Neural Networks for High-Storage Content-Addressable Memory: VLSI Circuit and Learning Algorithm

MICHEL VERLEYSEN, MEMBER, IEEE, BRUNO SIRLETTI, ANDRÉ M. VANDEMEULEBROECKE, AND PAUL G. A. JESPERS, FELLOW, IEEE

Abstract—**Neural networks used as content-addressable memories show unequalled retrieval and speed capabilities in problems such as vision and pattern recognition. We propose a new implementation of a VLSI fully interconnected neural network with only two binary memory points per synapse. The small area of single synaptic cells allows implementation of neural networks with hundreds of neurons. Classical learning algorithms like the Hebb's rule show a poor storage capacity, especially in VLSI neural networks where the range of the synapse weights is limited by the number of memory points contained in each connection; we propose a new algorithm for programming a Hopfield neural network as a high-storage content-addressable memory. The storage capacity obtained with this algorithm is very promising for pattern recognition applications.**

I. Introduction

ARTIFICIAL neural networks have been studied for several years. These studies have sought to achieve the fabrication of machines that share some of the abilities of the human brain. In 1943, MacCullogh and Pitts [1] introduced a new mathematical model of the brain's structure. Many researchers tried to use this model in order to build machines that could learn from experience.

The new concept has been studied from both a theoretical and an experimental point of view. One of the best known achievements was the perceptron [2]. This multistage learning machine showed some interesting features, but Minsky and Papert pointed out some drawbacks and limitations [3]. At the same time, new interest in the study of artificial intelligence trapped many researchers as well as available funds; hence, the study of artificial neural net models was almost forsaken. In our opinion, this was foreseeable. Although artificial intelligence seemed to offer very promising short-term applications, the lack of simulation tools and appropriate mathematical background in the field of neural networks appeared to be a major obstacle for further developments.

When, in 1982, Hopfield proposed his new simplified model of a biological neural network [4], computer algorithms and nonlinear systems mathematics were ready to find interesting applications. Hopfield's model of a neural network consists of a massive parallel array of simple processing elements (neurons) connected through a coupling network that allows each neuron to be connected to the others through a set of connections called synapses. The information contained in the neural network is stored in the actual connection weights, and thus, it is distributed within the whole system. Every single neuron computes a weighted sum of the values of the other neurons, including sign discriminations through the synapses [5]. Hence, Hopfield's model is fully asynchronous: the system is set first to its initial state by turning appropriate neurons on or off and it is then left free until it reaches a stable state [4].

Manuscript received September 23, 1988; revised January 12, 1989. M. Verleysen and B. Sirletti were supported by IRSIA. A. M. Vandemeulebroecke was supported by FNRS.

The authors are with the Laboratory of Microelectronics, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

IEEE Log Number 8927704.

II. Advantages of an Analog VLSI Implementation for Neural Networks

In this section, we will compare the strengths and weaknesses of several realizations of neural networks: digital as well as analog VLSI implementations are contrasted to point out some of the advantages of analog VLSI circuits for synthetic neural networks.

A. Continuous Synaptic Weights

First, we will suppose that each synaptic weight can take any value between two predefined bounds. Several solutions have been proposed to realize such networks: specialized hardware or software to interface with conventional computers is available (HNC, Neural Ware, Neuronics, etc. [6]). Even though this is very attractive for simulation and experimentation tasks, the sequential algorithms programmed on classical Von Neumann machines completely ruin the speed properties of neural networks.

Dedicated processors, both analog and digital, also exist. In analog solutions, synaptic weights can be stored on a floating-gate structure, but slow charge variation on these capacitors represents a major drawback. In digital solutions, many bits are necessary to keep the accuracy needed for synaptic coefficients; the area occupied by synapses

0018-9200/89/0600-0562$01.00

and neurons rapidly becomes too large to realize neural networks with a great number of neurons.

B. Discrete Synaptic Weights

We now examine the case of a neural network in which the synaptic weights can take only a restricted number of values. Although this restriction limits the performances of neural networks, it allows the realization of much larger networks, which compensates for other drawbacks. Furthermore, at the end of this paper, we will show that it is possible to find learning rules that are well adapted to this kind of circuit with almost the same properties as if the synaptic weights had been continuous. We will suppose that each synaptic weight can take only three different values: $+1$, 0, and -1.

One of the advantages of digital realizations of neural networks stems from well-established state-of-the-art digital VLSI design tools. A digital VLSI chip can be designed with available CAD tools; standard cells and libraries can be used, and digital chips are tolerant to parameter dispersion and noise. However, as mentioned above, a fully digital neural network must be synchronous in practice. Moreover, synapses and neurons occupy a large chip area due to the arithmetic processors. Finally, a large amount of memory is necessary to store the weights of synapses as well as neuron values with an additional area penalty.

The asynchronism of natural neural networks can be kept in analog neural networks. For the same functionality, basic cells (neurons and synapses) are much smaller than those of digital neural networks. In Table I, we show the approximate synapse density that has been achieved for some recent VLSI realizations (the synapse density is defined as the number of synapses per square millimeter).

Although it is very difficult to compare realizations for connection processes that can differ substantially, one may conclude from this nonexhaustive table that analog cells are indeed smaller than their digital counterparts. Moreover, one of the unique features of neural nets—they are fault-tolerant—is lost to some extent because digital implementations consume a lot of area for an accuracy which is not always needed.

It seems obvious that the smaller area of analog realizations is an important advantage for the implementation of large neural networks. To realize large digital networks, one solution, however, could be to replace spatial complexity by temporal complexity (clocks, phases, multiplexors, etc.); but the introduction of synchronism and multiplexing in a neural network has a severe drawback on the computation speed. In the next section, we will examine analog asynchronous networks, which are more appropriate and much faster than conventional computers for solving, for instance, optimization and vision problems. Since only three different synaptic weights are required for each synapse, digital memory points will be preferred to floating-gate structures; the problems encountered in analog memories are consequently suppressed, and the area loss is not very significant. Another interesting approach could be the use of digital EEPROM's, but because such circuits need dedicated technological processes, they will not be studied in this paper.

TABLE I
SYNAPSE DENSITY

	synapse density (μm^2)	technology
Murray [7]	12.5	digital (pulse-stream) (2μ metal2)
Blayo [8]	5 9	digital (2μ metal2)
Weinfeld [9]	0.67	digital (2μ metal2)
Graf [10]	67	analog (2μ metal2)
Verleysen [11]	33	analog (3μ metal1)

III. HOPFIELD'S NETWORK

A. Description of the Network

Neural networks can solve a wide variety of problems; they can be used in vision (i.e., pattern recognition), in optimization (i.e., traveling salesman problem), in CAD (placing, routing), and in any other problem where perception is more important than a huge amount of accurate computation [12]. For pattern recognition and associative memories, an excellent trade-off between silicon area and computation efficiency is offered by Hopfield's model.

Hopfield's network consists of an array of fully interconnected synapses and neurons connected to each other through programmable connections (Fig. 1). The output of each neuron is fed back into the network; each synapse computes a new output value according to the output of the control neuron and the connection weights stored in the synapses. The outputs of all the synapses connected to the same neuron input are then summed; this sum determines the neuron activity and fixes the neuron output through its nonlinear transfer function.

We define:

- V_i value of neuron i, i.e., its state after the nonlinear activation function of the neuron has taken place,
- x_i neuron activity, i.e., the input of the neuron prior to the activation function,
- T_{ij} synaptic weight, i.e., the strength of the connection between neurons i and j,
- N number of neurons in the network, and
- I_i input of neuron i.

Further, let us assume

$$1 \le i \le N \quad \text{and} \quad 1 \le j \le N, \quad -1 \le T_{ij} \le 1$$
$$V_i = \pm 1.$$

The equations of the network are then

$$\partial x_i / \partial t = \sum T_{ij} V_j + I_i$$

with

$$V_i = F(x_i)$$

where F is the activation function.

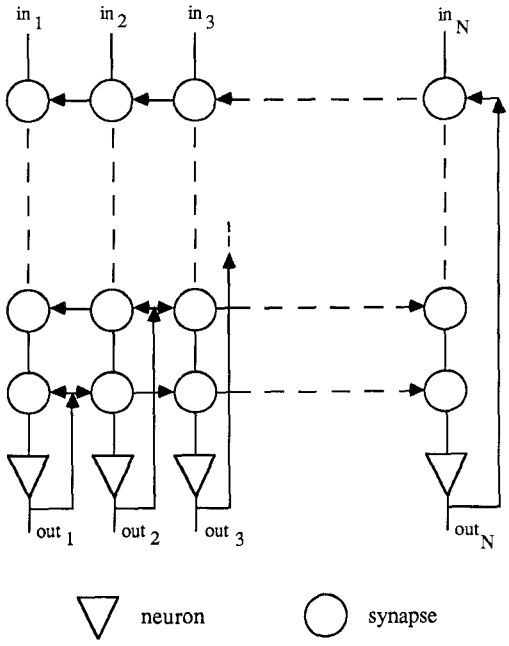


Fig. 1. Hopfield's network.

If the connection weights T_{ij} are adequately chosen by means of some appropriate "learning rule," the convergence process of the network can be predicted. This process can be described as follows: the values of the neurons are first set to an initial state; neuron activities are then computed via the connection network; and the new neuron values are determined by the activation function. This process is repeated until a stable state is reached.

B. Learning Rules for Hopfield's Network

Neural networks often are used in associative memories. With adequate connection weights, the final neuron values can be associated with initial input data. Of course, the capacity of the network, i.e., the number of patterns that can be memorized into the network as final neuron values after convergence has taken place, is limited and depends on the number of neurons in the network and on the learning rule used. For example, consider the well-known Hebb's learning rule. It can be formulated by

$$T_{ij} = \sum V_{ik} V_{jk}$$

where $1 \leq k \leq p$, p is the number of patterns to memorize, V_{ik} is the bit i of pattern k to memorize, and $V_{ik} = 1$ or $V_{ik} = -1$. This learning rule is very simple: each weight T_{ij} is increased if bits i and j are equal in pattern k; it is decreased in the opposite case. It has been shown that the capacity of the network is limited to about $0.15N$ arbitrary patterns [4].

Other learning rules have been proposed, including the projection rule [13], which seems more adapted to correlated patterns; the storage capacity is increased, but a higher accuracy is needed for the coefficients [14].

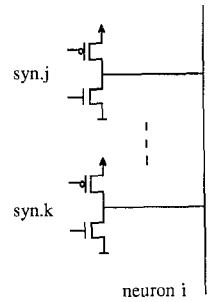


Fig. 2. P-and n-type current sources.

IV. The VLSI Analog Neural Network

A. Drawbacks of Conventional Implementations

Our goal was to realize an analog neural network with as much storage capacity as possible. Because the capacity increases with the number of neurons, the dimension of the network is the most important factor; since the main part of the circuit consists of synapses, we have tried to reduce their complexity and their area as much as possible. In digital implementations, the size of the circuit, i.e., the number of neurons in the network, has no influence on the synapse structure: the same design can be used in a ten-neuron network as well as in a 500-neuron network. We will see this is not true in analog implementations.

The basic idea underlying the proposed circuit was that each synapse should be able to source or sink current to the input line of the neuron to which it is connected, depending on the value $T_{ij}V_j$, i.e., the product of the synapse strength times the output of the neuron to which the synapse is connected. One realization, represented in Fig. 2, was proposed by Graf [10].

The drawback of this architecture is that one can never assume the excitatory and inhibitory currents to be exactly the same; even with adjustments of the size of the p- and n-type transistors, there is always a risk of mismatching between the sourced and sunk currents because of the technological mobility differences between the two types of transistors. Since the function of each neuron is to detect the sign of the weighted sum of the other neurons values, the mismatch between sourced and sunk currents is increased with larger numbers of synapses. In order to make a neuron able to discriminate the sign of its input even when the difference between excitatory and inhibitory currents equals only a single synaptic current, the latter must exceed n times the difference between the p- and n-type current sources; this considerably limits the size of the network that can be implemented with the principle shown in Fig. 2.

B. New Circuit Design

The problem can be avoided by using the same type of transistors to sink and source current on the input line of the neuron. To realize this, we used two distinct lines to

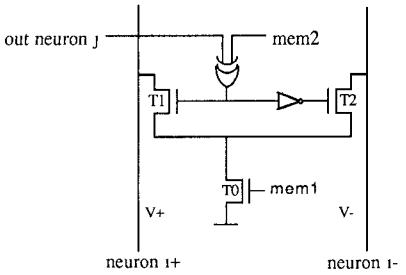


Fig. 3. Synapse.

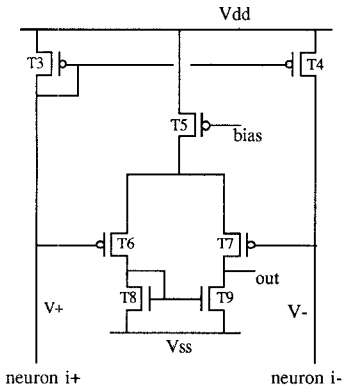


Fig. 4. Neuron.

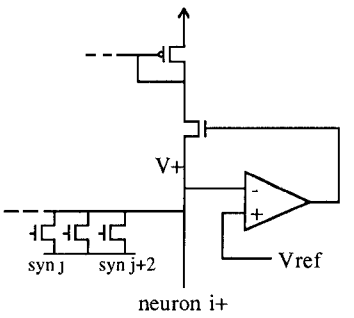


Fig. 5. Feedback loop.

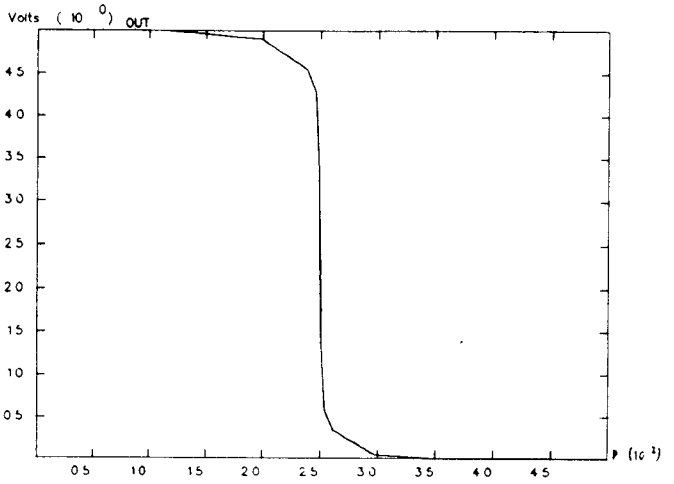


Fig. 6. Neuron output.

sum the currents: one for excitatory currents and the other for inhibitory currents.

Each synapse is a programmable current source controlling a differential pair (see Fig. 3). Three connection values are allowed in each synapse. If *mem*1 = 1, current is delivered to one of the two lines with the sign of the connection determined by the product of *mem*2 and the output of the neuron to which the synapse is connected. If *mem*1 = 0, no connection exists between neurons i and j, and no current flows to either excitatory or to inhibitory lines.

Depending on the state of the XOR function, the current may be sourced either on the line $i+$ or on the line $i-$. In the neuron, the comparison of the two total currents on the lines $i+$ and $i-$ must be achieved. This is done by means of the current reflector shown in Fig. 4. The currents on the lines are converted into voltages across transistors $T3$ and $T4$; these voltages themselves are compared in the differential input reflector formed by transistors $T5$–$T9$. Because of the two-stage architecture of the neuron, the gain may be very large, and the output (*out*) is either 5 V if the current in *neuron* $i-$ is greater than the one in *neuron* $i+$, or 0 V in the opposite case.

With increasing numbers of synapses connected to the same neuron, voltage drops $V+$ and $V-$ across $T3$ and $T4$ tend to increase. Because $V+$ and $V-$ are in fact the drain voltages of transistors $T1$ and $T2$ (see Fig. 3), saturation of the synaptic transistors must be avoided. A feedback loop was introduced to keep the voltage V^* (see Fig. 5) fixed to V_{ref}. Because no high gain is needed for the feedback loop, the amplifier shown in Fig. 5 can be very simple (even a single transistor can be used).

C. Experimental Results

Fig. 6 shows the output characteristic of one neuron. For this simulation, 512 active synapses are connected to the neuron. The X-axis represents the number of synapses (p) connected to the positive input line of the neuron; the others (512-p) are connected to the negative input line. The diagram shows that the output of the neuron is always saturated when there is a sufficient difference between the two inputs; this voltage can thus be directly fed back to the synapse inputs through the connection network.

With a smaller difference between the two inputs, the neuron output voltage varies between 0 and 5 V. This situation is incompatible with the synapse structure that needs a digital input. A buffer providing a binary output has therefore been inserted between the neuron output and the synapse inputs.

In order to determine the buffer characteristics and the maximum number of synapses allowed in the network, we measured the output voltage of the neuron (before the buffer) with an increasing number of active connected synapses. Fig. 7(a) and (b) shows the experimental results without and with the feedback loop, respectively, described previously (for this experience, a single-transistor feedback loop was used). The X-axis represents the number N of synapses connected to the neuron, and the Y-axis represents the neuron output voltage. This analysis has been

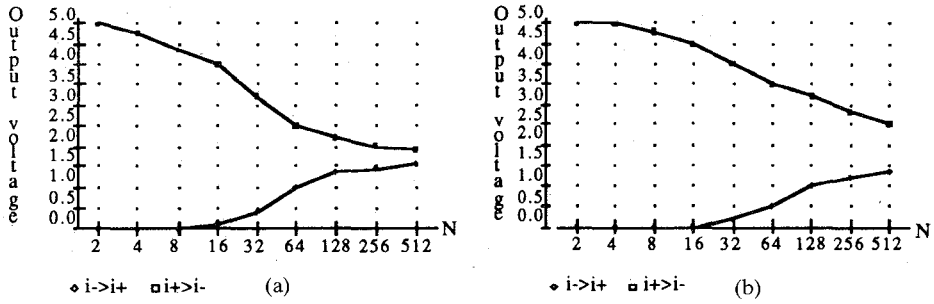


Fig. 7. Neuron output: (a) without feedback loop, and (b) with feedback loop.

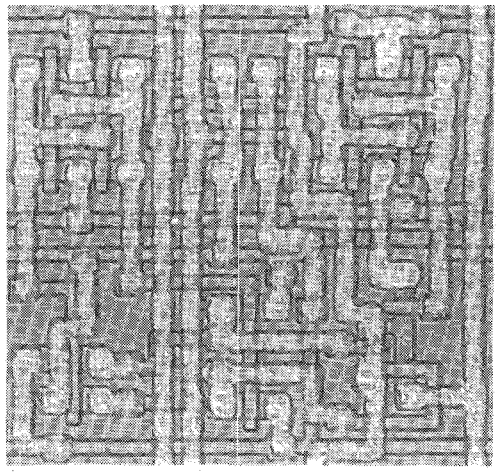


Fig. 8. Synapse.

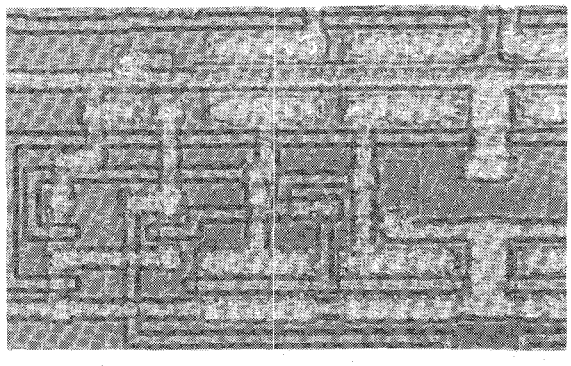


Fig. 9. Neuron.

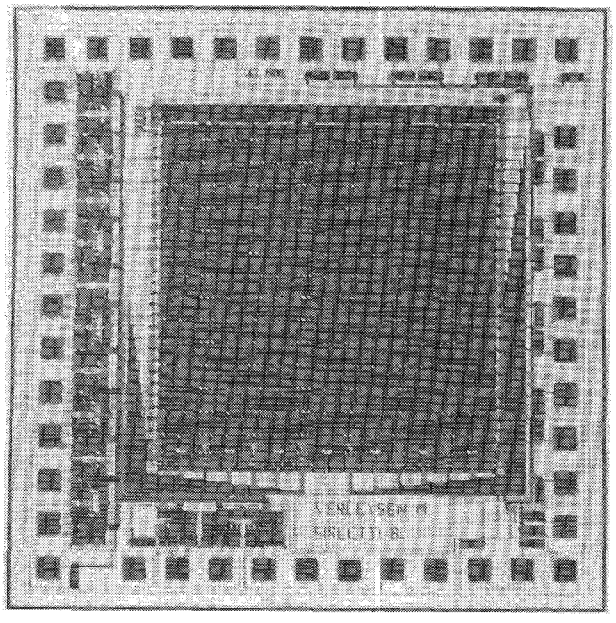


Fig. 10. Chip.

made under worst-case conditions (i.e., when $N/2-1$ synapses are excitatory and $N/2+1$ inhibitory ($i->i+$), or the opposite ($i+>i-$)). We see in Fig. 7 that the dynamic output range decreases with the increasing number of synapses. This is due to the common-mode voltage of the neuron amplifier: if the two input currents increase simultaneously, the amplifier gain will decrease, and the output will no longer be saturated. It seems obvious that the difference between the two curves must be large enough to correctly turn the buffer on or off (a voltage difference of 1 V is acceptable). With the feedback loop, a larger number of synapses (more than 500) is obviously possible.

In our circuit, the single synaptic current equals 10 μA; to achieve this, the synaptic transistor connected to *mem*1 is long ($W/L=0.1$), and the one connected to *mem*2 is minimal ($W/L=1.5$); such a current is acceptable in a neural network with a restricted number of neurons, as illustrated below. In larger networks, it has to be reduced for power density reasons. For example, the power dissipated by a 128-neuron network with synaptic currents equal to 1 μA will be about 100 mW; such a circuit can be made in a 64-mm^2 chip with a CMOS 2-μm double-metal technology (power density is about 1 mW/mm^2).

Speed properties are one of the most interesting features of neural networks. Experience showed that a change in a synapse value introduces a change in the correspondent neuron value and is fed back in the synapse in about 30 ns. Practically, it means that it takes about 120–150 ns for a 128-neuron network to converge to a stable state.

In order to verify the performances of synaptic currents and neuron response time, a small test chip was realized, which contained 14 neurons and 196 synapses. In order to make the circuit fully programmable and to exploit its learning capability, two memory points were embedded in each synapse. Figs. 8, 9, and 10, respectively, show a micrograph of a synapse, a neuron, and the complete chip.

In this part of this paper, we will consider an algorithm that allows programming of the circuit as a content-addressable memory. Since only three different connection weights are allowed (0, 1, and -1), the algorithm is adapted to this restriction.

V. A Learning Algorithm for Content-Addressable Memories

Many learning rules have been reported in the literature [4], [13]. They all have respective advantages and drawbacks, but few seem really suited for VLSI circuits. Indeed, in such realizations, the number of possible connection values is limited by the number of memory points contained in each synapse: if one synapse contains two memory points, only three or four connection values will be permitted (for example, -1, 0, and 1). Synaptic coefficients computed by a classical learning algorithm like the Hebb's rule have a larger dynamic range (for example, if k patterns are stored in an N-neuron network using Hebb's rule, each connection can take as much as $2k+1$ different values). To adapt these algorithms to VLSI circuits with 2-bit connections, the coefficients are simply truncated. This causes an important decrease in the storage capacity of the network.

The object of the second part of this paper is to describe a new learning rule that allows a good storage capacity of patterns with only three different connection weights. To achieve this goal, the restriction regarding the number of connection weights was included in the algorithm, rather than truncating the values after computation. It is important to note that the number of different connection values (in this case, three) does not rely on the number of recorded patterns. (This is not true for the Hebb's rule, which requires $2k+1$ possible values per synapse.)

We propose a new way to compute the connection strengths using a linear algebra optimization method (known as the simplex method) in order to maximize the stability of the recorded patterns. Connections between neurons in Hopfield's model are therefore represented by a $(n \times n)$ matrix in which element T_{ij} is the value of the connection between neuron i and neuron j (our algorithm allows T_{ij} to be different from T_{ji}). If we make V_i the Boolean state of the ith neuron and Θ the threshold value, the dynamic behavior of the network can be described by

$$V_i(t+\delta_t) = \operatorname{sign}\left(\sum T_{ij}V_j(t) - \Theta\right).$$

The network reaches a stable state when

$$\forall i: V_i(t+\delta t) = V_i(t).$$

In order to program stable states into the network and appropriate connection strengths, the following procedure is used. Let us suppose we want to store k n-bit patterns in a n-neuron Hopfield network and consider a single column (number one for instance) to illustrate the algorithm.

Fig. 11. Hebb's rule.

Let us call 1) V_{ij} the value of the ith neuron from the jth pattern to memorize $(1 \le i \le n, 1 \le j \le k, V_{ij} = 1 \text{ or } V_{ij} = -1)$; 2) T_{r1} the value of the connection between neurons r and 1 $(1 \le r \le n)$; and 3) $S_{1k} = \Sigma T_{r1}V_{rk}$ the input of the first neuron when the network outputs correspond to the training pattern k.

Assume that each neuron acts as a Boolean threshold function whose output is 1 in the case of a positive input and -1 in the other cases (Θ thus is set to 0). The problem is to choose the set T_{r1} in order to maximize the difference between S_{1k} and the threshold of the neuron. If the sign of S_{1k} is forced to be the same as the one of V_{1k}, we obtain the highest possible stability for bit 1 of pattern k. To ensure the right sign to S_{1k}, the quantity to maximize is actually $Z_{1k} = S_{1k}V_{1k}$. This has to be done simultaneously for all values of k. The equation to solve by the simplex method is then

$$\text{maximize } M \text{ where } M = \min(Z_{1k}) \text{ for all values of } k.$$

To avoid unbounded solutions $(M \to \infty)$, T_{r1} is bounded by the inequalities:

$$-1 \le T_{r1} \le 1.$$

Practical algorithms to solve such simplex problems are described in the literature [15].

The linear algebra theory assumes that at least $n-k$ coefficients will take the maximum values -1 or $+1$. In addition, experience showed that statistically all the other coefficient values were near -1, 0, or $+1$. Hence, this learning rule is well suited for such VLSI implementation as the one considered in the first part. Simulations showed that the results obtained with synaptic weights restricted only to -1, 0, and $+1$ are practically the same as with continuous values.

Let us now compare our results with those of the Hebb's rule. Experiments with 12-bit patterns were carried out in order to compare the efficiency of both rules to discriminate several patterns. Three target patterns were taught to the network using both Hebb's rule and the new rule respectively. The network was then run with the 2^{12} possible different input patterns. Finally, the output of the

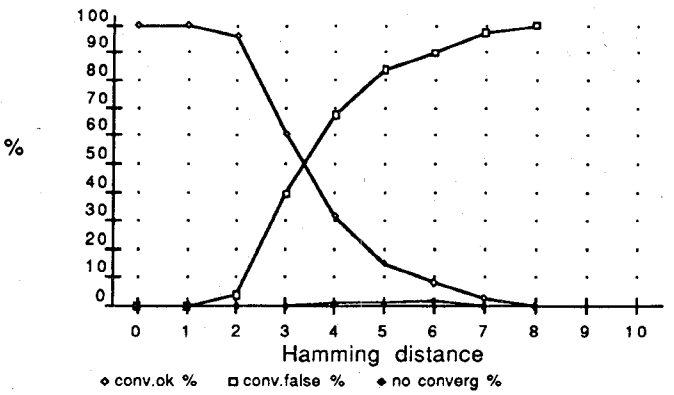


Fig. 12. New algorithm.

network was compared with the closest target pattern. We show, in Figs. 11 and 12, the percentage of well retrieved, nonretrieved, and unstable patterns, respectively, for the Hebb's rule and for the one presented here. In pattern recognition problems, observations are restricted to the left side of the diagram, i.e., where input patterns are close to recorded ones (the X-axis represents the Hamming distance or number of different bits between these two patterns). The conclusion was that with the new learning rule, more than 95 percent of the input patterns were correctly retrieved, and unstable patterns were almost nonexistent within the Hamming distance of 2. In the new learning rule, since the number of connection weights is small and independent from the number of memorized patterns, it is well adapted to VLSI circuits, as stated before in this paper.

VI. Conclusion

A neural array that realizes a content-addressable memory using Hopfield's network is described. Its analog VLSI implementation is also described. The synapse design is extremely simple for density reasons. Simulations have shown that circuits based on this architecture can contain up to hundreds of neurons. Although the circuit was initially designed for content-addressable memory applications, it can be used in a large number of other applications including optimization, pattern recognition and image processing.

A new learning rule also is described. The storage capacity of neural networks is increased when compared with respect to the classical Hebb's and projection rules. Furthermore, this rule is well adapted to VLSI circuits where the number of possible synaptic weights is restricted for compaction reasons.

The algorithm and the described circuit seem to be promising for the realization of high-storage content-addressable memories and applications using the speed and retrieval properties of neural networks.

References

[1] W. MacCullogh and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, pp. 115–133, 1943.

[2] R. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4–22, Apr. 1987.

[3] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA.: M.I.T. Press, 1969.

[4] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554–2558, Apr. 1982.

[5] J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad. Sci. USA*, vol. 81, pp. 3088–3092, may 1984.

[6] R. Hecht-Nielsen, "Neurocomputing: Picking the human brain," *IEEE Spectrum*, vol. 25, pp. 36–41, Mar. 1988.

[7] A. Murray, A. Smith, and Z. Butler, "VLSI bit-serial neural network," in *Proc. Int. Workshop VLSI for Artificial Intelligence* (Univ. of Oxford, U.K.), July 1988, pp. F4/1–F4/9.

[8] F. Blayo and P. Hurat, "A VLSI systolic array dedicated to Hopfield neural network," in *Proc. Int. Workshop VLSI Artificial Intelligence* (Univ. of Oxford, U.K.), July 1988, pp. E2/1–E2/10.

[9] M. Weinfeld, "A fully digital integrated CMOS Hopfield network including the learning algorithm," in *Proc. Int. Workshop VLSI Artificial Intelligence* (Univ. of Oxford, U.K.), July 1988, pp. E1/1–E1/10.

[10] H. Graf and P. de Vegvar, "A CMOS implemention of a neural network model," in *Proc. Stanford Conf. Adv. Res. in VLSI*. Cambridge, MA: M.I.T. Press, 1987.

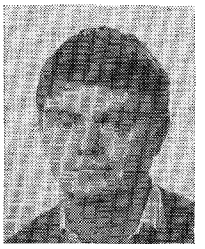
[11] M. Verleysen, B. Sirletti, and P. Jespers, "A new VLSI architecture for neural associative memories," in *Proc. nEuro* 88 (Paris, France), June 1988.

[12] E. Fahlman and G. Hinton, "Connectionist architectures for artificial intelligence," *IEEE Comput.*, pp. 100–109, Jan. 1988.

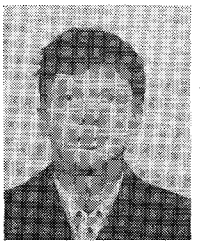
[13] L. Personnaz, I. Guyon, and G. Dreyfus, "Information storage and retrieval in spin-glass like neural networks," *J. Phys. Lett.*, no. 46, pp. 359–365, 1985.

[14] D. Gonze, E. Charlier, and E. Gilissen, "Réseaux de neurones: étude de mémoires associatives et conception d'un circuit CMOS programmable," Eng. thesis, Université Catholique de Louvain, Belgium, 1988.

[15] L. Lasdon, *Optimization Theory for Large Systems*. London: Collier-MacMillan, 1970.

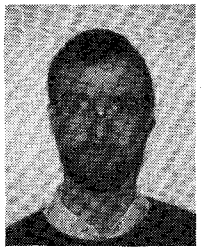


Michel Verleysen (M'86) was born in Bruxelles, Belgium, on October 27, 1965. He received the engineering degree from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 1987. In 1987, he was granted an IRSIA fellowship while working towards the Ph.D. degree in the field of neural networks at the Microelectronics Laboratory of the Université Catholique de Louvain. His research activities and interests are VLSI realization of neural networks, content-addressable memories, and analog integrated circuits and systems.



Bruno Sirletti obtained the engineering degree in electronical engineering at the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 1987.

He joined the Microelectronics Laboratory of the Université Catholique de Louvain where he started a new research activity on neural networks. Since October 1987 he has been working in the field of analog circuits and artificial neural networks with a fellowship from IRSIA.



André M. Vandemeulebroecke was born in Tournay, Belgium, on August 29, 1961. He received the engineering degree from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 1983. From 1983 to 1985, he was granted an IRSIA fellowship and worked in the field of silicon compilation for digital signal processors. In 1985, he was granted an FNRS fellowship while working towards the Ph.D. degree in the field of the theory and applications of redundant numbers systems at the Université Catholique de Louvain, Laboratoire de Microélectronique. His research activities and interest are centered around computational techniques for full-custom integrated circuits and applications to A/D conversion, cryptography, image processing, and neural networks.

Paul G. A. Jespers (M'60–SM'65–F'82) was born in Belgium on September 30, 1929. He received the engineering degree from the Université Libre de Bruxelles in 1953 and the Ph.D. degree from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 1958.

He first joined the Laboratoire Central d'Electricité, Brussels, working in RFI measurements until 1959. Since then he has been with the Department of Electrical Engineering, Université Catholique de Louvain, heading the Laboratoire de Microélectronique. He was a Visiting Professor at Stanford University, Stanford, CA, from September 1967 to January 1968. His current interest is in MOS integrated circuits and systems.

Dr. Jespers is Vice-Chairman of the Steering Committee of the European Solid-State Circuits Conference. He was appointed IEEE Regional Director of Region 8 from 1971 to 1972.