ELSEVIER

Discrete Applied Mathematics 52 (1994) 111–138

DISCRETE APPLIED MATHEMATICS

A review of combinatorial problems arising in feedforward neural network design

E. Amaldi*, E. Mayoraz[1], D. de Werra

Department of Mathematics, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland

(Received 21 August 1990; revised 11 February 1992)

Abstract

This paper is primarily oriented towards discrete mathematics and emphasizes the occurrence of combinatorial problems in the area of artificial neural networks. The focus is on feedforward networks of binary units and their use as associative memories. Exact and heuristic algorithms for designing networks with single or multiple layers are discussed and complexity results related to the learning problems are reviewed. Several methods do only vary the parameters of networks whose topology has been chosen a priori while others build the networks during the training process. Valiant's learning from examples model which formalizes the problem of generalization is presented and open questions are mentioned.

Key words: Feedforward neural networks; Associative memory; Learning process; Combinatorial optimization; Heuristics

1. Introduction

During the last decade there has been an increasing interest in the multidisciplinary field of neural networks; several specialized journals have been created and, as a consequence, have attracted most of the research contributions in this expanding area.

The purpose of this paper is not to exhaustively review the various results on artificial neural networks (ANNs), but rather to give an idea of some of the problems in this field which can be formulated as combinatorial problems. The text intends to address discrete mathematicians rather than specialists of ANNs and to give them some motivation for tackling some of the research problems which arise in this area.

*Corresponding author.

[1]Present address: RUTCOR, Rutgers University, New Brunswick, NJ, USA.

SSDI 0166-218X(92)00184-D

It should be pointed out that the ANNs are to be considered as models of computation where parallelism is introduced in a natural way. As specified in almost all papers in the area, there is no direct connection with a (human) brain—as far as we know what are its structure and its operating rules. For this reason we shall investigate some areas of ANNs which may appear to be far from real problems of learning and storing information in the nervous system. Discrete mathematical properties and results will guide our investigations; we shall concentrate on such problems without insisting on some others which may be important with respect to potentialities but have no immediate relation with discrete optimization and combinatorics.

We will be concerned with the design of ANNs for associative memory (pattern recognition, error correction, classification ...). This area seems promising from a practical point of view and there are many applications related to that use of ANNs.

In our framework, a set of input patterns paired with the desired output are presented to a network. The problem of learning is to store into the net (by adjusting its parameters) all input–output pairs presented during the *training phase* so that in a subsequent *testing phase* the system will be able to produce the required output to any input it has already seen, in other words it will make the desired associations.

The scope is to use combinatorial optimization for designing efficient ANNs. A different viewpoint would be the use of ANNs for solving combinatorial optimization problems. Several attempts have been carried out by various authors in this direction (see for instance [42, 1, 46, 62]). Among others the famous travelling salesman problem has been formulated in a way which can be mapped onto an ANN [42, 8]; these experiments have set the basis for new approaches to difficult combinatorial problems [29, 64]. But for the moment they do not seem to beat more classical techniques (as for example the tabu search heuristic [36, 30]). One should however recognize that the emergence of ANNs has certainly contributed to draw the attention of optimizers to parallel computation techniques; this has suggested fruitful ways of parallelizing previous sequential techniques as it happens for instance with the embedding of simulated annealing in Boltzman machines [1].

In the next section neural networks are briefly presented. Section 3 is devoted to the description of several training algorithms for single computing units. In Section 4 this problem is examined for feedforward networks and complexity results related to learning problems are discussed. Section 5 describes a plausible framework for tackling the generalization issue. Some open questions are mentioned throughout the text with emphasis on discrete optimization and combinatorics. All graph-theoretical terms not defined here can be found in [19]. The complexity-theoretical concepts and terminology related to $\mathcal{NP}$-completeness are thoroughly explained in [35].

2. The model

The ANN model considered here is a discrete-time system composed of (many) simple computational elements that interact through some pattern of connections. An

ANN can be viewed as an oriented graph $G = (X, U)$ where the node set X consists of the computing units and the arc set U represents the connections between some pairs of nodes. The nodes with predecessors ($V \subseteq X$) contribute to the overall computation by taking signals from their entering arcs and by calculating an output signal, whereas those without predecessors are simple interface units used to supply input to the network. Formally, each node $i \in V$ computes some function f_i of its inputs either given by each predecessor j if $j \in X - V$ or computed by the function f_j associated to predecessor j if $j \in V$. The f_i assigned to each node $i \in V$ is chosen among the members of a given family $\mathscr{F}$ of functions.

Although $\mathscr{F}$ may consist of continuous functions, in this paper the main focus is on Boolean functions

$$f_i : \mathbb{B}^{|P(i)|} \to \mathbb{B}, \tag{1}$$

where $\mathbb{B} = \{-1, 1\}$[1] and $P(i)$ is the set of predecessors of node i in G. Typical Boolean functions used in neural networks are linear threshold Boolean functions (LTB functions). A Boolean function f_i is an LTB function if there exist $\boldsymbol{w} \in \mathbb{R}^{|P(i)|}$ and $w_0 \in \mathbb{R}$ such that

$$\sum_{j=1}^{|P(i)|} w_j x_j \geqslant w_0 \quad \text{iff } f_i(\boldsymbol{x}) = 1. \tag{2}$$

In other words, there exists a hyperplane $H_i = \{\boldsymbol{x} \in \mathbb{R}^{|P(i)|} \mid \sum_{j=1}^{|P(i)|} w_j x_j = w_0\}$ which separates the vectors $\boldsymbol{x} \in \mathbb{B}^{|P(i)|}$ with $f_i(\boldsymbol{x}) = 1$ from those with $f_i(\boldsymbol{x}) = -1$. This kind of computing elements has been proposed by McCulloch and Pitts in their pioneering work on neural modeling [57]. Thus, a *formal neuron* computes a weighted sum of its inputs, subtracts a *threshold* w_0 and passes the result u through the nonlinearity

$$\operatorname{sgn}(u) = \begin{cases} 1 & \text{if } u \geqslant 0, \\ -1 & \text{otherwise.} \end{cases} \tag{3}$$

If $LTBF(n)$ denotes the set of linear threshold Boolean functions of n variables, then the usual choice is $\mathscr{F} = LTBF = \bigcup_{i \in V} LTBF(|P(i)|)$. Obviously, the family $\mathscr{F}$ contains functions f_i defined on all possible cardinalities of sets $P(i)$ of predecessors occurring in G. $\mathscr{F}$ may also consist of real-valued functions f_i defined on $\mathbb{R}^{|P(i)|}$, typical continuous nonlinear functions are the sigmoids used in back-propagation nets (see Section 4.2 and [66]). Following [45] an *architecture* is defined by the graph G and the set $\mathscr{F}$ of all possible node functions; it is denoted by $\mathscr{A} = (G, \mathscr{F})$.

We shall consider assignments F of some node function f_i in $\mathscr{F}$ to each computing node $i \in V$; such an assignment will be called a *configuration*. So an ANN should be specified by a pair (G, F), where G is the underlying oriented graph and F the choice of a function f_i for each node i in V.

[1] The bipolar representation is extensively used in the literature instead of $\{0, 1\}$ because it leads to some useful simplifications.

To complete the description of the model, its dynamic behavior has to be defined. In *feedback networks*, $G = (X, U)$ is a connected graph which contains at least one oriented circuit. The behavior of such networks depends on the order in which the units compute their output (parallel or sequential mode). If the graph G is circuit-free, the network is called a *feedforward network*. Only this second class of networks will be considered here. Feedback nets are extensively studied in [41, 49, 7, 37].

In a feedforward network, the set $I = X - V$ of input nodes consists of all nodes without predecessors (first layer) and the set $O \subseteq V$ of output nodes is the set of nodes without successors in G (last layer). All nodes i, except those in I, are assigned some function f_i. This kind of networks operates like combinatorial circuits where the information is processed layer by layer. An input $\boldsymbol{a} \in \mathbb{B}^{|I|}$ is imposed to the nodes of the first layer, and the nodes within the same layer evaluate their functions in parallel, from the second layer to the last one. The state $\boldsymbol{b} \in \mathbb{B}^{|O|}$ of the nodes in O is taken to be the output of the net. From now on we will set $m = |I|$ and $n = |O|$.

The type of associative problem considered here is to memorize into a feedforward ANN a set of p input–output pairs $(\boldsymbol{a}^k, \boldsymbol{b}^k) \in \mathbb{B}^m \times \mathbb{B}^n$ with $1 \leqslant k \leqslant p$; each pair is called an *association*. The *task* T is the set of associations that the network has to learn. Generally, every input vector in T is assigned to at most one output vector, so the task can be considered as a function which is partially defined on $\mathbb{B}^m$ and takes its values in $\mathbb{B}^n$. We shall say that an ANN *performs a task* T if it produces the output $\boldsymbol{b}^k$ for each input $\boldsymbol{a}^k$ $(1 \leqslant k \leqslant p)$. Whenever $\boldsymbol{a}^k = \boldsymbol{b}^k$, for all k, we have an *auto-association* problem (as in pattern recognition or content addressable memory), otherwise a *hetero-association* problem (as in classification or mapping approximation).

3. Threshold units

Let us consider a task $T = \{(\boldsymbol{a}^k, b^k)\}_{1 \leqslant k \leqslant p} \subseteq \mathbb{B}^m \times \mathbb{B}$ and a single unit with m inputs which computes an LTB function (i.e. $\mathscr{F} = LTBF(m)$). This *linear threshold unit* performs the task T if and only if the assigned $LTB(m)$ function f is an extension of the partial Boolean function associated to T (i.e. $f(\boldsymbol{a}^k) = b^k$ for $1 \leqslant k \leqslant p$). In other words, all points $\boldsymbol{a}^k$ with $b^k = 1$ lie in the first of the two half spaces defined by f, and those with $b^k = -1$ lie in the other one. Training a threshold unit is finding a node function $f \in LTBF(m)$ which performs the given task. T is said to be *linearly separable* if such a unit can realize all p associations. In the following subsection we shall present some basic results related to linear separability.

3.1. Linear separability

The number of linear separable tasks $T \subseteq \mathbb{B}^m \times \mathbb{B}$ of size $p = |T|$ is extremely small, for large m, compared to the total number 2^{2^m} of possible ones. To date no exact expression has been given for $N(m)$, the total number of LTB functions with m variables. The tighter lower bound known for $N(m)$ is $2^{m(m-1)/2+32}$ [60]. To find an upper

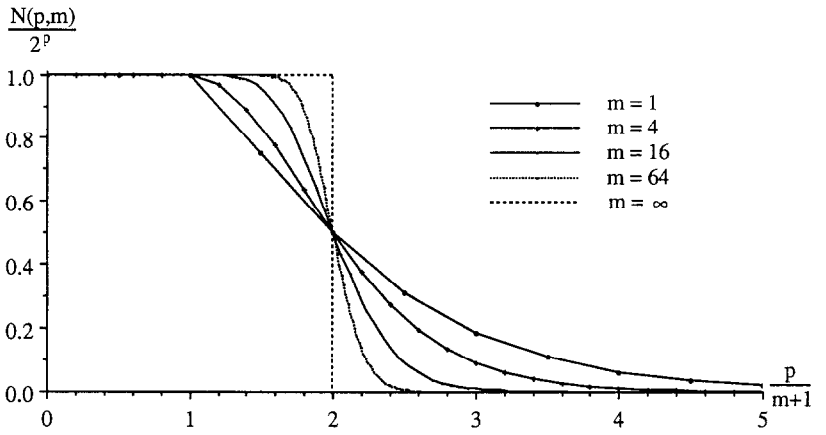

Fig. 1. The fraction of linear dichotomies of p points in $\mathbb{R}^m$ is plotted against the ratio $p/(m+1)$ for different values of m.

bound, one considers the problem of partitioning p points in $\mathbb{R}^m$. The p points are said to be in *general position* if there is no $(k-2)$-dimensional hyperplane containing k of them, for $k = 2, \ldots, m+1$. A *linear dichotomy* of p points in $\mathbb{R}^m$ is a partition of the points in two classes induced by the two open half spaces defined by a $(m-1)$-dimensional hyperplane. The number of different linear dichotomies of p points in general position in $\mathbb{R}^m$, denoted $N(p, m)$, is given by [26]

$$N(p, m) = 2 \sum_{i=0}^{\min(m, p-1)} \binom{p-1}{i}. \tag{4}$$

Fig. 1 plots the fraction $N(p, m)/2^{2^m}$ against the ratio $p/(m+1)$ for a few values of m (see [28]).

Note that infinitesimal moves of some points in a set S of points which are not in general position, may only increase the number of possible linear dichotomies of S. Thus $N(p, m)$ is an upper bound on the number of linear dichotomies of p points and the bound is reached when the points are in general position. Since p points in $\mathbb{B}^m$ are not necessarily in general position, $N(p, m)$ is an upper bound on the number of linearly separable tasks of size p and since $N(2^m, m) \leqslant 2^{m^2}$ for $m > 1$, 2^{m^2} is an upper bound on the cardinality of $LTBF(m)$ [4].

Clearly, $N(p, m) = 2^p$ when $p \leqslant m+1$, and therefore $m+1$ is the largest number p such that every task of p associations whose input vectors are in general position, is linearly separable. This quantity $m+1$ is frequently referred to as the *capacity of* $LTBF(m)$. Note that $m+1$ is the number of parameters determining an $(m-1)$-dimensional hyperplane $H = \{\boldsymbol{x} \in \mathbb{R}^m \mid \boldsymbol{x}^T \boldsymbol{w} = w_0\}$ which defines an $LTB(m)$-function. If the class $LTBF(m)$ is restricted to the LTB functions which can be defined by an $(m-1)$-dimensional hyperplane containing the origin of $\mathbb{R}^m$ (i.e. with $w_0 = 0$), we get

a new class denoted by $LTBF_0(m)$. Note that for any function f in $LTBF(m)$ there exists at least one $(m-1)$-dimensional hyperplane associated to f, containing no points of $\mathbb{B}^m$. This property is still true for an LTB_0 function f if and only if f is self-dual (i.e. $f(\boldsymbol{a}) = -f(-\boldsymbol{a})$ for every $\boldsymbol{a} \in \mathbb{B}^m$). It is easy to show that there is a bijection between the set of self-dual functions of $LTBF_0(m)$ and $LTBF(m-1)$. Then m is the maximum integer such that any task of this size which satisfies the two following conditions:

(i) the input vectors of T are in general position,

(ii) there is no antipodal input vectors $\boldsymbol{a}^k = -\boldsymbol{a}^l$ with $b^k = b^l$ for $1 \leqslant k, l \leqslant p$,

can be computed by an $LTBF_0$ function.

The concept of LTB functions can be extended to general threshold Boolean functions (TB functions) by considering general $(m-1)$-dimensional discriminant manifolds instead of hyperplanes. In other words, a function $f \in TB(m)$ is an LTB function in a new space $\mathbb{B}^h$, usually of higher dimension, so that f can be expressed as $g \circ \boldsymbol{e}$, where $\boldsymbol{e}$ is an application from $\mathbb{B}^m$ to $\mathbb{B}^h$ called the *structure* of f, and g is in $LTBF_0(h)$. If $TB(m, h, \boldsymbol{e})$ denotes the set of TB functions for a fixed structure $\boldsymbol{e}$, we can write $LTBF_0(m) = TBF(m, m, \mathbf{id}_m)$ and $\mathbf{id}_m$ is the identity of the hyper-cube $\mathbb{B}^m$; and $LTBF(m) = TBF(m, m+1, (\mathbf{id}_m, -1))$. Clearly, h is the maximum value such that any task of size h can be performed by a $TB(m, h, \boldsymbol{e})$ function, if $\boldsymbol{e}$ is such that the task $\{(\boldsymbol{e}(\boldsymbol{a}^k), b^k)\}_{1 \leqslant k \leqslant p=h} \subseteq \mathbb{B}^h \times \mathbb{B}$ satisfies the above conditions (i) and (ii).

A usual extension of the class $LTBF(m)$ consists of the *d-order threshold boolean functions* which are $TB(m, h, \boldsymbol{e})$ functions, where h is the number of combinations of 1 to d inputs and each e_i is the product of the associated inputs. This subclass of TB functions was investigated, among others, by Venkatesh and Baldi [77]. Another interesting subclass of TB functions was proposed in [24] and contains every $TB(m, h, \boldsymbol{e})$ functions, without restrictions on the e_i, but where h is bounded by a polynomial in m. Bruck showed that even if this class of *polynomial threshold functions* is wide, it is properly contained in the set of boolean functions which can be performed by a two-layer network whose set of node functions F is simply LTB.

In what follows, a linear threshold Boolean function will be a function in $LTB_0(m)$. This simplifies the notations without loss of generality. Indeed, all the results presented in this section can be generalized to any class of $TB(m, h, \boldsymbol{e})$ functions for a fixed structure $\boldsymbol{e}$. In Section 4.4, some results related to TB functions with adaptive structure will be presented.

From the computational point of view, training a single threshold unit is not easier than answering the following question known as *Linear Separability*:

Is a given task $T \subseteq \mathbb{B}^m \times \mathbb{B}$ of size p linearly separable?

The learning problem is equivalent to that of solving the system of linear inequalities

$$\check{A}\boldsymbol{w} > 0, \tag{5}$$

where the kth row of the $p \times m$ matrix $\check{A}$ is $\check{\boldsymbol{a}}^{k\mathrm{T}} = b^k \boldsymbol{a}^{k\mathrm{T}}$. So, an adequate linear programming algorithm (e.g. [47]) can solve *Linear Separability* in polynomial time in

p and m. Since large tasks have virtually no chance to be linearly separable, we are interested in a weight vector $\boldsymbol{w}$ that provides the right output for the largest fraction of input vectors in the task. Unfortunately, it is $\mathcal{NP}$-complete to decide whether a threshold unit can perform correctly at least K associations of T, when $1 \leqslant K < p$ [11]. In Section 4, we will see that a natural extension of *Linear Separability* becomes $\mathcal{NP}$-complete for general $TB(m)$ functions with adaptive structure $\boldsymbol{e}$.

In the next subsection, we shall describe alternative algorithms for finding a vector $\boldsymbol{w}$ that satisfies (5) as well as possible. These methods are thoroughly studied in the literature (see for instance [28, 49, 59, 12, 62]).

3.2. Training threshold units

The problem of finding an LTB function that performs a given task T can be formulated as that of minimizing a criterion function. The iterative process proposed by Rosenblatt [65], referred to as *perceptron rule*, minimizes the criterion:

$$\sum_{k \in M} -\check{\boldsymbol{a}}^{k\mathrm{T}} \boldsymbol{w}, \tag{6}$$

where M is the set of the indices of the associations misclassified by $\boldsymbol{w}$, and $\check{\boldsymbol{a}}^k = b^k \boldsymbol{a}^k$. The perceptron criterion is proportional to the sum of the distances from the misclassified input vectors to the hyperplane defined by $\boldsymbol{w}$. One step of this gradient descent is given by

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta_t \sum_{k \in M_t} -\check{\boldsymbol{a}}^k, \tag{7}$$

where $M_t = \{k \in \{1, \ldots, p\} \mid sgn(\boldsymbol{a}^{k\mathrm{T}} \boldsymbol{w}(t)) \neq b^k\}$ and η_t is the convergence parameter. It was proved [65] that, if a solution exists, this process converges in a finite number of steps even with a constant convergence parameter $\eta_t = \eta$. Unfortunately, the perceptron rule cannot be used to decide whether a task T is linearly separable because the time needed for convergence may increase exponentially with p [59]. Nevertheless, if an integer solution $\boldsymbol{w}^* \in Z^m$ exists, this procedure is guaranteed to find a solution in at most $(n+1)\|\boldsymbol{w}^*\|^2$ steps [33].

Another reasonable criterion is the minimization of the sum-of-squared-error:

$$\sum_{k=1}^{p} (c^k - \boldsymbol{a}^{k\mathrm{T}} \boldsymbol{w})^2, \tag{8}$$

where c^k has the same sign as b^k and can have any value. This allows to give a specific importance to each association k. If $\boldsymbol{A}$ denotes the $p \times m$ matrix whose rows are the $\boldsymbol{a}^{k\mathrm{T}}$ and $\boldsymbol{c}$ is the vector of the coefficients c^k, then (8) can be written as

$$\|\boldsymbol{c} - \boldsymbol{A}\boldsymbol{w}\|^2. \tag{9}$$

By definition of $A^{\dagger}$, the Moore–Penrose *pseudo-inverse* of A [5], the solution that minimizes (9) is

$$w = A^{\dagger}c. \tag{10}$$

$A^{\dagger}$ can be computed by the Greville algorithm in $O(\min\{m^2p, mp^2\})$. But $w = A^{\dagger}c$ can also be determined using the gradient descent process proposed by Widrow and Hoff [81], and known as *adaline* or *delta* rule:

$$w(t+1) = w(t) + \eta_t A^{\mathrm{T}}(c - Aw(t)). \tag{11}$$

It is worth noting that several variants of the above algorithms have been extensively studied in the framework of relaxation methods for solving large systems of linear inequalities [25, 71].

When a threshold unit is used as a content addressable memory, one is mainly interested in its error correction ability. Given the Boolean function f it performs, one defines, for each association (a^k, b^k) in the task T such that $f(a^k) = b^k$, the error-correcting power ρ^k as the maximum radius of the Hamming[2] ball $B^k \subseteq \mathbb{B}^m$ centered on a^k and such that the function f gives the output b^k for all points in B^k (i.e. $\rho^k = \max\{r \in \mathbb{N} \mid D(a^k, a) \leqslant r \Rightarrow f(a) = b^k\}$). If $f(a^k) = b^k$ for every k, then the overall error-correcting power of the unit is defined as $\rho = \min_{1 \leqslant k \leqslant p} \rho^k$. For an LTB function f associated to a separating hyperplane $H = \{x \in \mathbb{R}^m \mid x^{\mathrm{T}}w = 0\}$, it is easily verified that when $f(a^k) = b^k$, ρ^k increases with the distance $\check{a}^{k\mathrm{T}}w/\|w\|$ between a^k and H. Therefore, $\check{a}^{k\mathrm{T}}w/\|w\|$ is called the *stability* of the association k. This quantity is clearly negative whenever $f(a^k) \neq b^k$.

To find an LTB function that maximizes the stability ρ, Krauth and Mézard [50] suggested to bound $\|w\|$ by constraining each w_j in a fixed interval $[-\bar{w}, \bar{w}]$ and to maximize $\Delta = \min_{1 \leqslant k \leqslant p} \check{a}^{k\mathrm{T}}w$. This leads to the following linear program:

$$\begin{aligned} &\max \Delta \\ &\text{subject to} \quad \check{A}w \geqslant \Delta, \quad -\bar{w} \leqslant w_j \leqslant \bar{w} \quad \forall j = 1, \ldots, m, \end{aligned} \tag{12}$$

where Δ is the p-dimensional vector whose components are equal to Δ. Let Δ^* denote the optimal solution of (12). If $\Delta^* > 0$, then the task T is realized and the threshold unit corrects at least $\rho \geqslant \lceil \Delta^*/2\bar{w} \rceil - 1$ errors because flipping an input $a_j^k \leftarrow -a_j^k$ increments or decrements $\check{a}^{k\mathrm{T}}w$ by at most $2\bar{w}$. This polynomial algorithm gives an optimal solution for the error-correction criterion.

Note that in this section, the perceptron (7), the pseudo-inverse (10), the delta (11) and the maximization of the stability (12) algorithms are presented for LTB functions. However, the first three procedures can easily be generalized to the class of linear threshold real functions (LTR functions) defined on $\mathbb{R}^m$ and taking binary values. For LTR functions, the stability degree is taken as $\check{a}^{k\mathrm{T}}w/\|a^k\|\,\|w\|$. Thus, if the inputs of

[2] The Hamming distance between two vectors $x, y \in B^m$ is denoted by $D(x, y)$ and is equal to the number of different components.

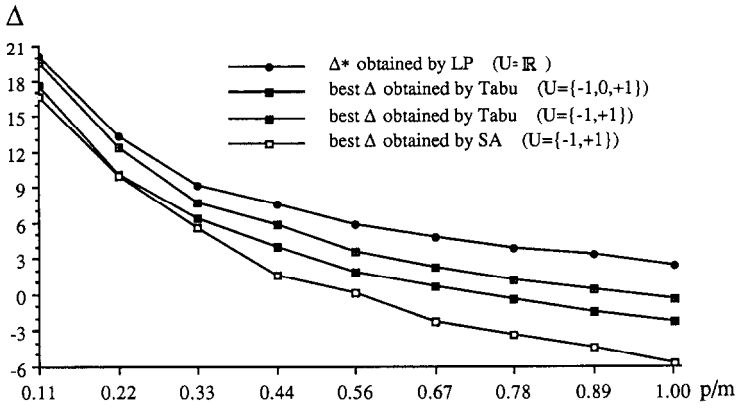

Fig. 2. The stability Δ for a random task is plotted against the ratio p/m for $m = 81$. The curves represent the optimal value of Δ in the case of real parameters and the best solutions obtained for the two- and three-value discretization with tabu search and simulated annealing. All values are averages on 20 random tasks.

the task are normalized ($-\bar{a} \leqslant a_j^k \leqslant +\bar{a}\ \forall k$), the maximization of Δ can once more be formulated as a linear program.

Since quantization of the parameters is a central problem of ANNs hardware implementation using numerical technology, several studies deal with restricted LTB_0 functions where the parameters of the corresponding hyperplane H can only take values in a discrete finite subset $U \subseteq \mathbb{R}$:

$$H = \{x \in \mathbb{R}^m \mid x^{\mathrm{T}} w = 0,\ w \in U^m\}.$$

For instance, a few algorithms have been proposed for training linear threshold units with binary weights $U = \{-1, +1\}$ [76, 63, 31].

In the discrete case, the maximization of the stability can be formulated as an integer linear program. Two heuristic methods, simulated annealing [48] and tabu search [36], are compared for $U = \{-1, +1\}$ in [10]; while in [56] tabu search is applied to the case $U = \{-1, 0, +1\}$ and the results are compared with those obtained for $w \in \mathbb{R}^m$. Typical curves are reported in Fig. 2. The stability is plotted against the ratio p/m with $m = 81$, for random associations with independent and equiprobable $+1$ and -1 components. The first curve represents Δ* obtained by linear programming when the parameters of the hyperplane are continuous. The second one is the best value of the stability Δ produced by tabu search when $U = \{-1, 0, +1\}$. The two last curves show the best values of Δ obtained, respectively, with tabu search and simulated annealing in the case where $U = \{-1, +1\}$. All values are averages on 20 tasks of p random associations. The value of p for which $\Delta = 0$ corresponds to the maximum number of random vectors that can be stored in the network simultaneously. Tabu search provides better results than simulated annealing for the two-level discretization and, as expected, the stability increases with

the number of discretization levels. It is worth noting that the storage capacity estimate given by tabu search is in agreement with those obtained subsequently by exact enumeration for $m \leqslant 25$ [52] and by analytical derivation using statistical mechanics techniques [51].

The above algorithms can also be used for training feedback networks like the Hopfield model where all units are totally connected [41]. An adaptation of tabu search to the design of this kind of networks is described in [9, 80].

4. Feedforward architectures

4.1. Multi-layer networks

As pointed out in Section 3, a linear programming algorithm can determine, in polynomial time, a linear separator for any linearly separable task. Unfortunately, the class of LTB functions is too small in practice. Since less than 2^{m^2} of the 2^{2^m} possible Boolean functions are linearly separable, large real-world problems have virtually no chance to be learnable by 2-layer networks with $\mathscr{F} = LTBF(m)$. The same kind of limitations apply to other nonlinear node functions, as sigmoids, whose $(m-1)$-dimensional discriminant manifolds are not restricted to be linear.

A way to improve the computational power of the system is to add "hidden" nodes that are neither input nor output nodes. This is equivalent to extending the set of variables of the system by defining new dependent variables that are LTB functions of the independent variables associated to the input nodes and of previously defined dependent variables.

In the case of binary inputs, a single hidden layer (with at most 2^{m-1} nodes) is clearly sufficient to implement any Boolean function of m variables. For continuous inputs, it is known [54] that 4-layer networks whose linear threshold units have real inputs can simulate any function from $\mathbb{R}^m$ to $\mathbb{B}$. Recently, a similar result has also been established for continuous sigmoidal node functions. It was shown in [27] that 3-layer networks with sigmoidal functions are able to approximate any Borel measurable function from one finite dimensional space to another with an arbitrary degree of accuracy, provided sufficiently many hidden nodes are available.

Notice that the capabilities of multi-layer nets stem from the node nonlinearities. If the computing nodes were performing linear functions, a 2-layer network with an appropriate configuration F could duplicate the computation carried out by any multi-layer net.

A feedforward architecture $\mathscr{A} = (G, \mathscr{F})$ and a configuration $F: V \to \mathscr{F}$ define a mapping from the input to the output space

$$\phi_F^{\mathscr{A}}: \mathbb{B}^{|I|} \to \mathbb{B}^{|O|},$$

which fully characterizes the behavior of the network. As mentioned above, a task $T = \{(\boldsymbol{a}^k, \boldsymbol{b}^k)\}_{1 \leqslant k \leqslant p} \subseteq \mathbb{B}^{|I|} \times \mathbb{B}^{|O|}$ can be considered as a partially defined function

from $\mathbb{B}^{|I|}$ to $\mathbb{B}^{|O|}$. T is a set of constraints on the mapping that the network has to perform:

$$\phi_F^{\mathscr{A}}(\boldsymbol{a}^k) = \boldsymbol{b}^k \quad \text{for } 1 \leqslant k \leqslant p.$$

Thus, $\phi_F^{\mathscr{A}}$ may be any extension of the partial function corresponding to the task. The *learning problem* can then be defined as follows:

Given a feedforward architecture $\mathscr{A} = (G, \mathscr{F})$ and a task T, find a configuration F for $\mathscr{A}$, i.e. an assignment of a response function f_i to each computing node $i \in V$, such that the derived mapping $\phi_F^{\mathscr{A}}$ includes the task T.

Before discussing the main results related to the complexity of the learning problem, we shall briefly mention two well-known methods for training multi-layer networks.

4.2. Learning algorithms

Although feedforward networks with hidden layers overcome many limitations of 2-layer nets, it is not straightforward to train such architectures because there is no direct way to know whether the output of a hidden node is correct for a particular input vector in the task. This problem is known as "credit assignment". The famous *back-propagation* algorithm [79, 66] circumvents the "credit assignment" problem by requiring continuous differentiable node functions defined on $\mathbb{R}^{P(i)}$. Typical functions used are sigmoids applied to the weighted sum of the inputs, i.e.

$$f_i(\boldsymbol{x}) = \frac{1}{1 + \exp(w_0 - \sum_{j=1}^{|P(i)|} w_{ij} x_j)}.$$

The back-propagation procedure is a gradient method for minimizing an error function, which is equal to the mean squared difference between the desired and the actual outputs, in the space of the weights w_{ij} and thresholds w_0. It is in fact a generalization of the delta rule (11) to feedforward networks with hidden layers and differentiable node functions. Although this algorithm is not guaranteed to converge towards a global minimum of the error, it has been successful on a variety of test problems like recognizing symmetries [66] as well as on some real world problems [69, 53]. However, the procedure turns out to be very time-consuming even for medium size tasks and the computational burden is still very high when more efficient optimization techniques are used to minimize the mean squared error [40].

A different algorithm has been proposed [39] for training multi-layer networks composed of linear threshold units. For the sake of simplicity, we consider 3-layer networks with a single output node ($|O| = 1$) and $\mathscr{F} = LTBF$. The task is a set of p associations $(\boldsymbol{a}^k, \boldsymbol{b}^k) \in \mathbb{B}^m \times \mathbb{B}$ with $1 \leqslant k \leqslant p$. For a given configuration F of the net, each input $\boldsymbol{a}^k$ triggers a set of outputs for the internal nodes denoted by the vector $\tilde{\boldsymbol{a}}^k \in \mathbb{B}^h$, where h is the size of the hidden layer ($h = |X| - m - 1$). $\tilde{\boldsymbol{a}}^k$ is called the *internal representation* of $\boldsymbol{a}^k$. Whereas back-propagation minimizes the mean squared error only with respect to the weights, this method considers also the internal

representations associated to the inputs in the task as the variables of the training procedure.

The approach is based on the observation that for any given set of internal representations $\{\tilde{\boldsymbol{a}}^k\}_{1\leqslant k\leqslant p}$ the linear threshold Boolean functions assigned to the hidden and output nodes can be determined (if they exist) by the perceptron rule. Indeed, the ith hidden node should perform the subtask $\{(\boldsymbol{a}^k, \tilde{a}_i^k)\}_{1\leqslant k\leqslant p}$ and the output node should realize the subtask $\{(\boldsymbol{a}^k, b^k)\}_{1\leqslant k\leqslant p}$.

The learning problem is expressed here as the search for proper internal representations, but it can also be viewed as the minimization of an error function which depends on the weights as well as on the internal representations [67].

The algorithm of choice of hidden internal representations (CHIR) works as follows:

(1) Choose an initial configuration $F_o: V \to LTBF$ of the network.

(2) Determine the set of internal representations $\{\tilde{\boldsymbol{a}}^k\}_{1\leqslant k\leqslant p}$.

(3) Try to find with the perceptron procedure a function $f_o \in LTBF(h)$ for the output node that performs the subtask $\{(\tilde{\boldsymbol{a}}^k, b^k)\}_{1\leqslant k\leqslant p}$. If such a function is obtained in less than *nbmax* steps, CHIR stops. Otherwise the perceptron procedure is interrupted and the current configuration F_c misclassifies a number

$$E = |\{k \mid f_o(\tilde{\boldsymbol{a}}^k) \neq b^k\}|$$

of input vectors $\boldsymbol{a}^k$ in the task.

(4) Determine a new set of internal representations which gives $E = 0$ for the current output function f_o. The new set is obtained by looking sequentially to the p current internal representations and by flipping the components of $\tilde{\boldsymbol{a}}^k$ that lead to the largest decrease in the error E. This is repeated until E vanishes.

(5) For each hidden node $i \in V - O$, try to find with the perceptron rule a function $f_i \in LTBF(|I|)$ that performs the subtask $\{(\boldsymbol{a}^k, \tilde{a}_i^k)\}_{1\leqslant k\leqslant p}$. If a set of appropriate functions is obtained in less than *nbmax* steps, CHIR stops. Otherwise the perceptron procedure is interrupted and one goes back to 2.

As for back-propagation, there is no guarantee of convergence and the overall process is iterated a maximum number of times. The CHIR algorithm is, however, more efficient and much faster than back-propagation for some standard classification problems as *symmetry* ($b^k = 1$ iff $a_i^k = a_{m+1-i}^k \; \forall i = 1, \ldots, m$), *l-contiguity* ($b^k = 1$ iff $\exists i$ such that $a_i^k = a_{i+1}^k = \cdots = a_{i+l-1}^k = 1$) and *parity* ($b^k = 1$ iff the number of $a_i^k = 1$ is odd) on which both procedures have been tested. If has also been generalized to networks with multiple outputs and multiple hidden layers [38].

We pointed out in Section 3 the relative effectiveness of linear programming and of the perceptron rule for learning linearly separable tasks in feedforward networks with a single layer of computing nodes. Unfortunately, there are no such guarantees for the learning procedures applicable to nets with more than one layer of computing nodes. Moreover, it is widely acknowledged that the amount of time required to store a task into a given multilayer network grows prohibitively fast with the size of the architecture (see for instance [72]). This scaling-up issue is one of the essential problems in

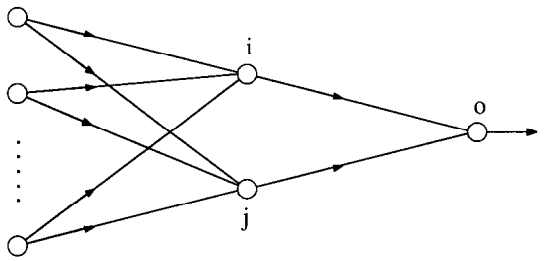

Fig. 3. Simple feedforward net with m input nodes, 2 hidden nodes i and j connected to all inputs and one output node o receiving its inputs only from the hidden layer.

neural network research, it addresses the fundamental questions related to the existence of efficient algorithms for learning in large feedforward networks. The study of the roots of intractibility should also suggest design constraints that any architecture must satisfy in order to be trainable in polynomial time.

4.3. *Complexity of learning*

To investigate the computational complexity of learning in multi-layer networks, one assumes that the architecture $\mathscr{A} = (G, \mathscr{F})$ and the task T are given a priori. Thus training the network is finding a configuration which provides the right output for all input vectors in T.

Let us consider the simple 3-layer network which consists of m input nodes, two hidden nodes i and j, and one output node o. As shown in Fig. 3, the nodes i and j are connected to all inputs and the output node o is connected to both hidden nodes. Each computing node performs a linear threshold Boolean function, that is $\mathscr{F} = LTBF$, and the task T is a given set of $O(m)$ associations $(\boldsymbol{a}^k, b^k) \in \mathbb{B}^m \times \mathbb{B}$, $1 \leqslant k \leqslant p$.

In this particular case, the decision version of the learning problem becomes:

Given such an architecture $\mathscr{A} = (G, LTBF)$ and such a task T, is there an assignment of LTB functions to the 3 computing nodes that produces outputs consistent with T?

This problem has a simple geometrical interpretation. The input vectors $\boldsymbol{a}^k \in \mathbb{B}^m$ can be viewed as points in $\mathbb{R}^m$, labeled $b^k = +1$ or -1, and the linear threshold Boolean functions associated to the hidden nodes i and j as $(m-1)$-dimensional hyperplanes in $\mathbb{R}^m$. The hyperplanes H_i and H_j divide the space into at most 4 quadrants that correspond to the 4 possible pairs of output for nodes i and j. Obviously, if H_i and H_j are parallel they form only 2 or 3 different regions. Since the output node o can distinguish only points lying in different quadrants, the answer to the above question is true if and only if either of the following condition is satisfied:

- a single plane (H_i or H_j) separates the $+1$ points from the -1 ones,
- H_i and H_j are such that either one quadrant contains all $+1$ points and only those or one quadrant contains all -1 points and only those.

So, the considered decision problem is equivalent to the 2-*Linear Separability* question:

Can two sets of O(m) Boolean vectors in m-dimensional space be linearly separated by two hyperplanes?

Blum and Rivest proved [20] by reduction from set splitting (hypergraph 2-colorability) [35]:

Proposition 4.1. *2-Linear Separability is $\mathcal{NP}$-complete.*

This result shows that linear separability, which is in $\mathcal{P}$ for a single hyperplane, becomes intractable as soon as 2 hyperplanes are available. Moreover, it is good evidence that one cannot circumvent the inherent complexity of the learning problem by considering only very simple and regular architectures. Some interesting consequences related to other simple networks are presented in [21].

The proof is valid only for linear threshold Boolean functions, i.e. in the case where $\mathcal{F} = LTBF$. Open questions are whether the learning problem is easier for richer sets of node functions as sigmoids or for networks with $h \geqslant 3$ hidden units. Another feature of the approach is the scaling up of the number of associations in the task and of the size of the input $|I|$. What happens when the size of the architecture grows while he size of the task remains constant?

The first $\mathcal{NP}$-completeness proof of the learning problem was given by Judd [43] in a more general framework which led to a coherent set of results [44, 45]. The end of this subsection is devoted to the most relevant ones.

The decision version of the learning problem is the *Performability* question ($Perf_{(\mathcal{A},T)}$):

Given a feedforward architecture $\mathcal{A} = (G, \mathcal{F})$ and a task T, is there a configuration F for $\mathcal{A}$ such that the derived mapping $\phi_F^{\mathcal{A}}$ includes the task T?

The following theorem is due to Judd [43].

Theorem 4.2. *$Perf_{(\mathcal{A},T)}$ is $\mathcal{NP}$-complete when $\mathcal{F}$ consists only of AND and OR functions.*

The proof is by reduction from $3SAT$ [35] and it is valid for any richer set $\mathcal{F}$ of Boolean functions as, for instance, LTBF. When $\mathcal{F}$ contains properly the LTB functions, the polynomial-time reduction holds but the problem is no longer in $\mathcal{NP}$, hence it is $\mathcal{NP}$-hard. The result is also true for any set of bounded and monotonic continuous node functions applied to a linear combination of the inputs [43, 45]. In particular, it holds for the sigmoidal functions used in back-propagation. This is good evidence that the difficulty of *Performability* and of the learning problem is independent of the type of node functions and derives from the connectivity pattern of the network.

The proof scales up the size of the architecture but keeps the number of associations in the task and the size of the input $|I|$ constant. Indeed, $Perf_{(\mathcal{A},T)}$ is $\mathcal{NP}$-complete even when $|T| = 3$ and $|I| = 2$. It is worth noting that the problem remains

$\mathcal{NP}$-complete if one relaxes the criterion of success, i.e. if one requires only that strictly more than two-thirds of the associations in the task are realized. *Performability* is still intractable when all these restrictions are imposed simultaneously.

We shall now discuss some architectural constraints that yield subcases solvable in polynomial time. One might also try to identify tractable problems by placing some restrictions on the task. Since $Perf_{(\mathcal{A},T)}$ is $\mathcal{NP}$-complete even when restricted to 3-layer architectures where each node has at most 3 predecessors, we focus on nets with bounded depth and unbounded width. In order to introduce the concept of *shallow networks*, we need a few definitions.

Definition 4.3. Consider an architecture $\mathcal{A} = (G, \mathcal{F})$.

- The support cone of a computing node $i \in V$ is the set of nodes $j \in V$ that can influence its output, i.e. the set of computing nodes j for which there is in G a path from j to i. It is denoted by $sc(i)$.
- A partial configuration for a node $i \in V$ is an assignment of functions in $\mathcal{F}$ to every node in $sc(i)$.
- The support cone configuration space (*sccs*) for an output node $o \in O$ is the set of all possible partial configurations for $sc(o)$. Note that the size of a *sccs* is finite when $\mathcal{F}$ consists of Boolean functions.

A family of architectures is *shallow* if the largest size of a support cone configuration space of any architecture is bounded by a constant. This is a way to bound simultaneously the depth of the net, the maximum number of predecessors for each node and the number of functions in $\mathcal{F}$. It ensures that all *sccss* can be searched exhaustively in constant time.

Unfortunately, limiting the size of the *sccss* is not sufficient to find tractable cases. Indeed, the proof of Theorem 4.2 shows, by default, that $Perf_{(\mathcal{A},T)}$ is $\mathcal{NP}$-complete for shallow architectures. Some locality constraints are needed to prevent that the support cones of the output nodes have too intricate intersection patterns.

Let us consider, for any architecture $\mathcal{A} = (G, \mathcal{F})$, a *support cone interaction graph* (Y, E) (SCIgraph) that represents the interactions between the support cones of the output nodes in G. A node $y_i \in Y$ is associated with each output node $i \in O$ and $[y_i, y_j] \in E$ iff $sc(i) \cap sc(j) \neq \emptyset$. An example of a feedforward architecture together with its SCIgraph is given in Fig. 4.

There is a constraint on the SCIgraph which leads to feasible problems. It involves the notion of *k-tree* which is a generalization of a tree.

Definition 4.4. A graph G is a k-tree if it satisfies either of the following conditions:

(i) G is the complete graph, K_k;

(ii) G has a vertex v of degree k whose neighbors form a clique, and the induced graph obtained by removing v from G is also a k-tree.

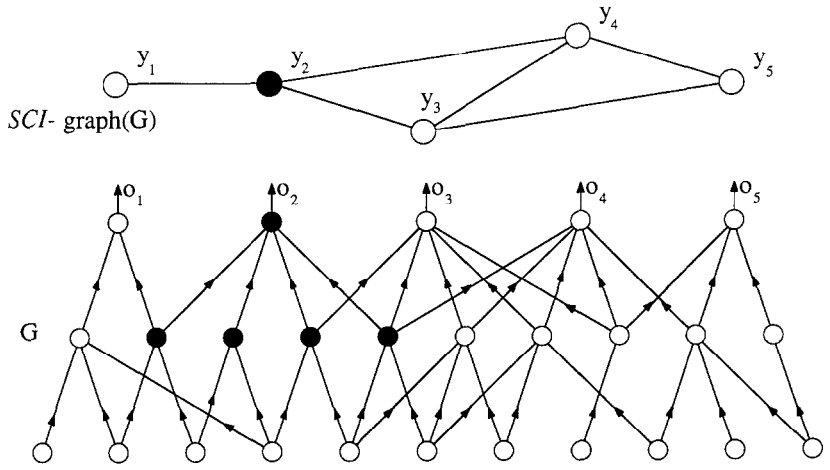

Fig. 4. Three-layer feedforward architecture with 11 input nodes and 5 output nodes together with its SCI graph. The support cone of the second output node (o_2) as well as the corresponding node (y_2) in the SCI graph are depicted in black.

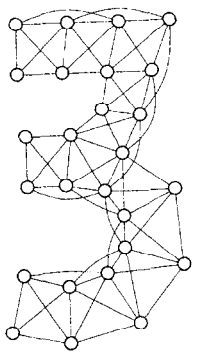

Fig. 5. Example of a 3-tree. Note that every node belongs to a clique of cardinality 4.

A 1-tree is a classical tree and a 3-tree is shown in Fig. 5. A graph G is a *partial k-tree* if it can be transformed into a k-tree by introducing a few edges in G. Thus, complete graphs K_n are not partial k-trees for any $k < n - 1$ because there exists no $k < n - 1$ such that K_n can be embedded in a k-tree. Also *grid*-graphs $G_{n_1 \times n_2}$, which have a very simple and regular local topology but which expand in 2 dimensions, are not partial k-trees for any $k < \min\{n_1, n_2\}$.

The next theorem summarizes the known tractable cases [45].

Theorem 4.5. *$Perf_{(\mathcal{A},T)}$ restricted to shallow architectures $\mathcal{A}$ whose SCIgraph are partial k-trees can be solved in polynomial time, provided an embedding of the SCIgraph in a k-tree is given.*

In fact, there is a dynamic programming algorithm which determines in $O(n^k)$ steps if the architecture can perform the task, where $n = |X|$ is the size of the architecture. This algorithm is only for purposes of demonstrating the polynomial time complexity of the problem: it searches exhaustively the set of all possible partial configurations. The result holds even if one relaxes the definition of shallow architectures by allowing the largest *sccs* size to be polynomial in n. It remains also true if $k = O(\log n)$ instead of being constant.

The theorem given in [45], which assumes that the SCIgraphs of the architectures have a limited *tree-width*, is equivalent to Theorem 4.5 because the tree-width of a graph is by definition the smallest integer k such that the graph is a partial k-tree. A weaker result is established in [44] by bounding the *bandwidth* of the SCIgraph. The bandwidth of a graph $G = (X, E)$ is the smallest integer k for which there exists a numbering $n: X \to \{1, \ldots, |X|\}$ of the nodes such that for any edge $[i, j] \in E$, $|n(i) - n(j)| \leqslant k$. It turns out that the bandwidth of a graph is greater or equal to its tree-width.

Performability is thus an additional example of $\mathcal{NP}$-complete or $\mathcal{NP}$-hard problem that is solvable in polynomial time when restricted to partial k-trees and when an embedding in a k-tree is given [14]. Note that the restriction is here on the auxiliary SCIgraph rather than on the graph G representing the topology of the architecture. The embedding is required a priori because the problem of finding the smallest number k such that a given graph is a partial graph of a k-tree (i.e. finding its tree-width) is $\mathcal{NP}$-complete [13].

Abu-Mostafa has investigated the complexity of the learning problem [3] in some other interesting particular cases [3]. Training feedforward networks of threshold units turns out to be $\mathcal{NP}$-complete even if the signs of the weights are given and one only needs to find their magnitude or if the absolute value of the weights are fixed and one only has to select their signs. The first hypothesis is biologically motivated because some synapses are predisposed to be excitatory and others inhibitory. The proofs are by simple reduction from *Performability*.

Since all these results are based on a worst-case analysis, they provide helpful guidelines for further investigations into the average-case complexity of the learning problem and they motivate the development of heuristic learning algorithms. The study of polynomially solvable cases should provide a substantial help in the design of heuristics for the general case.

4.4. Constructive training algorithms

The previous results show it is extremely unlikely that any procedure which only modifies the configuration of a given feedforward architecture can learn in time polynomial in the size of the task and of the architecture. However, a new family of learning algorithms has recently been proposed [58, 16, 32, 55] which can change the topology of the architecture as well as modify its configuration. In other words, the

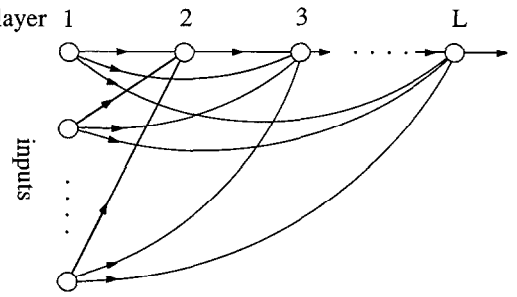

Fig. 6. L-layer feedforward architecture generated by the growth algorithm. There are m input nodes in the first layer and a single node in the other $L-1$ layers. The lth computing node ($l \geqslant 2$) is connected to the preceding $(l-1)$th node as well as to the input nodes.

architecture has not to be guessed in advance, it is an output of the training procedure as well as its configuration.

In this framework, the learning problem becomes:

Given a task T, find a feedforward architecture $\mathscr{A} = (G, \mathscr{F})$ and a configuration F for $\mathscr{A}$ such that $\phi_F^{\mathscr{A}}$ includes the task T.

These methods construct the network incrementally by adding units as they are needed. The *upstart* method [32] starts from an architecture with only the input and output layers and adds units in a single hidden layer as long as the network does not realize correctly all associations of the task. The *tiling* algorithm introduced in [58] builds the architecture node by node and layer by layer until convergence, i.e. until the given Boolean task $T = \{(\boldsymbol{a}^k, b^k)\}_{1 \leqslant k \leqslant p} \subseteq \mathbb{B}^m \times \mathbb{B}$ is performed. Thus the number of layers as well as the number of units are not fixed in advance. Unlike the back-propagation and the CHIR algorithms, convergence of these two methods is guaranteed by a simple argument.

For the sake of simplicity, we shall describe a basic variant of the tiling algorithm which creates only one node in each hidden layer [61, 34]. Although it has been devised for Boolean tasks with a single output, it can be adapted to tasks with real-valued inputs.

We consider special L-layer feedforward architectures with m input nodes in the first layer and with a single node in the other $L-1$ layers. The lth computing node ($l \geqslant 2$) is connected not only to the preceding $(l-1)$th node but also to the input nodes (see Fig. 6); it performs a LTB function.

Suppose that the lth node has been generated and that its function has been selected. The current configuration of the l-layer architecture produces the desired output for all except e_l of the p inputs $\boldsymbol{a}^k$ in the task T. If the number of errors e_l of the lth computing node is greater than 0, there is at least one LTB function for the new $(l+1)$th node such that $e_{l+1} \leqslant e_l - 1$. Such a function $f_{l+1} \in LTBF(m+1)$ can be determined easily. Let $(\boldsymbol{a}^{k_0}, b^{k_0})$ be one of the e_l associations which are not performed correctly by the l-layer network, then a possible

f_{l+1} is defined by

$$w_j = b^{k_0} a_j^{k_0} \quad \text{for } 0 \leqslant j \leqslant m$$

and $w = m$, where w is the weight connecting the lth hidden node to the $(l+1)$th node. The $(l+1)$-layer network realizes the k_0th association, because the $(l+1)$th node gives the output

$$sgn\left(-mb^{k_0} + b^{k_0} \sum_{j=0}^{m} a_j^{k_0} a_j^{k_0} \right) = b^{k_0}$$

for the input vector $\boldsymbol{a}^{k_0}$, and the $p - e_l$ associations which were performed by the l-layer net, because the $(l+1)$th node gives the output

$$sgn\left(mb^k + b^{k_0} \sum_{j=0}^{m} a_j^{k_0} a_j^k \right) = b^k$$

for the $p - e_l$ corresponding input vectors $\boldsymbol{a}^k$ with $k \neq k_0$. Thus, an algorithm that minimizes the error function e_l by adding such computing nodes will build at most p layers before obtaining a network with zero error.

To generate as few hidden layers as possible, one searches for a function f_{l+1} which yields the smallest number of errors. Therefore, a simple variant of the perceptron procedure (see Section 3.2) is used [34]. The *pocket algorithm* works like the perceptron rule but it keeps in memory (in the pocket) the weight vector which has produced the smallest number of errors. Gallant proved that these weights minimize the number of errors with a probability converging to 1 when the number of iterations tends to infinity. If the pocket procedure starts from one of the functions defined above, the constructive algorithm creates at most p units.

Given a Boolean task $T = \{(\boldsymbol{a}^k, b^k)\}_{1 \leqslant k \leqslant p} \subseteq \mathbb{B}^m \times \mathbb{B}$, the *growth algorithm* proceeds as follows:

(1) Build the initial 2-layer architecture with m input nodes in the first layer and a single node in the second layer. Set $l = 2$ and $T_2 = T$.

(2) Try to find with the pocket algorithm a function $f_l \in LTBF(m+1)$ for the lth node that performs the task T_l. If such a function is obtained in less than *nbmax* steps, the growth algorithm stops; otherwise, the pocket procedure is interrupted.

(3) Create the lth layer with a single node connected to the $(l-1)$th node and to the m input nodes. This new node should perform the task

$$T_l = \left\{ \left(\begin{pmatrix} \boldsymbol{a}^k \\ g^k \end{pmatrix}, b^k \right) \right\}_{1 \leqslant k \leqslant p} \subseteq \mathbb{B}^{m+1} \times \mathbb{B},$$

where g^k is the output given by the $(l-1)$th node for the input vector $\boldsymbol{a}^k$. Return to 2.

This growth algorithm adds new computing nodes and layers until the desired task is performed. The process is tantamount to embed the initial m-dimensional problem in a space of higher dimension ($m+1$ rather than m) by defining new dependent variables (corresponding to the new nodes) that are LTB functions of the independent variables associated to the m inputs and of a previously defined dependent variable.

This simple algorithm generates a larger number of nodes than the tiling method, but it suffices to establish the following result concerning the decision version of the learning problem ($Store_T$):

Given a task T, is there a feedforward architecture $\mathscr{A} = (G, \mathscr{F})$ and a configuration F for $\mathscr{A}$ such that $\phi_F^{\mathscr{A}}$ is an extension of T?

Theorem 4.6. *$Store_T$ can be solved in linear time for any Boolean task $T \subseteq \mathbb{B}^m \times \mathbb{B}^n$.*

Since the growth algorithm builds at most $p = |T|$ layers with a single node and since a similar chain of hidden nodes can be constructed for each output of the task, the algorithm is linear in p, m and n. Thus by Boolean task is learnable in feasible time if one considers procedures that have the freedom to change the topology of the architecture as well as to modify its configuration.

Actually, Baum has shown [15, 16] that any deterministic Turing machine (DTM) [35] of size σ which converges in time τ can be simulated by a feedforward network of size polynomial in σ and τ. This means that feedforward architectures with $\mathscr{F} = LTBF$ are capable of learning in polynomial time any task which can be represented as a DTM of polynomial size in m, n and p. In fact, this is true for any node function set from which one can build *AND*, *OR* and *NOT* functions for all possible cardinalities of predecessors.

The results obtained with the tiling and upstart methods for some standard classification problems are promising, but these algorithms are not efficient enough, in terms of the size of the networks they construct, to be used in real-world problems. In particular, effective extensions to multiple outputs networks are needed.

5. The generalization issue

So far we considered learning in ANN as the mere memorization of a given set of associations in a feedforward or a feedback architecture. For feedforward architectures this problem turns out to be hard if the architecture is fixed in advance and can be solved in linear time if the learning algorithm is allowed to vary the topology of the network. However, we shall now see why as small as possible networks are needed.

Since an arbitrary task can be stored in linear time on any sequential machine, one would hesitate to use ANNs just for distributed memorization and parallel retrieval. With these models we are in fact trying to achieve more than information storage. Usually, the p associations in the task $T \subseteq \mathbb{B}^m \times \mathbb{B}$ are examples of an underlying Boolean function $h^*: \mathbb{B}^m \to \mathbb{B}$, i.e. $T = \{(\boldsymbol{x}^k, h^*(\boldsymbol{x}^k))\}_{1 \leq k \leq p}$, and the goal of learning is to find a feedforward network with m input nodes and one output node defining a mapping $h: \mathbb{B}^m \to \mathbb{B}$ including the task T and which is a good approximation of h^* on all $\mathbb{B}^m$. In other words, we look for a network that not only performs the task but also *generalizes* it in a sense to be precised. This problem is referred to in the literature as *learning from examples.*

Let us consider learning algorithms that can only modify the configuration F of a given architecture $\mathscr{A}$. If H denotes the set of all Boolean functions $h: \mathbb{B}^m \to \mathbb{B}$ which can be implemented by $\mathscr{A}$, a learning procedure selects a function $h \in H$ that agrees as well as possible with h^* on the training set, i.e. on the input vectors in the task T. The question is to know whether such a function h is also consistent with h^* for the input vectors which are not in T. Notice that when we choose the architecture $\mathscr{A}$ we are guessing that the set H associated to $\mathscr{A}$ contains at least one function h which is a good approximation of the target function h^*. Frequently, it is even assumed that h^* belongs to H.

In general, the performances of a feedforward network during the testing phase are not closely related to its performances on the task. Indeed, the overall mapping performed by the net can be any extension of the task T. The behavior of the system for input vectors which do not occur in T, depends on how well the task represents the target function h^* and on how the testing examples are chosen.

Valiant has recently proposed a probabilistic definition of learning from examples which formalizes generalization as prediction [73]. This provides a sensible framework to consider generalization in feedforward neural networks.

Let us assume that a probability distribution P is defined on the input space $\mathbb{B}^m$ and that the associations in the task as well as the testing input vectors are drawn according to the same fixed but unknown distribution. This plausible assumption guarantees that, with high probability, the task T consists of typical examples of h^*. The probability that h does not agree with h^* for input vectors which are randomly selected according to the distribution P, i.e.

$$P\{x \in \mathbb{B}^m \mid h(x) \neq h^*(x)\},$$

is a natural measure of the accuracy to which h approximates the target function h^*. If this probability of error is at most ε with $0 \leqslant \varepsilon < 1$, we say that h is an ε-approximation of h^*.

A confidence parameter δ, $0 \leqslant \delta < 1$, is introduced in order to cope with the (unlikely) cases where the task is not a representative set of random examples of h^*. The learning algorithm is then allowed to produce a poor approximation of h^* with probability at most δ.

A *probably approximately correct* (*PAC*) learning algorithm is a procedure which produces, for any given accuracy and confidence parameters ε and δ, an ε-approximation h of h^* with probability greater than $1 - \delta$. Note that it is a *distribution-independent* definition of learning from examples, because h must be an ε-approximation of f for any distribution P.

We are clearly interested in PAC algorithms that (i) need a small number of examples $p = |T|$, (ii) are computationally efficient and (iii) work for all h^* in a given set H^* of Boolean functions. In particular, we would like PAC learning algorithms whose running time is polynomial in $1/\delta$, $1/\varepsilon$, m and p for any h^* in H^*. When δ and ε are small, this type of procedure determines with high probability a configuration

F for the given architecture which accurately approximates any function $h^* \in H^*$ in polynomial time in the size of h^*.

Baum and Haussler [18] have recently derived upper and lower bounds on the size p of the task that a feedforward network must perform correctly in order to have some confidence that it will provide the correct output for future input vectors drawn from the same distribution P. These results involve the notion of *Vapnik–Chervonenkis* (VC) *dimension* which is a generalization of the capacity of LTB functions discussed in Section 3.

Let Z be a set of Boolean-valued functions defined on some space S—for instance $\mathbb{R}^m$ or $\mathbb{B}^m$. A subset $Y \subseteq S$ is said to be *shattered* by Z if each of the $2^{|Y|}$ possible dichotomies of Y is consistent with at least one Boolean-valued function $z \in Z$.

Definition 5.1. The VC dimension of a set of Boolean-valued functions Z is the cardinality of the largest subset $Y \subseteq S$ which is shattered by Z; it is denoted by $VCdim(Z)$.

Since the empty set is shattered by any set Z, the VC dimension is well defined. It depends on the combinatorial characteristics of the Boolean-valued functions in Z and therefore of the subsets $z^{-1}(1) \subseteq S$ with $z \in Z$. In fact, the VC dimension of Z is equivalent to the *density* of the subsets $z^{-1}(1) \subseteq S$ mentioned in [68]. Note that only one subset Y of cardinality of $VCdim(Z)$ needs to be shattered by Z; in this sense $VCdim(Z)$ is a best case parameter of Z. Since a subset Y requires at least $2^{|Y|}$ distinct Boolean-valued functions to be shattered, $VCdim(Z) \leqslant \log(|Z|)$. If Z consists of Boolean functions, we have $VCdim(Z) \leqslant 2^m$. Obviously, $VCdim(Z) = 2^m$ when Z is the set of all Boolean functions. According to Eq. (5) of Section 3, $N(p, m) = 2^m$ for any $p \leqslant m + 1$. Therefore, $VCdim(LTBF(m)) = VCdim(LTRF(m)) = m + 1$. The VC dimension for other sets Z can be found in [78].

The work of Vapnik and Chervonenkis on the uniform convergence of empirical probability estimates and its applications to pattern recognition [74, 75], is relevant to the PAC learning model suggested by Valiant [2]. Blumer et al. have shown that the VC dimension is a useful parameter for studying distribution-independent learning. They derived the following general result (Theorem A3.1 in [22]):

Theorem 5.2. *Let Z be a set of well-behaved*[3] *Boolean-valued functions, $0 < \varepsilon, \delta < 1$ and $0 < \gamma \leqslant 1$ be a slack parameter. Suppose that p training examples with*

$$p \geqslant \max\left\{\frac{8}{\gamma^2\varepsilon}\ln\left(\frac{8}{\delta}\right), \frac{16\,VCdim(Z)}{\gamma^2\varepsilon}\ln\left(\frac{16}{\gamma^2\varepsilon}\right)\right\} \tag{13}$$

are generated according to a fixed unknown probability P on S. Then with probability at least $1 - \delta$ every function in Z that disagrees with at most a fraction $(1 - \gamma)\varepsilon$ of the

[3] Some measurability conditions are assumed, see [22].

p training examples will provide, with probability at least $1 - \varepsilon$, *the correct output for any input vector drawn according to the same distribution P.*

Note that by setting the slack parameter $\gamma = 1$ one considers only functions in Z that agree with all the training examples. In this particular case, Theorem 5.2 holds even if

$$p \geqslant \max\left\{\frac{4}{\varepsilon}\log\left(\frac{2}{\delta}\right), \frac{8\ VCdim(Z)}{\varepsilon}\log\left(\frac{13}{\varepsilon}\right)\right\}. \tag{14}$$

Thus the class of Boolean functions $H^* = LTBF(m)$ is PAC learnable. It suffices to draw p examples of the given target function $h^* \in H^*$ (with p equal the lower bound (14)) and to find an LTB function h that performs this task using a polynomial-time algorithm for linear programming. It is worth noting that if $\delta = 0.1$ and if we want to have confidence that h will provide the correct output for 90% of future examples (i.e. $\varepsilon = 0.1$), we should use a task T with at most $p = 562(m + 1)$ associations out of the 2^m possible ones. This is especially interesting for large sizes m of the input. If $m = 100$ one needs to store, for instance, at most 57 324 associations instead of 2^{100}. Clearly, the class of all linear threshold real functions of m variables ($LTRF(m)$) is also PAC learnable.

To apply Theorem 5.2 to the problem of learning from examples in feedforward ANNs [2], we need an upper bound on the VC dimension of multi-layer architectures. Baum and Haussler have shown in [18] the following result.

Proposition 5.3. *If H is the set of all Boolean functions which can be implemented by a feedforward architecture* $\mathcal{A} = (G, LTBF)$ *with the underlying graph* $G = (X, U)$ *where* $|X| \geqslant 2$ *and with* $W = |U| + |X|$ *parameters (i.e. total number of weights and thresholds), then*

$$VCdim(H) \leqslant 2W\log(e|X|),$$

where e *is the base of the natural logarithm.*

This large upper bound on the VC dimension of a given feedforward architecture can be used in (14). It guarantees that any feedforward network which memorizes correctly a set of

$$p \geqslant \frac{16W}{\varepsilon}\log(e|X|)\log\left(\frac{13}{\varepsilon}\right)$$

associations can be expected to produce the desired output for future inputs with probability at least $1 - \delta$. Note that this lower bound on p grows linearly with the size of the network given by W (see [18] for a tighter one). If one requires that at least a fraction $1 - \frac{\varepsilon}{2}$, $0 < \varepsilon \leqslant \frac{1}{2}$, of the p associations in the task are realized (i.e. $\gamma = 0.5$)

and $0 \leqslant \varepsilon \leqslant \frac{1}{2}$, it suffices to consider tasks of size

$$p \geqslant \frac{32W}{\varepsilon} \ln\left(\frac{32|X|}{\varepsilon}\right)$$

to have confidence at least $1 - 8e^{-1.5W}$ that the network will correctly classify all but a fraction ε of future inputs. Recall that this does not depend on the probability distribution used to select the task T and the testing examples.

Although we have considered only feedforward networks with a single output unit, the PAC learning theory has been recently extended to multiple outputs architectures [70].

Valiant's distribution-independent and complexity-based model of learning from examples provides a rigorous framework to tackle the problem of generalization in feedforward ANNs. However, one should be aware that any result, which makes no assumption on the probability distribution P defined on the input space, takes into account only the worst-case distribution. The distribution-independent condition is too restrictive from several points of view [17]. In particular, Theorem 5.2 and the related results established in [22] imply that one cannot expect that a class of Boolean-valued functions on $\mathbb{B}^m$ or $\mathbb{R}^m$ with exponential VC dimension in m is PAC learnable. This accounts for the fact that very few nontrivial classes are known to be PAC learnable. Moreover, some classes with infinite VC dimension can be learned in polynomial time if one restricts the set of possible distributions [17]. For example, the class of all Boolean-valued functions on $\mathbb{R}^m$ such that $h^{-1}(1)$ is convex can be learned polynomially for a uniform distribution P.

Important research areas would consist in refining these results by making some assumptions on the distribution P and in devising some modifications of the PAC learning model that lead to better bounds on the size of the task (see for instance [23]). One should also determine tighter bounds on the VC dimension of some specific architectures and investigate alternative measures of the capabilities of feedforward architectures that should not be based on a best-case analysis like the VC dimension.

6. Concluding remarks

Most of the problems discussed in this paper have a combinatorial flavor. Some of the learning problems were formulated as optimization problems. The high complexity encountered for the simplest networks compels us to identify collections of solvable cases as the partial k-tree situation. Knowledge of such cases may be a first step towards the development of efficient heuristic procedures for more general situations.

From a practical point of view, it would be very interesting to investigate the average-case complexity of the learning problem when instances from specific classes are drawn from some reasonable probability distribution. To our knowledge, no such results have been derived so far.

We saw that the complexity of the learning problem in feedforward architectures can be circumvented by considering more powerful learning algorithms which are allowed to change the topology of the network. However, efficient procedures constructing as small as possible networks are needed to ensure reliable generalization; those proposed to date are quite simple and not sufficiently effective to be applied to real-world problems requiring multiple outputs.

The model proposed by Valiant provides some insights into the information issue of learning from examples. It leads to bounds on the number of associations that must be learnt correctly in order to guarantee valid generalization. Nevertheless, the complexity issue is still open. It is not clear when, given enough examples, learning can be achieved in polynomial time in the size of the problem.

Instead of learning solely from examples, it may be useful to incorporate some available additional information concerning the task (hints) into the training procedure in order to try to reduce the complexity of the learning problem or at least to speed-up the training process. Hints may either be known precisely as, for instance, the signs of the weights [3] or be inferred from the task as it has been shown recently in [6].

Motivated by a huge variety of applications in fields like image recognition, pattern classification, error correction, etc., mathematicians will undoubtedly find in ANNs an unlimited source of interesting combinatorial and discrete optimization problems. Most important is the extension of joint work between mathematicians, algorithmicians and ANN scientists.

References

[1] E. Aarts and J. Korst, Simulated Annealing and Boltzmann Machines (Wiley, New York, 1989).
[2] Y.S. Abu-Mostafa, The Vapnik–Chervonenkis dimension: Information versus complexity in learning, Neural Comput. 1(3) (1989) 312–317.
[3] Y.S. Abu-Mostafa, Learning from hints in neural networks, J. Complexity 6 (1990) 192–198.
[4] Y.S. Abu-Mostafa and J.-M. St. Jacques, Information capacity of the Hopfield model, IEEE Trans. Inform. Theory IT-31 (1985) 461–464.
[5] A. Albert, Regression and the Moore–Penrose Pseudoinverse (Academic Press, New York, 1972).
[6] K.A. Al-Mashouq and I.S. Reed, Including hints in training neural nets, Neural Comput. 3 (1991) 418–427.
[7] D.J. Amit, Modeling Brain Function: The World of Attractor Neural Networks (Cambridge University Press, Cambridge, 1989).
[8] B. Angeniol, G. de la Croix Vaubois and J.-Y. le Texier, Self-organizing feature maps and the travelling salesman problem, Neural Networks 1 (1988) 289–293.
[9] E. Amaldi, Problèmes d'Apprentissage dans les Réseaux de Neurones, Diploma Project, Swiss Federal Institute of Technology, Lausanne (1988).
[10] E. Amaldi and S. Nicolis, Stability-capacity diagram of a neural network with Ising bonds, J. Physique 50 (1989) 2333–2345.
[11] E. Amaldi, On the Complexity of Training Perceptrons, in: T. Kohonen et al. eds., Artificial Neural Networks, Proc. ICANN-91 (North-Holland, Amsterdam, 1991) 55–60.
[12] E. Amaldi, E. Mayoraz, A. Hertz and D. de Werra, Apprentissage dans les Réseaux de Hopfield, Proc. Internat. Conf. on Artificial Neural Networks, EPF-Lausanne, Switzerland (Presses Polytechniques Romandes, 1989) 77–85.

[13] S. Arnborg, D.G. Corneil and A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (1987) 277–284.
[14] S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, Discrete Appl. Math. 23 (1989) 11–24.
[15] E.B. Baum, Complete representations for learning from examples, in: Y.S. Abu-Mostafa, ed., Complexity in Information Theory (Springer, Berlin, 1988).
[16] E.B. Baum, A proposal for more powerful learning algorithms, Neural Comput. 1(2) (1989) 201–207.
[17] E.B. Baum, The perceptron algorithm is fast for nonmalicious distributions, Neural Comput. 2(22) (1990) 248–260.
[18] E.B. Baum and D. Haussler, What size net gives valid generalization? Neural Comput. 1(1) (1989) 151–160.
[19] C. Berge, Graphs and Hypergraphs (North-Holland, Amsterdam, 1973).
[20] A. Blum and R. Rivest, Training a 3-node neural network is NP-complete, in: D. Haussler and L. Pitt, eds., Proc. 1st Workshop on Computational Learning Theory (Morgan Kaufmann, San Mateo, CA, 1988) 9–18.
[21] A. Blum and R. Rivest, Training a 3-node neural network is NP-complete, to appear in Neural Networks.
[22] A. Blumer, A. Ehrenfeucht, D. Haussler and M.K. Warmuth, Learnability and the Vapnik–Chervonenkis dimension, J. ACM 36 (1989) 929–965.
[23] A. Blumer and N. Littlestone, Learning faster than promised by the Vapnik–Chervonenkis dimension, Discrete Appl. Math. 24 (1989) 47–53.
[24] J. Bruck, Harmonic analysis of polynomial threshold functions, SIAM J. Discrete Math. 3 (1990) 168–177.
[25] Y. Censor, Row-action methods for huge and sparse systems and their applications, SIAM Rev. 14 (1965) 444–466.
[26] T.M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, IEEE Trans. Electronic Comput. 14 (1965) 326–334.
[27] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Systems 2 (1989) 303–314.
[28] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis (Wiley, New York, 1973).
[29] R. Durbin and D.J. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method, Nature 326 (1987) 689–691.
[30] C.-N. Fiechter, A parallel tabu search algorithm for large traveling salesman problems, Research Report ORWP90/01, Department of Mathematics, Swiss Federal Institute of Technology, Lausanne (1990).
[31] J.F. Fontanari and R. Meir, Evolving a learning algorithm for the binary perceptron, Network: Comput. Neural Systems 2 (1991) 353–359.
[32] M. Frean, The upstart algorithm: A method for constructing and training feedforward neural networks, Neural Comput. 2 (1990) 198–209.
[33] S.I. Gallant, A connectionist learning algorithm with provable generalization and scaling bounds, Neural Networks 3 (1990) 191 201.
[34] S.I. Gallant, Perceptron-based learning algorithms, IEEE Trans. Neural Networks 1 (1990) 179–191.
[35] M.R. Garey and D.S. Johson, Computers and Intractability: A Guide to the Theory of NP-Completeness (W.H. Freeman and Company, San Francisco, 1979).
[36] F. Glover, Tabu search – Part I, ORSA J. Comput. 1(3) (1989) 190–206; Tabu search – Part II, ORSA J. Comput. 2(1) (1990) 4–32.
[37] E. Goles and S. Martínez, Neural and Automata Networks: Dynamical Behavior and Applications (Kluwer Academic Publishers, Dordrecht, 1990).
[38] T. Grossman, The CHIR algorithm: A generalization for multiple-output and multi-layered networks, Complex Systems 3 (1989) 407–419.
[39] T. Grossman, R. Meir and E. Domany, Learning by choice of internal representations, Complex Systems 2 (1988) 555.
[40] J. Hertz, A. Krogh and G. Palmer, Introduction to the Theory of Neural Computation (Addison-Wesley, Redwood City, CA, 1991).

[41] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. USA 79 (1982) 2554–2558.
[42] J.J. Hopfield and D.W. Tank, Neural computation of decisions in optimization problems, Biol. Cybernetics 52 (1985) 141–152.
[43] J.S. Judd, Complexity of connectionist learning with various node functions, Tech. Rept. 87-60, Univ. of Massachusetts, Amherst, MA (1987).
[44] J.S. Judd, On the complexity of loading shallow neural networks, J. Complexity 4 (1988) 177–192.
[45] J.S. Judd, Neural Network Design and the Complexity of Learning (MIT Press, Cambridge, MA, 1990).
[46] B. Kamgar-Parsi and B. Kamgar-Parsi, On problem solving with Hopfield neural networks, Biol. Cybernetics 62 (1990) 415–423.
[47] N. Karmarkar, A new polynomial time algorithm for linear programming, Combinatorica 4 (1984) 373–395.
[48] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671.
[49] T. Kohonen, Self-Organization and Associative Memory (Springer, New York, 2nd ed., 1988).
[50] W. Krauth and M. Mézard, Learning algorithms with optimal stability in neural networks, J. Phys. A: Math. Gen. 20 (1987) 247–259.
[51] W. Krauth and M. Mézard, Storage capacity of memory, networks with binary couplings, J. Phys. France 50 (1989) 3057–3066.
[52] W. Krauth and M. Opper, Critical storage capacity of the $J = \pm 1$ neural network, J. Phys. A: Math. Gen. 22 (1989) 519.
[53] Y. LeCum, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, Back-propagation applied to handwritten zip code recognition, Neural Comput. 1 (1989) 541–551.
[54] R.P. Lippmann, An introduction to computing with neural nets, IEE ASSP Magazine (April 1987).
[55] M. Marchand, M. Golea and P. Ruján, A convergence theorem for sequential learning in two-layer perceptrons, Europhys. Lett. 11 (1990) 487–492.
[56] E. Mayoraz, Benchmark of some learning algorithms for single layer and Hopfield networks, Complex Systems 4 (1990) 477–490.
[57] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[58] M. Mézard and J.-P. Nadal, Learning in feedforward layered networks: the tiling algorithm, J. Phys. A: Math. Gen. 22 (1989) 2191–2203.
[59] M.L. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry. Expanded edition (MIT Press, Cambridge MA, 1988).
[60] S. Muroga, Threshold Logic and Its Applications (Wiley, New York, 1971).
[61] J.-P. Nadal, Study of a growth algorithm for a feedforward network. Int. J. Neural Systems 1(1) (1989) 55–59.
[62] P. Peretto, An introduction to the modeling of neural networks, in press.
[63] C.J. Pérez Vicente, J. Carrabina and E. Valderrama, Learning algorithm for feedforward neural networks with discrete synapses, in: A. Prieto, ed., Lecture Notes in Computer Science 540, Proc. IWANN'91, Grenade (Springer, Berlin, 1991) 144–152.
[64] C. Peterson and B. Södeberg, A new method for mapping optimization problems onto neural networks, Int. J. Neural Systems 1 (1989) 3–22.
[65] R. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms (Spartan Books, New York, 1962).
[66] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[67] D. Saad and E. Marom, Learning by choice of internal representations: An energy minimization approach Complex Systems 4 (1990) 107–118.
[68] N. Sauer, On the density of families of sets, J. Combin. Theory (A) 13 (1972) 145–147.
[69] T.J. Sejnowski and C.R. Rosenberg, Parallel networks that learn to pronounce english text, Complex Systems 1 (1987) 145–168.
[70] J. Shawe-Taylor and M. Anthony, Sample sizes for multiple-output threshold networks, Network 2 (1991) 107–117.

[71] J. Telgen, On relaxation methods for systems of linear inequalities, Eur. J. Oper. Res. 9 (1982) 184–189.
[72] G. Tesauro, Y. He and S. Ahmad, Asymptotic convergence of back-propagation, Neural Comput. 1 (1989) 382–391.
[73] L.G. Valiant, A theory of the learnable, Comm. ACM 27 (1984) 1134–1142.
[74] V.N. Vapnik and A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities, Theory Probab. Appl. 16 (1971) 264–280.
[75] V.N. Vapnik, Estimation of Dependences Based on Empirical Data (Springer, New York, 1982).
[76] S.S. Venkatesh, Directed drift: A new linear threshold algorithm for learning binary weights on-line, Presented at the Workshop on Neural Networks for Computing, Snowbird, Utah (April 1989).
[77] S.S. Venkatesh and P. Baldi, Programmed interactions in higher-order neural networks: Maximum capacity, J. Complexity 7 (1991) 316–337.
[78] R.S. Wenocur and R.M. Dudley, Some special Vapnik–Chervonenkis classes, Discrete Math. 33 (1981) 313–318.
[79] P. Werbos, Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D. Thesis, Harvard University (1974).
[80] D. de Werra and A. Hertz, Tabu search techniques: A tutorial and an application to neural networks, OR Spektrum 11 (1989) 131–141.
[81] B. Widrow and M.E. Hoff, Adaptative switching circuits 1960 IRE WESCON Convention Record 4 (1960) 96–104.