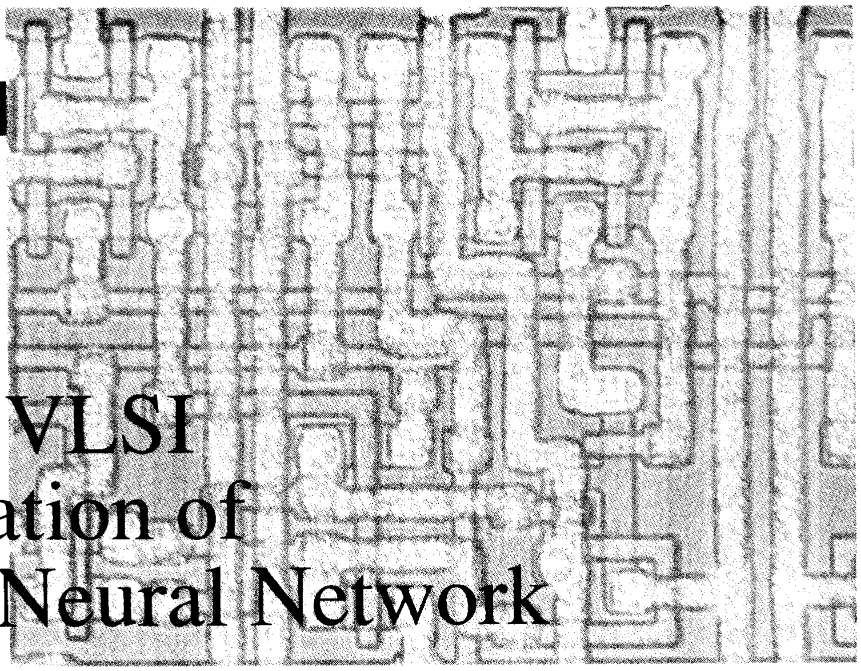
FEATURE



An Analog VLSI Implementation of Hopfield's Neural Network

This survey of the pros and cons of analog devices presents a 14-neuron test chip and an algorithm for fully interconnected networks.

Michel Verleysen
Paul G.A. Jespers

Catholic University of Louvain

Since the pioneering work of McCulloch and Pitts in 1943,[1] many researchers have tried to realize artificial devices that emulate the human brain. The challenge is to obtain a computational power unknown to conventional von Neumann computers. These computers do indeed achieve great speeds in the performance of repetitive tasks, but they cannot solve perceptual problems that are obvious to a five-year-old child. The need to discover new computer architectures increases in direct proportion to the number of tasks we want to perform with artificial devices.

Highly parallel artificial devices can emulate some of the characteristics of a human brain. Neural networks offer attractive solutions to most problems (such as image processing and optimization) in which perception is more important than intensive computation. We expect a number of important industrial applications for neural networks in the next few years.

Simulations performed on classical computers account for most of the actual research in artificial neural networks in recent years. However, the models have become too large and complicated to be handled by conventional machines. Further, simulations do not support two main characteristics of neuromorphic systems: speed and fault tolerance. Speed loses its importance in simulations because they are much slower intrinsically than electronic devices. Moreover, simulation of fault tolerance (that is, the ability to handle data properly when some elements of the network are damaged) poses a number of difficulties.

Since effective simulations of neural networks exceed the limits of conventional machines, researchers have been working on implementations that are more adapted to inherent neuronal properties. After a short discussion of fixed-value and digital networks, we point out the advantages of analog circuits for neural networks.

Electronic implementations

We can characterize electronic neural networks by their degree of connectivity. Hopfield's networks[2] interconnect fully. A feedback process connects the output of each neuron to the input of each other neuron. In Rumelhart's multilayer Perceptrons,[3] the output of each neuron connects to all neurons in the next layer—but no connection occurs within a layer. In locally interconnected networks, neurons connect only to their nearest neighbors. An example of this type of network is the silicon retina implemented by Mead.[4]

0272-1732/89/1200-0046$01.00 © 1989 IEEE

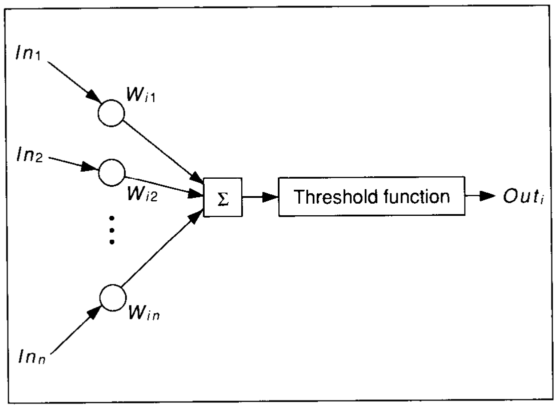


Figure 1. Structure of an artificial neuron. The neuron multiplies the synaptic weights W_{ij} by the input In_j, which is then summed.

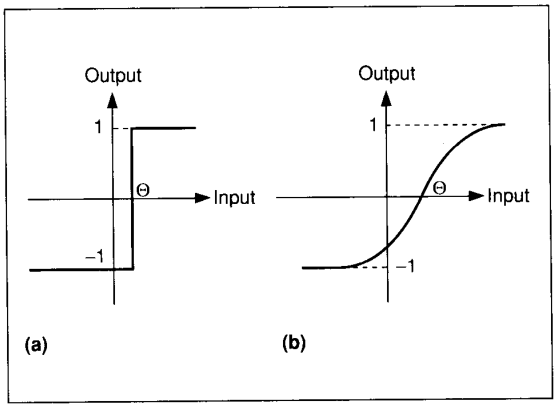


Figure 2. Nonlinear neuronal functions: threshold (a) and sigmoid (b).

In each of these three models, the functionality of neurons and synapses is roughly equal. A neuron performs a weighted sum of its inputs (see Figure 1). The result of this sum proceeds through a nonlinearity. Figure 2 shows examples of these threshold and sigmoid functions. The function of a synapse is to perform an operation—usually a simple multiplication—between the output value of the connected neuron and a weight contained in the synapse, as we explain later.

A learning rule generally associates with each type of neural network to compute the weights contained in each synapse. If this rule is off line, it computes weights separately before using the neural network in the convergence process. If the rule is on line, the weights adapt slightly each time a new pattern is presented to the network. The learning rule—as well as the kind of connectivity between the neurons—characterizes each type of neural network.

See the accompanying box for a discussion of a learning rule that we adapted to VLSI implementations.

The Learning Algorithm

Researchers have reported many learning rules in the literature.[2,13] These rules all possess respective advantages and drawbacks, but few seem really suited for VLSI circuits. Indeed, in such realizations, the number of memory points contained in each synapse limits the number of possible connection values. 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 Hebbian rule have a larger dynamic range. For example, if an arbitrary number of k patterns are stored in an n-neuron network using the Hebbian rule, each connection can take as much as $2k + 1$ different values. To adapt these algorithms to VLSI circuits with connections corresponding to a precision of 2 bits, we simply truncate the coefficients. This action causes an important decrease of the storage capacity of the associative memory.

Here we describe a new learning rule that allows a good storage capacity of patterns with three different connection weights only. We included the restriction regarding the number of connection weights in the algorithm, rather than truncating the values after computation. Note that the number of different connection values (here three) does not relate to the number of recorded patterns. This is not true for the Hebbian rule, which requires $2k + 1$ possible values per synapse.

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

$$V_i(t + \delta t) = \text{sign}\left[\Sigma\, T_{ij}V_j(t) - \Theta\right]$$

The network reaches a stable state when

$$\forall i\colon V_i(t + \delta t) = V_i(t)$$

In order to program stable states into the network and to compute the appropriate connection strengths, the following procedure stores k n-bit patterns in an n-neuron Hopfield network. The computation of a single column of the weight matrix (number 1, for instance) illustrates the algorithm. In this computa-

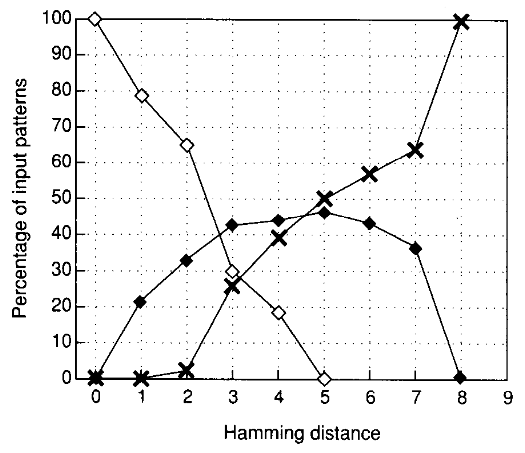


Figure A. Input patterns for the Hebbian rule.

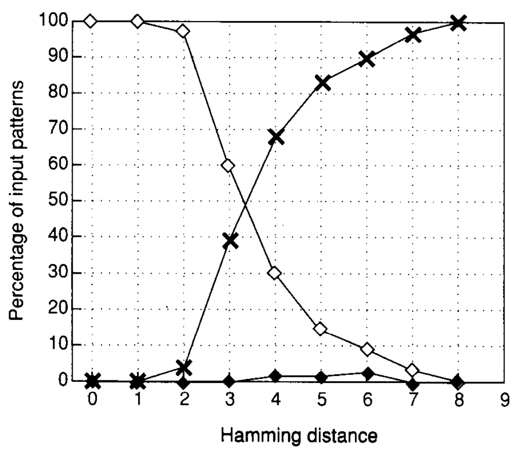


Figure B. Input patterns for the new algorithm.

tion, V_{ij} is the value of the ith bit from the jth pattern to memorize ($1 \leq i \leq n$, $1 \leq j \leq k$, $V_{ij} = 1$ or $V_{ij} = -1$). T_{r1} is the value of the connection between neurons r and 1 ($1 \leq r \leq n$). $S_{1k} = \sum T_{r1} V_{rk}$ is the input of the first neuron when the network output corresponds to the training pattern k.

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 arises in choosing the set T_{r1} 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 actually is $Z_{1k} = S_{1k} V_{1k}$. 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 ($M \rightarrow \infty$), T_{r1} is bounded by the inequalities $-1 \leq T_{r1} \leq 1$.

The literature describes practical algorithms to solve such simplex problems.[14]

The linear algebra theory assumes that at least $n - k$ coefficients will take the maximum values -1 or $+1$.[15] Experience has also shown that statistically all of the other coefficient values were near to -1, 0, or $+1$. Hence this learning rule works well for such VLSI implementations as we consider here. Simulations showed that the results obtained with synaptic weights restricted only to -1, 0, and $+1$ are practically the same as cases in which continuous values exist.

We compared the results of this rule with those of the Hebbian rule. We carried out experiments with 12-bit patterns to compare the efficiency of both rules to discriminate several patterns. We taught the network three target patterns using the Hebbian rule and the new rule. Then we ran the network with the 2^{12} possible different input patterns. Finally, we compared the output of the network with the closest target pattern. Figures A and B show the percentage of well-retrieved (good convergence), nonretrieved (false convergence), and unstable (no convergence) input patterns for the Hebbian rule and for the one presented here. In pattern-recognition problems, observations are restricted to the left side of the diagram, that is, where input patterns are close to the recorded ones. The x axis represents the Hamming distance or number of different bits between these two patterns. We concluded 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. The new learning rule, which contains a small number of connection weights that are independent from the number of memorized patterns, adapts well to VLSI circuits.

An electronic neural network consists of a set of elementary processors connected by simple cells (synapses) that realize the product between their inputs and an internal weight. The disposition of the basic cells (neurons and synapses) and the manner in which they are connected determine the type of neural network they occupy.

Here we discuss only Hopfield's fully interconnected networks. We have two reasons. Although Hopfield's networks generally need more neurons than other types of neural networks for the same functionality, this architecture is suitable for pattern recognition, which only requires a small number of neurons for practical applications. Secondly, fully interconnected networks have the most regular structure, which makes them particularly suited for VLSI implementations. However, most of the neural network architectures—and particularly the fully or locally interconnected networks and multilevel Perceptrons we mentioned—have one common important feature. The neurons and synapses must be as small as possible to allow the integration of a great number of them on a chip. Further, the design of the cells must allow many synapses to connect to the same neuron without electrical problems. Since the design goals are roughly equal for different types of neural networks, the solutions presented here not only pertain to Hopfield's networks but can easily adapt to other architectures. This flexibility is particularly true for the analog techniques that implement synapses and neurons.

The first—and simplest—idea for implementing neural networks uses resistors as synapses (shown in Figure 3).

A functional neuron realizes the function:

$$V_i = f\left(\sum_j T_{ij} V_j + I_i\right)$$

where V_i is the output of neuron i, T_{ij} is the weight of the synapse connecting neurons i and j, I_i is the input of neuron i, and f is the nonlinear transfer function of the neuron (as in a threshold or sigmoid function).

The value of each coupling resistor is given by

$$R_{ij} = 1/T_{ij}$$

In the synapses, the output voltage of each neuron thus converts to an input current for another neuron i:

$$I_{ij} = V_j / R_{ij} = T_{ij} V_j$$

Neurons sum all currents derived from the synapses connected to neuron i and the input current. The neuron must thus be a nonlinear conductance that converts its input current into an output voltage:

$$V_i = f\left(\sum_j V_j / R_{ij} + I_i\right)$$

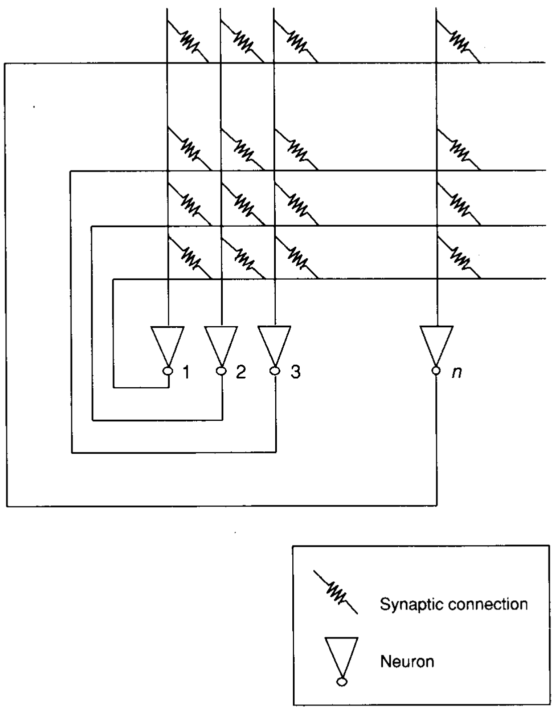


Figure 3. Resistor array.

The main problem with this kind of implementation is that resistors are not commonly used in standard complementary metal-oxide semiconductor (CMOS) technology. They usually occupy a large area on chip, which makes it impossible to implement networks with a huge number of interconnections. Further, programming the network requires variable resistors, which is very difficult to achieve.

Howard et al.[5] provide a possible solution to these problems. They implemented a resistor array by depositing amorphous silicon by electron-beam evaporation onto a patterned polyimide layer. The density of integration corresponds to about 6.10^6 resistors (300 kiloohms) per square centimeter, which makes it suited for large arrays of synapses. Other technologies can also serve to implement fixed-value resistors. However, such technologies do not allow programmability of synapses since the connection values have to be chosen during the fabrication process. This sort of fixed-weight realization must thus be restricted to optimization problems in which the weights must not be changed. This circuit is not suitable for a pattern-recognition system, in which the weights depend upon patterns to store data in the network.

If the network must be programmed (that is, if the

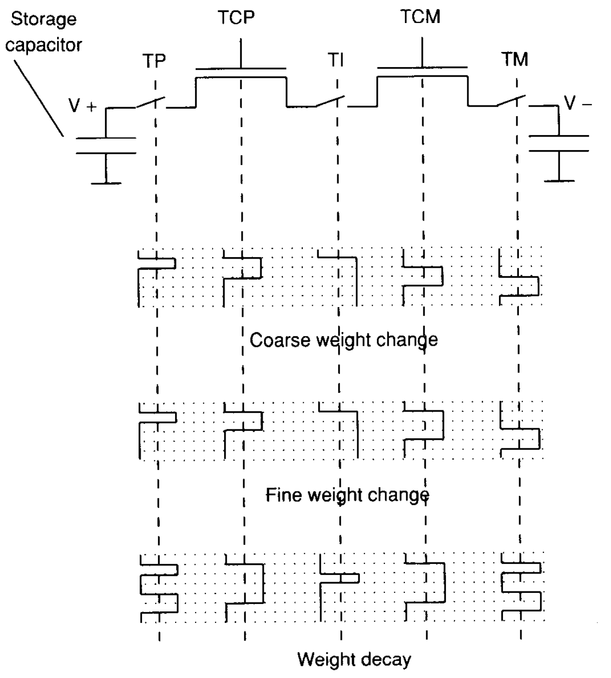


Figure 4. Capacitive memory points.

connection weights must be adjusted after the design), designers can use digital circuits to realize neurons and synapses. However, only a small number of digital cells—generally much larger than their analog counterparts—can fit on a chip. Digital computation, on the other hand, easily achieves precision. Moreover, when speed is not critical, temporal multiplexing can replace some of the spatial complexity. In this case, digital neural networks can be very efficient. However, when speed is important—and a great number of neurons and synapses must be used—we feel analog techniques are more appropriate.

Analog VLSI networks

As discussed, fully interconnected networks (or any type of networks with a large number of connections) are very difficult to realize with digital VLSI technology. Analog networks, although more suitable for neural networks, do pose some electrical problems that can occur with large systems. For example, one must design a neuron cell more carefully when it connects to 100 synapses than when it connects to only 10.

Memory points. The first challenge analog neural networks offer is the selection of memory points to store the synaptic weights. Before attempting to solve this problem, designers must know the precision required for the synaptic weights. A neural network that needs a precision corresponding to only 1 or 2 bits will be designed totally differently from a neural network with an 8- or 12-bit precision. In the first case we do not consider the size of a memory point to be important, but in the second case we reduce this size as much as possible.

Of course, digital memory points are the simplest to use. By digital here we mean that only logic values can be stored in such memories and that we need *n* identical cells to store one *n*-bit synaptic weight. Designers generally use static memory points, but dynamic ones also exist in some implementations. Designers do not often use dynamic memory points because neural networks are generally asynchronous. Computations—and memory-point access—can thus occur at any time. The design of a refreshment system would enhance the complexity of the system. However, since most of the actual chips used in neural networks consist of prototypes and test chips, their maximally simplified design generally avoids dynamic memory points with complex refreshment logic.

Designers can use analog memory points for networks that need more than a 1-bit precision. Analog memory points can occur in a capacitor that stores an analog synaptic weight. The value of this capacitor is then multiplied in the synapse by the output of the connected neuron. An ideal capacitor would store an exact weight with an infinite precision. In a VLSI chip, we must of course cope with technological limitations. Every capacitor has a leakage current, and the system must refresh its value periodically. The designer must thus try to accord this circuit to the following rule. During the various cycles necessary to the convergence of the neural network, the percentage of the charge loss of the capacitor must be small. Its values before and after the convergence should be exact at the precision needed for the network. We review several techniques to achieve this goal. They rely either on technological improvements to reduce the leakage currents of the capacitors or on a specific design to compensate for such currents.

Standard MOS devices can hardly achieve accurate capacitors with reduced leakages. Special technological processes like Flotox[6] (in which the floating-gate structures implement capacitors with a retention time of several days) can solve this problem. However, this solution contains the drawback that standard CMOS technology cannot support these supplementary technological steps without an important cost increase.

Another way to obtain high-precision capacitors is to adjust their design so that leakage currents have no effect on the behavior of the circuit. Since synaptic weights are generally positive or negative, using two capacitors can avoid such an effect.[7] The difference between their charges determines the value of the connection. The difference between the leakage currents then determines the charge loss, which is a much better solution than having only one capacitor. Added transistors and small variable capacitors provide weight adjustments (see Figure 4). This solution can combine with high-precision capacitors that have reduced leakage currents to obtain accurate synaptic weights. With this precision, the system proposed by Schwartz, Howard, and Hubbard[7] can obtain a resolution of 10 bits.

In Figure 4, long transistors TCP and TCM provide for charge transfer. TP, TM, and TI (narrow transistors) isolate the charge-transfer transistors from each other and from the storage nodes. These transistors provide ideal switching. Using different sequences in turning on the switches provides coarse or fine weight changes. For example, one achieves weight decay by first turning on all transistors except TI. When the two sides of the charge-transfer string have equilibrated with their respective storage nodes, the connections between the storage nodes (TM and TP) turn off, and the two charge-transfer transistors TCP and TCM exchange charges by turning on transistor TI. When TCM and TCP have equilibrated their charges, TI turns off again. The charges held by TCM and TCP inject back into the storage capacitors. The resulting change in the stored weight is $\Delta V = -(V_+ - V_-)$*Ceff/Cox, which corresponds to multiplying the weight by a factor $\alpha < 1$. Figure 4 illustrates coarse and fine weight changes and weight decay.

Although it is a seductive solution for analog synaptic weights, reducing the number of memory points for a required precision corresponding to several bits is not the best answer. Using only one digital memory point is optimum. Of course, this approach offers less precision and decreases global performance. However, one can use algorithms that cope with this restriction to program the network.[8] Such algorithms require only two or three different synaptic weights, and performance does not decrease the way it does in networks in which continuous synaptic weights are used.

Analog architectures. Once the type of memory points has been chosen, we can design the synapse. The synapse realizes the logical function between the value of the neuron that connects to it and an internal weight stored in the memory points. We first examine the case in which the output value of the neuron can take two different values and the synaptic weight can take three. The logical function that occurs between the neuron output value (+1 or −1) and the synaptic weight (+1, 0, or −1) is either AND or XOR (we expect both of these functions to become 0 when the synaptic weight is 0). However, simulations have illustrated that the XOR function is generally preferable in Hopfield's networks because it increases storage capacity.[9] The circuit we show implements a XOR function, but we can easily transpose it to an AND function.

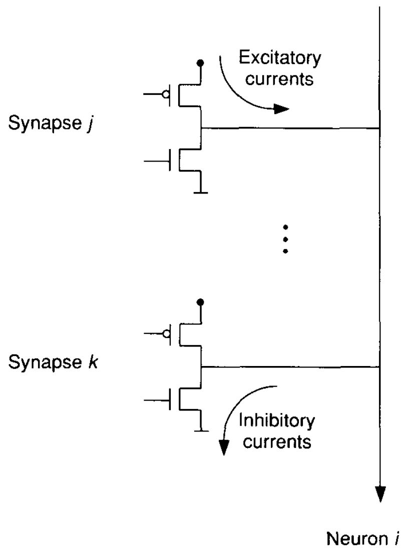


Figure 5. Synapse architecture.

Since three different synaptic weights are allowed, either one analog memory point or two digital memory points are necessary. We prefer the latter solution because of the reduced precision. The architecture that follows[10] is based on the principle of current summation. Synapses consist of programmable current sources, and the neuron sums all synaptic currents. The neuron is an amplifier that realizes the nonlinear transfer function (sigmoid, threshold, etc.). One solution consists of summing all synaptic currents on the input line of the neuron.[11] If the synapse is excitatory (a positive result of the XOR function), the current flows to this input line. If the synapse is inhibitory (a negative result), the current flows from the input line (shown in Figure 5).

If the neuron realizes a pure threshold function, with the threshold equal to 0, it must discriminate whether the total current on its input line is positive or negative. A simple amplifier can achieve this result. The major drawback of this architecture lies in the following. In standard CMOS technology, the currents that flow through the P- and N-channel types of transistors are very different. Adjusting the size of the transistors (usually a factor of 2.5 to 3 between the two types of transistors) achieves compensation. However, we do not know the exact ratio before the technological processes occur, and the currents flowing to or from the input line differ slightly. When the number of synapses connected to the same neuron increases, the system sums these differences. The total mismatching can exceed that of one synaptic current, and the neuron can

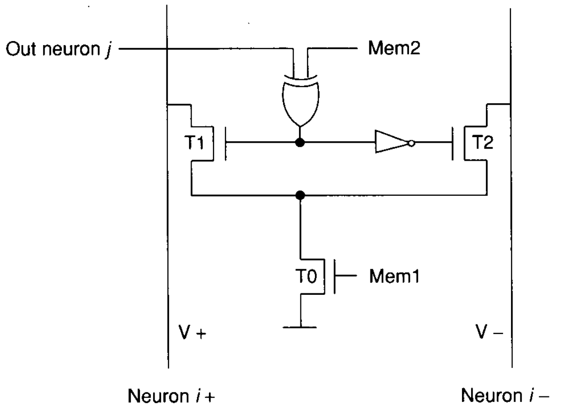


Figure 6. Artificial synapse.

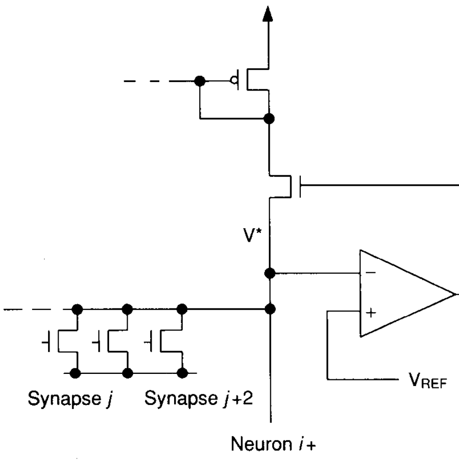


Figure 8. Feedback loop.

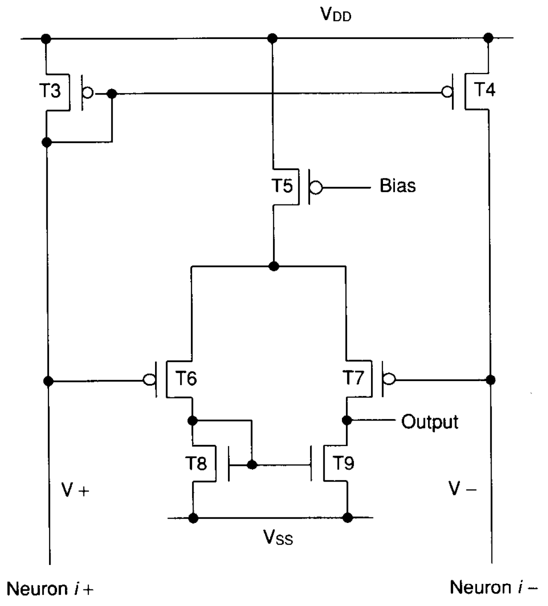


Figure 7. Current flow in an artificial neuron.

thus switch to a wrong state if the input current is close to the threshold. A solution to this problem[12] consists of summing all excitatory currents on one input line and all inhibitory currents on another. When the design employs the same type of transistors on each line of a two-line system, the mismatching problem disappears.

Each synapse is a programmable current source that controls a differential pair (see Figure 6). Three connection values occur in each synapse. If the memory point mem1 = 1, current flows to one of the two lines with the sign of the connection determined by the product of mem2 and the output of the neuron connected to the synapse. If mem1 = 0, no connection exists between neurons *i* and *j* nor does current flow to the excitatory and inhibitory lines.

Depending on the state of the XOR function, the current can flow either from the line *i* + or from the line *i* −. In the neuron, the comparison of the two total currents on the lines *i* + and *i* − must occur. The differential amplifier shown in Figure 7 performs this task. The current mirror formed by transistors T3 and T4 functions as a load for the two currents on lines *i* + and *i* −. The voltage drops across these two transistors are thus monotonic, increasing functions of the currents. The amplifier then compares these voltage drops in the differential input reflector formed by transistors T5 to T9. Because of the two-stage architecture of the neuron, the gain may be very large. A 5V output occurs if the current in neuron *i* − is greater than that of neuron *i*+. The output becomes 0V in the opposite case.

Because of the use of current mirrors, some asymmetries exist in this circuit. Impedances of T3 and T4 are different, and the same is true even for T8 and T9. Nevertheless, since we designed the circuit to produce saturated outputs, asymmetries inside the neuron are not important, provided that the output switches from one logic voltage to the other when the two input currents are equal. The design works well since the two transistors of each current mirror (T3/T4 and T8/T9) have exactly the same behavior when the currents they drive are equal.

With increasing numbers of synapses connected to the same neuron, voltage drops V + and V − across T3

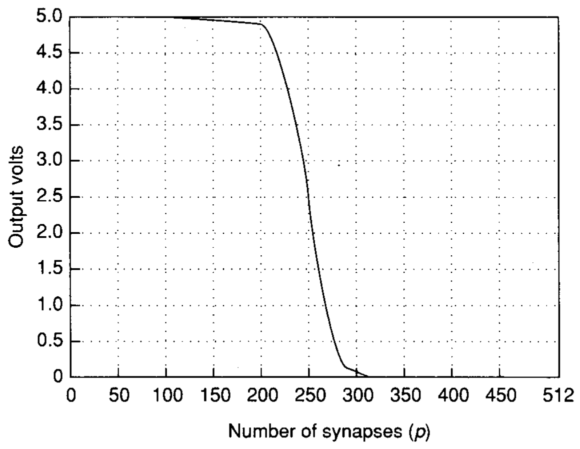


Figure 9. Output of an artifical neuron.

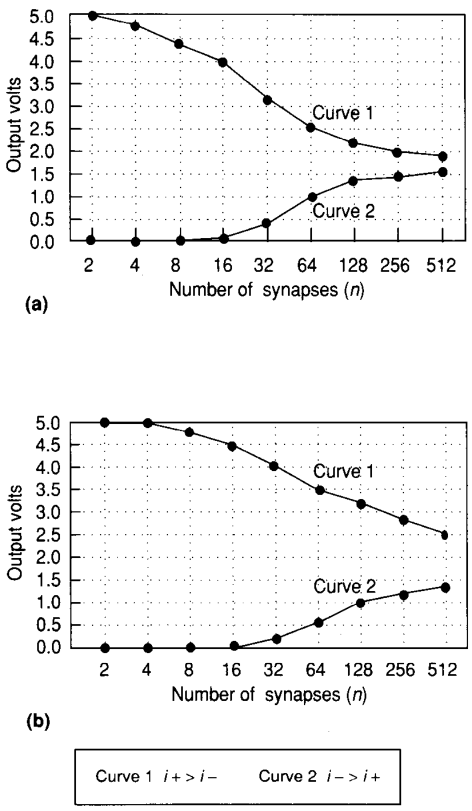


Figure 10. Neuron output without feedback loop (a) and with feedback loop (b). Curve 1 represents the output with a predominance of excitatory current; curve 2 is the opposite case.

and T4 tend to increase. Because V + and V − are in fact the drain voltages of transistors T1 and T2 (see Figure 6), we must avoid saturation of the synaptic transistors. We introduced a feedback loop to keep the Out voltage V* of the synapses (see Figure 8) fixed to V_{REF}. While no high gain is needed for the feedback loop, the amplifier in Figure 8 can consist of one transistor.

Experimental results. Figure 9 shows the output characteristic of one neuron. For this simulation, 512 active synapses connect to the neuron. The x axis represents the number of synapses (p) connected to the positive input line of the neuron. The other synapses (512-p) connect to the negative input line. The diagram shows that the neuron output always becomes saturated when a sufficient difference exists between the two inputs. This voltage thus can feed directly back to synapse inputs through the connection network.

With a smaller difference between the two inputs, the neuron output voltage varies between 0V and 5V. This situation is incompatible with the synapse structure, which needs a digital input. We therefore inserted a buffer to provide a binary output between the neuron output and the synapse inputs.

We measured the output voltage of the neuron (before the buffer) with an increasing number of active, connected synapses to determine the buffer characteristics and the maximum number of synapses allowed in the network. Figure 10 shows the experimental results with and without the feedback loop described previously. We used a single transistor feedback loop in the experiment. The x axis represents the N number of synapses connected to the neuron while the y axis represents the neuron output voltage. We performed this analysis under worst case conditions, that is, when $n/2 - 1$ synapses are excitatory and $n/2 + 1$ are inhibitory ($i - > i +$), or the opposite ($i + > i -$). Figure 10 shows that the dynamic output range decreases with an increasing number of synapses. This phenomenon results from the common-mode voltage of the neuron amplifier. If the two input currents increase simultaneously, the amplifier gain decreases and the output loses saturation. It seems obvious that the difference between the two curves must be large enough to correctly turn the buffer on or off (a difference of 1V is acceptable). The feedback loop can provide a larger number of synapses (more than 500).

In this circuit, the single synaptic current equals 10 microamps. To achieve this state requires that the synaptic transistor connected to mem1 be long (the width/length ratio of the transistor W/L = 0.1) and that the transistor connected to mem2 be minimal (W/L = 1.5). Such a current is acceptable in a neural network with a

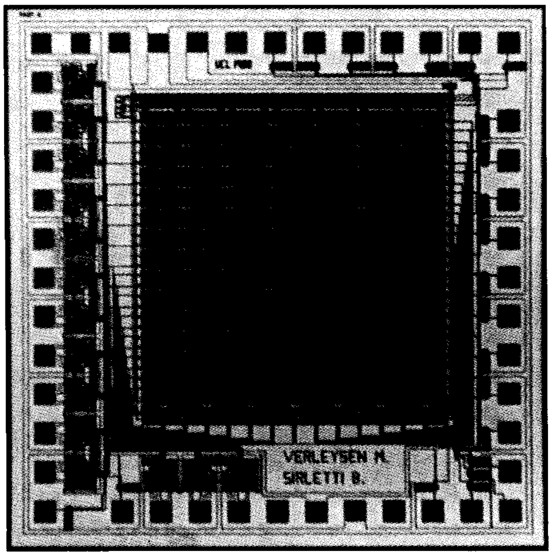


Figure 11. Microphotograph of the test chip containing 14 neurons and 196 synapses.

restricted number of neurons as illustrated later. In larger networks, one must reduce the current for power-density reasons. For example, the power dissipated by a 128-neuron network with synaptic currents equal to 1 milliamp approximates 100 mW. One can implement such a circuit in a 64-sq-mm chip with a CMOS 2-micrometer, double-metal technology (the power density is about 1 mW/sq mm).

Speed properties constitute one of the most interesting features of neural networks. Experience has shown that a change in a synapse value introduces a change in the correspondent neuron value, which is fed back into the synapse in about 40 nanoseconds. Practically, this means that it takes about 150-200 ns for a 128-neuron network to converge to a stable state. In general, about five iterations are necessary to retrieve the pattern stored in the network from the input pattern.

We implemented a small test chip to verify the performances of synaptic currents and neuron response time. This chip contains 14 neurons and 196 synapses. To make the circuit fully programmable and to exploit its learning capability, we embedded two memory points in each synapse. Figure 11 shows a microphotograph of the complete chip.

We expect the research in VLSI technology for neural networks to lead to the development of practical applications using small networks within the next few years. Such applications concern vision, speech and image processing, character recognition, or other tasks in which a small amount of data is handled simultaneously. We have pointed out the advantages and drawbacks of some types of VLSI analog neural networks and have shown how a larger number of neurons can connect through the use of analog summing of products. Researchers in this field will obviously try to integrate larger networks for use in more complex applications. One solution to the small size of the current artificial neural networks is to cascade chips. However, the number of cells in a fully interconnected neural network grows with the square of the number of neurons. The number of chips necessary for this type of implementation rapidly overwhelms the imagination. The solution to the integration of large networks is thus to reduce the size of the basic cells (the neurons and synapses), even despite a loss of precision.

Classical learning algorithms, like the Hebbian rule, become inefficient when the dynamics of the synaptic weights are restricted to some discrete values. One can avoid this problem by using learning rules similar to the one presented here. This algorithm helps program a fully interconnected network into a content-addressable memory and shows an increased storage capacity in comparison with that of other learning rules.

While a lot of work remains to be done towards the integration of large neural networks, the analog techniques we have presented here prove that networks with several hundreds of fully interconnected neurons are not unrealistic.

References

1. W.S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bull. Math. Biophysics*, Vol. 5, 1943, pp. 113-133.
2. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat'l Academy Sci.*, Vol. 79, Apr. 1982, pp. 2,554-2,558.
3. D. Rumelhart et al., *Parallel Distributed Processing; Vol 1: Foundations*, MIT Press, Cambridge, Mass., 1986.
4. M. Sivilotti, M. Mahowald, and C. Mead, "Real-Time Visual Computations Using Analog CMOS Processing Arrays," *Proc. 1987 Stanford Conf. Advanced Research VLSI*, P. Losleben, ed., MIT Press, 1987.
5. R. Howard et al., "An Associative Memory Based on an Electronic Neural Network Architecture," *IEEE Trans. on Electron Devices*, Vol. ED-34, No. 7, July 1987, pp. 1,553-1,556.
6. C. Bleiker, *Behavior and Characterization of EEPROM Cells with Floating-gate Structure* (in German), PhD thesis, Eidgenossischen Technical School, Zurich, 1987.
7. D. Schwartz, R. Howard, and W. Hubbard, "A Program-

mable Analog Neural Network Chip," *IEEE J. Solid-State Circuits,* Vol. 24, No. 2, Apr. 1989, pp. 313-319.

8. B. Sirletti et al., "An Algorithm for Pattern Recognition with VLSI Neural Networks," *Proc. IEEE First Int'l Neural Network Society,* IEEE Press, Piscataway, N.J., 1987.
9. J.J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Nat'l Academy Sci.,* Vol. 81, May 1984, pp. 3,088-3,092.
10. M. Verleysen, B. Sirletti, and P. Jespers, "A Large VLSI Fully Interconnected Neural Network," *Proc. 1988 Symp. VLSI Circuits,* IEEE Press, 1988, pp. 27-28.
11. H.P. Graf and P. de Vegvar, "A CMOS Associative Memory Chip Based on Neural Networks," *Proc. 1987 IEEE Int'l Conf. Solid-State Circuits* , IEEE Press, 1987, pp. 304-305.
12. M. Verleysen, B. Sirletti, and P. Jespers, "A New VLSI Architecture for Neural Associative Memories," *Neural Networks from Models to Applications,* L. Personnaz and G. Dreyfus (eds.), IDSET, Paris, pp. 692-700.
13. L. Personnaz, *Study of Formal Neural Networks: Conception, Properties, and Applications* (in French), PhD thesis, Pierre and Marie Curie University, Paris, 1986.
14. L. Lasdon, *Optimization Theory for Large Systems,* The Macmillan Publishing Co., Inc., London, 1970.
15. D. Pierre, *Optimization Theory with Applications,* John Wiley & Sons, Inc., New York, 1969, pp. 200-204.

Michel Verleysen is studying for his PhD in the field of neural networks on a Belgian IRSIA (Institute for the Encouragement of Scientific Research in Industry and Agriculture) fellowship at the Microelectronics Laboratory of the Catholic University of Louvain. His research activities and interests include VLSI realization of neural networks, content-addressable memories, and analog integrated circuits and systems.

Verleysen received the engineering degree from the Catholic University of Louvain, Belgium. He is a member of the IEEE.

Paul G.A. Jespers heads the Microelectronics Laboratory at the Catholic University of Louvain, where he teaches in the Department of Electrical Engineering. He has been a visiting professor at Stanford University in California. His current interest is in MOS integrated circuits and systems.

Jespers received the engineering degree from the Free University of Brussels and the PhD degree from the Catholic University of Louvain. He is vice-chair of the Steering Committee of the European Solid-State Circuits Conference. He was appointed IEEE Regional Director of Region 8 from 1971 to 1972 and is a fellow of the IEEE and a member of the International Neural Network Society.

Readers may address questions about this article to Michel Verleysen at the Catholic University of Louvain, Microelectronics Laboratory, 3 Place du Levant, 1348 Louvain-la-Neuve, Belgium.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 **Medium** 160 **High** 161