# 15

# Analog Implementation of an Associative Memory: Learning Algorithm and VLSI Constraints

MICHEL VERLEYSEN, JEAN-DIDIER LEGAT, and PAUL JESPERS

## 15.1. INTRODUCTION

Artificial neural networks can be classified by their architectures; the main classes of models are feedback networks (as the Hopfield net), layered networks (as the multilayer perceptron), and auto-organized networks (as the Kohonen map). All these networks realize associative memory processing; the Hopfield model corresponds to an autoassociative memory, perceptrons correspond to heteroassociative memories, and Kohonen maps to classifiers, although this distinction is quite artificial. Most current models of artificial neural networks are based on one of these three architectures; many differences in the structure, in the learning rule, or in the way the data are represented can, however, exist in other models. Some combine these concepts, for example, by adding feedback to multilayer perceptrons. The diversity of all models makes it difficult to build a complete list with their full characteristics.

What is important here is to point out the common features in all these models. The first characteristic relies on the common operations involved in nets. In the Hopfield model, a matrix product is performed between the input vector (or the output of the previous computation step) and a matrix of weights. In layered nets of perceptrons, a similar matrix–vector product is performed in each layer; they differ from the Hopfield model by the absence of feedback. In Kohonen maps, the first layer is devoted to the measure of distances between the input vector and the stored ones, which is also a matrix–vector product.

The second important common aspect between these models involves the precautions that must be taken regarding the precision and the dynamics of the values involved in the operations (synaptic weights, activation levels, ...), and the dimensions and cascadability of chips. Except for some particular networks, interesting properties of neural nets can only be found when their size (number of neurons and synapses) is large. Real parallel computing can then be performed, and the speed of the computations can become significant in comparison to classical serial computing methods. The advantages of analog VLSI, in particular situations, will be discussed in a later section, but it will be seen that the problems mentioned above, especially the cascadability, are not so simple to handle with analog design; again, the solutions proposed for one model can easily be transposed to others, and the topological layout of the net is of less importance.

This chapter emphasizes the Hopfield model. It has been chosen for several reasons. First, historically, it was one of the most used networks for associative memories; applications were lacking for self-organizing maps, and multiple-layer nets of perceptrons suffered from the complexity of their learning algorithms. Second, the simplicity of the Hopfield model makes it easier to study compared to more complex networks. Finally, as mentioned above, VLSI constraints for the implementation of neural networks are similar in the different models; it is thus natural to experiment with a simple model, keeping in mind that the results can easily be transposed to more complex ones.

## 15.2. HOPFIELD NETWORK

The Hopfield model is a simple recurrent network, as shown in Fig. 15.1. Refer to Chapter 1 for operation and details of this architecture.

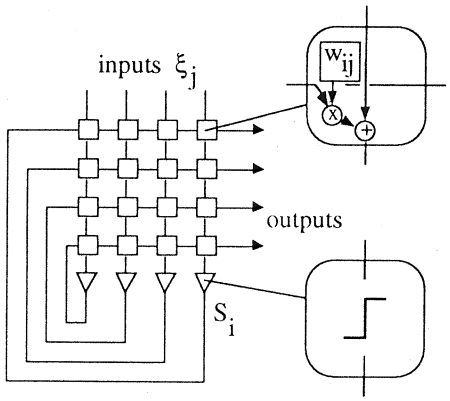


Fig. 15.1. Hopfield network.

## 15.3. LEARNING RULES FOR HOPFIELD NETWORKS

### 15.3.1. Hebb's Rule

The Hebb rule (Hebb, 1949) for storing $p$ bipolar patterns is given by

$$\left.\begin{aligned} w_{ij} &= \frac{1}{N}\sum_{k=1}^{p} \xi_i^k \xi_j^k \qquad \text{if } i \neq j \\ w_{ii} &= 0 \end{aligned}\right\} \tag{1}$$

where $w_{ij}$ is the weight of the synapse connecting neurons $i$ and $j$, and $\xi_i^k$ is bit $i$ of pattern $k$ to be memorized.

For random stored patterns, it is generally agreed that a capacity limit of $0.15N$ vectors must not be exceeded, to keep performances acceptable. This corresponds to the theoretical limits found by McEliece (1987) and Abu-Mostafa (1985). To integrate this learning rule in VLSI circuits, where the precision and dynamics of the coefficients are limited, the clipping of weights must be examined. Equation (1) shows that each pattern to be stored contributes to each weight a value $+1$ or $-1$. If $p$ patterns are stored, it is thus obvious that $2p + 1$ different values are allowed for the weights. The number of bits required for storing the weights is then given by

$$B = \log_2(2p + 1) \tag{2}$$

The smallest integer value greater than $B$ is the number of digital memory points necessary for storing the weight.

Clipping the Hebb rule to a one-bit accuracy is possible by applying a sign function to the weight obtained by (1). The capacity, simply measured as the number of random patterns that it is possible to store without error, is then reduced to about $0.11N$ vectors (compared to $0.15N$ for the unclipped Hebb rule). Morgenstern (1987) proposed a variant of this method, the zero model, which consists of setting the weight to zero if its absolute value is lower than some threshold $Z$, and applying the sign function for the other cases. The capacity can then be raised to a mean value of $0.12N$.

### 15.3.2. Projection Rule

The projection rule (generalized-inverse rule) is covered and discussed in Chapter 1 (see also Personnaz, 1985). It is given by

$$\mathbf{W} = \mathbf{\Psi} \cdot \mathbf{\Psi}^{+} \tag{3}$$

where

$$\mathbf{\Psi}^{+} = (\mathbf{\Psi}^{\mathrm{T}} \cdot \mathbf{\Psi})^{-1} \cdot \mathbf{\Psi}^{\mathrm{T}}$$

and

$$\mathbf{\Psi} = [\xi^1, \xi^2, \ldots, \xi^p] \tag{4}$$

Clipping the weights obtained with Eq. (3) leads to disastrous results. The reason is the matrix inversion involved in the rule; pseudo-inverse is indeed a generalization of matrix inversion, which involves many divisions in its computation. Depending on the vectors, the results can theoretically have infinite dynamics, and all the information contained in this dynamics is completely lost with any type of truncation. Clipping the projection rule will thus not be considered here.

### 15.3.3. Optimization Rule

The projection rule guarantees that patterns $\xi^k$ are fixed points as long as these patterns are linearly independent; but the method does not speak about the degree of stability of such patterns. Basins of attraction can be represented as in Fig. 15.2 for all patterns memorized with the projection rule; circles around each pattern define the minimum basins guaranteed by the rule. It must be mentioned that Fig. 15.2 is only a schematic 2-D representation of the attractiveness of vectors; the reality would require an $N$-dimensional representation with vectors situated on hypercube vertices, and Hamming distances instead of 2-D Euclidian ones.

Good behavior of the algorithm is shown for patterns situated on the left part of Fig. 15.2; however, when the size of the basins of attraction guaranteed by the rule is small in comparison with the distance between patterns, the limits of an attraction domain can be close to a pattern;



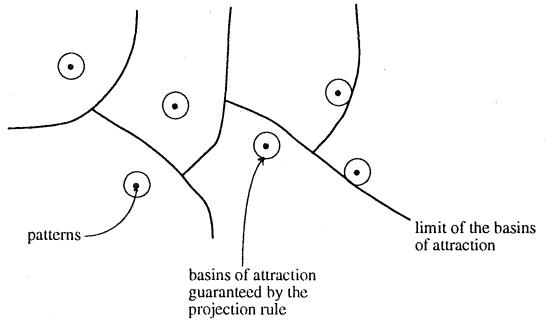


Fig. 15.2. Basins of attraction with the projection rule.

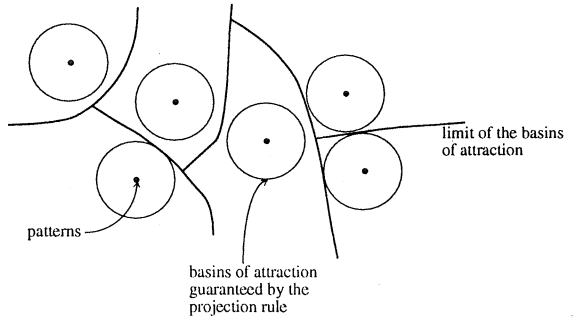


Fig. 15.3. Basins of attraction with the optimization rule.

this is nonoptimal, as shown in the right part of Fig. 15.2. We propose here an original algorithm (Verleysen et al., 1989) which maximizes the size of the basins of attraction for a set of given patterns (Fig. 15.3). This learning rule will be termed the "optimization rule."

If the Hopfield network of Fig. 15.4 is examined, it can be seen that a measure of the stability of a pattern is the difference between the activation of a neuron and its threshold $\Theta$. The activation $u_i$ is defined by

$$u_i = \sum_{j=1}^{N} w_{ij} y_j \tag{5}$$

A stability factor $S$ is defined here for a particular bit $i$ in a particular pattern $\xi^k$ as

$$S_i^k = |u_i - \Theta| \tag{6}$$

Without loss of generality, $\Theta$ will be set to 0 in the following, since a threshold can be formally replaced by a supplementary fixed input.

The proposed optimization rule relies on the following intuitive view. A change in one bit of the vector $\mathbf{y}$ (for example $-1$ instead of $+1$) can induce a change of a maximum of 2 (or $-2$) in the activation value $u_i$ (assuming that the weights $w_{ij}$ are in $[-1, +1]$). If the stability factor $S$ is constant for all bits of all stored patterns, and is set to 1 as in the projection rule, it means that one wrong bit in the pattern $\mathbf{y}$ can lead to a wrong calculation of any other bit in the pattern during the next iteration of the network. A remedy to this problem would be to increase the stability factor $S$, again for all bits $i$ and patterns $k$. If $S$ is set to $C$, a maximum number of $C/2$ erroneous

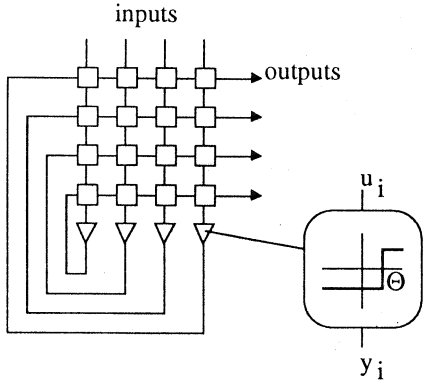


Fig. 15.4. Hopfield network.

bits will be corrected at the next iteration, since the signs of all $y_i$ are not affected by these wrong bits. Formally, for a stable state $\xi^k$,

$$|u_i| = \left| \sum_{j=1}^{N} w_{ij} \xi_j^k \right| \geq C \tag{7}$$

If $\mathbf{y}$ differs from $\xi^k$ by $D$ bits,

$$|u_i| = \left| \sum_{j=1}^{N} w_{ij} y_j \right| \geq C - 2D \tag{8}$$

Thus if $D < C/2$, the sign of $u_i$ in (8) will be identical to the sign of $u_i$ in (5), and this confirms that $\mathbf{y}$ will evolve to $\xi^k$ at the next iteration. $S$ is thus a good measure for the attractiveness of the patterns in one cycle; it is assumed here that a good attractiveness factor $S$ measured in one-cycle convergence will also lead to good attractiveness in several cycles.

The idea of the optimization rule is thus to maximize the value of $S$. Of course, $S$ cannot be increased indefinitely; there exists a maximum value of $S$ for which the system can be solved, with $w_{ij}$ in $[-1, +1]$. The problem is the dependence of this maximum on the value of patterns $\xi^k$. Intuitively, it is obvious that a weak correlation between the patterns $\xi^k$ will permit a higher value for $S$ than if the patterns are strongly correlated; this is, however, extremely difficult to compute analytically. It is thus proposed to rewrite the system as an optimization problem. The objective is now to maximize $S$,

$$S_i^k = \left| \sum_{j=1}^{N} w_{ij} \xi_j^k \right| \tag{9}$$

Forcing the same $S$ for all bits $i$ and all patterns $k$ is, however, too restrictive. In order to have only one variable to maximize, the minimum of all $S_i^k$ will be considered. The problem is thus

$$\text{Maximize } S \quad \text{where } S = \min_{i,k} (S_i^k) \tag{10}$$

and

$$S_i^k = \left| \sum_{j=1}^{N} w_{ij} \xi_j^k \right| = \sum_{j=1}^{N} w_{ij} \xi_j^k \xi_i^k \tag{11}$$

It can be seen in Fig. 15.4 and in equations (5) to (11) that the maximizations are independent of $i$. In order to simplify the algorithm that will solve the problem, different $S_i$ will be considered for the different bits. The complete problem can then be defined by:

$$\text{Maximize } S_i \quad \text{where } S_i = \min_{k} (S_i^k) \tag{12}$$

with

$$S_i^k = \sum_{j=1}^{N} w_{ij} \xi_j^k \xi_i^k - E_i^k \tag{13}$$

under the constraints

$$E_i^k \geq 0 \tag{14}$$

$$-1 \leq w_{ij} \leq 1 \tag{15}$$

the constraints being valid for all patterns $k$ and all bits $j$. The coefficients $E_i^k$ add free degrees to the system, so that the $S_i^k$ can be different. The problem (12)–(15) must be repeated $N$ times for the $N$ bits $i$, and can easily be solved by a standard simplex procedure (Lasdon, 1970).

This rule can be used in problems where on-line computation of weights is not necessary. Indeed the complexity of the computations, and the restriction that learning is nonadaptive (all weights must be recomputed each time information is added into the network), make the rule best suited to systems where the learning can be computed by an external computer before using the associative memory in its recall phase.

### 15.3.4. Clipping the Optimization Rule

The problem of clipping is quite different with this learning rule. Equation (15) limits the dynamics of the coefficients to the range $[-1, +1]$; it could be limited to any other symmetric range without changing any result, because of the above-mentioned scaling property. Only the precision must then be known to evaluate the number of necessary bits in each memory point.

The algorithm applied to one column of the synaptic matrix can be seen as an optimization problem with $N$ variables (the $N$ synaptic weights) and $p$ constraints (the $p$ equations 13) to maximize simultaneously); $N - p$ variables will then directly take the values $-1$ or 1, while the remaining $p$ may take any value in $[-1, +1]$. In most problems, however, $N$ is much greater than $p$; furthermore, if this were not the case, it can be seen that the performance of the memory would be too poor for efficient applications. The optimization method thus guarantees that most of the weights can be coded into one bit.

Equation (9) shows the quantity to maximize for each of the patterns. It can intuitively be seen that the only effect of any further restriction on the variables (the synaptic weights) is to decrease $S$ in (9) and (10). Since $S$ is the quantity to be maximized, these restrictions have the same effect as a nonoptimal convergence of the optimization process; the solution will not be the optimum, but will be near it. Clipping the weights to the values 1 or $-1$, or eventually 0, is such a restriction. It is, however, difficult to appreciate the difference between $S$ in the optimal solution



and $S$ in the solution obtained with these restrictions, since it depends on the absolute value of $S$, which itself depends on the patterns to be memorized. However, since the number of weights where this restriction is applied is less than $p$, with a small ratio $p/N$, the quantity $S$ will not decrease drastically.

Our simulations show that there is only a very small decrease in memory performance if the weights are truncated to $\{-1, 0, +1\}$ or $\{-1, +1\}$ or $\{-1, +1\}$; the reader is referred to Chapter 1 for additional discussion of the effects of weight accuracy on memory retrieval.

This concept becomes important when the associative memory is realized with analog or digital VLSI circuits. The number of bits will indeed determine the complexity of each synapse, i.e., their size on the chip. If only one or two bits are needed, the number of synapses that can be implemented on a single chip can be increased, enhancing the performance of the memory for a fixed area. Circuits using low synaptic weight accuracy will be developed in the rest of this chapter.

## 15.4. VLSI ASSOCIATIVE MEMORIES

Modern computers are based on up-to-date VLSI techniques, and their performance and reliability depend on the quality of the technological process, the quality of the design, and the use of appropriate architectures. Neural networks can be viewed as a way to realize defined functions and objectives by using adaptive nonconventional methods that differ from conventional processing by their lack of need for deterministic algorithms. It is, however, obvious that neural networks cannot replace von Neumann's architecture. Present artificial neural nets are most effective in specialized processing, in the same way that mathematical coprocessors increase the efficiency of CPUs in mathematical problems. All functions that are, or will be, realized with neural networks can also be realized with any kind of personal computer or minicomputer by programming it with sequential languages. This is usually called simulation, although the term is not appropriate: what is the limit between simulation and realization when speaking about neural networks? Neurons and synapses are indeed biological cells, and their electronic or optical realization is already a "simulation" of their biological behavior; an implementation of the algorithm on a PC is a "simulation" of the behavior of the electronic circuit, and the first study of the algorithm is a "simulation" of its behavior when it is used for real applications.

VLSI is thus only one way among many others of realizing artificial neural networks, but it is a way adapted to the requirements of the problems intended to be solved with these architectures. Neural networks are structures where many simple processing elements are connected together, and where the computational power comes from the collective behavior of the network. VLSI is therefore an excellent candidate to their realization, since it is easy to implement the simple processing elements involved, and since the recent advances in VLSI technology allow the integration of large numbers of transistors and devices, and thus of the large number of neuronal cells, needed for a collective behavior.

Other implementations could also be considered, such as optical ones (see the chapters 18 and 19 in this book). Neural networks are certainly not the panacea, and obviously cannot be used for the resolution of all problems in engineering fields. In their present form, neural networks are only interesting when connected to general-purpose computing machines; and the mixing of electronics and optical processing is still an emerging technology that will not be fully used for decades! The only valid method of research for short-term use of neural networks thus seems to be VLSI implementation, although other solutions are not excluded for the future.

### 15.4.1. Perspectives and Objectives

An ideal associative memory would be a device that can be used in any association problem, of any size, and that finds the correct solution after a negligible delay. Such an ideal artificial neural network of course does not exist, and any VLSI realization will always be a compromise between size, performance, and speed of computation. These three concepts can be detailed in terms of VLSI cells.

### 15.4.2. Size

The collective behavior of neural networks necessitates large numbers of neurons and synapses. Hopfield networks, for example, require $N$ neurons and $N^2$ synapses to handle patterns of $N$ bits of information. This number is even larger in layered nets of perceptrons. Size is thus one of the most important features in VLSI implementations of neural networks, and will determine the complexity allowed for each cell, since the total number of transistors allowed on

a chip cannot be increased indefinitely. This number depends on the technology used, and on the percentages of defects per unit area. The number of cells that can be implemented on a single chip will thus be directly determined by the number of transistors in each cell, and the goal here is to simplify these cells as much as possible.

### 15.4.3. Performance

The performance of a defined associative memory structure is an important aspect to consider. This is normally affected by the precision of the weights, the activation values (signals fed into activation nodes), and the state values (output of activation nodes). For example, it is quite different working with one- or two-bit weights in Hopfield networks than with 16-bit weights. Since precision is area-consuming, the goal here will be to consider the constraint of weight precision in the VLSI network. In later sections of this chapter, reduced precision architectures as well as a compromise between network precision and speed will be considered.

### 15.4.4. Computation Speed

Speed is of course one of the goals of VLSI implementations of associative memories. If no speed requirements exist in a given neural network application, it is not worth realizing it in electronics since conventional computers can be utilized effectively. However, in many applications, neural networks are used as signal-processing devices where the time constants depend on physical constraints, such as in speech or image processing. Speed will thus also be an important aspect of the realization, which goes in the same way as the simplicity of the elements.

### 15.4.5. Analog and Digital Implementations

The advantages and drawbacks of analog and digital VLSI implementations of neural networks have already been the theme of many papers and articles (Verleysen and Jespers, 1991a). However, it seems important to clarify some ideas before going further in the description of analog implementations.

Digital cells are powerful for the implementation of complex functions used in some algorithms; the precision of the computations can be extended depending on the requirements of the algorithm by increasing the number of bits in each cell of the processors. However, this greatly increases the size of the cells, and it rapidly becomes impossible to realize the massive interconnections required by a given algorithm.

The connections between the neurons occupy most of the silicon area on a chip; these synapses contain the memory points, and the elementary operations such as product and/or sums. The neurons, however, contain only nonlinear functions (in the case of Hopfield networks or multilayer perceptrons); even if the neurons are larger than the synapses, due to the complexity of their operations, the number of synapses is generally much larger than the number of neurons, and most of the area will be occupied by synapses.

It is not unusual to handle networks with many thousands of synapses; the complexity of digital cells makes it hard to implement all of them on a single chip. Multiplexing is then the solution: the same cells are sequentially used for several operations. This of course reduces the processing speed in the same way that the silicon area is reduced.

If speed is not to be compromised, analog cells can be used, since the area occupied can be much less than the area occupied by digital cells for similar realized operations. Drawbacks of the use of analog cells are the restricted precision of the computations that can be handled, and cascadability problems. The dynamical range of analog values is indeed limited due to mismatch between components on the same chip. This normally restricts bit accuracy to 5 or 6 bits, unless elaborate matching techniques are employed.

## 15.5. CURRENT-MODE APPROACH

There are generally three ways to represent data in VLSI circuits. First, the information can be represented by a voltage at a node on a line; its transmission is then made in a natural way along the line. Here, the resistance of the wire can affect the voltages between two distant points. Currents may also be employed to represent data; in this case, the information is represented as a current flow in a circuit. Leakage currents of junctions can affect the value of a current, but the transmission of the information between two points is generally realized better than with voltages. The third way is to represent the data in a temporal way, with frequencies or widths of pulses; although this obviously offers some advantages for the regeneration of corrupted information, the circuitry needed to handle the pulses generally requires more area on silicon,



and will therefore not be considered in this chapter.

Most analog VLSI neural networks are based on the following principle: a product is realized at a synapse, between its input and a stored weight. The output value of the synapse is converted into current with a current source, and all resulting currents of synapses connected to a single neuron are summed (for free!) on a current line.

The neuron connected to the end of the current line will then apply a nonlinear operation to the input current. The information at the input of the neuron is called the "activation," while the one at the output is called the "state value," or "state" of the neuron. For the sake of simplicity, the relation between activation and state values is assumed to be a step function with zero threshold. A nonzero value for the threshold can be modeled by adding one synapse connected to the neuron, with a fixed input equal to 1, and with a connection weight equal to the negative of the threshold. The step function can be replaced by a sigmoid by varying the gain of the amplifier in the neuron.

The goal is to realize large sizes of synaptic arrays; it is thus necessary to design the network in such a way that large numbers of synapses can be connected together with sufficient precision that the neuron can make accurate decisions. If the current flowing from each synapse to the current line is binary (one current unit or no current), the neuron must be able to count the number of current units flowing to the current line. If the precision on the current units is infinite, adequate design of the neuron will avoid any decision problem in the neuron. However, a finite precision on the current units limits the number of synapses that can be connected together without inducing any wrong decision in the neuron.

The property of fault-tolerance of neural networks can compensate for some imperfections. However, it is usually agreed that the fault-tolerance relies only on random errors that could occur in the synapses, for example their being stuck to 1 or 0 at their output (see Chapter 14 of this volume by Chung and Krile for a detailed analysis of the effects of hardware faults on memory performance). Systematic errors in the synapses or in the neurons are more difficult to handle, since they may change the behavior of the circuit, and its convergence process. Wrong decisions taken by the neuron on the number of active synapses that are connected are of the second class of errors, the systematic ones. The values of the synaptic currents indeed depend on physical parameters like the oxide thickness of the transistor gate, but the errors remain constant once the circuit is realized; it is then not correct to speak about fault-tolerance for this kind of error and the precision of the currents at the output of a synapse will be of importance.

## 15.6. TWO-LINE SYSTEM

In the beginning of this chapter, it was shown that learning rules for Hopfield networks can be adapted to the constraints imposed by VLSI, keeping only one- or two-bits precision for weights. The realization of the synapses is then greatly facilitated: products are digital ones and can be realized with logic gates, and memorization of weights in static registers is not too area-expensive since only one or two memory points are needed.

Figure 15.5 shows the first current-mode architecture proposed by Graf and De Vegvar (1987a, b). The synaptic weight is contained in two memory points, and its possible values are $+1$, 0 and $-1$. The input value of a synapse is either 1 or $-1$, and is materialized by a positive voltage on either "in" or "inb."

The excitatory state of the synapse will inject current into the current line through transistors T1 and T2, while the inhibitory state will sink current from the line through transistors T3 and T4. The content of memory points M1 and M2 will be respectively 0 and 0 for a positive connection, 1 and 1 for a negative connection, and 1 and 0 for no connection. The state with M1 = 0 and M2 = 1 is not allowed.

In Graf's circuit, the excitatory and inhibitory

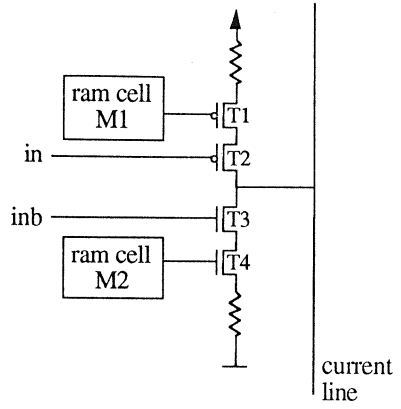


Fig. 15.5. Current-mode architecture.

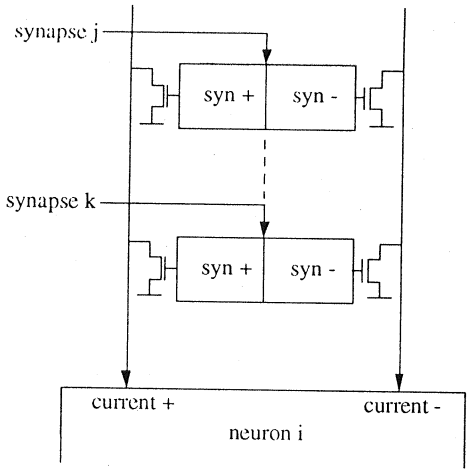


Fig. 15.6. Two-lines architecture.

currents flow through N-type and P-type current sources, respectively. The mismatch between the two types of transistors makes it impossible to obtain exactly the same currents for the two states of the synapses; as a consequence, systematic errors occur in the neuron decision function, and the convergence process of the network can be altered. Mismatch between N-type and P-type current sources come from the mobility differences between the two types of transistors. The size of the sources can of course be adjusted to compensate for these differences, but the exact ratio between the mobilities depends on the technological process, and cannot be known exactly before chip fabrication. Size compensation is thus not sufficient to ensure correct behavior of the circuit.

This mismatch problem can be avoided by using a two-line system with only one type of current source (Verleysen et al., 1989). The architecture is shown in Fig. 15.6. Its purpose is to use only one type of current source, in order to avoid the mismatching effects between N- and P-type transistors. However, even with one type of source, the currents will be slightly different one from another; this will be discussed later. As described in Fig. 15.6, all excitatory currents are summed on one current line, and all inhibitory currents are summed on another one. The role of the neuron will be to compare the two total currents, and to apply the nonlinear function to the difference.

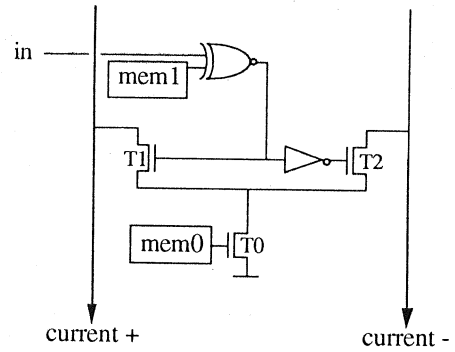


Fig. 15.7. Synapse.

### 15.6.1. Synapse

Each synapse in Fig. 15.6 is a programmable current source controlled by a differential pair (see Fig. 15.7). Three connection values are allowed in each synapse. If mem0 = 1, current is sunk to one of the two lines; mem1 and the input determines to which of them the current is sunk. If mem0 = 0, no connection exists between neurons $i$ and $j$, and no current flows to either the excitatory or the inhibitory line. This synapse is designed in accordance with the optimization learning rule described previously, where only 2-bit precision is necessary in the synaptic weights; the synapse input is here a logic (1-bit) value.

### 15.6.2. Neuron

Depending on the state of the XOR function (see Fig. 15.7), the current may be sunk either from the line current + or from the line current −. The comparison between the two total currents must be achieved in the neuron; this is done by means of the current mirror shown in Fig. 15.8. The currents on the two lines are converted into voltages across transistors T3 and T4; these

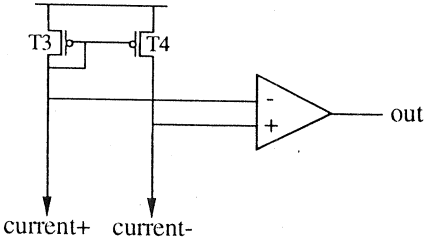


Fig. 15.8. Neuron architecture.



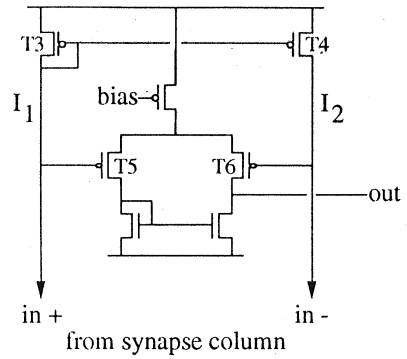


Fig. 15.9. Neuron amplifier.

voltages themselves are compared in the operational amplifier shown in the figure.

To obtain digital values at the output of a neuron, its gain must be very large; the opamp will thus be a simple amplifier with no feedback, as shown in Fig. 15.9.

## 15.7. PRECISION IN NEURON ELEMENTS

The design of the neuron clearly determines the sum-of-products precision achieved. Since each synaptic current is relatively small, the current mirror, and the other components in the neuron, have to be designed accurately to avoid computation errors. Several mismatch errors can be quantified in the neuron of Fig. 15.9 (Verleysen et al., 1989). First, threshold voltages of the two transistors in the mirror (T3 and T4) can differ (Verleysen et al., 1991b). For first order, the effect of this difference is directly related to the transconductance of a transistor in saturated mode:

$$\Delta I_{tm} \approx \sqrt{\left(2\mu C_{ox}\frac{W}{L}I_1\right)}\Delta V_{tm} \qquad (16)$$

where $\Delta I_t^m$ is the current error, $\mu$ is the mobility of carriers in the transistors, $C_{ox}$ is their oxide capacitance, $W/L$ is their size, and $\Delta V_{tm}$ is the difference of threshold voltages between T3 and T4. This leads to

$$\Delta I_{tm} \approx \frac{2}{V_{gs} - V_t}\Delta V_{tm} I_1 \qquad (17)$$

where $V_{gs}$ is the gate-to-source voltage of transistor T3, and $V_t$ its threshold voltage.

A second effect is the variation of $\beta$ factors between the two transistors. The resulting current difference is expressed by

$$\Delta I_\beta = \frac{\Delta\beta}{\beta} I_1 \qquad (18)$$

A third matching error in the mirror is due to the difference of the Early voltages between the two transistors. Taking the Early effect into consideration, and neglecting the Early effect of the N-type current sources, the expression for the current in the transistors is

$$I = \mu C_{ox}\frac{W}{L}\frac{(V_{gs} - V_t)^2}{2}\left(1 - \frac{V_{ds} - V_{dsat}}{V_{EA}}\right) \qquad (19)$$

where $V_{ds}$ is the drain-to-source voltage of the transistor, and $V_{dsat}$ is its pinch-off voltage. If we define

$$\lambda_p = \frac{1}{V_{EAp}} \qquad (20)$$

then the current in the two transistors becomes

$$I_{ds1} = \mu C_{ox}\frac{W}{L}\frac{(V_{gs1} - V_t)^2}{2}[1 - \lambda_{p1}(V_{ds1} - V_{dsat})] \qquad (21)$$

$$I_{ds2} = \mu C_{ox}\frac{W}{L}\frac{V_{gs1} - V_t)^2}{2}[1 - \lambda_{p2}(V_{ds2} - V_{dsat})] \qquad (22)$$

Equation (22) leads to

$$V_{ds2} = \frac{1}{\lambda_{p2}}\left(1 + \frac{2I_{ds2}}{\mu C_{ox}\dfrac{W}{L}(V_{gs1} - V_t)^2}\right) \qquad (23)$$

As proposed by Hoekstra (1990), the derivative of $V_{gs1}$ can be computed by differentiating (21) implicitly with respect to $\lambda_{p1}$. This leads to

$$0 = \frac{\partial V_{gs1}}{\partial\lambda_{p1}}[2 - \lambda_{p1}(V_{gs1} - V_t)] - (V_{gs1} - V_t)V_{gs1} \qquad (24)$$

$$\frac{\partial V_{gs1}}{\partial\lambda_{p1}} = \frac{(V_{gs1} - V_t)V_{gs1}}{2 - \lambda_{p1}(V_{gs1} - V_t)} \qquad (25)$$

Differentiating $V_{ds2}$ in (23) with respect to $\lambda_{p2}$ results in

$$\frac{\partial V_{ds2}}{\partial\lambda_{p2}} = \frac{V_{ds2}}{\lambda_{p2}} \qquad (26)$$

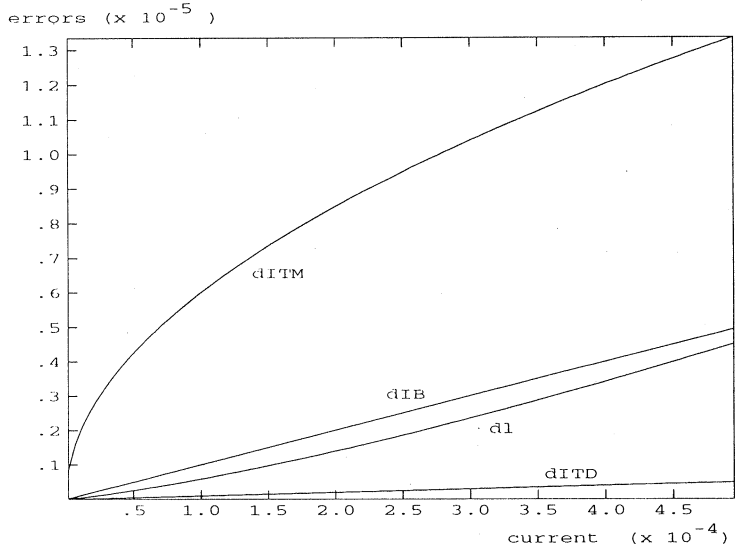


Fig. 15.10. Errors in a current mirror.

Substituting these values in (23) for a fixed $\Delta\lambda_p$ equal for the two transistors but with opposite signs (worst case) leads to a resulting difference between $V_{ds2}$ and $V_{ds1}$:

$$V_{ds2} - V_{gs1} = \left(\frac{V_{ds2}}{\lambda_{p2}} - \frac{(V_{gs1} - V_t)V_{gs1}}{2 - \lambda_{p1}(V_{gs1} - V_t)}\right)\Delta\lambda_p \tag{27}$$

which reported in terms of current difference between $I_1$ and $I_2$ gives approximately

$$\Delta I_\lambda = \left[\frac{V_{ds2}}{\lambda_{p2}} - \frac{(V_{gs1} - V_t)V_{gs1}}{2 - \lambda_{p1}(V_{gs1} - V_t)}\right]\Delta\lambda_p \frac{I_1}{V_{EA}} \tag{28}$$

A fourth and last error to consider is the mismatch between the two transistors at the input stage of the differential amplifier which measures the difference between the gate-to-source voltages of the two transistors in the mirror (i.e., the threshold voltage difference between T5 and T6 in Fig. 15.9). This mismatch, reported in terms of current difference, is given by

$$\Delta I_{td} = \Delta V_{td} \frac{I_1}{V_{EA}} \tag{29}$$

In order to compare these four errors, realistic values are chosen: $I$ from 0 to 500 μA; $\mu_p C_{ox} = 1.5 \times 10^{-5}$ A/V$^2$ (standard CMOS process); and $W/L$ must be chosen to cope with the maximum current (500 μA). If $(V_{gs} - V_t)$ can be up to 1.5 V, then

$$\frac{W}{L} = \frac{I_{max}}{\mu_p C_{ox}(V_{gs} - V_t)^2/2} \approx 30 \tag{30}$$

with $V_t = 0.8$ V, $V_{EA} = 20$ V, $\Delta V_{tm} = 10$ mV, $\Delta\beta/\beta = 0.01$, $\Delta V_{td} = 10$ mV, and $\Delta\lambda_p = 0.01$ V$^{-1}$. These values can be reached with careful design of the mirrors and comparators. The errors are illustrated in Fig. 15.10.

The error generated by the threshold voltage difference in the current mirror is obviously the most significant one, especially for small currents; this results from the fact that the error is proportional to the square root of the size of the transistors in the mirror. Even for small currents, the error is thus significant because the two transistors must be large enough to drive the maximum current (here 500 μA). A solution to this problem would be to connect several mirrors in parallel, each of them being active only when necessary to drive the total current.

## 15.8. SUMMARY

This chapter presents a general method for implementing neural networks with VLSI circuits. It shows how basic learning algorithms for associative memory can be adapted to the requirements of VLSI circuits. The basic principles of analog VLSI architectures are presented, together with most precision limitations encountered with such technology. It is also shown how to overcome such limitations by appropriate design of the neuron.

The principles explained in this chapter are illustrated for the Hopfield memory network. They are, however, more general, and can be used in most neural network implementations where the basic operation is the sum-of-product (i.e., the product of a matrix and a vector), and where binary weights may be used.

This chapter also presents a learning rule for the Hopfield memory adapted to such circuits; the rule shows good performance even when the weights' dynamic range is restricted to one or two bits.



## REFERENCES


Abu-Mostafa Y. S. and St. Jacques, J.-M. (1985). "Information Capacity of the Hopfield Model," *IEEE Trans. Information Theory*, **31**(4), 461–464.

Graf, H. P. and De Vegvar, P. (1987a). "A CMOS Associative Memory Chip Based on Neural Networks, in *Proceedings of the IEEE International Solid-State Circuits Conference*, New York.

Graf, H. P. and De Vegvar, P. (1987b). "A CMOS Implementation of a Neural Network Model," in Losleben, P., ed. *Proceedings of the 1987 Stanford Conference on Advanced Research in VLSI*. MIT Press, Cambridge, MA.

Hebb, D. O. (1949). *The Organization of Behaviour*. Wiley, New York.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA.

Hopfield, J. J. (1982). "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci., USA*, **79**, 2554–2558.

Lasbon, L. (1970). *Optimization Theory for Large Systems*. Collier-Macmillan, London.

McEliece, R. L., Posner, E. C., Rodemich, E. R., and Venkatesh, S. S. (1987). "The Capacity of the Hopfield Associative Memory," *IEEE Trans. Information Theory*, **33**(4), 461–482.

Morgenstern, I. (1987). "Neural Networks and Chip Design," *Condensed Matter*, **67**, 265–270.

Hoekstra, A. (1990). "An Associative Memory Based on Neural Networks in CMOS VLSI Circuits," Master Thesis, Faculty of Electrical Engineering, University of Twente.

Personnaz, L., Guyon, I., and Dreyfus, G. (1985). "Information Storage and Retrieval in Spin-glass-like Neural Networks," *J. de Physique Lett.*, **46**, 359–365.

Verleysen, M. and Jespers, P. (1991a). "VLSI Chips for Neural Networks," in *Progress in Neural Networks*, Vol. 1, O. M. Omidvar, ed. Ablex, Norwood, NJ.

Verleysen, M. and Jespers, P. (1991b). "Precision of Computations in Analog Neural Networks," in *VLSI Design of Neural Networks*, Ramacher, U. and Rückert, U., eds. Kluwer Academic Publishers, Boston.

Verleysen, M., Sirletti, B., Vandemeulebroecke, A., and Jespers, P. (1989). "Neural Networks for High-storage Content-addressable Memory: VLSI Circuit and Learning Algorithm," *IEEE J. Solid-State Circuits*, **24**(3), 562–569.