AUTOMATA NETWORKS AND ARTIFICIAL INTELLIGENCE

F. FOGELMAN SOULIE*, P. GALLINARI**, Y. LE CUN***, S. THIRIA**

Laboratoire de dynamique des réseaux
c/o CESTA
1 rue Descartes
75005 PARIS, FRANCE

and

* Laboratoire LIA
Ecole des Hautes Etudes en informatique
Université Paris V
45 rue des Saints Pères
75006 PARIS, FRANCE

** Conservatoire National des Arts et Metiers
292 rue Saint Martin
75003 PARIS, FRANCE

*** Laboratoire IAAI
Département d'informatique
ESIEE
89 rue Falguière
75015 PARIS, FRANCE

to appear in:
"Automata Networks in Computer Science, Theory and Applications"
F. Fogelman Soulie, Y. Robert, M. Tchuente Ed.
Manchester University Press

July 1986

Chapter 7

AUTOMATA NETWORKS AND ARTIFICIAL INTELLIGENCE

1. INTRODUCTION

At the very beginnings of Artificial Intelligence (AI), one could find two main goals, which originally seemed to be considered as two aspects of only one question: design "intelligent" machines and understand human "intelligence". J. von Neumann [59] wanted a theory to establish the logical differences -if any- between "natural" and "artificial" automata. Mc Culloch and Pitts [54] had shown how formal neurons, similar to the nervous cells - the neurons- could be used to achieve the same computational performances as a Turing machine. The question of how to make them LEARN to compute was then raised by Rosenblatt [69], Minsky and Papert [56] who tried to build a learning machine, the "perceptron", based on such components. The issue in these early studies was very clear: would it be possible to build artificial systems capable of intelligence or, stated in other words, are people inherently smarter than machines? Perceptrons served as the prototype for all such systems ... and battle field for the controversy!

The first results could at first seem rather deceiving while it is almost trivial to make neuron-based machines learn easy things (such as classify objects into two categories), designing general purpose learning machines appeared out of the range of the same techniques. As Rosenblatt states it: perceptron «does not generalize well to similar forms occurring in new positions in the retinal field, and its performance in detection experiments, where a familiar figure appears against an unfamiliar background, is apt to be weak». These "limitations" were made clear by Minsky-Papert [56] in their book which almost stopped any further research along these lines.

However, the ideas around the perceptron were used later on in the first developments of pattern recognition by Duda Hart [17] and Nilsson [60]. But in the 70's, AI mainly developed around tools based on logic [11]. Its achievements during this decade are amazing, and we understand today a lot more about our cognitive processes, but still people are much better than machines at certain tasks: understanding scenes or speech, playing chess or learning .

During the 70's, the development of automata theory [16,19,26,66] and, more recently, work in statistical physics [45,46,61] has led to re-consider these

During the 70's, the development of automata theory [16,19,26,66] and, more recently, work in statistical physics [45,46,61] has led to re-consider these early models. At the same time, work on associative memories [29,49,57,58] in connection with brain theory [18], has shown that the design of parallel architectures [35,36,43] might be inspired from the brain.

A new line of research based on automata networks -connectionism [20, 21, 22,24,25,28,38,40] or parallel distributed processing [70]- is now developing trying to push the limits imposed by the perceptron limitation theorem. More sophisticated machines have been designed -the Boltzmann machine [1,37,74,75], the Hierarchical Learning Machine [50,51], layered feed-forward nets [63,70]...- that have already proved that general learning procedures could achieve what the perceptron could not. The applications of these new techniques cover many domains: vision [6, 8, 9, 10,30,55,62,73], cognitive science [3,4,53,71,72], language [68,76,79], learning [28, 51, 63, 70,41,42]..

We will review in this chapter the different techniques used for learning by automata networks. the linear associator [48,49], the perceptron [56,69], the threshold networks [45] and the multi-layered networks [50,51,70,76] We will show how the dynamical behavior of an automata network can be used to represent knowledge and realize associative memories, pattern recognition, automatic categorization, structure extraction ...

2. ADAPTIVE MACHINES

2.1. The threshold automaton.

The first works on adaptive machines have been based on the so-called FORMAL NEURON introduced by McCulloch and Pitts [54]. The formal neuron is also refered to as THRESHOLD AUTOMATON [32] or linear threshold function (fig1).

It is an automaton, i.e. a mapping f: $\mathbf{E}^n \rightarrow \mathbf{E}$, where $\mathbf{E}$ is the state space, (see [32,67]). As such, it is interpreted as a cell which receives n inputs $x_1,x_2,...x_n$, taking their value in $\mathbf{E}$, and sends one output S. This output is computed as: $S = f(x_1,x_2,...x_n)$, i.e by performing a weighted sum of the inputs, the output being 1 if this sum exceeds a threshold θ, and 0 otherwise.

$$S = \begin{cases} 1 & \text{if } \sum_j W_j x_j \geq \theta \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

We will indifferently use this definition or the following, which can obviously be interchanged without any difficulty

$$S = \begin{cases} 1 & \text{if } \sum_j W_j x_j \geq \theta \\ -1 & \text{otherwise.} \end{cases} \qquad (2)$$

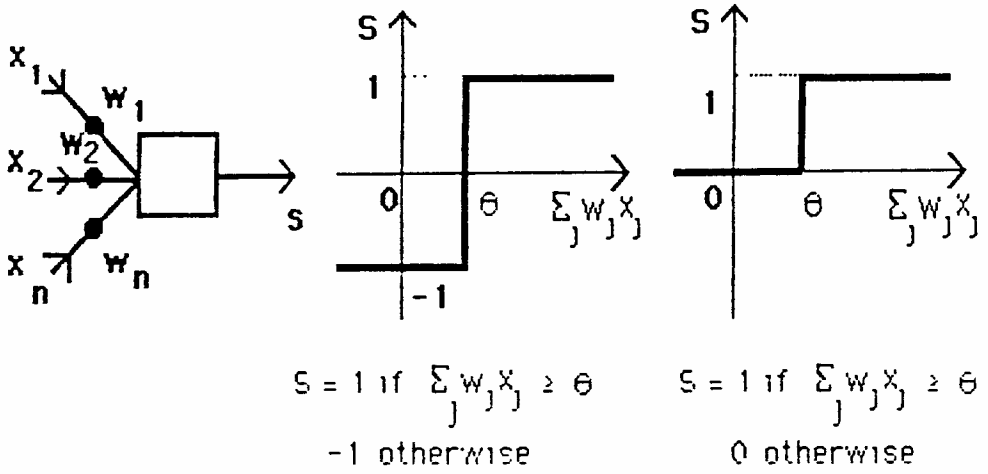


Figure 1: Threshold automaton
The figure shows an automaton (left) which receives inputs $x_1,...,x_n$ along connections with weights $W_1,...,W_n$, computes its state S and sends it as its output S may be computed using state space $\mathbb{E} = \{-1,1\}$ (middle) or $\mathbb{E} = \{0,1\}$ (right)

The W_j are called the (connection) WEIGHTS of the automaton, θ is called its THRESHOLD. In case (1), $\mathbb{E}=\{0,1\}$, in case (2), $\mathbb{E}=\{-1,1\}$.

2.2. Learning the weights of a threshold automaton.

In this context, learning is just the problem of correctly adapting the weights of a linear threshold function. One of the first algorithm that could learn the weights was the PERCEPTRON learning rule proposed by Rosenblatt in 1957 [69]. However, a number of other learning rules are known [49] which are generally expressed in terms of a minimization problem of a particular cost function We will describe the Widrow-Hoff rule [81] also called Least Mean Square algorithm or ADALINE (which stands for ADAptive LINear Element).

Assume that we want to teach a linear threshold automaton to classify a set of pattern vectors $\mathbf{X}_k$, k=1,...,m, elements of $\mathbb{E}^n$, into two classes C^+ and C^- The class of pattern $\mathbf{X}_k$ (vectors and matrices will be printed in bold in the following, scalars in plain text) is labeled by a variable Y_k which can take one of the two values +1 for C^+ and -1 for C^-. The desired input-output mapping is thus given by the sequence of pairs $(\mathbf{X}_1,Y_1)$, $(\mathbf{X}_2,Y_2)$, $(\mathbf{X}_k,Y_k)$, ... $(\mathbf{X}_m,Y_m)$.

The mapping of a linear threshold function -(1) and (2)- can be defined, using a DISCRIMINANT FUNCTION $A(\mathbf{x})$

$$A(\mathbf{x}) = \sum_j W_j x_j \qquad (3)$$

or using vector notation:

$$A(\mathbf{x}) = \mathbf{W}^t \mathbf{x} \tag{4}$$

where $\mathbf{W}^t$ is the transposed weight vector. An elementary classification can be implemented by assigning $\mathbf{x}$ to C^+ if $A(\mathbf{x}) \geq 0$, and to C^- otherwise. The effective output of the classifier for an input pattern $\mathbf{x}$ will then be:

$$S = \mathbf{1}[A(\mathbf{x})] \tag{5}$$

where function $\mathbf{1}$ has value +1 if its argument is positive, and -1 otherwise. In the following, we will consider that $\mathbf{x}$ contains an additional coordinate x_0 whose value is always -1 and whose corresponding weight is interpreted as the threshold. Note that equation $A(\mathbf{x})=0$ defines a decision surface which is in our case a HYPERPLANE, since $A(\mathbf{x})$ is linear

Learning the weight vector can be viewed as minimizing a cost function $C(\mathbf{W})$ Different choices can be made for C, each leading to a particular learning rule One of the most widely used criterion is the mean squared error

$$C(\mathbf{W}) = 1/(2m) \sum_k (\mathbf{W}^t.\mathbf{X}_k - Y_k)^2 \tag{6}$$

where m is the number of patterns and factor 2 is introduced for notation convenience.

This criterion can be minimized by various techniques which will be extensively studied in the following sections We will only consider here the simplest method, namely the GRADIENT DESCENT procedure. The basic idea of gradient descent is the following Take an arbitrary weight vector $\mathbf{W}_0$, and compute the gradient vector $\text{grad}_{\mathbf{W}}[C(\mathbf{W}_0)]$ The next value $\mathbf{W}_1$ is obtained by moving away from $\mathbf{W}_0$ along the direction of steepest descent which is just the negative of the gradient. This process is repeated until convergence, giving the following general formulation:

$$\mathbf{W}_{h+1} = \mathbf{W}_h - \lambda_h \, \text{grad}_{\mathbf{W}}[C(\mathbf{W}_h)] \tag{7}$$

Here λ_h is a small positive factor that determines the step size towards the minimum. Since in our case $\text{grad}_{\mathbf{W}}[C(\mathbf{W})] = \sum_k [A(\mathbf{X}_k) - Y_k] \mathbf{X}_k$ we obtain the following learning rule.

$$\mathbf{W}_{h+1} = \mathbf{W}_h - \lambda_h \sum_k [A(\mathbf{X}_k) - Y_k] \mathbf{X}_k \tag{8}$$

It can be shown that this process converges if λ_h is chosen "sufficiently small" to avoid overshots [17,49]

We can use the linearity property of the gradient operator to derive a simpler algorithm called STOCHASTIC GRADIENT, which is essentially a noisy version of the original gradient descent. First, an infinite time sequence of pairs $(\mathbf{X}_k,Y_k)$ is built by repeatedly presenting the original sequence of patterns. At time h, a pattern pair $(\mathbf{X}^h,Y^h)$ is presented and an update of the weight vector $\mathbf{W}$ is performed according to the rule:

$$\mathbf{W}_{h+1} = \mathbf{W}_h - \varepsilon_h [\, A(\mathbf{X}^h) - Y^h \,] \mathbf{X}^h \qquad (9)$$

where h is now a time index. This is known as the WIDROW-HOFF RULE.

If a precise solution is not required, λ_h can be chosen constant, in which case the weight vector fluctuates around the optimal solution. These fluctuations increase with λ_h Updating the network to add new information is easy since this adaptation rule proceeds continuously. In that case, the learning procedure minimizes the AVERAGED quadratic error

A number of other criterion functions can be chosen that are suitable for gradient descent [17,49,77], including the so-called "perceptron criterion":

$$C(\mathbf{W}) = -1/2 \sum_k Y_k . \mathbf{W}^t . \mathbf{X}_k \qquad (10)$$

where the sum is taken over the MISCLASSIFIED PATTERNS only. The resulting learning rule is simply:

$$\begin{aligned} &\mathbf{W}_{h+1} = \mathbf{W}_h + \varepsilon_h Y^h . \mathbf{X}^h && \text{iff } \mathbf{X}^h \text{ is misclassified} \\ &\mathbf{W}_{h+1} = \mathbf{W}_h && \text{otherwise} \end{aligned} \qquad (11)$$

The parameter λ_h influences the length of the final weight vector but has no effect on its direction The rule can be rewritten.

$$\mathbf{W}_{h+1} = \mathbf{W}_h - \lambda_h (S^h - Y^h) \mathbf{X}^h \qquad (12)$$

where S^h is the actual output when the input $\mathbf{X}^h$ is presented. This formula can be viewed as a DISCRETE version of the Widrow Hoff rule.

Unlike the Widrow Hoff rule, the Perceptron learning rule always finds a solution, provided that such a solution exists. Unfortunately when no exact solution exists, i e when the classes are not linearly separable, it does not even produce an approximate solution. The Widrow Hoff rule is known to give a good approximate solution in both the separable and the non-separable cases [17,77] The Widrow Hoff rule has been extensively used in signal processing for implementing adaptive filters [44], or in pattern recognition for the synthesis of linear (or polynomial) classifiers.

In section 3, a multidimensional version of the least square algorithm will be developed using the theoretical framework of matrix algebra. It will be applied to the problem of associative recall.

2.3 The perceptron

A PERCEPTRON **P** is a learning machine based on an automata network (fig 2) made of:

• a RETINA or set of cells where the input to the machine is displayed. Let $\mathfrak{E}$ be the retina cells space, we will use $\mathfrak{E} \subset \mathbb{R}$ for example, with $\mathfrak{E}=\{-1,1\}$, each cell can be "on" (x=1) or "off" (x=-1); with $\mathfrak{E}=\{0,1,...,k\}$ we can code the gray levels of each pixel in an image.

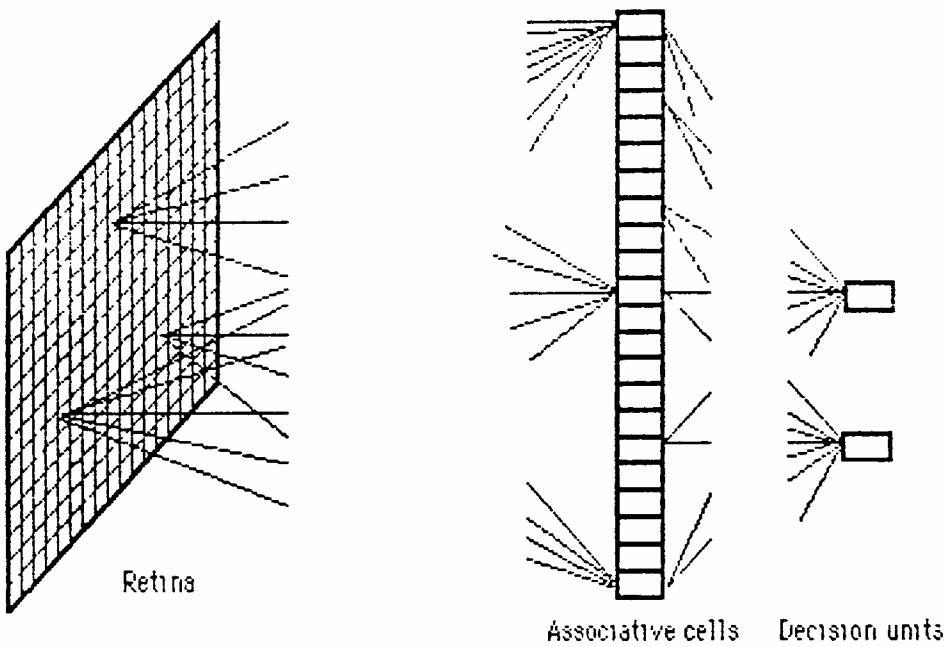


Figure 2: Perceptron.
The figure shows a perceptron with a 2-dimensional retina, and 2 decision units.

• a layer of ASSOCIATIVE cells (or FEATURE EXTRACTORS): each cell is an automaton with transition function ϕ_i defined on the retina: $\phi_i: \mathfrak{E}^n \rightarrow \{-1,1\}$, where n is the number of retina cells. Function ϕ_i will be said of CONNECTIVITY $k \leq n$ if ϕ_i depends on only k variables in $\mathfrak{E}^n$.

$$\exists i_1,...,i_k \in \{1,...,n\} \ \forall x,y \in \mathfrak{E}^n, \ (x_j=y_j, \forall j \in \{i_1,...,i_k\}) \Rightarrow \phi_i(x) = \phi_i(y)$$

• a layer of DECISION units (usually one) which are threshold automata Ψ

$$\Psi(x) = \begin{cases} 1 & \text{if } \sum_i W(\phi_i)\,\phi_i(x) \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

which we denote $\Psi = \mathbb{1}[\sum_i W(\phi_i)\,\phi_i - \theta]$, where the $W(\phi_i)$ are the CONNECTION WEIGHTS of the associative cell i to the decision unit and θ the THRESHOLD of the decision unit.

Perceptrons are interesting because they can LEARN how to compute a predicate: let **P** be a perceptron structure (i.e. a set of retina cells, associative cells and decision units), then a predicate Ψ is COMPUTABLE by **P** if there exists a set of functions $\phi_1,\ldots,\phi_k$, weights $W^*(\phi_1),\ldots,W^*(\phi_k)$ and a threshold θ such that $\Psi = \mathbb{1}[\sum_i W^*(\phi_i)\,\phi_i - \theta]$. In this case, Minsky-Papert [56] have shown that weights $W(\phi_i)$ such that $\Psi = \mathbb{1}[\sum_i W(\phi_i)\,\phi_i - \theta]$ can be LEARNT by presenting examples Let $\mathfrak{X}$ be a family of "examples", i.e. of vectors $\mathbf{x}$ in $\mathbb{E}^n$, we will present a sequence of examples in the family in such a way that all examples are shown infinitely often We then have the following theorem:

Theorem (perceptron convergence theorem)
Let Ψ be a predicate computable by a perceptron **P** and functions ϕ_i. Then there exists a learning procedure which converges in a finite number of steps towards a set of weigths $[W(\phi_i)]_i$ and threshold θ such that:

$$\Psi = \mathbb{1}[\sum_i W(\phi_i)\,\phi_i - \theta].$$

This learning procedure is given by:

1● choose an initial value: $[W_0(\phi_i)]_i \qquad \forall\, i$

2● at step k choose an "image" $\mathbf{x}(k)$ in $\mathbb{E}^n$ on the retina and compute $W_k(\phi_i), \forall\, i$, by the following algorithm.

if $\left[\left(\Psi(\mathbf{x}(k))=1\right)$ *and* $\left(\sum_i W_k(\phi_i)\,\phi_i(\mathbf{x}(k)) < 0\right)\right]$ *then*

$$W_{k+1}(\phi_i) = W_k(\phi_i) + \phi_i(\mathbf{x}(k))$$

else

if $\left[\left(\Psi(\mathbf{x}(k))=-1\right)$ *and* $\left(\sum_i W_k(\phi_i)\,\phi_i(\mathbf{x}(k)) \geq 0\right)\right]$ *then*

$$W_{k+1}(\phi_i) = W_k(\phi_i) - \phi_i(\mathbf{x}(k));$$

$k = k+1,$

Remarks:

1● This algorithm just makes a change on $W_k(\phi_i)$ if pattern $\mathbf{x}(k)$ is misclassified We have seen before in §2.2 that this rule can be viewed as a gradient descent method (see (11)) and written as a discrete version of the Widrow Hoff rule (12)

2• We have presented the learning algorithm in the case where $\theta = 0$. It is always possible to make this assumption by adding an auxiliary cell, which state is constantly set equal to -1 (§2.2). Its transition function is then $\phi = -1$ and Ψ can be rewritten as: $\Psi = \mathbb{1}[\sum_{i=1,\dots,k} W(\phi_i)\,\phi_i + \theta\phi] = \mathbb{1}[\sum_{i=1,\dots,k+1} W(\phi_i)\phi_i]$. The threshold θ is then learnt as an extra weight along the same algorithm. We will usually assume 0 threshold in the following, by eventually adding this auxiliary cell.

3• The limit weights $W_\infty(\phi_i) = \lim_{k\to\infty} [W_k(\phi_i)]$ may be different from the $W^*(\phi_i)$ For a computable predicate, there usually exists an infinite number of weight vectors to compute it. This is easily seen on figure 3: slightly moving around the line with equation x+y-3/2=0 does not change the relative position of points -1-1, -11, 1-1 and 11 with respect to the line and thus still provides a function which computes function "and".

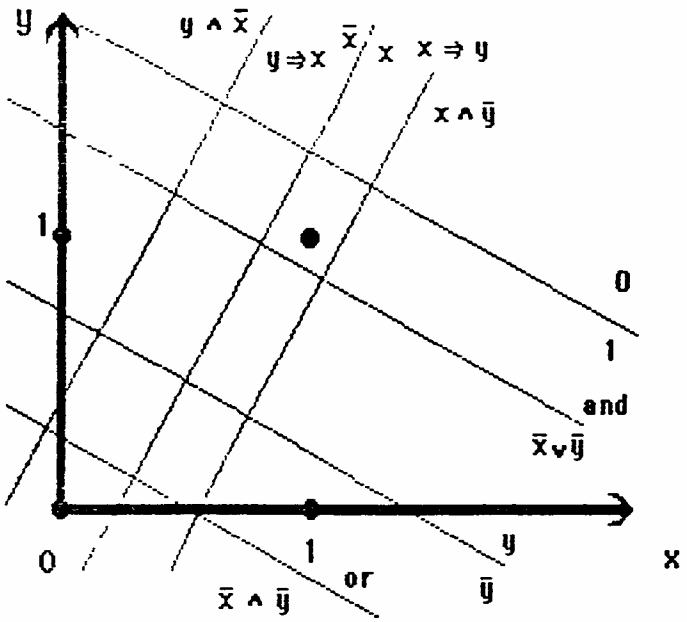


Figure 3: Representation of boolean functions as threshold functions.
The figure shows the representation of the boolean functions in 2 variables, as threshold functions. For example, function AND is represented by $\mathbb{1}[x+y-3/2]$: the figure shows the line with equation x+y-3/2=0. It is easy to check that points 00, 01 and 10 are on the side of this line where $x+y-3/2 < 0$, and thus the value of the threshold function -and of AND- is 0 Point 11 is on the other side. the value of the threshold and the AND functions in 11 is 1.

4• The learning procedure is convergent for any choice of the initial condition $[W_0(\phi_i)]_i$ and image sequence $\mathbf{x}(k)$, which visits each example infinitely often

5• If $C^+ = \{\mathbf{x} \in \mathbb{E}^n: \Psi(\mathbf{x}) = 1\}$ and $C^- = \{\mathbf{x} \in \mathbb{E}^n: \Psi(\mathbf{x}) = -1\}$, then classes C^+ and C^- are said to be SEPARABLE by Ψ with the perceptron P. The perceptron can thus be viewed as a machine for automatic classification: presented

with "examples" of class C^+ and examples of class C^-, it learns the weights $W_\infty(\phi_i)$, which will further allow it to GENERALIZE to any example of class C^+ or C^-. This means that, after learning, $\forall x \in C^+, \Psi(x) = 1$ and $\forall x \in C^-$, $\Psi(x) = -1$, even if such x has not been presented as an example in the learning process. It is this learning ability which is the main interest of perceptron-like machines, but the main problem is to decide whether a given predicate is computable by a given perceptron (i.e. the existence problem of preliminary weights $W^*(\phi_i)$).

6• Classification in more than two classes just requires to use more than one decision unit.

We will say that a predicate $\Psi\ \mathcal{E}^n \to \{-1,1\}$ is of ORDER k [56] if k is the smallest integer for which there exists a family of mappings ϕ_i: $\mathcal{E}^n \to \{-1,1\}$ such that $\forall$ i, ϕ_i is of connectivity $\leq$ k, and $\Psi = \mathbb{1}[\sum_i W(\phi_i)\ \phi_i - \theta]$ Of course the order k of Ψ is always less than its connectivity

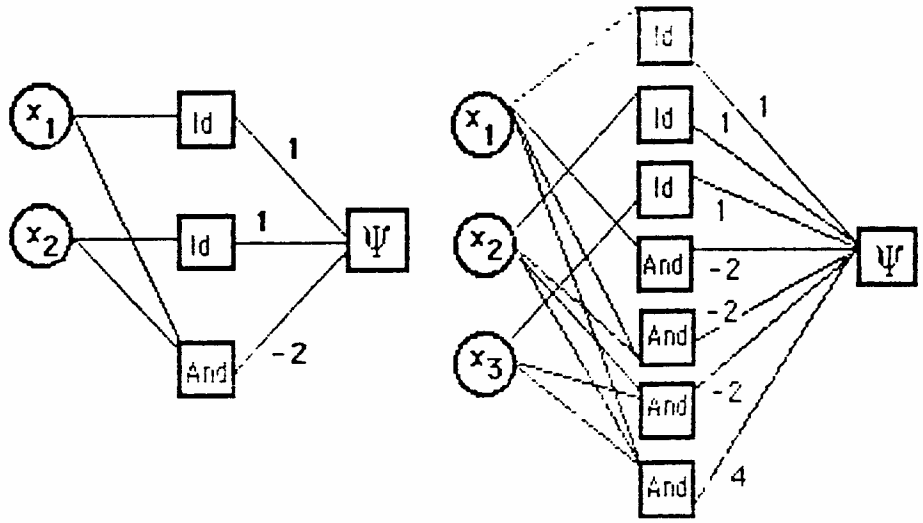


Figure 4: Computing the XOR and the n-parity predicate with a perceptron. The figure shows perceptrons which have been designed to compute the XOR (left) and the 3-parity predicate (right). For example (in space $\mathcal{E} = \{0,1\}$).

$$XOR(x_1,x_2) = x_1 + x_2 - 2AND(x_1,x_2)$$

Examples:

1• With $\mathcal{E} = \{0,1\}$, 2 retina cells, 2 associative cells and one decision unit, we can realize all 16 boolean mappings in 2 variables by a perceptron. For 2 mappings XOR and $\Leftrightarrow$, the associated perceptron will be of order 2, for the remaining 14 of order 1 only (see fig. 3). This means that 14 boolean mappings in 2 variables can be realized by a threshold automaton [56].

For example: $\Psi(x,y) = x \vee y = OR(x,y) = \mathbb{1}[x+y-1/2]$

2• The n-arity predicate. $\Psi(x) = 1$ if $card\{i(1,...,n): x_i=1\}$ is odd, is of order $card(\mathbf{E})$ for example, with $\mathbf{E} = \{0,1\}$, Ψ = XOR, which is of order 2.

It has been shown [56,75] that the perceptron can only compute problems of first order, unless ad-hoc connections specific to the problem are built between the retina and the associative units (see fig. 4). We will see that problems of higher order can be solved by networks with more than one layer of weights, but then the learning algorithm must specify how to modify the internal weights (this is the classical CREDIT ASSIGNMENT PROBLEM).

3. LINEAR LEARNING

Work on perceptrons had shown that it was possible to design a machine which was capable of learning to automatically categorize, i.e. to associate with various input patterns X_i their class Y_i. The use of threshold functions by the decision units was central in this approach. More recently, LINEAR models have been studied and the theory of such systems is now well developped (see Kohonen [47,48,49]) and has been used for a number of applications. In this paragraph, we will give an outline of this theory and refer the reader to Kohonen [49] for more details.

3.1. The model.

We have defined elsewhere [67] an automata network as a mapping $F: \mathbf{E}^n \rightarrow \mathbf{E}^n$. This model can be represented as a "black box" type model (see fig.5): we replicate $\mathbf{E}^n$ twice, each version is interpreted as a network or "layer" and mapping F allows to define the state of the second layer in terms of the state of the first layer. More precisely, if $\mathbf{x}(t)$, t=1,2, is the state vector of layer t and j is a cell in the second layer, then j will have state:

$$x_j(2) = F_j[\mathbf{x}(1)]$$

Then the dynamics on F can be defined as usual, by feeding-back the output of the second layer to the first one.

This representation allows us to extend the notion of automata network to mappings $F: \mathbf{E}^n \rightarrow \mathbf{E}^p$, with $n \neq p$ (see fig.5). The connections between the two layers defining F will be endowed with a WEIGHT $W_{ij} \in \mathbf{R}$. Let $\mathbf{W}=(W_{ij})$ be the weight matrix (if there is no connection between i and j, then by convention $W_{ij}=0$) (fig 5).

We want to MEMORIZE into the network the ASSOCIATIONS between m input patterns $X_i \in \mathbf{E}^n$, i=1...m, and m output patterns $Y_i \in \mathbf{E}^p$. Usually we will take $\mathbf{E} = \mathbf{R}$ or a finite space, {-1,1} for example. $\mathbf{X}$ will denote the nxm matrix with columns X_i and $\mathbf{Y}$ the pxm matrix with columns Y_i: X_{ji} is thus

FIGURE 5-1

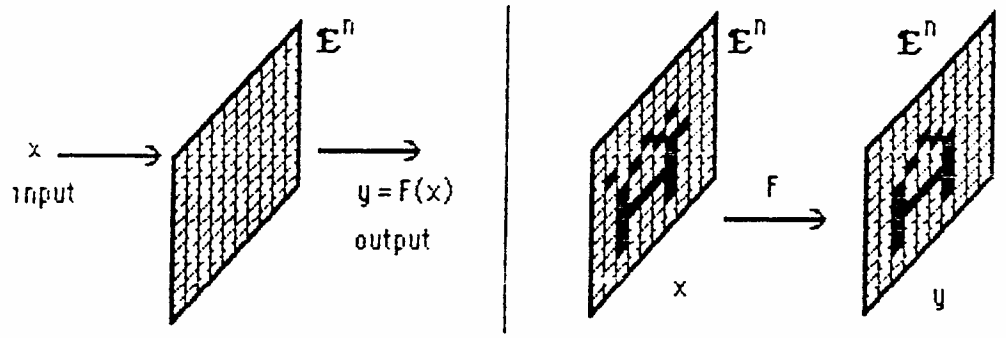


FIGURE 5-2.

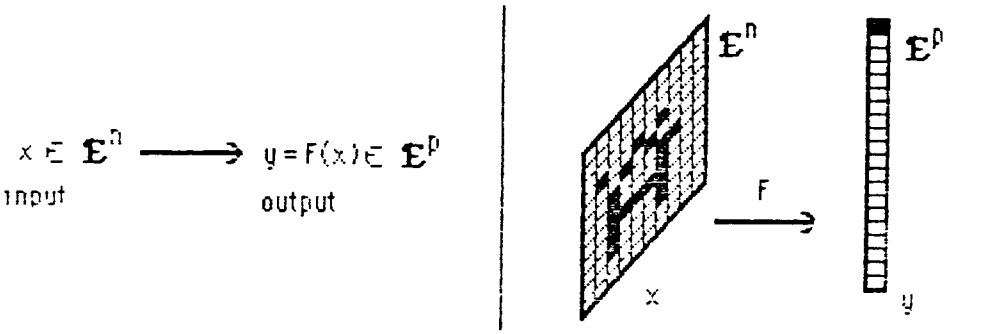


Figure 5: Automata network.
The figure shows an automata network (left) as a mapping F. $\mathfrak{E}^n \rightarrow \mathfrak{E}^n$ (Figure 5-1) or F. $\mathfrak{E}^n \rightarrow \mathfrak{E}^p$ (figure 5-2) and its equivalent (right) as a two-layered network. In the layered network representation, the input x to network F is the state of the input layer (we represented here this input as a noisy "A") and the output y is the state of the output layer (here y is a "pure" A in case of fig 5-1 and a code -1 for the first letter of the alphabet- in fig.5-2).

the state of automaton j in pattern $\mathbf{X_i}$. In the case where $\mathbf{X_i}=\mathbf{Y_i}$, for all i, we will say that the network is performing AUTO-ASSOCIATION (for example in pattern recognition, index searching ...), HETERO-ASSOCIATION otherwise (as in automatic classification).

The linear learning scheme assumes that these associations are realized by a LINEAR automata network: when presented with an input pattern $\mathbf{x}$, the network F computes an output pattern $\mathbf{y} = \mathbf{W}\mathbf{x}$. The desired associations will be encoded in the weight matrix $\mathbf{W}$ In order to store the desired associations, one thus has to make sure that: $\mathbf{W}\,\mathbf{X_i} = \mathbf{Y_i}$ i=1...m

i e to solve equation. $\mathbf{W}\,\mathbf{X} = \mathbf{Y}$ (13)

or in the case of auto-association. $\mathbf{W}\,\mathbf{X} = \mathbf{X}$ (14)

where $\mathbf{X}$ and $\mathbf{Y}$ are given.

Remark. in general the state space $\mathfrak{E}$ is finite, and thus equation (13) must be replaced by $\mathbf{1}[\mathbf{W}\,\mathbf{X}] = \mathbf{Y}$ (13')

where function $\mathbf{1}$ denotes thresholding: for example, if $\mathbf{E} = \{0,1\}$, $\mathbf{1}[u]=1$ if $u \geq 0$, 0 otherwise, if $\mathbf{E} = \{-1,+1\}$, $\mathbf{1}[u]=1$ if $u \geq 0$, -1 otherwise.

3.2. The generalized inverse matrix.

A formal solution to equation (13) or (14) is given by the generalized inverse matrix theory [13,65,78]. In the following, we will use only the pseudo-inverse [33] By definition, $\mathbf{X}^+$ is a PSEUDO INVERSE of matrix $\mathbf{X}$ iff.

1• $\mathbf{X}\mathbf{X}^+\mathbf{X} = \mathbf{X}$
2• $\mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+$
3• $\mathbf{X}\mathbf{X}^+$ and $\mathbf{X}^+\mathbf{X}$ are hermitian (if $\mathbf{X}$ is a real matrix, this just means that $\mathbf{X}\mathbf{X}^+$ and $\mathbf{X}^+\mathbf{X}$ are symmetric)

It has been shown [33,65] that for any matrix $\mathbf{X}$, there exists one and only one pseudo-inverse $\mathbf{X}^+$. $\mathbf{X}^+$ can be computed by

$$\mathbf{X}^+ = \lim_{d \to 0} (\mathbf{X}^t\mathbf{X} + d^2 \mathbf{I})^{-1} \mathbf{X}^t = \lim_{d \to 0} \mathbf{X}^t(\mathbf{X}\mathbf{X}^t + d^2 \mathbf{I})^{-1}$$

where $\mathbf{X}^t$ denotes the transpose of matrix $\mathbf{X}$, $\mathbf{I}$ the identity matrix, $\mathbf{Y}^{-1}$ the inverse of matrix $\mathbf{Y}$ and $d \in \mathbb{R}$.

Pseudo inverses can be used to solve matrix equations such as: $\mathbf{W}\mathbf{X}=\mathbf{Y}$ (13) where $\mathbf{X}$ and $\mathbf{Y}$ are given and $\mathbf{W}$ is unknown. It has been shown that:

Theorem equation (13) has an exact solution iff:

$$\mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{Y} \qquad (15)$$

and in this case, there exists an infinite number of solutions given by

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^+ + Z(\mathbf{I} - \mathbf{X}\mathbf{X}^+) \qquad (16)$$

where Z is an arbitrary matrix with the same dimensions as $\mathbf{W}$

Remarks

• condition (15) is clearly necessary, since

$$\mathbf{Y} = \mathbf{W}\mathbf{X} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{W}\mathbf{X}\mathbf{X}^+\mathbf{X} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{W}\mathbf{X} = \mathbf{Y}$$

• if the columns $\mathbf{X}_i$ of $\mathbf{X}$ are linearly independent, then $\mathbf{X}^+ = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t \Rightarrow \mathbf{X}^+\mathbf{X} = \mathbf{I} \Rightarrow \mathbf{Y}\mathbf{X}^+\mathbf{X} = \mathbf{Y}$ and condition (15) is satisfied for all $\mathbf{Y}$. In this case the solution $\mathbf{W} = \mathbf{Y}\mathbf{X}^+$ can be computed easily by computing only $(\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t$

Theorem. among all solutions to equation (13), $W = YX^+$ is of minimum quadratic norm.

PROOF. $\| YX^+ + Z(I - XX^+)\|^2 = tr[YX^+ + Z(I - XX^+)]^t [YX^+ + Z(I - XX^+)]$

$$= \| YX^+\|^2 + \| Z(I - XX^+) \|^2 + 2\, tr[Z^t Y X^+] - 2\, tr[(X^+)^t X^t Z^t Y X^+]$$

where tr(A) is the trace of matrix A.

By using the classical properties. $tr(PQ) = tr(QP)$, $(X^+)^+ = X$ and $XX^t(X^+)^t = X$, we show that.

$$tr[(X^+)^t X^t Z^t Y X^+] = tr[Z^t Y X^+ (X^+)^t X^t] = tr[Z^t Y X^+]$$

and thus $\| YX^+ + Z(I - XX^+)\|^2 = \| YX^+ \|^2 + \| Z(I - XX^+) \|^2$.

Matrix $W = YX^+$ (17)

is thus the solution to equation (13) of minimum quadratic norm. □

In the case of auto association, the solution to equation (14) of minimum quadratic norm is: $W = XX^+$ (17')

If condition (15) is not satisfied, then equation (13) has no solution, but it can be shown that matrices W for which the quadratic norm $\| WX-Y \|$ is minimum are just those given by equation (16), and of course, among those, matrix $W = YX^+$ is of minimum quadratic norm. Hence, when (15) is not satisfied, (16) gives the general expression of approximate solutions to (13) and (17) the best approximate solution.

Usually, the dimension of the state space S^n (i.e. the number of automata in the network) is large (for example, in an image [49], n=3024 -number of pixels- and |S| = 8 -number of gray levels-), and thus presumably $m \ll n$ and the assumption of linear independence of the m patterns X_i is thus not very restrictive In the case $m<n$, the operator XX^+ is the orthogonal projection on the vector space $\mathfrak{L}$ spanned by patterns $X_1,...,X_m$ and $I-XX^+$ the orthogonal projection on $\mathfrak{L}^\perp$. In the case of auto association, W being just XX^+, mapping F is then nothing more than the projection on $\mathfrak{L}$

3.3. Computational techniques.

Exact methods to compute X^+ have been derived : they usually are very time consuming, since they involve computations of inverses of matrices. However, every computer math-library always includes its efficient

program to compute the pseudo-inverse. We will present here various methods

The first one is a recursive algorithm, due to Greville [33], which does not need to invert matrices and may be implemented by successively presenting the examples $X_1,\ldots,X_m$ let us denote $\mathfrak{X}_k = (\mathfrak{X}_{k-1}\ X_k)$ a matrix with k columns, where $\mathfrak{X}_{k-1}$ is the submatrix of X formed with its k-1 first columns and X_k is the k-th column of X. We thus have: $X = \mathfrak{X}_m$. Greville's theorem states that X^+ can be computed in m steps by:

$$\mathfrak{X}_k^+ = \begin{bmatrix} \mathfrak{X}_{k-1}^+ (I - X_k p_k^t) \\ p_k^t \end{bmatrix} \qquad (18)$$

$$\text{and } p_k = \frac{(I-\mathfrak{X}_{k-1}\mathfrak{X}_{k-1}^+)X_k}{\|(I-\mathfrak{X}_{k-1}\mathfrak{X}_{k-1}^+)X_k\|^2} \quad \text{if the numerator is non zero}$$

$$= \frac{(\mathfrak{X}_{k-1}^+)^t\mathfrak{X}_{k-1}^+X_k}{1+\|\mathfrak{X}_{k-1}^+X_k\|^2} \quad \text{otherwise}$$

As $\mathfrak{X}_1$ is just the first column X_1 of X we have.

$$\mathfrak{X}_1^+ = \begin{cases} (X_1^tX_1)^{-1}X_1^t & \text{if } X_1 \text{ is non zero} \\ 0 & \text{otherwise} \end{cases}$$

and of course $X^+ = \mathfrak{X}_m^+$

The complexity of Greville's algorithm is of order n^2m^2, which may be a problem in practical situations We will thus try to compute W directly instead of computing X^+. A slight adaptation of Greville [33] allows to recursively compute W By conserving the same notations for $\mathfrak{X}_k$ and evident notations for $\mathfrak{Y}_k$ and denoting $W_k = \mathfrak{Y}_k\mathfrak{X}_k^+$, we have:

$$W_k = W_{k-1} - (W_{k-1}X_k - Y_k)p_k^t \qquad (19)$$

where p_k is the same vector as in Greville

At iteration step k, W_k is the solution to equation $W\mathcal{X}_k = \mathcal{Y}_k$ with minimum norm This matrix thus encodes the associations between input patterns $X_1,...,X_k$ and output patterns $Y_1,...,Y_k$.

If the columns of X are independent, the computation of W becomes simpler. We then have

$$W_k = W_{k-1} - (W_{k-1}X_k - Y_k)[\mathcal{X}_k^t / \|\mathcal{X}_k\|^2] \quad \text{if } \mathcal{X}_k \neq 0 \qquad (20)$$
$$W_{k-1} \quad \text{otherwise}$$

with

$$\mathcal{X}_k = S_{k-1} X_k \qquad \mathcal{X}_1 = X_1 \qquad (20')$$
$$S_k = S_{k-1} - \mathcal{X}_k \mathcal{X}_k^t / \|\mathcal{X}_k\|^2 \qquad S_0 = I$$

The $\mathcal{X}_k$ can also be computed through the Gram-Schmidt orthogonalization process

$$\mathcal{X}_1 = X_1 \qquad (20'')$$
$$\mathcal{X}_k = X_k - \sum_{i=1...k-1} \langle X_k , \mathcal{X}_i \rangle \cdot \mathcal{X}_i / \|\mathcal{X}_i\|^2 \quad k=2,...,m$$

where the summation runs on all non zero $\mathcal{X}_i$. This result is similar to Pyle [64] More details can be found in Kohonen [49].

3.4. Approximated computation

The preceding algorithms may imply a huge computational burden if the dimension n of the patterns is large Moreover, simulations show that the linear learning scheme is very fault tolerant (see below). even approximate values of matrix W would do. Therefore, approximation methods are used to estimate W instead of computing the exact matrix YX^+, we compute a matrix W which minimizes the error E between the output value WX and the desired value Y Of course many error functions can be used, the most classical being the quadratic norm which requires the minimization of.

$$E = \sum_{k=1...m} \sum_{i=1...p} (\sum_{j=1...n} W_{ij} X_{jk} - Y_{ik})^2 \qquad (21)$$

This is a classical minimization problem, which can be solved by using gradient methods $\text{grad}_W E = 2(WX - Y)X^t$ is the gradient of E with respect to W We have presented various methods in §2.2 for example, the gradient method with increment (λ_k) is given by:

$$W_1 \text{ arbitrary}$$
$$W_k = W_{k-1} - \lambda_k(W_{k-1} X - Y)X^t \qquad (22)$$

This procedure has been shown to converge provided the λ_k are chosen such that

$$\sum_{k=1}^{\infty}\lambda_k = +\infty \text{ and } \sum_{k=1}^{\infty}\lambda_k^2 < \infty$$

This is satisfied in particular for $\lambda_k = \lambda_1/k$, but convergence may be very slow.

One weakness, for our purpose, of these gradient methods is that they require the full knowledge of matrices X and Y. We would prefer instead methods which allow the "knowledge" of the network to gradually evolve when new data are presented, without of course destroying previous memories In other words, we want to design a method which would allow to compute W_k in terms of W_{k-1}, by making the correction needed to take into account the observed error when retrieving the new pattern X_k by W_{k-1}. This can be done by using a correction term proportional to $(W_{k-1} X_k - Y_k)$, which is the discrepancy between the desired output and the output computed with W_{k-1}. This gives the following algorithm:

$$W_k = W_{k-1} - (W_{k-1} X_k - Y_k)c_k^t \qquad (23)$$

where c_k^t is a vector which will allow to adapt the weights to the new incoming pattern X_k If the columns X_k are linearly independent, then one possible choice is. $c_k = p_k$, and then, at each iteration step, matrix W_k is optimal for patterns $X_1, ,X_k$ and adding one more pattern just requires to iterate formula (23). This is just the case discussed in (19).

In the general case, going in the direction computed with $c_k = \lambda X_k$ decreases the error, since $(W_{k-1} X_k - Y_k)X_k^t$ is the gradient of the error E_k on the k-th pattern X_k

There also exist more methods, which do not require the optimality of matrix W_k at each iteration step In "stochastic" gradient methods, each step minimizes the error on one pattern only. One of the most famous of these stochastic methods is the Widrow-Hoff procedure [81]. patterns $X_1, ,X_m$ are introduced sequentially and repeatedly, until convergence To simplify notation, we will denote this sequence (X_1,Y_1), (X_2,Y_2), ...,(X_m,Y_m), (X_1,Y_1), ... by $(X^1,Y^1),(X^2,Y^2),..,(X^m,Y^m),(X^{m+1},Y^{m+1})$, ... At each step, a new pattern is presented and the error on this pattern is minimized. it is necessary to go in the opposite direction of the gradient of the error E_{i+1} on pattern x^{i+1}. This gives the following rule·

$$W^0 \text{ arbitrary} \qquad (24)$$
$$W^{i+1} = W^i - \lambda_1(W^i X^{i+1} - Y^{i+1})[X^{i+1}]^t$$

There is no existing proof of the convergence of this algorithm. However, in practice, it has been observed that with $\lambda_1=\lambda_1/1$, the algorithm usually converges to a solution. The problem of adding one pattern to the memory is quite easy, it is sufficient to start with $W^0 = W$ and iterate with the new pattern until convergence. As the starting matrix is probably close to the solution, the computing time will not be long before convergence.

3.5. Other models

Constrained gradients

It is sometimes useful to impose on the connection matrix some constraints which reflect properties of the information flows in the network For example, the connection weight of an element with itself, W_{ii}, could be bounded, or even cancelled, so that its influence would not overkill the others In this case, matrix W is constrained to have a bounded -or zero- diagonal and can be computed by gradient projection methods In the case of $W_{ii}=0$, there exist solutions to equation (13) of the form: $W=YX^+ + Z(I-XX^+)$.

Hebb law

In 1949, D Hebb [34] proposed his "reinforcement law" based on biological evidence any two neurons firing at the same time, when "shown" a given pattern, will have their connection strength reinforced. This is also related to the theory of learning by evolution of the synapse structure (see Changeux [14]). Stated in unformal fashion by Hebb, this rule has been repeatedly used to compute weights W_{ij}, through algorithms such as.

$$W_{ij} = W_{ij} + X_{ik}X_{jk} \qquad (25)$$

if pattern X_k is shown to the network. $X_{ik}=1$ if automaton i is on in pattern X_k, -1 otherwise. In the case of auto-association between patterns X_1, X_2, ..,X_m, algorithm (25) is successively applied to all patterns and the corresponding weights added Matrix W is thus given by $W = XX^t$ (26)
In the case of hetero-association between patterns (X_1,Y_1), (X_2,Y_2), ..., (X_m,Y_m), this rule can be extended. $W_{ij} = W_{ij} + Y_{ik}X_{jk}$ and $W = YX^t$

Note that, when patterns X_k are linearly independent and orthonormal, i.e. $\langle X_i,X_j\rangle=0$, $\forall\ i\neq j$, and $\|X_k\| =1$, then $X^+ = X^t$ and $W = YX^t$ is the solution of minimum norm of equation (13) Hence Hebb rule, based on biological evidence, can be considered as an approximated version of the theoretical results in section 3 2.

4. RETRIEVAL

4.1. Introduction.

After the learning session, during which the network connection matrix **W** has been computed, we can use the network to perform recognition tasks. The main interest of these learning techniques is that they lead to efficient retrieval even if the data are noisy or partial This will thus allow to

- classify patterns close to memorized items.
- retrieve memorized items from noisy or incomplete patterns.

that is to say, implement a function of ASSOCIATIVE MEMORY.

Various procedures have been designed for retrieval. remember that our learning procedure was based on a linear network, but, as can be expected, non linear retrieval rules may be used for improved performances

4.2. Linear retrieval.

We will now discuss, in the case of a linear retrieval process, the properties of resistance to noise depending on the model of noise which is applied to the inputs. we will derive the "best" choice of matrix **W** for various models of noise.

We have seen that the output of a linear network, with connection matrix **W**, when the input is some vector **x**, is $\mathbf{y} = \mathbf{W}\mathbf{x}$. If $\mathbf{x}=\mathbf{X}_k$, then **y** will be exactly $\mathbf{Y}_k$ If **x** is only close to some $\mathbf{X}_k$, then **y** will be close to $\mathbf{Y}_k$

suppose $\mathbf{W}=\mathbf{Y}\mathbf{X}^+$

then $\mathbf{y} = \mathbf{Y}\mathbf{X}^+\mathbf{x} = \mathbf{Y}\mathbf{X}^+[(\mathbf{X}\mathbf{X}^+)\mathbf{x}]$ (because $\mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+$, by the definition of the pseudo-inverse)

$$\Rightarrow \quad \mathbf{y} = \mathbf{Y}\mathbf{X}^+\mathbf{x}_{\mathfrak{L}}$$

where $\mathbf{x}_{\mathfrak{L}}$ denotes the projection of **x** onto the vector space $\mathfrak{L}$ spanned by the $\mathbf{X}_k$ (see §3.1)

$$\mathbf{x}_{\mathfrak{L}} = \sum_{i=1\,..\,m} \alpha_i \mathbf{X}_i$$

$$\Rightarrow \quad \mathbf{y} = \mathbf{Y}\mathbf{X}^+ \sum_{i=1..m} \alpha_i \mathbf{X}_i = \sum_{i=1...m} \alpha_i \mathbf{Y}\mathbf{X}^+\mathbf{X}_i = \sum_{i=1..m} \alpha_i \mathbf{Y}_i$$

But if **x** is close to $\mathbf{X}_k$, then α_k is large compared to the other α_i, and thus **y** is close to $\mathbf{Y}_k$

Simulations (see Kohonen [49]) show that results are better if the memorized patterns are close to mutually orthogonal, thus any preprocessing which tends to orthogonalize the patterns would lead to increased efficiency

The signal-to-noise ratio is improved after retrieval.

PROOF suppose $\mathbf{x} = \mathbf{X}_k + \mathbf{v}$, where $\mathbf{v}$ is some noise vector Then:

$$\mathbf{y} = \mathbf{Y}\mathbf{X}^+\mathbf{x} = \mathbf{Y}\mathbf{X}^+[(\mathbf{X}\mathbf{X}^+)(\mathbf{X}_k + \mathbf{v})] \quad \text{(because } \mathbf{X}^+\mathbf{X}\mathbf{X}^+ = \mathbf{X}^+\text{)}$$

$$\Rightarrow \quad \mathbf{y} = \mathbf{Y}\mathbf{X}^+[\mathbf{x}_{\mathfrak{L}} + \mathbf{v}_1]$$

where $\mathbf{v}_1$ denotes the orthogonal projection of $\mathbf{v}$ on $\mathfrak{L}$.

It has been shown [49], for auto-association, that if $\mathbf{v}$ is of constant length and has a direction uniformly distributed in $\mathbb{R}^n$ then the variance

$$\mathrm{Var}(\|\mathbf{W}\mathbf{x} - \mathbf{X}_k\|) = (m/n)\|\mathbf{x} - \mathbf{X}_k\|$$

Hence, if $m<n$, the noise is damped in the orthogonal projection. In the case of hetero association, performances are usually quite good, but the output noise-to-signal ratio may eventually become large even for small input noise-to-signal ratio, when m/n increases and the determinant of $\mathbf{X}\mathbf{X}^t$ goes to 0 [78]

More generally, suppose $\mathbf{x} = \mathbf{X}_k + \mathbf{v}$, then $\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{W}(\mathbf{X}_k + \mathbf{v}) = \mathbf{W}\mathbf{X}_k + \mathbf{W}\mathbf{v}$. Hence, to improve the efficiency in terms of signal-to-noise ratio, we can try to find the matrix $\mathbf{W}$ which minimizes the dispersion of the remaining noise at the output $\mathbf{W}\mathbf{v}$

If $\mathbf{W}$ is of the form of equation (16) and $\mathbf{v}$ is of zero mean, then the dispersion of the remaining noise at the output is

$$\mathrm{Var}(\mathbf{W}\mathbf{v}) = E[\mathbf{W}\mathbf{v}\,(\mathbf{W}\mathbf{v})^t] = E[\mathbf{W}\mathbf{v}\,\mathbf{v}^t\mathbf{W}^t] = \mathbf{W}\,E[\mathbf{v}\,\mathbf{v}^t]\,\mathbf{W} = \mathbf{W}\,\boldsymbol{\Sigma}\,\mathbf{W}^t$$

where $\boldsymbol{\Sigma}$ is the dispersion matrix of noise $\mathbf{v}$

$$\Rightarrow \quad \mathrm{Var}(\mathbf{W}\mathbf{v}) = [\mathbf{M}_0 + \mathbf{Z}\,\mathbf{M}_1]\,\boldsymbol{\Sigma}\,[\mathbf{M}_0 + \mathbf{Z}\,\mathbf{M}_1]^t \text{ with } \mathbf{M}_0 = \mathbf{Y}\mathbf{X}^+$$

$$= [\mathbf{M}_0 + \mathbf{Z}\,\mathbf{M}_1]\,\boldsymbol{\Sigma}\,[\mathbf{M}_0{}^t + \mathbf{M}_1{}^t\mathbf{Z}^t] \text{ and } \mathbf{M}_1 = \mathbf{I} - \mathbf{X}\mathbf{X}^+$$

$$= \mathbf{M}_0\boldsymbol{\Sigma}\,\mathbf{M}_0{}^t + \mathbf{M}_0\boldsymbol{\Sigma}\mathbf{M}_1\mathbf{Z}^t + \mathbf{Z}\,\mathbf{M}_1\,\boldsymbol{\Sigma}\,\mathbf{M}_0{}^t + \mathbf{Z}\,\mathbf{M}_1\,\boldsymbol{\Sigma}\,\mathbf{M}_1\mathbf{Z}^t$$

$$= \mathbf{M}_0\boldsymbol{\Sigma}\,\mathbf{M}_0{}^t + \mathbf{A}\mathbf{Z}^t + \mathbf{Z}\mathbf{A}^t + \mathbf{Z}\mathbf{B}\mathbf{Z}^t$$

with evident notations for **A** and **B**. Note that $\mathbf{M_1}$ and **B** are symmetric. Let V_{ij} be the coefficient in line i and column j of matrix Var(**Wv**). We have

$$V_{ij} = (\mathbf{M_0 \Sigma M_0}^t)_{ij} + \sum_{k=1\ldots n} A_{ik}(Z^t)_{kj} + \sum_{k=1\ldots n} Z_{ik} A_{jk} + \sum_{k=1\ldots n} Z_{ik} \sum_{\ell=1\ldots n} B_{k\ell}(Z^t)_{\ell j}$$

$$= (\mathbf{M_0 \Sigma M_0}^t)_{ij} + \sum_{k=1\ldots n} A_{ik} Z_{jk} + \sum_{k=1\ldots n} A_{jk} Z_{ik} + \sum_{k,\ell=1\ldots n} B_{k\ell} Z_{ik} Z_{j\ell}$$

To minimize Var(**Wv**), it is thus sufficient to choose Z such that.

$$\partial V_{ij} / \partial Z_{\alpha\beta} = 0 \qquad \forall \alpha, \beta$$

with.

$$\partial V_{ij} / \partial Z_{\alpha\beta} = 0 \qquad \forall \alpha \neq j,i$$

$$\partial V_{ij} / \partial Z_{i\beta} = \delta_{ij} A_{i\beta} + A_{j\beta} + \sum_{\ell=1\ldots n} B_{\beta\ell} Z_{j\ell} + \delta_{ij} \sum_{k=1\ldots n} B_{k\beta} Z_{ik}$$

$$\partial V_{ij} / \partial Z_{j\beta} = A_{i\beta} + \delta_{ij} A_{j\beta} + \delta_{ij} \sum_{\ell=1\ldots n} B_{\beta\ell} Z_{j\ell} + \sum_{k=1\ldots n} B_{k\beta} Z_{ik}$$

where δ_{ij} is 1 iff i=j, 0 otherwise. Thus Z satisfies the equation:

$$\mathbf{ZB = -A} \text{ or } \mathbf{Z M_1 \Sigma M_1 = -M_0 \Sigma M_1}$$

which is of the same kind as equation (13). Thus there exists a matrix Z of minimum quadratic norm which minimizes the dispersion of the noise at the output, it is given by $\mathbf{Z = -AB^+}$.

If noise **v** has a diagonal dispersion matrix $\mathbf{\Sigma} = \sigma^2\mathbf{I}$, then Z is such that.

$$\mathbf{Z M_1 \Sigma M_1 + M_0 \Sigma M_1 = 0 \Rightarrow Z[I-XX^+][I-XX^+]}\sigma^2 \mathbf{+ YX^+[I-XX^+]}\sigma^2 \mathbf{= 0}$$

$$\Rightarrow \mathbf{Z[I-XX^+] = 0} \text{ (if } \sigma^2 \neq 0) \Rightarrow \mathbf{Z = 0} \text{ or } \mathbf{I-XX^+ = 0}$$

Hence, for a noise **v** with 0 mean and dispersion $\sigma^2\mathbf{I}$, the solution to (13) $\mathbf{W = YX^+}$ (with Z=0) minimizes the remaining noise at the output □

In many practical situations, the state space **S** is not $\mathbb{R}$, but a discrete space, for example |S|=r for an image in r gray levels Then, the input patterns $\mathbf{X_k}$ are in this discrete state space, but **W** has its elements in $\mathbb{R}$ and thus the output **y** is in $\mathbb{R}^p$ The output must then be thresholded so as to be interpreted in the same terms as the inputs (e g image) Usually, performances will be improved, in the case of auto association, if the thresholded

output is fed-back to the network and the whole process of projection, thresholding .. reiterated (fig 6). this can be seen when the linear operator is used With input **x**, the projection XX^+ builds vector $x_{\mathcal{L}}$ which may lie in between the memorized items X_i, very close -but not identical- to some X_k As further projections will keep $x_{\mathcal{L}}$ fixed, there will be no way to move $x_{\mathcal{L}}$ towards X_k repeated thresholding, by taking $x_{\mathcal{L}}$ out of $\mathcal{L}$ will allow further approximations to take place, thus improving performances. This leads us to retrieval processes which are in fact far from linear and there does not exist any more theoretical reason why matrix $W = YX^+$ should be optimal for these retrieval processes.

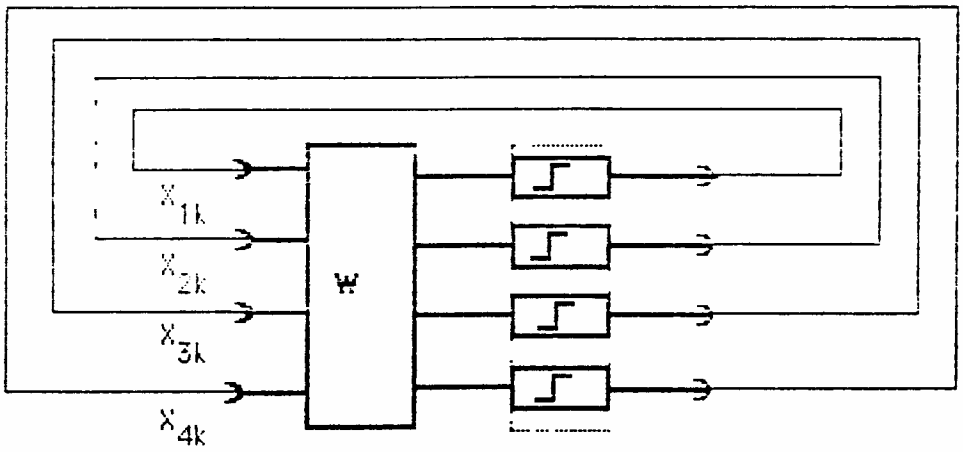


Figure 6: Iterative network.
The figure shows a network, with connection matrix **W** The retrieval process consists in sending the input, X_k in the figure, through the network which computes WX_k, then thresholding this result, component by component, then feeding-back the thresholded value into the network. The process can be re-iterated until a fixed point is reached, or stopped after a certain number of iterations

4.3. Threshold networks.

The model of threshold networks has been largely studied [27,31,45,52, 61,80] almost independently from the work presented so far in the previous paragraphs. However, when those networks are proposed to implement associative memories [27,45], it is easy to see, as we mentioned before (§3 5), that they are very close to the linear model, with thresholded output when patterns X_k are linearly independent and orthonormal, then the optimal associative memory, i.e the solution of minimum quadratic norm of equation (13) $WX=Y$ is given by (26). $W = XX^t$ This rule, inspired from Hebb reinforcement law [34], was first introduced in Little [52] and further elaborated by Hopfield [45] who proposed to use an ENERGY function.

FIGURE 7-1

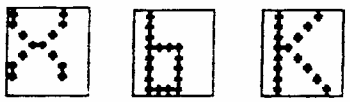
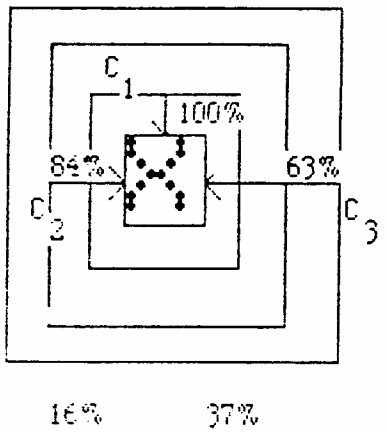
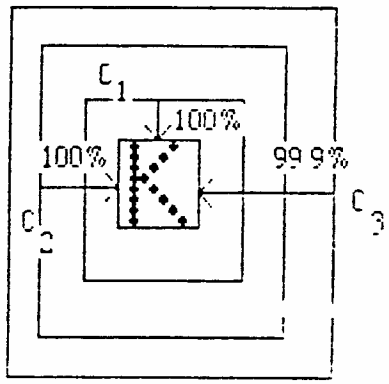
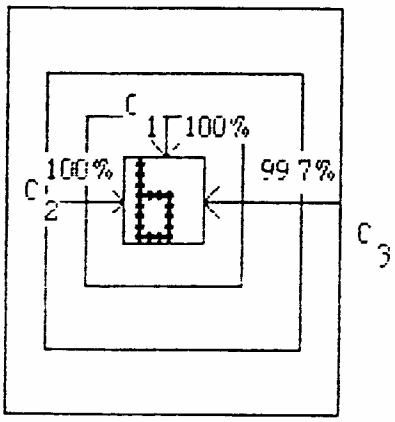


FIGURE 7-2







0 3%

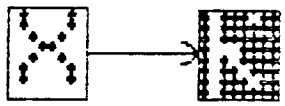


FIGURE 7-3

Figure 7: Threshold network to retrieve the 3 letters "X", "b","K"
The learning rule is Hebb law with 0-diagonal. The memorized patterns (Fig 7-1) are clearly not independent, which leads to fault tolerance properties rather weak The figure 7-2 shows the attractivity of each memorized item in its three first neighborhoods C_i (i bits different) Spurious attractors appear to be very close to the memorized items, since, for example, 37% of the items in neighborhood C_3 of "X" are attracted to a spurious attractor

Note also that such networks cannot achieve pattern invariance as is shown in the bottom part of the figure (fig.7-3): an "X" translated leads to a spurious attractor Going beyond this limitation requires networks with more complex structure (§5).

One very distinctive features of the approach developed below is the use of ITERATIONS in the linear retrieval model, a pattern $\mathbf{x}$ is projected (in case of auto-association) onto $\mathfrak{L}$, and this gives the final output. We will present here models where the output is the result of an iteration process run on the network Various kinds of iteration modes have been defined on networks [67] which might eventually lead to different results.

In this section, we will fully develop this model and show how tools introduced in other chapters [32,67] can be used to give theoretical results on the retrieval performances. The material presented here can be found in more details in [23,31]

Using the definition of a threshold automaton (1), we have.

Definition We will say that $F : \mathfrak{E}^n \rightarrow \mathfrak{E}^n$ is a THRESHOLD NETWORK iff all its components $(f_1, .., f_n)$ are threshold automata (see (1), (2)):

$$\forall i(1,...,n), \forall \mathbf{x} \in \mathfrak{E}^n, \quad f_i(\mathbf{x}) = \begin{array}{ll} 1 & \text{if} \quad \sum_j W_{ij} x_j - \theta_i \geq 0 \\ -1 & \text{otherwise} \end{array}$$

where $\mathbf{W}=(W_{ij})$ is a real nxn matrix and $\boldsymbol{\theta}=(\theta_i)$ is a real n-vector, denoting, as usual, the weights and thresholds of the network.

In the following, we will use, unless specifically mentioned, the case where $\mathfrak{E} = \{0,1\}$. We have seen before that this case could easily be rewritten into the $\mathfrak{E} = \{-1,1\}$ case

f_i is called a STRICT threshold automaton (see chapter 4) iff.

$$\begin{array}{lll} f_i(\mathbf{x}) = 1 & \text{if} & \sum_j W_{ij} x_j - \theta_i > 0 \\ -1 & \text{if} & \sum_j W_{ij} x_j - \theta_i < 0 \end{array}$$

It is easy to show that any threshold automaton is equivalent to a strict threshold automaton [31] by changing the thresholds. We will thus, in the following, make the assumption of strict threshold, which is necessary in some proofs

It was shown [66,67] that different modes of iteration could be defined on a network We will use the following notations.

Notation: Let $F \quad \mathfrak{E}^n \rightarrow \mathfrak{E}^n$ be an automata network of n elements and $\mathbf{x}^0 \in \mathfrak{E}^n$ The different ITERATION MODES on F with initial condition $\mathbf{x}^0$ are defined by $\mathbf{x}^0=\mathbf{x}(0)$ and $\mathbf{x}(t) \in \mathfrak{E}^n$, the state of F at time t, where $\mathbf{x}(t)$ is

- PARALLEL ITERATION . $\forall t \geq 0, \mathbf{x}(t+1) = F[\mathbf{x}(t)]$

- SEQUENTIAL ITERATION associated to the permutation π of $\{1,...,n\}$:

$$\forall t \geq 0,\ \mathbf{x}(t+1) = F_{\pi}[\mathbf{x}(t)]$$

Elements i of the network change state one at a time in the order prescribed by π

- BLOCK-SEQUENTIAL ITERATION associated to a partition $(I_k)_{k=1...p}$ of $\{1,..,n\}$, ordered (which means that $\forall \alpha \in I_i, \forall \beta \in I_j$, $i<j \Rightarrow \alpha < \beta$) as:

$$\mathbf{x}(t+1) = F_{(I_k)}[\mathbf{x}(t)]$$

Elements i in I_k change state in parallel, and the blocks I_k are iterated serially one after the other

- RANDOM ITERATION, with visiting strategy $(i_t)_{t\geq 0}$ (where $(i_t)_{t\geq 0}$ is a random process with values in $\{1,..,n\}$)

$$\forall t \geq 0,\ \mathbf{x}(t+1) = F_{(i_t)}[\mathbf{x}(t)]$$

Elements i of the network change state one at a time in the order prescribed by i_t

Definition : Let $F : \mathbb{E}^n \rightarrow \mathbb{E}^n$ be a threshold network. We define the ENERGY of the network for the different iteration modes as follows:

- sequential, block sequential or random iterations $\forall \mathbf{x} \in \mathbb{E}^n$,

$$E(\mathbf{x}) = -1/2 \sum_{i=1}^{n} x_i \sum_{j\neq i} W_{ij} x_j + \sum_{i=1}^{n} (\theta_i - W_{ii})\, x_i$$

- parallel iteration $\forall \mathbf{x} \in \mathbb{E}^n$, $E[\mathbf{x}(t)] = \mathfrak{E}[\mathbf{x}(t),\mathbf{x}(t-1)]$

with $$\mathfrak{E}[\mathbf{u},\mathbf{v}] = \sum_{i=1}^{n} u_i \sum_{j=1}^{n} W_{ij} v_j + \sum_{i=1}^{n} \theta_i\, [u_i + v_i]$$

Remark: this energy coincides with the energy of a spin glass with field θ, as defined for example in [45], only in the case of sequential, block sequential and random iterations The energy for the parallel iteration does not have any physical meaning in this context (see also [32])

The importance of the concept of energy is due to the following theorem (see [32] for proofs)

Theorem: Let $F : \mathbb{E}^n \rightarrow \mathbb{E}^n$ be a threshold network.

Then. $$\forall t \geq 0,\ \mathbf{x}(t+1) \neq \mathbf{x}(t) \Rightarrow E[\mathbf{x}(t+1)] < E[\mathbf{x}(t)]$$

- for any sequential iteration, where **W** is a symmetric matrix, with non negative diagonal elements

- for any block-sequential iteration associated to the ordered partition (I_k), where **W** is a symmetric matrix with its blocks $W_k = (W_{ij})_{i,j\in I_k}$ non negative definite on the set $\{-1,0,1\}$, for all k.

- for the parallel iteration, where **W** is a symmetric matrix.

(In each case, E is the energy defined for that iteration mode)

This theorem thus shows that the energy is a LYAPUNOV FUNCTION of the network for the different iteration modes, which leads to:

Corollary: Under the conditions of the previous theorem.

- all sequential, random and block sequential iterations on threshold networks only have FIXED POINTS

Furthermore, the TRANSIENT LENGTH of any trajectory is bounded below .

$$\forall x^0 \in \mathbb{B}^n, \qquad t(x^0) \leq [1/\varepsilon]\,[1/2 \sum_{i,j=1}^{n} |W_{ij}| + \sum_{i=1}^{n} |\theta_i|\,]$$

with $\varepsilon = \mathrm{Min}\,\{\|E[x(t)]-E[x(t+1)]\| / x^0 \in \mathbb{B}^n,\ x(t+1) \neq x(t)\}$

Moreover, if **W** is an integer matrix $(W_{ij} \in \mathbb{Z})$

$$t(x^0) \leq \sum_{i,j} |W_{ij}| + 2 \sum_i |\theta_i|$$

and if **W** has its elements in $\{-1,0,1\}$ $(W_{ij} \in \{-1,0,1\})$.

$$t(x^0) \leq 3nv$$

where $\forall\ i$, $\mathrm{card}\,\{\, j \in \{1, \ldots, n\}\ \ W_{ij} \neq 0\} \leq v \quad (v \leq n)$

- the parallel iteration on a threshold network only has limit cycles of period 1 or 2 Furthermore, the transient length of any trajectory is bounded below

$$\forall\, x^0 \in \mathbb{B}^n, \qquad t(x^0) \leq [1/\varepsilon]\,[1/2 \sum_{i,j} |W_{ij}| + \sum_i |\theta_i|\,]$$

with $\varepsilon = \mathrm{Min}\,\{\,|\,\mathfrak{E}[F^2(x),F(x)]-\mathfrak{E}[F(x),x]\,|\,/\,x \in \mathbb{B}^n,\ x \neq F^2(x)\,\}$

Moreover, if **W** is an integer matrix or if **W** has its elements in $\{-1,0,1\}$, the bounds are as before.

This corollary allows us to use the energy for studying the associative memory capabilities of threshold networks [27,80]

Definition. Let $F: \mathbb{E}^n \to \mathbb{E}^n$ be a threshold network, with connection matrix **W** and threshold vector θ Let **X** be a matrix with column vectors $\mathbf{X}_k$ in $\mathbb{E}^n$ to be memorized. We will say that F is an ASSOCIATIVE MEMORY for **X** iff:

- all $\mathbf{X}_k$ are fixed points of the sequential iterations on F.

- all $\mathbf{X}_k$ are attractive in their neighborhoods

$$\forall k, \exists V_k \subset \mathbb{E}^n \quad \mathbf{X}_k \in V_k$$
$$\forall \mathbf{x}(0) \in V_k, \exists T \geq 0 \ \forall T' \geq T, F_\pi^{T'}[\mathbf{x}(0)] = \mathbf{X}_k$$

According to this definition, the memorized patterns, the $\mathbf{X}_k$, are stable in the recognition process (the iteration), and any pattern sufficiently close to an $\mathbf{X}_k$ will be recognized as this $\mathbf{X}_k$ (see fig 7). Note that what is intended here is quite similar to what we discussed in §4.2: we want to retrieve memorized patterns from noisy inputs. But there are two main differences:

- the concept of noise is replaced here by the notion of HAMMING DISTANCE: a memorized pattern $\mathbf{X}_k$ must be retrieved from patterns **x** in a neighborhood V_k of $\mathbf{X}_k$

- the retrieval process is REITERATED until convergence is obtained (fixed point).

Within, this formulation it is possible to give conditions which ensure that F is an associative memory for **X**:

Theorem: If **W** is a symmetric matrix, with non negative diagonal elements, if **W**, θ and **X** satisfy the following conditions.

$\forall k(1,\ldots,m), \forall i,j.$

$$\text{grad}_i E(\mathbf{X}_k)\ (X_{ik} - \neg X_{ik}) \leq 0$$
$$[\text{grad}_j E(\mathbf{X}_k) + (X_{ik} - \neg X_{ik})]\ [X_{jk} - \neg X_{jk}] \leq 0$$

then F is an associative memory for **X** (where $\neg 0 = 1$ and $\neg 1 = 0$).

The first condition in this theorem can be written as a condition on thresholds when the W_{ij} are given.

$$\forall i(1,\ldots,n) \qquad (28)$$
$$\mathrm{Max}\{\textstyle\sum_{j=1} W_{ij} X_{jk} \,/ k(1\ldots m)\ X_{ik}=0\} < \theta_i \leq \mathrm{Min}\{\textstyle\sum_{j\neq i} W_{ij} X_{jk} \,/ k(1\ldots m)\cdot X_{ik}=1\}$$

The theorem gives a NECESSARY AND SUFFICIENT CONDITION which ensures that the patterns are fixed points attractive in their first neighborhood This condition holds whatever the learning rule used to compute the W_{ij}. This is made possible by choosing appropriately the thresholds θ_i, (see [27] for more details on this), which is different from the approach taken in linear learning, where thresholds are learnt just in the same way as weights

Condition (28) clearly holds for small values of m (eg m=1), but, when m increases, it may become infeasible This allows us to derive a theoretical bound on the storage capacity of the network.

Theorem if patterns $\mathbf{X_k}$ are random, independent samples of a Bernoulli variable with parameter 1/2, then the probability that (28) holds for all thresholds θ_i tends, for large n, to

$$P = \mathrm{erf}^{nm}[(n-1/2(m-1)]^{1/2} \qquad (29)$$

and the capacity of the network $c = m/n$ is then:

$$c \sim 1/8 \ \mathrm{Log}\, n \qquad (30)$$

See [27] for a proof of that Simulations (fig 8) support relatively well the theoretical expression given in (29), especially for large values of n. However, when the patterns are not random (which is generally the case in practice!), the capacity is usually reduced.

Moreover patterns do NOT need to be independent or orthogonal as was requested in the original Hopfield's work But then, in the case where **W** is computed by Hebb rule (26), the optimality of **W**, implied by the theory developed in section 3.2., does not hold anymore and performances are usually very degraded

We have represented, in figures 7, 9, two examples of threshold networks, with matrix **W** computed using Hebb rule (26) and a zero-diagonal constraint, which are used to retrieve letters and words Note that Hebb rule is much easier to compute than the usual optimal associative memory matrix given by (16). However, performances, in terms of capacity and noise tolerance, are far from being as good when the patterns are not independent and orthogonal

Another nice feature of the Hebb model compared to the linear model is that the retrieval process leads to the EXACT stored pattern. stored patterns $\mathbf{X_k}$ are made fixed points of the dynamics and thus there is an exact match

between the retrieved pattern and the pattern closest to the noisy item (when the distance is not too large). In linear associators, the retrieved pattern is the projection of the noisy pattern on the linear space spanned by the memorized patterns and thus may eventually be non IDENTICAL to the closest memorized pattern (even with thresholding) linear associators achieve the best possible matching However, for patterns too far away from a stored item, linear associators still lead to relatively good matching, where threshold networks with Hebb rule may lead to SPURIOUS attractors completely different from the memorized items (see fig. 7 and 9).

This model can be improved by changing the state space to $\mathfrak{E} = \{-1,1\}$ (which doubles the capacity [80]· see fig 9) or choosing optimally the thresholds θ within the bounds given by (28). But its performances for practical purpose can never be compared to the linear associator. It should be considered mainly when the number of patterns is small and/or when the patterns are preprocessed so as to be made orthogonal. Kohonen also discusses the issue in the framework of the linear model

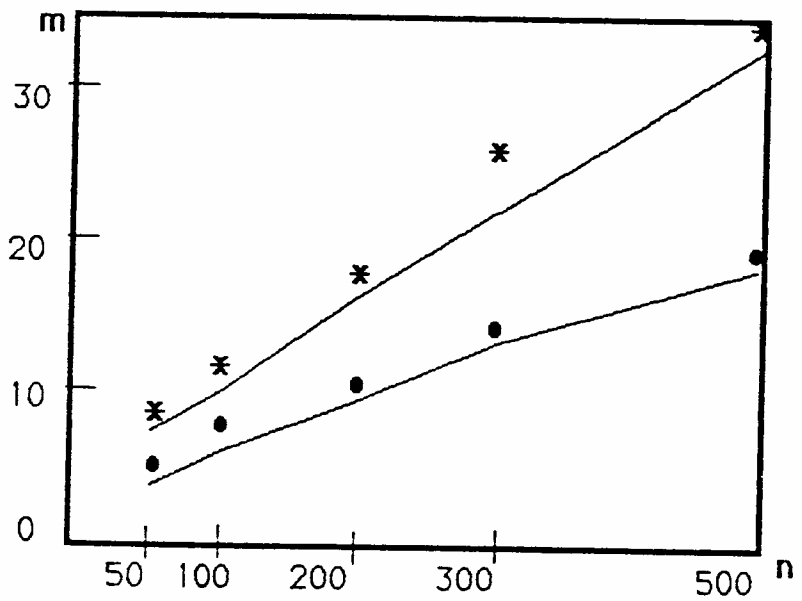


Figure 8 capacity of a threshold network.
The figure shows the variation of m, number of random memorized items, as a function of n, size of the network. The solid lines correspond to the theoretical value of m, computed by equation (29) with P=.5, with 0-threshold (bottom) and (top) thresholds computed by

$$\theta_i = (1/2) \sum_k \sum_{j \neq i} [2X_{ik}-1][2X_{jk}-1]$$

which is an optimal choice [80] The simulation results are shown, with the same choice for thresholds, respectively by dots and stars They are obtained by testing nm inequalities (28) per network and adjusting m to get frequency 5 of getting all inequalities satisfied.

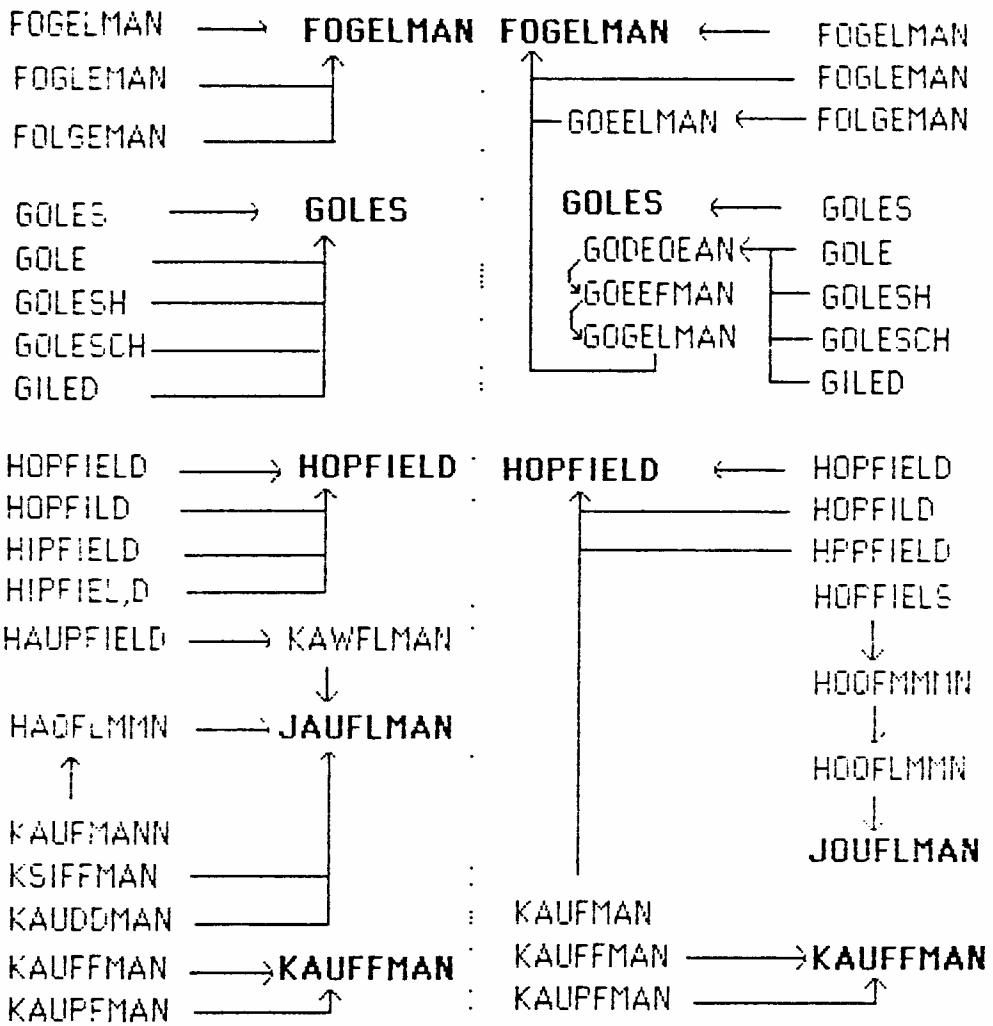


Figure 9: Threshold network to retrieve words.
The network is of size n=104, each word is coded by concatenating the binary values of the ASCII codes of its letters. We memorized the 4 words: FOGELMAN, GOLES, HOPFIELD, KAUFFMAN, by using a Hebb law with 0 diagonal But a network of this size is capable of memorizing about 8 words Words in **bold** are fixed points.
Obviously, these words are not (linearly) independent Nethertheless, our system allows to retrieve words several bits away from the original word (one letter different is equivalent to at most 8 bits different). This shows that the second condition in our theorem (sufficient for attractivity in the first neighborhood) is not very restrictive we did not impose it and the system still work well.
Performances are much improved by working in $\{-1,+1\}$ (left in the figure) instead of $\{0,+1\}$ (right).

4.4. Brain state in the box.

Let us call "feed back" the output $\mathbf{Wx}$ to a given input $\mathbf{x}$, computed by a network with connection matrix $\mathbf{W}$ Biological studies in perception -vision for example- have shown that this feed-back is very rapidly evoked, in comparison with the effects of the visualized pattern. It may be interesting

to take this observation into account and assume that the pattern and the feed-back are present at the same time during the retrieval process

This leads to use, for auto association, a weight matrix of the form

$$M = \alpha W + \beta I$$

Anderson [5] gives a model based on such a matrix **M** with the additional assumption that the activities of the "neurons", i.e. the states of the automata in the network, are bounded. The retrieval process is known as the "Brain State in the Box" model (BSB). If the state space is **B**=[s,S] (**B** is the "box"), then for a given input **x**, the process iteratively computes a sequence **y**(i) given by

$$\mathbf{y}(0) = \mathbf{x}$$

$$y_j(i+1) = \begin{cases} s & \text{if } M_j\mathbf{y}(i) < s \\ M_j\mathbf{y}(i) & \text{if } M_j\mathbf{y}(i) \in B \\ S & \text{if } M_j\mathbf{y}(i) > S \end{cases}$$

where M_j is line j in matrix **M**

This iteration procedure is either repeated until a fixed point $\mathbf{y}^*$ is reached or a maximum number of iterations is done: this will be the output of the network. In general the transient time is relatively large and the states y^*_i of the automata are almost all "saturated", i.e. either s or S

4.5. Generalization

Let $W = XX^+ + Z(I - XX^+)$ a solution of equation. $WX = X$ (13)

The BSB algorithm uses the operator $M = \alpha W + \beta I$ on an original pattern **x** inside the box **B**

This linear operator **M** can be written as follows

$$\begin{aligned} M &= \alpha W + \beta I \\ &= (\alpha XX^+ + \alpha Z(I - XX^+) + \beta I)(XX^+ + I - XX^+) \\ &= (\alpha + \beta)[XX^+ + ((\alpha Z + \beta I)/(\alpha + \beta))(I - XX^+)] \\ &= (\alpha + \beta)M' \end{aligned}$$

The bracketed expression **M'** is just a solution of (14) with no particular properties With $\alpha + \beta > 1$ this function **M** allows for various iterations, the box limiting the growth of the vectors so that the process will eventually converge (see fig 10)

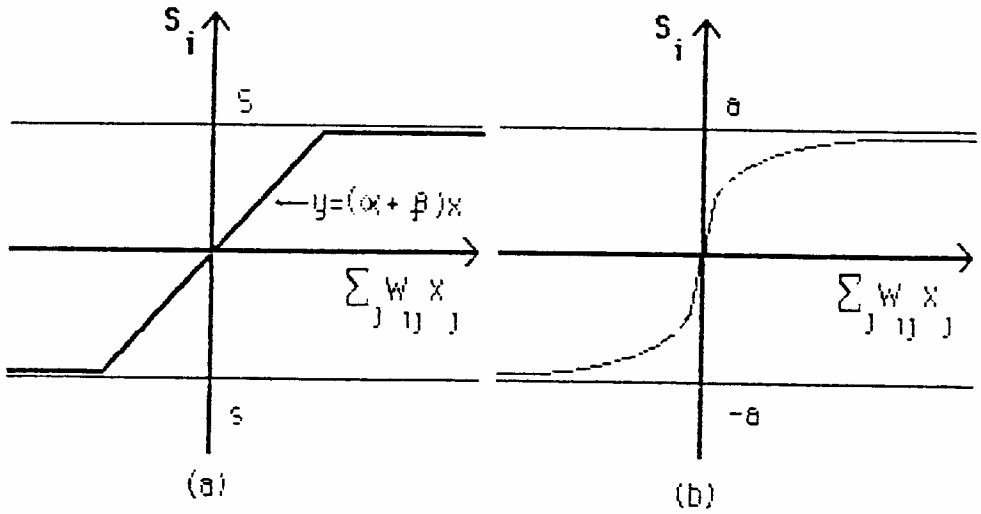


Figure 10: Smooth network.
The figure shows a smooth function (right) given by an expression such as (27 The BSB model is represented (left) by the variations of the output S_i in terms of its total output A_i The variations are limited by the bounds of the box s,S

This approach can be interpreted in the following way. **M'** can be seen as the weight matrix of the network, computed during the learning phase

Let $A_i = \sum_j W_{ij}X_j$ the total input to cell i through the linear tranform **W**

The BSB retrieval process can then be seen as a limiting case of the following retrieval process $y = f(A_i)$, f being the sigmoïd function·

$$f(x) = a(1 - e^{-k*x})/(1 + e^{-k*x}) \qquad (27)$$

The size of the box and the slope of the function are respectively defined through the a and k coefficients

f can be iterated in the same way as with the BSB model, but, usually the fixed point is reached only after a large number of iterations. In our simulations, we stopped the computations after a given number of iterations (usually 10). The parameters a and k must be optimized to allow for good retrieval performances A comparison of the two methods performances will be given in figure 13

4.6. Simulations

Simulations were performed on the 26 letters of the alphabet (fig 11), encoded with -1, 1 values on a n=8*8 grid (corresponding to 64 automata) Three memory sizes have been tested m=10, 18 and 26 letters

In the following, we present the conditions of the learning and retrieval phases of the experiences

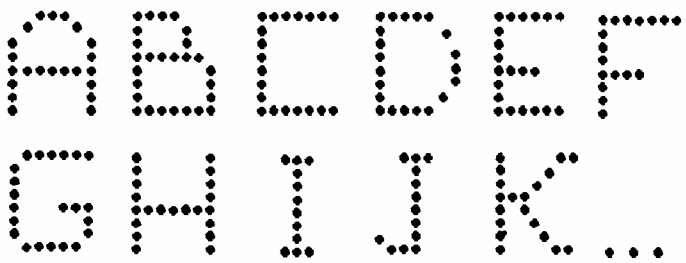


Figure 11: Letters
The figure shows the letters of the alphabet, digitized on an 8x8 grid

Learning

The following two algorithms have been tested, both of them with the three memory sizes mentioned above (see fig 12)

1) the exact solution of minimal norm of $\mathbf{WX} = \mathbf{X}$, i.e. $\mathbf{W} = \mathbf{X}\mathbf{X}^{+}$, computed through the Greville's algorithm (curves W10,W18, W26)

2) a constrained solution of this equation (0-diagonal) computed through a projected gradient method (curves W10Ø, W18Ø, W26Ø)

In all cases, the desired memorization has been obtained. But, for the first method, just one presentation of the alphabet was sufficient, where, for method 2, it was necessary to present the alphabet a number of times which increases with the number of memorized patterns: e.g. 4 presentations of the patterns were needed for m=18 and 10 for m=26.

Retrieval

Three retrieval algorithms have been tested with the 18 patterns case and a 0-diagonal matrix (W18Ø)

- threshold network the $\mathbf{1}[\mathbf{W}x]$ transform (with 0 as threshold) is iterated until a fixed point is reached.

- BSB network. the function $\mathbf{M} = \alpha\mathbf{W} + \beta\mathbf{I}$ with $\alpha = 0.2$ and $\beta = 0.9$ (as inAnderson [5]) has been iterated inside a $[-1.3,1.3]^{64}$ box, the number of iterations being bounded to 60 At the end of the process, i e when a fixed point has been reached or 60 iterations have been done, the output y is 0-thresholded ($\mathbf{1}[y]$)

- "smooth" network. the function $f(x) = a(1 - e^{-k*x})/(1 + e^{-k*x})$ with a=1.7 and k=4 has been iterated with exactly 10 iterations, and the result ultimately thresholded with 0 threshold.

We give below (fig 12-13) curves representing the retrieval ratio as a function of the exact number of inverted pixels on memorized patterns Each point on the curve is an average computed from 100 random trials (of inverted pixels) on each pattern X_k

Discussion

The second learning method appears to be better for retrieval performances, whatever the retrieval process, to a degree which increases with the memory size. When the number of memorized patterns is high, XX^+ tends to be the identity so that the input noise is less and less corrected The minimum norm solution of (14) is not the best anymore, especially when the input noise is not gaussian uncorrelated

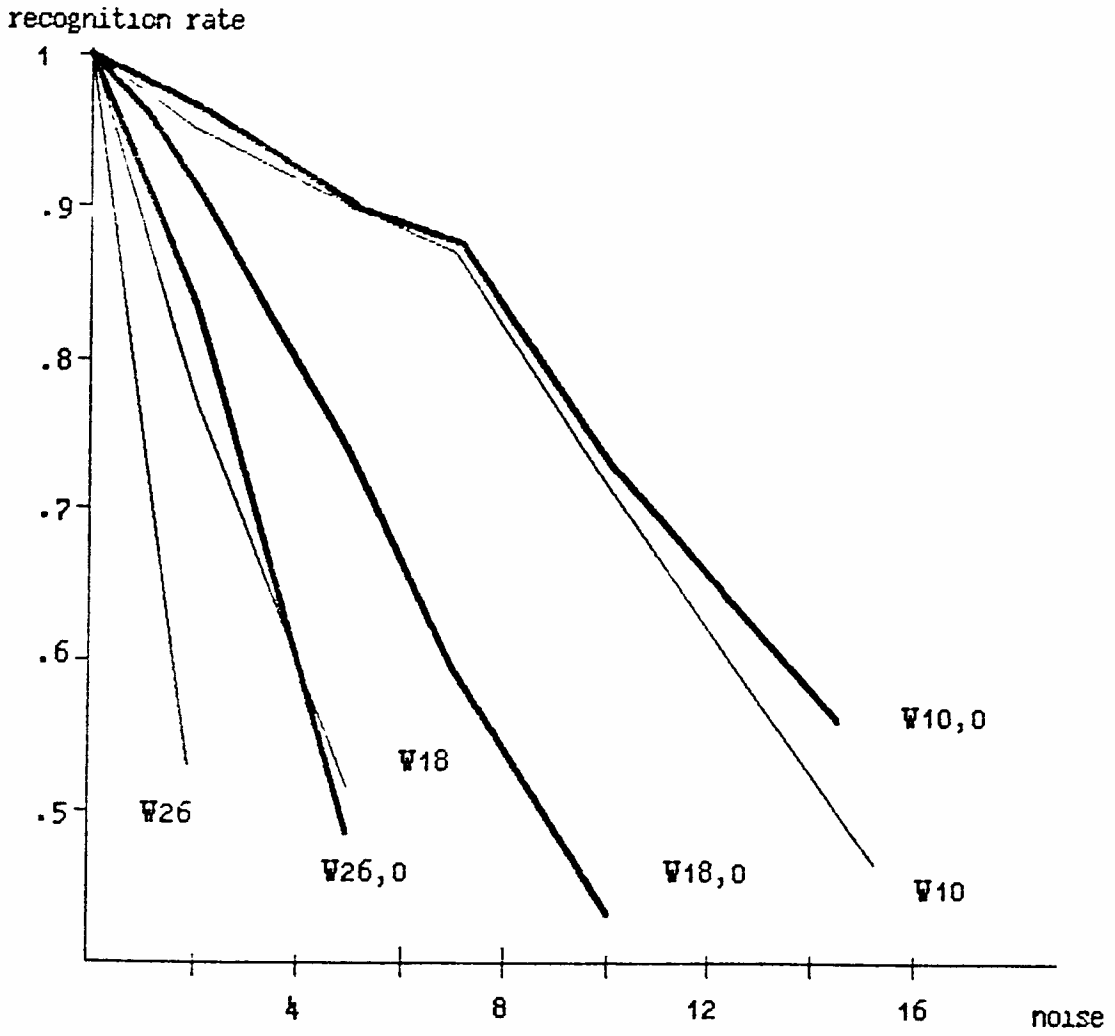


Figure 12: Comparison of two learning algorithms

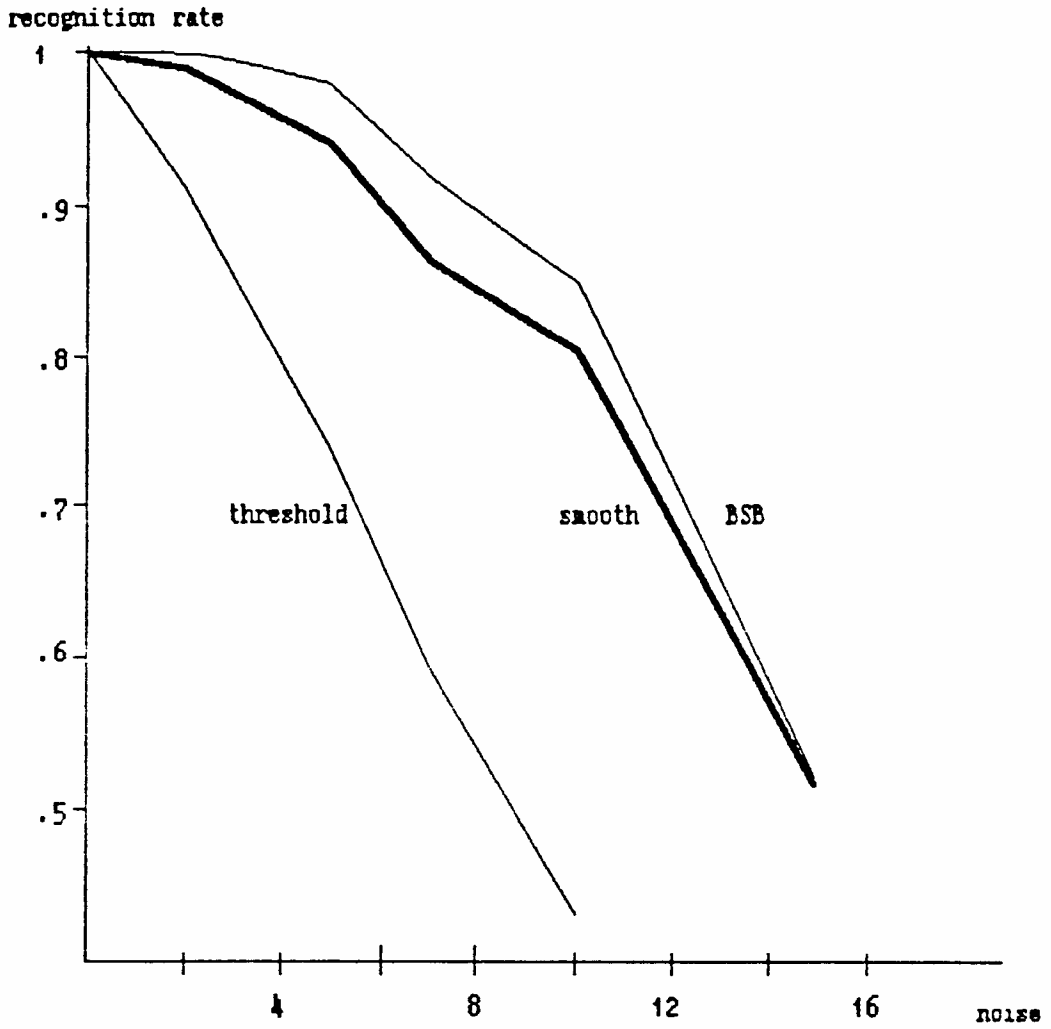


Figure 13 Comparison of three retrieval processes.

During the retrieval, the simple threshold method appeared to allow for only a few iterations and its performances were clearly dominated by the two others. But, as the noise -i.e number of inverted pixels- increases, so does the number of iteration steps, leading to an increase in the computational burden This is particularly true for the BSB model which needs more iterations than the smooth case, however no computation of an exponential -time consuming!- is required A full theoretical analysis of these last two methods is not available at present

Remark letters are always linearly independent. But the recognition rate usually differs between letters and the standard deviation of this rate increases with the number of inverted pixels (see fig 14)

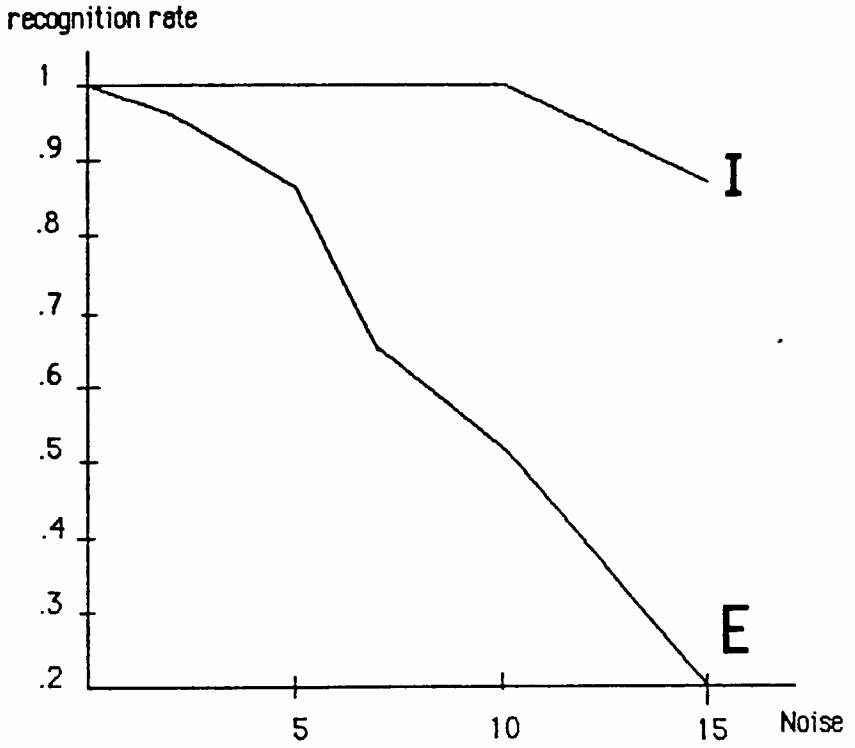


Figure 14: Comparison of retrieval performances for various letters
We show the retrieval rate for two different letters: "I" and "E". The performances are clearly better for the "I".

5. MULTILAYERED NETWORKS

5.1. Introduction.

As pointed out in the previous sections, most of the connectionist models are based on the same kind of basic operator. namely a weighted sum of the inputs followed by a non-linear transformation.

In section 2.3 we introduced the concept of order of a predicate, and we noted that a perceptron could not learn functions of order larger than one, unless it had been specifically designed to do so (fig4). We noted that even a boolean function as simple as the exclusive-or is of order two. This limitation is due to the nature of the threshold function which basically dichotomizes its input space with a hyperplane. And it is known that the proportion of linearly separable functions (over all possible boolean functions) goes to zero as the number of input variables increases [15].

Thus, a classification problem of practical complexity has virtually no chance to be directly learnable by a linear classifier. Note that the limitations of linear threshold functions apply as well to the associative nets (linear models) described in the previous sections.

Various solutions have been proposed to turn these limitations:

• the first possibility is to parametrize a high order polynomial hyper-surface and to fit its parameters by learning. [60] The limits of this approach are well known (even a simple n-dimensional surface of order 2 has n^2 parameters).

FIGURE 15-1

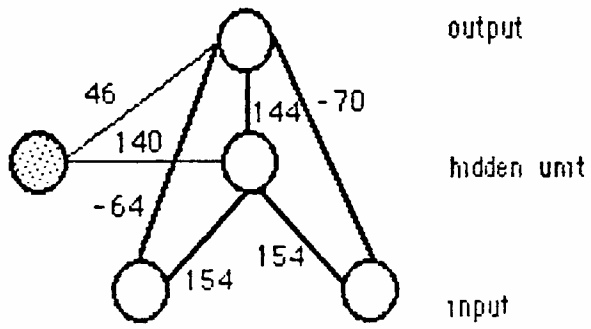


FIGURE 15-2

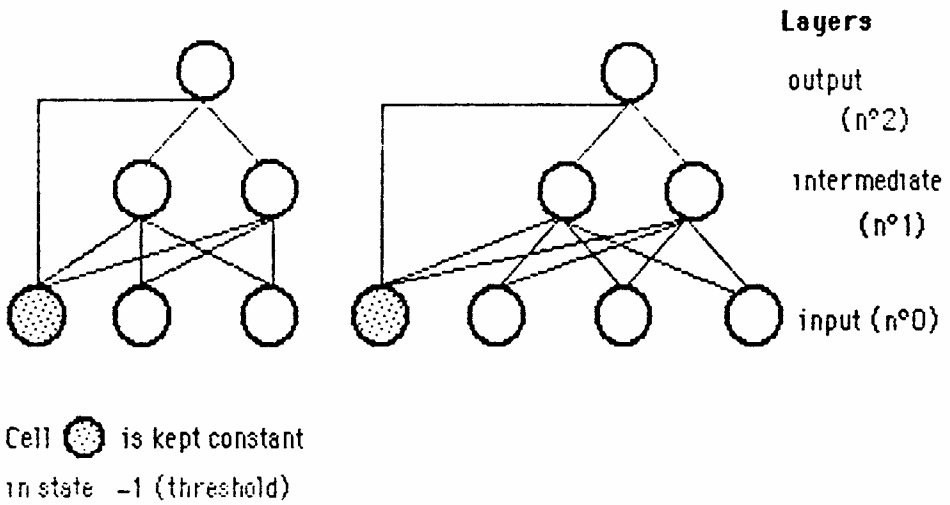


Figure 15. Computing XOR and 3-parity predicate with hidden units.
The figure shows various networks with hidden units that compute a XOR.
Figure 15-1 is a network which was found as the result of the learning algorithm called "Boltzmann Machine" (from [75]).
Figure 15-2 shows networks with 2 hidden units that compute a XOR (left) and the 3-parity predicate (right). These networks have been computed using the Gradient Back Propagation algorithm
Note that the solution found by the Boltzmann Machine algorithm only uses one hidden unit, instead of two for the GBP solution, but connections are different in the two cases At present, there is no complexity theory about the minimum number of hidden units required

• Another solution is to precede the learning phase by an adequate "decoding stage", so as to extract the relevant features before performing the classification. Such an approach has been taken by various authors and proved to perform quite well even on problems of high order [72]. We have seen, for example, that hand-wired feature extractors could allow to compute the exclusive-or (fig 4). The problem with this technique is that

the decoding step has to be crafted by-hand. It would be most desirable to be able to produce this "decoder" automatically.

• A solution to this coding problem has been recently proposed by various authors [1, 37, 50, 51, 70], making use of networks with so-called "hidden units". Their idea is to take advantage of these units so as to extract AUTOMATICALLY from the examples the relevant features.

This approach was first developed by Hinton and colleagues [1, 37]: the Boltzmann Machine, a network of probabilistic automata, composed of two sorts of cells (visible and hidden), was the first one in demonstrating that it was possible to automatically learn to perform order-2 tasks, such as the exclusive-or (see fig 15).

In multi layered networks, introduced later on [50, 51, 63, 70], the network contains, as in the linear models discussed in the previous sections, an input and an output layers, and moreover intermediate layer(s) which will serve as feature extractors The cells in these intermediate layers are analogous to the hidden units in the Boltzmann Machine: they have NO direct interaction with the environment, their state is fully determined by the dynamics on the network itself

Learning the weights of these internal or "hidden" units is not straightforward since no external signal specifies their ideal outputs. So there is no direct way to know whether the output of an internal unit is correct for a particular input pattern or not. This problem has been called the CREDIT ASSIGNMENT PROBLEM [56].

Let us illustrate the importance of hidden units on a simple example. Consider an automata network as in §3 (fig.5) with which we would like to compute an exclusive-or. This is equivalent to making the following configurations stable points:

000xx...x
011xx...x
101xx...x
110xx...x

where xx .x is an arbitrary sequence of bits (the same for the four vectors) There is no set of weights (whatever the method used to compute them) that can make these vectors stable without making stable all the other vectors. Each of the first three bits of a line is a XOR of the remaining two. Computing the exclusive-or or any other non-linearly separable function with threshold operators may be done by using additionnal cells. Those cells can compute intermediate predicates such that any output value of a cell can be expressed as a (thresholded) linear combination of the others. For example, the problem just described could be solved by adding a spare unit which state has been clamped by hand.

0000xx...x
0111xx...x
1011xx...x
1101xx...x

Now any of the four first bits in the patterns can be computed from the other three through a linear threshold function: in figure 4, for example, we have represented a network with 1 hidden unit which realizes the XOR.

In an attempt to design a general learning rule, the state of the spare units should not be defined by the outside world but should eventually be generated by the network itself, since these bits are not really part of the patterns

These additionnal operators are called "hidden units" [70] because they have no direct interaction with the outside world but are only supposed to cope with the underlying regularities that cannot be taken into account by simple linear pairwise interactions. The concept of hidden (or internal) unit can be used in the case of feed-forward nets (with no loop in the interaction graph) which is a special case of the fully-connected net seen in §3.

We will extensively study multi-layered networks in the following sections. In this case, the hidden units compute intermediate representations that decompose the input-output function into elementary successive steps (see fig 16). However, learning these intermediate predicates without requiring the environment (or the designer) to explicitly specify them is not a simple task, because what is addressed here is the problem of GENERATING INTERNAL REPRESENTATIONS. It can be compared, in a different context, to the problem of decomposing a complex task (or a function) into simpler sub-tasks (or elementary operators)

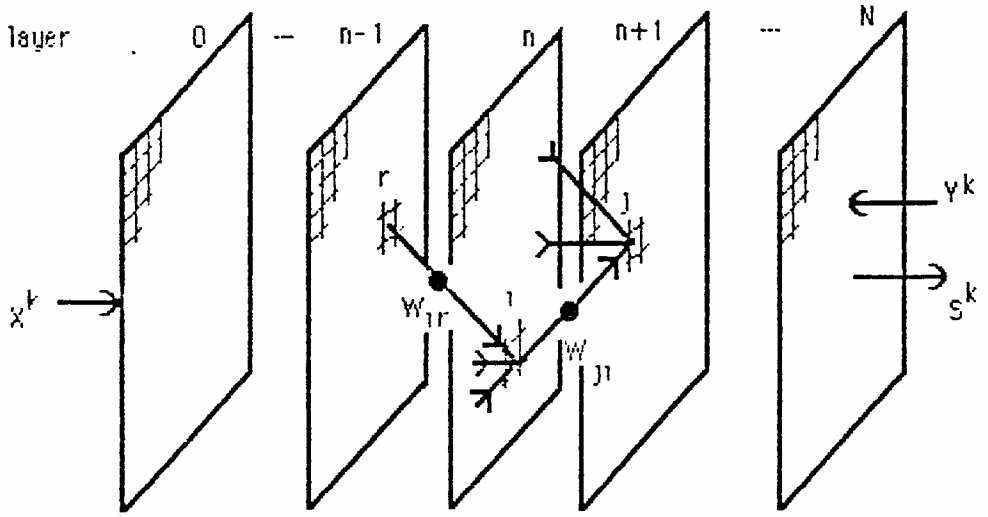


Figure 16. Multi layered network.
The figure shows a multilayered network with N-1 intermediate layers. The connection are only between successive layers and the weights are shown.

The credit assignment problem was an unsolved problem until recently. Some authors [56] even suggested in the late '60s that such generalizations of the neural network paradigm might not exist. This provides a possible explanation for the decrease of interest in the field during the '70s.

Its recent renewal originated from the learning procedures recently proposed for the BOLTZMANN MACHINE [1] and for multilayered feed-forward networks. This last algorithm, called GRADIENT BACK PROPAGATION (GBP), was developed by several authors independently [50, 51, 70]. It is based on the propagation of error signals and solves the credit assignment problem. In the following, we will describe this last procedure, using intuitive arguments. We will then give a general theoretical method for deriving such algorithms and finally give examples of applications.

5.2. GBP: a learning algorithm for multilayered networks.

This section will be concerned with networks composed of several layers of neuron-like processing elements.

The first one (number 0) is the input layer, its state (the input vector) will be denoted $\mathbf{X}^h$. The last layer (number N) is the output layer, its state, the output vector will be denoted $\mathbf{S}^h$.

We will assume for sake of simplicity that the connections can only go from lower layers to higher layers, and that there is no connection between cells in the same layer. Connections can skip layers.

The elements compute a weighted sum of their inputs, followed by a non-linear transformation f. Let x_i be the output of cell i:

$$x_i = f[\ \sum_j W_{ij}.x_j\ ] \qquad (28)$$

where W_{ij} is the weight between cell j and cell i. As usual we will define A_i the TOTAL INPUT to cell i as:

$$A_i = \sum_j W_{ij}.x_j \qquad (29)$$

such that we can now write.

$$x_i = f[\ A_i\ ] \qquad (30)$$

Here, f is a DIFFERENTIABLE FUNCTION. In the following, it will be assumed that f is odd, with a strictly positive first derivative (e.g. a sigmoid function: see

fig.8). At time h, one element $\mathbf{X^h}$ of the family of input patterns $\mathbf{X_k}$, is presented on the first layer, the stable state of the network is computed by propagating the information through the layers using equation (30). Simultaneously, a desired output vector $\mathbf{Y^h}$ is given to the cells of the last layer (see fig 14). In the previous sections, we have presented some learning procedures for finding the weights of a formal neuron from examples of desired input-output pairs. All these methods are based, in an implicit or explicit way, on the iterative minimization of a particular criterion attached to the automaton. In the simplest case, this criterion is the squared difference between the computed output and the desired output of the unit. In a multilayer network, only the units belonging to the last layer are given a desired output.

For those cells, a criterion C_k measuring the discrepancy between the desired output $\mathbf{Y_k}$ and the actual output $\mathbf{S_k}$ of each output unit for the k^{th} input pattern $\mathbf{X_k}$, is defined

$$C_k = \sum_s [S_{sk} - Y_{sk}]^2 \qquad (31)$$

where the sum is taken over the units, indexed by s, of the output layer only.

S_{sk} thus corresponds to the state $\mathbf{x_s}$ of cell s in the output layer. We now define a global criterion $\mathbf{C}$ as the sum of the C_k , the sum being taken over the m possible input-output pairs:

$$\mathbf{C(W)} = 1/m \sum_k C_k \qquad (32)$$

Instead of minimizing $\mathbf{C}$ directly, we will minimize an estimate of $\mathbf{C}$ using the stochastic gradient method similar to the Widrow-Hoff rule described in §3.4 From now on, for notation convenience, we will skip the temporal index h: C will denote the cost associated with ONE pattern.

In order to modify the weights using gradient descent, we have to calculate the effect of their variations on the value of the criterion, i.e. the partial derivatives of the criterion with respect to these weights: $\partial C/\partial W_{ij}$. Rather than computing all those derivatives directly, we will first compute the gradient of the criterion relative to the total input of each cell. $\partial C/\partial A_i$ Once these derivatives are known, the derivatives relative to the weights are easily computed with the following formula (given by the definition of A_i)

$$\partial C/\partial W_{ij} = \partial C/\partial A_i \, . \, x_j \qquad (33)$$

that enables us to apply the gradient descent procedure.

The $\partial C/\partial A_s$ associated to the output cells immediately follow from equations (28) and (31).

$$\partial C/\partial A_s = 2\,[\,S_s - Y_s\,]\,.\,\partial S_s/\partial A_s$$

which can be rewritten using (30).

$$\partial C/\partial A_s = 2\,[\,S_s - Y_s\,]\,.\,f'(A_s) \qquad (34)$$

The computation of the $\partial C/\partial A_i$ associated to the hidden cell i can be easily performed, provided that all the $\partial C/\partial A_j$ of the cells j to which cell i sends its output are known. We simply state (using chain rule):

$$\partial C/\partial A_i = \sum_j \partial C/\partial A_j \,.\, \partial A_j/\partial A_i$$

We now need to compute $\partial A_j/\partial A_i$:

$$\partial A_j/\partial A_i = \partial A_j/\partial x_i \;\; \partial x_i/\partial A_i$$

The first factor in the right member is equal to W_{ji} , while the second is $f'(A_i)$ Combining the two previous equations gives:

$$\partial C/\partial A_i = f'(A_i).\sum_j W_{ji}.\partial C/\partial A_j$$

For clarity, we will denote, in the following, the gradient of C with respect to A_i by: $y_i = \partial C/\partial A_i$ (35)

The previous equation and equation (34) then respectively become:

$$y_i = f'(A_i)\,.\sum_j W_{ji}\,.\,y_j \qquad (36)$$

$$y_s = 2\,[\,S_s - Y_s\,]\,.f'(A_s) \qquad (37)$$

Equation (36) provides a mean for computing the gradient of the cost C with respect to the total input of a cell by performing the weighted sum of the y's corresponding to the cells to which it sends an output. The weights involved in this operation are the same as in the forward pass, but used backwards Thus in a multilayered feed-forward net, the y's are back-propagated from the last layer to the first one. Note that despite the fact that this section was primarily concerned with feed-forward networks, the arguments developed here can be easily transposed to any network (with loops).

In short, one iteration of the Back-Propagation learning algorithm consists in (see figure 17).

- Present pattern X^h, and comput the state x of the network by forward propagation (eq 30).

for an input cell i $\quad x_i = X^h_i$

from layer 1 to layer N $\quad x_i = f(A_i) \qquad A_i = \sum_j W_{ij} x_j$

• Present the desired output Y^h, compute the gradients y_s on the last layer using equation (37), and compute the y_i corresponding to the hidden units by back-propagation (36).

for an output cell s $\quad y_s = 2 [S_s - Y^h_s] . f'(A_s)$

from layer N-1 to layer 1 $\quad y_i = f'(A_i) . \sum_j W_{ji} . y_j$

• Modify the weights by inserting (33) in a stochastic gradient method

$$W^h_{ij} = W^{h-1}_{ij} - \lambda^h y_i x_j \qquad (38)$$

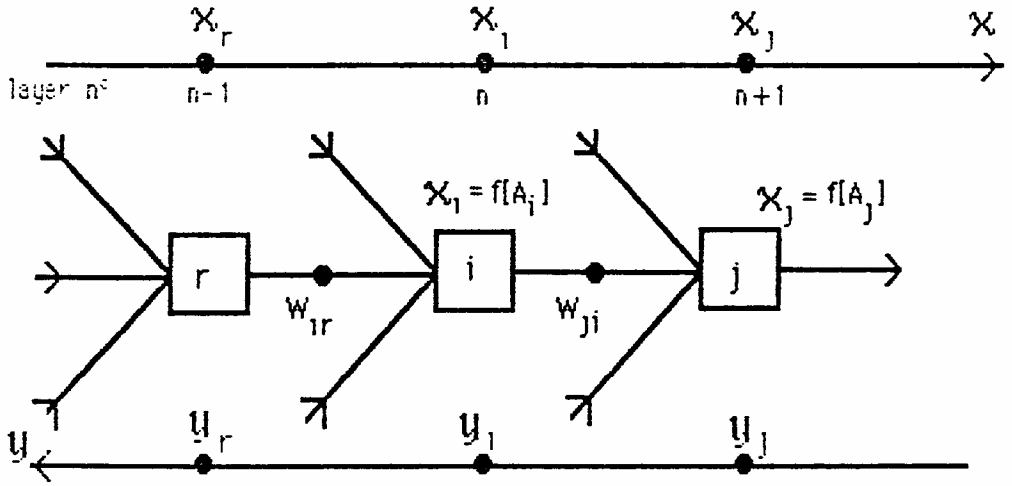


Figure 17 Gradient Back Propagation algorithm.
The figure shows the forward pass (top) to compute the states x, from layer 1 to layer N, and the backward pass (bottom) to compute the gradients y, from layer N-1 to layer 1.

5.3. Simplified GBP algorithm: the Hierarchical Learning Machine

Several back-propagation procedures for learning the weights of a multilayer network had been proposed before a theoretical formulation appeared

The Hierarchical Learning Machine algorithm (HLM) [50, 51] is one of them. The HLM learning procedure is basically an approximation of the GBP in the case of binary units (threshold functions: f=1), that uses as criterion:

$$C_k = \sum_s [Y_{sk} - A_s]^2$$

where A_s is the total input to cell s in the last layer, when the input is X_k.

Then, it is easy to see that the back-propagated variables are not error signals but desired states. To keep formula comparable, we will use, in this section, the same symbol y for the desired states than for the gradients in the previous section.

Instead of a full mathematical development, which can be found in [50, 51], we will just describe here the basic idea of the HLM learning algorithm. The simplest learning strategy is to use a Widrow-Hoff-like rule, provided that we can associate a desired state to each unit, including the hidden units.

While the desired states of the output units are defined externally, the problem is to compute the desired states of the hidden units. The following unformal argument will be used. assume that we want to compute the desired state y_i of the i^{th} hidden unit, and that the desired states of the cells to which this cell sends its output are known; in other words, y_j is supposed to be known if j is such that $W_{ji} \neq 0$. We choose y_i such that the contribution of the i^{th} cell to the input of the other ones is optimal, i.e. such that

$$y_i = 1(\sum_j W_{ji}\, y_j) \qquad (39)$$

which states that the desired output of cell i is the binary value (+1 or -1) nearest to the average of the desired states of the following cells. The sum is weighted with the W_{ji} because we want to get a cell i, followed by a cell j,to fall into a desired state which "satisfies" cell j state in proportion to weight W_{ji} (the larger W_{ji}, the more likely $y_i = y_j$) Compare equation (39) with equations (11) -perceptron criterion- and (36) -back propagation of gradients.

A learning iteration is composed of the three following phases.

- Present pattern X^h, and compute the state x of the network by forward propagation.

for an input cell i : $x_i = X^h_i$

otherwise $x_i = 1(A_i)$ $\qquad A_i = \sum_j W_{ij}\, x_j$

- Present the desired output Y^h on the last layer, and compute the y_i corresponding to the hidden units by back-propagation

for an output cell s : $y_s = Y^h_s$

otherwise $y_i = 1(\sum_j W_{ji} . y_j)$

- Modify the weights using the standard Widrow-Hoff rule

$$W^{h}_{ij} = W^{h-1}_{ij} - \lambda^{h} (A_i - y_i) x_j$$

This procedure can be derived in a more formal way from the criterion defined above. This algorithm is less stable than the "continuous GBP" but it is simpler and exhibits some interesting properties (see [50, 51])

5.4 The GBP algorithm as a constrained minimization process

A general framework for deriving GBP-type procedures is presented in this section. Extensive use of the LAGRANGIAN formalism is made in a context similar to optimal control theory [6]. For sake of clarity, this method will be described in the special case of multilayered feed-forward networks where connections cannot skip layers

Consider a network with N+1 layers, the state vector of the k^{th} layer is denoted $\mathbf{x}(k)$, the corresponding weight matrix, connecting the $k-1^{th}$ layer to the k^{th} layer is $\mathbf{W}(k)$. The global state is thus defined by

$$\mathbf{x}(k+1) = F[\, \mathbf{W}(k+1)\, \mathbf{x}(k)\,]$$

or with the usual notations:

$$\mathbf{x}(k+1) = F[\, \mathbf{A}(k+1)\,] \qquad \text{with} \qquad \mathbf{A}(k+1)=\mathbf{W}(k+1)\, \mathbf{x}(k) \qquad (40)$$

with $\mathbf{x}(0) = \mathbf{X}^{h}$ and F the function with all its components equal to f.

Let $\mathbf{Y}$ be the desired state of the last layer. We define the LAGRANGE function L:

$$L = [\mathbf{Y} - \mathbf{x}(N)]^{t}[\mathbf{Y} - \mathbf{x}(N)] + \sum_{k=0\ldots N-1} Z^{t}(k+1).\, [\mathbf{x}(k+1) - F[\, \mathbf{W}(k+1)\, \mathbf{x}(k)\,]] \qquad (41)$$

The first term in this equation is the usual quadratic cost function to be minimized (we can consider that L depends only on one pattern pair $\mathbf{X},\mathbf{Y}$ without loss of generality, since L is additive).

The second term represents the constraints of the problem, $\mathbf{Z}(k)$ being the Lagrange multipliers The constraints terms are equal to 0 if equation (40) is satisfied The optimality conditions which describe the behavior of the network are

$$\partial L / \partial Z_i(k) = 0 \qquad \text{for all i and k}$$

$$\partial L / \partial x_i(k) = 0 \qquad \text{for all i and k}$$

$$\partial L/\partial W_{ij}(k) = 0 \qquad \text{for all } i,j \text{ and } k \tag{42}$$

The first condition simply gives $\mathbf{x}(k+1) = F[\ \mathbf{A}(k+1)\ ]$ for k=0..N-1, which describes the forward pass.

The second condition gives two different equations depending whether k=N or not. If k=N we obtain:

$$\mathbf{Z}(N) = 2\,[\mathbf{Y} - \mathbf{x}(N)] \tag{43}$$

if k≠N we have:

$$\mathbf{Z}(k) = \mathbf{W}^t(k+1)\,.\,\mathbf{D}_F[\ \mathbf{A}(k+1)\ ]\,.\,\mathbf{Z}(k+1) \tag{44}$$

where $\mathbf{D}_F[\ \mathbf{A}(k+1)\ ]$ is the jacobian matrix of $\mathbf{F}$ at point $\mathbf{A}(k+1)$. In our case, this matrix is diagonal, and its i^{th} term is equal to $f'[A_i(k+1)]$ For clarity, the previous equation can be rewritten in scalar form:

$$Z_i(k) = \sum_j W_{ji}(k+1).\ f'[A_j(k+1)].\ Z_j(k+1)$$

By denoting $y_i = f'[A_j(k+1)].\ Z_j(k+1)$ and substituting in (43) and (44) we obtain:

$$\mathbf{y}(N) = 2\,\mathbf{D}_F[\ \mathbf{A}(N)\ ][\mathbf{Y} - \mathbf{x}(N)]$$

$$\mathbf{y}(k) = \mathbf{D}_F[\ \mathbf{A}(k)\ ]\,\mathbf{W}^t(k+1)\,\mathbf{y}(k+1) \tag{45}$$

which describe the backward pass in the GBP algorithm.

The third condition in (42) does not give a practical method for computing the weights but just gives an optimality condition.

$$\mathbf{y}(k)\,\mathbf{x}^t(k-1) = 0 \qquad \text{for all } k$$

However, the weights that satisfy this condition can be easily computed using a gradient descent algorithm:

$$W_{ij}^{h+1}(k) = W_{ij}^{h}(k) - \lambda^h\,.\,\partial L/\partial W_{ij}^{h}(k)$$

This gives

$$W_{ij}^{h+1}(k) = W_{ij}^{h}(k) + \lambda^h\,.\,y_i(k)\,x_j(k-1) \tag{46}$$

or using matrix notation.

$$\mathbf{W}^{h+1}(k) = \mathbf{W}^{h}(k) + \lambda^h\,.\,\mathbf{y}(k)\mathbf{x}^t(k-1) \tag{47}$$

These results can be summarized by the following system.

- for k = 0...N-1

$$\mathcal{X}(k+1) = F[\, \mathbf{A}(k+1)\,] \qquad \text{with} \qquad \mathbf{A}(k+1) = \mathbf{W}(k+1)\,\mathcal{X}(k)$$

- for k = N-1...0

$$\mathcal{Y}(k) = \mathbf{D}_F[\, \mathbf{A}(k)\,]\, \mathbf{W}^t(k+1)\, \mathcal{Y}(k+1)$$

- for k = 0...N-1

$$\mathbf{W}^{h+1}(k+1) = \mathbf{W}^h(k+1) + \lambda^h \,.\, \mathcal{Y}(k+1)\, \mathcal{X}^t(k)$$

- and the "boundary conditions".

$$\mathcal{Y}(N) = 2\, \mathbf{D}_F[\, \mathbf{A}(N)\,][\mathbf{Y}^h - \mathcal{X}(N)]$$

$$\mathcal{X}(0) = \mathbf{X}^h$$

This system is identical to classical GBP, except that the output of a layer depends exclusively on the outputs of the previous one. However, a similar derivation can be made without this assumption.

Many different Lagrange functions can be chosen, each of them leading to a different learning algorithm which depends on the choice of the cost function, the non-linear transformation **F**, and the particular architectural assumptions. For example, L can be chosen such that **W** does not depends on the layer index k, in that case, the matrix connecting a layer to the next one is the same for every layer. The index k can now be interpreted as a time index, and the Lagrange function can be viewed as describing the time evolution of a cross-connected network. The learning rule involves an initial state **X** at time k=0, and a final (target) state **Y** at time k=N. The rule is identical to the one described in [70] for pseudo-iterative nets and depends on the sum of the gradients over the trajectory

5.5 Experiments with the GBP algorithm

The dynamic behavior of the GBP algorithm can be analyzed using simple learning tasks We propose in this section some typical examples.

As shown below, a multilayer network can be trained to compute boolean functions such as the EXCLUSIVE-OR, the n-parity problem, or the product of two three-bits binary numbers. The function f chosen in our simulations is

$$f(x) = a[1-e^{-k*x}]/[1+e^{-k*x}] \qquad (48)$$

This function is odd, strictly increasing, and varies continuously between -a and +a (see (27)).

• Synthesis of boolean functions

The problem here is to synthesize a network which computes a boolean function defined by its truth table. This is not strictly speaking a learning task, since we do not expect any generalization: it is just rote learning. However, besides memorizing the truth table, the network has also built in its weight structure a COMPACT representation of the function. This is similar to the classical problem in electronic circuits design of decomposing a complex function into a small set of basic components.

We have seen before (see fig 4) that a perceptron cannot compute an EXCLUSIVE-OR unless specifically hand-wired. In contrast, it is very easy to build a multi-layer network which learns how to perform this task: the input layer has 2 units, the output layer one, and the intermediate layer may have only one hidden cell. Of course, various architectures can do. We present in figure 14 an example of such a network. Obviously, the solution in weight space is not unique there exist many decompositions of the EXCLUSIVE-OR function into threshold functions, which can be obtained from various initial conditions on W and computing the "threshold equivalent" of the smooth functions obtained (see fig.3).

The n-parity predicate (see §2.3.) can be learned by a similar network, but with a number of hidden units which depends on n (see fig.15).

It is also possible to propose a network which learns to compute the product of two 3-bits binary numbers The first layer, with two groups of 3 units encode the numbers to be multiplied, the second one has 30 hidden cells, and the last one outputs the result,in binary form, on 6 cells. There is a full connectivity between cells in successive layers (see fig.18).

With only 40 presentations of the 64 possible inputs, the network achieves 100% success.

• Associative memory

We will present here some results of simulations intended to compare the performances on an auto-association task of various algorithms discussed so far.

The patterns to be memorized are characters $\mathbf{X}_k$ coded on 8x8 bits (-1 or +1) matrices We use the 18 first letters of the alphabet, digitized by hand (see fig.11).

We use the following learning algorithms:

FIGURE 18-1

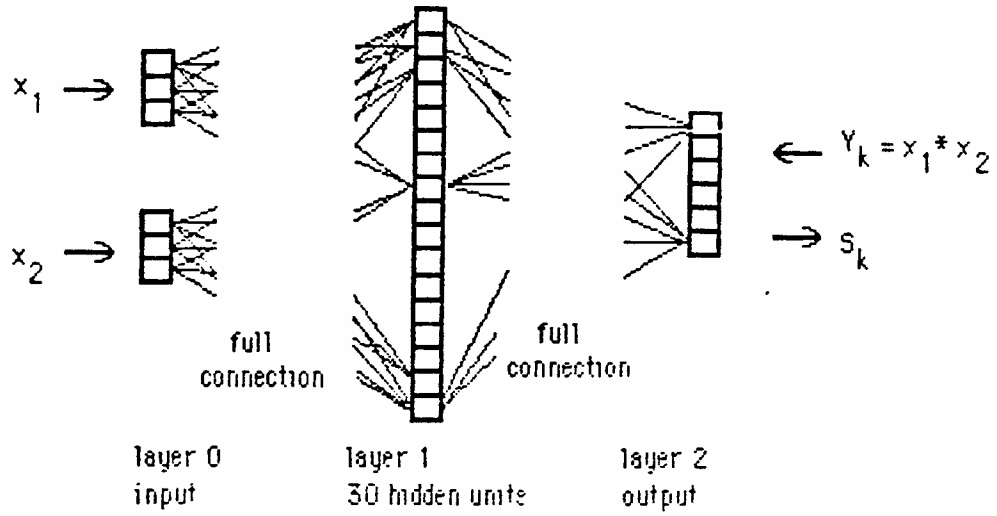


FIGURE 18-2

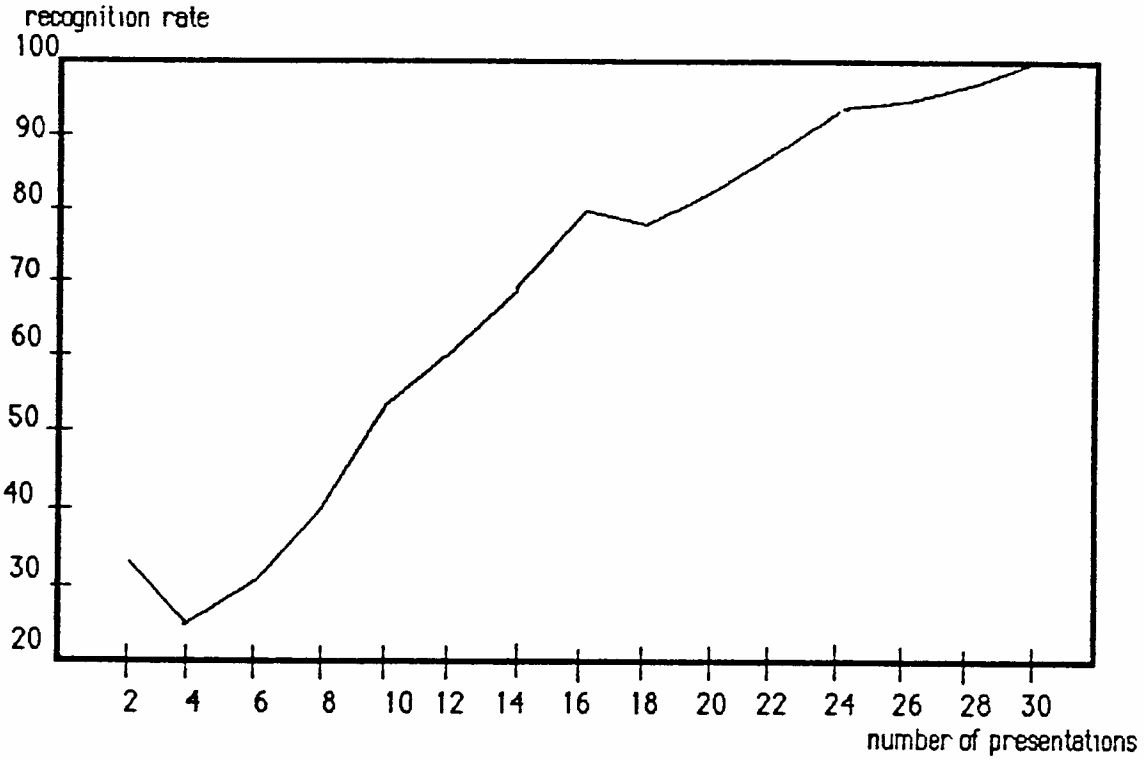


Figure 18.Network for computing the multiplication of two 3-bits numbers. Figure18-1 shows the architecture of the network and figure 18-2 the recognition rate as function of the number of presentations of the full set of examples.

1- the auto-associative model with a matrix **W** computed from (20) (Gram-Schmidt orthogonalization) and a constraint of 0-diagonal (§3.5) on **W** (**Z** is thus non zero)

2- an auto-associative two-layer network, with matrix **W** computed by a Widrow-Hoff rule and 0-diagonal constraint, on a criterion similar to (31):

$$C(\mathbf{W}) = 1/m \sum_k C_k$$

with. $C_k = \sum_s [f(\sum_j W_{sj} X_{jk}) - X_{sk}]^2$

where f is a sigmoid function as in (48), with G = 1.7 and k = 1.3 (which implies that. $f(1) \sim 1$).

3- the same network as before but with a different learning strategy. During the first part of the learning session, for each pattern $\mathbf{X_k}$, various noisy versions $\underline{\mathbf{X_k}}$ (the pattern $\mathbf{X_k}$ where 5 bits, chosen randomly, have been inverted) are presented to the network. This means that, instead of associating the $\mathbf{X_k}$ to itself, we first associate noisy versions of $\mathbf{X_k}$ (the $\underline{\mathbf{X_k}}$) to $\mathbf{X_k}$: the input pattern $\underline{\mathbf{X_k}}$ is thus associated to the desired output $\mathbf{X_k}$ In a second phase, when the weights are stabilized, the "pure" patterns $\mathbf{X_k}$ are stored in the network, ensuring that they will be correctly memorized. Formally, this looks like a hetero-association process, but in fact gives a constrained solution to the auto-association problem.

FIGURE 19-1

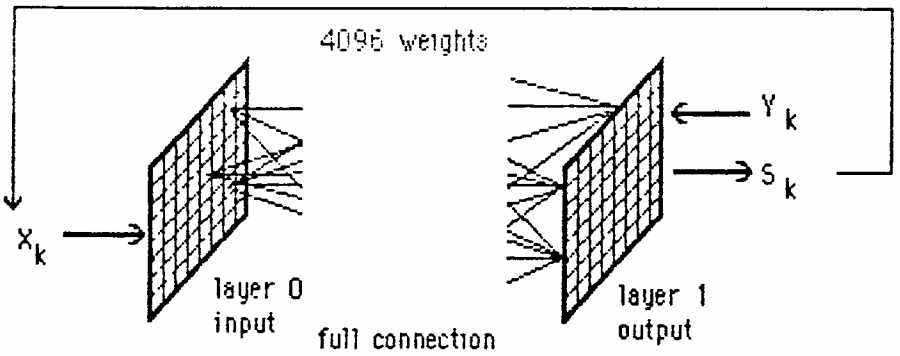


FIGURE 19-2

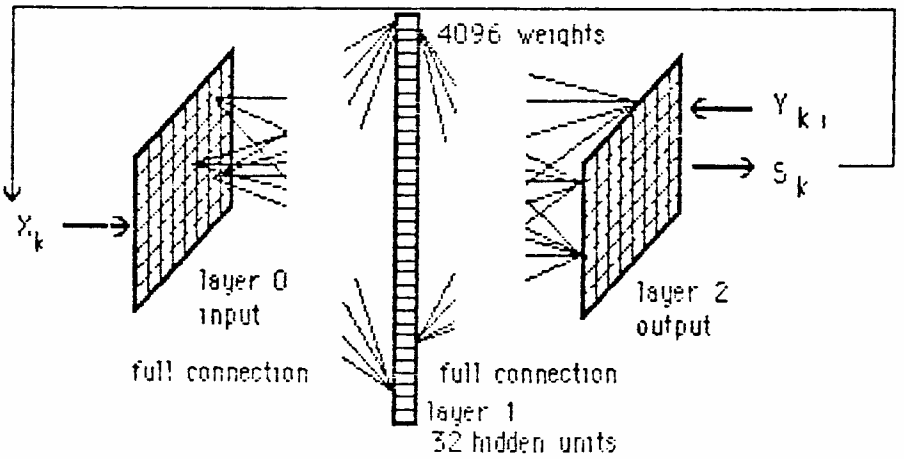


FIGURE 19-3

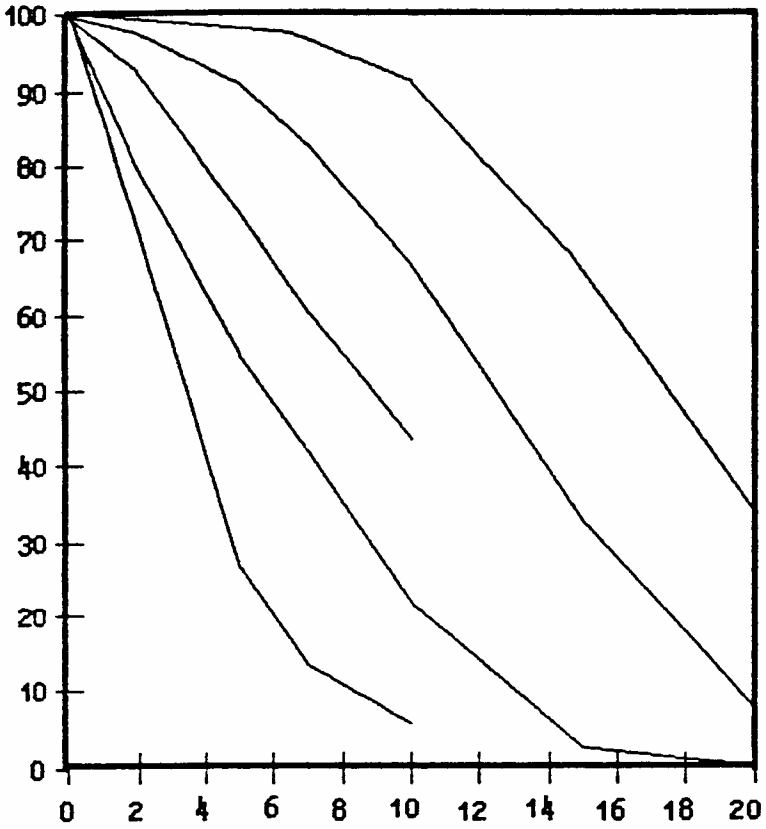


Figure 19: Comparison of various models of associative memory.
The figure shows the architectures of various networks used for associative memory. Figure 19-1 is a two-layered network, used in cases 1,2 and 3 (see text). Figure 19-2 is the three-layered network, used in cases 4 and 5. Figure 19-3 gives the rate of correct retrieval as a function of the distance (in bits) of the input pattern to the memorized item. We show the five curves associated to the five learning models discussed in the text: from top to bottom, curves 5,3,1,2 and 4.

4- a multi-layer network, with the same number of weights (4096) as the previous one-layer networks. the input and the output layers are the same (64 cells each), the intermediate layer has 32 hidden units. There is full connectivity among successive layers.

We use the Back-Propagation Algorithm with the same learning strategy as in 2

5- same network as in 4, with the learning strategy of 3, i.e. 3-layer network and noisy auto-association

The retrieval process is an iterated thresholding of the output of the network computed by (48) i.e the output computed through the smooth function is thresholded (+1 if positive, -1 otherwise) and the result is fed back into the network at the input layer. This process is repeated for 5 iterations. Figure 19 shows, in the five different cases listed above, the

curves giving the success rate (percentage of correct retrieval of pattern $\mathbf{X}_k$ from a noisy pattern $\underline{\mathbf{X}}_k$) as a function of the Hamming distance from $\underline{\mathbf{X}}_k$ to $\mathbf{X}_k$. Each point in the curves is an average over 50 samples of patterns $\underline{\mathbf{X}}_k$ for each of the 18 patterns $\mathbf{X}_k$

We observe that:

- Performances are always better when a "model of noise" has been given to the network (through the examples). The purpose of the retrieval is to cancel the noise present in the input patterns. There is no reason why there should exist a universal solution, independent of the type of noise under consideration. We have shown, in the linear framework (§4.2), that for each model of noise there exists an appropriate choice for $\mathbf{W}$ The learning strategy used in 3 and 5 can be viewed as a method for providing the network with an implicit model of noise, which constrains the choice of $\mathbf{W}$

  However, we have seen that the linear learning model (curve 1) has an inherent model of Gaussian uncorrelated noise (§4.2). The Widrow-Hoff rule for one-layer networks, which does not find the corresponding minimum norm solution, can nevertheless capture some model of noise, though in a very limited way (curves 2 and 3).

  In contrast, the multi-layer model do not have any built-in assumption on noise. This explains the poor performances obtained in case 4, which can be easily fixed, as demonstrated by curve 5, with just providing the network with a model of noise during learning.

- the multi-layer architecture clearly achieves better performances, which proves that it is more capable to capture the regularities of the environment, storing them in the weights in a more efficient way

It must be stressed that these kinds of problems are very under-constrained Providing some information on the noise thus helps constraining the set of solutions, hence finding a solution more adapted

The learning processes described above can be interpreted as moving the weight vector on a high-dimensional error surface. The shape of this error surface determines the complexity of the task to be learned.

When there is no hidden unit, this surface is a paraboloid. Since it is convex, the steepest descent procedure always finds an optimal solution for any initial weight It can frequently happen that the solution is degenerate, i.e an entire subspace in the weight space is solution to the problem In that case, the particular solution found by the gradient descent, depends on the initial weight configuration

For multilayer networks (with hidden units), the situation is very different since the error surface is generally NOT CONVEX This has several important implications.

First, the final solution depends on the initial weight configuration; second, the order of presentation of the patterns, as well as their relative frequencies of occurence, influence the result, by modifying the shape of the error surface locally in time. The third consequence is that the weights can be trapped in a local minimum of the error surface.

- **tic-tac-toe**

The game of tic-tac-toe is a 2-players game, played here on a 3x3 board. Each player plays at his turn by writing a cross (for player 1) or a circle (for player 2). The first one who succeeds in getting 3 figures in line wins. We present a multi-layered network which learns to classify the configurations of the game into win, loss or draw (we always take the view-point of the player who plays cross and assume that players play their best strategy)

The network has 3 layers the input layer codes the board configuration (i.e. a vector with 27 elements. 3 groups of 3x3 cells coding the state of the cell on the board), the output layer has 3 cells, coding the outcome corresponding to the configuration (i.e. a vector with 3 components; win, loss, draw). the intermediate layer, with 50 cells, is fully connected, both to the input and the output layers.

We use a learning set containing half of the 5478 legal configurations and test the results of the learning process on the remaining half. The recognition rate (fig 20) steadily increases with the number of presentations of the examples. after 10 such presentations, the learning rate is about .9 and the generalization rate .76 (the learning rate measures the percentage of correct answer for an input in the learning set and the generalization rate, the percentage of correct answer for an input in the test set).

Of course, those rates depend on the number of presentations of the examples, on the number of hidden units and on the parameter λ^h used in the learning rule (38). Note also that those performances have been obtained with the "standard" network and learning algorithm No a-priori knowledge about the domain has been used to improve performances

5.6 Conclusion

The Gradient-Back-Propagation algorithm presented in this section provides an answer to the problem of the limitation of all linear associator-type machines It can compute high-order predicates, through an algorithm which is universal (i e. does not depend on the domain) The learning session allows

FIGURE 20-1

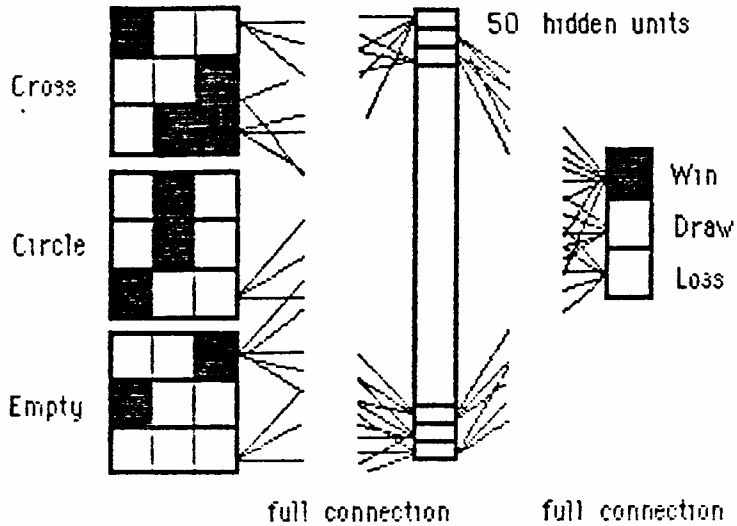


FIGURE 20-2

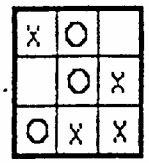


FIGURE 20-3

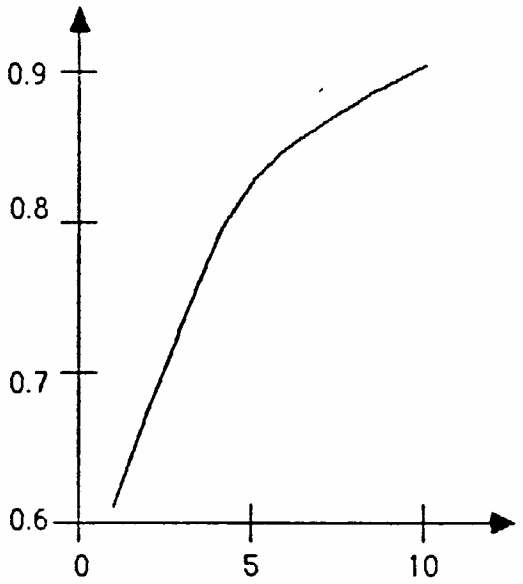


Figure 20 Network for the TIC-TAC-TOE game.
The figure shows a network that learn to tell winning configurations of a tic-tac-toe game from nul or losing
Figure 20-2 shows an admissible configuration
Figure 20-1 gives the structure of the network. 3 blocks of 3x3 cells receive the input (the configuration of fig20-2 is coded on the input layer). The intermediate layer has 50 hidden cells and the output layer 3 cells which code the output: the configuration shown in figure 20-2 is winning, so the output 100 is shown. There is complete connection between the input layer and the intermediate layer, and between the intermediate layer and the output layer
In figure 20-3, we have given the learning rate as a function of the number of presentations of the examples.

the network to elaborate intermediate predicates, that decompose the original task into elementary simple steps. The resulting representation of the characteristic features of the examples allow the network to generalize to new input patterns However, the learning session must take into account samples of the possible inputs, i.e. the learning set must include a "model of noise" (fig 19).

Let us end this paragraph with some general remarks on the algorithms we have proposed

- increasing the number of weights usually increases performances, but may decrease drastically the generalization rate: the network, when possible, just performs rote learning. Hence, a critical balance must be kept, depending on the task.

To achieve rote learning (as in the problem of automatic synthesis of boolean functions), a minimum number of hidden units is necessary (no "complexity" theory to determine this number is available) to perform the learning Increasing the number of units usually helps, but varying the architecture (number of layers, connection structure) may have similar effects.

Obtaining good generalization rates is more difficult: the exact number of hidden units, number of layers, architecture of connections must be precisely tuned for each problem.

- there are many parameters in a learning problem: number of hidden units, number of intermediate layers, connection structure between layers, pedagogy (order of presentation of the examples and number of presentations), choice of the examples, coding of the data, parameters of the learning algorithm (λ_h and initial distribution of weights). Today, all these parameters have to be hand-crafted and the whole outcome of the learning session depends heavily of the options taken.

- finally, it must be stressed that these algorithms are always very time consuming A learning session for small sized problems may take hours of computation But implementing those techniques on parallel machines may considerably speed-up the learning session. for example, some applications are presently programmed on the Connection Machine [36] Recognition is always very fast.

6. CONCLUSION

We have shown in this chapter how automata networks could be used to solve problems in Artificial Intelligence: associative memory, learning, knowledge representation...

These models are based on the use of an automata network. Its connection weights are produced automatically by a learning rule, that extracts the relevant information from examples presented to the network. The task is thus decomposed into a learning session, when examples are presented to the network, and a retrieval session, when new patterns are shown to the network which is supposed to process them, using the internal representations elaborated during the learning session.

The learning algorithms presented here can be classified in two different families the linear-type (perceptron, linear associator,...) and the multi-layer type (GBP, Boltzmann Machine,...). The performances of these families are quite different: the linear family can perform only low-order computations, whereas the multi-layered networks have been shown to perform well on a variety of more complex tasks

The Gradient Back Propagation, proposed only two years ago, is not yet fully understood Its properties must be explored in more detail to elucidate in particular:

- the exact role of the different parameters.
- the minimum requirements in terms of number of cells.
- the kind of generalization which takes place.
- the relationship between this type of learning and the classical logic-based learning

This line of research has already proved on "toy" examples that it was capable to provide interesting learning capabilities. It remains now to test the model on real-world applications, this will most probably help understanding better its limits

A characteristic of this field -as of the general field of automata- is that many authors from various scientific domains have contributed it, often independently physicists, neurophysio-logists, mathematicians, computer scientists, psychologists Much work remains to be done and tools from these different areas might prove useful.

7. REFERENCES

[1] D H ACKLEY, G.E. HINTON, T.J SEJNOWSKI: A Learning Algorithm for Boltzmann Machines. Cognitive Science, 9, (1985),147-169.

[2] S.I.AMARI: Learning patterns and patterns sequences by self-organizing net of threshold elements IEEE Trans. Com., Vol.C-21,n°11, (Nov 1972).

[3] J.A. ANDERSON. The Architecture of Cognition. Harvard Univ. Press, (1983).

[4] J.A. ANDERSON. Cognitive and Psychological Computation with Neural Models. IEEE Trans. on Syst.Man and Cybern., SMC 13, (1983), 799-815.

[5] J.A. ANDERSON: Cognitive capabilities of a parallel system. In [12], (1986), 109-226.

[6] M.A. ARBIB, A.R. HANSON Eds: Vision, brain and cooperative computation, Cambridge, MIT Press, (1985).

[7] M. ATHANS, P.L. FALB. Optimal control. McGraw Hill, (1966)

[8] D.H. BALLARD. Parameter Nets. Artificial Intelligence, 22, (1984), 235-267.

[9] D.H. BALLARD, C.M. BROWN Computer Vision. Prentice Hall, (1982).

[10] D.H. BALLARD, G.E. HINTON, T.J. SEJNOWSKI: Parallel Visual Computation. Nature, Vol 306, n° 3, (1983), 21-26.

[11] A. BARR, P.R. COHEN, E.A. FEIGENBAUM:Cohen P.R., Feigenbaum E.A.: The Handbook of Artificial Intelligence. Vol 3, Pitman, (1982).

[12] E. BIENENSTOCK, F. FOGELMAN SOULIE, G. WEISBUCH Eds: «Disordered Systems and Biological Organization», Springer Verlag, NATO Asi Series in Systems and Computer Science, n°F20, (1986).

[13] S.L. CAMPBELL, C.D. MEYER: Generalized inverses of linear transformations. Pitman, (1979).

[14] J.P. CHANGEUX. L'Homme Neuronal. Fayard, (1983).

[15] T.M. COVER. Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. IEEE Trans. Electronic Computers, Vol EC 14 (3), (1965), 326-334.

[16] J. DEMONGEOT, E. GOLES, M. TCHUENTE Eds Dynamical Systems and Cellular Automata Academic Press, (1985).

[17] R.O DUDA, P.E. HART: Pattern classification and scene analysis. Wiley, (1973)

[18] G.M. EDELMAN, G.N REEKE: Selective networks capable of representative transformations, limited generalizations and associative memory PNAS USA, vol 79, (1982), 2091-2095.

[19] D FARMER, T. TOFFOLI, S. WOLFRAM Eds: Cellular Automata Physica 10D, North-Holland, (1984).

[20] J A. FELDMAN, D.H. BALLARD: Connectionnist Models and their Properties Cognitive Science, 6, (1982), 205-254.

[21] J.A. FELDMAN. Dynamic connections in neural networks. Biol. Cybern., 46, (1982), 27-39.

[22] J.A. FELDMAN: Connections Byte, (april 1985), 277-283.

[23] F FOGELMAN SOULIE. Contribution à une théorie du calcul sur réseaux. Thesis, (1985)

[24] F. FOGELMAN SOULIE: Cerveau et machines: des architectures pour demain? «Cognitiva 85», CESTA-AFCET Ed, actes du forum, (1985).

[25] F. FOGELMAN SOULIE, P. GALLINARI, S. THIRIA. Linear learning and associative memory. In «Pattern Recognition, Theory and Applications », J. Devijver Ed., NATO ASI Series in Computer Science, Springer Verlag, to appear.

[26] F. FOGELMAN SOULIE, E. GOLES-CHACC: Knowledge representation by automata networks. In «Computers and Computing», P. Chenin, C. di Crescenzo, F. Robert Eds, Masson-Wiley, (1986), 175-180.

[27] F FOGELMAN SOULIE, G. WEISBUCH. Random iterations of threshold networks and associative memory. SIAM J. on Computing, (1986).

[28] F FOGELMAN SOULIE,LE CUN. Modèles connexionnistes de l'apprentissage. n° spécial sur «Apprentissage et Machine», Intellectica, to appear.

[29] K. FUKUSHIMA, S. MIYAKE: A Self-organizing Neural Network with a Function of Associative Memory. Feedback-Type Cognitron. Biol. Cyber., 28, (1978), 201-208.

[30] S GEMAN, D GEMAN Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images IEEE Trans PAMI, vol. 6, n° 6, (1984), 721-741.

[31] E GOLES. Comportement dynamique de reseaux d'automates. Thesis (1985).

[32] E. GOLES. This volume, chapter 4.

[33] T.N.E. GREVILLE. Some applications of the pseudo inverse of a matrix. SIAM Rev. 2, (1960), 15-22.

[34] D.O. HEBB. The Organization of Behavior.Wiley, (1949).

[35] W.D. HILLIS: New computer architectures and their relationship to physics or Why computer science is no good. Int. J. of Theor. Physics, vol21, n°3/4, (1982),255-262.

[36] W.D. HILLIS: The Connection Machine. MIT Press, (1986).

[37] G.E. HINTON, T. SEJNOWSKI: Optimal perceptual inference. IEEE Conf. on Computer vision and pattern recognition, (1983), 448-453.

[38] G.E HINTON. Distributed representations. Working paper, CMU-CS-84-157, (1984)

[39] G.E. HINTON. Private communication, (1985).

[40] G.E. HINTON. Learning in Parallel Networks. Byte, (april 1985), 265-273.

[41] G.E. HINTON: Learning to recognize shapes in a parallel network. In the proceedings of the Fyssen meeting on vision, Paris, (march 1986).

[42] G.E. HINTON, J.A. ANDERSON (Eds): Parallel Models of Associative Memory. Hillsdale, Erlbaum, (1981).

[43] T. HOGG, B.A. HUBERMAN. Understanding biological computation: reliable learning and recognition. P.N.A.S. USA, vol 81, (1984), 6871-6875.

[44] M.L. HONIG, D.G. MESSERSCHMITT: Adaptive filters, structures; algorithms and applications. Kluwer academic publishers, (1984).

[45] J.J. HOPFIELD: Neural Networks and Physical Systems with Emergent Collective Computational Abilities, P.N.A.S. USA , vol 79, (1982), 2554-2558.

[46] S KIRKPATRICK, C.D. GELATT, M.P. VECCHI: Optimization by Simulated Annealing. Science, vol 220, n° 4598, (1983), 671-680

[47] T. KOHONEN, M. RUOHONEN. Representation of associated data by matrix operators. JEEE Trans. Computers, (July 1973)

[48] T. KOHONEN : An adaptive associative memory principle. IEEE T. Comp, (Apr 74).

[49 T. KOHONEN: Self-Organization and Associative Memory. Springer Series in Information Sciences, vol 8, Springer Verlag, (1984).

[50] Y. LE CUN. A learning scheme for asymmetric threshold network. In «Cognitiva 85», CESTA-AFCET Ed., (1985), 599-604. (in French)

[51] Y. LE CUN: Learning process in an asymmetric threshold network. In [12], (1986),233-240.

[52] W. A. LITTLE: Existance of persistent states in the brain. Math. Biosc. , 19, (1974), 101-120.

[53] J.L. MAC CLELLAND, D.E. RUMELHART: Distributed memory and the representation of general and specific information. J. Exp. Psychology, Vol 114, n°2, (1985), 159-188.

[54] W.S MAC CULLOCH, W. PITTS: A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophysics, 5, (1943), 115-133.

[55] D. MARR, T. POGGIO. Cooperative Computation of Stereo Disparity. Science, vol 194, (1976), 283-287.

[56 M. MINSKY, S. PAPERT: Perceptrons, an Introduction to Computational Geometry. Cambridge, MIT Press, (1969).

[57] S. MIYAKE, K. FUKUSHIMA: A neural network model for the mechanism of feature extraction. Biol. Cybern., 50, (1984), 377-384.

[58] K. NAKANO : Associatron, a model of associative memory. IEEE Trans Syst. Man Cyb., Vol SMC -2, No 3, (July 1972).

[59] J. von NEUMANN Theory of self reproducing automata. A.W. Burks Ed. Univ. Illinois Press, (1966).

[60] N.J. NILSSON. Learning machines. McGraw Hill, (1965).

[61] P. PERETTO: Collective properties of neural networks. a statistical physics approach Biol. Cybern , 50, (1984), 51-62.

[62] T. POGGIO, V. TORRE, C. KOCH: Computational vision and regularization theory. Nature, vol 317, 26, (1985), 314-319.

[63] D.C PLAUT, S.J. NOLAN, G.E HINTON: Experiments on learning by back propagation CMU-CS 86-126, (1986).

[64] L.D. PYLE: Generalized inverse computations using the gradient projection method. J. of the A.C.M., 11, (1964), 422-428.

[65] C.R. RAO, S.K. MITRA: Generalized inverse of matrices and its applications. Wiley, (1971).