y}{\partial \theta_j} + f\,\frac{\partial U_z}{\partial \theta_j}$$

$$\Delta C_{ij} = g\,\frac{\partial U_y}{\partial C_{ij}} + f\,\frac{\partial U_z}{\partial C_{ij}}$$

$$\Delta D_{ii'j} = g\,\frac{\partial U_y}{\partial D_{ii'j}} + f\,\frac{\partial U_z}{\partial D_{ii'j}}$$

where g and f are control parameters. These corrections may eventually contain more terms, depending on the number of internal representation layers. The umbrella function of the z-layer depends on the couplings from the y-layer to the z-layer as well as on the couplings from the x-layer to the y-layer consequently the theory should contain all those corrections with $g = f = 1$. From a biological point of view the f-terms actually represent a kind of local feedback in the sense that the behaviour of the z-layer feeds back some information for the correction of the connections between the x-layer and the y-layer; that may be some sort of (local) modulation and if so it certainly requires a control parameter.

In Chapter IV the effects of this type of feedback, for the linear corrections only and feeding back only one layer, are discussed. With an architecture having four successive internal representation layers named a to d, the linear coupling correction for layer a to layer b, should be of the form:

$$\Delta C_{a \to b} = g \frac{\partial U_b}{\partial C_{ab}} + f \frac{\partial U_c}{\partial C_{ab}} + h \frac{\partial U_d}{\partial C_{ab}}$$

My studies would apply to the case $f \neq 0, g = 1, h = 0$ as well as $f \neq 0, g = 0, h = 0$.

II-3 Output protocols

Since the umbrella algorithm does not use any desired output whatsoever, the system does not spontaneously produce any kind of significant output. This is completely natural because the algorithm is designed to construct an internal representation as coherent with the problem at hand as possible. To generate an output with this device, one needs to "read" or to interpret the internal representation; that is the purpose of the output protocol. Three protocols were used: the "zero protocol", the "comparison protocol" and the "calculated perceptron protocol". The last two protocols were used in most simulations and applications.

II-3-1 The zero protocol

Used in the early stages of the development of the algorithm, it consist simply in taking the state of a (chosen) neuron in the final internal representation layer as "the" output. The two classes are taken to be signalled by the neuron state being above or below its threshold. The choice of the neuron was made on the basis of the least error on the learning set or on some test set. In some simple problems the zero protocol was found to be sufficient. It is called the "zero" protocol because it requires no extra output layer.

II-3-2 Comparison protocol

The idea is to use the R^{00} and R^{11} function values obtained for the learning set R^{00}_{LS} and R^{11}_{LS} . The protocol for determining to which class a new configuration belongs, is as follows:

(1) Calculate the R^{00} function with the LS plus the new configuration assumed to belong to the class 0; call it R^{00}_{0}.

(2) Calculate the R^{11} function with the LS plus the new configuration assumed to belong to the class 1; call it R^{11}_{1}.

(3) Compare $(R_0^{00} - R_{LS}^{00})$ and $(R_1^{11} - R_{LS}^{11})$; the larger determines the class to which the new configuration is assigned.

II-3-3 Calculated perceptron

The idea here is that if the algorithm has been succesful in building an internal representation significantly coherent with the problem, the internal representation, at some deep enough layer, will be linearly separable and a single cell perceptron will suffice to produce the desired output. For that purpose, define the center of the class as the class average vector. The calculation actually projects the (new) internal representation vector on an axis that goes through the two centers of the classes (in the internal representation where it is connected); the sign of the projection determines the class. A further interpretation is required since the sign is not predictable; the association "class-to-sign" can be verified with learning set examples.

These output protocols can be applied at any layer; they are connected to the last learning layer for most of the measurements presented. The comparison protocol, less demanding on the linear separation of the internal representation, is probably more efficient in more difficult problems or, simply, earlier in the learning process. In the long run the two protocols should produce the same output, indicating that the problem has been linearly separated.

II-3-4 Comparison and calculated perceptron protocols in the asymptotic regime

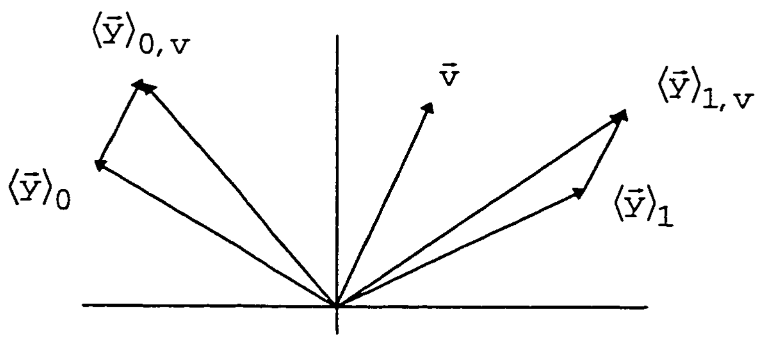
In the asymptotic regime (see section II-4-10), the angle between the two class average vectors is large (> $\pi/2$).; Since R^{00} and R^{11}, the similarity measures (for layer y, say) are given by

$$R^{\alpha\alpha} = \frac{1}{d_y}\left|\langle\vec{y}\rangle_\alpha\right|^2,$$ then

with the adjunct vector $\vec{v}$ added to class α we have for the new class average

$$\langle\vec{y}\rangle_{\alpha,v} = \frac{N^\alpha\langle\vec{y}\rangle_\alpha}{N^\alpha + 1} + \frac{1}{N^\alpha + 1}\vec{v}$$

which is to be used in place of $\langle\vec{y}\rangle_\alpha$ above to give $R^{\alpha\alpha}$.



$\langle\vec{y}\rangle_{0,v}$
$\vec{v}$
$\langle\vec{y}\rangle_{1,v}$
$\langle\vec{y}\rangle_0$
$\langle\vec{y}\rangle_1$

Figure II-5 Output protocol comparison

In the asymptotic regime, a vector $\vec{v}$ having a positive (or larger) component $\vec{v}\,.\langle\vec{y}\rangle_\alpha$ along one class average (class 1 in figure II-5) will have a negative (or smaller) component along the other class average (class 0 in Figure II-5) and consequently the variation of the R will be larger for the class having its average vector closer (angularly) to $\vec{v}$, the vector to be classified.

In the limit, that is when the two class vectors point in opposite directions, the comparison protocol will cut the whole configuration space by a plane perpendicular to the common direction of the two (limit) average vectors; and that is exactly the same as the construction of the calculated perceptron. So in the limit the two protocols will not only produce the same output but they will also produce the same generalization; this has been observed in the simulations (see Chapter IV).

II-4 Comments and Justifications

The present section collects various theoretical arguments that led to the crucial choices in the definitions and the construction of the algorithm.

II-4-1 About the sigmoid function

If the problem is linearly separable (see Figure II-6; the figure is drawn in two dimensions but the argument applies to any number of dimensions),

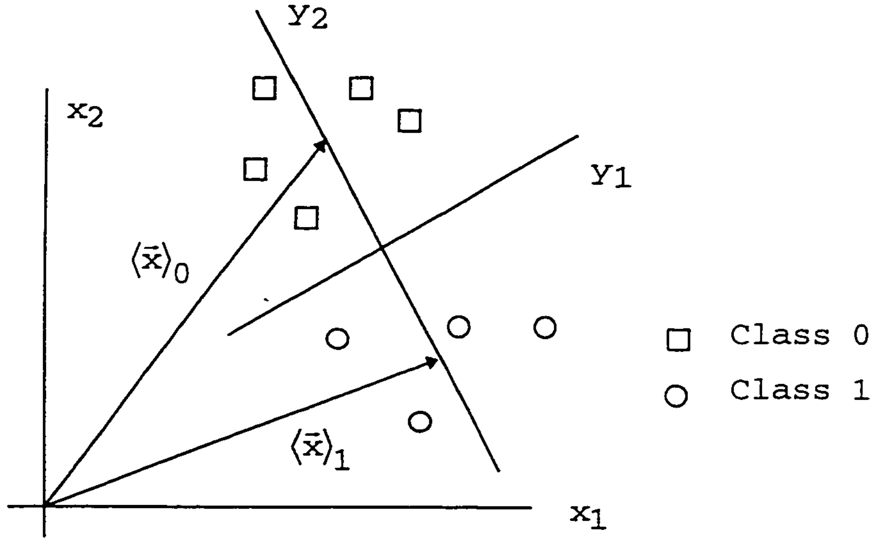


Figure II-6 Internal representation without sigmoid compression

there exists at least one straight line (the y_1 axis in the figure II-6) or a plane or hyper-plane in higher dimensions, that separates the two classes. One can then choose a new coordinate system of two dimensions only, regardless of the original dimensions of the problem: y_1 in the separator line or plane and y_2 perpendicular to the separator. The mapping: $\mathbf{x} \rightarrow \mathbf{y}$ by rotation and translation will solve the problem; the sign of the y_2 component will provide the class.

Further, if the input space and the internal representation space are measured in the same units, then the U-function, being the square of the distance between the two average vectors will not be modified and one has:

$$U_x = \frac{1}{d_x}\left[\langle\vec{x}\rangle_0 - \langle\vec{x}\rangle_1\right]^2 = \frac{1}{d_y}\left[\langle\vec{y}\rangle_0 - \langle\vec{y}\rangle_1\right]^2 = U_y$$

(the average vectors are calculated without the use of the sigmoid functions here).

To have $U_x < U_y$ one needs the mapping to do some dilation or compression in a non-linear fashion. If one applies a sigmoid function with an appropriate constant and zero threshold on the y_2 coordinate the two classes will tend to spread apart.

To be able to use a gradient search for the solution, one needs a differentiable signal function, preferably bounded; the sigmoid is a classical candidate in the field. Its asymptotic behaviour is a necessary feature for the umbrella algorithm to converge. It may also trivially avoid floating point overflow which is a danger when the number of connections becomes large.

Support for the existence of a sigmoid function is provided by the Leshno theorem: multilayer feedforward networks with a nonpolynomial activation function can approximate any function (Leshno et al. 1995), the activation function being the actual (neuron) output calculated after the neuron state. The sigmoid is one of the non-polynomial favorites in the field even if, in computer simulations, it is approximated by some polynomial.

And finally, use of the sigmoid function is encouraged by the fact that artificial neural networks are, in some sense, based on biological systems (see Appendix I-2 for details), and the sigmoid may be very close to a real neuron activation function (Kohonen 1989, Anderson et al. 1989).

II-4-2 About the mapping

To show that the argument of the previous section works for nonlinearly separable problems, let us look at a simple one dimensional problem: $d_x=d_y=1$. The input layer has one cell called x and the first internal representation layer has one cell named y. Let the learning set contain two examples in each of the two classes, indicated by squares and circles, Figure II-7.

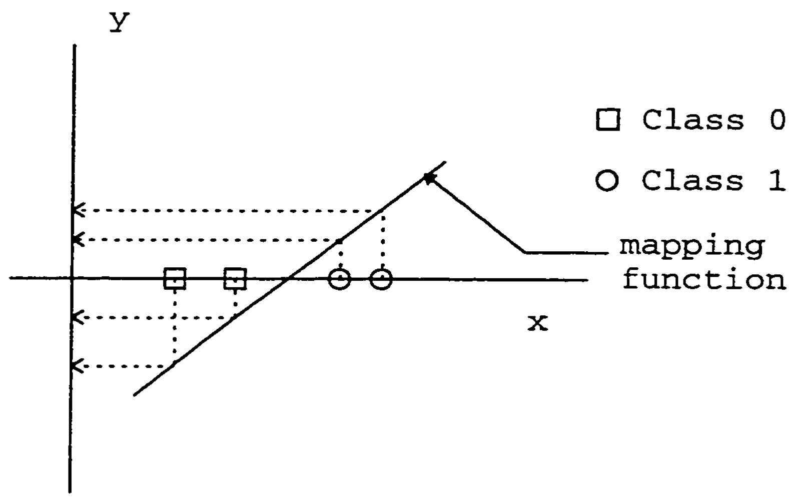


Figure II-7 linearly separable problem.

For linearly separable problems, an affine mapping is sufficient to separate the classes.

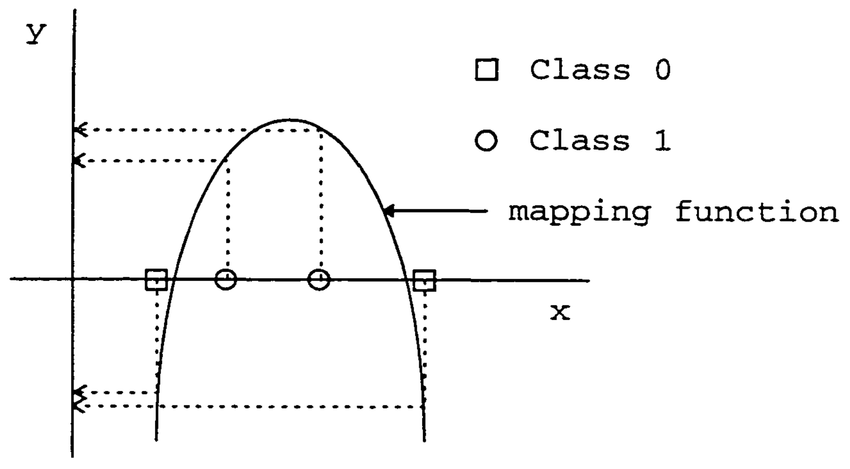


Figure II-8 "Quadratically" separable problem.

For non linearly separable problems, more powerful mapping may be required: products or higher powers of the inputs signals may be necessary to properly separate the classes.

The sign of the y component used to discriminate between the classes is the idea behind my calculated perceptron in the output protocol section; it can also be seen as an extension of the so-called singular value decomposition in standard statistics.

II-4-3 The EXOR example

This is another example to show that the product of the inputs may be necessary to discriminate between non linearly separable classes.

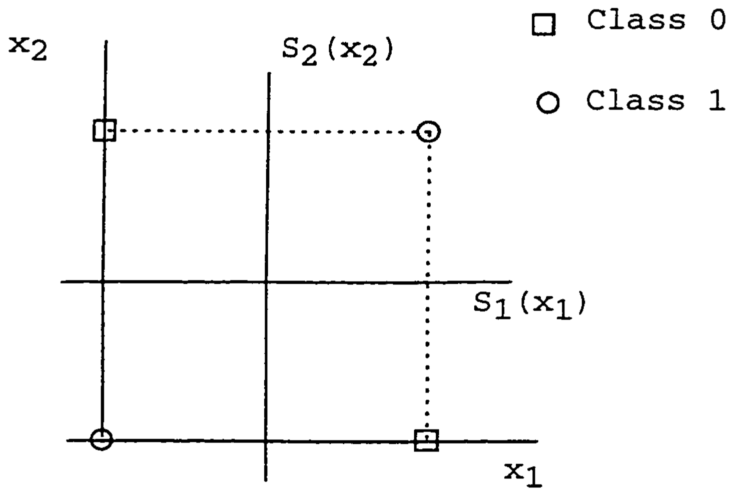


Figure II-9 the EXOR problem

Figure II-9 and Table II-1 present the EXOR problem. The first three columns of the table are the usual truth table for the exclusive OR problem; the fourth and the fifth are possible signal function outputs of the input and layer cells;

Table II-1 EXOR problem

x_1	x_2	Class	$S_1(x_1)$	$S_2(x_2)$	$S_1(x_1)S_2(x_2)$
0	0	0	-1/2	-1/2	+1/4
0	1	1	-1/2	+1/2	-1/4
1	0	1	+1/2	-1/2	-1/4
1	1	0	+1/2	+1/2	+1/4

and the sign of the product of the signals (column six) may be used to discriminate correctly between the classes.

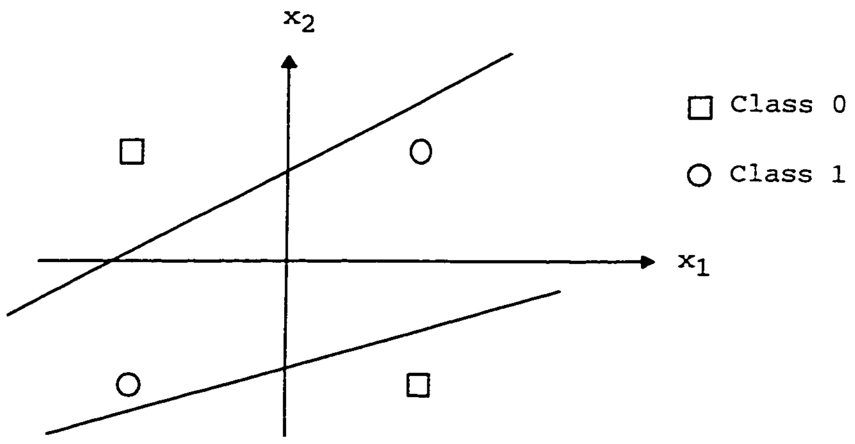


Figure II-10 EXOR is not linearly separable in 2D

From Figure II-10 we see that the EXOR is not linearly separable in 2D. But it is in three, as can be seen from Figure II-11.

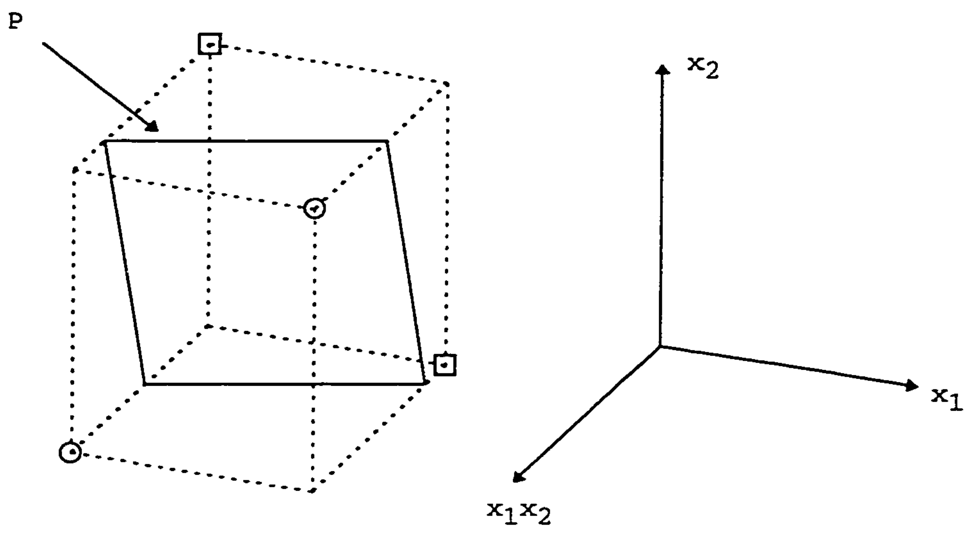


Figure II-11 EXOR can be linearly separated in 3D

Any plane, like P, nearly parallel to the plane of the axes x1 and x2, cutting the cube, will linearly separate the classes.

Thus, the additional variables labelling the added dimensions can be products of the original problem variables.

A similar observation, regarding the products, can be seen in two dimensions: clouds of points may not be separable by a straight line but may be separated by a parabola; that too amounts to products of the variables as additional inputs of the problem. It is then natural to say: any non linearly separable problem in n dimensions can be linearly separated in m dimensions, with m>n; and, the dimensions to add are some appropriately chosen products of the original n coordinates. The appropriateness of the products chosen is called the power of the mapping and is discussed elsewhere.

II-4-4 About the learning rates

It is useful, in practice, to use different learning rates for the different types of architecture variables. This is also called "the heuristic rule number one: every adjustable network parameter of the cost function should have its own individual learning rate" (Haykin 1994, p193). It was observed, in the simulations, that the thresholds should be adjusted more slowly than the linear couplings in order to have faster learning. In fact there are four umbrella functions in the code: one for the sigmoid constants, one for the thresholds, one for the linear couplings and one for the non linear couplings (for each internal representation layer). There are also the two umbrella functions for the input: one for the sigmoid constants and one for the thresholds. Consequently the **monitoring umbrella function** was defined, for observation purposes, with $\beta = 1$ and $\gamma = -2$, its asymptotic value being 4. The discrepancy, during the learning process, between the current value of the umbrella function and its limiting value is a powerful indicator of the quality of the learning. In the result sections the monitoring umbrella functions divided by 4 are presented as a function of p, the number of passes.

With general learning rates β and γ an actual umbrella function is:

$$U_Y = \frac{1}{d_Y}\left[\beta\langle\bar{y}\rangle_0^2 + \gamma\langle\bar{y}\rangle_0 \cdot \langle\bar{y}\rangle_1 + \beta\langle\bar{y}\rangle_1^2\right]$$

which can be written:

$$U_Y = \frac{\beta}{d_Y}\left[\langle\bar{y}\rangle_0^2 - 2\langle\bar{y}\rangle_0 \cdot \langle\bar{y}\rangle_1 + \langle\bar{y}\rangle_1^2 + \left(\frac{\gamma}{\beta} + 2\right)\langle\bar{y}\rangle_0 \cdot \langle\bar{y}\rangle_1\right]$$

and finally:

$$U_Y = \frac{\beta}{d_Y}\left[\left(\langle\bar{y}\rangle_0 - \langle\bar{y}\rangle_1\right)^2 + \left(\frac{\gamma}{\beta} + 2\right)\langle\bar{y}\rangle_0 \cdot \langle\bar{y}\rangle_1\right]$$

where one can see now that the first term depends on the distance between the two average vectors and the second term depends on the angle between these two average vectors. Since the dot product hides a cosine, and since the goal is to have the U-function as large as possible when the two average vectors are pointing as far away (from each other) as possible, the coefficient in the second term should be negative: $\frac{\gamma}{\beta} + 2 < 0$. Stated another way, to fulfill the condition on the similarity measures, one should have the learning rates satisfy $\gamma < -2\beta$ in order to put emphasis on the separation of the classes relative to their internal compaction.

II-4-5 About the asymptotic value of the U functions

To explore the limiting value of the U-functions let us write:

$$R_Y^{\alpha\alpha'} = \frac{1}{d_Y}\langle\vec{y}\rangle_\alpha \cdot \langle\vec{y}\rangle_{\alpha'} = \frac{1}{d_Y}\left\|\langle\vec{y}\rangle_\alpha\right\| \left\|\langle\vec{y}\rangle_{\alpha'}\right\| \cos(\varphi)$$

where φ is the angle between the two vectors $\langle\vec{y}\rangle_\alpha$ and $\langle\vec{y}\rangle_{\alpha'}$; in the asymptotic regime one can assume $\cos(\varphi) \le 0$ for $\alpha \ne \alpha'$. Since the components of the $\langle\vec{y}\rangle_\alpha$ vectors are bounded, by the sigmoid functions, to lie in the range between -1 and +1, the norms $\left\|\langle\vec{y}\rangle_\alpha\right\|$ of the $\langle\vec{y}\rangle_\alpha$ vectors are bounded by $\sqrt{d_Y}$; the similarity measures are themselve bounded by -1, which occurs when the angle between the two vectors is π, or by +1 when $\alpha = \alpha'$; and the umbrella functions, for two classes: $U_Y = \beta(R_Y^{00} + R_Y^{11}) + \gamma R_Y^{01}$ are bounded above (the U functions are positive and their lower bound is simply 0; it is not used here) by $2\beta - \gamma$. The same argument applied to the monitoring functions lead to $0 \le W \le 4$ for any layer.

II-4-6 About the behaviour of the U-function during single layer learning

Single layer learning here means that among the coefficients f, g and h at the end of Section II-2, only one is positive. Then, for any architecture variable v and a U-function depending on v, one has:

$$U(v + \Delta v) = U(v) + \frac{\partial U}{\partial v}\Delta v + \cdots$$

where higher order terms are neglected. Since the learning rule for any architecture variable is $\Delta v = \frac{\partial U}{\partial v}$ one has

$U(v + \Delta v) = U(v) + \left(\frac{\partial U}{\partial v}\right)^2$ and $U(v + \Delta v) \geq U(v)$. Since the architecture variables are corrected at each pass, $v|_{p+1} = v|_p + \Delta v$ and finally $U|_{p+1} \geq U|_p$ where p is the current pass number. That is: during the learning process, provided that the corrections (the learning rates) have appropriate values, and if only one architecture variable is corrected at each pass, the umbrella function will never decrease and will eventually reach a local extremum. The learning rates are obtained by trial and error. When they are "too small" the system W-functions hardly change and when they are "too large", the W-functions behave erratically.

II-4-7 About the error

The name **zero error** is here given to the state of the learning at and after which all the configurations of the learning set are assigned to the desired class by the chosen output protocol.

The typical behaviour of the complete algorithm, umbrella plus output protocol, is: when the value of the umbrella function is high enough and zero error has been reached, the error remains zero. This has been observed in the simulations and will now be justified.

Assumptions:

(i) the mapping is powerful enough to separate the classes

(ii) the **error** (the number of configurations from the learning set that are misclassified) is zero and

(iii) the monitoring umbrella function of the last internal representation (to which the output protocol is linked) is in its asymptotic regime.

Claim: the error has reached its absolute minimum

Argument: the error being used here is an integer-valued function, which can therefore not be differentiated; one may still use the argument that the error value shall not be affected by small variations of architecture variables near its extremum points. Since the umbrella function is in its asymptotic regime, its values will not be affected by a small variation of any of the architecture variables meaning that the internal representation will vary very little and consequently the error will not change. So the error is not affected by small changes of the architecture variables and since it is zero (it cannot be negative) it is at its absolute minimum.

II-4-8 About generalization

In the context of artificial neural networks, generalization is measured (after a learning session) by presenting input configurations that do not appear in the learning set but have a well-known class (this is usually called a test set). The quality of the generalization can be measured by the percentage of properly classified examples.

In the literature, a learning session is usually terminated when the error level has reached a small (non-zero), predetermined value. The justification for the small non zero error is that, for a large majority of learning algorithms, the network may over-specialize on the learning set; it tends to behave like an exact memory thereby losing its generalization capability (see, e.g. Hecht-Nielsen 1989, paragraph 5.1.3; Haykin 1995 pp 180-181).

The umbrella algorithm cannot over-specialize. When zero error has been reached, the class regions (as defined by the learning set) have no overlap: there is no vector belonging to class 0 in the class 1 region and vice versa, as in Figure II-12. This happens whenever the learning process is successful (i.e. whenever the learning rates are such that the U-function continues to increase). Let us consider, at some given pass during the learning when zero error has been reached, the angle between the two class average vectors. As the learning proceeds, the U function increases and that angle increases (Figure II-12).

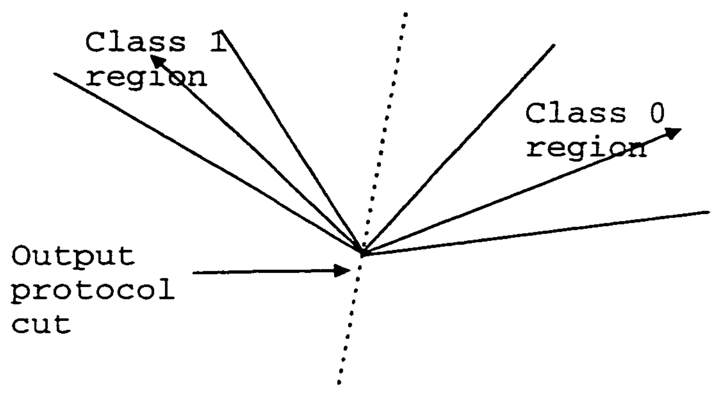


Figure II-12 Generalization in internal representation space.

The calculated perceptron establishes a cut (plane or hyperplane) which divides the internal representation space into two classes. (It will assign a class to each new configuration according to which side of the cut the last internal representation vector falls.) The cut actually moves during the learning process and the generalization changes accordingly. All the vectors in the internal representation will be assigned to a class, always, so the actual absolute generalization is actually constant. There is an **ambiguous region** which includes the protocol cut that may result in the wrong class assignment for some vectors in that region. As the angles of the class regions decrease (with increasing U-function), the ambiguous region around the output protocol cut narrows down, and in the limit the ambiguous region will become a plane. That is what happens when the W-function reaches 3.99 as it does for the EXOR (in Chapter IV).

The quality of the generalization will depend only on the faithfulness of the learning set, provided that:

(i) The zero error has been reached.

(ii) U-function value is high, so that there is no overlapping of the class regions (see Figure II-12);

The originality of the umbrella algorithm lies precisely in not choosing an *a priori* distance function. The umbrella algorithm tries to maximize the distance between the classes (in the internal representation) and minimize the distance between internal representation vectors of events belonging to the same class.

The quality of the generalization depends on the learning set in the following manner. If the learning is successful (i.e. if the zero error has been reached and the U function of the last internal reresentation layer is high enough), the comparison and the calculated protocols will cut the input configuration space into two regions each assigned to a class. The significance of the generalization in that context depends on the position of the boundary between the two class regions. And the position of the boundary depends (in the case of succesful learning) on the appropriateness of the learning set in representing the problem to be solved.

II-4-9 About supervision

The present approach can be thought of as lying somewhere between the "supervised" and "unsupervised", in the following sense. It cannot be called supervised because there is no supervisor to impose a predetermined output for a given input; there is no "error" function to be estimated throughout the entire learning process. It cannot be called unsupervised either, because the learning device actually uses information about input data: the system has to "know" that some input configurations are to be classified together while others are to be classified separately. The supervision occurring in the umbrella algorithm is rather in the nature of telling the system "same as" or "not same as" rather than "right" or "wrong". It may well be called a form of the reinforcement type of supervision (Hecht-Nielsen 1989).

II-4-10 About the convergence and the asymptotic regime

The present section contains a discussion of the convergence of the umbrella algorithm in the context of a single internal representation layer with a single cell called ω, taken as the output. It is also assumed that

$$\omega = \sum_{i=1}^{d_x} C_i S_i(x_i) + \sum_{i=1}^{d_x} \sum_{i'=i}^{d_x} D_{i\,i'}\, S_i(x_i)\, S_{i'}(x_{i'})$$

can be written:

$$\omega = \sum_{\iota=1}^{n} E_\iota \chi_\iota$$

with the enlarged inputs and couplings:

$$\chi_\iota = S_i(x_i)\,, \quad E_\iota = C_i \quad \text{for} \quad \iota = i = 1, \ldots, d_x$$

$$\chi_\iota = S_i(x_i)\, S_{i'}(x_{i'})\,, \quad E_\iota = D_{i\,i'} \quad \text{for} \quad \begin{cases} i = 1, \ldots, d_x \\ i' = i, \ldots, d_x \\ \iota = d_x + 1, \ldots, n \end{cases}$$

The original input vectors of the problem are supplemented by the required products. If d_x is the original number of inputs in the problem, then $n=d_x$ if the problem is linearly separable and $n>d_x$ if the problem is not linearly separable. If the problem is not linearly separable, the coefficients E_ι for $\iota>d_x$ correspond to the $D_{i\,i'}$.

The linearised system has the architecture of a classical perceptron for which the convergence is well established (see Appendix VI).

To analyse the convergence of the umbrella learning rule, define:

Learning set: Let $\{\bar{\chi}^{\kappa}\}$ and $\{\bar{\chi}^{\lambda}\}$ be input vector examples representing the classes Γ^0 and Γ^1; κ and λ are indices running from 1 to N^0 and N^1 respectively.

Umbrella definitions

Applying the umbrella definitions to the linearised system provides:

$$\omega^{\kappa} = \sum_{\iota=1}^{n} E_{\iota}\chi_{\iota}^{\kappa} \text{ and } . \ \omega^{\lambda} = \sum_{\iota=1}^{n} E_{\iota}\chi_{\iota}^{\lambda}$$

Let: $\langle S_{\omega}(\omega)\rangle_0 \equiv \frac{1}{N^0} \sum_{\kappa\in\Gamma^0} S_{\omega}(\omega^{\kappa})$ and $\langle S_{\omega}(\omega)\rangle_1 \equiv \frac{1}{N^1} \sum_{\lambda\in\Gamma^1} S_{\omega}(\omega^{\lambda})$. Then

$R^{00} = \langle S_{\omega}(\omega)\rangle_0^2$, $R^{11} = \langle S_{\omega}(\omega)\rangle_1^2$, $R^{01} = \langle S_{\omega}(\omega)\rangle_0\langle S_{\omega}(\omega)\rangle_1$ and, with learning rates $\beta = 1$ and $\gamma = -2$,

$$U_{\omega} = V_{\omega}^2 \text{ where } V_{\omega} \equiv \left[\langle S_{\omega}(\omega)\rangle_0 - \langle S_{\omega}(\omega)\rangle_1\right].$$

The correction of the (now) linear coefficients in going from pass p to pass p+1:

$$E_{\iota}\Big|_{p+1} = E_{\iota}\Big|_{p} + \frac{\partial U_{\omega}}{\partial E_{\iota}}\bigg|_{p} \quad \text{leads to}$$

$$U_{\omega}\Big|_{p+1} = U_{\omega}\Big|_{p} + \sum_{\iota} \frac{\partial U_{\omega}}{\partial E_{\iota}}\bigg|_{p} \Delta E_{\iota} = U_{\omega}\Big|_{p} + \sum_{\iota}\left(\frac{\partial U_{\omega}}{\partial E_{\iota}}\bigg|_{p}\right)^2$$

which ensures that $U_\omega\big|_{p+1} \geq U_\omega\big|_p$; and, provided that the correction is small enough, the U-function cannot decrease.

The equality holds when $\sum_\iota \left(\frac{\partial U_\omega}{\partial E_\iota}\bigg|_p \right)^2 = 0$, that is when all the derivatives vanish individually: $\frac{\partial U_\omega}{\partial E_\iota}\bigg|_p = 0 \ \forall \iota$.

$$\frac{\partial U_\omega}{\partial E_\iota}\bigg|_p = 2V_\omega\big|_p \frac{\partial V_\omega}{\partial E_\iota}\bigg|_p$$

and

$$\frac{\partial V_\omega}{\partial E_\iota}\bigg|_p = \frac{\partial}{\partial E_\iota}\Big[\langle S_\omega(\omega)\rangle_0 - \langle S_\omega(\omega)\rangle_1\Big]_p$$

$$= \frac{\partial}{\partial E_\iota}\left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0} S_\omega\!\left(\sum_{\iota=1}^{n} E_\iota \chi_\iota^\kappa\right) - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1} S_\omega\!\left(\sum_{\iota=1}^{n} E_\iota \chi_\iota^\lambda\right)\right]_p$$

$$= \left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0} \frac{\partial S_\omega(\omega)}{\partial \omega}\bigg|_{\omega^\kappa} \frac{\partial}{\partial E_\iota}\sum_{\iota=1}^{n} E_\iota \chi_\iota^\kappa - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1} \frac{\partial S_\omega(\omega)}{\partial \omega}\bigg|_{\omega^\lambda} \frac{\partial}{\partial E_\iota}\sum_{\iota=1}^{n} E_\iota \chi_\iota^\lambda\right]_p$$

$$\frac{\partial V_\omega}{\partial E_\iota}\bigg|_p = \left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0} \frac{\partial S_\omega(\omega)}{\partial \omega}\bigg|_{\omega^\kappa} \chi_\iota^\kappa - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1} \frac{\partial S_\omega(\omega)}{\partial \omega}\bigg|_{\omega^\lambda} \chi_\iota^\lambda\right]_p$$

$$\Rightarrow \left.\frac{\partial U_\omega}{\partial E_\iota}\right|_p = \left[\langle S_\omega(\omega)\rangle_0 - \langle S_\omega(\omega)\rangle_1\right]_p$$

$$\times \left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0}\left.\frac{\partial S_\omega(\omega)}{\partial\omega}\right|_{\omega^\kappa}\chi_\iota^\kappa - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1}\left.\frac{\partial S_\omega(\omega)}{\partial\omega}\right|_{\omega^\lambda}\chi_\iota^\lambda\right]_p$$

which can vanish in two ways:

(1) $\left[\langle S_\omega(\omega)\rangle_0 - \langle S_\omega(\omega)\rangle_1\right]\Big|_p = 0 = V_\omega \Rightarrow U_\omega = 0$

which is clearly a pathological case that does not apply to the umbrella algorithm; and

(2) $\left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0}\left.\frac{\partial S_\omega(\omega)}{\partial\omega}\right|_{\omega^\kappa}\chi_\iota^\kappa) - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1}\left.\frac{\partial S_\omega(\omega)}{\partial\omega}\right|_{\omega^\lambda}\chi_\iota^\lambda)\right]_p = 0$

There are two ways again that (2) can hold: (i) the two class averages cancel each other; this is a local trap, which can be monitored easily, and (ii) the two class averages vanish individually. This happens provided that all the derivatives $\frac{\partial S_\omega(\omega)}{\partial\omega}$ vanish, for all the examples of both classes. This is THE solution: all the output states for all the vectors of the two classes are very large, such that the sigmoids S_ω are very close to their asymptotic values (+1 or -1) and their derivatives: $\frac{\partial S_\omega(\omega)}{\partial\omega}$ approach zero.

It remains now to show how and why the output states ω for all input vectors of the two classes will be large in absolute value.

$$E_\iota\Big|_{p+1} = E_\iota\Big|_p + \left[\langle S_\omega(\omega)\rangle_0 - \langle S_\omega(\omega)\rangle_1\right]_p \times \left[\frac{1}{N^0}\sum_{\kappa\in\Gamma^0} \frac{\partial S_\omega(\omega)}{\partial\omega}\Big|_{\omega^\kappa} \chi_\iota^\kappa - \frac{1}{N^1}\sum_{\lambda\in\Gamma^1} \frac{\partial S_\omega(\omega)}{\partial\omega}\Big|_{\omega^\lambda} \chi_\iota^\lambda)\right]_p$$

After zero error has been reached (a zero protocol is implicitely assumed here) the sign of the first factor in the correction term will not change. That is because all the input vectors of one class will produce an output above the output threshold and the other class input vectors will produce output below the threshold; the two class averages, in that factor, will have opposite signs. Also, since the input vectors are constant (the learning set does not change during the learning process), and the (output) sigmoid is a monotonic function, the correction term (while diminishing) will have a constant sign from pass to pass. The absolute values of the couplings cannot decrease and therefore the absolute value of the output state ω cannot decrease. This is the asymptotic regime.

The sigmoid theorems in Appendix III further show that a finite number of steps will be required for the U-function (U_ω) to reach a predetermined value.

III - PROGRAM OUTLINES

The learning algorithm was implemented in two successive programs, and eventually a third one for detection. The first is the implementation of the umbrella algorithm itself; the second is the implementation of the output protocols. The first does the learning and the second does the measuring. I give here an outline of these programs. The detailed C-code of typical versions are given in Appendix VII. Some other programs were also required; e.g. the learning set files, the test set files, etc., had to be prepared by other programs. The plotting of the data was done by other software as well. These latter, called "service programs", are quite straightforward and are not given in Appendix VII.

For the sake of versatility as required by the research, the structure: program $\rightarrow$ file $\rightarrow$ program $\rightarrow$ file $\rightarrow$ program $\rightarrow$ file, etc. was followed.

The main program flowcharts are given in Figures III-1, III-2 and III-4, and are commented on thereafter.

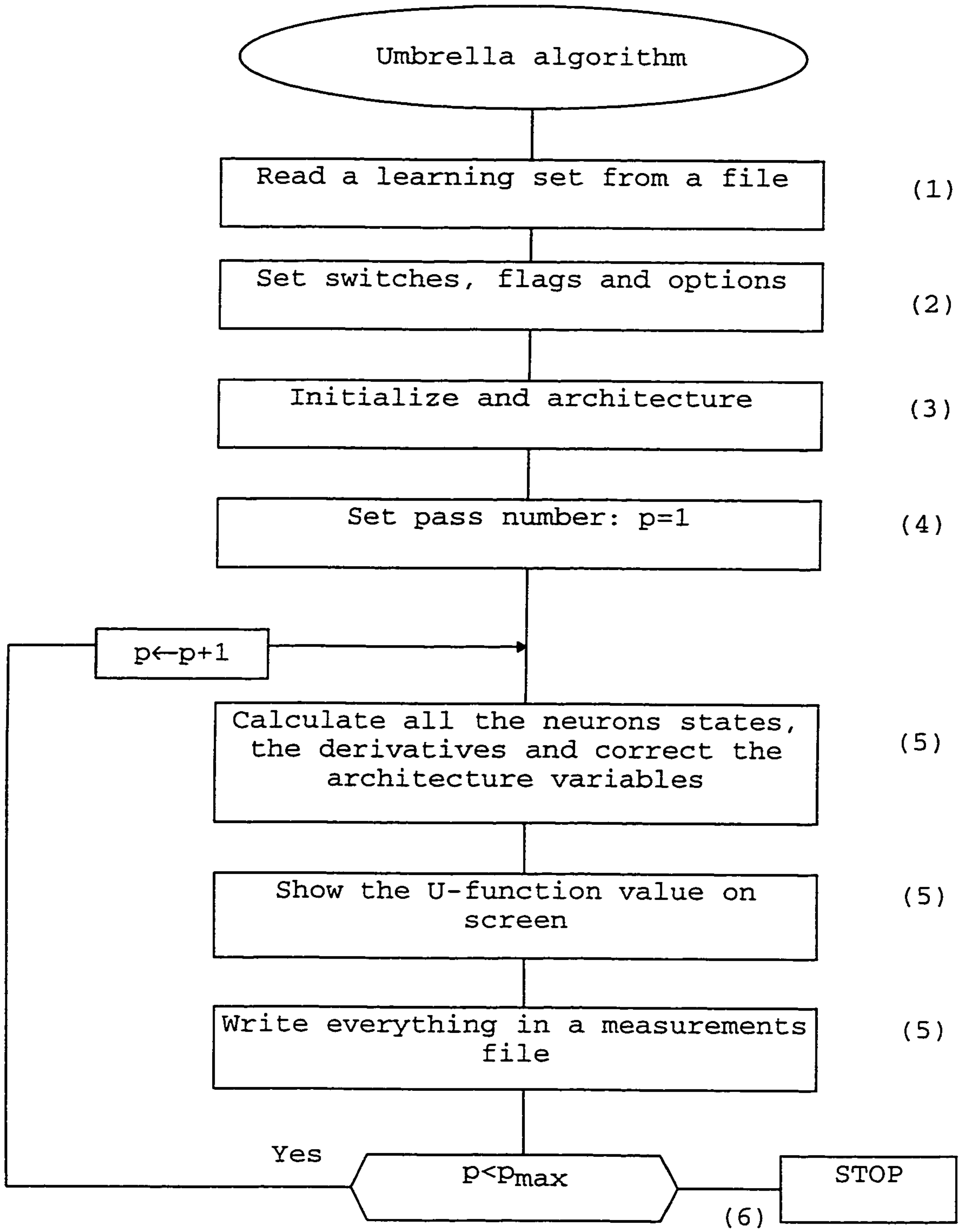

Umbrella algorithm
Read a learning set from a file
(1)
Set switches, flags and options
(2)
Initialize and architecture
(3)
Set pass number: p=1
(4)
p←p+1
Calculate all the neurons states, the derivatives and correct the architecture variables
(5)
Show the U-function value on screen
(5)
Write everything in a measurements file
(5)
Yes
p<Pmax
(6)
STOP

Figure III-1 Learning program flow chart

(1) The learning set files are either completely constructed artificially, as in the EXOR and Toy Data problems; or are real data files reorganised in a format readable by the program. This step required many service programs, which are not included in the appendix because they are straightforward and would easily fill one hundred pages.

(2) The user has many choices; a typical set is the following:

```
number of layers: 3 : 2-2-2
synchronisation: Y
normalization: N
center the problem: Y
density of non vanishing couplings: 1.0
f mixed products=1.0
f squares=0.0
decreasing learning rates with layer depth=0.0
maximal couplings=1.0
random seed=0.610000
learning rates
  sigmoid constants: β=0.0  γ=0.0
  thresholds: β= 0.50  γ=1.0
  linear couplings: β=1.0  γ=2.0
  non linear couplings: β=1.0  γ=2.0
two-layer bias= 10.000000
```

- Number of layers in the network (the internal representation is not limited to two in the code); layer dimensions.

- Set synchronous or asynchronous learning.

- Normalize or not normalize the inputs; this option was never used in the present research, being a remnant from earlier versions.

- Center the problem: "yes" means that the thresholds for the input layer will be set to the parameters values averaged over the whole learning set (as described in Section II-2); or they may be set randomly, or to zero.

- Set a certain percentage of the couplings (linear or nonlinear) to non-zero values; 1.0 means that all the cells of a layer are connected to all the cells of the following layer. This option, also a remnant from previous versions, was always set to 1.0. A value of less than 1.0 is nevertheless consistent with biological observations.

- The f switches for the squares and mixed products; if they vanish, their corresponding products of "previous layer" parameters do not exist in the network. In the example the mixed products exist but not the squares.

- Decreasing learning rates with layer depth: the learning rates may take the same values for all the layers (flag = 0.0) or they may be set to decrease with layer depth (flag > 0.0).

- Upper bound of randomly set architecture variables such as C's, D's etc.

- Learning rates; if they are set to zero the corresponding architecture variables are frozen. That is the case for the sigmoid constants in most of the simulations and applications performed.

- Two-layer bias: sets the f and g parameters (cf. Section II-2)

In addition the user may also:

- Standardize, or not, the inputs. Standardization here means setting the largest learning set vector norm (regardless of its class) to one unit and scaling all the other vectors (of both classes) accordingly. This option was written to handle properly cases of real data that take very large or very small values, and was not used in the present research.

- Set the n to sample the learning at every n passes.

- Set the maximum number of passes p_{max}

(3) Initialization:

- Set the sigmoid constants to some chosen value (such as 0.5), all of the same sign or not; or randomly.

- Set all other architecture variables randomly between the chosen bounds.

(4) Scan the learning set and for each input configuration.

- Calculate the states of all the neurons and everything else required to calculate the derivatives and the similarity measures.

When the learning set has been exhausted:

- Calculate the corrections using the learning rules

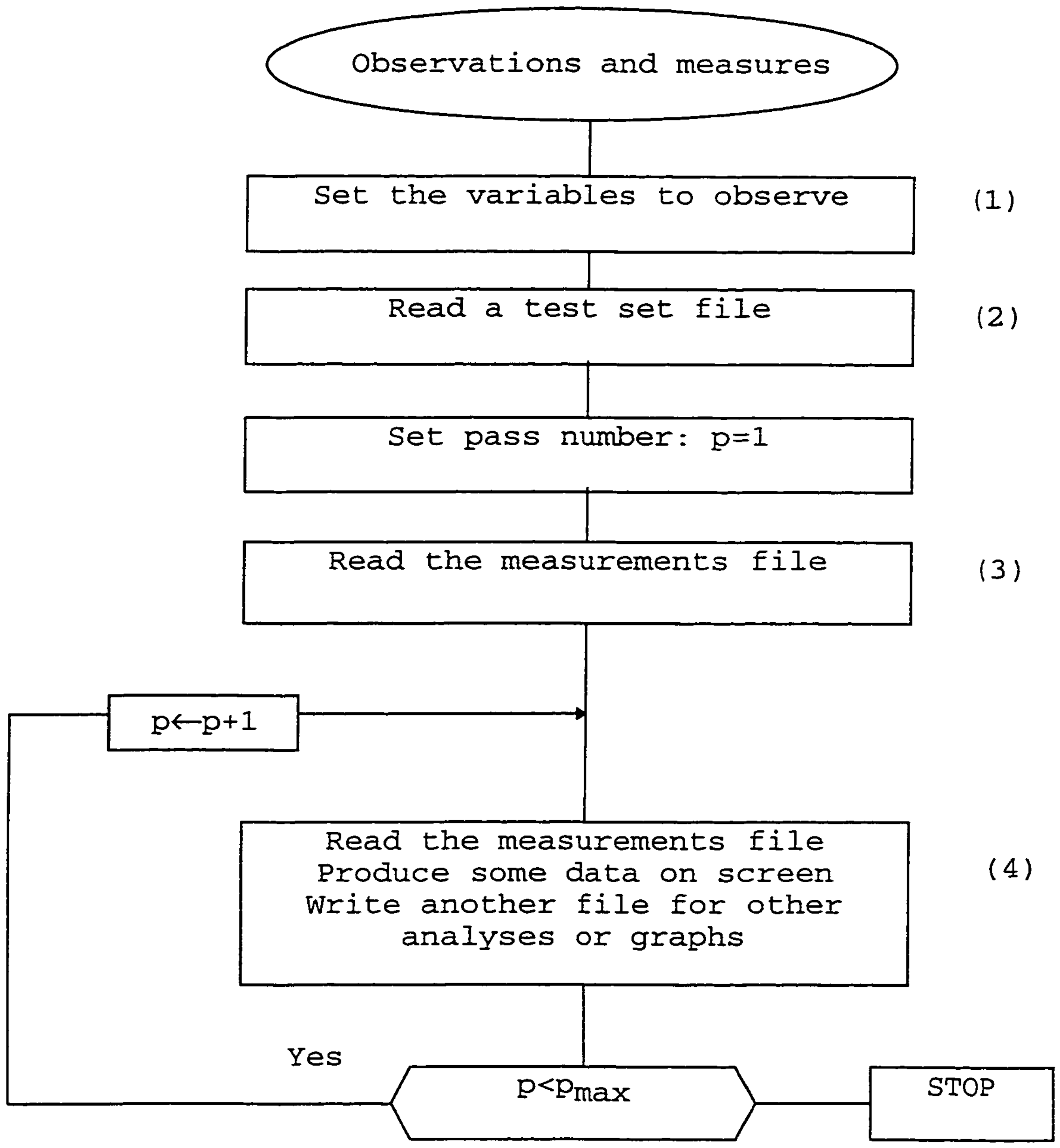
- Apply the corrections to the learning variables. In the case of asynchronous learning, only certain randomly chosen architecture variables are actually updated at each pass.

Step (4) constitutes one pass.

(5) If the pass number has reached a multiple of some chosen value, the similarities and the monitoring U-functions are presented on the screen allowing the trained user to follow the learning. The program also writes all the architecture variables and the similarity measures in the measurements file for further analysis.

(6) The program can be terminated by the classical panic button CTRL-C or, if all goes well, when the preset number of passes p_{max} is reached.

Many different measuring programs have been set up, which actually implement the various output protocols. Their essential structure is given in Figure III-2.



Observations and measures
Set the variables to observe
(1)
Read a test set file
(2)
Set pass number: p=1
Read the measurements file
(3)
p←p+1
Read the measurements file
Produce some data on screen
Write another file for other analyses or graphs
(4)
Yes
p<pmax
STOP

Figure III-2 Measurements program

(1) The user can choose to observe any architecture variable versus the number of passes or even one architecture variable versus another. Or, to observe the response of the architecture to a **test set.** A test set

contains input configurations from both classes and, for each configuration, the class to which it belongs. This is exactly the same information and file structure as for a learning set; it differs just in the number of examples, being about 50 for the LS and about 1000 for the TS. New measures appear here, such as the number of errors and the generalization. Again, any measure can be observed as a function of pass number.

(2) Read a test set constructed by a previous service program. The test sets as well as the learning sets can either be completely constructed as in the EXOR and Toy Data problems, or can originate from real physics experiments such as the astrophysics problem.

For problems in two dimensions it is possible to scan the entire input space rather than using test set data; this provides the representation maps, such as those for the EXOR problem.

(3) In this step the program scans the test set, calculates all the neuron states and applies the output protocols in order to determine the output, which is then compared with the desired class to provide the error count.

(4) Used for further studies or future programs.

A typical set of measurements done on a learning process is give in Figure III-3. On the horizontal axis is the number of passes and on the vertical axis are the normalized measures.

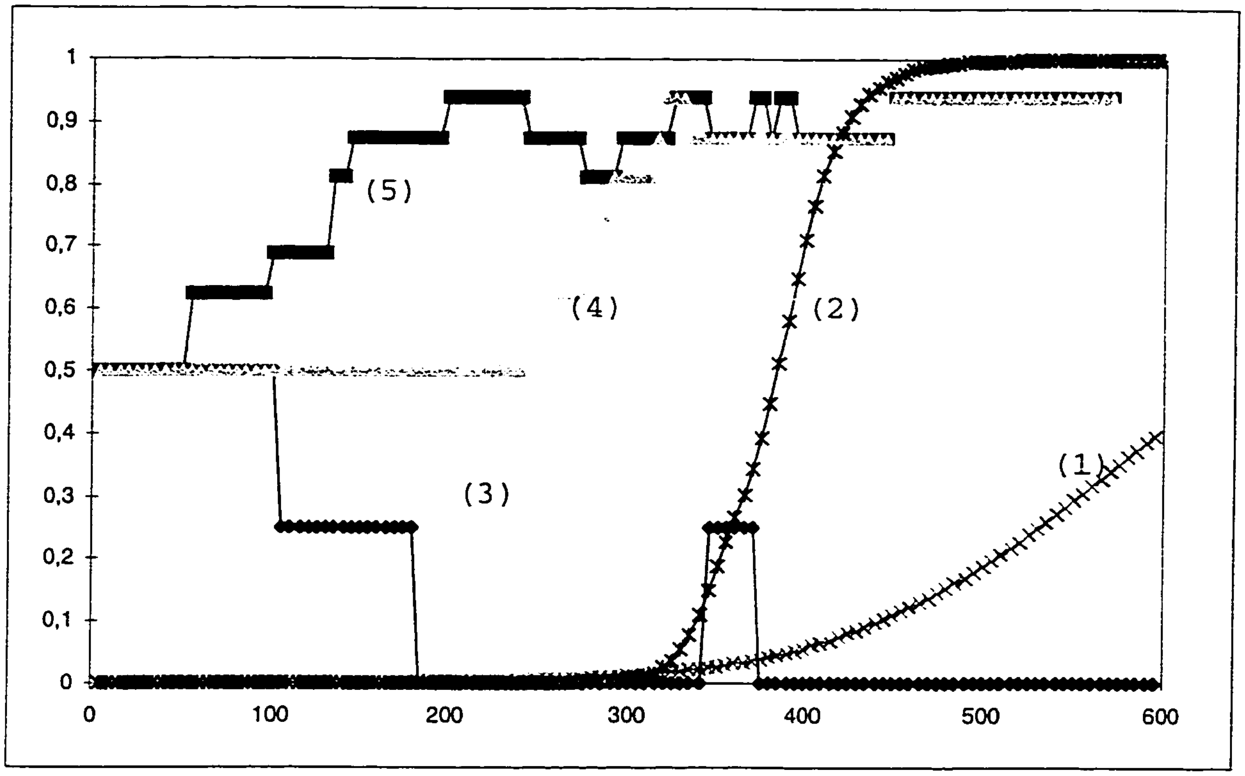


Figure III-3 Typical example of measures

The different curves appearing in Figure III-3 are:

(1) The value of the umbrella monitoring function (normalized by a factor of 1/4) for the y-layer.

(2) The monitoring umbrella function of the z-layer, also normalized.

(3) The number of errors determined by a calculated perceptron; the ordinate is, here, the number of errors divided by the number of configurations in the learning set.

(4) The generalization, measured on a test set, produced by the comparison output protocol. The ordinate is the number of properly classified input configurations divided by the number of configurations in the test set.

(5) The generalization produced by the calculated perceptron protocol, with the same conventions as in (4).

Figure III-4 presents the detection program used in the astrophysics applications. It implements the neural network constructed by the learning program, takes events from a real data file, determines the class of each event and, when applicable, calculates some "discrimination coefficients" and eventually produces a file containing physically significant data which allows the determination of the spectral index.

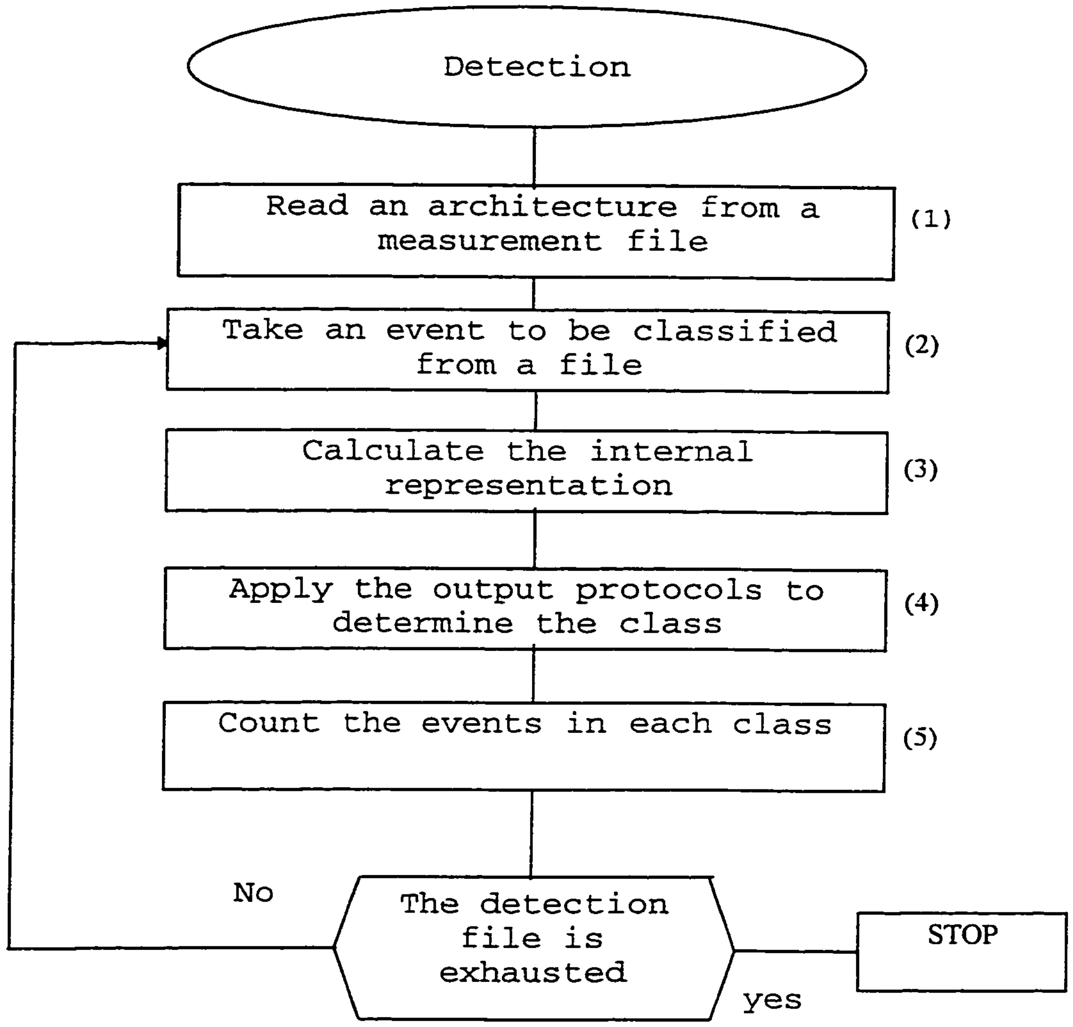

Detection
Read an architecture from a measurement file
(1)
Take an event to be classified from a file
(2)
Calculate the internal representation
(3)
Apply the output protocols to determine the class
(4)
Count the events in each class
(5)
No
The detection file is exhausted
yes
STOP

Figure III-4 Detection program

(1) The neural network architecture can be chosen at any recorded pass of the learning process.

(2) The events to be classified are in a file with the proper format.

(3) Is the actual neural-network processing of the input data.

(4) The actual classification made by the neural network is decided here.

(5) The detection program can itself contain some measurement apparatus to determine the quality of the discrimination, such as the excess in sigma for the astrophysics problem (see Chapter VI).

The detection program was used in the astrophysics problem only, and contained extra features used for the determination of the spectral index.

Used in this research were various IBM types of computers: 8088 640 Kb-4MHz, 80286 2Mb-16MHz, in the early stages; 486 8Mb-66MHz and pentium 16Mb-75 Mhz, later; with TURBO-C on all the machines, and GCC in a Linux environment on the 486 and the pentium. The minimal memory required by the learning program is "a few megabytes" depending on the size of the neural network, the number of layers and the layer dimensions. The minimal disk space required for the measurement files is a few hundred megabytes; that is why the Concordia computer mainframes could not be used. The learning program takes a few minutes of CPU, on the 486, for a small network and a small learning set as in the EXOR problem; it takes up to a few days on the pentium for the astrophysics problem, with a "large network" (6 layers of 8 cells and 12000 passes) and a large learning set (400 examples).

IV - SIMULATIONS

IV-1 EXOR: Definition of the problem and preliminary remarks

Considering the EXOR problem in the format of Table II-1, it is obvious that

(1) A statistical discriminator based on the class averages and the associated standard deviations cannot solve such a problem because the distributions of the two classes are completely identical.

(2) The crucial quantity that would discriminate between the two classes is the second moment involving the products of the inputs, as discussed in Section II-4-3.

(3) It is the necessity for these products that makes the problem unsolvable by the classical perceptron.

The learning behaviour umbrella algorithm for the classical, EXOR problem, and particularly the generalisation that would result using various implemented versions are now described.

(1) The linear learner: The architecture connections allow for linear combinations only of the pre-synaptic layer states in constructing the signal to the post-synaptic layer.

(2) The linear learner plus squares: the post-synaptic layer receives linear combinations of the pre-synaptic

states and of the squares of the states.

(3) The linear learner plus the mixed products of the states: combinations of the type $x_1 x_2$, $x_1 \neq x_2$ (i.e. no squares involved).

(4) All combinations: linear plus squares plus mixed products of two terms ((1), (2) and (3) together).

The **learning set** was constructed with entries between zero and one, and with input values such that the class averages would not vanish (the umbrella algorithm requires non vanishing class average vectors). Table IV-1 contains the learning set used for the EXOR problem.

Table IV-1-1 EXOR learning set

x1	x2	class
-0.55	-0.45	0
-0.45	0.56	1
0.54	-0.43	1
0.45	0.58	0

Table IV-1-2 EXOR test set

x1	x2	class
-0.5	-0.5	0
-0.5	0.5	1
0.5	-0.5	1
0.5	0.5	0
0.9	0.9	0
-0.9	0.9	1
0.9	-0.9	1
-0.9	-0.9	0
0.79	0.85	0
-0.78	0.84	1
0.95	-0.85	1
-0.72	-0.83	0

The layer dimensions are: 2 inputs, two processing units in each of the first and second internal representation layers and one cell for the output protocol.

The values of the learning parameters as well as the architecture parameters (the number of layers, the layer dimensions etc.) are always obtained by "trial and error", essentially by watching the behaviour of the W-functions during the learning process. The results presented here are for typical learning sessions.

The variable names in the graphs are: **ErEA** is the Error measured on the learning set; the calculated perceptron output protocol was used for error measurement. **CaPe** is the generalization, measured on the test set, for the calculated perceptron output. **Comp** is the generalization, measured on the test set, for the comparison protocol. $\mathbf{W_y}$ and $\mathbf{W_z}$ are the monitoring umbrella functions for the y and z layers repectively.

The normalization factor for **ErEA** is 1/4; the two generalizations (**Cape** and **Com**) are given by the number of correctly classified examples from the test set divided by 12; the umbrella monitoring functions $\mathbf{W_y}$ and $\mathbf{W_z}$ are both normalized by a factor of 1/4. The horizontal axis is labelled by the number of passes.

Since the input is two dimensional it is possible to systematically scan the inputs, and to calculate and use the output produced to construct the **map** response of the network. The coordinates of the point in the plane are the actual input values and the symbol (a dot for a below-threshold output state and a star for an above-threshold state) at the coordinate position represents the network classification by the calculated perceptron connected to the z layer. The "size" of the maps, namely the extreme input values, are -3.0 to +3.0 for both axes. The "Z" (for class zero) and the "U" (for class one) in the maps are the scanned points closest to the learning set points. The maps illustrate the generalization very clearly but can only with difficulty be used in higher dimensions.

IV-1-1 Linear learner

This is an architecture for which there are no D type couplings (see Section II-1). The linear couplings (the C's) and the thresholds are adapted during the learning process. As expected the umbrella algorithm cannot solve the problem; It is presented as an example of a "non-converging" learning process.

Architecture, learning parameters and learning rates:

```
number of layers: 3 : 2-2-2
synchronisation: Y
normalization: N
center the problem: Y
density of non vanishing couplings: 1.0
f mixed products=0.0
f squares=0.0
decreasing learning rates with layer depth=0.0
maximal couplings=1.0
random seed=0.260000
learning rates
  sigmoid constants: β=0.0  γ=0.0
  thresholds: β= 0.55  γ=-8.0
  linear couplings: β=0.3330  γ=-17.0
  non linear couplings: β=0.0  γ=0.0
two-layer bias= 44.000000
```

Figure IV-1 presents the learning performances and Figure IV-2 is the generalization map at pass 6000.

At pass 6000 the system was still making one error in the learning set, and the W functions barely reached 0.15; this is a good example of bad learning.

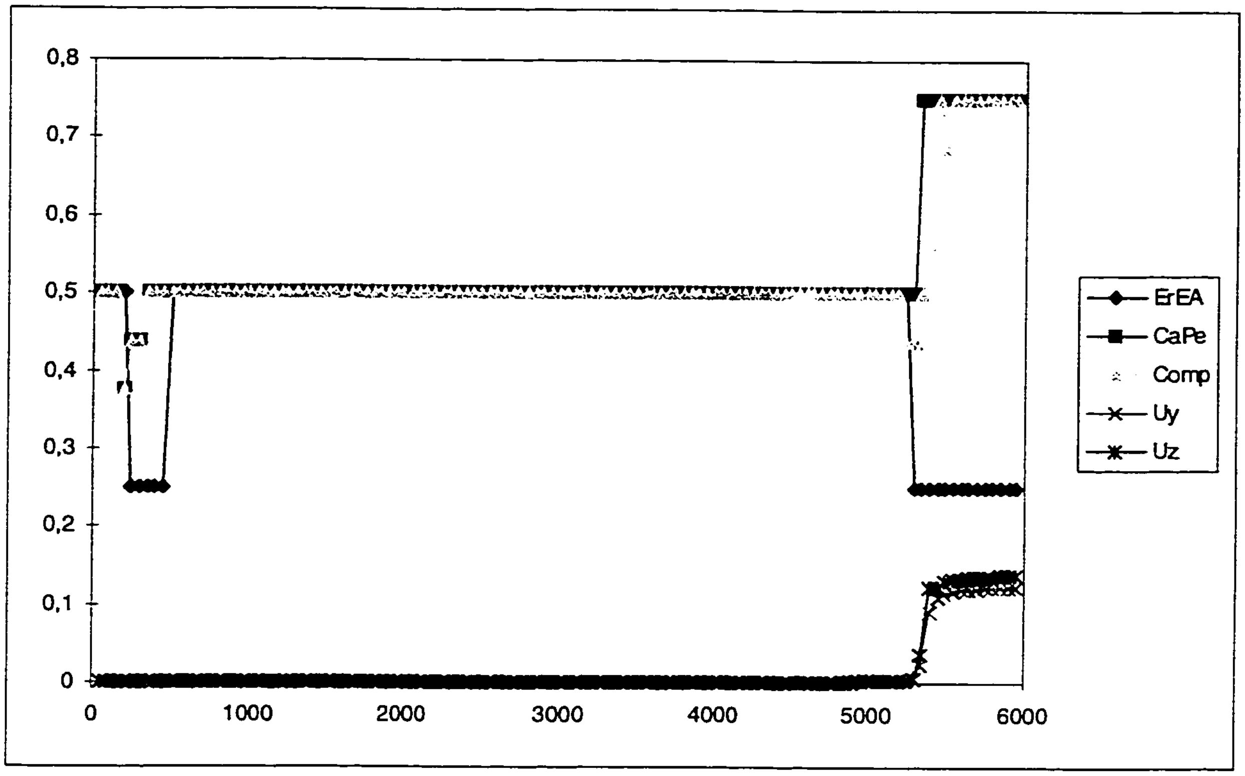


Figure IV-1 Linear learner performance in the EXOR problem

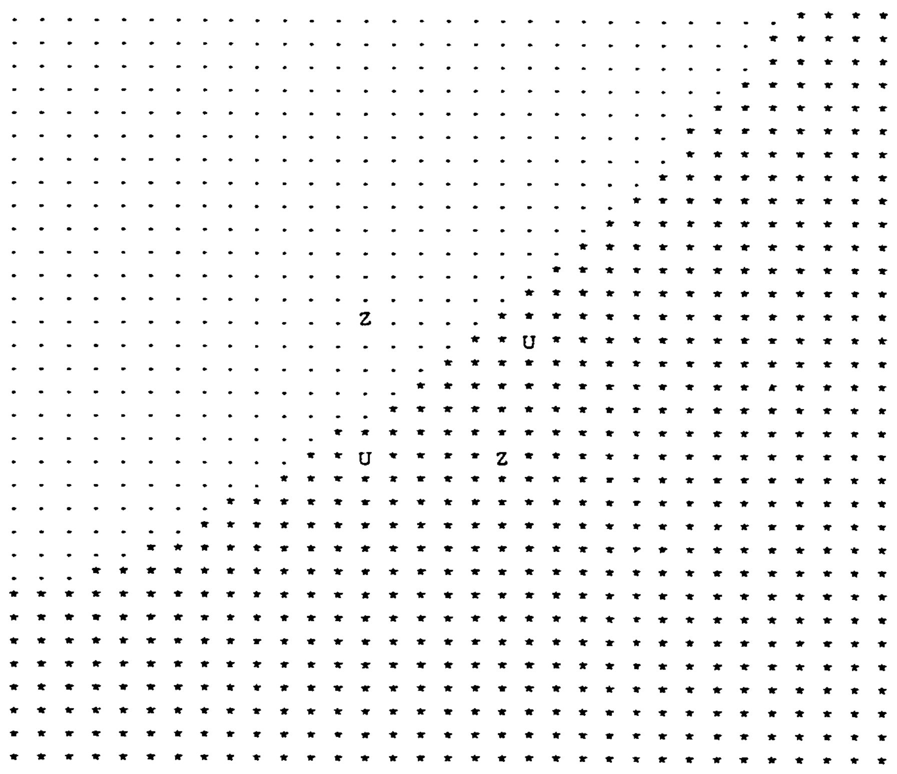


Figure IV-2 linear learner generalization map

Remark: Why is it that the umbrella algorithm cannot find the so-called "stripe solution" for the EXOR problem? (The stripe solution has the portion of input configurations in the region between the two oblique lines in Figure II-10 belonging to class zero and the rest of the input space assigned to class one.) It is possible that this is because the stripe solution is not one mapping but two; these two mappings have to (both) be built with the information available at the input level and therefore cannot be performed by successive layers. Perhaps a solution would be to use a very strong heuristic bias, for instance on the relative values of some thresholds and C_{ij}'s.

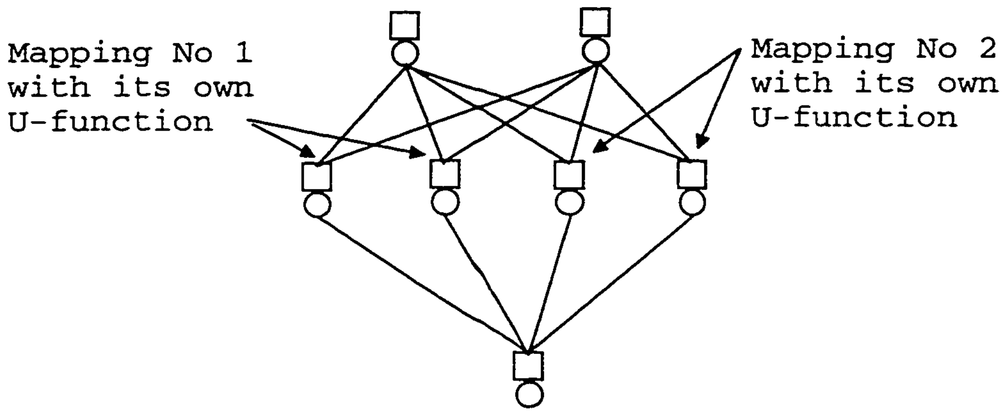


Figure IV-3 Architecture for the EXOR stripe.

The heuristic constraints can be as follows:

1) Split the U-function as illustrated in Figure IV-3: one for each mapping. Optimize them separately, hoping they don't find the same mapping. It may be the type of solution

obtained by backpropagation in an architecture with linear couplings only (Rumelhart and McClelland 1989).

2) In a 2-2-2 architecture, do not use the sigmoid as the signal function; rather use a function like a sigmoid (centered, as in Section II-1) which suddenly changes sign at some point near each asymptotic region. If fact, this solution appeared as a result of a bug in the development of the code.

IV-1-2 Linear learner plus squares

In this architecture the connections are of C and D types (Section II-1) but the non vanishing D's are for i=i' only.

Learning rates and parameters:

number of layers: 3 : 2-2-2
synchronisation: Y
normalization: N
center the problem: Y
density of non vanishing couplings: 1.0
f mixed products=0.0
f squares=1.0
decreasing learning rates with layer depth=0.0
maximal couplings=1.0
random seed=0.610000
learning rates
  sigmoid constants: β=0.0 γ=0.0
  thresholds: β= 0.50 γ=1.0
  linear couplings: β=1.0 γ=2.0
  non linear couplings: β=5.0 γ=12.0 (not used)
two-layer bias= 10.000000

Figure IV-4 presents the learning performances and Figure IV-5 is the generalization map at pass 6000. At about pass 3100 the system has reached zero error. Even if the error had remained zero for the learning set, as can be seen in the generalization map, it is nevertheless a bad performance because (1) the monitoring function (W_y) of the second layer is (besides being very low, at 0.25) higher than the W-function of the following layer. This is another example of bad learning.

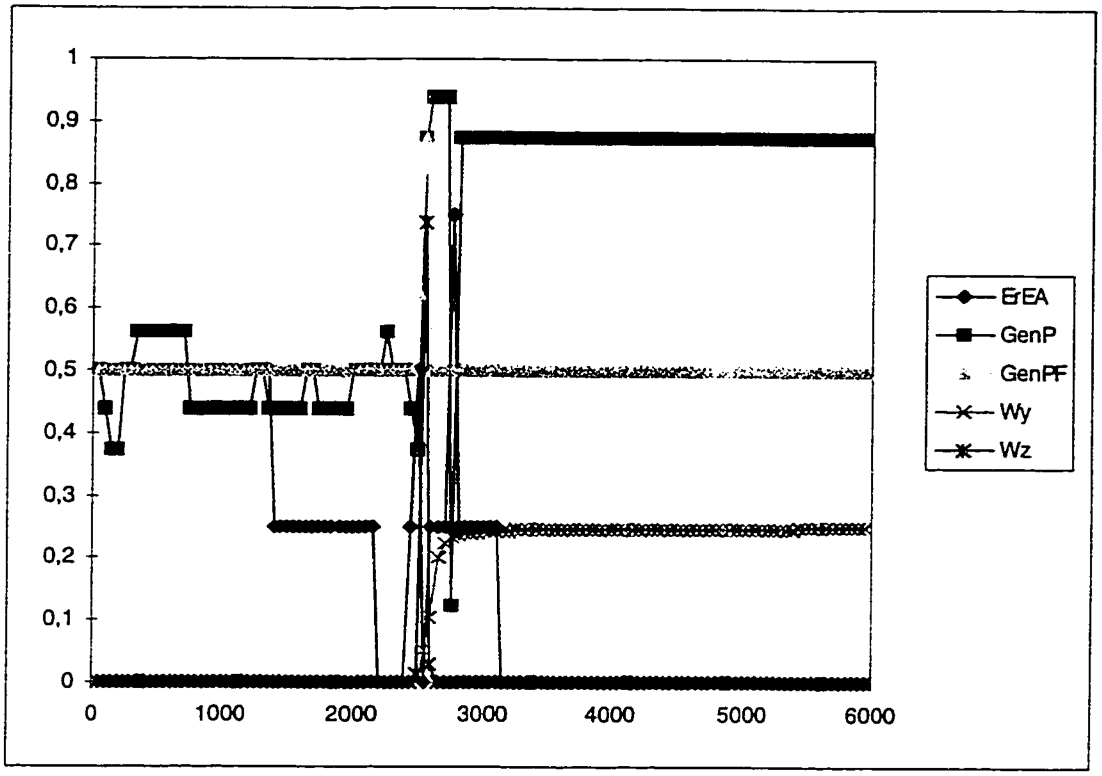


Figure IV-4 Linear plus squares learner performance

```
. . . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . * * * * * * * * * * * . . . .
. . . . . . . . . . . . . Z . . . * * * * * * * * . . . . . . . .
. . . . . . . . . . . . . . . . * * * U * * * . . . . . . . . . .
. . . . . . . . . . . . . . . * * * * * * . . . . . . . . . . . .
. . . . . . . . . . . . . * * * * * * * . . . . . . . . . . . . .
. . . . . . . . . . * * * * * * * * . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * U * * * . Z . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * . . . . . . . . . . . . . . . . . . . . . . .
```

Figure IV-5 Generalization map, linear plus squares learner

IV-1-3 Linear learner plus mixed products

In this architecture the connections are of C and D types (Section II-1) but the non-vanishing D's are for $i \neq i'$ only. Learning rates and parameters:

number of layers: 3 : 2-2-2
synchronisation: Y
normalization: N
center the problem: Y
density of non vanishing couplings: 1.0
f mixed products=1.0
f squares=0.0
decreasing learning rates with layer depth=0.0
maximal couplings=1.0
random seed=0.610000
learning rates
  sigmoid constants: β=0.0 γ=0.0
  thresholds: β= 0.50 γ=1.0
  linear couplings: β=1.0 γ=2.0
  non linear couplings: β=1.0 γ=2.0
two-layer bias= 10.000000

Figure IV-6 presents the learning performances and Figure IV-7 is the generalization map at pass 6000. Zero error is reached at about pass 400, the generalization on the test set reaches 1.0 and the generalization map clearly shows the suitableness of the internal representation that uses products of inputs. Figure IV-8 is the learning performance measured after 600 passes rather than after 6000; Figure IV-9 is the generalization map at pass 600, showing that between passes 600 and 6000 the system was not overspecializing over the learning set and its generalization slightly increased.

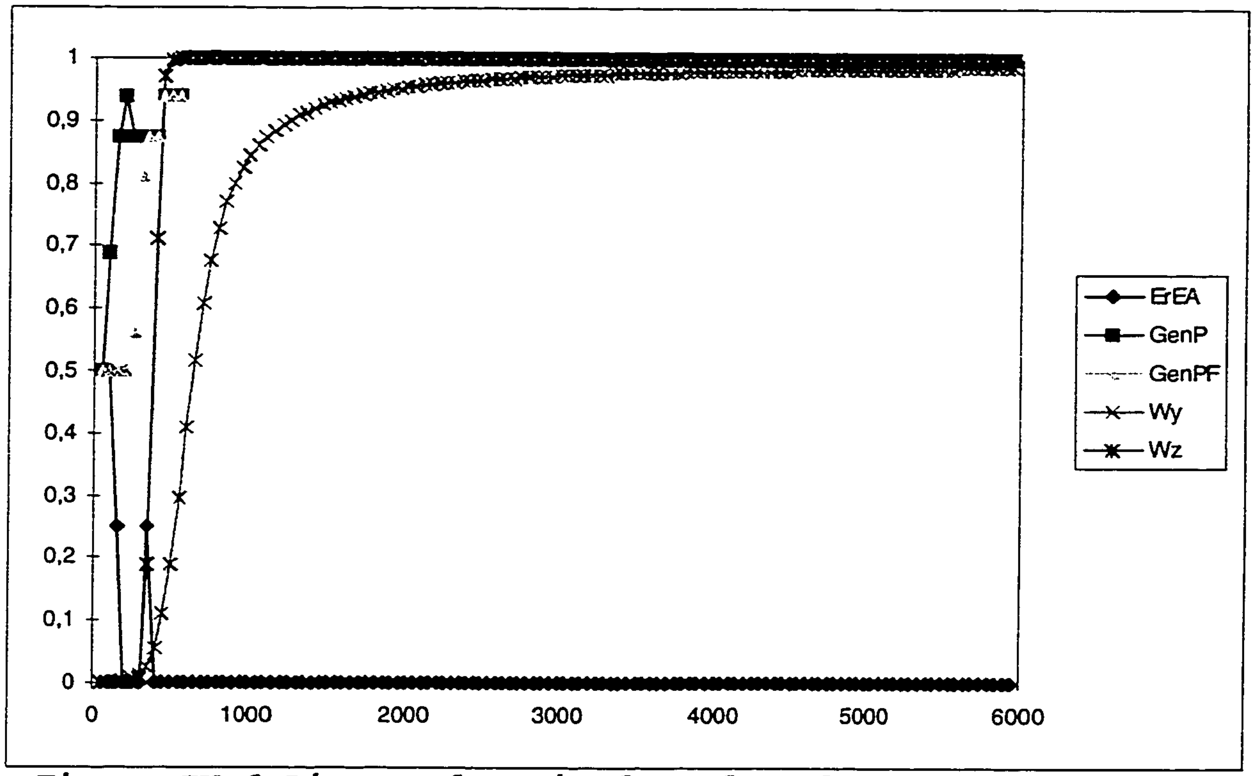


Figure IV-6 Linear plus mixed product learner performance

```
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . Z . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * U * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * U * * . . Z . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
```

Figure IV-7 Generalization map, linear plus mixed product learner, at pass 6000

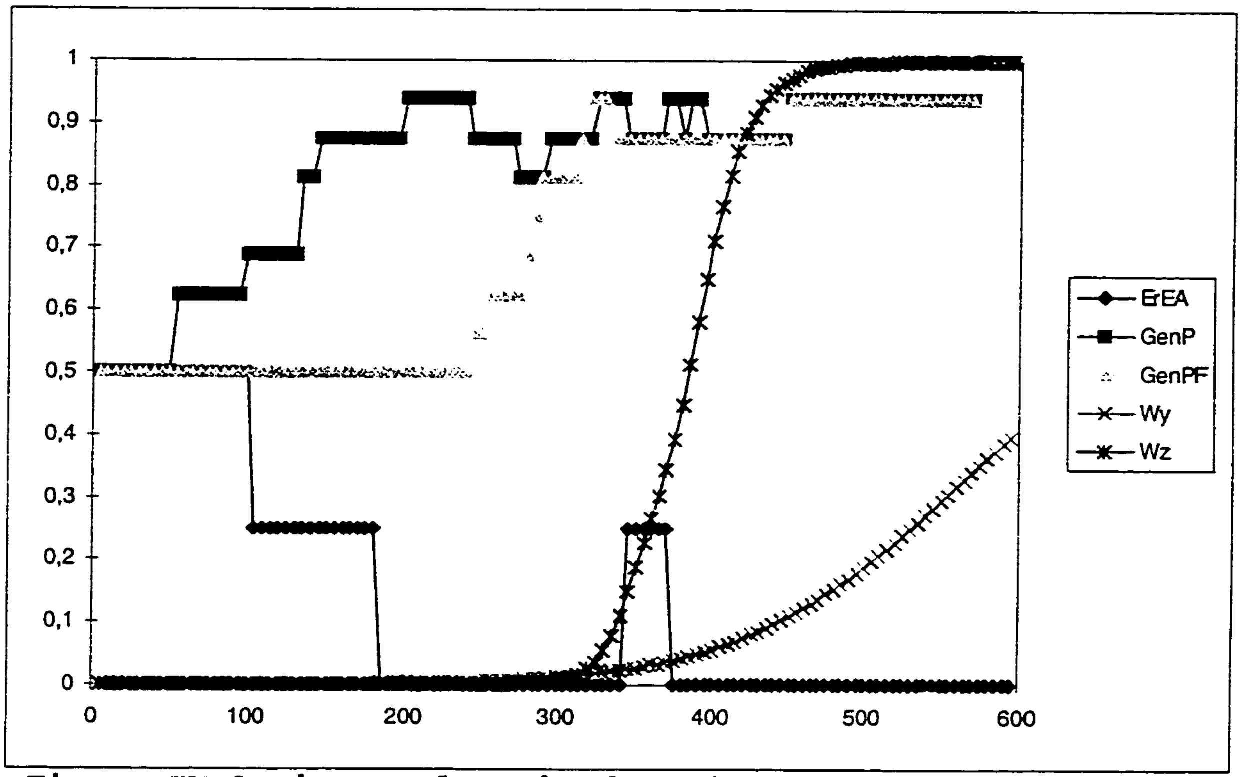


Figure IV-8 Linear plus mixed product learner, passes 0 to 600

```
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . Z . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * U * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . * * * * * * * * * * *
* * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * U * * . . Z . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
```

Figure IV-9 Linear plus mixed product learner, generalization map at pass 600

IV-1-4 Linear learner plus mixed products, higher rates

In this architecture the connections are of C and D types (Section II-1) but the non vanishing D's are for $i \neq i'$ only.

Learning rates and parameters:

number of layers: 3 : 2-2-2
synchronisation: Y
normalization: N
center the problem: Y
density of non vanishing couplings: 1.0
f mixed products=1.0
f squares=0.0
decreasing learning rates with layer depth=0.0
maximal couplings=1.0
random seed=0.610000
learning rates
  sigmoid constants: β=0.0 γ=0.0
  thresholds: β= 0.50 γ=1.0
  linear couplings: β=1.0 γ=2.0
  non linear couplings: β=5.0 γ=12.0
two-layer bias= 10.000000

Figure IV-10 presents the learning performances and Figure IV-11 is the generalization map at pass 6000. Since the learning rates (for the mixed products) are higher, the results are very much like those in Section IV-1-3 except for the rapidity of the "convergence"; for instance zero error is reached at about pass 40. Figures IV-12 and IV-13 present the same measures in 0 to 600 passes.

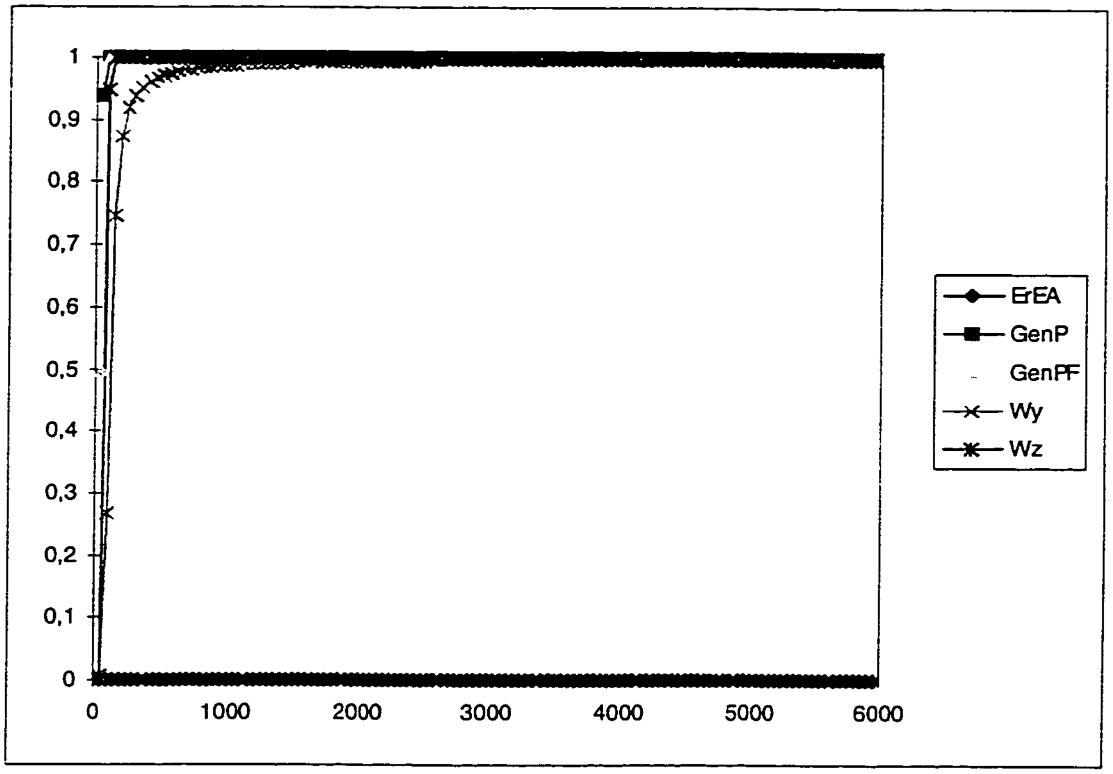


Figure IV-10 Linear plus mixed product learner performance

```
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . Z . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * U * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * U * * . . Z . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
```

Figure IV-11 Generalization map, linear plus mixed product learner, at pass 6000

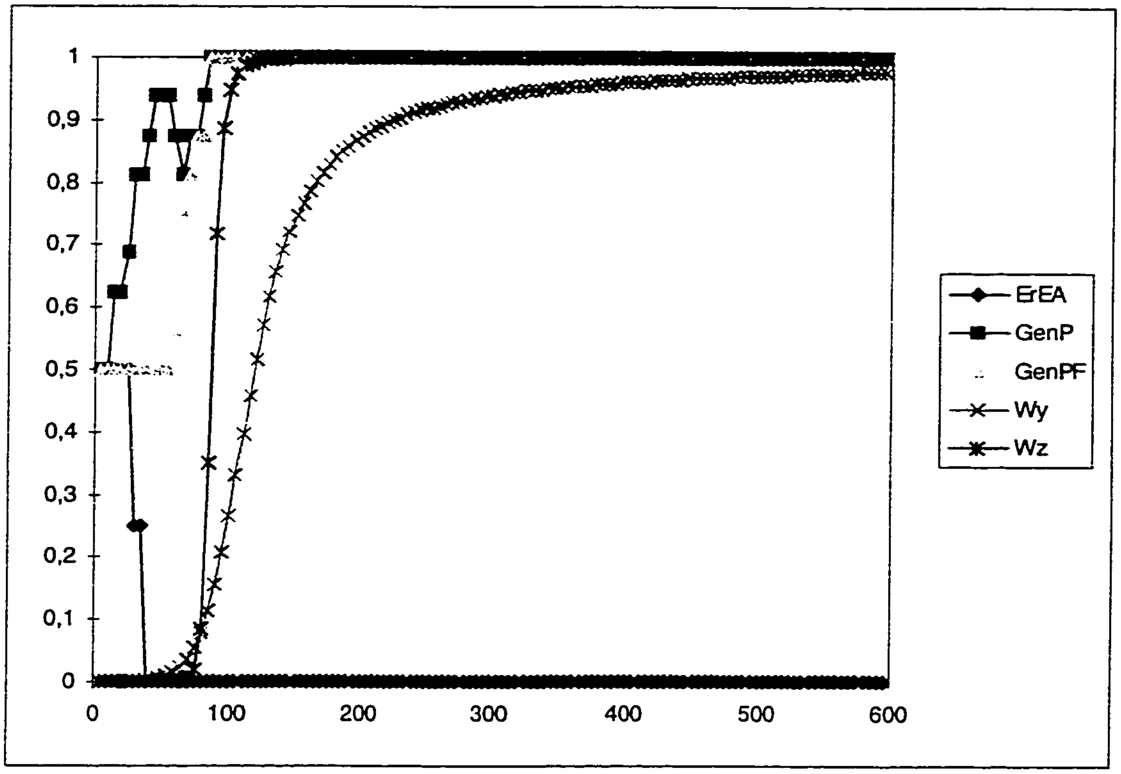


Figure IV-12 Linear and mixed product learner, high rates.

```
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . Z . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * U * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . * * * * * * * * * * * * * * * * *
. . . . . . . . . . . . . . . . . . . * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * U * * . . Z . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
* * * * * * * * * * * * * * * * . . . . . . . . . . . . . . . . .
```

Figure IV-13 Linear and mixed product learner, generalization map at pass 600

IV-2 - TOY DATA EXPERIMENT

IV-2-1 General description

A classical data discrimination problem can be described in the following manner. Events appear as vectors, each component (also called parameters) taking some real value; we will assume, here, that the components are all in the range [0, 1]. These events belong to classes; we will here assume two classes. The criterion which makes vectors belong to one class or the other is some (usually) unknown correlation between the components or some particular set of the component values, called a signature of the class. Artificial neural networks have the reputation for being very efficient in finding such hidden signatures. This is true for biological brains (for a very good example cf. the 3-D images of Hankinson et al 1994) and the goal is to achieve the same type of capability with artificial neural networks. To fully control the experiment we designed data with well-known distributions and correlations, in order to measure the ability of the algorithm to ferret out the crucial criteria and correctly classify the events. This is the toy data experiment.

IV-2-2 Generation of the toy data

The construction scheme for the toy data is as follows. The data are four dimensional vectors. The first two components are generated randomly, using a Montecarlo algorithm with Gaussian distributions having chosen characteristics. The two next components are defined as some function of the first two. The components were artificially cut off at 0 and 1 whenever they went out of range. Table IV-2-1 provides the details.

TABLE IV-2-1 Parameters of the gaussian distributions and functions used for the parameter generation.

Class 0

para.	mu	sigma	function
1	0.25	0.15	--
2	0.50	0.30	--
3	--	--	1-(p(1)+p(2))/2
4	--	--	p(1)*p(2)

Class 1

para.	mu	sigma	function
1	0.40	0.35	--
2	0.75	0.25	--
3	--	--	1-2*p(1)*p(1)
4	--	--	sqrt(p(1)*p(2))

This Montecarlo scheme was used to produce "raw" data files of 1000 events per class. Some descriptive statistics of these events are given in Table IV-2-2, and the histograms are given in Figures IV-2-1 a and b.

TABLE IV-2-2 Statistics of the generated event vectors (1000 per class).

para: parameter index
min: minimal value (after truncation, <0 rejected)
max: maximal value (after truncation, >1 rejected)
ave: average value
sigma: standard deviation
mfv: most frequent value (in a 20-bin histogram)

Class 0

para.	min	max	ave	sigma	mfv
1	0.000	0.725	0.263	0.139	0.275
2	0.000	0.975	0.492	0.238	0.525
3	0.179	1.000	0.623	0.159	0.575
4	0.000	0.660	0.142	0.113	0.075

Class 1

para.	min	max	ave	sigma	mfv
1	0.000	0.974	0.438	0.250	0.425
2	0.000	0.975	0.670	0.193	0.775
3	0.001	1.000	0.552	0.359	0.025
4	0.000	0.968	0.516	0.216	0.575

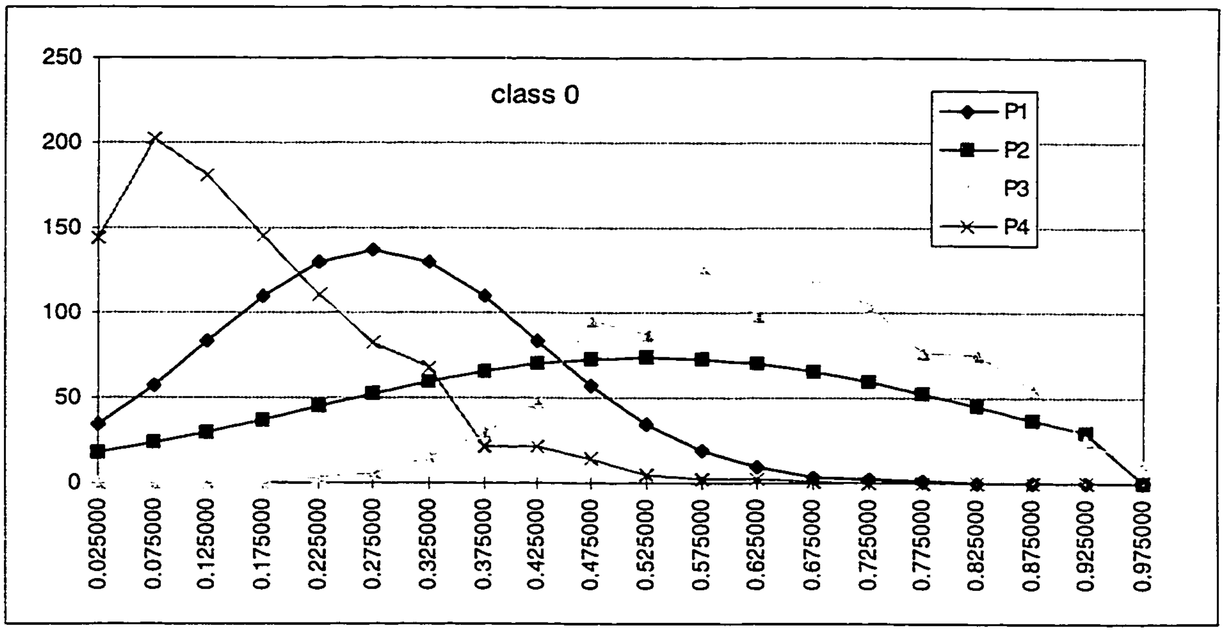


Figure IV-2-1a Class zero histogram

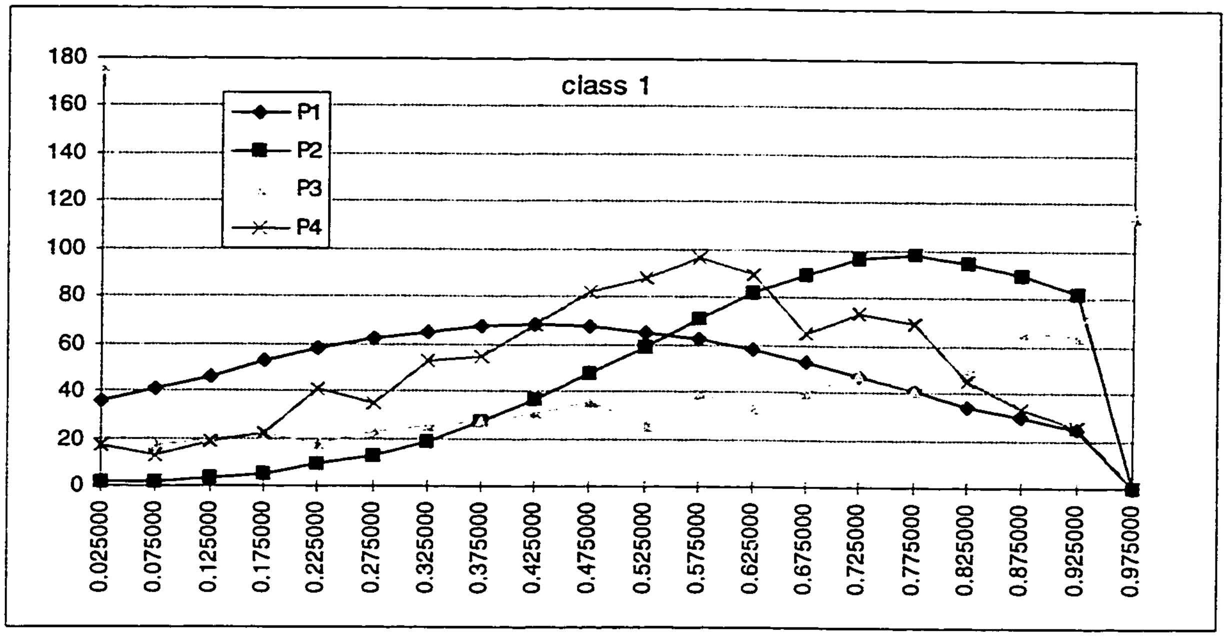


Figure IV-2-1b Class one histogram

A first learning set (LS-1) was then constructed using the following scheme: for each component and each class, the class average (mu) ± the class standard deviation (sigma) were used as "typical" values; that led to 2^4 vectors per class and 32 examples for the LS, given in Table IV-2-3.

TABLE IV-2-3 First learning set (LS-1) constructed from the average and standard deviations of Table IV-2-2. The last column gives the class of the event.

0.414	0.763	0.734	0.188	0
0.414	0.763	0.734	0.000	0
0.414	0.763	0.416	0.188	0
0.414	0.763	0.416	0.000	0
0.414	0.287	0.734	0.188	0
0.414	0.287	0.734	0.000	0
0.414	0.287	0.416	0.188	0
0.414	0.287	0.416	0.000	0
0.136	0.763	0.734	0.188	0
0.136	0.763	0.734	0.000	0
0.136	0.763	0.416	0.188	0
0.136	0.763	0.416	0.000	0

```
0.136 0.287 0.734 0.188 0
0.136 0.287 0.734 0.000 0
0.136 0.287 0.416 0.188 0
0.136 0.287 0.416 0.000 0
0.675 0.968 0.384 0.791 1
0.675 0.968 0.384 0.359 1
0.675 0.968 0.000 0.791 1
0.675 0.968 0.000 0.359 1
0.675 0.582 0.384 0.791 1
0.675 0.582 0.384 0.359 1
0.675 0.582 0.000 0.791 1
0.675 0.582 0.000 0.359 1
0.175 0.968 0.384 0.791 1
0.175 0.968 0.384 0.359 1
0.175 0.968 0.000 0.791 1
0.175 0.968 0.000 0.359 1
0.175 0.582 0.384 0.791 1
0.175 0.582 0.384 0.359 1
0.175 0.582 0.000 0.791 1
0.175 0.582 0.000 0.359 1
```

Note: the learning set was constructed from the distributions of the generated data; none of the events in the learning set could be found in the "raw data". To use the raw data for learning-set construction (which may be more realistic especially when the distributions are not Gaussian) the following Bayesian type of procedure was used (Kohonen 1989):

Definition of **species**: for each class and for each parameter a range of values (preferably a simply connected interval) was chosen for which the class distribution is significantly dominant. For example, comparing the histograms in Figure IV-2-1, the parameter P4 distribution of class zero dominates the P4 distribution of class one in the interval 0.000 to 0.3; then events having their fourth parameter in that range will be declared "being of species one" for the class zero. The same is done with all the parameters. A binary number is constructed having a one or a zero at the parameter position for the parameter value being respectively in the chosen interval or not. In four dimensions there are $2^4=16$ species per class. When the dominating region of a class was not simply connected, the interval containing the most frequent value was chosen. Some species may be empty.

From the raw data file, one event per species per class was chosen (the first one ecountered when there was more than one available). When the species was empty, an event from class species 0 was picked. This procedure was used to construct another learning set of 32 examples: LS-2, given in Table IV-2-4.

TABLE IV-2-4 Second learning set (LS-2) constructed with the species scheme. The second last column contains the class of the event and the last column contains the binary value of the species.

0.078	0.190	0.866	0.015	0	0000
0.124	0.485	0.695	0.060	0	0001
0.002	0.843	0.577	0.002	0	0010
0.085	0.773	0.571	0.066	0	0011
0.021	0.547	0.716	0.012	0	0100
0.173	0.532	0.648	0.092	0	0101
0.359	0.535	0.553	0.192	0	0110
0.428	0.624	0.474	0.267	0	0000
0.285	0.460	0.628	0.131	0	1000
0.265	0.192	0.771	0.051	0	1001
0.276	0.573	0.575	0.158	0	1010
0.385	0.825	0.395	0.317	0	0000
0.253	0.544	0.601	0.138	0	1100
0.029	0.674	0.648	0.020	0	0000

0.294	0.521	0.593	0.153	0	1110
0.074	0.169	0.878	0.013	0	0000
0.176	0.518	0.938	0.302	1	0000
0.684	0.517	0.065	0.595	1	0001
0.727	0.906	0.001	0.811	1	0010
0.709	0.505	0.001	0.599	1	0011
0.051	0.765	0.995	0.198	1	0100
0.469	0.758	0.560	0.596	1	0101
0.837	0.791	0.001	0.814	1	0110
0.294	0.467	0.827	0.371	1	0000
0.444	0.955	0.606	0.651	1	1000
0.421	0.818	0.645	0.587	1	1001
0.185	0.486	0.931	0.300	1	0000
0.583	0.311	0.320	0.426	1	0000
0.192	0.520	0.926	0.316	1	0000
0.415	0.786	0.655	0.571	1	1101
0.242	0.541	0.883	0.362	1	0000
0.327	0.572	0.787	0.432	1	0000

Another set of "raw data", 1000 events per class, was used as a test set.

IV-2-3 Bayesian discrimination

The **species method** can itself be used as a discrimination procedure. The dominant species of a class can be determined by simply counting the events from raw data belonging to a particular species. The most or the few most populated species can then be used to define a filter that assigns a class to each new event. This is a "first Bayesian separator".

Another implementation of a Bayesian separator, called the **class scores,** does not use species but does use the same intervals found for the species method. With four parameters, there will be four intervals for class zero and four intervals for class one. The class scores procedure is then, for an unclassified event:

- Each of its parameters is compared to the corresponding interval of class zero; if the parameter value falls in that interval, one point is scored for class zero.
- Each of its parameters is compared to the corresponding interval of class one; if the parameter value falls in that interval, one point is scored for class one.
- The class with the largest score is the class assigned to the event. In case of equality, the class is declared undecidable.

A comparison of either of the Bayesian separator results and a neural network classification reveals that the neural network is more appropriate for the problem at hand. This is the purpose of the toy data experiment. The effect of the f bias of Section II-2 was also measured.

The performance of the class-scores Bayesian separator is now presented.

TABLE IV-2-5 Class-scores Bayesian separator measured on a test set of 2000 events (1000 per class).

event's class in the test file	number of events assigned to class 0	number of envents assigned to class 1	number of undecidable events
0	896	48	56
1	184	512	304
	1080	560	360

The last line of Table IV-2-5 is the classification given by the Bayesian separator: of the 2000 events, 1080 will be recognized as class 0 events, 560 will be recognized as class 1 events and 360 will not be decidable, their class scores being equal. For class zero: 896 events will be assigned to class zero; 48 events will be assigned to class one and 56 will not be assigned to any class.

The performance of the Bayesian separator indicates that roughly 88%-90% of events from class 0 are properly classified, and 52%-58% of events from class 1 are properly classified. The numbers vary, depending on the modality of application and on the data set used.

IV-2-4 Umbrella algorithm: linear with bias

The learning sets LS-1 and LS-2 were used to make a three-layer architecture, with 4 neurons per layer, learn the toy data discrimination problem. All the learnings were stopped at 600 passes; the monitoring U-function of the third layer was, in every case, above 3.9. The zero error measured here for the learning process was obtained using a calculated perceptron, and the generalization was measured using both the calculated perceptron and the comparison protocol.

TABLE IV-2-6 LS-1 (ave+-sigma); results at pass 600 measured on the same test set as in Table IV-2-5

				output, BOTH protocols		
f	Wy	Wz	Er=0	Class 0	Class 1	Figure
0	3,640	3,999	80	868	631	2a
.01	3,669	3,999	75	868	636	2b
0.1	3,786	3,999	60	871	654	2c
1.0	3,908	3,999	40	875	665	2d
10	3,939	3,999	35	880	669	2e
100	3,943	3,999	35	880	670	2f
10^9	3,943	3,999	35	880	670	2g

TABLE IV-2-7 LS-2 (species) results at pass 600 measured on the same test set as Tables IV-2-5 and IV-2-6

				output, BOTH protocols		
f	Wy	Wz	Er=0	Class 0	Class 1	Figure
0	3,454	3,999	135	960	944	3a
.01	3,486	3,999	-	961	948	3b
0.1	3,496	3,999	-	964	956	3c
1.0	3,801	3,999	45	964	959	3d
10.	3,861	3,999	40	965	961	3e
100	3,868	3,999	40	965	962	3f
10^9	3,868	3,999	40	965	962	3g

The Class 0 and Class 1 columns contain the number of events assigned to their proper class. For instance 880, 670 of the last line of Table IV-2-6 implies that 120 events belonging to class zero were assigned to class one and 330 events belonging to class one were assigned to class zero; there were no undecidable events here.

For Class 0, Table IV-2-6 very closely reproduces the results of the Bayesian separator. The discrepancy for Class 1 might arise from the fact that our discrimination device does not have an ambiguity (0 AND 1) class. Table IV-2-6 shows that the umbrella algorithm can learn the Bayesian separator "exactly".

As for the bias, although it slightly affects the final generalisation and the umbrella function of the intermediate layer, as expected, we cannot claim that it drastically improves the learning. The most important effect of the bias is the earlier pass number at which zero error is obtained for the learning set; the learning process is simply accelerated by the bias. While the existence of the bias is nothing more than mathematical correctness about the gradient of the U_Z function depending on the couplings of the input to the *y-layer*, it is not obvious that such a process exists in biological systems.

Tables IV-2-6 and IV-2-7 show the importance of the choice of the learning set. LS-2 learning appears to be much more conformal to the Toy Data problem than LS-1 learning. The learning sets constitute the only difference between Tables IV-2-6 and IV-2-7; the architectures are identical, the learning rates are the same, as are the initial connections, thresholds, everything!

Moreover, it is seen from Table IV-2-7 that the system can recognize "higher" moments (i.e., beyond the average and the sigma); to reach 96% of recognition (i.e.,far above 60% in Class 1), the system must certainly have learned something about the hidden correlations of the parameters.

The reported learning rates and parameters were obtained after only 3 trials, and were the same for all the learnings of Tables IV-2-6 and IV-2-7; consequently the differences in learning performances are due to the learning set only.

TABLE IV-2-8 Learning parameters for Table IV-2-6 and Table IV-2-7

Architecture: 3 layers (input included)
Layers dimensions: 4, 4, 4
Thresholds β=2.0 γ=4.0
Sigmoid constants β=0.0 γ=0.0
Linear couplings β=10.0 γ=20.0
Non-linear couplings β=0.0 γ=0.0
decreasing learning rates with depth=1.0
Maximal value of the randomly set couplings=1.0
Initial density of couplings=1.0
Synchronous adjustments=YES
Normalization of inputs=NO
Problem centering (input)=YES
Seed (for random number generator)=0.5
Pass Max=600
Sample learning at every 5 passes

Variables measured:

ErEA = Number of errors of the calculated perceptron output, on the learning set and normalised (divided by 32 here)

GenP = Generalisation of the calculated perceptron output, normalised; for instance, in Table IV-2-6, first line, the total number of input configurations correctly classified is 868 + 631 = 1499; and the generalization, in Figure IV-2-4-1a, at the end of the process, is 1499/2000=0.749. (2000 is the number of examples in the test set.)

GenPF = generalization obtained with the comparison output protocol.

$\mathbf{W_y}$ and $\mathbf{W_z}$ = Monitoring U-functions of the last two layers (here: 2 for y and 3 for z).

The calculated perceptron and the comparison output protocol are connected to the last layer (z, here).

The measurements were done up to pass 600 for all the Figures, IV-2-4-1, IV-2-4-2 and IV-2-4-3, but we plotted the functions up to 250 because the crucial parts of the graphs are between, roughly, 0 and 200. Basically, above pass 250 the curves come ever closer to their asymptotic values (4.00 for the umbrella monitoring functions W_y and W_z)or remain as they are. The error, for example, remains at zero, while the generalizations might improve by one or two events in two thousand.

The measurements were done at every 5 learning steps (passes); that is why there are no measures at step zero and the x-axis starts at the value 5, where the error is already low or even zero in Figures IV-2-4-2a, IV-2-4-2b and IV-2-4-2c. The error goes up and down in Figures IV-2-4-1a, IV-2-4-1b, IV-2-4-1c and IV-2-4-1d and in all the figures IV-2-4-3 (a to g), clearly indicating that the error function cannot be trapped in a local minimum: the error function goes to zero (for successful learning), and zero is an absolute minimum. There may be other zero error minima, corresponding to different achitectures, and corresponding also to different internal representations conformal with the problem; but those minima too are absolutes. There are many different devices that can solve the same problem.

A remarkable feature of these measures is that the two output protocols differ only in the early stages of the learning, when the monitoring functions are low; one can say roughly that when $W_Z > 0.7$ the two output protocols are barely distinguishable. Furthermore, the generalization never decreases after zero error is reached. This is a very important property of our algorithm: it doesn't "over-specialise" in the learning set, as already pointed out in the sections on theory and the EXOR problem.

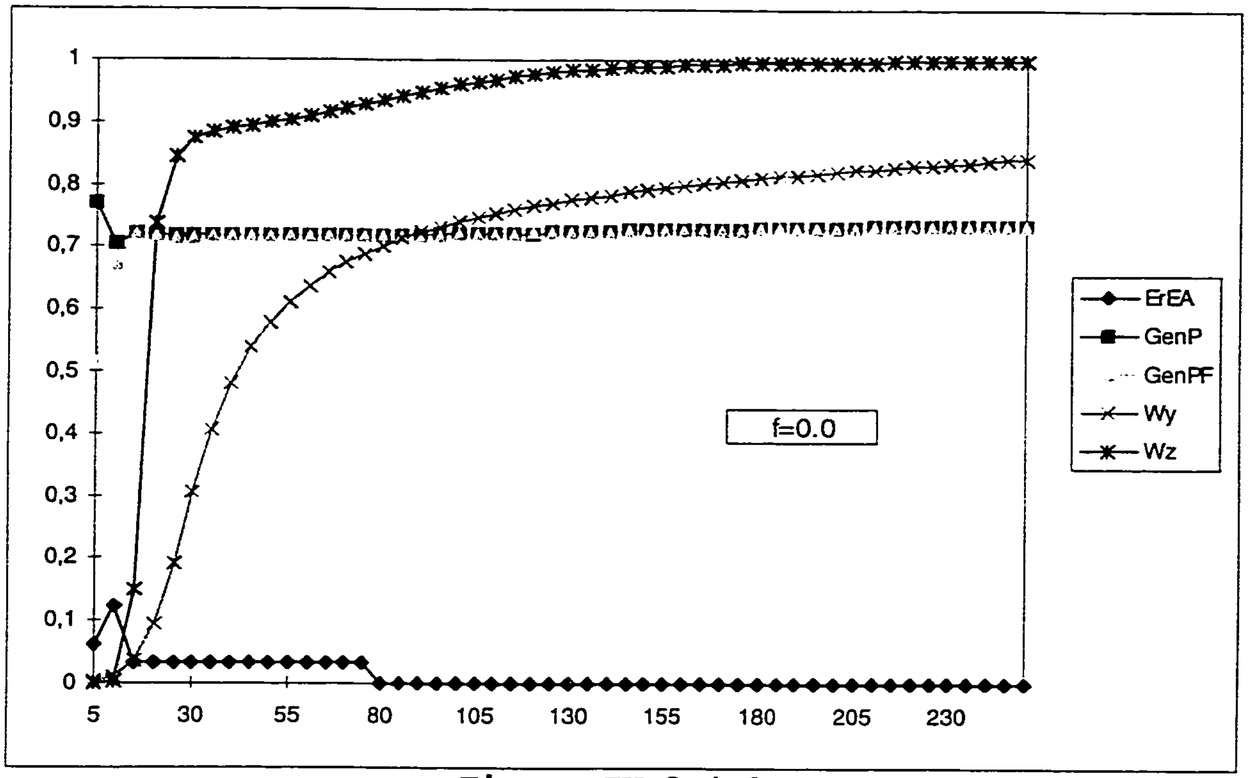


Figure IV-2-4-2a

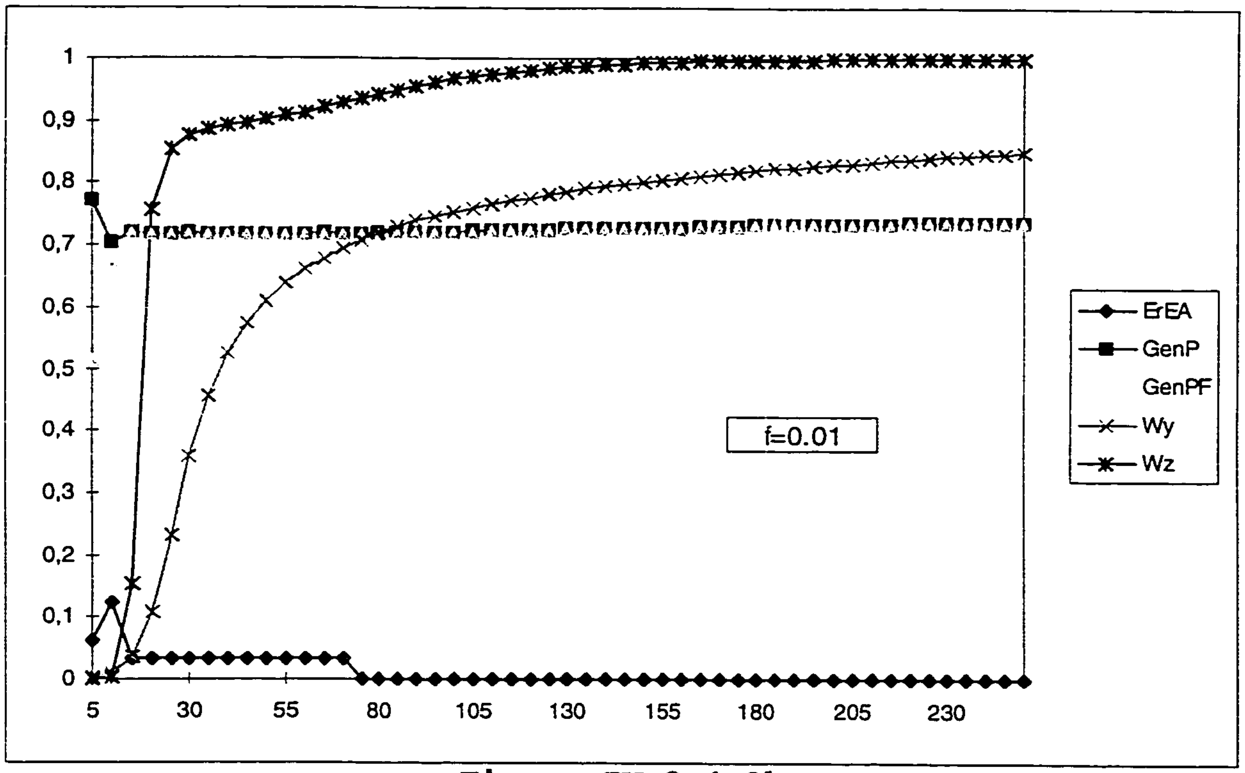


Figure IV-2-4-2b

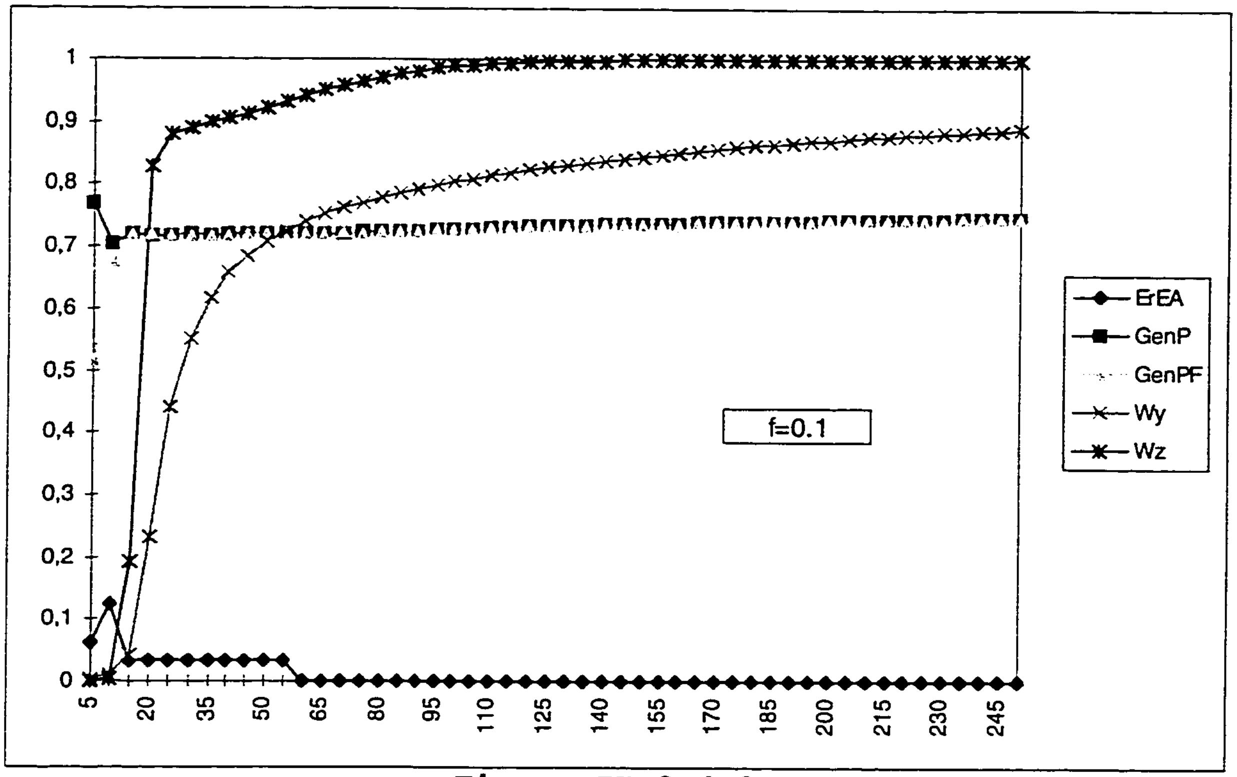


Figure IV-2-4-2c

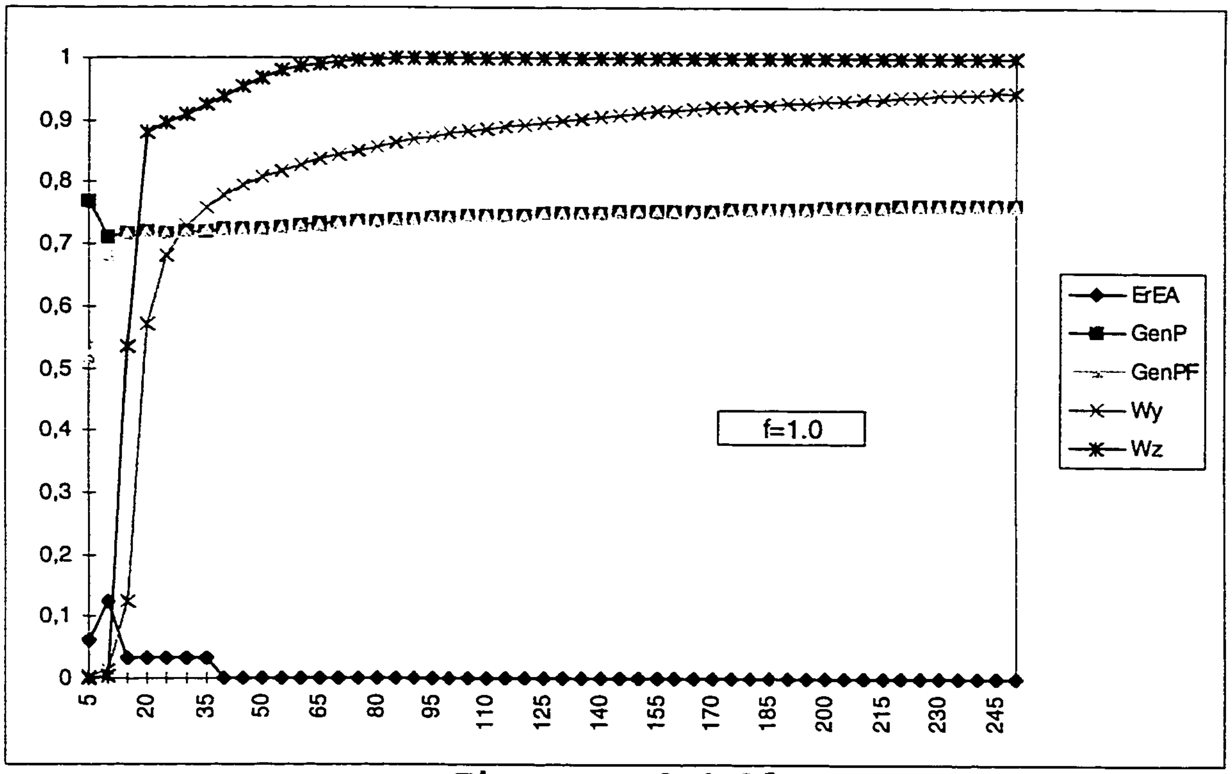


Figure IV-2-4-2d

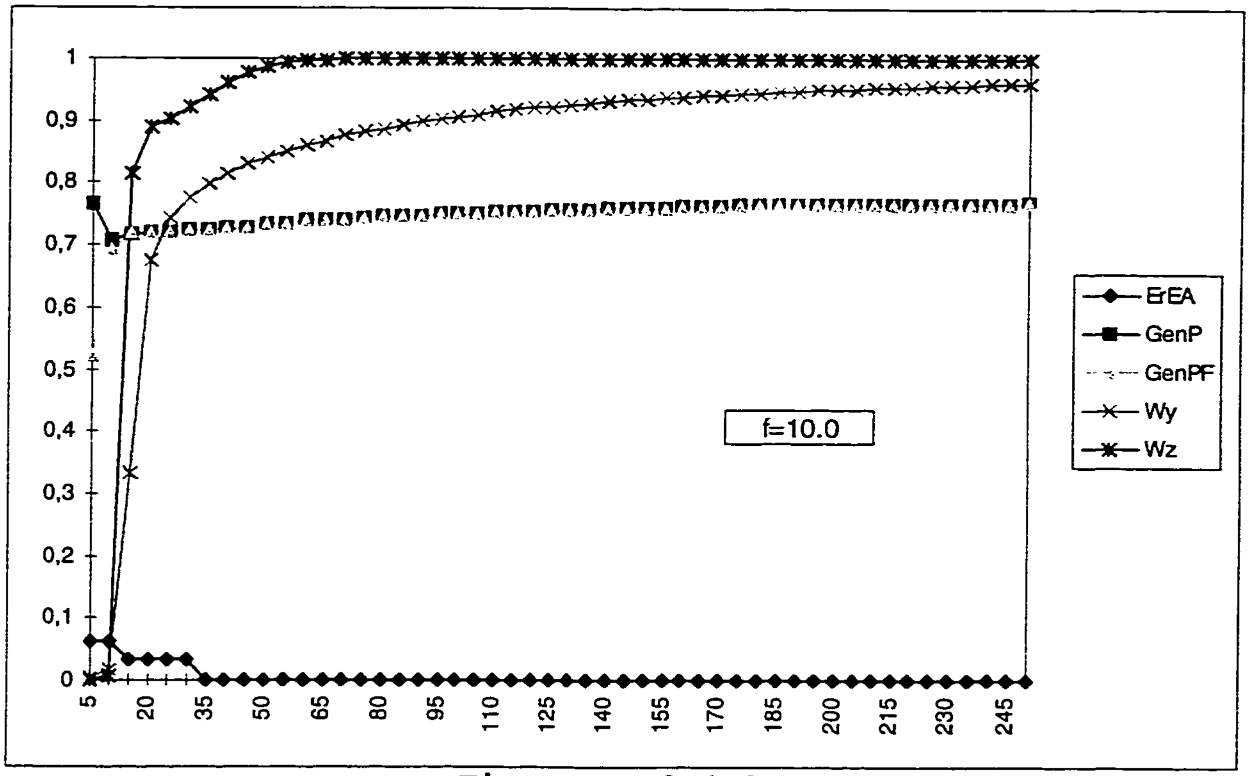


Figure IV-2-4-2e

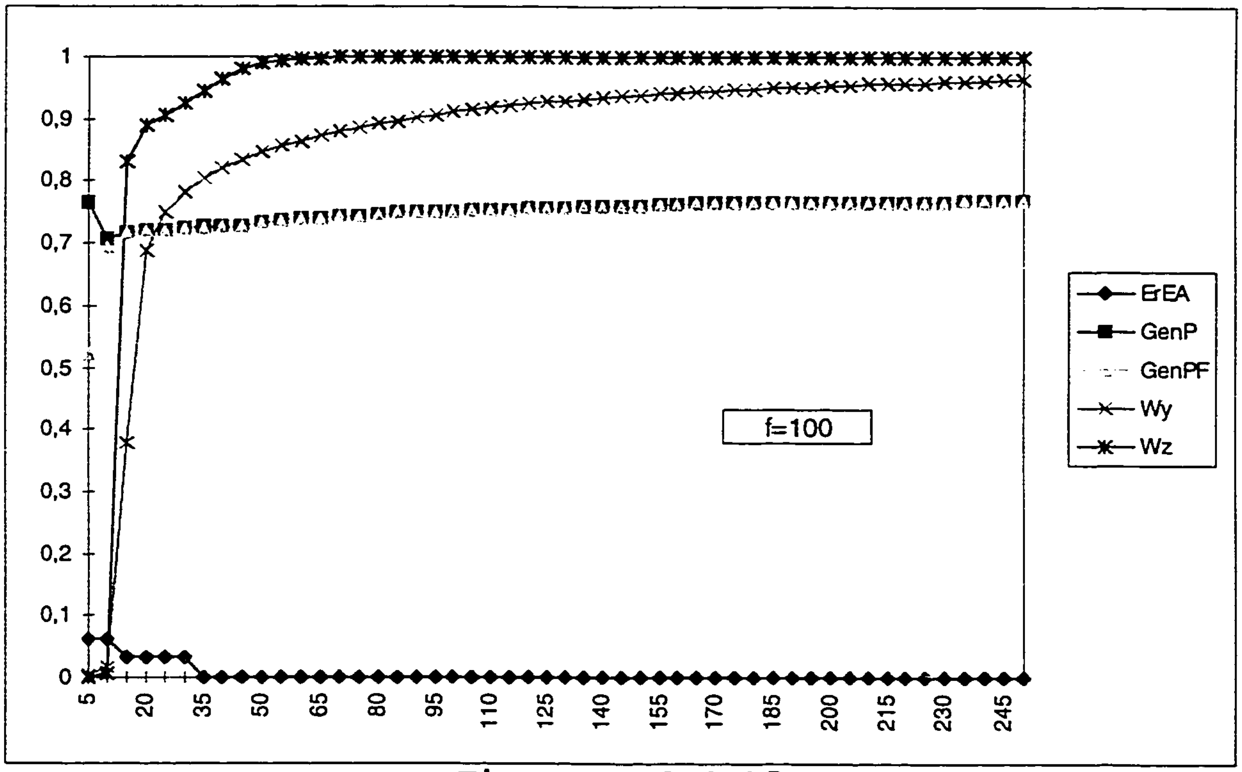


Figure IV-2-4-2f

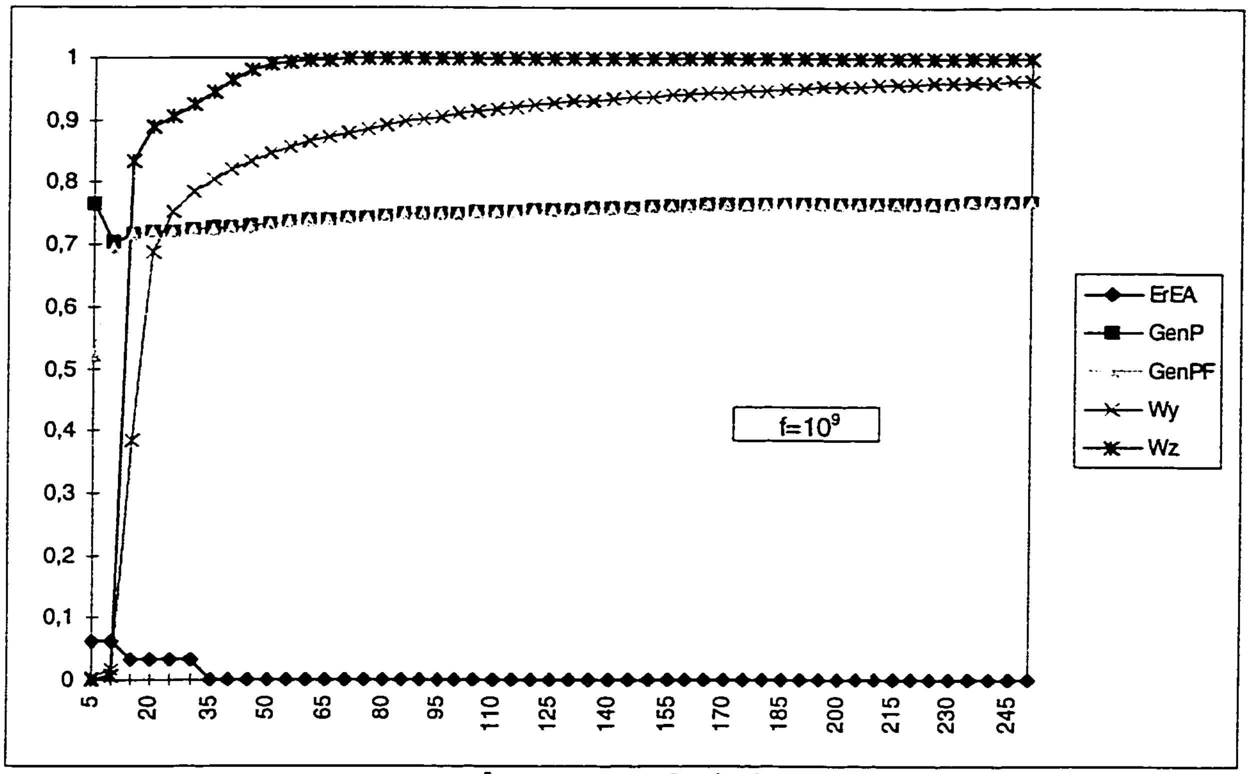


Figure IV-2-4-2g

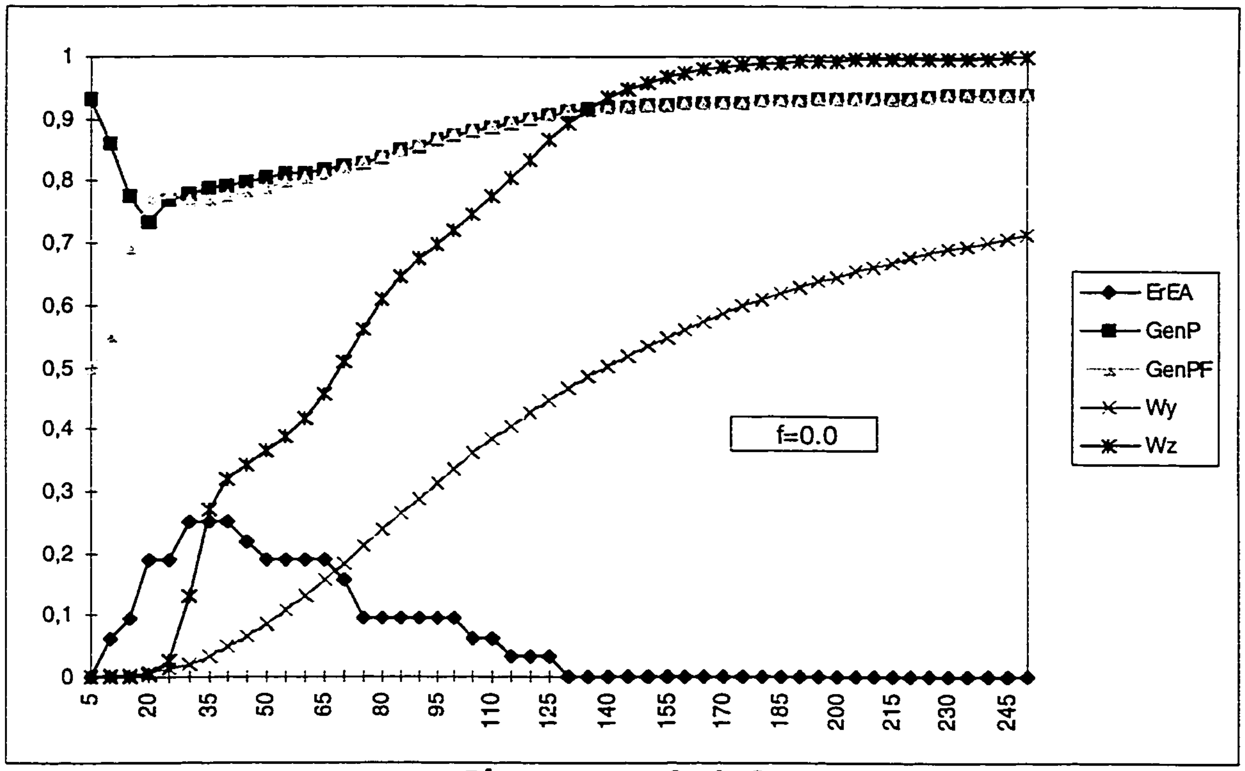


Figure IV-2-4-3a

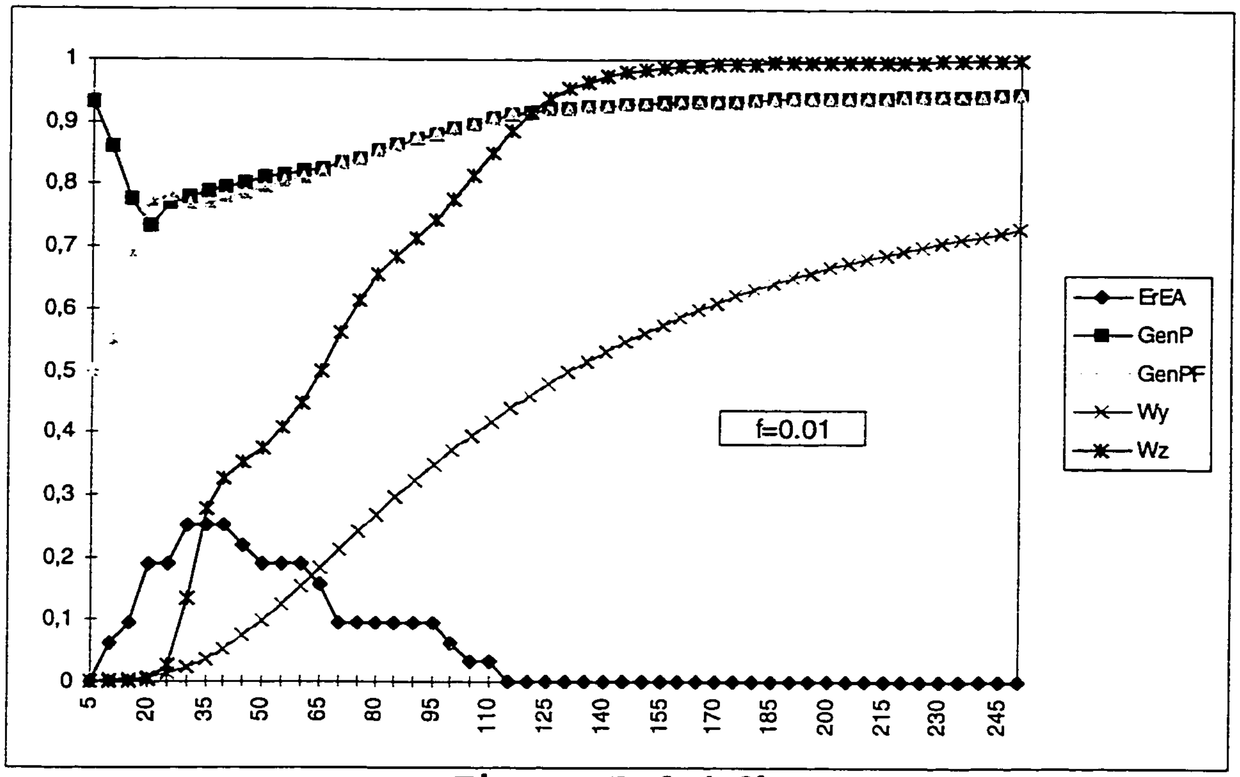


Figure IV-2-4-3b

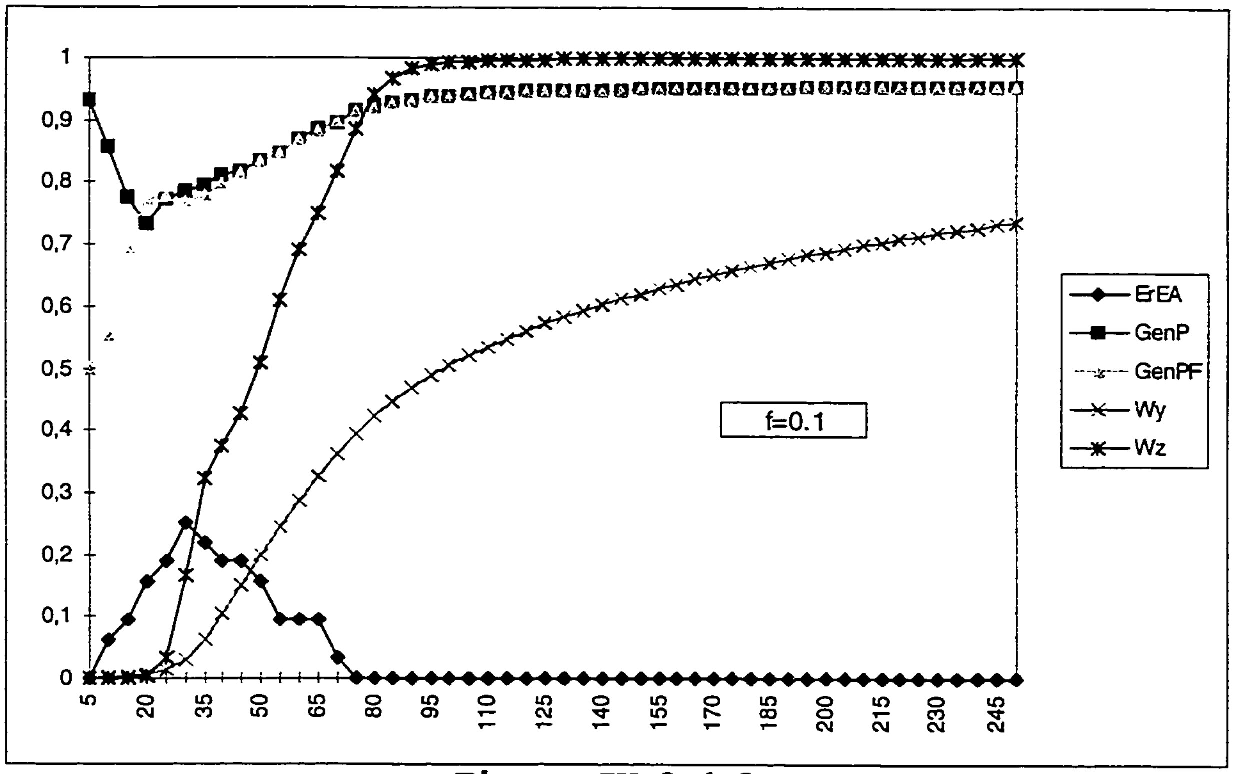


Figure IV-2-4-3c

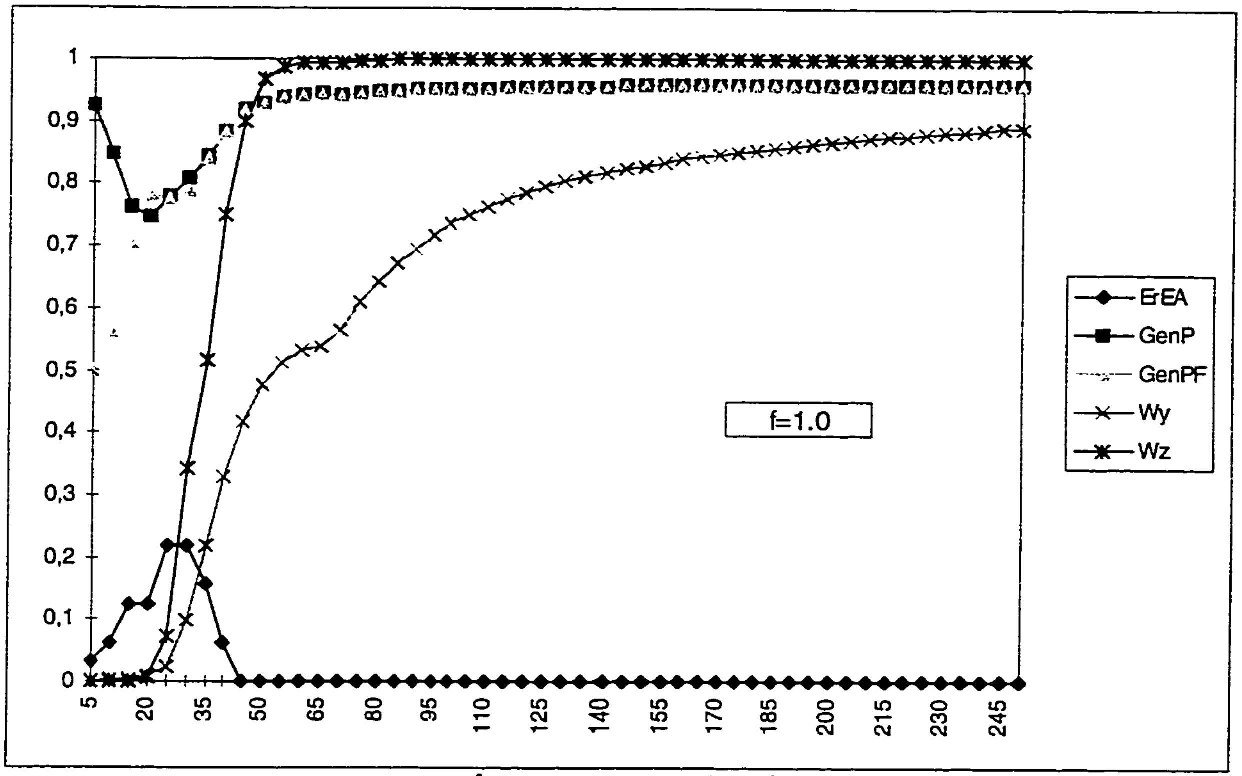


Figure IV-2-4-3d

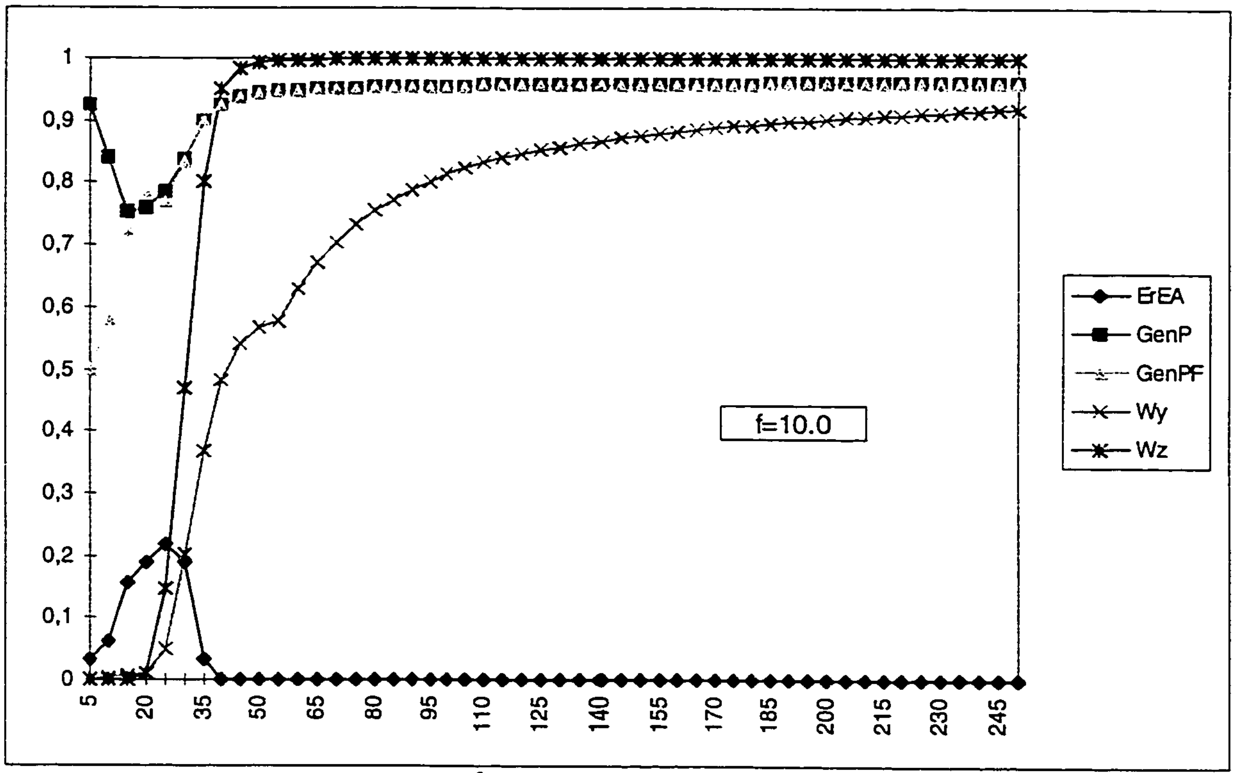


Figure IV-2-4-3e

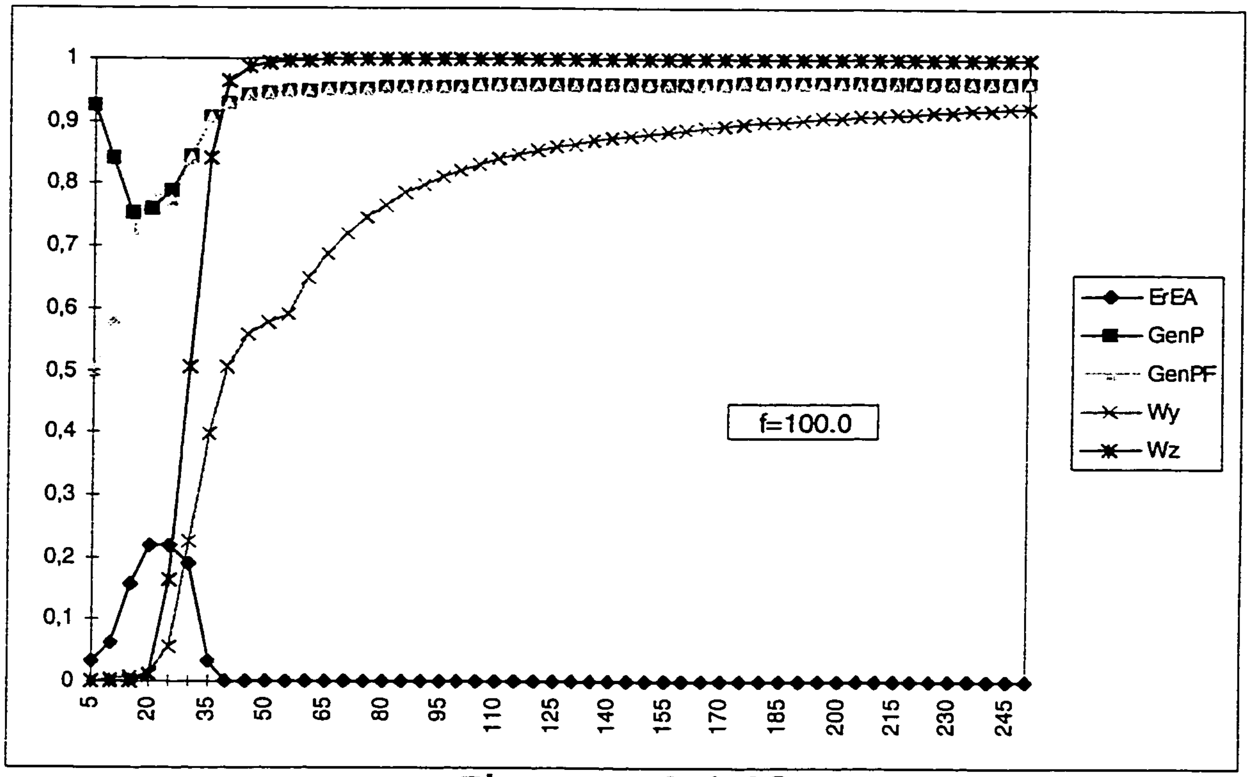


Figure IV-2-4-3f

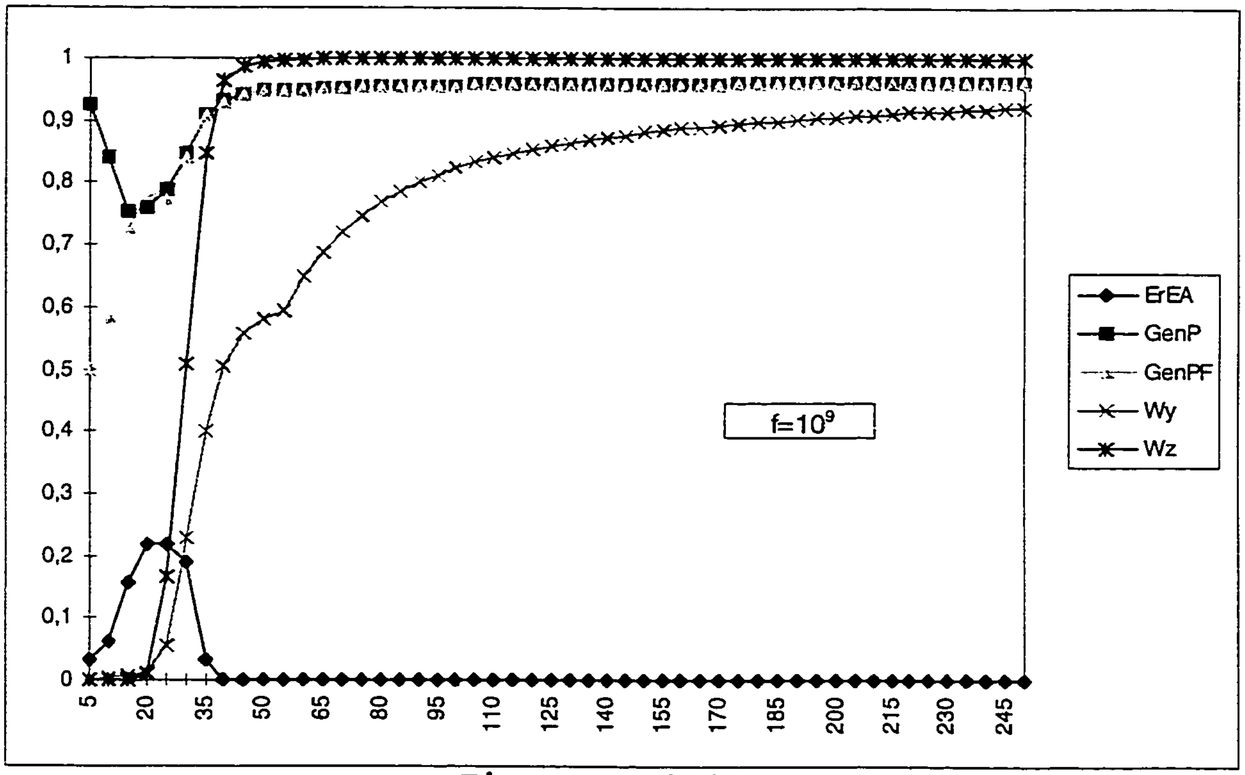


Figure IV-2-4-3g

TABLE IV-2-9 Learning parameters for the Table IV-2-10

Architecture: 3 layers (input included)
Layers dimensions: 4, 4, 4
Thresholds β=2.0 γ=4.0
Sigmoid constants β=0.0 γ=0.0
Linear couplings β=10.0 γ=20.0
Non-linear couplings β=15.0 γ=30.0
f=10.0
$f(i \neq i')$ = 0 (squares only) or 1 (others)
$f(i = i')$ = 1 (squares and squares+mixed) or 0 (mixed only)
decroi=1.0
Maximal value of the randomly set couplings=1.0
Initial density of couplings=1.0
Synchronous adjustments=YES
Normalization of inputs=NO
Problem centering (input)=YES
Seed (for random number generator)=0.5
Pass Max=600
Sample learning at every 5 passes

TABLE IV-2-10 Non-linear learner, LS by species; results at pass 600 Test set: ASD.TSU. There was a f-bias of 10.0.

Products					output, BOTH protocols		
mixed	squares	Uy	Uz	Er=0	Class 0	Class 1	Fig.
ON	ON	3,895	3,999	40	965	964	4a
ON	OFF	3,883	3,999	40	965	961	4b
OFF	ON	3,832	3,999	-	965	962	4c

In the following three figures, calculated perceptron error (ErEA) is measured over the learning set and divided by 32 for the graphic scale. GenP is the generalization produced by the calculated perceptron (divided by the number of examples in the test set: 2000); GenPF is the generalization of the comparison protocol (also divided by 2000); W_y and W_z are the umbrella monitoring functions (divided by 4) for the first and second internal representation layers respectively.

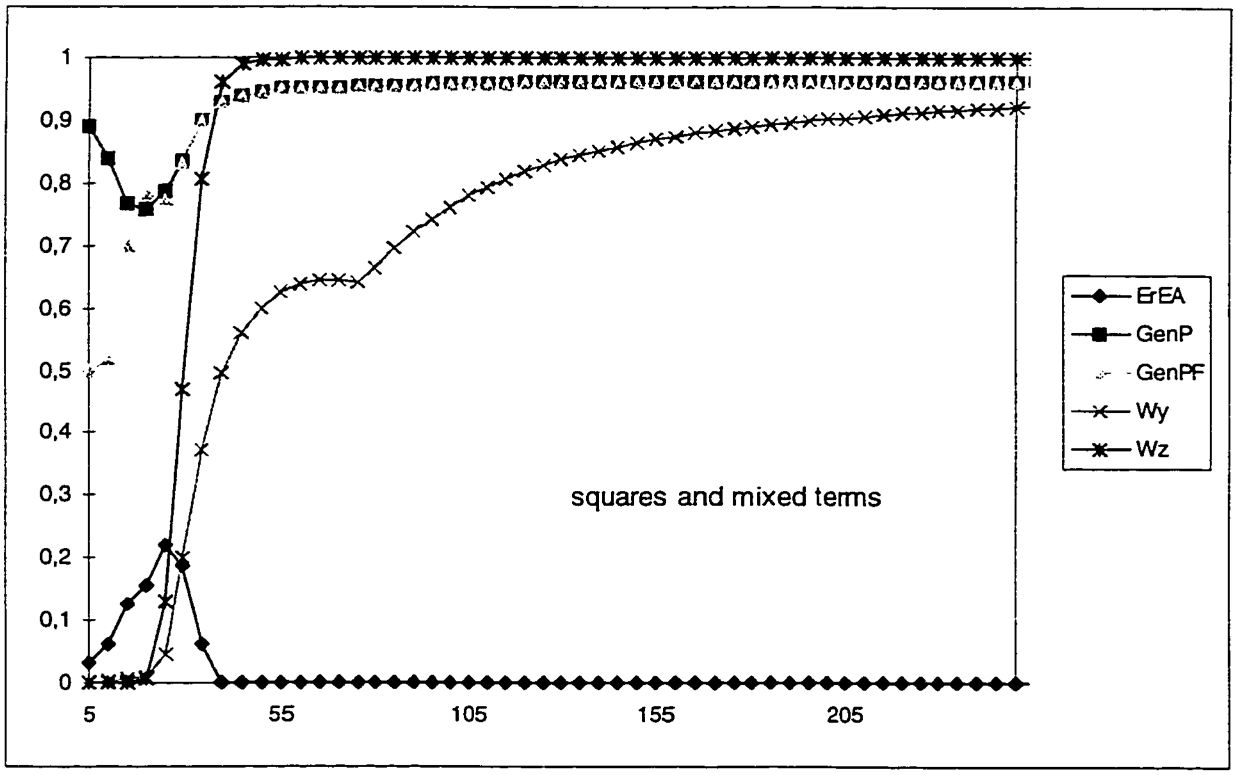


Figure IV-2-4-4a

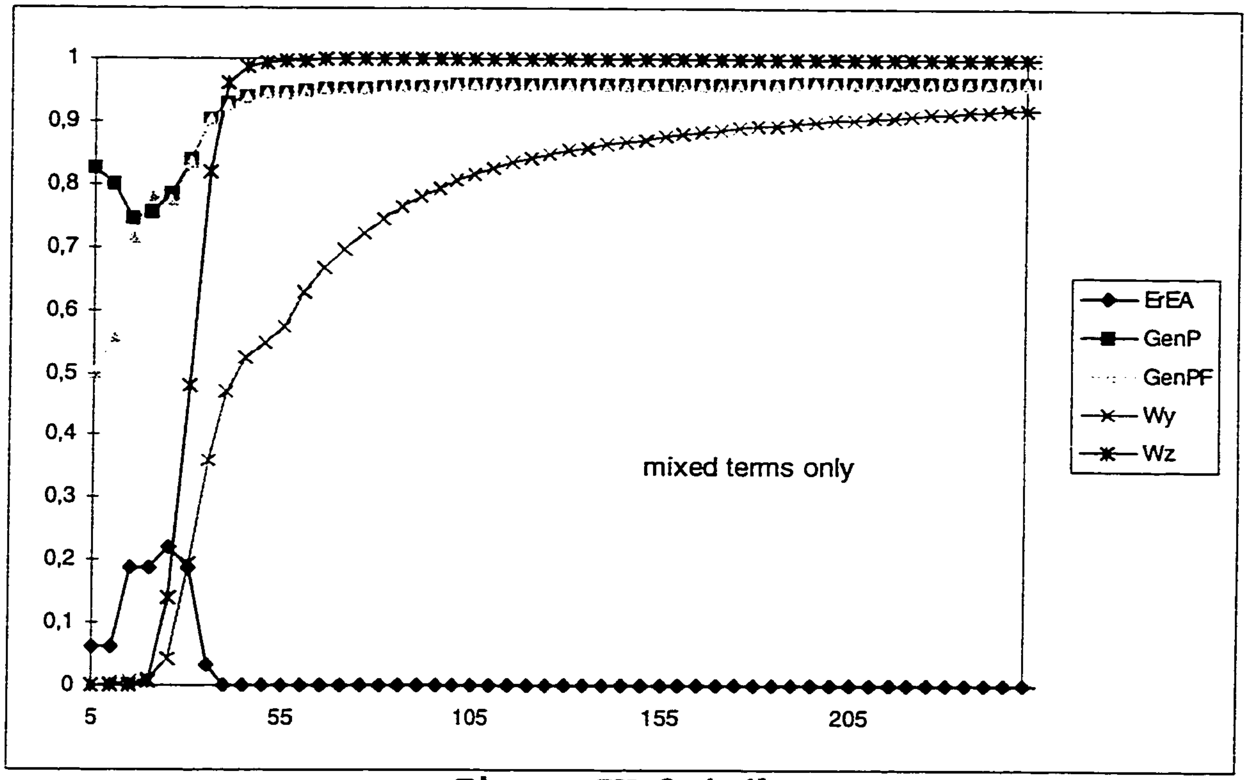


Figure IV-2-4-4b

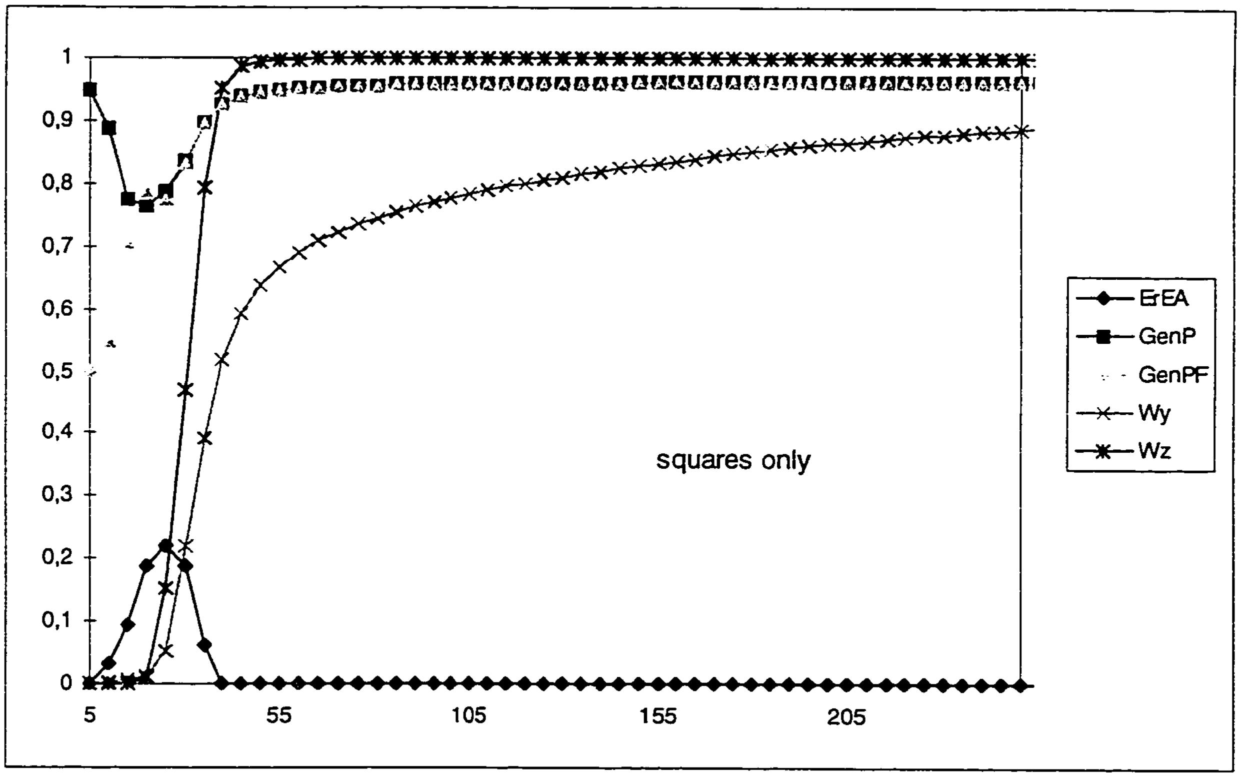


Figure IV-2-4-4c

IV-3 Learning times; comparison with backpropagation

Norio Baba published the learning performance of an improved version of the backpropagation learning algorithm (Baba 1989). His paper contains the learning set used for the parity problem over 7 inputs. It is thus possible to compare the learning performance of the umbrella and backpropagation algorithms for that case. An architecture comprising 7 input cells, only one internal representation layer composed of 7 units, and two output cells was chosen. The connection weights C_{ij} from the input layer to the internal representation layer were adjusted synchronously. There were no non-linear couplings. All thresholds (θ's) were maintained at zero, and the sigmoid constants (c's) at 1. The zero output protocol was used (Section II-3-1), the output cell producing the earliest zero error being taken as the single output cell for the network, the other output cell never being used afterward. The connections from the internal representation to the output layer were constructed via the umbrella algorithm as if it were just another internal representation layer. These choices were made early in the development of the umbrella algorithm, before the development of the calculated perceptron and the comparison output protocols.

The performances of a typical umbrella learning session are given in Figure IV-3-1. The learning parameters were determined, as usual, by trial and error. U_Y is the monitoring umbrella function for the first learning layer and the R's are the similarity measures for the same layer.

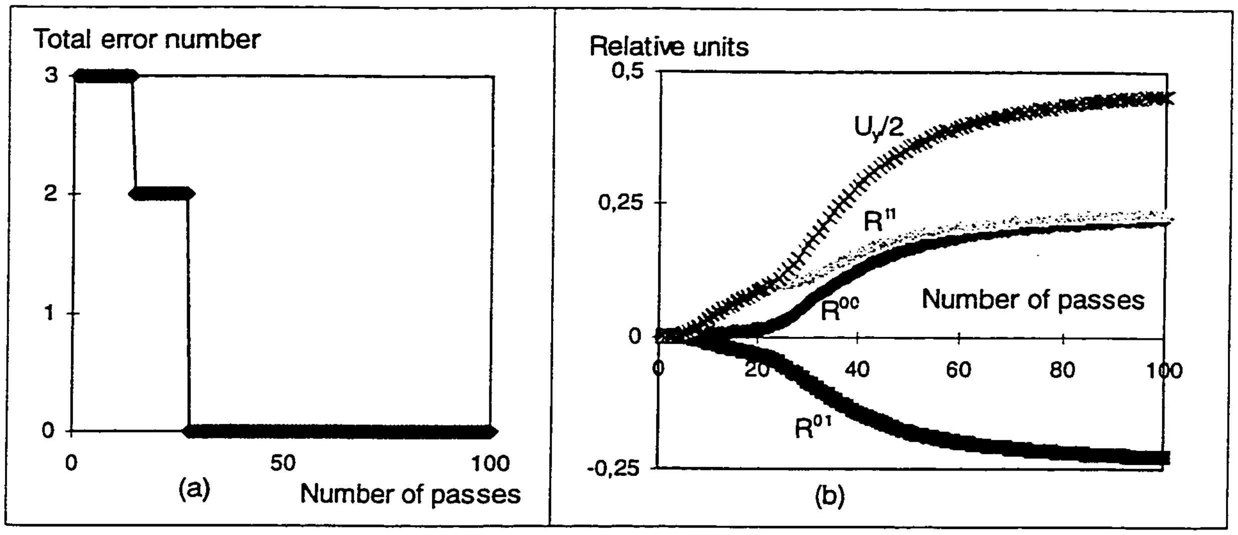


Figure IV-3-1. Parity problem: (a) total error vs number of passes; (b) $R^{\alpha\alpha'}$'s vs number of passes.

At pass #28 the system has reached the zero error state: each example of the LS is associated with its proper class. If the learning session is not terminated at this point, however, the learning continues, in the sense that the $R^{\alpha\alpha'}$s move ever closer to their limiting values. As a consequence, the probability of an input configuration from outside the learning set being correctly classified will continue to increase. Thus overspecialisation does not occur, in contrast to backpropagation.

The Baba backpropagation algorithm never reaches the zero error state; his process terminates when his error function E is less than 0.0001. We note that this function has a non-zero value only when at least one input vector is misclassified. Moreover, though the same learning set is used, the umbrella algorithm requires only one 7-unit internal representation layer as opposed to Baba's two hidden 7-unit layers. Our zero error goal was reached in 28 passes, as opposed to Baba's best case of 447 passes for the (less stringent) $E \leq 10^{-4}$ goal. The latter is itself far faster [8] than the roughly 11,000 passes needed by conventional backpropagation (*also* for the less stringent $E \leq 10^{-4}$ goal).

The main difference between the umbrella approach and that of backpropagation is that the latter minimizes on the classical error function, which focuses on the discrepancy between the output state and the "desired" output state; whereas the former maximizes the linear combination, U, of the intra-umbrella closures and the inter-umbrella separations at each internal representation layer. The umbrella approach is also more transparent, as one can monitor the extent of the separations at each layer.

Generally it is found that zero error is reached long before the umbrella function nears its maximum. Thus, the maximization of the umbrella function is a strong condition on the problem for which the attainment of zero error is a weak condition. Continuing the learning beyond the point where zero error occurs (i.e. where linear separability first appears) involves claiming more and more of the input configuration space as being mapped into the "inside" of one or other of the umbrellas. This increase in generalization as learning takes place beyond zero error has since been observed in more advanced versions of this theory.

In the process of setting up the architecture for a given problem, there are three basic ways in which failure has been observed to occur. (i) The system may have pseudo-cycles in the values of the R's. This is easily monitored via cycling in the error number, which thus serves as a valuable failure signal. (ii) The system may learn extremely slowly (albeit constantly converging). (iii) The system may paralyze: the R's change by less than 1 part in 10^7 on each pass. In all three cases, adding neurons to the IR layers or changing various learning rates will generally solve the problem. A great advantage of our method is that (apart from case (ii)) when the system crashes, it does so very early, within 100 passes. Incidentally, for the parity problem, using the architecture (minus one hidden layer) of Baba (1989) and the same learning set, our system never did fail.

Values of the learning rates are crucial. Our best choices for the learning rates associated with the couplings are β=1, $|\gamma|$ = 1.8. For other choices, e.g., if β and $|\gamma|$ for the θ's only are increased by a factor of 10, the system "paralyzes", as it does when all the β's and $|\gamma|$'s are too small. Experience shows that if all the learning rates are too large, the learning takes place far too erratically.

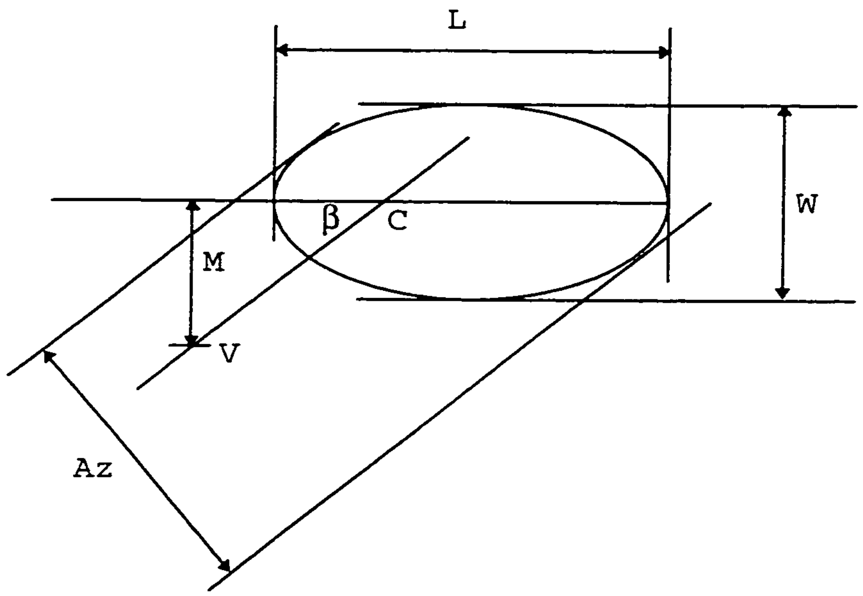
V - HIGH ENERGY GAMMA RAYS DETECTION

V-1 Description of the problem and data

Very high energy (10^{11} to 10^{16} eV) gamma rays of cosmic origin enter the atmosphere. At about 10 km altitude they interact with nuclei and generate cascades of relativistic particles. These particles produce Cerenkov radiation, typically 300-450 nm. The Cerenkov radiation is collected by an optical telescope equipped with photomultiplier tubes at the focus and the Cerenkov flashes are recorded (Harwit 1988, Fegan 1994, 1995). The elliptical flash of light produced in the photomultiplier plane is characterized by 8 parameters, the Hillas parameters; 6 are geometric and 2 are intensities (Danaher et al. 1993, Fegan 1993). The six geometric parameters are illustrated in Figure V-1-1. The two intensity measures are the ratio of the two largest tube signals (called Frac2 or F2) and the ratio of the first and the third largest tube signals (called Frac3 or F3). The events are recorded together with the time of their arrival. There is also an energy measure, called size, which is not used in the discrimination procedures because it is a quantity crucial to the determination of the spectral characteristics of the source.

The problem is that gamma rays are not the only cause of Cerenkov flashes; hadrons can also produce Cerenkov flashes, and the hadron flux outnumbers the gamma flux by a factor of a few thousands.

To detect gamma signals, a flat field procedure is followed: the events are recorded for a certain amount of time (15 or 30 minutes) with the telescope pointing to a potential source (this is the so called data ON); then the events are recorded for the same amount of time with the telescope pointing slightly off the source (this produces the so called data OFF). One ON and the corresponding OFF files, together, constitute a pair. The typical number of events in the ON and OFF files is 10000 to 30000. The files have to be filtered and the excess of ON gammas over OFF gammas reveals the activity of the source.



C: Image centroid
V: Center of the telescope field
β: Angle between the major axis and VC
Az: Azimuthal width
W: Width
M: Miss
D: Dist=VC
L: Length
Al: Alpha=sin(β)

Figure V-1-1 Hillas parameters

Three types of files were received from the Whipple Collaboration:

- 19 pairs of 1993-94 of the Crab nebula, about 60 Mb in my file format.

- 17 pairs of 1994 for the Markarian 421 (active galactic nuclus), about 120 Mb.

- one file of 1000 simulated gamma ray events as they would be recorded.

The detailed enumeration of pairs is given in Tables V-1-1 and V-1-2.

Table V-1-1 Markarian files

file name	Status	Nb events	Nb Gamma	Supercut excess in σ
m1185	OFF	24072	300	
m1186	ON	25769	21	-
m1187	ON	29010	17	
m1188	OFF	29093	369	-
m1247	ON	36009	725	
m1248	OFF	36316	622	2.806
m1269	ON	31674	635	
m1270	OFF	31291	503	3.913
m1288	ON	34217	691	
m1289	OFF	31477	587	2.909
m1370	ON	29295	510	
m1371	OFF	31617	543	-
m1384	OFF	31752	631	
m1385	ON	31991	668	1.027
m1410	ON	32882	539	
m1411	OFF	33068	466	2.303
m1431	ON	31967	611	
m1432	OFF	31301	495	3.488
m1501	ON	33696	549	
m1502	OFF	33903	481	2.119
m1521	ON	36496	723	
m1522	OFF	36863	690	0.878
m1574	ON	32760	578	
m1575	OFF	32813	467	3.434
m1576	ON	32666	594	
m1577	OFF	32453	455	4.292
m1654	ON	34375	702	
m1655	OFF	33919	572	3.642
m1656	ON (lost)			
m1657	OFF(not used)	32410	597	-
m2021	ON	25464	357	
m2022	OFF	25570	296	2.387
m2065	ON	25113	374	
m2066	OFF	24885	249	5.008

Table V-1-2 Crab nebula files

file name	Status	Nb events	Nb Gamma	Supercut excess in σ
cr1110	OFF	9471	10	
cr1111	ON	9639	14	0.816
cr1145	ON	9688	18	
cr1146	OFF	9440	13	0.898
cr1164	ON	4608	10	
cr1165	OFF	4537	4	1.604
cr1183	ON	5478	6	
cr1184	OFF	5443	11	-
cr1206	OFF	10214	9	
cr1209	ON	10389	23	2.475
cr1223	OFF	9668	6	
cr1224	ON	9624	9	0.775
cr1240	ON	9460	18	
cr1241	OFF	9558	14	0.707
cr1253	ON	9062	22	
cr1254	OFF	9004	10	2.121
cr1255	ON	3464	3	
cr1256	OFF	3619	5	-
cr1401	ON	10826	27	
cr1402	OFF	10586	10	2.795
cr1415	ON	10172	19	
cr1416	OFF	10222	9	1.890
cr1436	ON	10129	20	
cr1437	OFF	10137	16	0.667
cr1438	ON	10361	14	
cr1439	OFF	10067	10	0.816
cr1609	OFF	11054	15	
cr1610	ON	11317	18	0.522
cr1611	ON	11183	17	
cr1612	OFF	11553	20	-
cr1613	ON	4638	4	
cr1614	OFF	4998	8	-
cr1656	ON	10247	21	
cr1657	OFF	10566	18	0.480
cr1823	OFF	17038	14	
cr1824	ON	16915	37	3.221
cr1825	ON	10850	15	
cr1826	OFF	11121	9	1.225

V-2 Supercuts

To determine the gamma excess, numerous filters, based on statistical methods have been applied (Punch 1990, Punch 1991, Paré et al. 1991, Reynolds 1993,Fegan 1994, Fegan 1995). Even an artificial neural network with backpropagation learning algorithm has been tried (Reynolds 1994). But it seems that one of the most efficient and universal methods, called supercuts, is simply to consider an event being a gamma if four of its Hillas parameters lie within chosen intervals: Table V-2-1 (Punch et al. 1991).

TABLE V-2-1 Supercut intervals

$$0.16 < \text{Length} < 0.3$$
$$0.073 < \text{Width} < 0.150$$
$$0.51 < \text{Dist} < 1.1$$
$$\text{alpha} < 0.258819 = \sin(15^\circ)$$

The results of the supercuts applied to the files received are given in Tables V-1-1, V-1-2 for the details and in Table V-2-2 for the totals. The **excess in sigma** is also given, being the standard measure of excess when the event probabilities are very low:

$$\text{excess in sigma} \equiv \frac{\text{Ngon} - \text{Ngoff}}{\sqrt{\text{Ngon} + \text{Ngoff}}}$$

where Ngon and Ngoff are the numbers of gammas in the ON and OFF files respectively that pass through the statistical or neural network filters.

TABLE V-2-2 Original files, supercuts, and excess in sigma.

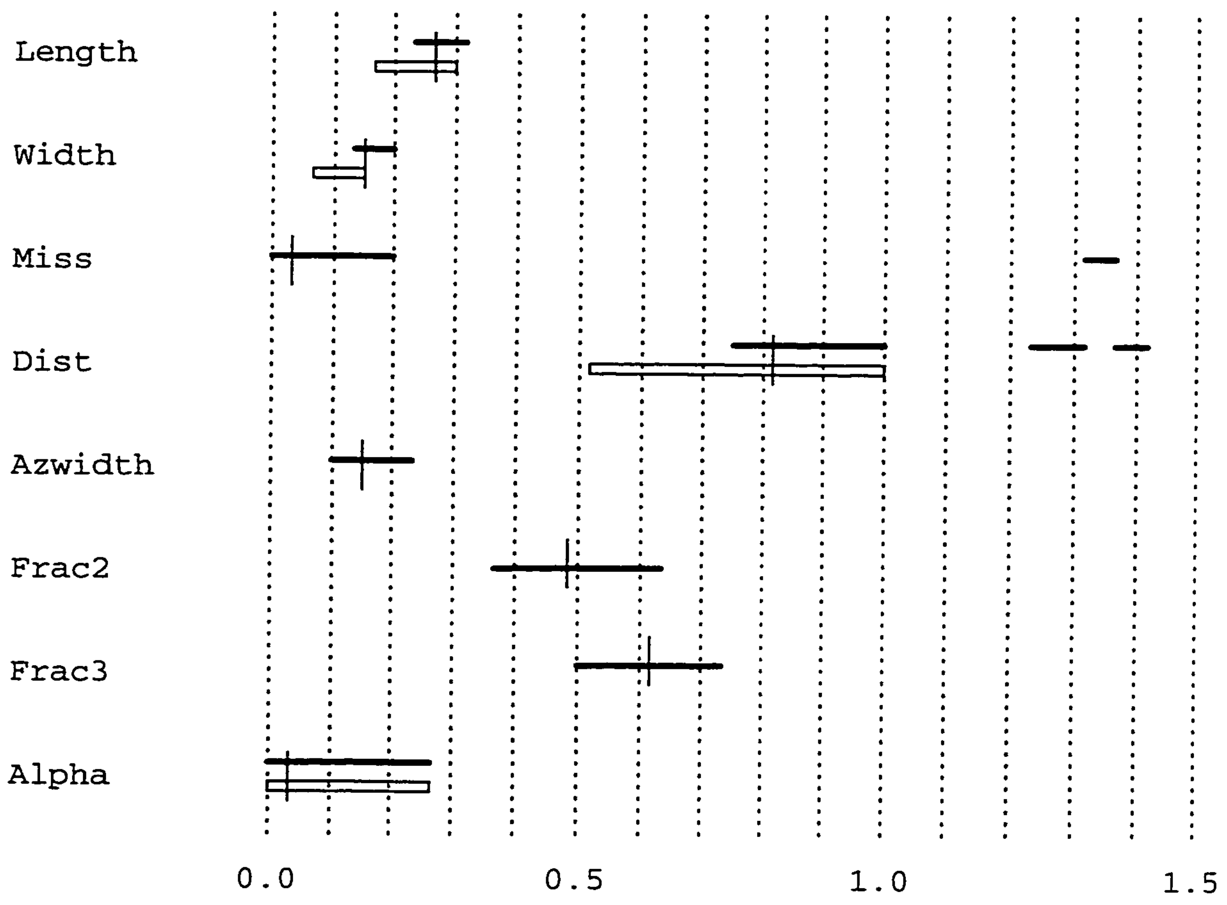
Source	Nb pairs	Ngon (all pairs)	Ngoff (all pairs)	excess in sigma
Markarian 421	16	8294	7726	4.49
Crab nebula	19	315	211	4.53

The published results for the supercuts, for the Crab data are 1828 gamma from the ON files, 940 OFF leading to an excess in sigma of 16.88 (Fegan 1995). The pair m1656, m1657 of Table V-1-1, missing in the present analysis, cannot explain the discrepancy. It is possible that the files received were preprocessed (by the Whipple collaboration) and did not correspond to the same data; the figures will be used for comparison purposes only.

V-3 Neural network classifier

To use the umbrella algorithm in a data discrimination problem one needs a reliable learning set, conformal to the problem at hand. The simulations are not completely reliable since supercuts, which seems to be the best discriminating method available, rejects 50% of the 996 valid events of the simulation file (see Table V-3-5). Since the quality of the learning set is crucial, it was decided, after many unfruitful attempts, to use a Bayesian separator with the species to select some real events for a learning set. Part of the data was used to construct the histograms of all the parameters, 75 bins of 0.02 width. (the following four Markarian 421 OFF files: 1185, 1188, 1248 and 1371 for Class 0 (hadron) type of events and the gamma simulations for the Class 1 type of events.)

The intervals where the gamma simulations histogram frequency value is higher than the OFF data were chosen for the Bayesian separator as in Sections IV-2-2 and IV-2-3. For instance the simulated gamma frequency of events is higher than the OFF data, for the Length (first parameter), in the interval 0.22 to 0.32; see Figure V-3-1.



━━: Bayesian intervals where the simulation histogram frequencies are larger than the OFF data frequencies.
▭: Supercut intervals (Table V-2-1).

The vertical bars indicate the most frequent value of the parameter in the simulations.

Length	0.22-0.32	mfv=0.27		
Width	0.12-0.20	mfv=0.15		
Miss	0.0-0.20 & 1.32-1.36		mfv=0.03	
Dist	0.74-1.0 & 1.26-1.32 & 1.36-1.42			mfv=0.81
Azwidth	0.10-0.22	mfv=0.15		
Frac2	0.36-0.64	mfv=0.49		
Frac3	0.50-0.74	mfv=0.61		
Alpha	0.0-0.26	mfv=0.03		

Figure V-3-1 Bayesian intervals based on Markarian 421 data.

The species were assigned to the events in the following manner: for each event, construct a binary number of 8 bits, one bit per parameter, having a 1 when the event parameter is in the interval (or one of the intervals when it occurs) where the gamma distribution dominates; and zero otherwise. The binary number constitutes the species of the event (there is only one possible species per event). There are $2^8=256$ species, with binary numbers representing 0 to 255; the events in species number zero have none of their parameters in the intervals where the gamma distribution dominates, the species number 1 represents those events having their last parameter in the interval of alpha where the gamma distribution dominates, the species number 2 contain events having their Frac3 value in the interval where the gamma dominates, events in species number 3 have their alpha and Frac3 values in the respective intervals where the gamma dominates; and so on...

The species of all the events from all the files, ON and OFF, were determined by a "service program"; the results are given in Table V-3-1 for Markarian 421 and Table V-3-2 for the Crab nebula. The complete species table is given in Appendix VIII.

TABLE V-3-1 Markarian most significant species (species leading to an excess in sigma above 3) for all 16 pairs.

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	EX>2.999	Zero OFF	empty
50	0	0	0	0	-	empty
52	0	0	0	0	-	empty
58	0	0	0	0	-	empty
65	0	0	0	0	-	empty
67	0	0	0	0	-	empty
69	0	0	0	0	-	empty
71	1	0	1	0	Zero OFF	-
78	8022	7646	3.00387	1	-	-
83	1	0	1	0	Zero OFF	-
99	0	0	0	0	-	empty
101	0	0	0	0	-	empty
103	0	0	0	0	-	empty
111	1823	1523	5.186308	1	-	-
112	0	0	0	0	-	empty
114	0	0	0	0	-	empty
115	0	0	0	0	-	empty
116	0	0	0	0	-	empty
117	0	0	0	0	-	empty
118	0	0	0	0	-	empty
122	0	0	0	0	-	empty
124	0	0	0	0	-	empty
127	1165	960	4.447074	1	-	-
139	3	0	1.732051	0	Zero OFF	-
147	0	0	0	0	-	empty
163	5	0	2.236068	0	Zero OFF	-
178	0	0	0	0	-	empty
180	0	0	0	0	-	empty
182	0	0	0	0	-	empty
184	0	0	0	0	-	empty
186	0	0	0	0	-	empty
188	0	0	0	0	-	empty
191	1665	1493	3.060711	1	-	-
193	0	0	0	0	-	empty
195	0	0	0	0	-	empty
197	0	0	0	0	-	empty
199	0	0	0	0	-	empty
209	0	0	0	0	-	empty
211	0	0	0	0	-	empty
213	0	0	0	0	-	empty
215	0	0	0	0	-	empty
225	0	0	0	0	-	empty
227	0	0	0	0	-	empty
229	0	0	0	0	-	empty
231	0	0	0	0	-	empty
240	0	0	0	0	-	empty

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	EX>2.999	Zero OFF	empty
241	0	0	0	0	-	empty
242	0	0	0	0	-	empty
243	0	0	0	0	-	empty
244	0	0	0	0	-	empty
245	0	0	0	0	-	empty
246	0	0	0	0	-	empty
247	0	0	0	0	-	empty
250	0	0	0	0	-	empty
255	1802	1295	9.1103	1	-	-

TABLE V-3-2 Crab nebula (19 pairs) most significant species chosen with the Markarian determined intervals (not used in the construction of the LS.

SPECIES	TOTAL ON	TOTAL OFF	EXCESS IN SIGMA	EX>2.999	Zero OFF	empty
17	0	0	0	0	-	empty
19	0	0	0	0	-	empty
21	0	0	0	0	-	empty
23	0	0	0	0	-	empty
25	0	0	0	0	-	empty
27	0	0	0	0	-	empty
29	0	0	0	0	-	empty
31	0	0	0	0	-	empty
50	0	0	0	0	-	empty
51	0	0	0	0	-	empty
53	1	0	1	0	Zero OFF	-
55	3	0	1.732051	0	Zero OFF	-
58	0	0	0	0	-	empty
59	0	0	0	0	-	empty
61	0	0	0	0	-	empty
65	2	0	1.414214	0	Zero OFF	-
67	0	0	0	0	-	empty
69	0	0	0	0	-	empty
71	0	0	0	0	-	empty
77	27	9	3	1	-	-
81	0	0	0	0	-	empty
83	0	0	0	0	-	empty
85	0	0	0	0	-	empty
87	0	0	0	0	-	empty
89	0	0	0	0	-	empty
91	0	0	0	0	-	empty
93	0	0	0	0	-	empty
95	0	0	0	0	-	empty
99	0	0	0	0	-	empty
101	0	0	0	0	-	empty

SPECIES	TOTAL ON	TOTAL OFF	EXCESS IN SIGMA	EX>2.999	Zero OFF	empty
103	0	0	0	0	-	empty
112	0	0	0	0	-	empty
113	0	0	0	0	-	empty
114	0	0	0	0	-	empty
115	0	0	0	0	-	empty
116	0	0	0	0	-	empty
117	0	0	0	0	-	empty
118	0	0	0	0	-	empty
119	0	0	0	0	-	empty
122	1	0	1	0	Zero OFF	-
123	0	0	0	0	-	empty
145	0	0	0	0	-	empty
147	0	0	0	0	-	empty
149	0	0	0	0	-	empty
151	0	0	0	0	-	empty
153	0	0	0	0	-	empty
155	0	0	0	0	-	empty
157	0	0	0	0	-	empty
159	0	0	0	0	-	empty
177	0	0	0	0	-	empty
178	1	0	1	0	Zero OFF	-
179	0	0	0	0	-	empty
181	0	0	0	0	-	empty
185	0	0	0	0	-	empty
186	1	0	1	0	Zero OFF	-
187	0	0	0	0	-	empty
191	0	0	0	0	-	empty
193	0	0	0	0	-	empty
195	0	0	0	0	-	empty
197	0	0	0	0	-	empty
199	0	0	0	0	-	empty
209	0	0	0	0	-	empty
211	0	0	0	0	-	empty
213	0	0	0	0	-	empty
215	0	0	0	0	-	empty
217	0	0	0	0	-	empty
219	0	0	0	0	-	empty
221	0	0	0	0	-	empty
223	0	0	0	0	-	empty
225	0	0	0	0	-	empty
227	0	0	0	0	-	empty
229	0	0	0	0	-	empty
231	0	0	0	0	-	empty
239	1982	1435	9.357608	1	-	-
240	0	0	0	0	-	empty
241	0	0	0	0	-	empty
242	0	0	0	0	-	empty
243	0	0	0	0	-	empty

	TOTAL	TOTAL	EXCESS			
SPECIES	ON	OFF	IN SIGMA	EX>2.999	Zero OFF	empty
244	0	0	0	0	-	empty
245	0	0	0	0	-	empty
246	0	0	0	0	-	empty
247	0	0	0	0	-	empty
251	0	0	0	0	-	empty
253	0	0	0	0	-	empty
255	5	5	0	0	-	-

The significant features of the species tables (see also Appendix VIII) are

a) the species marked "empty" contain no event at all revealing a complex distribution of events in the 8-dimensional parameter space.

b) some species (marked Zero OFF) do have some events in the ON files but no events in the OFF files and that is for all the Markarian 421 pairs. Although the number of events in that case is very low, these events may represent some specific source activity.

250 events (from the 4 OFF data files) that were in the 255 gamma species and the 150 first events from the ON data (in the 255 species as well) were used as a learning set. This is not exactly the same as the toy data methodology where species were used for both classes. The rationale here is that the 255 species, with an excess in sigma of 9.11, must hide some source signature and one can try to make a neural network learn that signature to eventually discriminate, from the noise, the events carrying that signature.

The neural networks never learned it far enough for the generalization to be reliable as it was in the EXOR and the Toy Data experiments; the monitoring umbrella function barely reached 1.5 and zero error on the learning set was never attained regardless of the architecture tried (for example: 6 layers of 8 cells and, another "large" one: 3 internal representation layers of 17,11, and 7 cells) and learning times allowed up to 60000 passes. Typical best results for a four layer (8,17,7,5) architecture, after 36000 learning passes, are given in the Table V-3-3 for the learning performance.

TABLE V-3-3 Learning performance (layer 0 is the input)

	Similarity measures			
Layer	R00	R10	R11	W
1	0.22	-0.17	0.14	0.71
2	0.33	-0.25	0.21	1.05
3	0.42	-0.34	0.28	1.37

The output protocol was a calculated perceptron connected on layer 3, with a zero threshold; the comparison protocol produced almost the same results as the calculated perceptron. The learning set discrimination is in Table V-3-4

TABLE V-3-4 Learning set discrimination

Data type	Events in LS	Gamma class	Hadron class
OFF	250	46	204
ON	150	116	34

In order to quantify the filtering performance a quality factor (Q) was defined for events of well known class. Let N_{g0} be the number gammas in the test set, N_g the number of gammas passing through the filter; with N_{h0} and N_h similarly defined for the hadrons; then $Q \equiv \frac{N_g / N_{g0}}{\sqrt{N_h / N_{h0}}}$ (see for example Reynolds 1994).

The quality factor measured on the learning set is $Q = \frac{116 / 150}{\sqrt{46 / 250}} = 1.80$ which is not very high, astrophysics standards being around or above 8.0. The smallest quality factor measured on test sets constructed with randomly chosen events from the simulation file and events from the Markarian OFF files was 3.6.

Table V-3-5 contains simulations classifications made by supercuts, the Bayesian separator used to construct the learning set (species 255 in Table A-VIII-3) and the neural network classification. The neural network recognizes 63% of the simulations while supercuts recognizes 50% and the Bayesian separator 42%. Even though the W function was not very high, the excess in sigma (8.18, Table V-3-6) obtained by the neural network classifier together with the 63% might for some purposes be a better result than the Bayesian separator with an excess in sigma of 9.11 (Table V-3-6) but only 42% of the simulations properly classified.

TABLE V-3-5 Simulation separation

Separator	Nb. passing events	Nb. rejected events
Supercuts	497	499
Bayesian	417	579
Neural network	630	366

Tables V-3-6 and V-3-7 are the discrimination performance for the Markarian and the Crab sources respectively; both were obtained with the same neural network, trained with the Markarian species 255; the tables also contain the number of events in the 255 species for each file, for comparison purposes.

TABLE V-3-6 Neural network classification of Markarian 421 data.

		Neural network		Bayesian separator	
file name	Status	Nb Gammas	excess	Nb Gammas	excess
m1185	OFF	688		54	
m1186	ON	777	2.30	95	3.36
m1187	ON	934		105	
m1188	OFF	809	2.99	50	4.42
m1247	ON	1225		141	
m1248	OFF	1118	2.21	109	2.02
m1269	ON	1218		128	
m1270	OFF	1153	1.33	89	2.65
m1288	ON	1293		107	
m1289	OFF	1161	2.66	98	
m1370	ON	1024		92	
m1371	OFF	1008	0.35	87	0.37
m1384	OFF	1190		107	
m1385	ON	1297	2.15	98	0.77
m1410	ON	1062		120	
m1411	OFF	1000	1.37	78	2.98
m1431	ON	1166		105	
m1432	OFF	992	3.75	89	1.15
m1501	ON	1023		117	
m1502	OFF	1055	-	76	2.95
m1521	ON	1319		132	
m1522	OFF	1264	1.08	115	1.08
m1574	ON	1086		123	
m1575	OFF	994	2.02	83	2.79

		Neural network		Bayesian separator	
file name	Status	Nb Gammas	excess	Nb Gammas	excess
m1576	ON	1143		126	
m1577	OFF	962	3.95	74	3.68
m1654	ON	1178		131	
m1655	OFF	1023	3.04	78	3.67
m2021	ON	715		76	
m2022	OFF	652	1.70	60	1.37
m2065	ON	685		96	
m2066	OFF	593	2.57	58	3.06
Total ON		17145		1802	
Total OFF		15663	8.18	1295	9.11

The neural network classification, with an excess of 1482 events, almost triples the excess of the Bayesian separator (507) and the supercuts (526 in Table V-2-2).

TABLE V-3-7 Neural network classification of Crab data (The Markarian 255 species Bayesian separator gives very bad results for the crab, see Table V-3-2; the overall excess in sigma for that case is 0.00).

		Neural network	
file name	Status	Nb Gammas	excess
cr1110	OFF	183	
cr1111	ON	183	0.0
cr1145	ON	189	
cr1146	OFF	193	-
cr1164	ON	95	
cr1165	OFF	66	2.29
cr1183	ON	92	
cr1184	OFF	89	0.22
cr1209	ON	128	
cr1206	OFF	130	-
cr1224	ON	164	
cr1223	OFF	130	1.98
cr1240	ON	199	
cr1241	OFF	152	2.51
cr1253	ON	203	
cr1254	OFF	163	2.09
cr1255	ON	65	
cr1256	OFF	49	1.50
cr1401	ON	151	
cr1402	OFF	144	0.41
cr1415	ON	133	
cr1416	OFF	133	0.0
cr1436	ON	193	

		Neural network	
file name	Status	Nb Gammas	excess
cr1437	OFF	204	-
cr1438	ON	172	
cr1439	OFF	124	2.79
cr1610	ON	158	
cr1609	OFF	146	0.69
cr1611	ON	180	
cr1612	OFF	161	1.03
cr1613	ON	58	
cr1614	OFF	66	-
cr1656	ON	203	
cr1657	OFF	107	-
cr1824	ON	370	
cr1823	OFF	294	2.95
cr1825	ON	210	
cr1826	OFF	206	0.19
Total ON		3146	
Total OFF		2840	3.96

Again, an excess count of 306 almost triples the excess of supercuts, namely 104 as in Table V-2-2.

Tables V-1-1 and V-1-2 for the Supercuts results and Tables V-3-6 and V-3-7 for the neural network cuts, show that the neural network results are, regarding the excess in sigma, better than supercuts for the Markarian 421 source; this, despite the low values of the monitoring W-function (Table V-3-3) and the quality factors. The results are much less spectacular for the Crab source. The irony is that the Bayesian separator (Table V-3-6), used to construct the learning set, produces even better excess in sigma than the neural network for the Markarian 421 source. I was not able to reproduce, in the astrophysics context, the quality of discrimination reached in the Toy Data.

V-4 Spectral index estimates

It is possible, within the present research, to use the neural network discriminator to estimate the spectral index of the sources. Taking the size parameter as a measure of the energy received from the source; assuming also a linearity in the energy response of the apparatus one has that $N(S) \alpha S^{-\gamma}$ where N(S) is the difference between the ON and the OFF data number of events having a size larger than S, and γ is the spectral index (Harwit 1988, Gora 1994, Fegan 1995). The histogram (in bins of 100 units) of the size of the events selected by the neural network was constructed and the histogram was integrated to obtain an estimate of N(S). The results are in the Tables and Figures V-4-1 and V-4-2 for the Markarian and the Crab respectively. The equations of the lines on the graphs are:

Y = -1.986(±1) X + 7.861 in the Figure V-4-1 (the first 3 points and the last 4 were not used in the least squares fit); Y = -2.481(±1) X + 8.967 in Figure V-4-2 (the first 5 points were not used). The published spectral indices are: for observed extragalactic sources between 1.7 and 2.4 and "close to 2.0" for the Markarian 421 (Fegan 1994); 2.8 ± 0.6 (Petry et al. 1996); 2.1 ± 0.3 for the Crab (Fegan 1994).

TABLE V-4-1 Markarian size histogram of neural network selected gammas. The "Size" column contains the lower bound of the histogram bin; "ON" and "OFF" are the number of selected events in the bin; "Diff." is the difference ON-OFF, 0 when ON<OFF; "Intgr." is the sum of the Diff column entries larger than Size.

Size	ON	OFF	Diff.	intgr.
10	11	0	11	1388
110	5346	5161	185	1377
210	6785	6391	394	1192
310	1964	1654	310	798
410	584	421	163	488
510	299	174	125	325
610	154	113	41	200
710	99	74	25	159
810	78	44	34	134
910	52	26	26	100
1010	41	28	13	74
1110	19	13	6	61
1210	14	8	6	55
1310	16	5	11	49
1410	16	8	8	38
1510	8	8	0	30
1610	10	6	4	30
1710	4	4	0	26
1810	6	1	5	26
1910	5	3	2	21
2010	4	1	3	19
2110	2	2	0	16
2210	2	5	0	16
2310	4	1	3	16
2410	1	2	0	13
2510	2	2	0	13
2610	1	0	1	13
2710	3	4	0	12
2910	0	2	0	12
3010	2	0	2	12
3110	2	0	2	10
3210	1	1	0	8
3310	2	0	2	8
3410	0	1	0	6
3510	1	2	0	6
3610	0	1	0	6
3710	2	0	2	6
3910	1	0	1	4
4010	1	0	1	3
4210	0	1	0	2
4810	1	0	1	2
5110	1	0	1	1

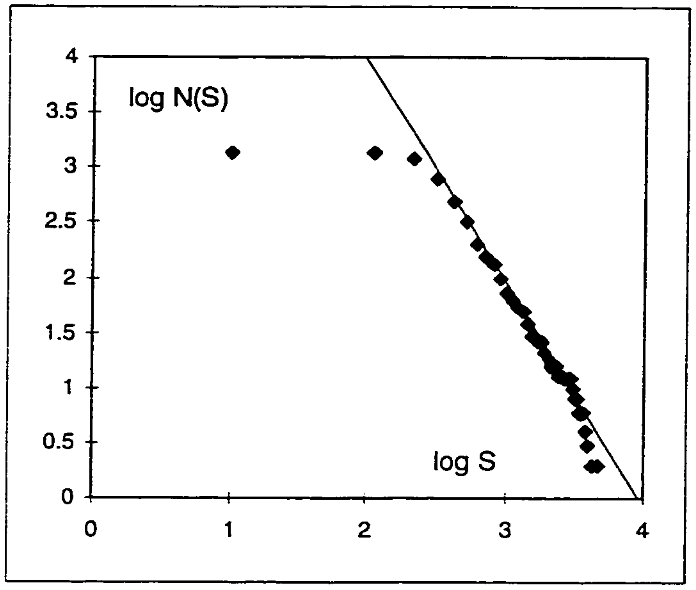


Figure V-4-1 Markarian 421 spectral index: Logarithm of the number of events with a size larger than S versus Log(S).

TABLE V-4-2 Crab size histogram of neural network selected gammas. The "Size" column contains the lower bound of the histogram bin; "ON" and "OFF" are the number of selected events in the bin; "Diff." is the difference ON-OFF, 0 when ON<OFF; "Intgr." is the sum of the Diff column entries larger than Size.

Size	ON	OFF	Diff.	intgr.
10	9	4	5	313
110	669	634	35	308
210	939	860	79	273
310	601	586	15	194
410	381	328	53	179
510	216	179	37	126
610	107	89	18	89
710	73	59	14	71
810	35	28	7	57
910	32	27	5	50
1010	26	22	4	45
1110	15	6	9	41
1210	19	9	10	32
1310	12	3	9	22
1410	7	4	3	13
1510	1	6	0	10
1610	3	2	1	10
1710	2	1	1	9
1810	3	3	0	8

1910	4	1	3	8
Size	ON	OFF	Diff.	intgr.
2010	2	0	2	5
2110	1	1	0	3
2310	1	1	0	3
2410	1	1	0	3
2510	0	1	0	3
2810	1	1	0	3
3010	1	0	1	3
3110	2	0	2	2
3910	1	1	0	0
4210	1	1	0	0
4410	0	1	0	0

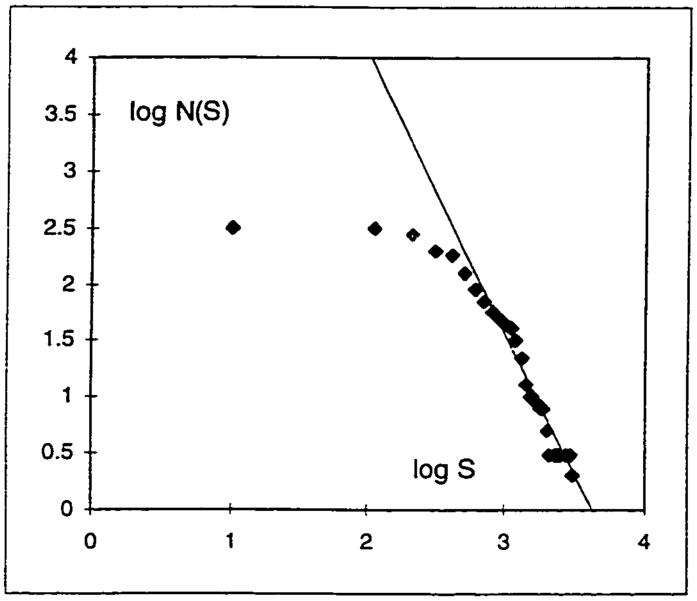


Figure V-4-2 Crab spectral index:Logarithm of the number of events with a size larger than S versus Log(S).

VI CONCLUSION AND FINAL DISCUSSION

Conclusion

The Umbrella Algorithm has been shown to be far faster than backpropagation for those problems where comparison could be made. Moreover, the Umbrella Algorithm has been shown to have properties such as learning beyond zero error, resulting in increased generalization, not possessed by the vast majority of learning algorithms in the literature.

It is desirable here to review the steps leading to the unprecedentedly large number of ON and OFF photons (with a concomitantly high excess in sigma) for Markarian 421. (i) The data were divided into species, depending on the ranges of the parameters for which the simulations had higher frequencies than four (4) randomly selected OFF files. In particular, the 255 species was defined as having all eight of the Hillas parameters with higher frequencies in the simulations than in these OFF files. (ii) This 255 species led to an excess in sigma of about 9 when all 32 ON & OFF files were used. This excess in sigma was about a factor of 2 larger than supercuts, while the number of ON-minus-OFF photons produced was about the same. If there is a signature to the gammas, it is contained within the 255 species. (iii) An architecture was constructed using the Umbrella Algorithm on the learning set consisting of events, from simulation data and the same four OFF-data files, whose parameters belong to the 255 species. That this is a "hard problem" is evidenced by the fact that the monitoring function W reached approximately 1.2 out of a maximum of 4. (iv) This architecture, when applied to the entire Markarian 421 set of files, produced an excess-in-sigma of 8.7 (i.e., not significantly different from the previous value 9), but with, now, about triple the number of ON-minus-OFF photons. The spectral function was plotted to find a spectral index of approximately 2.5 in the lower-energy region.

Compared to supercuts, the large photon flux found here, occurs chiefly in the high-energy region. It is not yet understood whether this high flux, a result of the combination of choice-of-learning-set and Umbrella Algorithm, is due mainly to the one or the other. This relatively large photon flux promises to be of great use in the analysis of other high-energy extra-galactic sources which may be fainter than Markarian 421.

Final discussion

In the Toy Data experiment the umbrella algorithm was seemingly able to recognize a signature of the events classes leading to a better discrimination quality than the Bayesian separator used to generate the learning set. In the

EXOR it also showed some generalization capabilities. It has some very short learning times. In the Gamma/Hadron discrimination problem the neural network does not seem to recognize a signature of the gamma events such as that found in the Toy Data; this means that the signature either does not exist or is not accessible to the umbrella method employed. Although theoretically better, through the nonlinearity involved in the neural network connections, a difficult data discrimination problem is best treated by a much simpler Bayesian separator.

If the signal is really random, the neural network cannot find a signature and its performance will, in the best case, be the same as statistics-based filters. In the Toy Data case the neural network may actually produce the same result as a Bayesian separator using higher moments as well; this is an open question. As pointed out by many authors, the neural network approach is not very far from statistical methods like singular value decomposition or other techniques also using higher moments but with the advantage of nonlinearity (see for example Westerhoff 1995).

Further developments may study the behaviour of the umbrella algorithm in problems having more than two classes; implement an annealing of some sort that has improved many algorithms in the neural network field; implement lateral inhibition to manage competition between neurons as has been shown very efficient in feature extraction techniques (Kohonen 1989). The umbrella ideas may even be useful in conjunction with an error-based learning algorithm. The umbrella algorithm, in that sense, may be seen as at least complementary to the backpropagation type of algorithm.

REFERENCES

Abeles, M.; 1991; Corticonics, neural circuits of the cerebral cortex; Cambridge University Press; 280p.

Aleksander, Igor & Morton, Helen; 1990; An introduction to neural computing; Chapman and Hall, London.

Alkon, Daniel L.; 1983; Learning in a marine snail; Sc. Am.; July 83, p70.

Alkon, Daniel L.; 1989; Memory storage and neural systems; Sc. Am.; July 89, p42

Alt Franz, L.; 1962; Digital pattern recognition by moments in Optical character recognition; ed. Georges L. Fisher Jr et al; Spartan books Washington 1962.

Anderson, J. A. & Rosenfeld, E. (Ed.); 1989; Neurocomputing: Foundation of Research; MIT Press, Cambridge; 729p.

Axelrod, Julius; 1974; Neurotransmitters; Sc. Am.; June 74, p59.

Baba, Norio; 1989; Neural Networks; **2,** p367.

Blanc, M. Ed.; 1988; La recherche en neurobiologie; Edition du Seuil; 375p.

Blayo, François & Verleysen, Michel; 1996; Les réseaux de neurones artificiels; Presse Universitaires de France, Que sais-je?; 126p.

Cooper, Leon N.; 1988; Brain research: theory and experiment; Computers in physics; p29; nov-dec. 88.

Cotterill, Rodney M. J.; 1986; The brain: An intriguing piece of condensed matter; Physica Scripta, **T13**, p161-168.

Danaher S. et al.; 1993; Application of singular value decomposition in high energy gamma-ray astronomy, Astroparticle physics **1** 357.

de Bono, E.; 1969; The Mechanism of mind; Penguin Books 281p.

Dowling, John E.; 1992; Neurons and networks, an introduction to neuroscience; The Belknap Press of Havard U P; 447p.

Eccles, Sir John; 1956; The synapse; Sc. Am., Jan. 56; p6.

Fegan D.; 1994; Gamma ray astronomy above 0.1 TeV; in High energy astrophysics; Matthews James, ed.; World Scientific; Singapore.

Fegan D. J. et al.; 1995; The processing and analysis of TeV gamma-ray images; internal UCD publication.

Frank B.; 1995; Presentation of the author's main umbrella algorithm ideas to a meeting at Mount Hopkins TeV Astrophysics Collaboration Group; Jan. 27; UCD Ireland.

Frean M.; 1990; Neural Computation; **2**, p198.

Gora Dal; 1994; Comments on ISU/Leeds crab spectra; internal UCD publication.

Grossman T., Meir R., Domany E.; 1988 Complex Systems; **2**, p555.

Hankinson, Bob & Hermida, Alfonso; 1994; Hidden images: making random dot stereograms; QUE corporation, Indianapolis.

Harwitt Martin; 1988; Astrophysical concepts; Springer-Verlag; Berlin.

Haykin, Simon; 1994; Neural networks, a comprehensive foundation; Macmillan College Publishing Company; 696p.

Hebb, D. O.; 1949; The organization of the behaviour; Wiley, New York.

Hecht-Nielsen, Robert; 1989; Neurocomputing; Addison Wesley, New York; 433p.

Hinton, G. E.; 1989; Distributed representations; Ch. 3 in Rumelhart & McClelland 1989.

Hinton, G. E.; 1989b connectionists learning procedures; Artificial Intelligence; **40**, p185.

Hopfield, J.J.; 1982; Neural networks and physical systems with emergent collective computational abilities; Proceedings of the National Academy of Science; **79**, 2554-2558.

Hubel, David H.; 1979; The brain; Sc. Am., 44, Sept. 79

Kirkpatrick, S. et al; 1983; Optimization by simulated annealing; Science; **220** p671.

Kohonen, T.;1989; Self-organisation and associative memory; Springer-Verlag; 312p.

Kolmogorov Andrei, 1957 cited in Hecht-Nielsen 1989; p122.

Kosko, Bart; 1992; Neural networks and fuzzy systems, a dynamical approach to machine intelligence; Prentice Hall, 449 p.

Lafortune Jacques; 1991; personal communication.

Leshno, M et al.;1995; Multilayer feedforward networks with a nonpolynomial activation function can approximate any function; Neural Networks; **8,** p31-37.

Lindsay, Peter H. & Donald A. Norman; 1977; Human Information Processing, An introduction to psychology; Academic press N.Y.; 777p.

Lukashin, A. V. et al; 1987; Physical models of neurone networks; Biophysics, **32,** 1000-1012.

Macot L.& Frank B; 1996; Umbrella algorithm; PC1 in poster session; CAP congress, Ottawa.

McCulloch, W. S. & Pitts, W.;1943; A logical calculus of the ideas immanent in nervous activity; Bull. Math. Biophys.; **5** p115-133.

Mézard M. & Nadal J. P.; 1989; J. Phys. A: Math. Gen.; **22** p2191.

Minsky, Marvin & Papert, Seymour; 1969; Percetrons; MIT Press, 258p.

Morrison, Philip & Morrison, Emily (Edited by); 1961; Charles Babbage and his calculating engines, Selected Writings by Charles Babbages and Others; Dover Publications; 400p.

Paré E. et al.; 1991; Image shapes of showers in UV and visible Cerenkov light; 22nd International cosmic ray conference; Dublin, August 91; Vol. I.

Petry et al.; 1996; Detection of VHE gamma rays from Markarian 421 with the HEGRA Cerenkov Telescopes; Astronomy and Astrophysics; **L13** 311.

Punch Michael et al.; 1990; Simple multi-parameter cluster analysis applied to the crab database; internal publication UCD.

Punch Michael et al.; 1991; Supercuts: an improved method of selecting gamma rays; 22b<nd International cosmic ray conference; Dublin, August 91; Vol I.

Reynolds P. T. et al.; 1993; Application of a maximum likelyhood classifier in TeV gamma-ray astronomy; J. Phys. G: Nucl. Part. Phys. **19** 1217.

Reynolds P. T. and Fegan D.; 1995; Neural network classification of TeV gamme-ray images; Astroparticle Physics, **3** 137.

Rumelhart, David E., McClelland, James L. and the PDP Research Group; 1986; Parallel distributed processing Vol 1: Foundations; MIT Press; 547p.

Rosenblatt, F.; 1962; Principles of neurodynamics; Spartan Books, New York.

Selfridge, O.; 1959; Pandemonium: a paradigm for learning in symposium in mechanization of thought processes; HM Stationery Office, London.

Sompolinsky, Haim; 1988; Statistical mechanics of neural networks; Physics Today **70**. dec.88.

Smolensky, P. & Riley, M. S.; 1984; Harmony theory: problem solving, parallel cognitive models and thermal physics; Tech. Rep. No 8404; La Jolla, Unversity of California, San Diego, Institute for Cognitive Science.

Smolensky, P.; 1986; Information processing in dynamical systems: foundation of harmony theory; in Rumelhart et al. 1986.

Stevens, Charles F.; 1979; The Neuron; Sc. Am.; Sept. 79,p55.

Valet F. et al; 1989 Europhys. Lett.; **9**, (4) p315.

Wasserman, Philip D.;1989; Neural computing: Theory and practice; Van Nostrand Reinhold; 230p.

Watkin L. H. et al; 1993; Rev. Mod. Phys.; **65**, p499.

Westerhoff S et al.; 1995; Separating gamma ad hadron induced cosmic ray showers with feed-forward neural network; Atroparticle Physics; **4** 191.

Willows, A. O. D.; 1971; Giant brain cells in mollusks; Sc. Am.; feb. 71, p69.

Willshaw, David; 1981; Hollography, associative memory and inductive generalization, in parallel models of associative memory; Hinton G. E. & Anderson J. A. Ed.; Lawrence Erlbaum Associates; Hillsdale, New Jersey.

APPENDIX I

I-1: Biological bases

Here presented is the biological consensus that constitute the fundamental background for all artificial neural network research. (Willows 1971, Stevens 1979, Blanc 1988, Abeles 1991, Dowling 1992). Also discussed is the model of a neuron used in the present work.

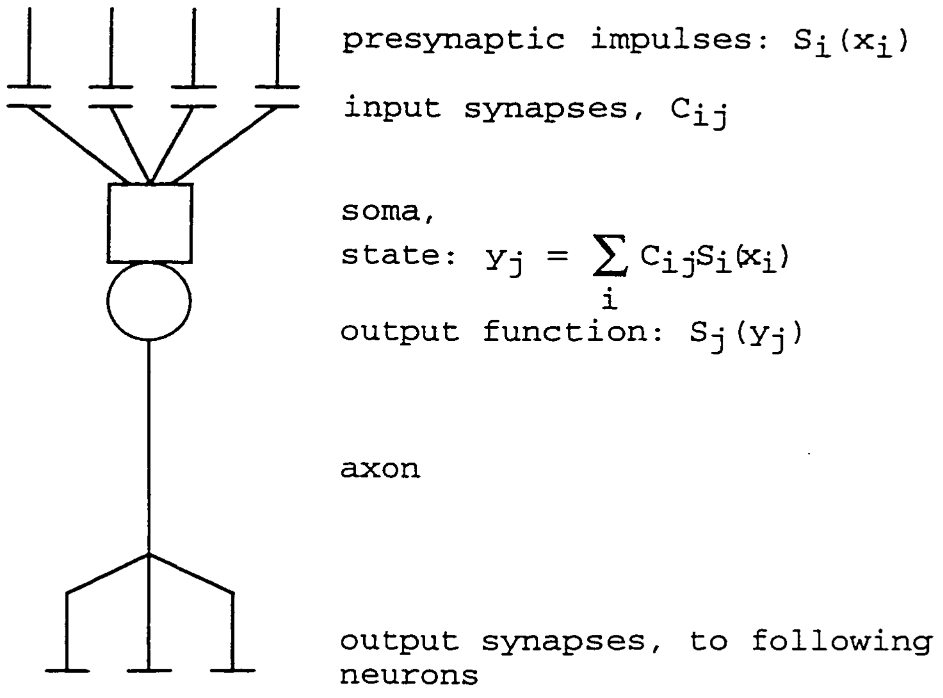

presynaptic impulses: $S_i(x_i)$
input synapses, C_{ij}
soma,
state: $y_j = \sum_i C_{ij} S_i(x_i)$
output function: $S_j(y_j)$
axon
output synapses, to following neurons

Figure A-I-1. Essential anatomy of the neuron

The cell possess a tree-like structure that comprises many dendrites and one axon. The dendrites receive the signals , from other neurons or through the senses, through input synapses. The synapse is the neuron-neuron junction (Eccles 1956). The synapses on the dendrites are the neuron inputs and the synapses at the axon terminations are the neuron outputs. The neuron sends its messages through its axon, which terminates in a tree-like structure, each branch ending with a synapse on some other neuron's dendrite.

A nerve impulse is an electrical signal carried, along the axon, by variation of the relative concentration of ions between the inside and the outside of the neuron (Fig. A-2).

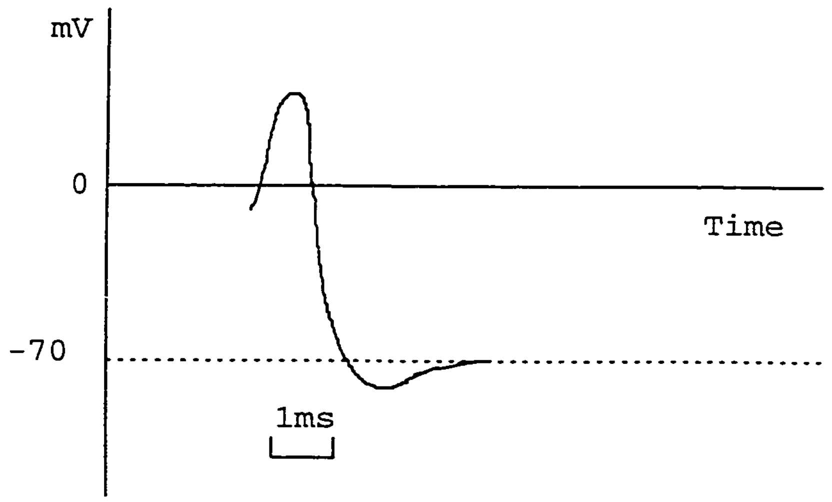


Figure A-I-2. Typical nerve impulse

The actual voltages are fixed by the biochemical processes of the cell membrane and not by the dimensions of the neurons or whatever physico-geometric parameter. The nerve impulse's velocity varies from a few cm/s to 2 m/s (Dowling 1992).

A nerve impulse reaches the end of its course at a synapse. The ion concentration variation at the synapses produces the release, in the gap between the cell membranes, of so-called neurotransmitters (Eccles 1956, Axelrod 1974, Dowling 1992) which affects the ionic permeability of the membrane of the post-synaptic neuron, modifying its potential.

These potential modifications (of the post-synaptic cell) can be excitatory (leading to an increase in voltage difference) or inhibitory. The nature of the inhibition can be twofold: it may be a decrease of potential, or it may block the increase of potential that would be induced by other synapses on the same cell. Nevertheless, inhibitory processes occur in biological systems; the former is used in the present research.

The actual importance of a synapse can (in principle) be measured by the quantity of neurotransmitter released at that synapse; I will call synaptic strength the quantity of neurotransmitter released; it will be a real number between -1 and +1, negative for inhibitions and positive for excitations. It is well established experimentally (Alkon 1983 and 1989), after being hypothesized by Hebb (Hebb 1949) that (i) memory lies in the synaptic strength and (ii) learning processes actually modify the synaptic strengths.

The neuron's essential function amounts to summing the inputs coming from the dendrites, and if the sum exceeds a certain threshold the neuron produces an impulse which travels along its axon and is further distributed to other neurons through the sysnapses. This is the process modeled in the equation given in figure A-1; the refractory period is not taken into account here, nor the time summation encountered in biology. Biological neurons sum their entries in space: the actual membrane potential difference depends on the sum of all the (active) synaptic connections of the neuron. They also sum their entries in time: the membrane potential at a given time t depends on what happened at the synapses of the cell in some time interval Δt ending at time t. In such a context the frequencies of the inputs do carry some information and the relevant signals need not appear at a particular time but only in the proper time interval. This time-frequency behaviour is another justification for the continuous response sigmoid function (Kohonen 1989).

I-2 Justification of the Sigmoid function

I-2-1 Biological origins

The output function of a neuron y_j can be approximated (Mc Cullogh & Pitts 1943) by a step function of the form:

$$\text{if } \sum_i C_{ij} x_i \geq T \Rightarrow y_j = 1$$

$$\text{if } \sum_i C_{ij} x_i < T \Rightarrow y_j = 0$$

where C_{ij} are the synaptic strengths, x_i are the outputs of the presynaptic neurons and T is a constant threshold.

Since this type of function is desired in the context of a Newton-Raphson, or gradient, approximation, a differentiable function is needed. The symmetric sigmoid function was chosen:

$$S(u) = \frac{1 - e^{-c(u-\theta)}}{1 + e^{-c(u-\theta)}} = \tanh\left(\frac{1}{2}c(u - \theta)\right)$$

where θ is the threshold and c, called the sigmoid constant, controls the slope of the sigmoid at $u = \theta$. One can get the "biological" step function by taking $c \to \infty$ and $\theta = T$

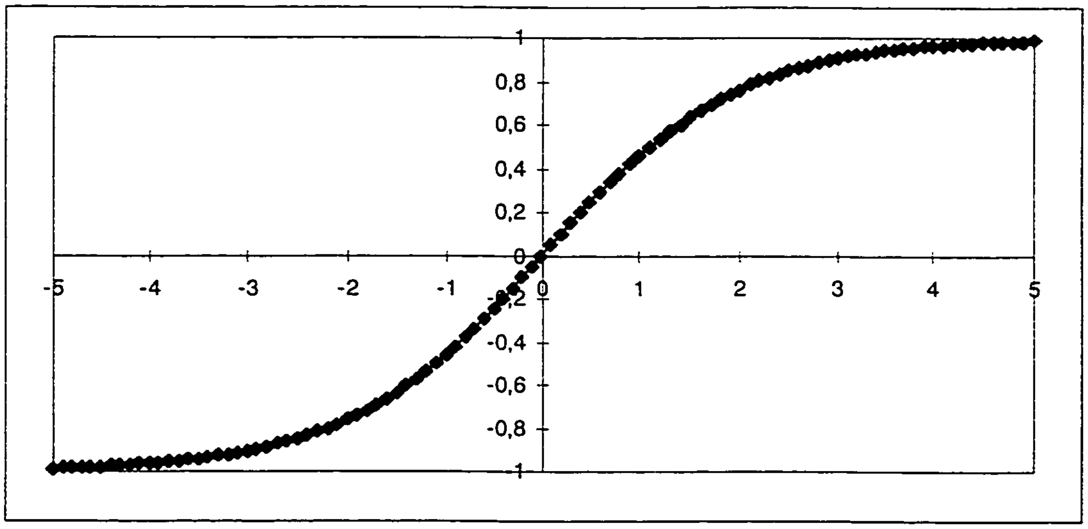


Figure A-I-3 The Sigmoid function for $c = 1$ **and** $\theta = 1$

In our model the threshold (θ) and the sigmoid constant (c) can vary from neuron to neuron and are adaptable in the learning process.

I-2-2 Theoretical remarks

- The time summation makes the neuron's response sensitive to pulse trains and makes it less drastic than the step function. This time-frequency behaviour is simulated through the use of the continuous reponse sigmoid function (Kohonen 1989)

- When the inputs of a neuron are very high, and when at the same time many synapses are simultaneously active, its output has to be compressed in such a way that the neurons following it remain operational, i.e. below saturation.

- Without such threshold and compressing functions the umbrella function would not have any extremum. The behaviour of the algorithm would be essentially divergent, the compressing functions actually bounding the process.

- There is also the theorem: "multilayer feedforward networks with nonpolynomial activation function can approximate any function" (Leshno et al 1993) which justify the use of the sigmoid.

I-3 Linear and nonlinear learners

The neuron state used in the present research,

$$y_j = \sum_{i=1}^{d_x} C_{ij} S_i(x_i) + \sum_{i=1}^{d_x} \sum_{i'=i}^{d_x} D_{ii'j}\, S_i(x_i)\, S_{i'}(x_{i'})$$

contains a linear dependence (the first term) and a nonlinear part. The nonlinear part involves biologically unrealistic synaptic strengths (D's) although they may simulate the so-called modulation processes.

APPENDIX II

A-II-1 A Deterministic partitioning device

Presented here is a very simple feedforward neural network together with a supervised learning algorithm to solve any partitioning problem. There are no random steps in the learning process. It nas the properties:

1) The learning time is a linear function of the size of the learning set.

2) The final architecture is independent of the order of presentation of learning set.

3) The learning process is robust: it can fail only through lack of sufficient memory.

4) The probability that a configuration taken from the learning set is properly classified is strictly 1.

5) Analysis of the generalization reveals the existence of a "generalization tendency mechanism".

Many problems in the field of neural networks can be reduced to the construction of a system architecture and a supervised learning algorithm for the following problem (see for example Valet 1989). Each element of a sampling (the learning set) of the configuration space of the input belongs to one and only one well-defined class. The proper class should be assigned, ideally, to: (i) all the configurations of the learning set ("exact memory"), and (ii) the largest possible number of configurations taken from outside the LS ("generalization"). Magnetic tape and the usual computer memory obey (i); biological memories tend to obey (i) and (ii). The device introduced in this work obeys both (i) and (ii). Backpropagation, by far the most widely-used learning algorithm for feedforward networks, is known to have convergence and local-minima problems (Rumelhart & Mc Clelland 1986). Simulated annealing (Kirkpatrick 1983, Watkin 1993) has been proposed as an escape procedure for the problem of local-minima, but it is time-consuming. Various other algorithms have been proposed to circumvent these two problems, see for instance: Grossman 1988, Baba 1989, Mézard et al 1989, Frean 1990); all use either modified backpropagation, or time-consuming stochastic steps, both of which have nonlinear learning times as a function of the cardinality of the learning set. The present algorithm does not use backpropagation or stochastic steps.

A-II-2 Architecture and algorithm

The architecture (see Fig A-II-1) consists of one layer of cells for the input, two layers of processing units, and a final, "output" or "classes", layer. The first layer has n_i "input" or "fanout" units; they do not process information, but serve as a link between the outside environment and the first processing layer.

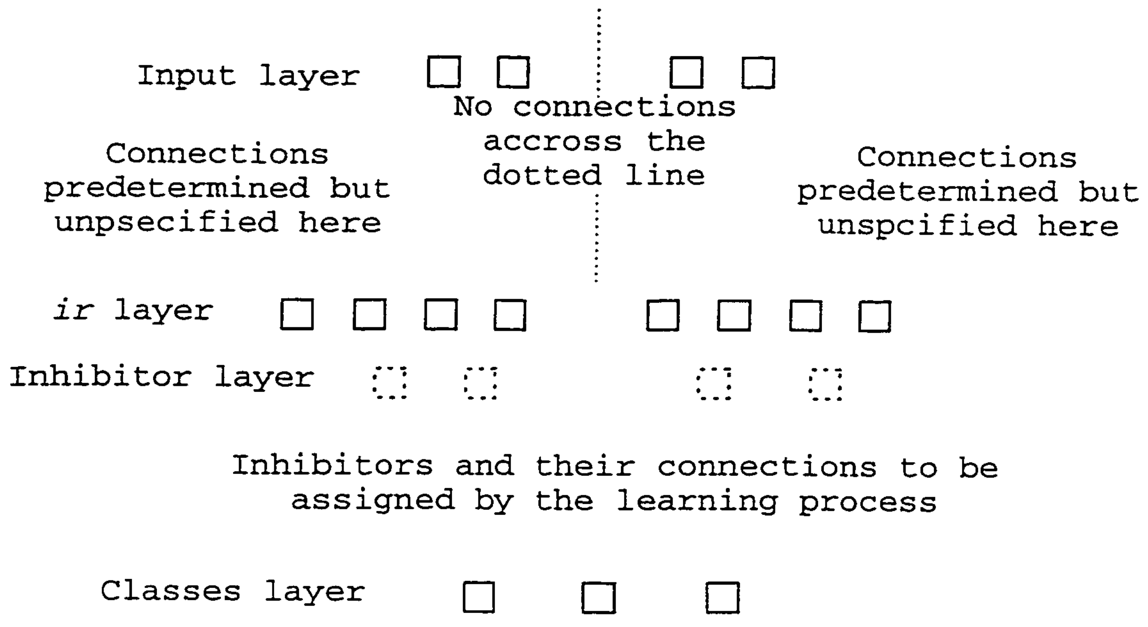

Input layer
No connections accross the dotted line
Connections predetermined but unpsecified here
Connections predetermined but unspcified here
ir layer
Inhibitor layer
Inhibitors and their connections to be assigned by the learning process
Classes layer

Figure A-II-1: Diagram of architecture for $n_g = 2$, $n_r = 8$, $n_p = 2$, $n_c = 3$.

The second, internal representation layer, is used to produce a one-to-one, normalized encoding of the inputs. Normalized here means that any input configuration will activate exactly the same number (n_g, defined below) of processing units out of the n_r processing units in the internal representation layer. The value of n_r depends on the particular encoding chosen. For a given encoding, the connections between the input layer and the internal representation layer are easily assigned (they are not learned), no further processing units being required.

The input cells were divided in some fashion into n_g groups of n_p cells ($n_i = n_p\, n_g$). Each group is encoded in 2^{n_p} dedicated processing units. The latter units have the usual structure and operation (Haykin 1994), with threshold $T = 1$. The total number of processing units in the internal representation layer is then $n_r = n_g\, 2^{n_p}$, and any input will then activate exactly n_g processing units in the internal representation layer. The last layer contains the n_c processing units, one per class of the partition. Between the internal representation layer and the output layer will lie an inhibitor layer of n_h processing units, introduced by the learning process. The learning algorithm assigns all the connections between i) internal representation and classes, ii) internal representation and inhibitors, and iii) inhibitors and classes, and is implemented as follows (Lafortune 1991).

Step 1: first scanning of the learning set: Take the next example from the LS; if the learning set has been exhausted go to Step 3; otherwise go to Step 2.

Step 2: Case 1: No class is activated. Connect all the activated internal representation units to the proper class (hebbian-type rule Hebb 1949) by connections of strength $1/n_g$ and go to Step 1. Case 2: The proper class and no other class is activated. Go to Step 1. Case 3: The proper class is not activated and some other classes are. (i) Assign connections to the proper class as in Case 1; and (ii) For each unwanted class introduce an inhibitor connected forward to that class with strength -1 and backward to each activated unit of the internal representation layer with strength $1/n_g$ and go to Step 1. Case 4: The proper class is activated together with some other classes. Introduce one inhibitor for each unwanted class as in Case 3 and go to Step 1.

Step 3: second scanning of the learning set: Take the configurations of the learning set one by one (in the same order as in Step 1) and verify the output states; two cases can occur for each member of the learning set: Case 1: No class is wrongly activated. Resume the scanning. Case 2: Some classes are wrongly activated. Add the appropriate inhibitors and resume scanning. The termination of this scanning ends the learning process. Two scannings are sufficient, for, consider (worst case) that configurations #a and #b induce wrong connections to each other's class. Then after two scannings, both inhibitors will be in place.

A-II-3. Theoretical remarks

From the description of the learning process the following points may be noted: (i) The linearity of the learning time as a function of $N_{(a)}$, the number of elements in the learning set, is due to

(i) the scannings of the learning set being direct (no scanning within the scanning) and

(ii) the connections being assigned only once, without modification, during the learning process. (ii) The learning set is perfectly classified.

(iii) the system can fail only through an insufficient number of available inhibitor units.

(iv) the resulting architecture is independent of the presentation order of the learning set. This is of interest especially when compared to Boltzmann-type machines (Rumelhart & McClelland 1986, Hecht-Nielsen 1989, Anderson 1989 and references therein) where not only is the final architecture highly sensitive to the initial order of presentation of the learning set, but also the connection values might well oscillate when the learning set is scanned repeatedly in the same order.

(v) The system has some generalization capability, which depends on the learning set and on n_g (see also section A-II-5).

(vi) After completion of the learning process, a configuration, R, chosen randomly from the configuration space can be placed into one of four categories:

(a) R belongs to the learning set and the proper class is activated;

(b) R does not belong to the learning set and no class is activated (a) "non-recognized configuration"(NR));

(c) R does not belong to the learning set and exactly one class is activated. This class may be (c_1) correct, (c_2) incorrect (c_1, and c_2 apply to the problems in Fig. A-II-2b where the class assignment rule is clear, and not to those in Fig. A-II-2a), or (c_3) undefined as to correctness (this will occur when the problem rules are undefined, as e.g. those considered in Fig. A-II-2a); and (d) R does not belong to the LS and more than one class is activated. This configuration thus belongs to the generalization of at least two classes; we call it a "confused configuration"(CONF). Because of (b) and (d) one might choose to add the "non-recognized" class and the "confused" class to the number n_c of original output classes. Categories b) and d) can be though of as errors of the network and/or of the learning process.

A-II-4. Measurements

A-II-4.1 Definitions

In computer simulations, to analyse the generalization in the system, the number of input configurations falling under the various categories (a) to (d) was measured. The problems studied have four inputs and one or two outputs. The one-output problem is treated by excluding from the learning set all configurations not belonging to the one class. The two-output problem is defined such that the number of configurations (out of any learning set) assigned to the two classes are equal. Within these restrictions all possible learning sets have been analysed.

The learning curves are plotted in Fig A-II-2 where we are interested in the probability p that a randomly-chosen configuration is properly classified (i.e. in categories (a) or (c_3) for Fig. A-II-2a with $p = \left\langle N_{(a)} + N_{(c_3)} \right\rangle / 16$, in an obvious notation, where $\langle \ldots \rangle$ represents the average over all the LS's; and in categories (a) or (c_1) for Fig. A-II-2b with $p = \left\langle N_{(a)} + N_{(c_1)} \right\rangle / 16$).

A-II-4.2 Scanned problems

We presented all (subject to the restrictions in the above paragraph) possible learning sets of 1, 2, ..., 16 configurations to our algorithm to learn. They number $2^{16} - 1 = 65535$ for one-output problems and 2598313 for two-output problems. The particularity of the one-output problems is that confusion cannot exist; and since there is only one output class, the learning algorithm will generate no inhibitor. The results are given in Fig A-II-2a.

A-II-4.3 Parity problems

To further investigate the quality of the generalization, I measured generalization for two well-defined problems, namely (i) the (linearly separable) simple parity problem and (ii) the bit-sum parity problem (non-linearly separable). The latter deals with the parity of the sum of the bits of the input binary number; all the input bits, rather than just the last, are significant, making this harder to learn (but also more realistic) than the simple parity problem. The results are given in Fig A-II-2b, for the one-class problem for simple parity only, and for the two-class problem for both.

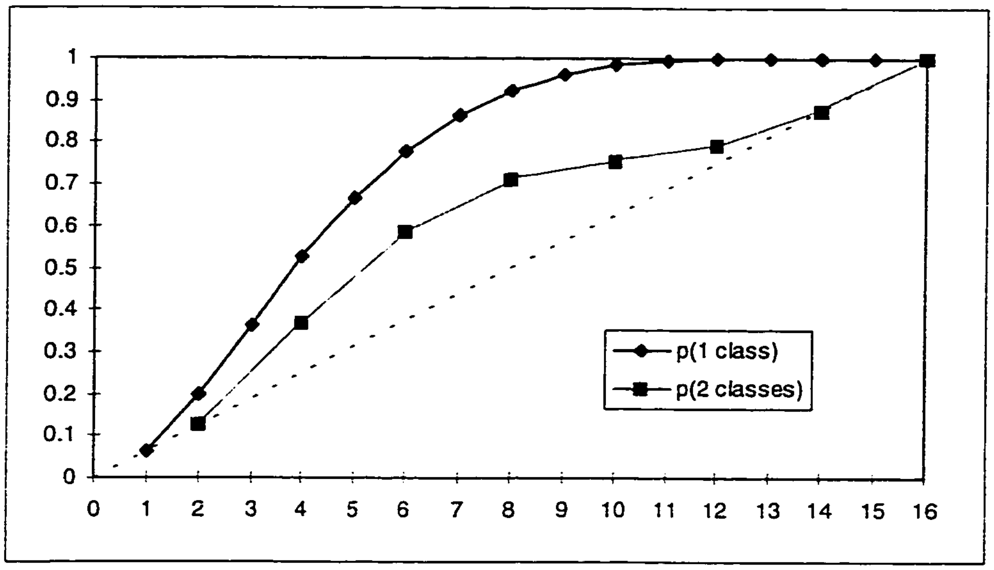


Figure A-II-2a: Learning curves for all the problems (averaged over all the possible learning sets, every generalization counted as correct). The diagonal is the exact memory performance.

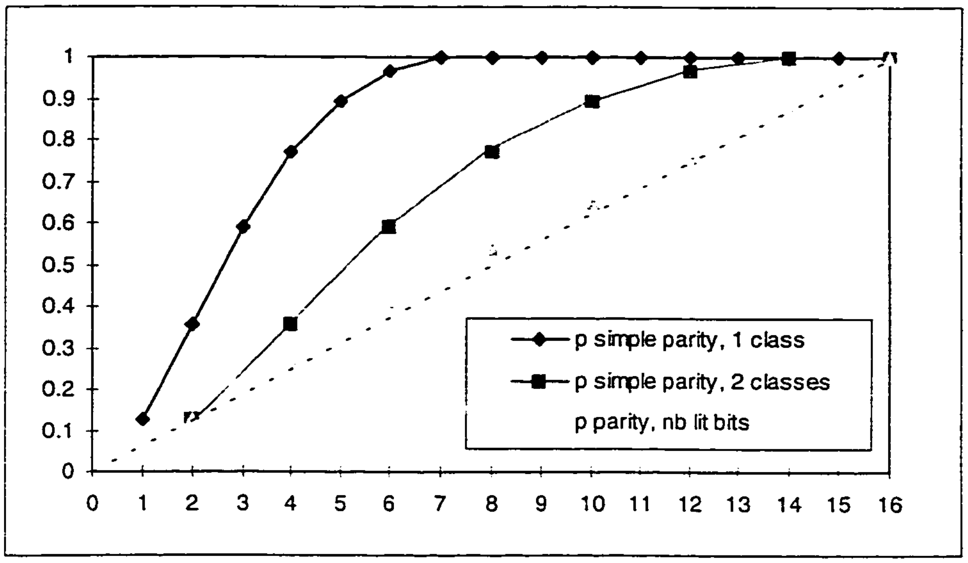


Figure A-II-2b: Learning curves for the parity problems (the diagonal is the exact memory performance).

A-II-4.4 Random Boolean functions

We have also tested our system for 9-input, 3-output random Boolean functions. The learning sets were constructed on the following principles: the number of configurations in the learning set was a random number between 6 and 100; the inputs were randomly assigned the values 0 or 1; the output class (1, 2 or 3) was randomly assigned to the input configuration; care was taken that an input configuration occurred only once in the LS and consequently occurred in only one class. The input cells were divided into 3 groups of 3 and the internal representation had three groups of 8 cells. When the learning process was terminated, the network was exposed to the $2^9 = 512$ input configurations and its ouputs were counted in the four categories. Figure A-II-3a gives CONF as a function of LS. Figure A-II-3b gives NR as a function of LS. Besides classification of inputs we also counted the numbers of neurons (number of inhibitors + 24 (internal representation) + 3 (number of outputs)) in order to evaluate the storage ratio:

$$SR = \frac{\text{LS + generalization category c}}{\text{number of neurons}}$$

Figure A-II-3c gives the generalization as a function of the storage ratio.

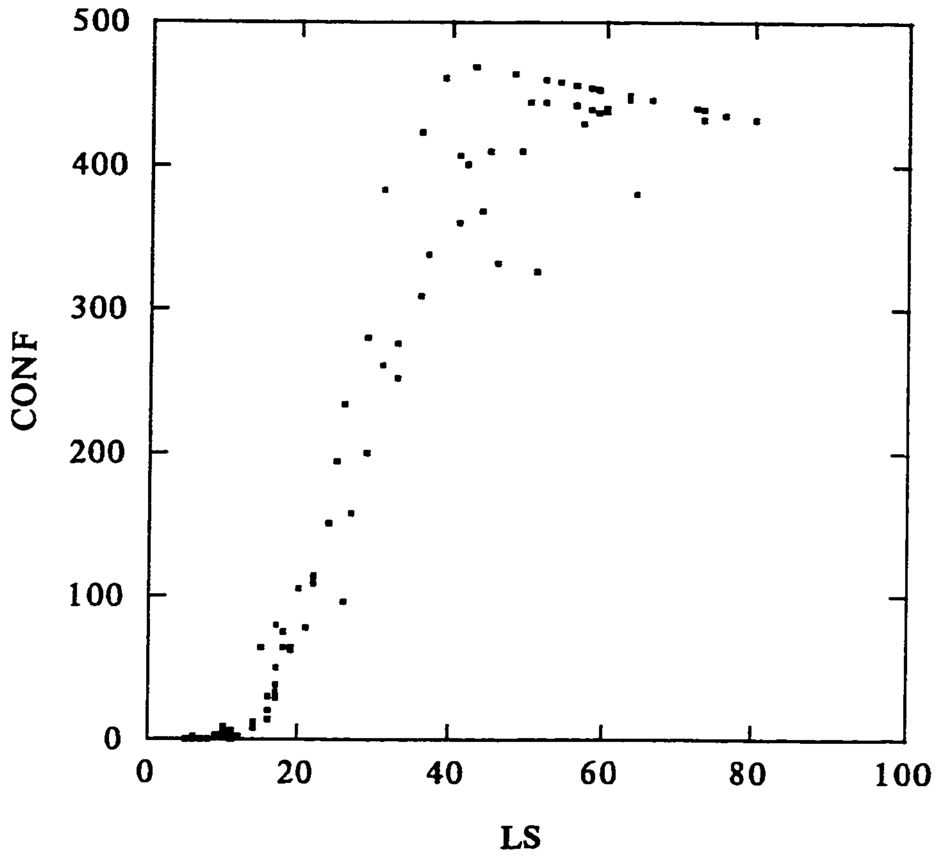


Figure A-II-3a: Random Boolean function learning measures. Number of confused configurations versus learning set cardinality. After the maximum is reached, there is an absolute straight-line limit given by CONF = 512 – LS**; for any case on this line ALL the configurations outside the learning set produce confusion at the output level.**

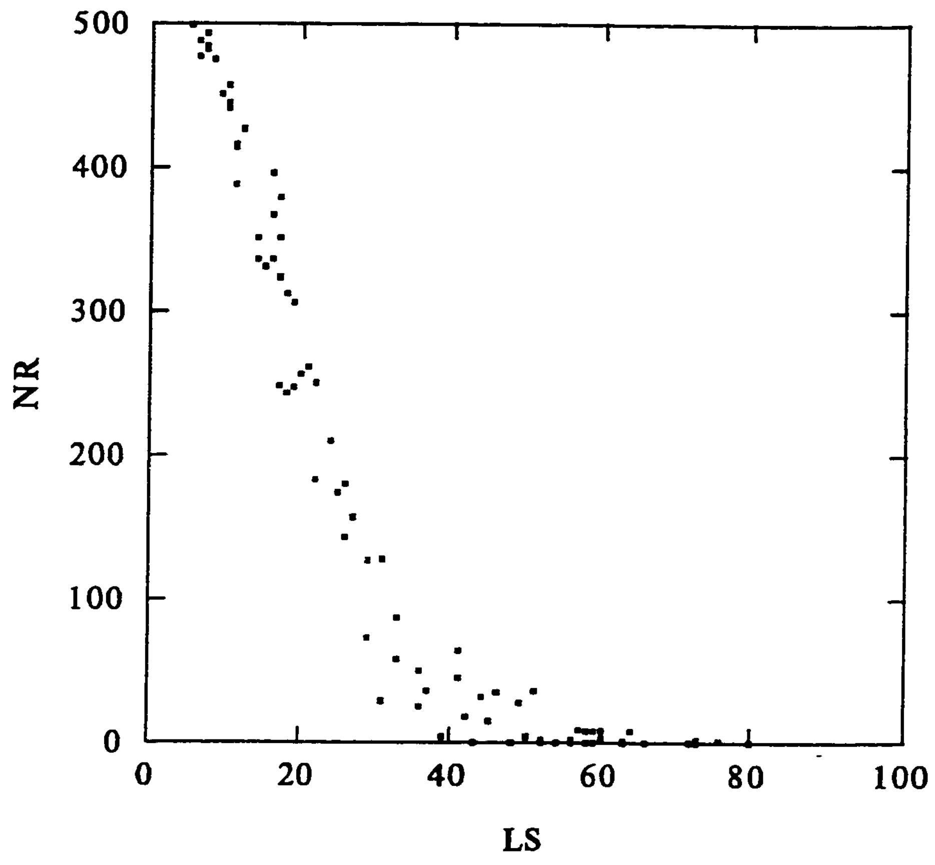


Figure A-II-3b: Random Boolean function learning measures. Number of non-recognized configurations versus learning set cardinality. It is natural that the function decreases with increasing LS; but a caveat: if a configuration is NOT non-recognized it could be in the LS itself, or in the generalization, or (worse) in confusion.

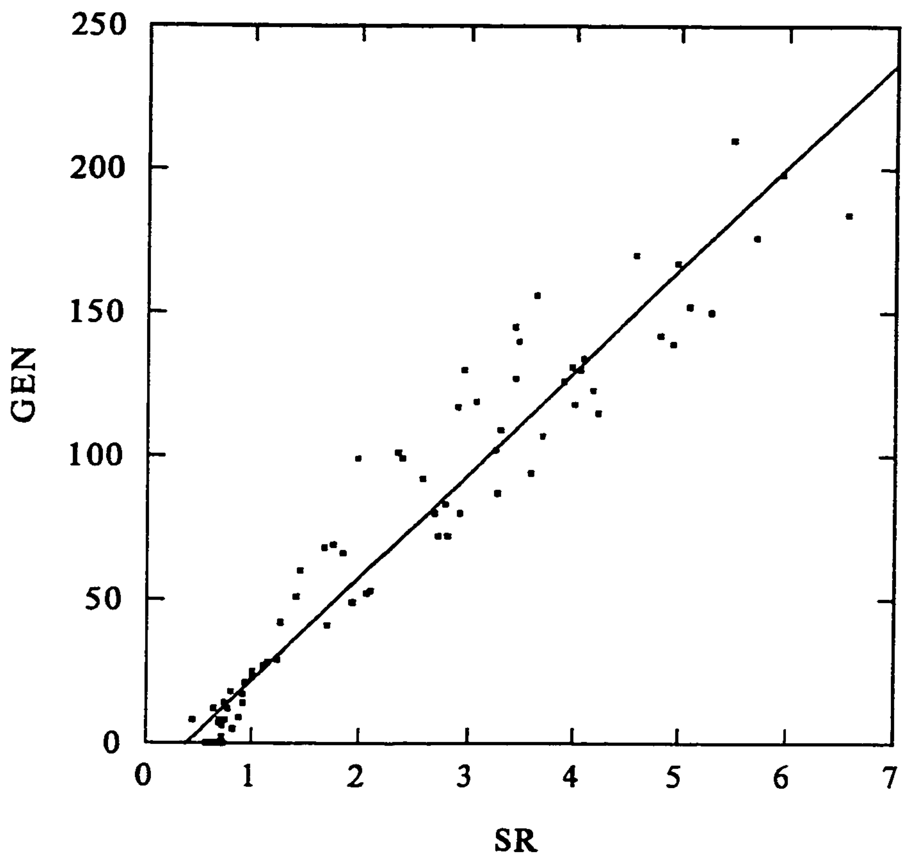


Figure A-II-3c: Random Boolean function learning measures. Generalization versus storage ratio. The linear equation is $GEN = -13.7 + 35.8\,SR$ **. The linearity may be due to the small range of storage ratios encountered in the simulation. The current idea that generalization should increase with storage ratio is clearly substantiated here.**

A-II-5. Results and discussion

From Figs. A-II-2a and A-II-2b our system clearly generalizes, as the curves lie above the exact memory curve. The average generalization is seen to be far better for the one-class problem than for the two-class problem. For the simple parity problem (Fig A-II-2b) the results are comparable to those cited in Anderson 1989, Wasserman 1989, Kosko 1992. It is not surprising to find generalization for this linearly separable case; it is of interest, however, to note that our system does lead to generalization, however reduced, for the non-linearly-separable bit-sum problem. It is clear from Figs. A-II-2a and A-II-2b that two-output problems are more difficult to learn than one-output problems. This is because (i) no confusion at all can arise in the one-class problem, and (ii) there is lower generalization in the two-output cases as can be seen from the smaller number of multiple connections (different cells in the same group connected to the same class) possible in the two-class problem, for a given number of configurations in the LS. Our choice of the number of cells per input group, $n_p = n_i / n_g = 2$ was motivated by computer-time considerations. Higher values of n_p could easily be handled by our method; however, time constraints would then not have allowed for an exhaustive analysis of the "scanned problems". The question arises as to how generalization would be affected by a different choice of n_g . E.g., if $n_p = n_i$, there is no generalization possible in our scheme. If $n_g = 1$ and $n_p = n_i$, generalization is impossible; and if $n_p = 1$ the generalization will be different again. The generalization occurring in our particular architecture arises in a very specific manner, which can be thought of as an implicit generalization-tendency (IGT) mechanism. For example, say that cells a and b from group 1, and cells e and f from group 2 (for a two-group ir), are connected to class C_1 (see Fig. A-II-4).

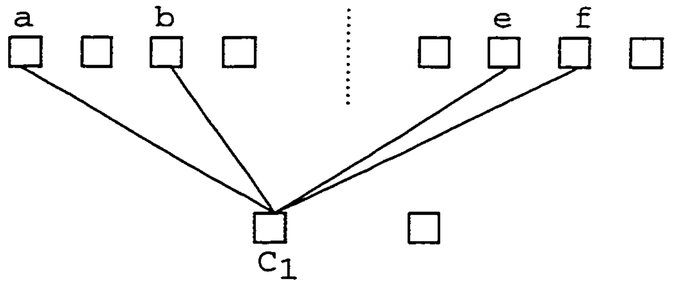


Figure A-II-4: Generalization tendency.

The configurations leading to "ae" and "bf" (though not "af" and "be") having been selected as part of the LS and assigned to C_1 . Then C_1 will also result from the "af" and "be" inputs; the configurations leading to "af" and "be" are said to belong to the generalization of C_1 . If, instead, "be" (say) were indeed part of the LS and assigned to a different class, the learning process (Step 2, Case 3) would introduce an inhibitor to remove "be" from C_1 's generalization tendency. The inhibitor layer may thus be thought of as a neutralizer of the generalization tendency in such cases. The results of the simple parity two-class problem are quite illuminating in this regard: none of the learning sets required inhibitors, implying that the IGT mechanism in this case conformed highly to the parity definition. By contrast, the poor generalization (Fig. A-II-2b) obtained for the bit-sum parity problem reveals that for it, the IGT mechanism was not compatible with the bit-sum parity definition; here, only the trivial LS's did not require inhibitors. An improved algorithm would yield higher generalization for both parity problems if one could arrange for the IGT rule to depend on the learning set itself. This is, explicitly, the problem addressed in Grossman et al. 1988 where the number of configurations stored per processing unit was roughly 9 (for 1-output problems); ours is 4.15, but for 3-output problems.

APPENDIX III

A Theorem about the sigmoid

Discussed here is the following question: is the value $1-\varepsilon$, for the sigmoid function, reachable in a finite number of steps if the steps are proportional to the derivative of the sigmoid function. The sigmoid function and its derivative are given respectively by

$$S(x) = \frac{1 - e^{-cx}}{1 + e^{-cx}} \quad ; \qquad \frac{\partial S(x)}{\partial x} = \frac{2ce^{-cx}}{\left[1 + e^{-cx}\right]^2} \qquad c > 0$$

Theorem I

A process, starting at $x=0$, which is incremented proportionally to the derivative of the sigmoid, will reach the point x_T where $S(x_T)=1-\varepsilon$ in a finite number of steps.

Proof:

$$\text{Let } x_T \text{ such that } S(x_T) = \frac{1 - e^{-cx_T}}{1 + e^{-cx_T}} = 1 - \varepsilon \qquad (\varepsilon < 1)$$

$$\text{then } x_T = \frac{-1}{c} \ln\left(\frac{\varepsilon}{2 - \varepsilon}\right)$$

$$\text{so that: } e^{-cx_T} = \frac{\varepsilon}{2 - \varepsilon}$$

$$\text{and } \left.\frac{\partial S(x)}{\partial x}\right|_{x_T} = \frac{2c\varepsilon\,(2 - \varepsilon)}{4}$$

And this is the smallest value of the sigmoid derivative in the process. If we now take $\Delta x = \alpha \frac{2c\varepsilon(2-\varepsilon)}{4}$ as the minimal step in the process, it will take less than $\frac{x_T}{\Delta x} + 1$ steps to reach the value x_T; so the required number of steps (M) is the smallest integer larger than ($\lceil\ \rceil$):

$$M = \left\lceil \frac{\frac{-1}{c} \ln\left(\frac{\varepsilon}{2-\varepsilon}\right)}{\alpha \frac{2c\varepsilon(2-\varepsilon)}{4}} \right\rceil$$

Theorem II

If the process starts at a value of $x \neq 0$, it will also take a finite number of steps for the sigmoid function to reach the value 1-ε.

Proof:

1) If the initial value of x is positive then the number of steps is less or equal to M of Theorem I.

2) If the initial value of x is negative, by the same argument as in the proof of Theorem I, it will take some M′ steps for x to reach the value zero. Adding the further M steps for the sigmoid to reach 1-ε, the finite number M + M′ of steps is required.

Theorem III

If the sigmoid constant is modified, while x remains fixed, by an amount proportional to the derivative of the sigmoid function with respect to the varying constant, it will take a finite number of steps for the sigmoid to reach the value $1-\varepsilon$.

Proof:

If it is the sigmoid constant that is modified (rather than x), the results of Theorems I and II are not affected; it is just a matter of renaming the variable and the constant.

Theorem IV

If the threshold of a sigmoid constant is modified proportionally to the derivative of the sigmoid, with respect to the threshold, the value $1-\varepsilon$ will be reached in a finite number of steps.

The proof this Theorem is achieved by to setting $c=1$, $-x$ being the threshold, in the proof of Theorem I, and starting anywhere as in Theorem II.

Theorem V

The Theorems I to IV remain valid under the changes:
$c \to -c$ and $1-\varepsilon \to -1+\varepsilon$.

APPENDIX IV

The derivatives

Presented here are some calulations used in the theory. First the derivative of the sigmoid function:

$$\frac{\partial S(u)}{\partial u} = \frac{\partial}{\partial u}\left[\frac{1 - e^{-c(u-\theta)}}{1 + e^{-c(u-\theta)}}\right] = \frac{2ce^{-c(u-\theta)}}{\left[1 + e^{-c(u-\theta)}\right]^2}$$

and in particular its value at $u = \theta$:

$$\left.\frac{\partial S(u)}{\partial u}\right|_{u=\theta} = \frac{2c}{[1 + 1]^2} = \frac{c}{2}$$

The sigmoid constant sets the slope of the sigmoid in the sensitive (or transition) region.

$$\frac{\partial S}{\partial c} = \frac{\partial}{\partial c}\left[\frac{1 - e^{-c(u-\theta)}}{1 + e^{-c(u-\theta)}}\right] = \frac{2(u - \theta)e^{-c(u-\theta)}}{\left[1 + e^{-c(u-\theta)}\right]^2}$$

$$\frac{\partial S}{\partial \theta} = \frac{\partial}{\partial \theta}\left[\frac{1 - e^{-c(u-\theta)}}{1 + e^{-c(u-\theta)}}\right] = \frac{-2ce^{-c(u-\theta)}}{\left[1 + e^{-c(u-\theta)}\right]^2}$$

For the similarity measures and the linear coupling constants:

$$\frac{\partial R_Y^{\alpha\alpha'}}{\partial C_{ij}} = \frac{1}{d_Y}\sum_{j'=1}^{d_Y}\frac{\partial}{\partial C_{ij}}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_{j'}(y_{j'}^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_{j'}(y_{j'}^{\kappa'})\right]$$

$$= \frac{1}{d_Y}\sum_{j'=1}^{d_Y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial}{\partial C_{ij}} S_{j'}(y_{j'}^\kappa)\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_{j'}(y_{j'}^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_{j'}(y_{j'}^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial}{\partial C_{ij}} S_{j'}(y_{j'}^{\kappa'})\right)\right]$$

Since, assuming that the couplings are independent:

$$\frac{\partial S_{j'}(y_{j'}^\kappa)}{\partial C_{ij}} = \frac{\partial}{\partial C_{ij}} S_{j'}\left(\sum_{\iota=1}^{d_X}\left[C_{\iota j'}\, S_\iota(x_\iota^\kappa) + \sum_{\iota'=\iota}^{d_X} D_{\iota\iota' j'}\, S_\iota(x_\iota^\kappa)\, S_{\iota'}(x_{\iota'}^\kappa)\right]\right)$$

$$= \delta_{jj'}\sum_{\iota=1}^{d_X}\delta_{i\iota}\left.\frac{\partial S_{j'}(u)}{\partial u}\right|_{u=y_{j'}^\kappa} S_\iota(x_\iota^\kappa) = \left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^\kappa} S_i(x_i^\kappa)$$

where $\delta_{jj'}$ and $\delta_{i\iota}$ are the usual Kronecker delta, therefore:

$$\frac{\partial R_Y^{\alpha\alpha'}}{\partial C_{ij}} = \frac{1}{d_Y}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^\kappa} S_i(x_i^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_j(y_j^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_j(y_j^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^{\kappa'}} S_i(x_i^{\kappa'})\right]$$

and, for two classes named zero and one, the derivative of the umbrella function is:

$$\frac{\partial U_Y}{\partial C_{ij}} = \beta\left(\frac{\partial R_Y^{00}}{\partial C_{ij}} + \frac{\partial R_Y^{11}}{\partial C_{ij}}\right) + \gamma\,\frac{\partial R_Y^{01}}{\partial C_{ij}}$$

The same calculation, with the replacements: $i\to j$, $j\to k$, $x\to y$ and $y\to z$ provides the derivative of the U_Z function with respect to the linear couplings from the first internal representation layer (y) to the second (z):

$$\frac{\partial U_z}{\partial c_{jk}} = \beta\left(\frac{\partial R_z^{00}}{\partial c_{jk}} + \frac{\partial R_z^{11}}{\partial c_{jk}}\right) + \gamma\,\frac{\partial R_z^{01}}{\partial c_{jk}}$$

For the similarity measures and the sigmoid constants:

$$\frac{\partial R_y^{\alpha\alpha'}}{\partial c_j} = \frac{1}{d_y}\sum_{j'=1}^{d_y}\frac{\partial}{\partial c_j}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_{j'}(y_{j'}^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_{j'}(y_{j'}^{\kappa'})\right]$$

$$= \frac{1}{d_y}\sum_{j'=1}^{d_y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial}{\partial c_j}s_{j'}(y_{j'}^\kappa)\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_{j'}(y_{j'}^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_{j'}(y_{j'}^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial}{\partial c_j}s_{j'}(y_{j'}^{\kappa'})\right)\right]$$

$$\frac{\partial}{\partial c_j}s_{j'}(y_{j'}^\kappa) = \frac{2\,(y_{j'}^\kappa - \theta_j)\,e^{-c_j(y_{j'}^\kappa-\theta_j)}}{\left[1 + e^{-c_j(y_{j'}^\kappa-\theta_j)}\right]^2}\,\delta_{jj'} = \frac{\partial s_j(y_j^\kappa)}{\partial c_j}\,\delta_{jj'}$$

$$\frac{\partial R_y^{\alpha\alpha'}}{\partial c_j} = \frac{1}{d_y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial s_j(y_j^\kappa)}{\partial c_j}\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_j(y_j^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_j(y_j^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial s_j(y_j^{\kappa'})}{\partial c_j}\right)\right]$$

And for the umbrella function:

$$\frac{\partial U_y}{\partial c_j} = \beta\left(\frac{\partial R_y^{00}}{\partial c_j} + \frac{\partial R_y^{11}}{\partial c_j}\right) + \gamma\,\frac{\partial R_y^{01}}{\partial c_j}$$

the expressions are the same for the other layers with y→x and j→i for the input layer; and y→z and j→k for the second internal representation layer.

For the similarity measures and the thresholds:

$$\frac{\partial R_y^{\alpha\alpha'}}{\partial\theta_j} = \frac{1}{d_y}\sum_{j'=1}^{d_y}\frac{\partial}{\partial\theta_j}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_{j'}(y_{j'}^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_{j'}(y_{j'}^{\kappa'})\right]$$

$$= \frac{1}{d_y}\sum_{j'=1}^{d_y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial}{\partial\theta_j}s_{j'}(y_{j'}^\kappa)\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_{j'}(y_{j'}^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_{j'}(y_{j'}^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial}{\partial\theta_j}s_{j'}(y_{j'}^{\kappa'})\right)\right]$$

$$\frac{\partial}{\partial\theta_j}s_{j'}(y_{j'}^\kappa) = \frac{-2c_{j'}e^{-c_{j'}(y_{j'}^\kappa-\theta_{j'})}}{\left[1+e^{-c_{j'}(y_{j'}^\kappa-\theta_{j'})}\right]^2}\delta_{jj'} = \frac{\partial s_j(y_j^\kappa)}{\partial\theta_j}\delta_{jj'}$$

$$\frac{\partial R_y^{\alpha\alpha'}}{\partial\theta_j} = \frac{1}{d_y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial s_j(y_j^\kappa)}{\partial\theta_j}\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} s_j(y_j^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} s_j(y_j^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial s_j(y_j^{\kappa'})}{\partial\theta_j}\right)\right]$$

and for the umbrella function, as usual:

$$\frac{\partial U_y}{\partial\theta_j} = \beta\left(\frac{\partial R_y^{00}}{\partial\theta_j}+\frac{\partial R_y^{11}}{\partial\theta_j}\right)+\gamma\frac{\partial R_y^{01}}{\partial\theta_j}$$

For the non linear couplings:

$$\frac{\partial R_Y^{\alpha\alpha'}}{\partial D_{ii'j}} = \frac{1}{d_Y}\sum_{j'=1}^{d_Y}\frac{\partial}{\partial D_{ii'j}}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_{j'}(y_{j'}^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_{j'}(y_{j'}^{\kappa'})\right]$$

$$= \frac{1}{d_Y}\sum_{j'=1}^{d_Y}\left[\left(\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\frac{\partial}{\partial D_{ii'j}} S_{j'}(y_{j'}^\kappa)\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_{j'}(y_{j'}^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_{j'}(y_{j'}^\kappa)\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial}{\partial D_{ii'j}} S_{j'}(y_{j'}^{\kappa'})\right)\right]$$

Assuming the independence of the couplings again:

$$\frac{\partial S_{j'}(y_{j'}^\kappa)}{\partial D_{ii'j}} = \frac{\partial}{\partial D_{ii'j}} S_{j'}\left(\sum_{l=1}^{d_X}\left[C_{lj'}\,S_l(x_l^\kappa) + \sum_{l'=l}^{d_X} D_{ll'j'}\,S_l(x_l^\kappa)\,S_{l'}(x_{l'}^\kappa)\right]\right)$$

$$= \delta_{jj'}\,\delta_{il}\,\delta_{i'l'}\,\left.\frac{\partial S_{j'}(u)}{\partial u}\right|_{u=y_{j'}^\kappa} S_l(x_l^\kappa)\,S_{l'}(x_{l'}^\kappa)$$

$$\frac{\partial R_Y^{\alpha\alpha'}}{\partial D_{ii'j}} = \frac{1}{d_Y}\left[\frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^\kappa} S_i(x_i^\kappa)\,S_{i'}(x_{i'}^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_j(y_j^{\kappa'}) + \frac{1}{N^\alpha}\sum_{\kappa\in\Gamma^\alpha} S_j(y_j^\kappa)\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^{\kappa'}} S_i(x_i^{\kappa'})\,S_{i'}(x_{i'}^{\kappa'})\right]$$

and of course:

$$\frac{\partial U_Y}{\partial D_{ii'j}} = \beta\left(\frac{\partial R_Y^{00}}{\partial D_{ii'j}} + \frac{\partial R_Y^{11}}{\partial D_{ii'j}}\right) + \gamma\,\frac{\partial R_Y^{01}}{\partial D_{ii'j}}$$

The above is for what is called "one-layer learning". The U_Z function depends on the sigmoid constants and on the thresholds of the x and y layers, as well as on the linear and nonlinear couplings; needed for the "two-layer learning" were the quantities $\frac{\partial U_z}{\partial C_{ij}}$ and $\frac{\partial U_z}{\partial D_{ii'j}}$, of which the first only was used.

$$\frac{\partial R_z^{\alpha\alpha'}}{\partial C_{ij}} = \frac{1}{d_z}\sum_{m=1}^{d_z}\frac{\partial}{\partial C_{ij}}\left[\frac{1}{N^{\alpha}}\sum_{\kappa\in\Gamma^{\alpha}} S_m(z_m^{\kappa})\,\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_m(z_m^{\kappa'})\right]$$

$$= \frac{1}{d_z}\sum_{m=1}^{d_z}\left[\left(\frac{1}{N^{\alpha}}\sum_{\kappa\in\Gamma^{\alpha}}\frac{\partial}{\partial C_{ij}} S_m(z_m^{\kappa})\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_m(z_m^{\kappa'}) + \frac{1}{N^{\alpha}}\sum_{\kappa\in\Gamma^{\alpha}} S_m(z_m^{\kappa})\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}}\frac{\partial}{\partial C_{ij}} S_m(z_m^{\kappa'})\right)\right]$$

$$\frac{\partial}{\partial C_{ij}} S_m(z_m^{\kappa}) = \left.\frac{\partial S_m(u)}{\partial u}\right|_{u=z_m^{\kappa}} \frac{\partial z_m^{\kappa}}{\partial C_{ij}}$$

For linear couplings only:

$$\frac{\partial z_m^{\kappa}}{\partial C_{ij}} = \frac{\partial}{\partial C_{ij}}\sum_{j'=1}^{d_y} C_{j'm}\, S_{j'}(y_{j'}^{\kappa}) = \sum_{j'=1}^{d_y} C_{j'm}\,\frac{\partial S_{j'}(y_{j'}^{\kappa})}{\partial C_{ij}}$$

and since the last derivative has already been obtained:

$$\frac{\partial S_{j'}(y_{j'}^{\kappa})}{\partial C_{ij}} = \delta_{jj'}\left.\frac{\partial S_{j'}(u)}{\partial u}\right|_{u=y_{j'}^{\kappa}} S_i(x_i^{\kappa}) \quad ,$$

we have $\frac{\partial z_m^{\kappa}}{\partial C_{ij}} = C_{jm}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^{\kappa}} S_i(x_i^{\kappa})$

$$\frac{\partial R_z^{\alpha\alpha'}}{\partial c_{ij}} =$$

$$\frac{1}{d_z}\sum_{m=1}^{d_z}\left[\left(\frac{1}{N^{\alpha}}\sum_{\kappa\in\Gamma^{\alpha}} c_{jm}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^{\kappa}} S_i(x_i^{\kappa})\right)\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} S_m(z_m^{\kappa'}) + \frac{1}{N^{\alpha}}\sum_{\kappa\in\Gamma^{\alpha}} S_m(z_m^{\kappa})\left(\frac{1}{N^{\alpha'}}\sum_{\kappa'\in\Gamma^{\alpha'}} c_{jm}\left.\frac{\partial S_j(u)}{\partial u}\right|_{u=y_j^{\kappa'}} S_i(x_i^{\kappa'})\right)\right]$$

and for the U-function:

$$\frac{\partial U_z}{\partial c_{ij}} = \beta\left(\frac{\partial R_z^{00}}{\partial c_{ij}} + \frac{\partial R_z^{11}}{\partial c_{ij}}\right) + \gamma\,\frac{\partial R_z^{01}}{\partial c_{ij}}$$

to allow for the corrections of the type:

$$\Delta c_{ij} = g\,\frac{\partial U_y}{\partial c_{ij}} + f\,\frac{\partial U_z}{\partial c_{ij}}$$

as discussed in the theory section II-2.

APPENDIX V

Linear separability

Let $\left\{\mathbf{x}^{\kappa}\right\}$ and $\left\{\mathbf{x}^{\lambda}\right\}$ be two sets of class vectors. The question is to decide if the two sets can be linearly separated or not. The linear separability condition is that there exists a hyperplane such that all the points of one class lie on one side of it and the points of the other class lie on the other side. This can be written: $\exists\, \mathbf{a} \mid \mathbf{a} \cdot (\mathbf{x}^{\kappa} - \mathbf{x}^{\lambda}) > 0 \;\forall\, \kappa, \lambda$; i.e. all the difference vectors, from one class to the other must have a component of the same sign along a vector ($\mathbf{a}$) normal to the separating hyperplane. If the sign of the component is negative one can always replace $\mathbf{a}$ by $-\mathbf{a}$. That is because if all the points of one class are on one side of the hyperplane, all the vectors starting at any λ point and terminating at κ point will have to cross the hyperplane at some point and consequently have a nonzero component along the normal to the hyperplane. All the difference vectors having a component in the normal direction of the same sign then means that all the points of one class are on one side of the hyperplane and the points of the other class are on the other side.

The actual linear separation process is then to find a vector **a**. One algorithmic approach is the following:

(1) Start with the average difference vector, say **b** as a first approximation to **a**.

(2) Either **b** fulfills the condition, in which case a solution has been found ,or, some difference vectors have a negative component along **b.** If so,

(3) Set

$$\mathbf{b} \rightarrow \mathbf{b}_{\text{new}} = \mathbf{b} + \Delta\mathbf{b} \text{ where } \Delta\mathbf{b} = \alpha \sum_{(\mathbf{x}^{\kappa}-\mathbf{x}^{\lambda}).\mathbf{b}<0} (\mathbf{x}^{\kappa} - \mathbf{x}^{\lambda}),$$

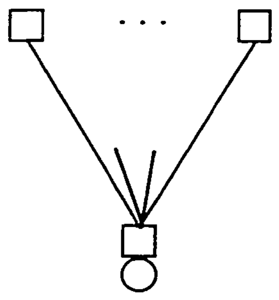
where α is some appropriate weighting factor (or learning rate in the neural network context); and then go to (2), using $\mathbf{b}_{\text{new}}$.

This is the essence of the perceptron learning algorithm and **b** converges to $\mathbf{a} + \vec{\varepsilon}$, $|\vec{\varepsilon}| << |\mathbf{a}|$ in a finite number of steps (see Appendix VI).

APPENDIX VI

The perceptron convergence theorem

For the sake of completeness the proof of the perceptron convergence theorem is presented here; it is a rewriting of Cotrell's proof as given in Blayo and Verlayen (1996).



x_i inputs

C_i couplings

Output state $\omega = \sum_{i=1}^{n} C_i x_i$

Output function: $\sigma(\omega) = \mathrm{sign}(\omega)$

Figure A-VI-1 Perceptron architecture

The perceptron learning rule is:

$$C_i\Big|_{p+1} = C_i\Big|_p + \alpha\left[\omega^*\Big|_p - \sigma\left(\sum_{i=1}^{n} C_i\Big|_p x_i\right)\right] x_i$$

where $\alpha>0$ is a learning rate, p indicates the pass. The perceptron algorithm updates the architecture variables at every configuration, not on the averages as in the umbrella algorithm; $\omega^*\Big|_p$ is the desired output for the current (p) configuration.

If $\omega^*\big|_p$ and $\sum_{i=1}^{n} C_i\big|_p x_i$ carry the same sign, there is no coupling correction (the actual output being the desired one). If they are of opposite sign (the actual output is erroneous):

$$\omega^*\big|_p > 0 \quad \text{and} \quad \sum_{i=1}^{n} C_i\big|_p x_i < 0 \text{ , } C_i \text{ is increased}$$

$$\omega^*\big|_p < 0 \quad \text{and} \quad \sum_{i=1}^{n} C_i\big|_p x_i > 0 \text{ , } C_i \text{ is decreased}$$

and the learning rule can be written:

$C_i\big|_{p+1} = C_i\big|_p$ when the actual output and the desired output carry the same sign; and,

$C_i\big|_{p+1} = C_i\big|_p \pm \alpha x_i$ when the actual output and the desired output do not carry the same sign.

The **theorem** is: If the two classes are linearly separable and if the input vectors are bounded, the perceptron learning algorithm will converge to a solution in a finite number of steps.

Proof: One can assume that the vectors $\left\{x^{\kappa}\right\}$, $\left\{x^{\lambda}\right\}$ to be classified are normalized; one can also replace all vectors (say x^{λ}) that must produce a negative output by $-x^{\lambda}$, such that if $\vec{C}^*$ is a solution: $\vec{C}^* \cdot \vec{x} > 0$, $\forall \vec{x} \in \left\{x^{\kappa}\right\}, \left\{x^{\lambda}\right\}$.

The strong linear separability hypothesis is now that there exist δ and $\vec{c}^*$ (that can be assumed to be normalized as well) such that $\vec{c}^* \cdot \vec{x} > \delta$, $\forall \vec{x}$.

$$\text{Define: } \beta\left(\vec{c}|_p\right) \equiv \frac{\vec{c}^* \cdot \vec{c}|_p}{\left|\vec{c}|_p\right|} = \cos\left(\vec{c}^*, \vec{c}|_p\right) \le 1 .$$

When the perceptron misclassifies an example $\vec{\chi}$ the numerator will be changed to

$$\vec{c}^* \cdot \vec{c}|_{p+1} = \vec{c}^* \cdot \left\{\vec{c}|_p + \alpha\left[\omega^*|_p - \sigma\left(\vec{c}|_p \cdot \vec{\chi}\right)\right] \vec{\chi}\right\}$$
$$= \vec{c}^* \cdot \vec{c}|_p + \left\{\alpha\vec{c}^*\omega^*|_p - \alpha\vec{c}^*\sigma\left(\vec{c}|_p \cdot \vec{\chi}\right)\right\} \vec{\chi}$$
$$= \vec{c}^* \cdot \vec{c}|_p + \alpha\vec{c}^* \cdot \vec{\chi} \ge \vec{c}^* \cdot \vec{c}|_p + \alpha\delta$$

the last inequality being the linear separability hypothesis. And after M corrections, taking p=0 for the initial state, this reads $\vec{c}^* \cdot \vec{c}|_M \ge \vec{c}^* \cdot \vec{c}|_0 + \alpha M\delta$.

For the denominator, in erroneous cases only,

$$\left|\vec{c}|_{p+1}\right|^2 = \left|\vec{c}|_p\right|^2 + 2\alpha\vec{c}|_p \cdot \vec{\chi} + \alpha^2|\vec{\chi}|^2 \le \left|\vec{c}|_p\right|^2 + \alpha^2$$

since $|\vec{\chi}|^2=1$ and the middle term is negative in erroneous cases. After M corrections, starting at p=0 this reads:

$$\left|\vec{C}\big|_M\right|^2 \leq \left|\vec{C}\big|_0\right|^2 + \alpha^2 M .$$

Assuming $\vec{C}\big|_0 = \vec{0}$ which does not affect the results, and collecting terms,

$$\frac{\alpha M \delta}{\alpha \sqrt{M}} \leq 1 \Rightarrow \frac{1}{\delta^2} \geq M$$

which bounds the number of steps for the inequality:

$$\vec{C}\big|_M \cdot \vec{\chi} > \delta \qquad \forall \vec{\chi} \in \left\{x^\kappa\right\} \left\{x^\lambda\right\}.$$

APPENDIX VII

Program C-code

A-VII-1 Learning program

```
/*ORTHO_15.C    Version EUROCO TURBO C
        derniere revision 22-juillet-97
   version avec le facteur g et SIGON
 27-juillet-97 pret a repasser en LINUX ou UNIX on le mettra
sur ALCOR Couplages lineaires, non lineaires, constantes des
sigmoides et seuils, tous reglables separement par les taux
d'apprentissage. Apprentissage a deux couches (avec biais),
pour les couplages lineaires seulement.
Le programme BYTES_15.C calcule le nombre de bytes requis
par ORTHO_15.C.
Contient les facteurs de decroissance comme dans les
versions LINUX. La version _15 contient et fonctionne
qu'avec les i=iprime. A partir de la version 15 on
s'efforcera de ne plus avoir une version TURBO et une
version UNIX, on essayera d'avoir une version UNIQUE!*/
#include<stdio.h>  /*pour printf, scanf etc*/
#include<stdlib.h>     /*pour les acces aux fichiers
disque*/
#include<fcntl.h>  /*pour les valeurs O_WRONLY etc*/
#include<math.h>   /*pour SIN, SQRT,...des variables
dependantes*/
/*#include<conio.h pour kbhit() enleve pour LINUX et UNIX*/

/* valeurs extremes requises pour les definitions des
matrices */
#define NICO  11     /* Nb maximum de couches */
#define NCL  2     /* Nb Max de classes */
#define DRMAX  20     /* Nb Max de cellules par couche */
#define NILS  500     /* Nb Max d'exemples dans l'ensemble
d'apprentissage */

float Sigmoide(float x,float c, float theta);/*sigmoide
centree avec seuil*/
float dSdx(float x,float c, float theta);/*del S/del x*/
float dSdc(float x,float c, float theta);/*del S/del c*/
float dSdtheta(float x,float c, float theta);/*del S/del
theta*/
float monrandom(float y);
float absf(float x);/*valeur absolue d'un point flotant*/
```

```
void journal(char *typp,char *nf);

main()
{
/*FICHIERS*/
 FILE *fp3;
 char *nomfichier3;/*deux classes melangees (fichier
d'entree)*/
 char rep;
 FILE *fp5;/*fichier d'architecture, pour les couplages non-
aleatoires*/
 char *nomfichier5;
 FILE *fp6;
 char *nomfichier6;/*fichier de mesures d'apprentissage.Ax*/
 char *ty,*nf;/*pour le journal automatique*/


/*ENSEMBLE D'APPRENTISSAGE*/
 int Nk[NCL];/*nombre d'exemples EFFECTIF de chaque classe*/
 int nils;/*nb total d'exemples = Nk[0] + Nk[1]
       mais nils est lu dans le fichier, les autres sont
comptes
       par la suite*/
 int classe[NILS];
 int np;/*nombre de parametres, nb d'entrees du reseau*/
 int ncl;/*nombre de classes*/

/*COMPTEURS*/
 int ils;/*compteurs d'exemples de l'ensemble
d'apprentissage*/
 int i;/*cellules du <<premier>> niveau*/
 int j;/*cellules du <<second niveau>>*/
 int k,kprime;/*indices de classes*/
 int ico;/*compteur de couches*/
 unsigned int npasse;/*compteur du nombre de passages dans
l'orthogonalisation*/
 int ii,jj,iiprime,iico;/*bidons pour la lecture d'une
architecture*/

/*ARCHITECTURE NEURONALE    */
 int nico;/*nombre de couches*/
 int dr[NICO];/*nb de neurones de la [couche No]*/
 int imax,jmax,lmax,pre,iprime,post;
 float COMAX;/*valeur maximale pour les couplages
aleatoires*/
 float racine;
 float z[NICO][DRMAX][NILS];
 float c[NICO][DRMAX];
 float theta[NICO][DRMAX],moytheta[DRMAX];
 float C[NICO][DRMAX][DRMAX];
 float D[NICO][DRMAX][DRMAX][DRMAX];
/*couplages: l'indice de couche est la couche
initiale:[pre][i][i'][j]*/
 int flag_couches_pareilles;

/*APPRENTISSAGE*/
 float Rkk[NICO][NCL][NCL];
 float SOMkfy[NICO][DRMAX][NCL];
 float SOMkdfdyfx[NICO][DRMAX][DRMAX][NCL];
 float SOMkdfdyfxfx[NICO][DRMAX][DRMAX][DRMAX][NCL];
 float SOMkdfdc[NICO][DRMAX][NCL];
 float SOMkdfdt[NICO][DRMAX][NCL];
 float bec1,bec2,betheta1,betheta2;
 float beC1,beC2,beD1,beD2,f;
 float
decroi,bec1o,bec2o,betheta1o,betheta2o,beC1o,beC2o,beD1o,beD
2o;
 float
zj,cj,tj,fidexi,fxik,fjdeyj,delfjdelyjk,fiprimedexiprime,Xlm
ax;
 float zi,ci,ti,TA,CORR,Xjmax,Norm,A,B,ico4;
 float XNk,dri,densite,f_mixte,f_carre,g;
 int cl,couche_l,l;
 char flagsynchro,flagnorma,flagcentre;

/*VARIABLES POUR LES TESTS, LES MESURES ET L'AFFICHAGE*/
 unsigned int pasaf;/*Nb de passe entre chaque affichage et
mise en fichier*/
 char car;/*caractere pour getchar*/
 unsigned int passeMax;
 float U;
/*VARIABLES SPECIALES pour le bloc de correction des taux
avec la fonction U
 a la toute fin du programme*/
  float xyz;
 float avCOc,avCOt,avCOC,avCOD;
 int NCc,NCt,NCC,NCD;
 float valeurs_des_c;

/*-----initialiser tout, utilise ou non-----*/
/*variables a un seul indice*/
 for(k=0;k<NCL;k++){Nk[k]=0;}
 for(ils=0;ils<NILS;ils++){classe[ils]=0;}
 for(i=0;i<DRMAX;i++){moytheta[i]=0.00;}
/*Variables a plusieurs indices*/
 for(ico=0;ico<NICO;ico++)
 {
  dr[ico]=0;
  for(i=0;i<DRMAX;i++)
  {
   c[ico][i]=1.00;theta[ico][i]=0.00;
   for(ils=0;ils<NILS;ils++)
   {
    z[ico][i][ils]=0.00;
   }/*ils*/
   for(j=0;j<DRMAX;j++)
   {
    C[ico][i][j]=0.00;
    for(iprime=0;iprime<DRMAX;iprime++)
    {
    D[ico][i][iprime][j]=0.00;
       for(k=0;k<NCL;k++)
    {
     SOMkdfdyfxfx[ico][i][iprime][j][k]=0.00;
    }/*k*/
    }/*iprime*/
    for(k=0;k<NCL;k++)
    {
    SOMkdfdyfx[ico][i][j][k]=0.00;
    }/*k*/
   }/*j*/
   for(k=0;k<NCL;k++)
   {
    SOMkfy[ico][i][k]=0.00;
    SOMkdfdc[ico][i][k]=0.00;
    SOMkdfdt[ico][i][k]=0.00;
   }/*k*/
   }/*i*/
   for(k=0;k<NCL;k++)
   {
   for(kprime=0;kprime<NCL;kprime++)
   {
    Rkk[ico][k][kprime]=0.00;
   }/*kprime*/
   }/*k*/
  }/*ico*/
/*DEBprintf("\nFini les initialisations getch");getchar();*/
/*PARAMETRES ET NOMS DE FICHIERS A FIXER AVANT D'EXECUTER*/
/* fichiers disponibles:  XOR.DAT  ASD.EA  PAIR10.DAT
PAIR10PO.DAT*/
nomfichier3="/home/bandeco/ortho/markaria/eamark.255";/*fich
ier de donnees*/
/*nomfichier5="C:\\TC\\ORTHO\\TOY_DATA\\SCEAVAX.A03";archite
cture anterieure*/
nomfichier6="/home/bandeco/ortho/markaria/mar255.M200";/*mes
ures a ecrire*/
/*TAUX D'APPRENTISSAGE  1: classes pareilles   2: classes
differentes----*/
betheta1=30.0;betheta2=-70.0;     /*seuils*/
```

```
bec1=0.000;bec2=0.000;           /*constantes des
sigmoides*/
valeurs_des_c=0.5473;
beC1=34.0;beC2=-92.0;          /*couplages lineaires*/
beD1=44.0;beD2=-106.0;          /* couplages non lineaires*/
   /*Si beD1=-beD2 Il y a probleme aux alentours de la ligne
517*/
f=0.1;
   /*biais pour l'apprentissage a deux couches pour enlever
la dependence
   en Uy de la correction des Cij, voir aux alentours de la
ligne 490*/
f_mixte=1.00;/*facteur pour les termes xi*xiprime avec
iprime!=i*/
f_carre=1.00;/*facteur pour les termes xi*xiprime avec
iprime=i*/
   /*ATTENTION: leurs valeurs n'ont de sens que pour UN et
ZERO, 309 & 310*/
g=1.0;/*facteur devant delUy_delCij pour un 3 couches au
sens de Frank
     1.0:Macot     0.0:Frank*/
decroi=0.0;
   /*facteur de decroissance des taux d'une couche a
l'autre. Si on met
   decroi=0.0 la fonction qui calcule la decroissance des
taux avec la
   couche est profondeur de la est annulee, aux alentour de
la ligne 610*/
/*ouvrir le fichier 3*/
if((fp3=fopen(nomfichier3,"r"))==NULL)
 {printf("\nImpossible d'ouvrir le fichier d'entree");goto
hell;}
 fscanf(fp3,"%d %d %d",&nils,&np,&ncl);
/*--EST-TU BIEN SUR QUE C'EST LE BON NOM DE FICHIER!!!---*/
nico=6;    /*Nb de couches*/
dr[0]=8;dr[1]=8;dr[2]=8;dr[3]=8;dr[4]=8;dr[5]=8;
pasaf=200;/*mesures et affichage a tous les pasaf*/
passeMax=24000;/*Nb max. de passe allouees pour
l'apprentissage*/
racine=0.90;/*pour le generateur aleatoire*/
COMAX=46.0;/*valeur maximale des couplages aleatoires*/
/*COMAX et racine sont inscrites au journal ET dans le
fichier de mesures*/
flagsynchro='O';/*On ne teste que la valeur 'N', n'importe
quoi=OUI 630 a 670*/
flagnorma='N';/*on teste la valeur 'O', n'importe quoi=NON
ligne 440*/
flagcentre='N';/*Centrer le probleme en fixant les seuils
des entrees,
            on ne teste que le oui, environ ligne 455*/
```

```
flag_couches_pareilles=1;/*valide seulement quand les
nombres de cellules
                          sont compatibles, naturellement*/
densite=0.875;/*determine la densite des couplages non nuls,
avec 0.75
    1/4 des couplages sont forces a la valeur zero (et le
restent?)*/
/* IL Y A AUSSI
 - les constantes des sigmoides gelees: ligne 332
 - facteur 1/(1+f) aux alentours de la ligne 714
(Actuellement il est installe)
  DECLARER SON ETAT LIGNE 269
 - Diminution des taux d'apprentissage au fur et a mesure
que l'on
 avance dans les couches, hard coded aux environs de la
ligne 615
*/
/*-----test de compatibilite des valeurs extremes------*/
 if(nico>NICO){printf("\nNICO est trop petit");goto hell;}
 if(ncl>NCL){printf("\nNCL est trop petit");goto hell;}
 if(nils>NILS){printf("\nNILS est trop petit");goto hell;}
 for(ico=0;ico<nico;ico++)
 {
  if(dr[ico]>DRMAX){printf("\nDRMAX est trop petit pour la
couche %d",ico);goto hell;}
 }/*ico*/
/*---------------ecriture au journal------------------*/
/* ty="J-M-A";nf="...";journal(ty,nf);
 ty="Ordinateur:";nf="PENTIUM, LINUX C";
 ty="Programme: (avec i'=i)";nf="ORTHO_15.C";journal(ty,nf);
 ty="Fichier de donnees:";journal(ty,nomfichier3);
 ty="Fichier d'une architecture
anterieure:";journal(ty,nomfichier5);
 ty="Fichier de mesures:";journal(ty,nomfichier6);
 ty="Synchronisation:";
     if(flagsynchro=='N'){journal(ty,"NON");}
     else{journal(ty,"OUI");}
 ty="Centrer avec les seuils d'entrees:";
     if(flagcentre=='O'){journal(ty,"OUI");}
     else{journal(ty,"NON");}*/
/*ECRITURE DE L'ENTETE AU FICHIER DE MESURES ET A L'ECRAN*/
/*ouvrir le fichier 6 (mesures)*/
 if((fp6=fopen(nomfichier6,"w"))==NULL)
  {
  printf("\nImpossible d'ouvrir le fichier de mesures");
   goto hell;
 }
  fprintf(fp6,"Fichier_de_mesures: %s",nomfichier6);
 fprintf(fp6,"   Programme_ORTHO_15.C");
```

```
  fprintf(fp6,"\n-------------------------------------------
------------");
  fprintf(fp6,"\nensemble_d'apprentissage: %s",nomfichier3);
  fprintf(fp6,"\n  nombre_d'exemples: %d",nils);
  fprintf(fp6,"\n   nombre_d'entrees: %d",np);
  fprintf(fp6,"\n  nombre_de_classes: %d",ncl);
  fprintf(fp6,"\n  nombre_de_couches: %d",nico);
  printf("\n\n\n\n");
  printf("\nORTHO_15.C  Fichier_de_mesures: %s",nomfichier6);
  printf("\n--------------ensemble d'apprentissage-----------
--------------");
  printf("\nfichier: %s",nomfichier3);
  printf("\n nils: %d   np: %d   ncl: %d",nils,np,ncl);
  printf("\n--------------------architecture------------------
--------------");
  printf("\n           nombre_de_couches: %d",nico);
  printf("\ndensite de couplages non-nuls: %f  COMAX:
%f",densite,COMAX);
  printf("\n nombre de cellules par couche");
  for(i=0;i<nico;i++)
  {
   fprintf(fp6,"\n      couche %d: %d neurones ",i,dr[i]);
   printf("\n       couche %d: %d neurones ",i,dr[i]);
  }
  fprintf(fp6,"\nsynchronisation: %c",flagsynchro);
  fprintf(fp6,"\nnormalisation_des_entrees: %c",flagnorma);
   fprintf(fp6,"\nflagcentre: %c",flagcentre);
  fprintf(fp6,"\ndensite_des_couplages_non_nuls:
%f",densite);
  fprintf(fp6,"\nbiais_pour_deux_couches: %f",f);
  fprintf(fp6,"  f_mixte=%f  f_carre=%f",f_mixte,f_carre);
  printf("\nf_mixte=%f  f_carre=%f",f_mixte,f_carre);
  fprintf(fp6,"\ndecroi=%f--COMAX=%f--racine=%f--
g=%f",decroi,COMAX,racine,g);
  printf("\n--------------------apprentissage------------
TAUX---------k=k'---k<>k'");
  printf("\n               synchronisation: %c",flagsynchro);
  printf("          Constantes   %5.3f  %5.3f",bec1,bec2);
  printf("\n    normalisation des entrees: %c",flagnorma);
  printf("          Seuils       %5.3f
%5.3f",betheta1,betheta2);
  printf("\n                          racine: %f",racine);
  printf("    C. lineaires %5.3f  %5.3f",beC1,beC2);
  printf("\n    pasaf: %3d     passe MAX:
%d",pasaf,passeMax);
  printf("         C. NON-lin.  %5.3f  %5.3f",beD1,beD2);
  printf("\nBiais, C. lineaires SEULEMENT f: %f facteur
1/(1+f): ACTIF",f);
/*===INITIALISATION DE LA STRUCTURE NEURONALE===*/
  for(ico=0;ico<nico;ico++)
  {
   for(i=0;i<dr[ico];i++)
   {
    if(ico==0)/*seuils est constantes des sigmoides de la
premiere couche*/
    {
     theta[ico][i]=0.00;
     c[ico][i]=1.00;
    }
    else /*la couche courante n'est pas la premiere couche*/
    {
/*c'est ici qu'on regle les valeurs initiales des seuils et
des constantes
des sigmoides du reseau APRES la premiere couche*/
     theta[ico][i]=0.1;
     racine=monrandom(racine);
     if(racine<=0.5){theta[ico][i]=-0.1;}
/*CHOISIR ICI LES VALEURS DES CONSTANTES DES SIGMOIDES
LORSQUE LES TAUX!=0*/
/*racine=monrandom(racine);c[ico][i]=(0.5-racine)*2.00;*/
/*c[ico][i]=valeurs_des_c;initialisation uniforme des
constantes des sigmoides*/
/*initialisation + et - des constantes des sigmoides*/
     c[ico][i]=valeurs_des_c;
     racine=monrandom(racine);
     if(racine<=0.5){c[ico][i]=-1.0*valeurs_des_c;}
  /*les sigmoides gelees c'est pas ici, c'est aux alentours
de la ligne 340*/
    }
    if(ico<nico-1)/*Si ico est une <<pre>>, pour les
couplages C et D*/
    {
     for(j=0;j<dr[ico+1];j++)
     {
      racine=monrandom(racine);
      if(racine<=densite)
      {
       racine=monrandom(racine);
       C[ico][i][j]=COMAX*(0.5-racine);

if((ico>0)&&(flag_couches_pareilles==1)&&(dr[ico]==dr[ico+1]
)&&(dr[ico-1]==dr[ico]))
       {
        C[ico][i][j]=C[ico-1][i][j];
       }
      }
      else{C[ico][i][j]=0.00000;}
      for(iprime=i;iprime<dr[ico];iprime++)  /*i'=i ICI*/
      {
       if((beD1==0.0)&&(beD2==0.0))
      {D[ico][i][iprime][j]=0.00000;goto Katmandu;}
 /*si les (2) taux d'apprentissages pour les couplages non-
lineaires sont
  nuls, alors les couplages sont nuls aussi, ils ne sont pas
installes*/
      racine=monrandom(racine);
      if(racine<=densite)
      {
       racine=monrandom(racine);
       if(i==iprime)
       {
        D[ico][i][iprime][j]=f_carre*COMAX*(0.5-racine);

if((ico>0)&&(flag_couches_pareilles==1)&&(dr[ico]==dr[ico+1]
)&&(dr[ico-1]==dr[ico]))
        {
         D[ico][i][iprime][j]=D[ico-1][i][iprime][j];
        }
       }
       else
       {
        D[ico][i][iprime][j]=f_mixte*COMAX*(0.5-racine);

if((ico>0)&&(flag_couches_pareilles==1)&&(dr[ico]==dr[ico+1]
)&&(dr[ico-1]==dr[ico]))
        {
         D[ico][i][iprime][j]=D[ico-1][i][iprime][j];
        }
       }
 /*ici les f_ servent de <<flags>> et de facteurs en meme
temps*/
      }
      else
      {
       D[ico][i][iprime][j]=0.00000;
      }
Katmandu: ;}/*iprime*/
    }/*j*/
   }/*if*/
  }/*i*/
 }/*ico*/
 if((bec1==0.0)&&(bec2==0.0))
 {
  for(ico=0;ico<nico;ico++)
  {
   for(i=0;i<dr[ico];i++)
   {
/*CHOISIR ICI LES VALEURS DES CONSTANTES DES SIGMOIDES
GELEES*/
/*initialisation uniforme   c[ico][i]=valeurs_des_c;*/
```

```
/*initialisation + et - des constantes des sigmoides*/
    c[ico][i]=valeurs_des_c;
    racine=monrandom(racine);
    if(racine<=0.5){c[ico][i]=-1.0*valeurs_des_c;}
   }/*i*/
  }/*ico*/
 }/*if les sigmoides sont gelees, pour les constantes*/
 if((betheta1==0.0)&&(betheta2==0.0))
 {
  for(ico=1;ico<nico;ico++)
    /*ico=1: les seuils de la premiere couche ne sont JAMAIS
nuls*/
  {
   for(i=0;i<dr[ico];i++)
   {
/*mettre ici les valeurs assignees aux seuils des sigmoides
gelees*/
    theta[ico][i]=0.00;
   }/*i*/
  }/*ico*/
 }/*if les sigmoides sont gelees, pour les seuils*/
/*-----------------------------------------------------*/
 printf("\n----------------origine de l'architecture-------
--------------");
 printf("\n1 - couplages initiaux aleatoires");
 printf("\n2 - couplages initiaux lus d'un fichier
d'architecture VERIFIER!");
 printf("\n3 - Sortir  ");
/*DEB car='1';*/
 scanf("%c",&car);
 if(car=='3'){goto hell;}
 if(car=='1')
 {
  ty="Architecture initiale:";nf="aleatoire";journal(ty,nf);
  goto gogogo;
 }
 ty="Architecture initiale:";nf="anterieure";journal(ty,nf);
/*====LECTURE DE L'ARCHITECTURE ANTERIEURE====*/
/*VERIFIER LA COHERENCE D'AVEC MESARC_12.C et _15 C'est
encore a faire*/

 if((fp5=fopen(nomfichier5,"r"))==NULL)
 {
  printf("\nImpossible d'ouvrir le fichier d'architecture");
  goto hell;
 }
 fscanf(fp5,"%d",&nico);/*nombre de couches*/
/*VERIFIER!!!! (ces ligne n'existent pas dans la version
LINUX)
     fscanf(fp5,"%c",&car);fonctions non lineaires
```

```
              0: sigmoide
              1: compresseur polynomial 1*/
/*nombres de neurones par couches*/
 fscanf(fp5,"\n");for(ico=0;ico<nico;ico++){fscanf(fp5," %d
",&dr[ico]);}
/*Valeurs des parametres de l'architecture: c, theta, C et
D*/
 for(ico=0;ico<nico;ico++)
 {
  fscanf(fp5,"\n %d ",&iico);
  if(ico!=iico)
     {printf("\nico=%d incompatible avec
iico=%d",ico,iico);goto hell;}
  for(i=0;i<dr[ico];i++)
  {
   fscanf(fp5,"\n %d %f %f ",&ii,&c[ico][i],&theta[ico][i]);
   if(ico<nico-1)
   {
    for(j=0;j<dr[ico+1];j++)
    {
     fscanf(fp5,"\n %d %d %d %f
",&iico,&ii,&jj,&C[ico][i][j]);
     for(iprime=i;iprime<dr[ico];iprime++)/*i'=i On lira
tout meme si c'est des zeros*/
     {
      fscanf(fp5,"\n %d %d %d %d %f
",&iico,&ii,&iiprime,&jj,&D[ico][i][iprime][j]);
     }/*iprime*/
    }/*j*/
   }/*if*/
  }/*i*/
 }/*ico*/
 fclose(fp5);/*fichier d'architecture*/
 gogogo:
/*DEBprintf("\n---------------------gogogo---------------
-----------");getchar();*/
/*--------mise en fichier des valeurs initiales-------------
--*/
 fprintf(fp6,"\nValeurs des taux d'apprentissage
");/*fp6=fichier de mesures*/
 fprintf(fp6,"\nbec= %f %f",bec1,bec2);
 fprintf(fp6,"\nbetheta= %f %f",betheta1,betheta2);
 fprintf(fp6,"\nbeC= %f %f",beC1,beC2);
 fprintf(fp6,"\nbeD= %f %f",beD1,beD2);
 fprintf(fp6,"\nf[pour deux couches]= %f",f);
 fprintf(fp6,"\nValeurs initiales ");
 for(ico=0;ico<nico;ico++)
 {
  fprintf(fp6,"\nCouche(pre)=%d",ico);
  for(i=0;i<dr[ico];i++)
```

```
   {
   fprintf(fp6,"\n %d %f %f ",i,c[ico][i],theta[ico][i]);
   if(ico<nico-1)
   {
    for(j=0;j<dr[ico+1];j++)
    {
    fprintf(fp6,"\nC[%d][%d][%d]=%f ",ico,i,j,C[ico][i][j]);
    for(iprime=i;iprime<dr[ico];iprime++)      /*i'=i ICI*/
    {
     fprintf(fp6,"\nD[%d][%d][%d][%d]=%f
",ico,i,iprime,j,D[ico][i][iprime][j]);
    }/*iprime*/
    }/*j*/
   }/*if*/
   }/*i*/
  }/*ico*/
  fprintf(fp6,"\n---------------------------------------
------------");
/*===FIN DE L'INITIALISATION DE LA STRUCTURE NEURONALE===*/
/*DEBprintf("\n================ENTREE DES
DONNEES===================");getchar();*/
 for(i=0;i<ncl;i++){Nk[i]=0;}
 for(i=0;i<dr[0];i++){moytheta[i]=0.0;}
/*--------------normalisation des entrees----------------*/
 for(ils=0;ils<nils;ils++)
/*c'est en fait une boucle de lecture, pas une boucle de
normalisation*/
 {
  Norm=0.0;
  for(i=0;i<dr[0];i++)
  {
  fscanf(fp3,"%f",&z[0][i][ils]);
  Norm=Norm+z[0][i][ils]*z[0][i][ils];
  }/*i*/
  Norm=(float)sqrt((double)Norm);
  if(flagnorma=='O')
  {
  for(i=0;i<dr[0];i++)
  {
   z[0][i][ils]=z[0][i][ils]/Norm;
  }/*i*/
  }/*if <<il y a normalisation>>*/
  for(i=0;i<dr[0];i++)
  {
  moytheta[i]=moytheta[i]+z[0][i][ils];
  }/*i*/
  fscanf(fp3,"%d",&classe[ils]);
  k=classe[ils];
  Nk[k]++;
 }/*ils*/
 fclose(fp3);/*Fichier d'ensemble d'apprentissage*/
 if(flagcentre=='O')
 {
  for(i=0;i<dr[0];i++)
  {
  theta[0][i]=moytheta[i]/(float)nils;
  }/*i*/
 }/*if on veut centrer le probleme ou non*/
/*==fini de lire les entree et leur classe assignees==*/
/*=========CALCULSD'ORTHOGONALITES=========*/
 npasse=0;
/*DEB;printf("Juste avant boucle ncl=%d
nico=%d",ncl,nico);getchar();*/
boucle:
/*variables speciales d'observations*/
 avCOc=avCOt=avCOC=avCOD=0.00;
 NCc=NCt=NCC=NCD=0;
 npasse++;
 if(npasse%pasaf==0)
 {
  printf("\nPASSE: %u  fichier d'apprentissage:
%s",npasse,nomfichier3);
 }
/*--Toutes les SOMk...INITIALISER a zero A CHAQUE PASSE
D'APPRENTISSAGE---*/
 for(k=0;k<ncl;k++)
 {
  for(ico=0;ico<nico;ico++)  /*ico=couche*/
  {
  imax=dr[ico];
  for(i=0;i<imax;i++)
  {
   SOMkdfdc[ico][i][k]=0.00;
   SOMkdfdt[ico][i][k]=0.00;
   SOMkfy[ico][i][k]=0.00;/*la premiere couche est utilisee
pour c et theta*/
   if(ico<nico-1)/*tant que ico est un indice
presynaptique*/
   {
   jmax=dr[ico+1];
   for(j=0;j<jmax;j++)
   SOMkdfdyfx[ico][i][j][k]=0.00;
   for(iprime=0;iprime<imax;iprime++)
   {
    SOMkdfdyfxfx[ico][i][iprime][j][k]=0.00;
   }/*iprime*/
   }/*if ico est presynaptique*/
  }/*i*/
  }/*ico*/
 }/*k (pour la classe)*/
```

```
/*DEBprintf("\n------------calcul des etats des neurones----
-----------");*/
/*DEBprintf("\nnils=%d  ",nils);getchar();*/
 for(ils=0;ils<nils;ils++)
 {
  for(pre=0;pre<nico-1;pre++)
  {
  post=pre+1;
  imax=dr[pre];
  jmax=dr[post];
/*DEBprintf("\nils=%3d  post=%d jmax=%d pre=%d
imax=%d",ils,post,jmax,pre,imax);*/
  for(j=0;j<jmax;j++)
  {
   z[post][j][ils]=0.0;
   for(i=0;i<imax;i++)
   {
   zi=z[pre][i][ils];ci=c[pre][i];ti=theta[pre][i];
   fxik=Sigmoide(zi,ci,ti);
   z[post][j][ils]=z[post][j][ils]+C[pre][i][j]*fxik;
   if((beD1+beD2)!=0.0)
   {
    for(iprime=i;iprime<imax;iprime++)   /*i'=i ICI */
    {
zi=z[pre][iprime][ils];ci=c[pre][iprime];ti=theta[pre][iprim
e];
z[post][j][ils]=z[post][j][ils]+D[pre][i][iprime][j]*fxik*Si
gmoide(zi,ci,ti);
 /*c'est le couplage D qui joue le role de selecteur ici*/
    }/*iprime*/
   }/*if, il faut tenir compte des couplages non-lineaires*/
   }/*i*/
  }/*j*/
  }/*pre*/
 }/*ils*/
/*DEBprintf("\n------------calcul des termes des derivees---
-----------");*/
 for(ils=0;ils<nils;ils++)
 {
  cl=classe[ils];
  for(ico=0;ico<nico;ico++)   /*ico=couche*/
  {
  jmax=dr[ico];
  for(j=0;j<jmax;j++)
  {
   zj=z[ico][j][ils];cj=c[ico][j];tj=theta[ico][j];
   SOMkdfdc[ico][j][cl]=SOMkdfdc[ico][j][cl]+dSdc(zj,cj,tj);
SOMkdfdt[ico][j][cl]=SOMkdfdt[ico][j][cl]+dSdtheta(zj,cj,tj)
;
   fjdeyj=Sigmoide(z[ico][j][ils],c[ico][j],theta[ico][j]);
```

```
    SOMkfy[ico][j][cl]=SOMkfy[ico][j][cl]+fjdeyj;
   }/*j*/
   }/*ico*/
   for(pre=0;pre<nico-1;pre++)
   {
   imax=dr[pre];
   post=pre+1;jmax=dr[post];
   for(j=0;j<jmax;j++)
   {
delfjdelyjk=dSdx(z[post][j][ils],c[post][j],theta[post][j]);
    for(i=0;i<imax;i++)
    {
    fidexi=Sigmoide(z[pre][i][ils],c[pre][i],theta[pre][i]);

SOMkdfdyfx[pre][i][j][cl]=SOMkdfdyfx[pre][i][j][cl]+delfjdel
yjk*fidexi;
    for(iprime=i;iprime<imax;iprime++)     /*i'=i ICI*/
    {
fiprimedexiprime=Sigmoide(z[pre][iprime][ils],c[pre][iprime]
,theta[pre][iprime]);

SOMkdfdyfxfx[pre][i][iprime][j][cl]=SOMkdfdyfxfx[pre][i][ipr
ime][j][cl]+delfjdelyjk*fidexi*fiprimedexiprime;
    }/*iprime*/
    }/*i*/
   }/*j*/
   }/*pre*/
 }/*ils*/
/*DEBprintf("\n-------------------divisions par Nk----------
------------");*/
 for(cl=0;cl<ncl;cl++)
 {

  XNk=(float)Nk[cl];
/*Il est possible d'augmenter, artificiellement, l'effet
d'une classe sur l'apprentissage en modifiant la valeur de
XNk ici. Par exemple, si on divise XNk des hadrons par deux,
on augmente l'influence de la classe HADRON dans l'evolution
de l'apprentissage. J'essaye avec facteur[cl]. Ca bousille
la structure de la fonction U, il faut sans doute agir au
moment des corrections. C'est comme si on avait un taux
d'apprentissage pour une classe et un autre pour l'autre
classe. Ca pourait vouloir dire que l'on ne se deplace plus
dans la direction du gradient. Non, on reste dans la
direction du grdient mais, au moment des corrections, ca ne
fait que changer le taux d'apprentissage global parceque les
"produits des classes" dans les derivees et cela annule le
"favoritisme". La seule solution semble bien etre
ORTHO_17.C*/
  for(ico=0;ico<nico;ico++)
```

```
   {
   jmax=dr[ico];
   for(j=0;j<jmax;j++)
   {
    SOMkdfdc[ico][j][cl]=SOMkdfdc[ico][j][cl]/XNk;
    SOMkdfdt[ico][j][cl]=SOMkdfdt[ico][j][cl]/XNk;
    SOMkfy[ico][j][cl]=SOMkfy[ico][j][cl]/XNk;
   }/*j*/
   }/*ico pour les deux SOMk qui appartiennent a toutes les
couches*/
   for(pre=0;pre<nico-1;pre++)
   {
   imax=dr[pre];post=pre+1;jmax=dr[post];
   for(j=0;j<jmax;j++)
   {
    for(i=0;i<imax;i++)
    {
    SOMkdfdyfx[pre][i][j][cl]=SOMkdfdyfx[pre][i][j][cl]/XNk;
    for(iprime=i;iprime<imax;iprime++)    /*i'=i ICI*/
    {
SOMkdfdyfxfx[pre][i][iprime][j][cl]=SOMkdfdyfxfx[pre][i][ipr
ime][j][cl]/XNk;
    }/*iprime*/
    }/*i*/
   }/*j*/
   }/*pre*/
  }/*cl*/
/*DEBprintf("\ncorrections des couplages et des parametres
pour une couche");*/
 for(k=0;k<ncl;k++) /*pour alpha de la theorie*/
 {
  for(kprime=k;kprime<ncl;kprime++) /*pour alphaprime de la
theorie*/
  {
  for(ico=0;ico<nico;ico++)   /*ico=couche*/
  {
bec1o=bec1;bec2o=bec2;betheta1o=betheta1;betheta2o=betheta2;
   beC1o=beC1;beC2o=beC2;beD1o=beD1;beD2o=beD2;
   if((decroi!=0.00)&&(ico>0))
   {        /*de maniere a avoir un flag pour la
decroissance des taux*/
   ico4=(float)ico;
   bec1o=bec1/(ico4*decroi);
   bec2o=bec2/(ico4*decroi);
   betheta1o=betheta1/(ico4*decroi);
   betheta2o=betheta2/(ico4*decroi);
   beC1o=beC1/(ico4*decroi);
   beC2o=beC2/(ico4*decroi);
   beD1o=beD1/(ico4*decroi);
   beD2o=beD2/(ico4*decroi);
```

```
    }
    imax=dr[ico];dri=(float)imax;
    for(i=0;i<imax;i++)
    {
if(flagsynchro=='N'){racine=monrandom(racine);if(racine<0.5)
{goto J1;}}
    if(k==kprime){TA=bec1o;}else{TA=bec2o;}/*constantes des
sigmoides*/
CORR=(SOMkdfdc[ico][i][k]*SOMkfy[ico][i][kprime]+SOMkdfdc[ic
o][i][kprime]*SOMkfy[ico][i][k])/dri;
    c[ico][i]=c[ico][i]+TA*CORR;/*correction des constantes
des sigmoides*/
/*speciale*/   avCOc=avCOc+TA*CORR;NCc++;
J1:
if(flagsynchro=='N'){racine=monrandom(racine);if(racine<0.5)
{goto J2;}}
    if(k==kprime){TA=betheta1o;}else{TA=betheta2o;}
CORR=(SOMkdfdt[ico][i][k]*SOMkfy[ico][i][kprime]+SOMkdfdt[ic
o][i][kprime]*SOMkfy[ico][i][k])/dri;
    theta[ico][i]=theta[ico][i]+TA*CORR;/*correction des
seuils*/
/*speciale*/   avCOt=avCOt+TA*CORR;NCt++;
J2:   if(ico<nico-1)  /*ico=pre*/
    {
     post=ico+1;jmax=dr[post];Xjmax=(float)jmax;
     for(j=0;j<jmax;j++)
     {
if(flagsynchro=='N'){racine=monrandom(racine);if(racine<0.5)
{goto J3;}}
     if(k==kprime)
     {
      TA=beC1o;
     }
     else
     {
      TA=beC2o;
     }
CORR=(SOMkdfdyfx[ico][i][j][k]*SOMkfy[post][j][kprime]+SOMkd
fdyfx[ico][i][j][kprime]*SOMkfy[post][j][k])/Xjmax;
     if(C[ico][i][j]!=0.00000)
     {
      C[ico][i][j]=C[ico][i][j]+g*TA*CORR;
/*speciale*/   avCOC=avCOC+TA*CORR;NCC++;
     }
     /*correction des couplages lineaires, une couche
apprenante*/
   /*si un couplage lineaire devient nul, il le reste
     Si il est nul il n'est plus modifie*/
J3:      for(iprime=i;iprime<imax;iprime++)
     {
```

```
if(flagsynchro=='N'){racine=monrandom(racine);if(racine<0.5)
{goto J4;}}
      if(k==kprime)
      {
      TA=beD1o;
      }
      else
      {
      TA=beD2o;
      }
CORR=(SOMkdfdyfxfx[ico][i][iprime][j][k]*SOMkfy[post][j][kpr
ime]+SOMkdfdyfxfx[ico][i][iprime][j][kprime]*SOMkfy[post][j]
[k])/Xjmax;
      if(D[ico][i][iprime][j]!=0.000000)
      {
      D[ico][i][iprime][j]=D[ico][i][iprime][j]+TA*CORR;
/*speciale*/   avCOD=avCOD+TA*CORR;NCD++;
      }
  /*si un couplage non-lineaire devient nul, il le reste
    Si il est nul il n'est plus modifie*/
J4: ;  }/*iprime*/
    }/*j*/
   }/*if ico=pre*/
   }/*i*/
  }/*ico*/
  if(f!=0.0)  /*test du biais pour l'apprentissage a deux
couches*/
  {
/*DEBprintf("\n----Correction des couplages lineaires, deux
couches----");*/
if(flagsynchro=='N'){racine=monrandom(racine);if(racine<0.5)
{goto J5;}}
   for(pre=0;pre<nico-2;pre++)
   {
/*DEBprintf("\npre=%d",pre);*/
   imax=dr[pre];
   post=pre+1;jmax=dr[post];
   couche_l=post+1;lmax=dr[couche_l];
   Xlmax=(float)lmax; /*c'est le dz de la theorie*/
/*DEBprintf("\nimax=%d post=%d jmax=%d couche_l=%d lmax=%d
Xlmax=%f",imax,post,jmax,couche_l,lmax,Xlmax);*/
   for(i=0;i<imax;i++)
   {
/*DEBprintf("\ni=%d",i);*/
     CORR=0.0;
     for(j=0;j<jmax;j++)
     {
     for(l=0;l<lmax;l++)
     {
      A=SOMkdfdyfx[pre][i][j][k]*SOMkfy[couche_l][l][kprime];
      B=SOMkdfdyfx[pre][i][j][kprime]*SOMkfy[couche_1][l][k];
      CORR=CORR+C[post][j][l]*(A+B);
/*DEBprintf("\nj=%d l=%d A=%f B=%f CORR=%f",j,l,A,B,CORR);*/
     }/*l*/
     CORR=CORR/Xlmax;
     if(k==kprime){TA=f*beC1o;}
          else{TA=f*beC2o;}
     C[pre][i][j]=C[pre][i][j]+TA*CORR/(g+f);
     /*correction des couplages lineaires pour deux couches*/
 /*  METTRE OU ENLEVER ICI LE FACTEUR  1/(1+f) ATTENTION la
presence de ce
  facteur n'est pas inscrite au fichier de mesures */
/*DEBprintf("\nC[%d][%d][%d]=%f",pre,i,j,C[pre][i][j]);*/
     }/*j*/
    }/*i*/
    }/*pre*/
   }/*if apprentissage a deux couches*/
/*DEBprintf("\non vient de sortir du test de deux
couches");*/
J5: ; }/*kprime*/
 }/*k*/
/*----------calcul des recouvrements------------*/
 for(k=0;k<ncl;k++)
 {
  for(kprime=k;kprime<ncl;kprime++)
  {
  for(ico=0;ico<nico;ico++)   /*ico=couche*/
  {
   Rkk[ico][k][kprime]=0.0;
   for(j=0;j<dr[ico];j++)
   {
Rkk[ico][k][kprime]=Rkk[ico][k][kprime]+SOMkfy[ico][j][k]*SO
Mkfy[ico][j][kprime];
   }/*j*/
   Rkk[ico][k][kprime]=Rkk[ico][k][kprime]/(float)dr[ico];
  }/*ico*/
  }/*kprime*/
 }/*k*/
 if(npasse>passeMax)
 {
  printf("\nLe nombre de passes demandees est ateint");
  goto hell;
 }
/*--------AFFICHE L'ETAT DE L'APPRENTISSAGE----------*/
 if(npasse%pasaf==0)
 {
/*-------ecrire dans le fichier de mesures-----------*/
  fprintf(fp6,"\n-------------------------------------------
-------------");
  fprintf(fp6,"\n%u ",npasse);
```

```
   fprintf(fp6,"\n %f %f",bec1,bec2);
   fprintf(fp6,"\n %f %f",betheta1,betheta2);
   fprintf(fp6,"\n %f %f",beC1,beC2);
   fprintf(fp6,"\n %f %f",beD1,beD2);
   for(ico=0;ico<nico;ico++)
   {
   fprintf(fp6,"\n %d",ico);
   for(i=0;i<dr[ico];i++)
   {
    fprintf(fp6,"\n %d %f %f ",i,c[ico][i],theta[ico][i]);
    if(ico<nico-1)
    {
    for(j=0;j<dr[ico+1];j++)
    {
     fprintf(fp6,"\n%f",C[ico][i][j]);
     for(iprime=i;iprime<dr[ico];iprime++)    /*i'=i ICI*/
     {
     fprintf(fp6,"\n%f",D[ico][i][iprime][j]);
     }/*iprime*/
    }/*j*/
    }/*if*/
   }/*i*/
/*----mise en fichier et affichage des recouvrements----*/
  fprintf(fp6,"\n");printf("\n");
  U=0.0;
  for(k=0;k<ncl;k++)
  {
   for(kprime=k;kprime<ncl;kprime++)
   {
   fprintf(fp6,"%2.6f ",Rkk[ico][k][kprime]);
   if(k==kprime){U=U+Rkk[ico][k][kprime];}else{U=U-
2.0*Rkk[ico][k][kprime];}
   printf("R[%d][%d][%d]=%2.6f
",ico,k,kprime,Rkk[ico][k][kprime]);
   }/*kprime*/
  }/*k*/
  printf("  U=%2.6f",U);
/*Correction des taux*/
/*   xyz=(U-3.99)*(U-3.99)/1000.0;
     bec1=bec1*xyz*10.0;bec2=bec2*xyz*10.00;
   betheta1=betheta1*xyz*10.0;betheta2=betheta2*xyz*10.0;
*/
/*     beC1=beC1*0.90;beC2=beC2*0.90;  */
/*------------------------------------------------------*/
  }/*ico*/
/*speciale*/   if(NCc!=0){avCOc=avCOc/(float)NCc;}
/*speciale*/   if(NCt!=0){avCOt=avCOt/(float)NCt;}
/*speciale*/   if(NCC!=0){avCOC=avCOC/(float)NCC;}
/*speciale*/   if(NCD!=0){avCOD=avCOD/(float)NCD;}
```

```
/*speciale*/   printf("\navCOc=%f  avCOt=%f  avCOC=%f
avCOD=%f",avCOc,avCOt,avCOC,avCOD);
/*speciale  getch();*/

 }/*test de passaf*/
 goto boucle;
hell:
 fclose(fp6);/*fichier de mesures*/
 printf("\n ORTHO_15.C termine!");
 getchar();
}/*fin du main*/
/*===================================================*/
float Sigmoide(float x,float c, float theta)/*sigmoide
centree avec seuil*/
{                 /*corrigee le 23 avril*/
 float num,den,expon;
 double y;
 y=(double)-c*(x-theta);
 if(y<-20.00){num=1.00;den=1.00;goto saute;}
 if(y>20.00){num=-1.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=1.0-expon;
 den=1.0+expon;
saute:
 return(num/den);
}/*fin de sigmoide*/
/*===================================================*/
float dSdx(float x,float c, float theta)/*del S/del x*/
{
 float num,den,expon;
 double y;

 y=(double)-c*(x-theta);
 if((y<-30.00) || (y>30.00)){num=0.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=2.0*c*expon;
 den=(1.0+expon)*(1.0+expon);
saute:
 return(num/den);
}/*fin dSdx*/
/*===================================================*/
float dSdc(float x,float c, float theta)/*del S/del c*/
{
 float num,den,expon;
 double y;

 y=(double)-c*(x-theta);
 if((y<-30.00) || (y>30.00)){num=0.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=2.0*(x-theta)*expon;
 den=(1.0+expon)*(1.0+expon);
saute:
 return(num/den);
}/*fin dSdc*/
/*====================================================*/
float dSdtheta(float x,float c, float theta)/*del S/del
theta*/
{
/* ATTENTION: dS/dtheta=-dS/dx et c'est bien ca*/
 float num,den,expon;
 double y;

 y=(double)-c*(x-theta);
 if((y<-30.00) || (y>30.00)){num=0.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=-2.0*c*expon;
 den=(1.0+expon)*(1.0+expon);
saute:
 return(num/den);
}/*fin dSdtheta*/
/*====================================================*/
/*RANDOM.SOU*/
float monrandom(float y)
{
 float pi;
 float co;

 pi=3.141592654;
 co=100.0*cos(pi*y/2.00);
 y=co-(int)co;
 return(y);
}/*monrandom*/
/*====================================================*/
float absf(float x)/*valeur absolue d'un point flotant*/
{
 if(x<0.0){return(-x);}
 else{return(x);}
}
/*====================================================*/
void journal(char *typp,char *nf)
{
 FILE *fpj;
 char *nomj;

 nomj="jou.jou";
 if((fpj=fopen(nomj,"r+t"))==NULL)
 {printf("\nPeut pas ouvrir le journal, MERDE!");goto elle;}
 fseek(fpj,-1L,2);
 fprintf(fpj," ");
 fclose(fpj);
 if((fpj=fopen(nomj,"a+"))==NULL)
 {
 printf("\nJournal impossible a ouvrir");
 getchar();goto elle;
 }

 if(*typp==*"J-M-A")
 {
 fprintf(fpj,"\n28-juillet-97");
 }
 else
 {
 fprintf(fpj,"\n  %s %s ",typp,nf);
 }
 fclose(fpj);

elle:
 ;
}
/*sous-routine journal*/
/*==============================================*/
```

A-VII-2 Measurement program

```
/* MESEP_15.C    Version TURBO C
   Creation: juin 97

      <<MEsure la SEParation des classes>>
Suite naturelle de ORTHO_15.C qui est le programme
d'apprentissage, celui-ci est le programme qui permet de
faire des observations sur l'apprentissage, evolution
performances. MESEP_15.C est egalement valable pour
ORTHO_12.C et ORTHO_15.C ATTENTION, en de nombreux endroits
le programme est limite a DEUX classes.
      La difference entre ORTHO_12.C et ORTHO_15.C c'est que
dans la version _15 il y a une option qui permet de faire un
fichier de classification: c'est le fichier-test qui est re-
ecrit avec, a la suite de la classe, la classe assignee par
le perceptron exact et la classe assignee par le protocole
comparatif. Ca peut servir a visualiser les coupures des
classes avec un programme comme TDR4D.C ou bien ca peut
servir a comparer des classifications en detail, par exemple
si le separateur bayesien (ou les supercuts dans les cas
d'astrophysique) fournissaient ces renseignements on pourait
savoir <<exactement>> les differences de performances des
differents classificateurs. /C'etait prevu dans la version
_12, c'est le fichier de classement, le nomfichier7, mais ca
n'a jamais ete implemente correctement dans la version _12.
Dans la version _15 on savait mieux ce que l'on voulait
garder. En fait, la construction d'un fichier de classement
est le travail de DETEC_xx.C. En le faisant ici, dans
MESEP_15.C on se donne la possibilite d'etudier le
classement a differents endroits de l'apprentissage. Plutot
que de re-ecrire un DETEC_12.C pour faire quelque fichier de
classement avec les Toy_Data, on va mettre un interrupteur
dans MESEP_15.C qui lui fear sauter toutes les passes
d'apprentissage et simplement produire un fichier de
classement a la fin. TDR4D.C fera les decomptes detailles
des classements, en plus de leur presentation graphique, de
plus on l'organisera pour 2, 3 ou 4 dimensions. Lorsqu'il y
aura plus de quatre dimensions, la partie graphique ne
traitera que les quatres premieres (ou les quatre choisies
par l'usager).*/
#include<stdio.h>  /*pour printf, scanf etc*/
#include<io.h>     /*pour les acces aux fichiers disque*/
#include<fcntl.h>  /*pour les valeurs O_WRONLY etc*/
#include<math.h>   /*pour SIN, SQRT,...des variables
dependantes*/
#include<dos.h>    /*pour la date du journal*/
#define NICO 3
#define NCL 2
```

```
#define DRMAX 5
#define NILS 40
float Sigmoide(float x,float c, float theta);/*sigmoide
centree avec seuil*/
void journal(char *typp,char *nf);
void saute_chaine(int nc);
FILE *fp1;
char chaine[100];
main()
{
  extern char chaine[100];
/*FICHIERS*/
  extern FILE *fp1;/*fichier de mesures provenant de
ORTHO_12.C*/
  char *nomfichier1;
  FILE *fp3;
  char *nomfichier3;/*fichier d'ensemble d'apprentissage
necessaire pour le perceptron de la derniere coucheou le
protocole comparatif*/
  FILE *fp4;
  char *nomfichier4;/*fichier d'architecture (fichier
d'entree)*/
  FILE *fp5;
  char *nomfichier5;/*fichier.TST (entree), pour le mesures
de generalisation*/
  FILE *fp6;
  char *nomfichier6;/*fichier de resultats*/
  FILE *fp7;
  char *nomfichier7;/*fichier de classement de l'ensemble
d'apprentissage*/
  char rep;
  char *ty,*nf;
/*ENSEMBLE D'APPRENTISSAGE*/
  int nils;/*nb total d'exemples*/
  int classe[NILS];/*pas d'indice, classe courante
seulement*/
  int np;/*nombre de parametres, nb d'entrees du reseau*/
  int ncl;/*nombre de classes*/
  int Nk[NCL],Nkt[NCL];
/*COMPTEURS*/
  int ils;/*compteurs d'exemples de l'ensemble
d'apprentissage*/
  int i,imax;/*cellules du <<premier>> niveau*/
  int j,jmax;/*cellules du <<second niveau>>*/
  int k,kprime;/*compteur d'exemples DANS <<la>> classe*/
  int c0,c1;/*compteurs de classes*/
  int ico;/*compteur de couches*/
  int inf;/*indice des fonctions non-lineaires*/
  int po,pr;/*post et pre synaptique*/
  int paserzerEA,paserzerTST,paserzerFR,premcroi;
```

```
/*STRUCTURE NEURONALE*/
   float z[NICO][DRMAX][NILS];
   int nico;/*nombre de couches*/
   int dr[NICO];/*nb de neurones de la [couche No]*/
   float c[NICO][DRMAX];/*constantes pour la sigmoide*/
   float theta[NICO][DRMAX];/*seuils pour la sigmoide*/
   float C[NICO][DRMAX][DRMAX];
   float D[NICO][DRMAX][DRMAX][DRMAX];/*couplages pour les
produits*/
   int cl;/*classe desiree*/
   char car;
   int ii,jj,iico,iprime,iiprime,pre,ib,nex,inb,nb_a_lire;
   float fxik;
   float Norm;
/*VARIABLES POUR LES TESTS ET LES MESURES*/
   int
sdEA[NCL][NCL],sdTST[NCL][NCL],sdFR[NCL][NCL];/*compteurs de
resultats sd[classe de l'ensemble d'apprentissage][signe de
la cellule de sortie] 0 (zero) = moins, 1 = plus*/
   int lx,ly,ily;/*colone et ligne pour des gotoxy()*/
   int flagf;/*0 aucune classe inscrite au fichier -> MODE DE
DECTECTION mais il faut avoir choisi un protocole de sortie
auparavant
              1 fichier a une seule classe (?)
              2 ou plus, fichier a plusieurs classes*/
   int flagnor;/*0: DETEC_12 n'a pas a normaliser les entrees
                1: DETEC_12 doit normaliser les entrees*/
   int post;
   float
RkkEA[NICO][NCL][NCL],UEA[NICO],SOMkfyEA[NICO][DRMAX][NCL];
   float
RkkFR[NCL][NCL],UFR,SOMkfyFR[DRMAX][NCL],Rkk[NICO][NCL][NCL]
;
   float zi,ci,ti,fact;
   float Cj[DRMAX];/*couplages de la derniere couche a la
sortie "Perceptron"*/
   int nilst,npt,nclt;/*nils,np et ncl pour le fichier de
tests*/
   float zt[NICO][DRMAX];/*etats des neurones pour les test*/
   float norm,norme[NCL],sortie,gain[NCL];
   int couche_choisie,clFR,ipasse;
   float bec1,bec2,betheta1,betheta2,beC1,beC2,beD1,beD2;
   char ca;
   float flaglisep;/*verifie si la couche choisie est
lineairement separable*/
   float x1,x2,x3,x4,y1,y2,y3,y4,T1,T2;
   int passelisep,iclmt,flag_classement;
/*--initialisations de toutes les variables indexees---*/
   for(k=0;k<NCL;k++){Nk[k]=0;}
   for(ils=0;ils<NILS;ils++){classe[ils]=0;}
   for(ico=0;ico<NICO;ico++)
   {
    dr[ico]=0;
    for(i=0;i<DRMAX;i++)
    {
     c[ico][i]=1.00;theta[ico][i]=0.00;
     for(ils=0;ils<NILS;ils++)
     {
      z[ico][i][ils]=0.00;
     }/*ils*/
     for(j=0;j<DRMAX;j++)
     {
      C[ico][i][j]=0.00;
      for(iprime=0;iprime<DRMAX;iprime++)
      {
       D[ico][i][iprime][j]=0.00;
      }/*iprime*/
     }/*j*/
     for(k=0;k<NCL;k++)
     {
      SOMkfyEA[ico][i][k]=0.00;SOMkfyFR[i][k]=0.00;
     }/*k*/
    }/*i*/
    for(k=0;k<NCL;k++)
    {
     for(kprime=0;kprime<NCL;kprime++)
     {
Rkk[ico][k][kprime]=0.00;RkkEA[ico][k][kprime]=0.00;RkkFR[k]
[kprime]=0.00;
     }/*kprime*/
    }/*k*/
    UEA[ico]=0.00;
   }/*ico*/
   clrscr();
/*---PARAMETRES A FIXER AVANT L'EXECUTION---*/
/*Ensembles d'apprentissage:
  Ensembles de tests:
  Ensembles de detection:
  Ensembles de mesures:*/
  nb_a_lire=120;
  iclmt=nb_a_lire-1;
  /*pour avoir le classement a la fin de l'apprentissage,
mais on peut l'avoir ailleurs en changeant la valeur de
iclmt*/
  nomfichier1="C:\\TC\\ORTHO\\XOR\\XOLIM.M21";  /*Fichier de
mesures*/
  nomfichier3="C:\\TC\\ORTHO\\XOR\\XOR.DAT"; /*Ensemble
d'apprentissage*/
```

```
  nomfichier5="C:\\TC\\ORTHO\\XOR\\XOR.TST";  /*Fichier de
Test ou de detection, pour les TOY_DATA il y a ASD.TST et
ASD.TSU*/
  nomfichier6="C:\\TC\\ORTHO\\XOR\\XOLIM21.RES";  /*Fichier
des resultats*/
  nomfichier7="C:\\TC\\ORTHO\\XOR\\XOLIM21.CLA";
/*classement de l'EA C'est le programme MESGRA.C qui fait le
fichier normalise pour EXCEL*/
  dr[0]=2;dr[1]=3;dr[2]=2;
/*TRES IMPORTANT DE METTRE LES NOMBRES DE NEURONES PAR
COUCHE COMME DANS ORTHO_12.C et ORTHO_15.C ET LE FICHIER DE
MESURES QUE L'ON VA UTILISER. Ca serait trop chiant de le
faire lire dans le fichier de mesures  */
  cl=1;/*si c'est un fichier de classeq, inscrire ICI la
classe elle ne sera PAS lue dans le fichier*/
  flagnor=0;/*ATTENTION c'est pas le meme <<flag>> que dans
ORTHO_15,
              ici c'est un entier
              0 (zero) pas de normalisation
              1 Il y a normalisation (des entrees) et c'est
le 1 qu'on teste*/
  couche_choisie=2;
  /*choisir ici l'indice C de la couche ou le protocole de
sortie sera branche*/
  flaglisep=0;/*1: verifie   0: ne verifie pas */
              /*valable pour deux cellules et deux
configurations par classe
               seulement 16-Aout-96*/
  flag_classement=0;/*0:tout+classement a la fin, 1:
classement seulement
    il y a un bogue avec flag_classement=1*/
/*NE PAS OUBLIER D'ALLER INSCRIRE LE BON NOMBRE DE CHAINES A
SAUTER (ligne 300)*/
/*-------Ecriture au journal-----------*/
  ty="J-M-A";nf="...";journal(ty,nf);
  ty="Programme:";nf="MESEP_15.C";journal(ty,nf);
  ty="Ensemble d'apprentissage:";journal(ty,nomfichier3);
  ty="Fichier de mesures:";journal(ty,nomfichier1);
  ty="Donnees d'entree:";journal(ty,nomfichier5);
  ty="Fichier de resultats:";journal(ty,nomfichier6);
  ty="Fichier de Classement:";journal(ty,nomfichier7);
/*ouvrir le fichier 6
    IL NE FAUT AS TOUCHER A fp6 SI flag_classement VAUT 1*/
  if(flag_classement==0)
  {
   if((fp6=fopen(nomfichier6,"w"))==NULL)
   {
    printf("\nImpossible d'ouvrir le fichier de resultats");
    goto hell;
   }
   fprintf(fp6,"Fichier de resultats: %s",nomfichier6);
   fprintf(fp6,"\ndu programme MESEP_12.C pour le Fichier de
Mesures: %s",nomfichier1);
   fprintf(fp6,"\navec l'Ensemble d'Apprentissage:
%s",nomfichier3);
   fprintf(fp6,"\net le Fichier de Test: %s",nomfichier5);
   fprintf(fp6,"\ncouche_choisie: %d",couche_choisie);
  }/*flag_classement==0*/
/*AFF*/clrscr();
       gotoxy(1,1);printf("MESEP_12.C, Mesures:
%s",nomfichier1);
       gotoxy(1,2);printf("Ensemble d'apprentissage:
%s",nomfichier3);
/*AFF*/gotoxy(1,3);printf("Fichier de test ou de detection:
%s",nomfichier5);
/*------ouvrir le fichier de classement--------*/
  if((fp7=fopen(nomfichier7,"w"))==NULL)/*ouvrir le fichier
de classement*/
  {
   printf("\nImpossible d'ouvrir le fichier de classement");
   goto hell;
  }
/*----------Lecture de l'ensemble d'apprentissage--------*/
  if((fp3=fopen(nomfichier3,"r"))==NULL)
  {printf("\nImpossible d'ouvrir l'ensemble
d'apprentissage");goto hell;}
  fscanf(fp3,"%d %d %d",&nils,&np,&ncl);
/*DEBprintf("nils=%d np=%d ncl=%d",nils,np,ncl);*/
  for(i=0;i<ncl;i++){Nk[i]=0;}
  for(ils=0;ils<nils;ils++)
  {
   Norm=0.0;
   for(i=0;i<dr[0];i++)
   {
    fscanf(fp3,"%f",&z[0][i][ils]);
/*DEBprintf("\nz[0][%d][%d]=%f",i,ils,z[0][i][ils]);*/
    Norm=Norm+z[0][i][ils]*z[0][i][ils];
   }/*i*/
   Norm=(float)sqrt((double)Norm);
   if(flagnor==1)
   {
    for(i=0;0<dr[0];i++)
    {
     z[0][i][ils]=z[0][i][ils]/Norm;
    }/*i*/
   }/*if flagnor*/
   fscanf(fp3,"%d",&k);
/*DEBprintf("\nclasse lue: %d",k);*/
   classe[ils]=k;
   Nk[k]++;
```

```
/*DEBprintf("\nils=%d  classe=%d
NK[classe]=%d",ils,classe[ils],Nk[k]);*/
  }/*ils*/
  fclose(fp3);/*fini de lire le fichier d'apprentissage*/

/*DEBprintf("\n-Ligne 247----Lecture du fichier de mesures--
-------------");*/
  if((fp1=fopen(nomfichier1,"r"))==NULL)
  {
   printf("\nImpossible d'ouvrir le fichier de mesures");
   goto hell;
  }
/*-----------Lire l'entete du fichier de mesures------------
--*/
  saute_chaine(7);fscanf(fp1,"%d",&nex);
  saute_chaine(1);fscanf(fp1,"%d",&np);
  saute_chaine(1);fscanf(fp1,"%d",&ncl);
  saute_chaine(1);fscanf(fp1,"%d",&nico);
/*DEB  printf("\nligne 263 nex=%d np=%d ncl=%d
nico=%d\n",nex,np,ncl,nico);getch();*/
  for(i=0;i<nico;i++)
  {
   saute_chaine(2);fscanf(fp1,"%d",&dr[i]);saute_chaine(1);
/*DEB   printf("  dr[%d]=%d ",i,dr[i]);getch();*/
  }
/*--------Sauter les valeurs initiales-----------*/
/*ecrire le bon nombre de chaines a sauter
    59 XOR a deux couches seulement      2-2
    76 pour le XOR 2-2-2
    92 pour l'architecture 2-3-2
    93 pour le XOR 2-4-1
    130 pour XOR 2-5-2
    152 pour la serie ASD (toy data) avec l'architecture 4 4
4 ORTHO_12.C i'=i+1
    184 pour la serie ASD (toy data) avec l'architecture 4 4
4 ORTHO_12.C i'=i
    184 -------------------------------------------------
(ORTHO_15.C)???
    198 pour la serie ASD (toy data) avec l'architecture 4 6
3
    385  pour la serie ASD (toy data) avec l'architecture 4
5 5 5 5
    491 pour la serie PAIR10PO avec l'architecture 7 7 7
    273 pour la serie PAIR10PO avec l'architecture 7 7
    186 pour ASDASD avec ORTHO_15.C 4-4-4 i'=i && i'!=i
*/
  saute_chaine(92);/*saute toutes les valeurs initiales*/
/*DEBgetch();*/
 saute_chaine(2);/*pour les pointilles*/
/*-----------boucle des passes (LECTURE)---------------*/
```

```
 paserzerEA=-1;paserzerTST=-1;paserzerFR=-1;premcroi=-
1;passelisep=-1;
 for(inb=0;inb<nb_a_lire;inb++)
 {
/*DEB  clrscr();*/
  fscanf(fp1,"%d",&ipasse);
/*AFF*/  gotoxy(60,1);printf(" passe No: %d  ",ipasse);
         gotoxy(60,13);printf(" Er(EA)=0: %d  ",paserzerEA);
         gotoxy(60,14);printf("Er(TST)=0: %d  ",paserzerTST);
         gotoxy(60,15);printf(" Er(FR)=0: %d  ",paserzerFR);
/*AFF*/  gotoxy(60,16);printf("    Uy<Uz: %d  ",premcroi);
  fscanf(fp1,"%f %f",&bec1,&bec2);
  fscanf(fp1,"%f %f",&betheta1,&betheta2);
  fscanf(fp1,"%f %f",&beC1,&beC2);
  fscanf(fp1,"%f %f",&beD1,&beD2);
/*DEB printf("\n ligne 246 bec1=%f  beD2=%f",bec1,beD2);*/
  for(ib=0;ib<nico;ib++)
  {
   fscanf(fp1,"%d",&ico);
   for(ii=0;ii<dr[ico];ii++)
   {
    fscanf(fp1,"%d",&i);
    fscanf(fp1,"%f %f",&c[ico][i],&theta[ico][i]);
/*DEBprintf("\nc[%d][%d]=%f
theta[%d][%d]=%f",ico,i,c[ico][i],ico,i,theta[ico][i]);getch
();*/
    if(ib<nico-1)
    {
     for(j=0;j<dr[ico+1];j++)
     {
      fscanf(fp1,"%f",&C[ico][i][j]);
      for(iprime=i;iprime<dr[ico];iprime++)
      {
       fscanf(fp1,"%f",&D[ico][i][iprime][j]);
      }/*iprime*/
     }/*j*/
    }/*if ib=pre*/
   }/*ii*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k;kprime<ncl;kprime++)
    {
     fscanf(fp1,"%f",&Rkk[ico][k][kprime]);
/*DEBgotoxy(6,20);printf("\n
Rkk[%d][%d][%d]=%f",ico,k,kprime,Rkk[ico][k][kprime]);getch(
);*/
    }/*kprime*/
   }/*k*/
  }/*ib*/
```

```
if((flag_classement==0)||((flag_classement==1)&&(inb==iclmt)
))
  {
   for(ico=0;ico<nico;ico++)
   {
    for(j=0;j<dr[ico];j++)
    {
     for(k=0;k<ncl;k++)
     {
      SOMkfyEA[ico][j][k]=0.00;
     }/*k*/
    }/*j*/
    UEA[ico]=0.00;
   }/*ico*/
/*DEBprintf("\n--calcul des etats des neurones et des
recouvrements-----");*/
/*On a besoin des vecteurs (de classe) moyens pour les Cj du
<<Perceptron terminal>>, et on besoin des etats des neurones
pour calculer les Rkk (ou les fonctions U) de r,f,rence pour
le protocole comparatif*/
   for(ils=0;ils<nils;ils++)
   {
    cl=classe[ils];
    for(pre=0;pre<nico-1;pre++)
    {
     post=pre+1;imax=dr[pre];jmax=dr[post];
     for(j=0;j<jmax;j++)
     {
      z[post][j][ils]=0.00;
      for(i=0;i<imax;i++)
      {
       zi=z[pre][i][ils];ci=c[pre][i];ti=theta[pre][i];
       fxik=Sigmoide(zi,ci,ti);
       z[post][j][ils]=z[post][j][ils]+C[pre][i][j]*fxik;
       if(i<imax-1)
       {
        for(iprime=i;iprime<imax;iprime++)
        {
zi=z[pre][iprime][ils];ci=c[pre][iprime];ti=theta[pre][iprim
e];
z[post][j][ils]=z[post][j][ils]+D[pre][i][iprime][j]*fxik*
Sigmoide(zi,ci,ti);
        }/*iprime*/
       }/*if il y a de la place pour iprime*/
      }/*i*/
      zi=z[post][j][ils];ci=c[post][j];ti=theta[post][j];
SOMkfyEA[post][j][cl]=SOMkfyEA[post][j][cl]+Sigmoide(zi,ci,t
i);
/*on ne calcule pas du tout les SOMk pour la couche d'entree
(OK)*/
```

```
       }/*j*/
      }/*pre*/
     }/*ils*/
 /*-------------Divisions par Nk-------------------*/
     for(ico=1;ico<nico;ico++)
     {
      jmax=dr[ico];
      for(j=0;j<jmax;j++)
      {
       for(k=0;k<ncl;k++)
       {
        SOMkfyEA[ico][j][k]=SOMkfyEA[ico][j][k]/(float)Nk[k];
       }/*k*/
      }/*j*/
     }/*ico*/
 /*-calcul des Rkk et des U pour l'ensemble d'apprentissage*/
     for(k=0;k<ncl;k++)
     {
      for(kprime=k;kprime<ncl;kprime++)
      {
       for(ico=0;ico<nico;ico++)
       {
        RkkEA[ico][k][kprime]=0.00;
        for(j=0;j<dr[ico];j++)
        {
RkkEA[ico][k][kprime]=RkkEA[ico][k][kprime]+SOMkfyEA[ico][j]
[k]*SOMkfyEA[ico][j][kprime];
        }/*j*/
 RkkEA[ico][k][kprime]=RkkEA[ico][k][kprime]/(float)dr[ico];
        fact=-2.00;if(k==kprime){fact=1.00;}
        UEA[ico]=UEA[ico]+fact*RkkEA[ico][k][kprime];
       }/*ico*/
      }/*kprime*/
     }/*k*/
/*On teste le croisement des fonctions U pour les deux
dernieres couches*/
    if((UEA[nico-2]<UEA[nico-1])&&(premcroi==-
1)){premcroi=ipasse;}
/*---COUPLAGES pour la cellule de sortie unique-------*/
/*----Perceptron pour deux classes seulement--------*/
/*On met les vecteurs Cj dans la direction de la difference
entre les deux vecteurs moyens de classe. Si les vecteurs
moyens sont normalises, cette direction est aussi celle de
la perpendiculaire a la bissectrice de l'angle entre les
deux vecteurs moyens.*/
    ico=couche_choisie;
    jmax=dr[ico];
    for(k=0;k<ncl;k++)
    {
     norme[k]=0.00;
     for(j=0;j<jmax;j++)
     {
  norme[k]=norme[k]+SOMkfyEA[ico][j][k]*SOMkfyEA[ico][j][k];
     }/*j*/
     norme[k]=(float)sqrt((double)norme[k]);
    }/*k*/
    for(j=0;j<jmax;j++)
    {
     Cj[j]=(SOMkfyEA[nico-1][j][0]/norme[0])-(SOMkfyEA[nico-
1][j][1]/norme[1]);
    }/*i*/
/*AFF*/for(ico=1;ico<nico;ico++)
        {
         for(k=0;k<ncl;k++)
         {
          for(kprime=k;kprime<ncl;kprime++)
          {
           ly=4+ico;lx=1;
           if(k+kprime==1){lx=22;}
           if(k+kprime==2){lx=44;}
           gotoxy(lx,ly);
printf("R[%d][%d][%d]=%2.6f",ico,k,kprime,RkkEA[ico][k][kpri
me]);
          }/*kprime*/
         }/*k*/
         printf("  W=%2.6f",UEA[ico]);
        }/*ico*/
/*AFF*/ily=ly;
/*initialise les compteurs des sorties*/
   for(i=0;i<ncl;i++)
   {
    for(j=0;j<2;j++) /*2 c'est le nombre de signes de la
sortie*/
    {
     sdEA[i][j]=0;/*sd[classe EA][signe de la sortie du
perceptron]*/
     sdTST[i][j]=0;
     sdFR[i][j]=0;
    }/*j pour le signe des etats de la sortie*/
    Nkt[i]=0;
   }/*i pour les classes*/
/*-Protocole de sortie pour l'ensemble d'apprentissage--*/
   ico=couche_choisie;
   jmax=dr[ico];
/*Perceptron, solution exacte*/
   if(ncl>2)
   {
    printf("\nle protocole du perceptron n'est pas
(actuellement)");
```

```
    printf("\nadapte a un probleme a plus de deux
classes");goto hell;
   }
   for(ils=0;ils<nils;ils++)/*roule sur l'ensemble
d'apprentissage*/
   {
    sortie=0.00;cl=classe[ils];
    for(j=0;j<jmax;j++)/*roule sur la couche choisie ou le
perceptron est connecte*/
    {

sortie=sortie+Cj[j]*Sigmoide(z[ico][j][ils],c[ico][j],theta[
ico][j]);
    }/*j*/
    if(sortie>0.00){sdEA[cl][1]++;}
    else{sdEA[cl][0]++;}
/*C'est probablement ici qu'il faut examiner les
configurations mal classees*/
   }/*ils*/
/*AFF*/ly=ily+2;gotoxy(1,ly);printf("Ensemble
d'apprentissage ");
       ly=ly+1;gotoxy(1,ly);printf("Protocole de sortie:
Perceptron connecte a la couche %d",ico);
       ly=ly+1;gotoxy(1,ly);printf("classe 0  sortie<0:%d
sortie>0:%d    ",sdEA[0][0],sdEA[0][1]);
       ly=ly+1;gotoxy(1,ly);printf("classe 1  sortie<0:%d
sortie>0:%d    ",sdEA[1][0],sdEA[1][1]);
/*AFF*/ily=ly;
/*--test de la sepration lineaire de la couche choisie--*/
/*valide seulement pour une couche de deux cellules, avec
deux classes et
seulement deux exemples par classe, pour le XOR par exemple
ATTENTION, pour que le test soit valable il faut que les
exemples soient dans
les classes et DANS L'ORDRE 0110*/
   if(flaglisep==1)
   {
    passelisep=-1;
    x1=Sigmoide(z[ico][0][0],c[ico][0],theta[ico][0]);
    x2=Sigmoide(z[ico][0][3],c[ico][0],theta[ico][0]);
    x3=Sigmoide(z[ico][0][1],c[ico][0],theta[ico][0]);
    x4=Sigmoide(z[ico][0][2],c[ico][0],theta[ico][0]);
    y1=Sigmoide(z[ico][1][0],c[ico][1],theta[ico][1]);
    y2=Sigmoide(z[ico][1][3],c[ico][1],theta[ico][1]);
    y3=Sigmoide(z[ico][1][1],c[ico][1],theta[ico][1]);
    y4=Sigmoide(z[ico][1][2],c[ico][1],theta[ico][1]);
    T1=y3-((y2-y1)/(x2-x1))*x3-(y1*x2-y2*x1)/(x2-x1);
    T2=y4-((y2-y1)/(x2-x1))*x4-(y1*x2-y2*x1)/(x2-x1);
/*DEBily=ily+1;ly=ily;gotoxy(5,ly);printf("T1*T2=%f
",T1*T2);*/
```

```
     if((T1*T2)>=0)
     {/*la separation lineaire est atteinte*/
      passelisep=ipasse;
      flaglisep=0;/*on ne verifie plus la separation lineaire
apres l'avoir eue*/
     }
    }/*if test de la separation lineaire*/
/*---detection de l'erreur zero, perceptron et ensemble
d'apprentissage---*/
    if(paserzerEA==-1)
    {

if((sdEA[0][0]*sdEA[0][1]==0)&&(sdEA[1][0]*sdEA[1][1]==0)&&(
sdEA[0][0]*sdEA[1][0]==0)&&(sdEA[0][1]*sdEA[1][1]==0))
     {
      paserzerEA=ipasse;
     }/*on a trouve l'erreur-zero pour l'ensemble
d'apprentissage*/
    }/*il faut chercher la premiere occurence de l'erreur
zero*/
/*===Lecture du fichier de test ou de detection===*/
    if((fp5=fopen(nomfichier5,"r"))==NULL)  /*ouvrir le
fichier 5*/
    {
     printf("\nImpossible d'ouvrir le fichier de test ou de
detection");
     goto hell;
    }
    fscanf(fp5,"%d %d %d",&nilst,&npt,&nclt);   /*Lecture de
l'entete*/
/*ecriture de l'entete du fichier de classement, apres la
classe on mettra la classe du perceptron puis celle du
protocole comparatif*/
   if(inb==iclmt){fprintf(fp7,"%d %d %d",nilst,npt,nclt);}
/*Cette lecture du fichier test change nils, np, ncl que
l'on a lu pour l'ensemble d'apprentissage, il faudrait
distinguer les deux series de parametres */
    if(npt!=np)
    {
     printf("\nle nombre de parametres de l'ensemble
d'apprentissage et");
     printf("\ndu fichier test ne sont pas compatibles");
     goto hell;
    }
    if(nclt>ncl)
    {
     printf("\nle nombre de classes de l'ensemble
d'apprentissage et");
     printf("\ndu fichier test ne sont pas compatibles");
     goto hell;
   }
   flagf=nclt;    /*0: fichier de detection */
/*UN FICHIER DE DETECTION DOIT ABSOLUMENT AVOIR UN ZERO A
L'ENDROIT DU NOMBRE DE CLASSES*/
/*---------------ENTREE DES DONNEES----------------------*/
/*On lis les donnees et on les traites au fur et a mesure,
on ne les
gardes pas en memoire comme dans les programmes
d'apprentissage parcequ'on
ne les traites qu'une seule fois*/
   for(ils=0;ils<nilst;ils++)
   {
    if(inb==iclmt){fprintf(fp7,"\n");}
    for(i=0;i<dr[0];i++)
    {
     fscanf(fp5,"%f",&zt[0][i]);
     if(inb==iclmt){fprintf(fp7,"%f ",zt[0][i]);}
    }/*i*/
    if(flagf>0)/*les classes des entrees sont dans le
fichier*/
    {
     fscanf(fp5,"%d",&cl);
     if(inb==iclmt){fprintf(fp7,"%d ",cl);}
     Nkt[cl]++;
    }
    if(flagnor==1)/*il faut normaliser les entrees*/
    {
     norm=0.0;
     for(i=0;i<dr[0];i++)
     {
      norm=norm+zt[0][i]*zt[0][i];
     }/*i*/
     norm=(float)sqrt((double)norm);
     for(i=0;i<dr[0];i++)
     {
      zt[0][i]=zt[0][i]/norm;
     }/*i*/
    }/*flagnor*/
/*------------calcul des etats des neurones---------------*/
    for(post=1;post<nico;post++)    /*post OK*/
    {
     jmax=dr[post];
     pre=post-1;imax=dr[pre];    /*pre=pre*/
     for(j=0;j<jmax;j++)
     {
      zt[post][j]=0.0;
      for(i=0;i<imax;i++)
      {
       fxik=Sigmoide(zt[pre][i],c[pre][i],theta[pre][i]);
       zt[post][j]=zt[post][j]+C[pre][i][j]*fxik;
```

```
        for(iprime=i;iprime<imax;iprime++)
        {
         zt[post][j]=zt[post][j]+D[pre][i][iprime][j]*fxik*
Sigmoide(zt[pre][iprime],c[pre][iprime],theta[pre][iprime]);
        }/*iprime*/
       }/*i*/
      }/*j*/
     }/*post*/
/*------Protocole de sortie pour l'ensemble test--------*/
    ico=couche_choisie;
    jmax=dr[ico];
/*Perceptron, solution exacte*/
    if(ncl>2)
    {
     printf("\nle protocole du perceptron n'est pas
(actuellemment)");
     printf("\nadapte a un probleme a plus de deux
classes");goto hell;
    }
    sortie=0.00;
    for(j=0;j<jmax;j++)/*roule sur la couche choisie pour le
perceptron*/
    {
sortie=sortie+Cj[j]*Sigmoide(zt[ico][j],c[ico][j],theta[ico]
[j]);
    }/*j*/
    if(flagf>0)
    {
     if(sortie>0.00)
     {
      sdTST[cl][1]++;
      if(inb==iclmt){fprintf(fp7,"0 ");}
                  /*la sortie du perceptron peut etre a
l'envers*/
     }
     else
     {
      sdTST[cl][0]++;
      if(inb==iclmt){fprintf(fp7,"1 ");}
                  /*la sortie du perceptron peut etre a
l'envers*/
     }
    }/*si les classes sont definies dans le fichier*/
    else /*les classes ne sont pas inscrites au fichier*/
    {
     if(sortie>0.00)
     {
      sdTST[0][1]++;
      if(inb==iclmt){fprintf(fp7,"0 ");}
    /*la sortie du perceptron peut etre a l'envers*/
```

```
      }
      else
      {
       sdTST[0][0]++;
       if(inb==iclmt){fprintf(fp7,"1 ");}
     /*la sortie du perceptron peut etre a l'envers*/
      }
     }/*si les classes NE sont PAS definies dans le fichier*/
/*------protocole comparatif pour l'ensemble test---------*/
/*Description
  On ajoute l'exemple du test ou de la detection a
l'ensemble d'apprentissage dans une classe, on calcule les
Rkk (ou la fonction U); ensuite on met l'exemple dans une
autre classe on recalcule les Rkk et ainsi de suite jusqu'a
epuisement des classes. En comparant les differents Rkk
obtenus (par exemple, par le Rkk qui a recu la plus grande
majoration) on attribue une classe a l'exemple en question.
  (Je ne vois pas comment utiliser le Rkk de classe a classe
pour le moment) Comme on a garde les SOMkfyEA, on a pas
besoin de rebalayer l'ensemble d'apprentissage, on a
simplement a reprendre la sommation sur k en ajoutant
l'exemple courant une fois dans une classe, une fois dans
une autre... et conserver les differents Rkk obtenus.*/
/*initialiser pour chaque nouvelle configuration*/
    for(k=0;k<ncl;k++)
    {
     for(kprime=0;kprime<ncl;kprime++)
     {
      RkkFR[k][kprime]=0.00;
     }/*kprime*/
    }/*k*/
    UFR=0.00;
/*calcul des SOMk*/
    for(k=0;k<ncl;k++)
    {
     for(j=0;j<jmax;j++)
     {
SOMkfyFR[j][k]=(SOMkfyEA[ico][j][k]*(float)Nk[k])+Sigmoide(z
t[ico][j],c[ico][j],theta[ico][j]);
     }/*j*/
    }/*k*/
/*Division par (Nk+1)*/
    for(j=0;j<jmax;j++)
    {
     for(k=0;k<ncl;k++)
     {
      SOMkfyFR[j][k]=SOMkfyFR[j][k]/((float)(Nk[k]+1));
     }/*k*/
    }/*j*/
```

```
/*calcul des Rkk et U avec la configuration courante
ajoutee*/
    for(k=0;k<ncl;k++)
    {
     for(kprime=k;kprime<ncl;kprime++)
     {
      for(j=0;j<jmax;j++)
      {
RkkFR[k][kprime]=RkkFR[k][kprime]+SOMkfyFR[j][k]*SOMkfyFR[j]
[kprime];
      }/*j*/
      RkkFR[k][kprime]=RkkFR[k][kprime]/(float)jmax;
      fact=2.00;if(k==kprime){fact=1.00;}
      UFR=UFR+fact*RkkFR[k][kprime];
     }/*kprime*/
    }/*k*/
/*Gains des classes*/
    for(k=0;k<ncl;k++)
    {
     gain[k]=RkkFR[k][k]-RkkEA[ico][k][k];
    }/*k*/
    for(k=0;k<ncl;k++)
    {
     for(kprime=k+1;kprime<ncl;kprime++)
     {
      clFR=k;
      if(gain[k]<gain[kprime]){clFR=kprime;}
     }/*kprime*/
    }/*k*/
    if(inb==iclmt){fprintf(fp7,"%d ",clFR);}
    if(flagf>0)
    {
     sdFR[cl][clFR]++;
    }/*si les classes sont definies dans le fichier*/
    else
    {
     sdFR[0][clFR]++;
    }/*si les classes NE sont PAS definies dans le fichier*/
   }/*ils*/
/*---detection de l'erreur zero, perceptron et cpmparaison,
pour l'ensemble Test---*/
   if(paserzerTST==-1)
   {

if((sdTST[0][0]*sdTST[0][1]==0)&&(sdTST[1][0]*sdTST[1][1]==0
)&&(sdTST[0][0]*sdTST[1][0]==0)&&(sdTST[0][1]*sdTST[1][1]==0
))
    {
     paserzerTST=ipasse;
    }/*on a trouve l'erreur-zero pour l'ensemble test avec
le preceptron exact*/
   }/*il faut chercher la premiere occurence de l'erreur
zero*/
   if(paserzerFR==-1)
   {
if((sdFR[0][0]*sdFR[0][1]==0)&&(sdFR[1][0]*sdFR[1][1]==0)&&(
sdFR[0][0]*sdFR[1][0]==0)&&(sdFR[0][1]*sdFR[1][1]==0))
    {
     paserzerFR=ipasse;
    }/*on a trouve l'erreur-zero pour l'ensemble test avec
le protocole comparatif*/
   }/*il faut chercher la premiere occurence de l'erreur
zero*/
/*-----------Affichage et mise en fichier----------------*/
/*AFF*/ly=ily+1;gotoxy(1,ly);printf("Separation lineaire,
passe: %d   ",passelisep);
       ly=ily+3;gotoxy(1,ly);printf("Ensemble TEST");
       if(flagf>0)
       {
        ly=ly+1;gotoxy(1,ly);printf("classe 0: %d ",Nkt[0]);
        gotoxy(16,ly);printf("classe 1: %d ",Nkt[1]);
        ly=ly+1;gotoxy(1,ly);printf("Protocole de sortie:
Perceptron connecte a la couche %d",ico);
        ly=ly+1;gotoxy(1,ly);printf("classe 0  sortie<0:%d
sortie>0:%d    ",sdTST[0][0],sdTST[0][1]);
        ly=ly+1;gotoxy(1,ly);printf("classe 1  sortie<0:%d
sortie>0:%d    ",sdTST[1][0],sdTST[1][1]);
       }
       else
       {
        ly=ly+1;gotoxy(1,ly);printf("Protocole de sortie:
Perceptron connecte a la couche %d",ico);
        ly=ly+1;gotoxy(1,ly);printf(" sortie<0:%d
sortie>0:%d    ",sdTST[0][0],sdTST[0][1]);
       }
       ly=ly+1;gotoxy(1,ly);printf("protocole comparatif
pour la couche: %d",ico);
       if(flagf>0)
       {
        ly=ly+1;gotoxy(1,ly);printf("classe 0  PF 0: %d
PF 1: %d    ",sdFR[0][0],sdFR[0][1]);
        ly=ly+1;gotoxy(1,ly);printf("classe 1  PF 0: %d
PF 1: %d    ",sdFR[1][0],sdFR[1][1]);
       }
       else
       {
        ly=ly+1;gotoxy(1,ly);printf(" sortie<0:%d
sortie>0:%d    ",sdFR[0][0],sdTST[0][1]);
/*AFF*/}
```

```
    saute_chaine(1);/*pour les pointilles entre les passes*/
    fclose(fp5);/*fichier de test ou de detection*/
    if(flag_classement==0)/*il faut faire encore les test
parcequ'on passe ici dans deux conditions*/
    {
     fprintf(fp6,"\n%d %d %d %d %d
",ipasse,sdEA[0][0],sdEA[0][1],sdEA[1][0],sdEA[1][1]);
     fprintf(fp6,"%d %d %d %d
",sdTST[0][0],sdTST[0][1],sdTST[1][0],sdTST[1][1]);
     fprintf(fp6,"%d %d %d %d
",sdFR[0][0],sdFR[0][1],sdFR[1][0],sdFR[1][1]);
     fprintf(fp6,"%f %f ",UEA[1],UEA[2]);
    }/*flag_classement, 1 condition*/
   }/*fin du test de flag_classement, 2 conditions*/
  }/*inb*/
  fclose(fp1);/*fichier de mesures*/
  if(flag_classement==0)
  {
   fprintf(fp6,"\nERREUR-ZERO");
   fprintf(fp6,"\n Er(EA)=0: %d  (Ens. d'app. Percep.
exact)",paserzerEA);
   fprintf(fp6,"\nLa separation lineaire est atteinte a la
passe: %d",passelisep);
   fprintf(fp6,"\nEr(TST)=0: %d  (Ens. Test Percep.
exact)",paserzerTST);
   fprintf(fp6,"\n Er(FR)=0: %d  (Ens. Test Proto.
comparaison)",paserzerFR);
   fprintf(fp6,"\n    Uy<Uz: %d  (1ier croisement des fnct
U)",premcroi);
  }/*flag_classement, 1 condition*/
 fclose(fp7);/*fichier de classement configuration par
configuration*/
hell:
  if(flag_classement==0){fclose(fp6);/*fichier de
resultats*/}/*1 condition*/
  printf("\nMESEP_15.C termine!");
  getch();
}/*fin du main*/
/*===============================================*/
float Sigmoide(float x,float c, float theta)/*sigmoide
centree avec seuil*/
{                 /*Sigmoide corrigee*/
 float num,den,expon;
 double y;

 y=(double)-c*(x-theta);
 if(y<-20.00){num=1.00;den=1.00;goto saute;}
 if(y>20.00){num=-1.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=1.0-expon;
 den=1.0+expon;
saute:
 return(num/den);
}/*fin de sigmoide*/
/*====================================================*/
void saute_chaine(int nc)
{
 extern FILE *fp1;
 extern char chaine[];
 int i;
 for(i=0;i<nc;i++)
 {
  fscanf(fp1,"%s",&chaine);
/*DEB printf("\nsaut_chai. %d %s",i+1,chaine);getch();*/
 }
}/*fin de saute_chaine*/
/*====================================================*/
void journal(char *typp,char *nf)
{
(Same as in the learning program)
}/*sous-routine journal*/
/*====================================================*/
```

A-VII-3 Detection program

```
/* DETEC_15.C
           Creation: ?-94
       derniere revision: 27-juillet-97
Suite de ORTHO_15.C qui est le programme d'apprentissage,
celui-ci est le programme de mesure de classement, il est au
meme niveau que MESEP_15.C, il mesure les classements avec
la derniere architecture d'un fichier de mesures, et rien
d'autre.
Lis un fichier d'architecture construit par ORTHO_15.C, lis
un fichier de donnees .DAT ou .TST et determine les sorties
du reseau pour les entrees prises dans le fichier de test
On prevoit deux modes de fonctionnement:
- un mode test, ou les sorties voulues sont dans le fichier
de donnees. Cette utilisation revient maintenant a
MESEP_15.C!
- un mode de detection, ou le programme donne la classe de
l'evenement a l'entree du reseau de neurones. DETEC_15.C
fait un fichier de classement qui peut passer dans TDR4D.C
ou dans l'analyseur de classement /pas encore ecrit/ au
moment d'ecrire ces lignes. Detec_15.c est maintenant muni
d'un nouvel appareil, il peut ecrire les etats des neurones
d'une couche choisie dans un fichier qui poura alors servir
d'ensemble d'apprentissage pour la couche suivante (dans un
contexte d'apprentissage d'une couche a la fois) Ce qui
manque c'est un programme de detection qui lirait les
fichier de mesures d'apprentissage des differentes couches
successives de maniere a constituer le reseau de toutes les
couches apprises successivement et d'effectuer les mesures
de detection habituelles.*/
#include<stdio.h>  /*pour printf, scanf etc*/
#include<stdlib.h> /*pour les acces aux fichiers disques*/
#include<fcntl.h>  /*pour les valeurs O_WRONLY etc*/
#include<math.h>   /*pour SIN, SQRT,...des variables
dependantes*/
#define NICO   6
#define NCL    2
#define DRMAX  20
#define NILS  500
#define ON 1
#define OFF 0
float Sigmoide(float x,float c, float theta);/*sigmoide
centree avec seuil*/
void journal(char *typp,char *nf);
void saute_chaine(int nc);
FILE *fp1;
char *nomfichier1;
char chaine[100];
```

```
main()
{
/*FICHIERS*/
 FILE *fp3;
 char *nomfichier3;/*fichier d'ensemble d'apprentissage
 necessaire pour le perceptron de la derniere couche ou le
protocole comparatif*/
 extern FILE *fp1;
 extern char *nomfichier1;/*fichier de mesures provenant
d'ORTHO_15.C*/
 FILE *fp5;
 char *nomfichier5;/*Data ON*/
 FILE *fp6;
 char *nomfichier6;/*Data OFF*/
 FILE *fp7;
 char *nomfichier7;/*Fichier de detection (resultats)*/
 FILE *fp8,*fp9;
 char *nomfichier8,*nomfichier9;/*ensembles d'apprentissages
pour le reseau suivant
 qui ne contiendra que la classe des gammas*/
 FILE *fp10;
 char *nomfichier10;/*fichier des etats de la couche choisie
qui servira d'ensemble d'apprentissage pour la couche
suivante*/
 char rep;
 char *ty,*nf;
/*ENSEMBLE D'APPRENTISSAGE*/
 int nils;/*nb total d'exemples de l'ensemble
d'apprentissage*/
 int classe[NILS];/*pas d'indice, classe courante
seulement*/
 int np;/*nombre de parametres, nb d'entrees du reseau*/
 int ncl;/*nombre de classes*/
 unsigned int Nk[NCL],Nkt[NCL];
/*COMPTEURS*/
 unsigned int ils,ilsmin,ilsmax;
 int i,imax;/*cellules du <<premier>> niveau*/
 int j,jmax;/*cellules du <<second niveau>>*/
 int k,kprime;/*compteur d'exemples DANS <<la>> classe*/
 int c0,c1;/*compteurs de classes*/
 int ico;/*compteur de couches*/
 int inf;/*indice des fonctions non-lineaires*/
 int po,pr;/*post et pre synaptique*/
 int bidon;
/*STRUCTURE NEURONALE*/
 float z[NICO][DRMAX][NILS];
 int nico;/*nombre de couches*/
 int dr[NICO];/*nb de neurones de la [couche No]*/
 float c[NICO][DRMAX];/*constantes pour la sigmoide*/
 float theta[NICO][DRMAX];/*seuils pour la sigmoide*/
```

```
  float C[NICO][DRMAX][DRMAX];
   /*Couplages[niveau-niveau][cellule 1ier][cellule 2ieme]
          0  0 -> 1
          1  1 -> 2 etc...*/
  float D[NICO][DRMAX][DRMAX][DRMAX];/*couplages pour les
produits*/
  int cl;/*classe desiree*/
  char car;
  int ii,jj,iico,iprime,iiprime,pre;
  float fxik;
  float Norm;
/*VARIABLES POUR LES TESTS ET LES MESURES*/
 unsigned int sdEA[NCL][NCL],sdTST[NCL][NCL],sdFR[NCL][NCL];
/*compteurs de resultats
 sdEA[classe du fichier][classe de la sortie (signe)]
 sdTST[ON ou OFF][signe de la sortie]
 sdFR[ON ou OFF][classe assignee]
*/
 int flagnor;/*0: DETEC_15 n'a pas a normaliser les entrees
        1: DETEC_15 doit normaliser les entrees*/
 int post;
 float
RkkEA[NICO][NCL][NCL],UEA[NICO],SOMkfyEA[NICO][DRMAX][NCL];
 float RkkFR[NCL][NCL],UFR,SOMkfyFR[DRMAX][NCL];
 float zi,ci,ti,fact;
 float Cj[DRMAX];/*couplages de la derniere couche a la
sortie "Perceptron"*/
 unsigned int nilst;/*nils pour le fichier de tests*/
 int npt,nclt;
 float zt[NICO][DRMAX];/*etats des neurones pour les test*/
 float norm,norme[NCL],sortie,gain[NCL];
 int couche_choisie,clFR;
 float SeuilP,SeuilF,excess_per,excess_fra;
 float minP,minF,maxP,maxF,minEA,maxEA;
 int flag_prepro,fev;
 float Q_factor_bas,Q_factor_haut,Ng,Ngo,Np,Npo;
 int flag_seuil;
 int NVEA[DRMAX][NCL][2]; /*Protocole Zero (sortie=l'etat
d'une cellule)*/
/*NV: Nb de Vecteurs d'entree EA: de l'ensemble
d'apprentissage
     [de la cellule de la couche choisie]
     [de la classe]
     [0: inferieur au seuil, 1: superieur au seuil de la
cellule en question]*/
/*POUR LE FICHIER DE MESURES*/
 float bec1,bec2,betheta1,betheta2,beC1,beC2,beD1,beD2;
 float Rkk[NICO][DRMAX][DRMAX];
 int inb,nb_a_lire,ipasse,ib;
 unsigned int nex;/*Nb d'exemples d'un fichier*/
```

```
/*-initialisations de toutes les variables indexees----*/
 for(k=0;k<NCL;k++){Nk[k]=0;}
 for(ils=0;ils<NILS;ils++){classe[ils]=0;}
 for(ico=0;ico<NICO;ico++)
 {
  dr[ico]=0;
  for(i=0;i<DRMAX;i++)
  {
  c[ico][i]=1.00;theta[ico][i]=0.00;
  for(ils=0;ils<NILS;ils++)
  {
   z[ico][i][ils]=0.00;
  }/*ils*/
  for(j=0;j<DRMAX;j++)
  {
   C[ico][i][j]=0.00;
   for(iprime=0;iprime<DRMAX;iprime++)
   {
   D[ico][i][iprime][j]=0.00;
   }/*iprime*/
  }/*j*/
  for(k=0;k<NCL;k++)
  {
   SOMkfyEA[ico][i][k]=0.00;SOMkfyFR[i][k]=0.00;
  }/*k*/
  }/*i*/
  for(k=0;k<NCL;k++)
  {
  for(kprime=0;kprime<NCL;kprime++)
  {
   RkkEA[ico][k][kprime]=0.00;RkkFR[k][kprime]=0.00;
  }/*kprime*/
  }/*k*/
  UEA[ico]=0.00;
 }/*ico*/
/*---------PARAMETRES A FIXER AVANT L'EXECUTION-------*/
/*Ensemble d'apprentissage pour le protocole comparatif*/
    nomfichier3="/home/bandeco/ortho/markaria/eamark.255";
/*sorties pour le reseau suivant*/
    nomfichier8="/home/bandeco/ortho/bidon1";
    nomfichier9="/home/bandeco/ortho/bidon2";
/*Fichier de mesures provenant de l'apprentissage*/
    nomfichier1="/home/bandeco/ortho/m18_17_9.M03";
/*METTRE LE BON NOMBRE DE LIGNES A SAUTER POUR
saute_chaine()*/
/*Fichier data ON*/
    nomfichier5="/home/bandeco/ortho/markaria/zilch.dat";
/*Fichier data OFF correspondant*/
    nomfichier6="/home/bandeco/ortho/markaria/zilch.dat";
/*Fichier des resultats*/
```

```
    nomfichier7="/home/bandeco/ortho/premier_10";
/*Fichier d'ensemble d'apprentissage pour la couche
suivante*/
    nomfichier10="/home/bandeco/ortho/mar255_17.EA";
 flagnor=0;/*ATTENTION c'est pas le meme <<flag>> que dans
ORTHO_12,
       ici c'est un entier
       0 (zero) pas de normalisation
       1 Il y a normalisation (des entrees) et c'est le 1
qu'on teste*/
 couche_choisie=1;
/*choisir ici la couche ou le perceptron de sortie sera
branche*/
 nb_a_lire=120;/*position de l'architecture dans le fichier
de mesures*/
 flag_prepro=1;/*0:prepro actif, 1:prepro inactif, Dans ce
cas il faut l'installer comme dans SELASD.C dans les
sections data ON et data OFF*/
 flag_seuil=1;
 SeuilP=0.00;/*^^^^^^^^^^^^^^^^^^*/
 SeuilF=0.000;
 ilsmin=0;ilsmax=50000;
 /*mettre 0 et 50000 pour explorer tout le fichier*/
/*----fin des parametres a regler-----*/
/*--------Ecriture au journal------------------------*/
/* ty="J-M-A";nf="...";journal(ty,nf);
 ty="Programme:";nf="DETEC_15.C";journal(ty,nf);
 ty="Ensemble d'apprentissage:";journal(ty,nomfichier3);
 ty="Fichier de mesures:";journal(ty,nomfichier1);
 ty="  Data ON:";journal(ty,nomfichier5);
 ty=" Data OFF:";journal(ty,nomfichier6);
 ty="Resultats:";journal(ty,nomfichier7);
*/
/*ouvrir le fichier 7 (resultats)*/
 if((fp7=fopen(nomfichier7,"w"))==NULL)
  {
  printf("\nImpossible d'ouvrir %s
(resultats)",nomfichier7);
   goto hell;
 }
 fprintf(fp7,"\nProgramme DETEC_15.C Resultats:
%s",nomfichier7);
 fprintf(fp7,"\nFichier de mesures: %s", nomfichier1);
 fprintf(fp7,"\nEnsemble d'apprentissage: %s",nomfichier3);
 fprintf(fp7,"\n  Data ON: %s",nomfichier5);
 fprintf(fp7,"\n Data OFF: %s",nomfichier6);
/*-------Lecture du fichier de mesures------------------*/
/*comme dans MESEP_15.C*/
 if((fp1=fopen(nomfichier1,"r"))==NULL)
 {
   printf("\nImpossible d'ouvrir: %s",nomfichier1);
   goto hell;
  }
 /*-----Lire l'entete du fichier de mesures--------------*/
  saute_chaine(7);fscanf(fp1,"%u",&nex);
  saute_chaine(1);fscanf(fp1,"%d",&np);
  saute_chaine(1);fscanf(fp1,"%d",&ncl);
  saute_chaine(1);fscanf(fp1,"%d",&nico);
 /*DEB*/ printf("\nDEBligne 241 nex=%u np=%d ncl=%d
 nico=%d\n",nex,np,ncl,nico);
  for(i=0;i<nico;i++)
  {
   saute_chaine(2);fscanf(fp1,"%d",&dr[i]);saute_chaine(1);
 /*DEB*/ printf("%d  ",dr[i]);
  }
 /*--------Sauter les valeurs initiales------------------*/
 /*ecrire le bon nombre de chaines a sauter
   59 XOR a deux couches seulement      2-2
   76 pour le XOR 2-2-2
   92 pour l'architecture 2-3-2
   93 pour le XOR 2-4-1
   130 pour XOR 2-5-2
   152 pour la serie ASD (toy data) avec l'architecture 4 4 4
 ORTHO_12.C i'=i+1
   184 pour la serie ASD (toy data) avec l'architecture 4 4 4
 ORTHO_12.C i'=i
   184 -----------------------------(ORTHO_15.C)???
   198 pour la serie ASD (toy data) avec l'architecture 4 6 3
   385 pour la serie ASD (toy data) avec l'architecture 4 5 5
 5 5
   491 pour la serie PAIR10PO avec l'architecture 7 7 7
   273 pour la serie PAIR10PO avec l'architecture 7 7
   186 pour ASDASD avec ORTHO_15.C 4-4-4 i'=i && i'!=i
   814 pour 8-8-8
   1181 pour 8-15-3
   1191 pour l'archtecture 8-8-8-8
   1212 pour l'architecture 8-10-10
   1504 pour 8-15-3-2
   1568 pour l'architecture 8-8-8-8-8
   1945 pour 8-8-8-8-8-8
   2263 pour 8-17-7-5
   2418 pour 8-17-9
 */
  saute_chaine(2418);/*saute toutes les valeurs initiales*/
  saute_chaine(2);/*pour les pointilles*/
 /*---------boucle des passes (LECTURE)------------------*/
  for(inb=0;inb<nb_a_lire;inb++)
  {
   fscanf(fp1,"%d",&ipasse);
 /*AFF*/ printf("%d ",ipasse);
```

```
   fscanf(fp1,"%f %f",&bec1,&bec2);
   fscanf(fp1,"%f %f",&betheta1,&betheta2);
   fscanf(fp1,"%f %f",&beC1,&beC2);
   fscanf(fp1,"%f %f",&beD1,&beD2);
   for(ib=0;ib<nico;ib++)
   {
   fscanf(fp1,"%d",&ico);
   for(ii=0;ii<dr[ico];ii++)
   {
    fscanf(fp1,"%d",&i);
    fscanf(fp1,"%f %f",&c[ico][i],&theta[ico][i]);
    if(ib<nico-1)
    {
    for(j=0;j<dr[ico+1];j++)
    {
     fscanf(fp1,"%f",&C[ico][i][j]);
     for(iprime=i;iprime<dr[ico];iprime++)
     {
     fscanf(fp1,"%f",&D[ico][i][iprime][j]);
     }/*iprime*/
    }/*j*/
    }/*if ib=pre*/
   }/*ii*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k;kprime<ncl;kprime++)
    {
    fscanf(fp1,"%f",&Rkk[ico][k][kprime]);
    }/*kprime*/
   }/*k*/
   }/*ib ATTENTION IL Y A ib ET inb*/
    saute_chaine(1);/*pour les pointilles entre les passes*/
 }/*inb*/
 fclose(fp1);/*fichier de mesures*/
/*----Lecture de l'ensemble d'apprentissage---------------*/
 if((fp3=fopen(nomfichier3,"r"))==NULL)
 {printf("\nImpossible d'ouvrir l'ensemble
d'apprentissage");goto hell;}
 fscanf(fp3,"%d %d %d",&nils,&np,&ncl);
/*AFF*/
 printf("\n------------------------------------------
----------------------");
 printf("\nEnsemble d'apprentissage, nils=%d
np=%d,ncl=%d",nils,np,ncl);
 fprintf(fp7,"\n------------------------------------------
----------------------------");
 fprintf(fp7,"\nEnsemble d'apprentissage, nils=%d
np=%d,ncl=%d",nils,np,ncl);
/*AFF*/
 if(np!=dr[0])
```

```
  {printf("\nl'architecture (%d) et l'ensemble
 d'apprentissage (%d) ne sont pas compatibles",dr[0],np);goto
 hell;}
  for(i=0;i<ncl;i++){Nk[i]=0;}
  for(ils=0;ils<nils;ils++)
  {
   Norm=0.0;
   for(i=0;i<dr[0];i++)
   {
   fscanf(fp3,"%f",&z[0][i][ils]);
   Norm=Norm+z[0][i][ils]*z[0][i][ils];
   }/*i*/
   Norm=(float)sqrt((double)Norm);
   if(flagnor==1)
   {
   for(i=0;0<dr[0];i++)
   {
    z[0][i][ils]=z[0][i][ils]/Norm;
   }/*i*/
   }/*if flagnor*/
   fscanf(fp3,"%d",&classe[ils]);k=classe[ils];Nk[k]++;
  }/*ils*/
  fclose(fp3);/*fini de lire le fichier d'apprentissage*/
/*AFF*/
    printf("\n   Classe 0: %d   Classe 1: %d",Nk[0],Nk[1]);
    fprintf(fp7,"\n   Classe 0: %d   Classe 1:
%d",Nk[0],Nk[1]);
/*AFF*/
/*initialisation des SOMkfyEA*/
  for(ico=0;ico<nico;ico++)
  {
  for(j=0;j<dr[ico];j++)
  {
   for(k=0;k<ncl;k++)
   {
   SOMkfyEA[ico][j][k]=0.00;
   }/*k*/
  }/*j*/
  UEA[ico]=0.00;
  }/*ico*/
/*DEBprintf("\n~338--calcul des etats des neurones et des
recouvrements-----");*/
/*On a besoin des vecteurs (de classe) moyens pour les Cj du
<<Perceptron terminal>>, et on besoin des etats des neurones
pour calculer les Rkk (ou les fonctions U) de reference pour
le protocole comparatif*/
  if((fp10=fopen(nomfichier10,"w"))==NULL)
  {printf("\nImpossible d'ouvrir %s",nomfichier10);goto
hell;}
  fprintf(fp10,"%d %d %d\n",nils,dr[couche_choisie],ncl);
```

```
   for(ils=0;ils<nils;ils++)
   {
   cl=classe[ils];
   for(pre=0;pre<nico-1;pre++)
   {
    post=pre+1;imax=dr[pre];jmax=dr[post];
    for(j=0;j<jmax;j++)
    {
    z[post][j][ils]=0.00;
    for(i=0;i<imax;i++)
    {
     zi=z[pre][i][ils];ci=c[pre][i];ti=theta[pre][i];
     fxik=Sigmoide(zi,ci,ti);
     z[post][j][ils]=z[post][j][ils]+C[pre][i][j]*fxik;
     for(iprime=i;iprime<imax;iprime++)
     {
zi=z[pre][iprime][ils];ci=c[pre][iprime];ti=theta[pre][iprim
e];
z[post][j][ils]=z[post][j][ils]+D[pre][i][iprime][j]*fxik*Si
gmoide(zi,ci,ti);
     }/*iprime*/
    }/*i*/
    zi=z[post][j][ils];ci=c[post][j];ti=theta[post][j];
SOMkfyEA[post][j][cl]=SOMkfyEA[post][j][cl]+Sigmoide(zi,ci,t
i);
/*on ne calcule pas du tout les SOMk pour la couche d'entree
(OK)*/
     if(post==couche_choisie) {fprintf(fp10,"%f ",zi);}
    }/*j*/
    if(post==couche_choisie) {fprintf(fp10,"%d\n",cl);}
  }/*pre*/
  }/*ils*/
  fclose(fp10);
/*-----------Divisions par Nk--------------------------*/
  for(ico=1;ico<nico;ico++)
  {
  jmax=dr[ico];
  for(j=0;j<jmax;j++)
  {
   for(k=0;k<ncl;k++)
   {
   SOMkfyEA[ico][j][k]=SOMkfyEA[ico][j][k]/(float)Nk[k];
   }/*k*/
  }/*j*/
  }/*ico*/
/*----calcul des Rkk et des U pour l'ensemble
d'apprentissage---*/
  for(k=0;k<ncl;k++)
  {
  for(kprime=k;kprime<ncl;kprime++)
```

```
   {
    for(ico=0;ico<nico;ico++)
    {
    RkkEA[ico][k][kprime]=0.00;
    for(j=0;j<dr[ico];j++)
    {
RkkEA[ico][k][kprime]=RkkEA[ico][k][kprime]+SOMkfyEA[ico][j]
[k]*SOMkfyEA[ico][j][kprime];
    }/*j*/
RkkEA[ico][k][kprime]=RkkEA[ico][k][kprime]/(float)dr[ico];
    fact=-2.00;if(k==kprime){fact=1.00;}
    UEA[ico]=UEA[ico]+fact*RkkEA[ico][k][kprime];
    }/*ico*/
   }/*kprime*/
   }/*k*/
/*--COUPLAGES pour la cellule de sortie unique------------*/
/*----Perceptron pour deux classes seulement--------------*/
/*On met les vecteurs Cj dans la direction de la difference
entre les deux vecteurs moyens de classe. Si les vecteurs
moyens sont normalises, cette direction est aussi celle de
la perpendiculaire a la bissectrice de l'angle entre les
deux vecteurs moyens.*/
  ico=couche_choisie;
  jmax=dr[ico];
  for(k=0;k<ncl;k++)
  {
  norme[k]=0.00;
  for(j=0;j<jmax;j++)
  {
norme[k]=norme[k]+SOMkfyEA[ico][j][k]*SOMkfyEA[ico][j][k];
  }/*j*/
  norme[k]=(float)sqrt((double)norme[k]);
  }/*k*/
  for(j=0;j<jmax;j++)
  {
  Cj[j]=(SOMkfyEA[nico-1][j][0]/norme[0])-(SOMkfyEA[nico-
1][j][1]/norme[1]);
  }/*i*/
/*AFF*/for(ico=1;ico<nico;ico++)
     {
     printf("\nR[%d][0][0]=%2.6f",ico,RkkEA[ico][0][0]);
     printf(" R[%d][0][1]=%2.6f",ico,RkkEA[ico][0][1]);
     printf(" R[%d][1][1]=%2.6f",ico,RkkEA[ico][1][1]);
     printf("  U=%2.6f",UEA[ico]);
     fprintf(fp7,"\nR[%d][0][0]=%2.6f",ico,RkkEA[ico][0][0]);
     fprintf(fp7," R[%d][0][1]=%2.6f",ico,RkkEA[ico][0][1]);
     fprintf(fp7," R[%d][1][1]=%2.6f",ico,RkkEA[ico][1][1]);
     fprintf(fp7,"  U=%2.6f",UEA[ico]);
     }/*ico*/
/*AFF*/
```

```
/*DEBprintf("\n ~428 getchar()");getchar();*/

recalcule_seuil:

/*initialise les compteurs des sorties*/
  for(i=0;i<ncl;i++)
  {
  for(j=0;j<2;j++) /*2 c'est le nombre de signes de la
sortie*/
  {
   sdEA[i][j]=0;/*sd[classe EA][signe de la sortie du
perceptron]*/
   sdTST[i][j]=0;
   sdFR[i][j]=0;
  }/*j pour le signe des etats de la sortie*/
  Nkt[i]=0;
  }/*i pour les classes*/
/*DEBprintf("\n~440--Protocole de sortie pour l'ensemble
d'apprentissage--");*/
  ico=couche_choisie;
  jmax=dr[ico];
/*Perceptron, solution exacte*/
  if(ncl>2)
  {
  printf("\nle protocole du perceptron n'est pas
(actuellemment)");
  printf("\nadapte a un probleme a plus de deux
classes");goto hell;
  }
  if((fp8=fopen(nomfichier8,"w"))==NULL)
  {printf("\nImpossible d'ouvrir %s",nomfichier8);goto
hell;}
  fprintf(fp8,"%d %d %d\n",nils,np,ncl);
  if((fp9=fopen(nomfichier9,"w"))==NULL)
  {printf("\nImpossible d'ouvrir %s",nomfichier9);goto
hell;}
  fprintf(fp9,"%d %d %d\n",nils,np,ncl);
  minEA=1.0e6;maxEA=-1.0e6;
  for(ils=0;ils<nils;ils++)/*roule sur l'ensemble
d'apprentissage*/
  {
   sortie=0.00;cl=classe[ils];
   for(j=0;j<jmax;j++)/*roule sur la couche choisie ou le
perceptron est connecte*/
   {
sortie=sortie+Cj[j]*Sigmoide(z[ico][j][ils],c[ico][j],theta[
ico][j]);
   }/*j*/
   if(sortie<minEA){minEA=sortie;}
   if(sortie>maxEA){maxEA=sortie;}
```

```
/*dans les fprintf(fp8,...[ico][j][ils] etat des neurones de
la
   couche_choisie [0][j][ils] etat des entrees, rendre actif
le bon bloc*/
   if(sortie>SeuilP)/*superieurs au seuil*/
   {
    sdEA[cl][1]++;
    for(j=0;j<np;j++)
    {
     fprintf(fp9,"%f ",z[ico][j][ils]);
    }
    fprintf(fp9," %d\n",classe[ils]);
   }
   else/*inferieurs au seuil*/
   {
    sdEA[cl][0]++;
    for(j=0;j<np;j++)
    {
     fprintf(fp8,"%f ",z[ico][j][ils]);
    }
    fprintf(fp8," %d\n",classe[ils]);
   }
  }/*ils*/
  fclose(fp8);fclose(fp9);
/*------------Protocole ZERO pour l'EA-----------------*/
/*  NVEA[DRMAX][NCL][2]  */
  imax=dr[couche_choisie];
  for(i=0;i<imax;i++)
  {
   for(cl=0;cl<ncl;cl++){NVEA[i][cl][0]=NVEA[i][cl][1]=0;}
   for(ils=0;ils<nils;ils++)
   {
    cl=classe[ils];
    if(z[couche_choisie][i][ils]<theta[couche_choisie][i])
    {
     NVEA[i][cl][0]++;
    }/*if en dessous du seuil*/
    else
    {
     NVEA[i][cl][1]++;
    }/*egal ou superieur au seuil*/
   }/*ils*/
  }/*i*/
/*----------Fin du protocole Zero pour l'EA-------------*/
/*AFF*/
  printf("\ncouche_choisie= %d",couche_choisie);
  printf("\nclasse 0: s<=0: %d    s>0:
%d",sdEA[0][0],sdEA[0][1]);
  printf("\nclasse 1: s<=0: %d    s>0:
%d",sdEA[1][0],sdEA[1][1]);
```

```
  printf("\n minEA=%f    maxEA=%f",minEA,maxEA);
  printf("\nSeuil pour le perceptron: %e",SeuilP);
  printf("\n Seuil pour le protocole: %e",SeuilF);
  Ngo=(float)(sdEA[1][0]+sdEA[1][1]);
  Ng=(float)sdEA[1][0];/*ca pourait etre [1] comme 2ieme
indice*/
  Npo=(float)(sdEA[0][0]+sdEA[0][1]);
  Np=(float)sdEA[0][0];/*ca pourrait etre [1] comme 2 ieme
indice*/
  Q_factor_bas=(Ng/Ngo)/((float)sqrt((double)(Np/Npo)));
  Ng=(float)sdEA[1][1];
  Np=(float)sdEA[0][1];
  Q_factor_haut=(Ng/Ngo)/((float)sqrt((double)(Np/Npo)));
  printf("\nQ-factor on the LS = %f
%f",Q_factor_bas,Q_factor_haut);
  printf("\n-------Protocole Zero--------------------------
---------------------");
  for(cl=0;cl<ncl;cl++)
  {
   printf("\n");
   imax=dr[couche_choisie];
   for(i=0;i<imax;i++)
   {
    printf("%d %d  ",NVEA[i][cl][0],NVEA[i][cl][1]);
   }/*i*/
  }/*cl*/
  printf("\n--------------------------------------------
----------------------");
  fprintf(fp7,"\ncouche_choisie= %d",couche_choisie);
  fprintf(fp7,"\nles inferieurs au seuil sont dans:
%s",nomfichier8);
  fprintf(fp7,"\nles superieurs au seuil sont dans:
%s",nomfichier9);
  fprintf(fp7,"\nclasse 0: s<=0: %d    s>0:
%d",sdEA[0][0],sdEA[0][1]);
  fprintf(fp7,"\nclasse 1: s<=0: %d    s>0:
%d",sdEA[1][0],sdEA[1][1]);
  fprintf(fp7,"\n minEA=%f    maxEA=%f",minEA,maxEA);
  fprintf(fp7,"\nSeuil pour le perceptron: %e",SeuilP);
  fprintf(fp7,"\n Seuil pour le protocole: %e",SeuilF);
  fprintf(fp7,"\nQ-factor on the LS = %f
%f",Q_factor_bas,Q_factor_haut);
  fprintf(fp7,"\n---------------------------------------
---------------------------");
  fprintf(fp7,"\n-------Protocole Zero---------------------
---------------------------");
  for(cl=0;cl<ncl;cl++)
  {
   fprintf(fp7,"\n");
   imax=dr[couche_choisie];
```

```
    for(i=0;i<imax;i++)
    {
     fprintf(fp7,"%d %d  ",NVEA[i][cl][0],NVEA[i][cl][1]);
    }/*i*/
   }/*cl*/
   fprintf(fp7,"\n--------------------------------------------
-----------------------------");
   if(flag_prepro==1)
    {
     printf("\n                        PREPROCESSEUR ACTIF");
     printf("\n--------------------------------------------
--------------------------");
     fprintf(fp7,"\n                        PREPROCESSEUR ACTIF");
     fprintf(fp7,"\n----------------------------------------
-------------------------------");
    }
   else
    {
     printf("\n                        PREPROCESSEUR INACTIF");
     printf("\n--------------------------------------------
--------------------------");
     fprintf(fp7,"\n                        PREPROCESSEUR
INACTIF");
     fprintf(fp7,"\n----------------------------------------
-------------------------------");
    }
/*AFF*/
/*=========Traitement des Data ON===================*/
  if(flag_seuil==1){goto saute_data_ON_et_OFF;}
    cl=ON;minP=minF=1e44;maxP=maxF=-1e44;
  if((fp5=fopen(nomfichier5,"r"))==NULL)  /*ouvrir le
fichier 5*/
  {
  printf("\nImpossible d'ouvrir %s",nomfichier5);
  goto hell;
  }
  fscanf(fp5,"%u %d %d",&nilst,&npt,&nclt);   /*Lecture de
l'entete*/
  if(npt!=np)
  {
  printf("\nle nombre de parametres de l'ensemble
d'apprentissage et");
  printf("\ndu fichier test ne sont pas compatibles");
  goto hell;
  }
  if(nclt>ncl)
  {
  printf("\nle nombre de classes de l'ensemble
d'apprentissage et");
  printf("\ndu fichier test ne sont pas compatibles");
```

```
   goto hell;
   }
   for(ils=0;ils<nilst;ils++)
   {
   for(i=0;i<dr[0];i++)
   {
    fscanf(fp5,"%f",&zt[0][i]);
   }/*i*/
   fscanf(fp5,"%d",&bidon);/*pour ce qui est ecrit en bout de
 ligne*/
   Nkt[1]++;
   if(flagnor==1)/*il faut normaliser les entrees*/
   {
    norm=0.0;
    for(i=0;i<dr[0];i++)
    {
    norm=norm+zt[0][i]*zt[0][i];
    }/*i*/
    norm=(float)sqrt((double)norm);
    for(i=0;i<dr[0];i++)
    {
    zt[0][i]=zt[0][i]/norm;
    }/*i*/
   }/*if flagnor*/
   if((ils>=ilsmin)&&(ils<ilsmax))
   {/*fais les calculs*/
/*---------calcul des etats des neurones---------------*/
   if(flag_prepro==1)
   {
/*detecter si l'evenement courant appartient a l'espece
choisie
  comme dans SELASD.C:
  27-juillet-97 a la suite de CROISE.C avec les fichiers
M1185 M1188 M1248 et M1371 .CRO Je decide d'essayer les
inegalites suivantes*/
    fev=0;
    if((0.22<zt[0][0])&&(zt[0][0]<0.32)){fev=fev+128;}
    if((0.12<zt[0][1])&&(zt[0][1]<0.20)){fev=fev+64;}
if(((0.00<zt[0][2])&&(zt[0][2]<0.20))||((1.32<zt[0][2])&&(zt
[0][2]<1.36))){fev=fev+32;}
if(((0.74<zt[0][3])&&(zt[0][3]<1.00))||((1.26<zt[0][2])&&(zt
[0][2]<1.32))||((1.36<zt[0][2])&&(zt[0][2]<1.42))){fev=fev+1
6;}
    if((0.10<zt[0][4])&&(zt[0][4]<0.22)){fev=fev+8;}
    if((0.36<zt[0][5])&&(zt[0][5]<0.64)){fev=fev+4;}
    if((0.50<zt[0][6])&&(zt[0][6]<0.74)){fev=fev+2;}
    if((0.00<zt[0][7])&&(zt[0][7]<0.26)){fev=fev+1;}
    if(fev!=255){goto saute_ON;}
  }/*preprocesseur actif*/
  for(post=1;post<nico;post++)    /*post OK*/
```

```
   {
    jmax=dr[post];
    pre=post-1;imax=dr[pre];    /*pre=pre*/
    for(j=0;j<jmax;j++)
    {
    zt[post][j]=0.0;
    for(i=0;i<imax;i++)
    {
     fxik=Sigmoide(zt[pre][i],c[pre][i],theta[pre][i]);
     zt[post][j]=zt[post][j]+C[pre][i][j]*fxik;
     for(iprime=i;iprime<imax;iprime++)
     {
     zt[post][j]=zt[post][j]+D[pre][i][iprime][j]*fxik*
Sigmoide(zt[pre][iprime],c[pre][iprime],theta[pre][iprime]);
     }/*iprime*/
    }/*i*/
    }/*j*/
   }/*post*/
/*---------Perceptron calcule, Data ON------------*/
  ico=couche_choisie;
  jmax=dr[ico];
/*Perceptron, solution exacte*/
  if(ncl>2)
  {
   printf("\nle protocole du perceptron n'est pas
(actuellemment)");
   printf("\nadapte a un probleme a plus de deux
classes");goto hell;
  }
  sortie=0.00;
  for(j=0;j<jmax;j++)/*roule sur la couche choisie pour le
perceptron*/
  {
sortie=sortie+Cj[j]*Sigmoide(zt[ico][j],c[ico][j],theta[ico]
[j]);
  }/*j*/
  if(sortie<minP){minP=sortie;}
  if(sortie>maxP){maxP=sortie;}
  if(sortie>SeuilP) {sdTST[ON][1]++;}
  else {sdTST[ON][0]++;}
/*-------protocole comparatif pour les Data ON------------*/
/*Description Voir MESEP_15.C*/
/*initialiser pour chaque nouvelle configuration*/
  for(k=0;k<ncl;k++)
  {
   for(kprime=0;kprime<ncl;kprime++)
   {
   RkkFR[k][kprime]=0.00;
   }/*kprime*/
  }/*k*/
```

```
   UFR=0.00;
 /*calcul des SOMk*/
   for(k=0;k<ncl;k++)
   {
    for(j=0;j<jmax;j++)
    {
SOMkfyFR[j][k]=(SOMkfyEA[ico][j][k]*(float)Nk[k])+Sigmoide(z
t[ico][j],c[ico][j],theta[ico][j]);
    }/*j*/
   }/*k*/
 /*Division par (Nk+1)*/
   for(j=0;j<jmax;j++)
   {
    for(k=0;k<ncl;k++)
    {
    SOMkfyFR[j][k]=SOMkfyFR[j][k]/((float)(Nk[k]+1));
    }/*k*/
   }/*j*/
 /*calcul des Rkk et U avec la configuration courante
ajoutee*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k;kprime<ncl;kprime++)
    {
    for(j=0;j<jmax;j++)
    {
RkkFR[k][kprime]=RkkFR[k][kprime]+SOMkfyFR[j][k]*SOMkfyFR[j]
[kprime];
    }/*j*/
    RkkFR[k][kprime]=RkkFR[k][kprime]/(float)jmax;
    fact=2.00;if(k==kprime){fact=1.00;}
    UFR=UFR+fact*RkkFR[k][kprime];
    }/*kprime*/
   }/*k*/
 /*Gains des classes*/
   for(k=0;k<ncl;k++)
   {
    gain[k]=RkkFR[k][k]-RkkEA[ico][k][k];
   }/*k*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k+1;kprime<ncl;kprime++)
    {
    clFR=k;
    if(gain[k]<gain[kprime]-SeuilF){clFR=kprime;}
    if((gain[kprime]-gain[k])<minF){minF=gain[kprime]-
gain[k];}
    if((gain[kprime]-gain[k])>maxF){maxF=gain[kprime]-
gain[k];}
    }/*kprime*/
   }/*k*/
   sdFR[ON][clFR]++;
 saute_ON:
   if(ils==ilsmax-1){ils=nilst-1;}/*pour sortir de la
 boucle*/
   }/*ils*/
  }/*fin du if il faut faire les calculs*/
    fclose(fp5);/*fichier data ON*/
 /*------Affichage et mise en fichier----------------*/
 /*AFF*/
   printf("\n%s",nomfichier5);
   printf("\nData ON %u  ilsmin %u  ilsmax
 %u",nilst,ilsmin,ilsmax);
   printf("\nPerceptron  s<=Seuil: %u  s>Seuil: %u    min=%e
 max=%e",sdTST[ON][0],sdTST[ON][1],minP,maxP);
   printf("\n Protocole     cl 0: %u     cl 1: %u    min=%e
 max=%e",sdFR[ON][0],sdFR[ON][1],minF,maxF);
   printf("\n------------------------------------------------
 ----------------------");
   fprintf(fp7,"\n%s",nomfichier5);
   fprintf(fp7,"\nData ON %u  ilsmin %u  ilsmax
 %u",nilst,ilsmin,ilsmax);
   fprintf(fp7,"\nPerceptron  s<=Seuil: %u  s>Seuil: %u
 min=%e   max=%e",sdTST[ON][0],sdTST[ON][1],minP,maxP);
   fprintf(fp7,"\n Protocole     cl 0: %u     cl 1: %u
 min=%e   max=%e",sdFR[ON][0],sdFR[ON][1],minF,maxF);
   fprintf(fp7,"\n-----------------------------------------
 ----------------------------");
 /*AFF*/
 /*========Traitement des Data OFF====================*/
   cl=OFF;
   if((fp6=fopen(nomfichier6,"r"))==NULL)  /*ouvrir le
 fichier 6*/
   {
   printf("\nImpossible d'ouvrir %s",nomfichier6);
   goto hell;
   }
   fscanf(fp6,"%u %d %d",&nilst,&npt,&nclt);   /*Lecture de
 l'entete*/
   if(npt!=np)
   {
   printf("\nle nombre de parametres de l'ensemble
 d'apprentissage et");
   printf("\ndu fichier test ne sont pas compatibles");
   goto hell;
   }
   if(nclt>ncl)
   {
   printf("\nle nombre de classes de l'ensemble
 d'apprentissage et");
```

```
   printf("\ndu fichier test ne sont pas compatibles");
   goto hell;
   }
   for(ils=0;ils<nilst;ils++)
   {
   for(i=0;i<dr[0];i++)
   {
    fscanf(fp6,"%f",&zt[0][i]);
   }/*i*/
   fscanf(fp6,"%d",&bidon);/*pour ce qui est ecrit en bout de
 ligne*/
   Nkt[1]++;
   if(flagnor==1)/*il faut normaliser les entrees*/
   {
    norm=0.0;
    for(i=0;i<dr[0];i++)
    {
    norm=norm+zt[0][i]*zt[0][i];
    }/*i*/
    norm=(float)sqrt((double)norm);
    for(i=0;i<dr[0];i++)
    {
    zt[0][i]=zt[0][i]/norm;
    }/*i*/
   }/*if flagnor*/
    if((ils>=ilsmin)&&(ils<ilsmax))
   {/*fais les calculs*/
/*----------calcul des etats des neurones---------------*/
  if(flag_prepro==1)
  {
/*detecter si l'evenement courant appartient a l'espece
choisie
  comme dans SELASD.C: 27-juillet-97 a la suite de CROISE.C
avec les fichiers M1185 M1188 M1248 et M1371 .CRO Je decide
d'essayer les inegalites suivantes*/
    fev=0;
    if((0.22<zt[0][0])&&(zt[0][0]<0.32)){fev=fev+128;}
    if((0.12<zt[0][1])&&(zt[0][1]<0.20)){fev=fev+64;}
if(((0.00<zt[0][2])&&(zt[0][2]<0.20))||((1.32<zt[0][2])&&(zt
[0][2]<1.36))){fev=fev+32;}
if(((0.74<zt[0][3])&&(zt[0][3]<1.00))||((1.26<zt[0][2])&&(zt
[0][2]<1.32))||((1.36<zt[0][2])&&(zt[0][2]<1.42))){fev=fev+1
6;}
    if((0.10<zt[0][4])&&(zt[0][4]<0.22)){fev=fev+8;}
    if((0.36<zt[0][5])&&(zt[0][5]<0.64)){fev=fev+4;}
    if((0.50<zt[0][6])&&(zt[0][6]<0.74)){fev=fev+2;}
    if((0.00<zt[0][7])&&(zt[0][7]<0.26)){fev=fev+1;}
    if(fev!=255){goto saute_OFF;}
  }/*preprocesseur actif*/
  for(post=1;post<nico;post++)   /*post OK*/
```

```
    {
     jmax=dr[post];
     pre=post-1;imax=dr[pre];     /*pre=pre*/
     for(j=0;j<jmax;j++)
     {
     zt[post][j]=0.0;
     for(i=0;i<imax;i++)
     {
      fxik=Sigmoide(zt[pre][i],c[pre][i],theta[pre][i]);
      zt[post][j]=zt[post][j]+C[pre][i][j]*fxik;
      for(iprime=i;iprime<imax;iprime++)
      {
      zt[post][j]=zt[post][j]+D[pre][i][iprime][j]*fxik*
Sigmoide(zt[pre][iprime],c[pre][iprime],theta[pre][iprime]);
      }/*iprime*/
     }/*i*/
     }/*j*/
    }/*post*/
/*---------Perceptron calcule, Data OFF------------*/
  ico=couche_choisie;
  jmax=dr[ico];
/*Perceptron, solution exacte*/
  if(ncl>2)
  {
   printf("\nle protocole du perceptron n'est pas
(actuellement)");
   printf("\nadapte a un probleme a plus de deux
classes");goto hell;
  }
  sortie=0.00;
  for(j=0;j<jmax;j++)/*roule sur la couche choisie pour le
perceptron*/
  {
sortie=sortie+Cj[j]*Sigmoide(zt[ico][j],c[ico][j],theta[ico]
[j]);
  }/*j*/
  if(sortie>SeuilP) {sdTST[OFF][1]++;}
  else {sdTST[OFF][0]++;}
/*------protocole comparatif pour les Data OFF------------*/
/*Description Voir MESEP_15.C*/
/*initialiser pour chaque nouvelle configuration*/
  for(k=0;k<ncl;k++)
  {
   for(kprime=0;kprime<ncl;kprime++)
   {
   RkkFR[k][kprime]=0.00;
   }/*kprime*/
  }/*k*/
  UFR=0.00;
/*calcul des SOMk*/
```

```
   for(k=0;k<ncl;k++)
   {
    for(j=0;j<jmax;j++)
    {
SOMkfyFR[j][k]=(SOMkfyEA[ico][j][k]*(float)Nk[k])+Sigmoide(z
t[ico][j],c[ico][j],theta[ico][j]);
    }/*j*/
   }/*k*/
/*Division par (Nk+1)*/
   for(j=0;j<jmax;j++)
   {
    for(k=0;k<ncl;k++)
    {
    SOMkfyFR[j][k]=SOMkfyFR[j][k]/((float)(Nk[k]+1));
    }/*k*/
   }/*j*/
/*calcul des Rkk et U avec la configuration courante
ajoutee*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k;kprime<ncl;kprime++)
    {
    for(j=0;j<jmax;j++)
    {
RkkFR[k][kprime]=RkkFR[k][kprime]+SOMkfyFR[j][k]*SOMkfyFR[j]
[kprime];
    }/*j*/
    RkkFR[k][kprime]=RkkFR[k][kprime]/(float)jmax;
    fact=2.00;if(k==kprime){fact=1.00;}
    UFR=UFR+fact*RkkFR[k][kprime];
    }/*kprime*/
   }/*k*/
/*Gains des classes*/
   for(k=0;k<ncl;k++)
   {
    gain[k]=RkkFR[k][k]-RkkEA[ico][k][k];
   }/*k*/
   for(k=0;k<ncl;k++)
   {
    for(kprime=k+1;kprime<ncl;kprime++)
    {
    clFR=k;
    if(gain[k]<gain[kprime]-SeuilF){clFR=kprime;}
    }/*kprime*/
   }/*k*/
   sdFR[OFF][clFR]++;
saute_OFF:
  if(ils==ilsmax-1){ils=nilst-1;}
  }/*ils*/
 }/*fin du if: faire les calculs des data OFF*/
```

```
  fclose(fp6);/*fichier data OFF*/
/*------Affichage et mise en fichier----------------*/
/*AFF*/
  printf("\n%s",nomfichier6);
  printf("\nData OFF %u  ilsmin %u  ilsmax
%u",nilst,ilsmin,ilsmax);
  printf("\nPerceptron   s<=Seuil: %u  s>Seuil:
%u",sdTST[OFF][0],sdTST[OFF][1]);
  printf("\n   Comparaison        cl 0: %u     cl 1:
%u",sdFR[OFF][0],sdFR[OFF][1]);
  printf("\n-------------------------------------------------
----------------------");
  fprintf(fp7,"\n%s",nomfichier6);
  fprintf(fp7,"\nData OFF %u  ilsmin %u  ilsmax
%u",nilst,ilsmin,ilsmax);
  fprintf(fp7,"\nPerceptron   s<=Seuil: %u  s>Seuil:
%u",sdTST[OFF][0],sdTST[OFF][1]);
  fprintf(fp7,"\n   Comparaison        cl 0: %u     cl 1:
%u",sdFR[OFF][0],sdFR[OFF][1]);
  fprintf(fp7,"\n-------------------------------------------
---------------------------");
/*AFF*/
/*-----------------excess in sigma------------------*/
excess_per=(float)(sdTST[ON][0]-sdTST[OFF][0])
     /(float)sqrt((double)(sdTST[ON][0]+sdTST[OFF][0]));
if(sdTST[ON][0]<sdTST[OFF][0]){excess_per=-1e88;}
excess_fra=(float)(sdFR[ON][1]-sdFR[OFF][1])
     /(float)sqrt((double)(sdFR[ON][1]+sdFR[OFF][1]));
if(sdFR[ON][1]<sdFR[OFF][1]){excess_fra=-1e88;}
/*AFF*/
  printf("\n  Perceptron excess in sigma: %f",excess_per);
  printf("\nCompa Proto excess in sigma: %f",excess_fra);
  printf("\n-------------------------------------------------
----------------------");
  fprintf(fp7,"\n  Perceptron excess in sigma:
%f",excess_per);
  fprintf(fp7,"\nCompa Proto excess in sigma:
%f",excess_fra);
  fprintf(fp7,"\n-------------------------------------------
---------------------------");
/*AFF*/
saute_data_ON_et_OFF:
  if(flag_seuil==1)
  {
   printf("\nnouveau seuil? (pour sortir: >1000)
");scanf("%f",&SeuilP);
   if(SeuilP<1000){goto recalcule_seuil;}
   else{goto hell;}
  }
hell:
 fclose(fp7);/*fichier de resultats*/
 printf("\n DETEC_15.C termine!");
 getchar();
}/*fin du main*/
/*===================================================*/
float Sigmoide(float x,float c, float theta)/*sigmoide
centree avec seuil*/
{                  /*Sigmoide corrigee*/
 float num,den,expon;
 double y;
 y=(double)-c*(x-theta);
 if(y<-20.00){num=1.00;den=1.00;goto saute;}
 if(y>20.00){num=-1.00;den=1.00;goto saute;}
 expon=(float)exp(y);
 num=1.0-expon;
 den=1.0+expon;
saute:
 return(num/den);
}/*fin de sigmoide*/
/*===================================================*/
void journal(char *typp,char *nf)
{
(Same as in the learning program)
}/*sous-routine journal*/
/*===================================================*/
void saute_chaine(int nc)
{
 extern FILE *fp1;
 extern char chaine[];
 int i;
 for(i=0;i<nc;i++)
 {
 fscanf(fp1,"%s",&chaine);
/*DEB printf("\nsaut_chai. %d %s",i+1,chaine);getchar();*/
 }
}/*fin de saute_chaine*/
/*===================================================*/
```

APPENDIX VIII Species tables

TABLE A-VIII-1 Markarian 421 species, totals over 16 pairs.

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	SPECIES	Total ON	Total OFF	EXCESS IN SIGMA
0	111032	111403	-0.78663	50	0	0	-
1	286	303	-0.70047	51	12	9	0.654654
2	442	443	-0.03361	52	0	0	-
3	3	1	1	53	279	233	2.032932
4	2920	2735	2.460115	54	2	0	1.414214
5	57	73	-1.40329	55	303	245	2.477637
6	9076	9314	-1.75504	56	1	0	1
7	79	89	-0.77152	57	810	772	0.955389
8	8507	8392	0.884642	58	0	0	-
9	126	115	0.708572	59	34	24	1.313064
10	69	77	-0.66208	60	0	1	-1
11	1	1	0	61	219	172	2.376892
12	3544	3537	0.083186	62	3	2	0.447214
13	122	127	-0.31686	63	825	740	2.148631
14	4061	4060	0.011097	64	39860	39796	0.226762
15	137	123	0.868243	65	0	0	-
16	83257	83580	-0.79078	66	279	284	-0.21072
17	594	576	0.526235	67	0	0	-
18	248	259	-0.48853	68	2980	2982	-0.0259
19	5	4	0.333333	69	0	0	-
20	1266	1148	2.40167	70	6704	6675	0.250718
21	41	44	-0.3254	71	1	0	1
22	4132	4001	1.4526	72	1927	1987	-0.95905
23	32	40	-0.94281	73	37	38	-0.11547
24	2854	2533	4.373524	74	107	73	2.53421
25	61	62	-0.09017	75	4	2	0.816497
26	62	57	0.458349	76	990	1037	-1.04393
27	6	3	1	77	51	35	1.725324
28	953	970	-0.38767	78	8022	7646	3.00387
29	38	40	-0.22646	79	367	308	2.270911
30	2014	1965	0.7768	80	24802	24980	-0.79778
31	116	120	-0.26038	81	0	2	-1.41421
32	20207	19939	1.337561	82	175	157	0.987878
33	17939	17919	0.105618	83	1	0	1
34	7	8	-0.2582	84	1552	1583	-0.55366
35	6	11	-1.21268	85	1	1	0
36	179	140	2.183581	86	2725	2807	-1.10249
37	406	359	1.699289	87	4	3	0.377964

38	224	232	-0.37463	88	1556	1545	0.197534
39	477	465	0.390981	89	128	148	-1.20386
40	830	797	0.818126	90	71	41	2.834734
41	2076	1988	1.380403	91	2	2	0
42	33	32	0.124035	92	336	356	-0.76029
43	34	42	-0.91766	93	26	30	-0.53452
44	249	238	0.498458	94	2411	2366	0.651081
45	424	388	1.263352	95	153	153	0
46	965	862	2.409728	96	1148	1166	-0.37419
47	1144	1166	-0.45774	97	8	7	0.258199
48	20	15	0.845154	98	2	1	0.57735
49	5058	5037	0.20901	99	0	0	-

Markarian 421

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	SPECIES	Total ON	Total OFF	EXCESS IN SIGMA
100	60	54	0.561951	150	3318	3316	0.024555
101	0	0	-	151	12	10	0.426401
102	76	94	-1.38054	152	576	577	-0.02945
103	0	0	-	153	58	67	-0.80498
104	3200	3082	1.488789	154	35	35	0
105	4437	4374	0.671163	155	2	1	0.57735
106	24	27	-0.42008	156	640	651	-0.30615
107	51	56	-0.48337	157	19	27	-1.17954
108	302	300	0.081514	158	4934	4863	0.717318
109	482	424	1.926921	159	283	295	-0.49913
110	830	761	1.729872	160	323	351	-1.07852
111	1823	1523	5.186308	161	602	598	0.11547
112	0	0	-	162	1	3	-1
113	4	3	0.377964	163	5	0	2.236068
114	0	0	-	164	72	77	-0.40962
115	0	0	-	165	251	257	-0.26621
116	0	0	-	166	94	98	-0.28868
117	0	0	-	167	530	504	0.808562
118	0	0	-	168	143	154	-0.63828
119	0	1	-1	169	290	310	-0.8165
120	5	5	0	170	5	9	-1.06904
121	1456	1431	0.465282	171	17	8	1.8
122	0	0	-	172	211	187	1.203011
123	38	29	1.099525	173	85	88	-0.22809
124	0	0	-	174	1261	1148	2.30229
125	209	195	0.696526	175	1363	1312	0.986071
126	3	2	0.447214	176	1	1	0
127	1165	960	4.447074	177	769	808	-0.98208
128	6907	6785	1.042621	178	0	0	-
129	60	51	0.854242	179	1	3	-1
130	154	181	-1.47517	180	0	0	-
131	2	1	0.57735	181	231	191	1.94717
132	2207	2223	-0.24039	182	0	0	-
133	17	20	-0.4932	183	546	458	2.777255
134	12232	12222	0.063948	184	0	0	-
135	38	45	-0.76835	185	434	427	0.238559

136	774	739	0.899805	186	0	0	-
137	19	39	-2.62613	187	17	25	-1.23443
138	24	33	-1.19208	188	0	0	-
139	3	0	1.732051	189	88	82	0.460179
140	1243	1261	-0.35971	190	1	2	-0.57735
141	15	18	-0.52223	191	1665	1493	3.060711
142	5310	5181	1.259451	192	6578	6740	-1.40377
143	140	145	-0.29617	193	0	0	-
144	7107	7132	-0.20951	194	396	421	-0.87464
145	129	117	0.765092	195	0	0	-
146	42	52	-1.03142	196	1972	1959	0.207344
147	0	0	-	197	0	0	-
148	771	745	0.667765	198	12156	12213	-0.36514
149	19	14	0.870388	199	0	0	-

Markarian 421 16
pairs of 95

	Total	Total	EXCESS		Total	Total	EXCESS
SPECIES	ON	OFF	IN SIGMA	SPECIES	ON	OFF	IN SIGMA
200	1071	1119	-1.0257	228	9	17	-1.56893
201	41	57	-1.61624	229	0	0	-
202	125	113	0.777844	230	46	51	-0.50767
203	7	3	1.264911	231	0	0	-
204	446	450	-0.13363	232	304	259	1.896524
205	17	22	-0.80064	233	590	525	1.946596
206	4872	4880	-0.08101	234	22	17	0.800641
207	237	196	1.970334	235	33	37	-0.47809
208	3344	3423	-0.96035	236	117	124	-0.45091
209	0	0	-	237	235	223	0.560723
210	119	113	0.393919	238	814	710	2.664041
211	0	0	-	239	1519	1364	2.886751
212	987	984	0.067574	240	0	0	-
213	0	0	-	241	0	0	-
214	2863	2838	0.331104	242	0	0	-
215	0	0	-	243	0	0	-
216	1612	1614	-0.03521	244	0	0	-
217	130	132	-0.12356	245	0	0	-
218	99	89	0.729325	246	0	0	-
219	12	5	1.697749	247	0	0	-
220	550	572	-0.65679	248	2	1	0.57735
221	38	35	0.351123	249	970	895	1.736688
222	3385	3351	0.414265	250	0	0	-
223	210	217	-0.33875	251	58	49	0.870063
224	32	28	0.516398	252	0	1	-1
225	0	0	-	253	307	245	2.638895
226	1	1	0	254	2	2	0
227	0	0	-	255	1802	1295	9.110398

TABLE A-VIII-2 Crab species, totals over 19 pairs.

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	SPECIES	Total ON	Total OFF	EXCESS IN SIGMA
0	59174	59311	-1	50	0	0	-1
1	519	553	-1	51	0	0	-1
2	144	147	-1	52	2	2	-1
3	3	8	-1	53	1	0	1
4	1167	1199	-1	54	2	3	-1
5	43	34	1.025645	55	3	0	1.732051
6	2273	2223	0.745687	56	10	17	-1
7	51	57	-1	57	3	4	-1
8	381	383	-1	58	0	0	-1
9	19	22	-1	59	0	0	-1
10	89	92	-1	60	1	1	-1
11	3	1	1	61	0	0	-1
12	5	4	0.333333	62	2	3	-1
13	4	2	0.816497	63	0	1	-1
14	134	132	0.122628	64	9286	9730	-1
15	11	9	0.447214	65	2	0	1.414214
16	35575	35684	-1	66	134	113	1.336198
17	0	0	-1	67	0	0	-1
18	65	69	-1	68	851	858	-1
19	0	0	-1	69	0	0	-1
20	2950	2961	-1	70	2574	2504	0.982317
21	0	0	-1	71	0	0	-1
22	1325	1266	1.159093	72	823	830	-1
23	0	0	-1	73	71	81	-1
24	192	199	-1	74	76	57	1.647509
25	0	0	-1	75	9	5	1.069045
26	24	26	-1	76	127	121	0.381
27	0	0	-1	77	27	9	3
28	15	17	-1	78	660	599	1.719163
29	0	0	-1	79	58	55	0.282216
30	20	31	-1	80	4097	4560	-1
31	0	0	-1	81	0	0	-1
32	7375	7372	0.024704	82	58	71	-1
33	10244	10301	-1	83	0	0	-1
34	6	9	-1	84	749	766	-1
35	55	39	1.650274	85	0	0	-1
36	70	69	0.084819	86	1469	1451	0.333105
37	139	147	-1	87	0	0	-1
38	74	72	0.165521	88	7	7	-1
39	255	261	-1	89	0	0	-1
40	135	166	-1	90	18	19	-1
41	979	958	0.47715	91	0	0	-1
42	18	10	1.511858	92	3	3	-1
43	3	11	-1	93	0	0	-1
44	11	5	1.5	94	112	135	-1

45	43	44	-1	95	0	0	-1
46	56	32	2.558409	96	465	517	-1
47	74	64	0.851257	97	1	3	-1
48	214	187	1.348316	98	0	1	-1
49	33	35	-1	99	0	0	-1

Crab 93-94 19 pairs

SPECIES	Total ON	Total OFF	EXCESS IN SIGMA	SPECIES	Total ON	Total OFF	EXCESS IN SIGMA
100	24	25	-1	150	759	823	-1
101	0	0	-1	151	0	0	-1
102	46	40	0.646997	152	90	69	1.665408
103	0	0	-1	153	0	0	-1
104	1066	1126	-1	154	41	57	-1
105	1996	1951	0.716274	155	0	0	-1
106	14	12	0.392232	156	14	8	1.279204
107	26	27	-1	157	0	0	-1
108	134	99	2.292926	158	122	125	-1
109	194	224	-1	159	0	0	-1
110	210	224	-1	160	171	193	-1
111	526	479	1.482569	161	638	621	0.479111
112	0	0	-1	162	5	5	-1
113	0	0	-1	163	50	45	0.512989
114	0	0	-1	164	35	31	0.492366
115	0	0	-1	165	67	62	0.440225
116	0	0	-1	166	33	38	-1
117	0	0	-1	167	217	186	1.54422
118	0	0	-1	168	40	49	-1
119	0	0	-1	169	274	247	1.182891
120	59	53	0.566947	170	8	9	-1
121	7	5	0.57735	171	4	8	-1
122	1	0	1	172	9	12	-1
123	0	0	-1	173	57	55	0.188982
124	5	5	-1	174	46	41	0.536056
125	3	3	-1	175	144	135	0.538816
126	8	13	-1	176	5	2	1.133893
127	1	4	-1	177	0	0	-1
128	3066	3046	0.255822	178	1	0	1
129	101	103	-1	179	0	0	-1
130	81	78	0.237915	180	1	2	-1
131	9	5	1.069045	181	0	0	-1
132	397	402	-1	182	4	1	1.341641
133	10	8	0.471405	183	2	2	-1
134	1143	1122	0.44125	184	0	2	-1
135	42	42	-1	185	0	0	-1
136	357	295	2.428107	186	1	0	1
137	33	30	0.377964	187	0	0	-1
138	122	113	0.587095	188	1	1	-1
139	3	1	1	189	1	1	-1
140	51	38	1.377997	190	3	3	-1
141	12	6	1.414214	191	0	0	-1

142	410	429	-1	192	1099	1107	-1
143	43	47	-1	193	0	0	-1
144	2023	2031	-1	194	56	69	-1
145	0	0	-1	195	0	0	-1
146	35	52	-1	196	374	422	-1
147	0	0	-1	197	0	0	-1
148	410	403	0.245501	198	2215	2317	-1
149	0	0	-1	199	0	0	-1

TABLE A-VIII-2 Crab species, totals over 19 pairs.

	Total	Total	EXCESS		Total	Total	EXCESS
SPECIES	ON	OFF	IN SIGMA	SPECIES	ON	OFF	IN SIGMA
200	805	829	-1	228	10	6	1
201	60	66	-1	229	0	0	-1
202	127	101	1.721892	230	29	26	0.40452
203	12	8	0.894427	231	0	0	-1
204	285	305	-1	232	137	166	-1
205	17	24	-1	233	531	488	1.347043
206	3723	3630	1.084553	234	12	10	0.426401
207	288	285	0.125327	235	51	46	0.507673
208	813	839	-1	236	51	65	-1
209	0	0	-1	237	197	175	1.140647
210	51	61	-1	238	536	451	2.70558
211	0	0	-1	239	1982	1435	9.357608
212	316	337	-1	240	0	0	-1
213	0	0	-1	241	0	0	-1
214	1785	1885	-1	242	0	0	-1
215	0	0	-1	243	0	0	-1
216	22	29	-1	244	0	0	-1
217	0	0	-1	245	0	0	-1
218	31	24	0.94388	246	0	0	-1
219	0	0	-1	247	0	0	-1
220	18	16	0.342997	248	5	3	0.707107
221	0	0	-1	249	1	2	-1
222	1007	1079	-1	250	1	1	-1
223	0	0	-1	251	0	0	-1
224	18	31	-1	252	6	1	1.889822
225	0	0	-1	253	0	0	-1
226	2	2	-1	254	15	13	0.377964
227	0	0	-1	255	5	5	-1

TABLE A-VIII-3 Non-empty simulation species

SPECIES	Nb events	SPECIES	Nb events
0	2	165	1
4	2	166	3
8	2	167	4
16	1	169	3
32	4	173	5
33	10	174	2
37	2	175	24
38	2	177	1
41	1	183	3
45	1	185	1
46	2	187	1
47	2	189	1
63	4	191	57
64	1	198	1
73	1	200	5
78	3	205	4
79	1	206	10
104	1	207	6
105	7	208	1
109	3	216	1
110	9	222	7
111	17	223	4
121	5	232	1
127	16	233	12
128	1	235	3
132	1	236	1
134	3	237	18
135	1	238	10
136	2	239	238
144	1	249	18
150	1	251	7
158	2	253	8
160	1	255	417
161	5	Total	996
164	1		