The Simplicial Neural Cell and Its Mixed-Signal Circuit Implementation: An Efficient Neural-Network Architecture for Intelligent Signal Processing in Portable Multimedia Applications

Radu Dogaru, *Member, IEEE*, Pedro Julian, Leon O. Chua, *Life Fellow, IEEE*, and Manfred Glesner, *Fellow, IEEE*

***Abstract*—This paper introduces a novel neural architecture which is capable of similar performance to any of the "classic" neural paradigms while having a very simple and efficient mixed-signal implementation which makes it a valuable candidate for intelligent signal processing in portable multimedia applications. The architecture and its realization circuit are described and the functional capabilities of the novel neural architecture called a *simplicial neural cell* are demonstrated for both regression and classification problems including nonlinear image filtering.**

***Index Terms*—Cellular neural networks, feedforward neural networks, image reconstruction, mixed analog–digital integrated circuits, pattern classification, piecewise linear (PWL) approximation.**

I. Introduction

In recent years there has been a high demand for integrating multimedia applications in portable devices, cellular phones, digital cameras, video-cameras, or personal digital assistants being among some of the most widely known examples. Extensive computation is required by various signal processing tasks involving nonlinear filtering, classification, and other similar functions which are often performed by software realizations of neural-network models such as multilayer perceptrons (MLP), support vector machines (SVMs), and so on. While such realization may require costly digital signal processors or specialized hardware, the portable nature of the application impose a tradeoff between power consumption and compactness on one side and computational efficiency on the other side. In this paper, we focus on results in developing a *novel neural-network architecture* with performance similar to the most advanced neural architectures while having a simpler, more compact, and efficient circuit realization which leads to dramatic improvements in terms of implementation efficiency.

For example, the SVM networks [1], deeply rooted in statistical learning theory offer very good solutions to complex problems while using a guaranteed convergent learning algorithm. However, its complexity is large and does not appeal to very large scale integration (VLSI) implementations. On the other side, the Adaline is easy to train with the least-mean square (LMS) (or Widrow–Hoff) algorithm [7], the learning is guaranteed convergent, but the area of applicability is limited only to linearly separable problems.

The above observations led to the following question: Is it possible to develop novel neural-network models, with nonconflicting features? A positive answer was the goal of the research leading to the solution presented within this paper. It turns out that the theory of simplicial piecewise linear (PWL) function approximations, developed by circuit theorists in the 1970s, [2]–[4] is a powerful tool and it proved beneficial in defining novel neural-network models with an emphasis on their implementation capabilities.

Motivated by recent research in the area of cellular neural networks (CNNs) [5] this paper introduces a novel neural architecture, called a *simplicial neural cell*, and its corresponding circuit. The functional properties of the cell are investigated. The novel cell has several attractive properties which makes it well suited for embedding in reconfigurable architectures for intelligent signal processing in portable multimedia devices.

1) There are no multiplicative synapses.
2) The cell has a very simple and efficient mixed-signal implementation in common VLSI technologies but also in promising nanotechnologies of the near future.
3) The functional performances for either regression and classification problems are similar to those of classic yet more complicated neural network architectures.
4) The central device around which the entire cell is built is a *digital RAM* or its emulation via a *compact neural network* [6] which can be easily reprogrammed.
5) The learning process consists in a simple LMS algorithm, which has a guaranteed convergence and it is easy to implement in VLSI technologies.

The *simplicial neural cell* is defined as a linear perceptron operating in an expanded feature space. The expanded feature space with a dimension of 2^n is obtained by computing only

Manuscript received April 14, 2001; revised October 30, 2001. This work was supported in part by the "VolkswagenStiftung, Federal Republic of Germany" through its program "Cooperation with Natural and Engineering Scientists in Central and Eastern Europe.

R. Dogaru is with the Department of Applied Electronics and Information Engineering, Polytechnic University of Bucharest, Bucharest 75547, Romania (e-mail: radu_d@ieee.org).

P. Julian is with CONICET (Consejo Nacional de Investigaciones Cientificas y Technicas), 1033 Cap. Fed., Argentina, on leave from the Departamento de Ingineria Electrica, Universidad Nacional del Sur, 8000 Bahia Blanca, Argentina.

L. O. Chua is with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720 USA.

M. Glesner is with the Institute of Microelectronic Systems, Darmstadt University of Technology, D-64283 Darmstadt, Germany.

Publisher Item Identifier S 1045-9227(02)04426-0.

$n+1$ fuzzy-membership functions of the n-dimensional input vector. Therefore the computational complexity is only $O(n)$ for both learning and retrieval. The novelty of our approach consists in exploiting the theory of simplicial subdivision (previously used in PWL approximations of nonlinear circuits) [3], [19], [20] for defining the above fuzzy-membership functions.

The paradigm of CNN [5] is a VLSI-oriented paradigm for signal processing. Its lattice structure makes it particularly well suited for image processing applications. The *standard CNN architecture* is based on a lattice of interconnected *cells*, each of them being associated with an image pixel whose output is defined by a sum of weighted inputs (corresponding to a square neighborhood around the central pixel). The cells are said to be coupled if recurrent connections with neighboring cells are present, and uncoupled otherwise. In this paper, we will consider the case of uncoupled CNN cells. The uncoupled standard CNN cell can be also regarded as an Adaline, following the denomination introduced by Widrow [7]. An extension of the Adaline, called herein a *nested* Adaline was introduced in [6] (under the name *multinested universal CNN cell*), where it was shown that arbitrary Boolean functions can be represented using an $O(n)$ complexity adaptive structure defined by

$$c = \operatorname{sign}\left(z_n + \left|z_{n-1} + \left|\ldots z_2 + \left|z_1 + \sum_{i=1}^{n} b_i v_i\right|\right|\right|\right) \quad (1)$$

where c is the binary output of the nested Adaline CNN cell, v_i are n binary inputs and $\{z_i, b_i\}_{i=1,\ldots,n}$ are a set of trainable parameters. It is assumed that if $z_k = 0$ for $k \geq j$, there are only $j-1$ absolute value functions ("nests") in (1), to the right of $z_k = 0$. Hence, the Adaline or the linear threshold gate (LTG) is a special case of (1) for $k = 1$.

From the image processing perspective, black-and-white image processing corresponds to binary CNN cells (where both inputs and outputs belong to the set $\{0,1\}$) while gray level image processing corresponds to inputs and outputs belonging to the interval $[0,1]$.[1] Standard CNN cells were initially designed to have simple circuit implementations, despite of several drawbacks such as: 1) The local information processing is restricted to simple tasks, often falling to the class of linearly separable problems; 2) The lack of *learning* capabilities; and 3) The lack of robustness, particularly for gray level (continuous) image processing. So far several solutions were proposed to overcome some of the above drawbacks. However, none of these solutions is capable of removing *all* of these drawbacks. As a consequence, research on *universal CNN cells* was recently conducted [6], [8].

A universal CNN cell implements any arbitrary nonlinear mapping $y = f(\mathbf{u}, \mathbf{G})$ of an input vector $\mathbf{u} = [u_1, u_2, \ldots, u_n]$ to a desired output y by properly choosing the *gene*[2] vector G. An image processing task, or in general a signal processing problem, can be formalized by an expert as a sequence of local rules, or it might be specified by a set of input–output samples

[1]Without loss of generality and for the simplicity of exposition, the binary set used in this paper is $\{0, 1\}$ although most of the CNN literature uses the set $\{-1, 1\}$.

[2]A cell *gene* was defined in [5] as a vector formed of all parameters defining the cell model.

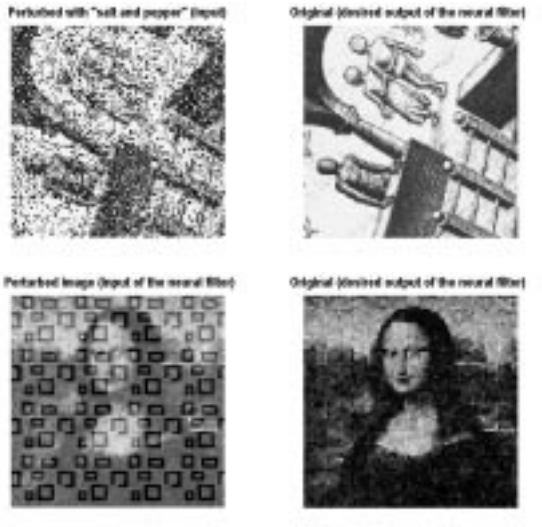


Fig. 1. A set of training image samples for the task of removing "salt and pepper" noise (upper row) and for the task of square scratches removal (lower row).

(for example, an image filtering task can be specified by image samples such as those shown in Fig. 1). In the first case, a sequence of logical inferences of the type IF $(\mathbf{u} = \mathbf{v}_j)$ THEN $(y = c_j)$ is derived in the form of a *decoding book* [5], while in the second case a learning algorithm should be employed to determine $\mathbf{G}$.

For the case of black–white image processing the n-dimensional input vector $\mathbf{u}$ associated with each cell (pixel), is confined to the vertices $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j, \ldots, \mathbf{v}_N$ of an n-dimensional hypercube (Fig. 2), where a vertex is an n-dimensional vector $\mathbf{v}_j = \left[v_1^j, v_2^j, \ldots, v_n^j\right]$, $v_i^j \in \{0,1\}$, $i = 1, \ldots, n$. The number of vertices is equal to $N = 2^n$. The brightness of each vertex in Fig. 2 corresponds to the desired output value $c_j = y(\mathbf{v}_j)$. Conceptually, the simplest universal CNN cell for the binary case is a ROM where $\mathbf{u}$ is the address bus while the content of the ROM is a N-dimensional binary gene vector $C = [c_1, c_2, \ldots, c_j, \ldots, c_N]$. In order to achieve programmability, a RAM may replace the ROM. Since in a CNN many cells are supposed to be implemented on the same chip, this solution is rather expensive in terms of occupied area due to the RAM implementation complexity of $O(2^n)$.

The multinested universal CNN cell (extensively described in [6]), herein called a *nested Adaline* and defined by (1) was proved to emulate the RAM, therefore being a universal programmable logic (Boolean) operator with an implementation complexity of only $O(n)$. A circuit implementation of this concept exploits the nonmonotone characteristic of resonant tunneling diodes (RTD) to achieve a very compact and fast realization in emerging nanotechnologies [9] although it can also have a convenient implementation in more usual CMOS technologies.

In practice, the number of useful binary signal and image processing tasks found so far is quite limited, and many of the interesting processing tasks deal with *continuous* multidimensional signal processing. The transition from *binary* to *continuous* information processing is graphically represented

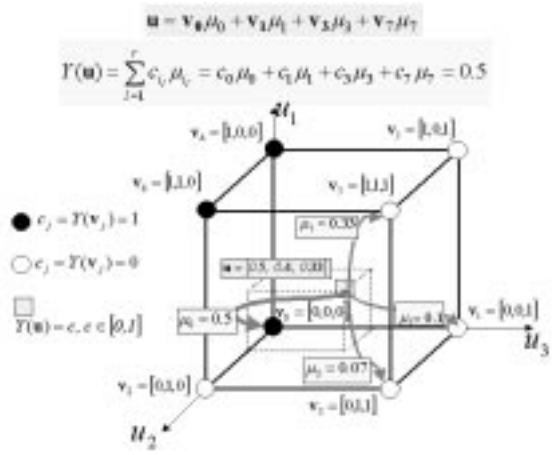


Fig. 2. The mapping of the input space using a simplicial fuzzy inference system.

by expanding the input space to the entire volume of the hypercube (see Fig. 2) and allowing that $y = c_j \in [0, 1]$.

In addition, besides image processing intelligent signal processing includes classification and other tasks. Therefore it is of great interest to develop a neural architecture that is universal not only for the binary case but also as a universal approximator, hence being capable of processing continuous signals. In addition, it is desirable to develop a neural architecture with an efficient circuit implementation.

Our main idea is to employ *simplicial PWL kernels* to expand the input feature space so that a simple Adaline can be trained in this expanded space. In order to decrease the approximation error each input can be further expanded to more inputs using a simple nonmonotone transformation, as shown in Section VI-C. The resulting networks are able to approximate not only binary-to-binary mappings but any type of functions, thereby expanding the range of applications from Boolean function representation to arbitrary classification and regression tasks.

All networks in this category have the major advantage of a guaranteed convergent and simple (gradient based) learning algorithm. As shown in Section III, their circuit implementation is simple and convenient and its complexity can be reduced to $O(n)$ as shown in Section III-A.

Although the nonlinear expansion of the input space is not a new idea, dating back to Cover's results in [10], it proved to be highly effective, being adopted in modern approaches, such as the support vector machines and other types kernel-based networks. The general model of a neural net in this category is given by

$$y = \sum_{j=1}^{\mathrm{ne}} c_j \mu_j(\mathbf{u}) \tag{2}$$

where $\mathbf{u} = [u_1, u_2, \ldots, u_n]$ is the n-dimensional input vector, ne is the dimension of the expanded space, and c_j is a set of parameters composing the gene vector $\mathbf{G}$. As long as the kernel functions $\mu_j(\mathbf{u})$ are predefined, learning from examples reduces to the task of training an Adaline in an input space formed by the outputs of the basis (or kernel) PWL functions $\mu_j(\mathbf{u})$. For the same task, the quadratic optimization techniques proved to converge toward an optimal margin separation hyper-plane [1] and they can also be employed with even more success to determine the parameters c_j. For this type of networks, the main issue is how to choose the kernel functions in such a way that they are easy to implement and compute, leading to simple structures with highly efficient mixed-signal implementations.

The *pyramidal cell* [8] was the first attempt to derive a kernel-based PWL neural network, which was demonstrated to be *universal at least for the class of Boolean functions*. With the goal of binary universality in mind, a special set of $\mathrm{ne} = 2^n$ basis functions $\mu_j(\mathbf{u})$ was developed. Each basis was entirely determined by the vertex vector $\mathbf{v}_j$ which is nothing but the binary representation of the index "j." In addition to its Boolean universality, the cell was proved to posses some interesting properties in approximating continuous mappings such as those required by various nonlinear image filtering tasks. Although it requires 2^n parameters, the pyramidal cell model was the first for which was shown that each parameter can be quantified with a very small number of bits, i.e., it is very robust. The same interesting property stands for the case of the *simplicial neural cell* as discussed in Section II-A and in Sections V and VI.

Despite its interesting features, the pyramidal universal cell still has two major drawbacks: First, there is no proof that it represents a universal approximator for continuous mappings (the proof stands only for binary mappings, i.e., Boolean functions). Second, although it has a small number of basic building blocks, its implementation complexity is of $O(2^n)$ and the only way of reducing it is by sacrificing computation time which then becomes $O(2^n)$ for an $O(n)$ implementation complexity. In order to overcome the above drawbacks, a *simplicial neural circuit* was recently proposed [11] following a theoretical motivation exposed in [12]. Although it still requires 2^n coefficients c_j in the model given by (2), it was shown that an elegant mixed-signal circuit implementation exists so that the implementation complexity can be effectively reduced to $O(n)$ while the computation time is of the same order. Moreover, there are theorems in [3] which guarantee that the resulting network is a universal approximator for any continuous input–output mapping and not only for the Boolean functions.

The model of the simplicial neural cell is a particular case of (2) for $\mathrm{ne} = n + 1$ and can be written as

$$y = \sum_{l=1}^{n+1} c_{i_l} \mu_{i_l}(\mathbf{u}) \tag{3}$$

where i_l is a vertex index ranging between zero and $2^n - 1$. It is equivalent to index j in the general kernel based architecture described by (2). An interesting feature of the *simplicial neural cell circuit*, presented in more detail in Section III, is the following (see also Fig. 3).

Generating a ramp signal increasing from zero to one (assuming that all inputs $u_i \in [0, 1]$) during a predefined period of time T (the systems' clock period), the correct sequence of $n+1$ binary words $_{i_l}$ in (3) is provided at the outputs $v_i^{i_l}$ of n comparators. The comparators are connected such that their inverting inputs are all connected to the unique ramp

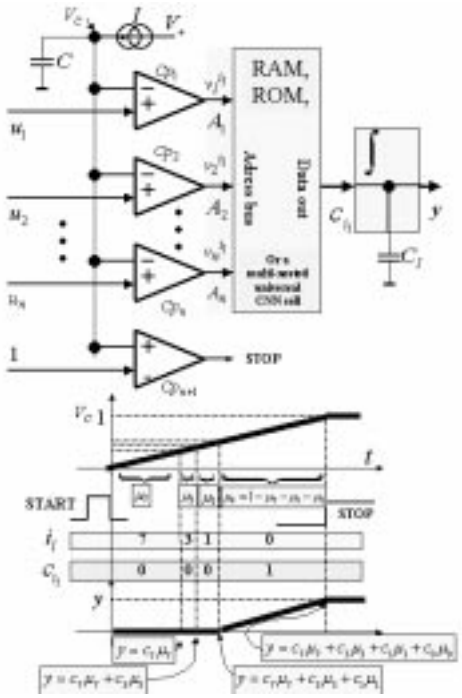


Fig. 3. An efficient mixed-signal circuit for implementing the simplicial neural cell.

signal while their noninverting inputs are the actual inputs of the simplicial neuron. Moreover, it is easy to prove that each binary code of i_l is present at the output of the comparators for a period of time, which is a fraction of T equal to $\mu_{i_l}(\mathbf{u}) \cdot T$. By employing a device (e.g., a RAM addressed by $\mathbf{v}_{i_l}$)[3] which maps the input binary code $\mathbf{v}_{i_l}$ to its corresponding coefficient c_{i_l} and assuming that this mapping is done almost instantaneously, a current signal c_{i_l} is also present at the output of the RAM for the same period of time $\mu_{i_l}(\mathbf{u}) \cdot T$. It is clear that employing a simple capacitor to integrate the output of the RAM, at the end of the entire computation cycle T the voltage on the capacitor is proportional to y in (3). The main component of the *simplicial neural network* or *S-NN* is the device mapping a binary input vector $\mathbf{v}_{i_l}$ (representing the index i_l in a binary form) to its associated coefficient c_{i_l} in (3). A detailed discussion on implementing this device is provided in Section III-A.

The c_{i_l} coefficients are subject to LMS (Widrow–Hoff) or other similar learning algorithms. For example one may use the algorithms for finding the optimal margin hyperplane introduced in the SVMs literature and proved to ensure best generalization performances. Here it is important to point out that although there are 2^n coefficients to learn, since only $n+1$ of the basis functions are nonzero at each presentation of an input stimuli the learning process has the same complexity as that of a simple linear perceptron with n inputs.

The theory of the simplicial subdivision is exploited showing that the inference performed by the simplicial neural network can be recast conveniently in a neuro-fuzzy formalism, detailed in Section II. A simple and efficient circuit implementation is then proposed in Section III, while the general training procedure is described in Section IV. Section V focuses on the regression functional capabilities of the novel neural architecture, evaluated for several nonlinear image filtering problems. In Section VI the functional capabilities for pattern classification of the simplicial neural network are investigated showing that it has similar performances to those obtained using "classic" neural architectures on a set of benchmark problems. A brief analysis of the hardware efficiency compared to classic solutions is given in Section VII and the concluding remarks are provided in Section VIII.

[3] $\mathbf{v}_{i_l}$ is the binary vertex (code) representing the index i_l as an unsigned binary word.

II. Simplicial Neural Networks as Neuro-Fuzzy Inference Systems

Let us consider an arbitrary point $\mathbf{u}$ belonging to the input hypercube, and linearly decomposed in its associated vertices $\mathbf{v}_j$ (see Fig. 2 for an example). From the fuzzy theory point of view, each vertex of the hypercube represents a crisp IF–THEN rule, more precisely it codes the rule IF $(\mathbf{u} = \mathbf{v}_j)$ THEN $(y = c_j)$. Such a list of rules can be loaded in a RAM (or ROM) with c_j corresponding to the data bus.[4] A crisp (Boolean) inference cell will provide an output only when $\mathbf{u}$ is one of the vertices. What happens when $\mathbf{u}$ represents an arbitrary vector coding a gray-scale portion of an image or a set of continuous signal samples? The crisp (Boolean) table stored in a RAM *cannot* provide a solution. However the fuzzy logic theory provides an elegant solution for determining the output. Indeed, the premises $(\mathbf{u} = \mathbf{v}_j)$ are no longer FALSE or TRUE, instead they are assigned a truth value $0 \leq \mu_j(\mathbf{u}) \leq 1$, where $\mu_j(\mathbf{u})$ denotes the fuzzy membership of vector $\mathbf{u}$ to the premise $(\mathbf{u} = \mathbf{v}_j)$. The choice of the fuzzy membership is subjective in nature, thereby giving rise to criticisms from many adversaries of fuzzy-logic. By establishing an equivalence between the nonlinear mapping (3) and a fuzzy inference system, this criticism is somehow overcome. Indeed, the theory of simplicial subdivision provides an effective method for *evaluating* $r \leq n+1$ nonzero membership functions instead of 2^n arbitrarily predefined membership functions, as in a typical fuzzy or neuro-fuzzy inference system. An efficient and simple algorithm for computing the simplicial membership functions can be defined via a mixed-signal circuit, as shown in Section III.

The resulting *simplicial neural cell* can be regarded as the equivalent of a Tsukamoto's fuzzy inference *system* [13, p. 154] and therefore its output is computed following the "center of gravity rule" as $y = \sum_{j \in J} c_j \mu_j / \sum_{j \in J} \mu_j = \sum_{j \in J} c_j \mu_j$, since $\sum_{j \in J} \mu_j = 1$ by definition in the simplicial decomposition [3]. The above relation is identical to (3). For the example in Fig. 2, the $r = 4$ vertices (rules) used to infer the output y are given by $J = \{i_l\} = [i_1, i_2, i_3, i_4] = [7, 3, 1, 0]$, which can be obtained using the simplicial decomposition algorithm in [3], [12] or its circuit implementation described in Section III. Consequently, the corresponding membership functions are evaluated and the output can be computed for a given choice of the gene vector.

[4] If c_j is not binary, a wider binary word coding it can still be stored in a digital RAM (or its multinested universal CNN cell emulation).

In order to use the simplicial neural cell for regression or classification, one has to determine the *gene*, i.e., the vector of coefficients $\mathbf{C} = [c_1, c_2, \ldots, c_j, \ldots, c_N]$ such that $\|d^p - \sum_{l=1}^{r} c_{i_l}\mu_{i_l}(\mathbf{u}^p)\| < \varepsilon$ for all pairs $\lfloor \mathbf{u}^p, d^p \rfloor$, $p = 1, \ldots, P$ of input vectors and desired outputs available from the P training samples. To solve this problem, the simplicial neural cell (3) can be regarded as a linear adaptive neuron (Adaline) in an expanded N-dimensional feature space defined by the sparse outputs $\mu_{i_l}(\mathbf{u}) = \mu_j(\mathbf{u})$. Consequently, for problems defined by training samples the simple LMS (Widrow–Hoff) rule [14, p. 66] is employed to determine the gene parameters c_j.

Therefore, in each step p of the learning process, the coefficients c_j are updated using the following algorithm.

1) Given $\mathbf{u}^p$, compute the sequence of vertices $\mathbf{v}_{i_l}$, and nonzero membership functions μ_{i_l}, as well as the $r < n+1$ indexes i_l (see Section III).
2) Compute the effective output of the simplicial CNN cell: $y^p = \sum_{l=1}^{r} c_{i_l}\mu_{i_l}(\mathbf{u}^p)$.
3) For all indexes i_l $(l = 1, \ldots, n+1)$ update the coefficients: $c_{i_l} = c_{i_l} + \eta(d^p - y^p)\ \mu_{i_l}$.

The training process uses a test set with Q samples and stops when $\|d^q - \sum_{l=1}^{r} c_{i_l}\mu_{i_l}(\mathbf{u}^q)\| < \varepsilon$, where ε is a prescribed minimal error, and $q = 1, \ldots, Q$. At the beginning of the algorithm all parameters are initialized to zero, i.e., $c_j = 0$, $j = 1, \ldots, 2^n$. The training rate is chosen large enough to ensure fast convergence of the algorithm. Theoretical bounds for choosing an optimal value of η are given in [14].

It is important to observe that in the case of the simplicial CNN cell only $n+1$ of the N weights c_j are updated at each training step because the simplicial decomposition provides only $r \leq n+1$ nonzero membership functions $\mu_j(\mathbf{u})$. Therefore the complexity of both learning and retrieval for the simplicial neural cell is $O(n)$, as in the case of the linear perceptrons.

A. The Quantization of Gene's Coefficients

The results obtained using various training sets for either regression (nonlinear filtering) and classification problems led to the conclusion that under a proper choice of the dimension n of the input space,[5] the overall performance (i.e., the error or the misclassification rate) does not change significantly when the coefficients c_j are quantified with a small number of bits.

An interesting case is when only one bit is used for quantization. The resulting cell in this case is called a *compressed simplicial neural cell* and the possibility to learn arbitrary problems while its gene parameters are stored in a binary (digital) format is one of the most interesting features of the *simplicial neural cell*. In this case, the implementation of the simplicial neural network becomes very compact. For example, we propose the following two algorithms to perform the gene coefficients quantization:

1) Direct quantization: In this case, c_j is replaced by $c_j = \alpha + \beta\,\mathrm{sgn}(c_j - \tau)$, where α, β, and τ are parameters which should be optimized independently for a maximum of performance (i.e., minimization of misclassification error on the test samples).
2) Direct quantization followed by evolutionary programming: In this case, a better performance of the overall system is obtained. This optimization procedure begins with an initial solution given by a direct quantization as above and continues with a set of evolutionary programming techniques applied to the string of binary genes c_j until the overall performance of the system reaches a global optimum.

[5]A large dimension n is preferred because it leads to a large number of gene parameters. Consequently the information allocated to each of these parameters will necessarily be small, therefore, at the limit one bit per parameter may suffice to encode the problem learned by the neural network. Although the number of inputs is in general fixed by the nature of the problem, the input space dimension can be easily expanded by some simple linear or nonlinear transforms (see Section VI-B and VI-C).

III. The Circuit Architecture and its Mixed-Signal Implementation

The key issue in computing the output of the *simplicial neural cell* for a given input vector $\mathbf{u}$ is the evaluation of all membership functions μ_{i_l} and their associated indexes i_l as well as the vertices $\mathbf{v}_{i_l}$. Recall that the vertices can be used as a unique base in which any arbitrary $\mathbf{u}$ can be decomposed (Step 1 in the algorithm in Section II). The theoretical foundations of this decomposition are given in [3], [12], and [20]. In order to exemplify the *simplicial decomposition* process let us consider the example in Fig. 1 corresponding to $\mathbf{u} = (0.5, 0.4, 0.33)$. As we are assuming that all components satisfy: $0 \leq u_i \leq 1$ all available data should be first scaled to $[0, 1]$. The algorithm follows.

1) Pick the minimum nonzero component of $\mathbf{u}^{(1)}$, i.e., 0.33, and decompose it as: $\mathbf{u}^{(1)} = 0.33(1,1,1) + \mathbf{u}^{(2)}$, where $\mathbf{u}^{(2)} = (0.17, 0.07, 0)$.
   Consequently: $\mathbf{v}_{i_1} = (1,1,1)$, $i_1 = 111_2 = 7$, $\mu_{i_1} = \mu_7 = 0.33$.
2) Pick the minimum nonzero component of $\mathbf{u}^{(2)}$ i.e., 0.07 and decompose it as: $\mathbf{u}^{(2)} = 0.07(1,1,0) + \mathbf{u}^{(3)}$, where $\mathbf{u}^{(3)} = (0.1, 0, 0)$.
   Consequently: $\mathbf{v}_{i_2} = (1,1,0)$, $i_2 = 011_2 = 3$, $\mu_{i_2} = \mu_3 = 0.07$.
3) Pick the minimum nonzero component of $\mathbf{u}^{(3)}$ i.e., 0.1 and decompose it as: $\mathbf{u}^{(3)} = 0.1(1,0,0) + 0$, where this step marks the end (no further decomposition is possible).
   Consequently: $\mathbf{v}_{i_3} = (1,0,0)$, $i_3 = 001_2 = 1$, $\mu_{i_3} = \mu_1 = 0.1$.
4) Choose $v_{i_4} = (0,0,0)$, $i_4 = 000_2 = 0$ and compute $\mu_{i_4} = \mu_0 = 1 - (\mu_7 + \mu_3 + \mu_1) = 0.5$.

It is easy to check that the above algorithm for simplicial decomposition can be efficiently implemented using the simple mixed-signal circuit in Fig. 3 (exemplified here for $n = 3$). Using a ramp signal, the n comparators generate a sequence of

binary words corresponding to the sequence of indexes i_l. The time interval between the occurrence of two consecutive words is proportional to the membership functions μ_{i_l}. This simple and efficient circuitry for computing the nonlinear expansion given by the fuzzy membership functions μ_{i_l} leads to further simplifications in evaluating the output of the neural cell following (3).

The START pulse acts on the switches used to discharge the two capacitors C, and C_I. A new computation cycle begins by constantly charging the capacitor C with a constant current I. The computation cycle ends when the voltage V_C across C equals 1 V (assuming that all inputs are scaled within $[0,1]$ Volts). The outputs of the n comparators $Cp_1,\ldots,Cp_n$ represent a binary address word corresponding to the index i_l in the above decomposition algorithm. It is easy to verify that the code i_l is present at the outputs of the comparators for a period of time proportional to μ_{i_l}. The RAM data bus (or the output of the local logic emulating the RAM) provides the previously stored coefficient c_{i_l} in the form of a current signal. The capacitor C_I at the output acts as an integrator and is efficiently used to compute the output. In fact, the output is $y=\alpha\sum_{j\in J}c_j\mu_j$ (where α is a circuit constant), is obtained when the signal STOP activates, as exemplified in the time diagrams in Fig. 3 for the case presented in Fig. 2. Unlike most neuro-fuzzy networks, no multiplier circuit is required by our simplicial neuron.

Note that if the simplicial neural cell is used for image processing, in a fully parallel CNN implementation there is no need to implement all nine comparators per cell. Instead one cell contains only the comparator associated with the central pixel, the digital RAM (or ROM in the case of a predefined function), and the output integrator. The current source I, the capacitor C and the comparator Cp_{n+1} are *common* to the entire chip. Since dynamic RAM densities in actual commercial technologies can easily reach 10^9 bits/chip, a rough calculation indicates that with minor modifications, on the same chip one can easily integrate about 10^6 simplicial neural cells, each composed of a 512 bits memory[6] plus a few components implementing the input comparator and the output integrator.

This corresponds to transforming an obsolete dynamic random access memory (DRAM) chip into a 1024×1024 pixel resolution programmable CNN which allows for up to $2^{512}=1.3\cdot 10^{134}$ different gray level and binary image processing functions!

The implementation density of the circuit can be increased furthermore by emulating the RAM with a *nested Adaline* [6] described by (1).

A. Considerations Regarding the Implementation of the Local Boolean Logic

The mapping $c_{i_l}=C(\mathbf{v}_{i_l})$ corresponding to the RAM or other local Boolean logic device in Fig. 3 can be implemented in the following ways.

- Using a RAM (or ROM) where the address bus receives the code $\mathbf{v}_{i_l}$ and the corresponding coefficient (using a limited number of bits, say p bits) is stored at the associated address. As shown in Sections V and VI, for many practical problems the one-bit quantization ($p=1$) may suffice. In such cases, the implementation complexity is $O(2^n)$, but this result should be interpreted optimistically since the RAM technology with increased densities is readily available and represents the driving force of the VLSI industry. On the other hand, one should consider the following question: What is easiest to implement? A system where all parameters are concentrated in a single device (i.e., the $p\cdot 2^n$ one bit RAM cells) or a system such as the multilayer perceptron (MLP) where one has to build several tens or hundreds of neurons, each with its own storage devices (e.g., analog RAMs or similar) for their synaptic weights? We suggest that for many applications, particularly those leading to a small number of inputs (e.g., $n<16$) the use of a digital RAM in a simplicial neural cell architecture is the most efficient and convenient solution.
- Emulating the RAM with a compact neural-network solution. Indeed, one can train a convenient neural-network architecture with binary inputs and outputs to learn the mapping $c_{i_l}=C(\mathbf{v}_{i_l})$ instead of storing the coefficients in a RAM memory table. It is expected that a dramatic reduction in complexity will occur, particularly for the cases with large n, when a problem is defined by a number of training samples $M<<2^n$. Indeed, if we consider the fixed point representation of c_{i_l} using p bits, each of these binary outputs represents a Boolean function with n inputs (the binary code $\mathbf{v}_{i_l}$) which can be conveniently learned by a *nested* Adaline described by (1). This architecture solution is represented in Fig. 8. Therefore, the overall complexity in terms of number of devices and parameters is now only $O(n)$. And this result stands while the resulting simplicial neural network is a universal approximator, therefore being capable to learn arbitrary problems! We should consider this result with caution because each parameter of the multinested cell may require a representation resolution of up to $2^n/n$ bits per parameter [6]. At this point, the question mentioned above becomes more evident. Indeed, which device is more convenient to implement: The RAM which is a readily available device containing an array of 2^n binary cells, or the $O(n)$ complexity multinested neural cell where each of the $2n$ parameters require some storage mechanism with a resolution of $O(2^n/n)$? Put in this way, the answer to the question will tend to be: The RAM. However, there are numerous practical situations for which the emulation of the RAM with a neural network is far more convenient. Let us consider the example of a Median filter, which will be discussed further in Section V. This example demonstrates another viewpoint of the simplicial neural cell. According to this viewpoint, the functional capability of a neural network (the one emulating the RAM) is highly increased by embedding it within the additional circuitry formed

[6]For a homogeneous CNN, the RAM in each cell can be replaced by a 2^n to 1 multiplexer which has the same implementation complexity as the RAM. Apparently simpler, this solution is unpractical since it requires a 2^n lines gene bus connecting all cells. Instead, a single 1 bit data bus suffices to program all cells, when the RAM is used.

by the input comparators and the output integrator. Let us first consider the simple linear threshold gate defined by: $y = \text{sign}\left(\sum_{i=1}^{9} u_i\right)$. If we apply this LTG as a local filter for a 3×3 neighborhood in a noisy (salt and pepper type) gray-level image, the result of processing will be a black-and-white image without any meaning. However if we employ the same simple neural structure as a mapping $c_{i_l} = C(\mathbf{v}_{i_l})$ in the simplicial neural cell architecture, the result is a highly efficient Median filter which restores the original gray scale image from the corrupted one (see Section V-B). The implementation complexity of the resulting simplicial cell is just a bit larger than that of the linear threshold gate, leading to the simplest mixed-signal implementation known so far for a Median filter.

Concluding, we suggest that the solution of emulating the RAM with nested Adalines is in fact highly desirable and can lead to dramatic reduction in complexity for many real-world problems.

The method of RAM emulation is very practical for situations where a large number of inputs are required, otherwise leading to dramatic increases of the RAM sizes. In most cases, these situations correspond to a number of samples $M << 2^n$, therefore the RAMs will be inefficiently used (many locations will be never accessed). For example, consider a problem with $n = 30$ inputs and 3000 training samples. The implementation solution using a RAM will require several RAM chips of 1024 Mbits each (still not available on the market) while at most $30 \times 3000 = 90.000$ of their locations will be effectively used (assuming that in the worst case, each of the input vectors will generate a set of vertices which has a void intersection with the sets generated by any other input). It is clear that in this case, independently of how complex the problem to be learned is, the RAM emulation represents a much more reasonable solution. In most cases it might be possible that a simple linear threshold gate or a multi-nested cell (nested Adaline) with a reasonably small number of "nests" [i.e., absolute value functions in (1)] will emulate well enough the RAM so that the overall simplicial cell will have good generalization performances.

B. Software Implementations

Although the simplicial neuron architecture is tailored to the mixed signal implementation, it has also a convenient implementation in software. Particularly, the main advantage of a software implementation is the possibility to replace the simple RAM structure with a dynamically allocated one. Indeed, for a large number of inputs (e.g., $n = 30$) the circuit implementation in Fig. 3 becomes prohibitive since it requires a RAM with a capacity of 2^{30} bits. However, in such cases only a small fraction of the RAM locations are actually accessed, since for most of the practical problems the number of training samples would be $M << 2^n$. Therefore, a dynamically allocated RAM, easily to program in a software implementation, would replace the linear RAM removing the memory space problems as long as M has reasonable values. It is easy to verify that in the worst case a maximum of $M(n+1)$ memory cells would be required to store the gene.

IV. General Methodology for Training the Simplicial Neural Network

In order to use the above architecture and its circuit realization for different tasks, the following steps should be considered.

1) Prepare a set of training samples using representative signals (images) for a given task. Also prepare a different set of test signals (images) to evaluate the generalization performance of the simplicial neural cell.
2) Using the above training samples determine the gene coefficients $c_j = c_{i_l}$ in the simplicial decomposition (3). As the structure is linear with respect to the outputs μ_{i_l}, the gene coefficients can be learned in a straightforward manner, for example using the LMS algorithm.

   As a result of learning the resulting gene coefficients are *not binary*.
3) Quantify the coefficients $c_j = c_{i_l}$ using *one bit per coefficient*. Quantization ensures that each coefficient can be computed as the output of a nested Adaline (1). For example, the direct quantization scheme described above was often employed with acceptable results. The investigation of better quantization schemes will be the subject of further research. One can also decide to quantify using p bits per coefficient, leading to a more accurate representation but also requiring more devices in the cells' circuit implementation.
4) Determine the parameters of the nested Adaline emulating the RAM using one of the methods in [6] such that it can represent the Boolean function defined by the pairs of samples $(\mathbf{v}_j, c_j)_{j=1,\ldots,N}$ where $\mathbf{v}_j$ is a binary input vector and c_j is the desired output. Simple perceptron learning or the LMS algorithm can be used for the no nests (classic) Adalines, which were found good enough to solve numerous nonlinear image filtering problems when embedded within the simplicial cell circuit. Further iterations trough Steps 3) and 4) can improve the quality and the compactness of the solution. Step 4) will be removed if the choice is to implement the local Boolean logic in the simplicial neuron with a RAM. Consequently, the RAM has to be loaded with the quantified values of the gene coefficients , as they resulted from Step 3).

V. Image Processing Applications

Within the framework of the CNN, the adaptive nature of the simplical neural cell makes it suitable for image filtering and other applications where the problem is specified by samples. We will illustrate such an application in this section through several problems, emphasizing the robustness and the efficiency of our cell.

A. Square Scratch Removal

The problem of square scratch removal is defined by a set of training image samples [such as in Fig. 1 (lower row)]. Other test image samples (see Fig. 4) are used to evaluate the performance of the resulting neural filter. An ideal filter should reconstruct any other original image when the CNN input is provided with a perturbed input image. After 100 epochs

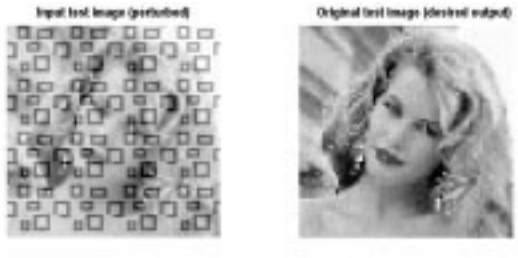
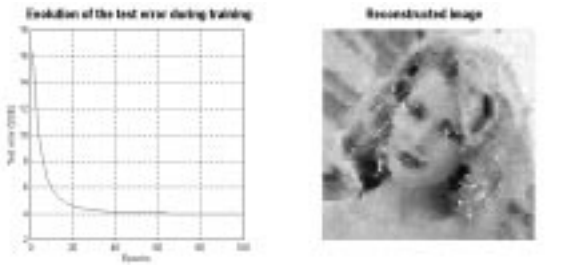
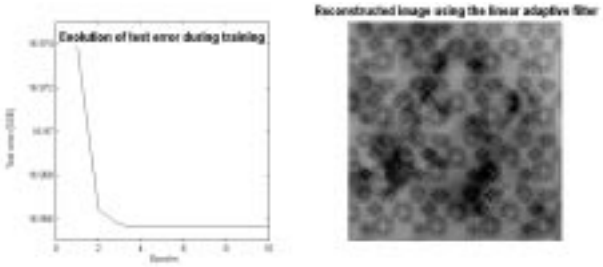


Fig. 4. Perturbed and original images used in our experiments.



(a)



(b)

Fig. 5. (a) Reconstructed image using the simplicial CNN cell. (b) The same image reconstructed with a linear (perceptron) cell trained for the same task.

of LMS training of the simplicial neural cell, a good quality reconstructed image is obtained [Fig. 5(a)]. In this case no quantization was performed after learning. For comparison, Fig. 5(b) presents the reconstructed image using the standard (linear) CNN cell with no feedback. Let us now consider the direct binary quantization of the gene (i.e., a Boolean gene) obtained after 100 epochs of learning by using the following transform: $c_j = 0.3 + 0.7\mathrm{sgn}(c_j - 0.2)$.

As shown in Fig. 6, the output image is not significantly altered when compared to the image in Fig. 5(a) obtained with real valued coefficients, and it resembles quite well the original image, thereby providing evidence that the compressed simplicial neural cell learned correctly the task of square scratch removal. Additional optimization of the gene using genetic algorithms may provide an even better result.

B. Median Filters

The problem of "salt and pepper" noise removal is usually solved by employing Median filters [15]. However, the same problem can be defined by a set of training image samples such as those presented in Fig. 1 (upper row). Such samples were used to train the simplicial neural cell for the above task. In other words, the simplicial neural cell was trained to mimic a Median

Fig. 6. Reconstructed image using the compressed simplicial CNN cell. The binary gene (the RAM content) is represented as a 16×32 matrix with black elements correpsonding to $c_j = -0.4$ and white elements corresponding to $c_j = +1$.

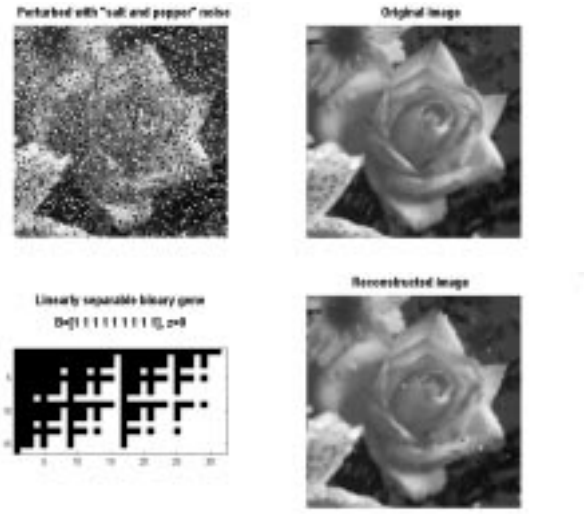


Fig. 7. A median filter has been learned by a compressed simplicial neural cells. The binary coefficients can be stored in a general purpose RAM but they can be also obtained by employing a simple linear threshold gate.

filter. The learning algorithm was followed by a quantization scheme where the goal was to find the optimal gene which can be represented by a linearly separable Boolean function. The results shown in Fig. 7 indicate a very good performance on a different data set (test set) and comparisons with results obtained by employing a median filter indicates a perfect similarity between the two filters in terms of functionality.

In addition, our simplicial neural filter leads to a much more compact implementation than that of the traditional Median filter. Indeed, the binary gene represented in Fig. 7 as a 16×32 matrix can be stored in a general purpose RAM which can be emulated using a very simple linear threshold gate defined by $c = \mathrm{sgn}(\sum_{i=1}^{n} b_i x_i + z_1)$, where $\mathbf{b} = [b_1, b_2, \ldots, b_n]$ is a vector of weights and z_1 is a threshold. For the particular case of the emulated median filter $b_i = 1, \forall\, i$ and $z_1 = 0$, leading to an implementation similar to that in Fig. 8 but with no "nest" in the *nested Adaline*. No more than 10 to 20 transistors are necessary to implement this cell. For comparison, an advanced Xilinx FPGA implementation of a cell performing median filtering requires several thousands transistors per cell [16].

It is straightforward to generalize the *simplicial neural cell* to Median filters with larger neighborhoods by simply employing a neural cell with the corresponding number n of inputs, $b_i = 1$, $i = 1, \ldots, n$ and $z_1 = 0$.

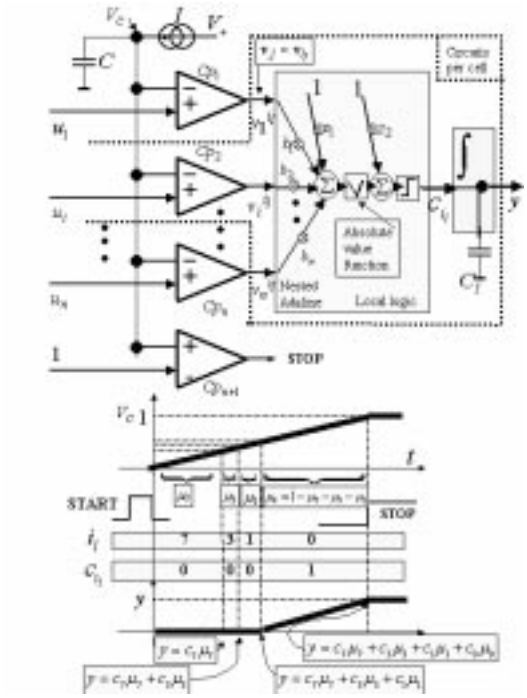


Fig. 8. The simplicial neural network with the RAM emulated by a nested Adaline. In this particular example the nested Adaline has one "nest".

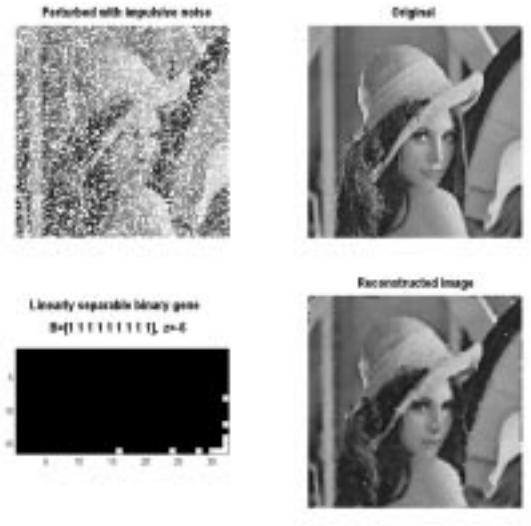


Fig. 9. The compressed simplicial neural filter trained for the task of impulsive noise removal. The underlying gene is a linearly separable Boolean function (represented here as a 16×32 matrix) and can be therefore implemented using a linear threshold gate.

C. Impulsive Noise Removal

The task of impulsive noise removal can be also learned by our simplicial neural cell, leading to a solution which does differ from the emulated median filter by only a parameter, namely $z_1 = -6$. The results of applying this filter to an image corrupted with impulsive noise with a probability of 0.25 are presented in Fig. 9.

D. Edge Detection

The previous two examples led to RAM emulations using very compact *nested Adalines* without "nests" i.e., "classic"

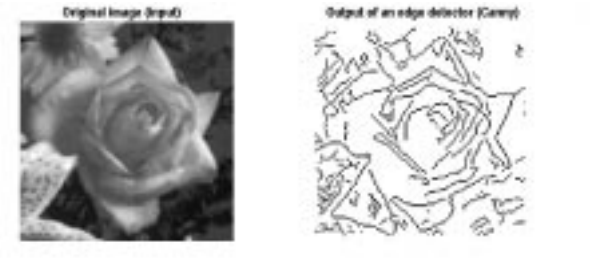


Fig. 10. Training samples for the "edge detection" task. For this task a series of 512 continuous gene coefficients c_j were determined as shown in Fig. 11 (upper row, left).

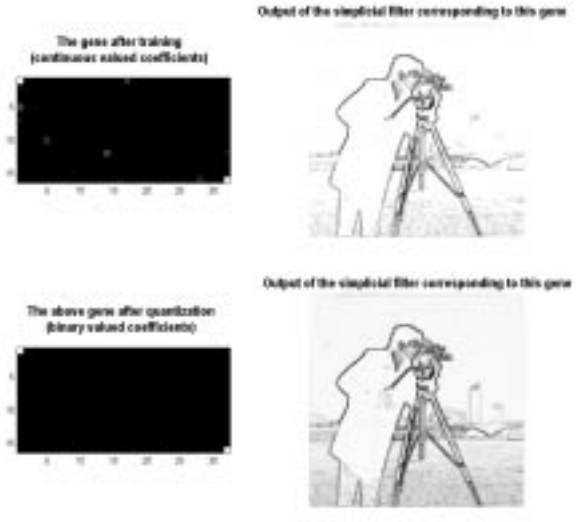


Fig. 11. Genes (left) and processed images (right) using the simplicial filter trained for the task of "edge detection." (Upper row) With continuous valued gene coefficients. (Lower row) With one bit quantified coefficients computed by a nested Adaline.

Adalines with a very simple hardware implementation. In this example we will consider the task of edge detection from a gray-level (continuous level) image. The training samples were obtained from the images in Fig. 10, where a "Canny" edge detector from the Matlab image processing toolbox was used to process the original.

After direct quantization (using $\alpha = 0$, $\beta = 1$, $\tau = 0.4$) it is seen that no major difference occur in the processed image, in fact the image obtained after quantization has better edges. A simple two nested Adaline with $\mathbf{B} = [1, 1, 1, 1, 1, 1, 1, 1, 1]$ and $\mathbf{z} = [z_1, z_2, z_3] = [0, -4, -4]$ is capable to represent the table of coefficients composing the binary gene in Fig. 11 (bottom). Observe the existence of gray levels in the output image of the simplicial nested Adaline which can be removed while emphasizing various details by an additional thresholding as seen in Fig. 12 where the original image and the image filtered with the "Canny" edge detector are simultaneously presented for comparison.

When the same filter as above is applied to the image (a rose) used initially for training, the resulting output is shown in Fig. 13. Observe that choosing a different threshold level tau for the output of the simplicial cell, i.e., $y = \mathrm{sign}(y - \tau)$, the quality of the edges can be controlled.

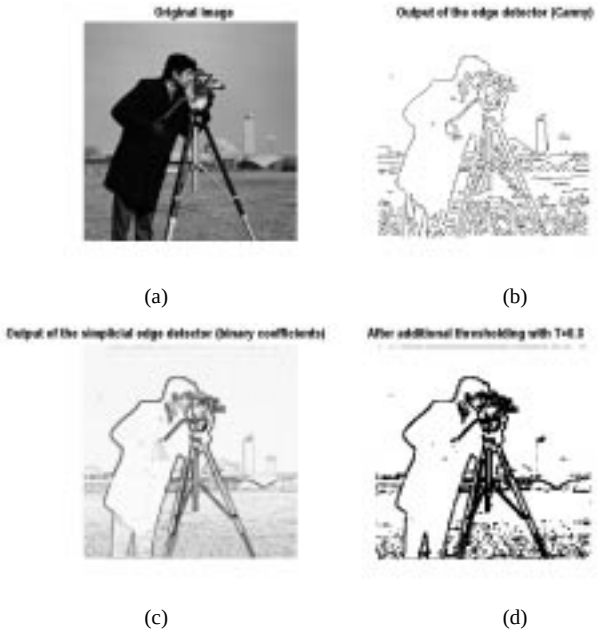


Fig. 12. (a) The original image. (b) The output of the "Canny" edge detector. (c) The output of the simplicial nested Adaline. (d) As in (3) but after additional thresholding of the output (i.e., $y = \mathrm{sign}(y - T)$).

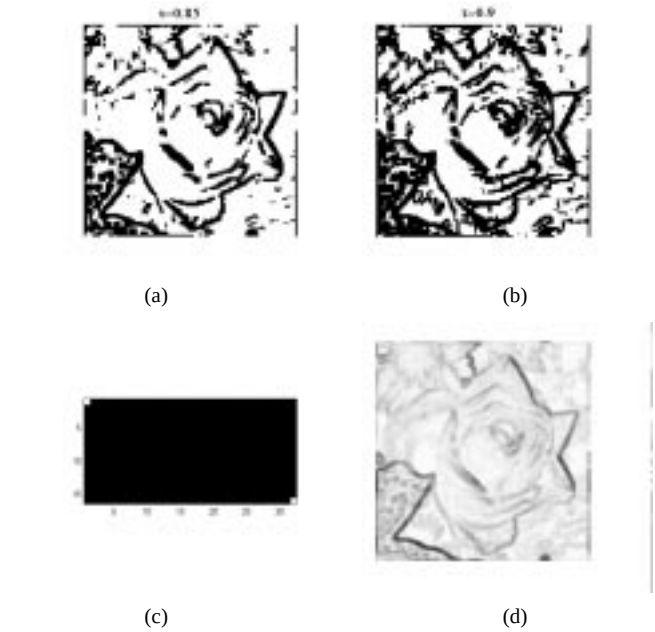


Fig. 13. (a) The filters' gene. (b) The output of the trained simplicial neuron acting as an edge detector. (c) and (d) The output of the simplicial nested Adaline but after additional thresholding of the output with two different tau values, i.e., $y = \mathrm{sign}(y - \tau)$.

VI. Applications in Pattern Classification

With the addition of a hard limiter at the output (providing a binary instead of a continuous output) the simplicial neural cell (3) can be trained for various pattern classification tasks. For problems where more than two classes are involved, a simplicial cell (except the input stage based on comparators) will be used for each output associated with a class assuming that an -1 output corresponds to a pattern which does not belong to that class while an output equal to $+1$ signify the membership of the input vector to that class. The hard limiter is introduced only during the retrieval phase, and the same learning algorithms as for the regression problems is employed. We are mainly interested in two issues.

- How can the classification performance be compared to that of other neural adaptive systems? To provide an answer, two benchmark problems were considered herein. The problems were previously learned using different neural classifiers and the comparative results are readily available in a technical report [17]. The first problem (IRIS) is a well known linearly not separable problem with four-dimensional inputs and three classes (75 samples in each the test and the training set), where the task is to recognize among three species of Iris flowers based on some features. The PHONEME problem is a difficult signal recognition problem with five-dimensional input vectors representing features extracted from a spectral analysis of the voice signal (2702 samples in each the training and the test set). The task is to recognize whether an instance represents a vowel phoneme or not, therefore the problem is a two-class problem.
- Is it possible to achieve near optimal classification performance (i.e., close to the one obtained using continuous valued coefficients c_j) using compressed simplicial neurons, i.e., cells where the coefficients c_j are represented with only one bit per coefficient?

A. The IRIS Problem

Several cases were considered.

- C1 corresponds to $n = 4$, i.e., all four inputs are applied unchanged to the cell. No quantization is employed.
- C1Q is the same as C1 but binary quantization followed by some simple evolutionary optimization of the gene was employed.
- C2 corresponds to $n = 7$, i.e., the input space was linearly extended. The input vector is formed now as $\mathbf{u} = [u_1, u_2, u_3, u_4, u_1 - u_2, u_2 - u_3, u_3 - u_4]$. No quantization was employed.
- C2Q is similar to C2, except that binary quantization followed by some simple evolutionary optimization was employed.
- C3 is similar to C2, except some more two inputs were added to form a nine-dimensional input vector: $\mathbf{u} = [u_1, u_2, u_3, u_4, u_1 - u_2, u_2 - u_3\ , u_2 - u_3, u_1 - u_4, u_2 - u_4]$.
- C3Q is similar to C3, except that binary quantization followed by some simple evolutionary optimization was employed.

In the above examples we used simple evolutionary algorithms for quantization and the gene solution is not the global optimum. We expect that the use of more advanced evolutionary algorithms will improve the classification performances for those cases where quantization was employed. Table I summarizes the results on the test data sets in all of the above six cases. The results are given for each class as percentages of misclassification.

Note that no misclassification error (0%) on the test set was obtained for the case C2. The above results also suggest that the

TABLE I
THE MISCLASSIFICATION PERCENTAGE FOR EACH CLASS AND THE SIX CASES DISCUSSED ABOVE (THE IRIS PROBLEM)

	C1	C2	C3	C1Q	C2Q	C3Q
Class 1:	0%	0%	0%	0%	0%	0%
Class 2:	4%	0%	0%	46%	2.67%	2.67%
Class 3:	0%	0%	1.33%	16%	10.6%	6.67%

technique used for artificially increasing the dimensionality of the input space may increase the classification performance, especially when quantization was employed. Note that when no quantization is employed, a too large input dimension n will usually lead to overfitting and consequently to a degradation of the generalization performance as seen for the case C3. We expect that the result in case C3Q can be further improved by employing more advanced quantization techniques. Increasing the input vector dimensionality is also expected to lead to a decrease of misclassification error to the limit of 0% demonstrated in the case of nonquantified coefficients.

B. The PHONEME Problem

In this case, the typical misclassification performance using a multilayer perceptron (MLP) is around 17% [17]. Consider the following six cases:

- C1 corresponds to $n = 5$ i.e., all five inputs are applied unchanged to the cell. There is no quantization employed.
- C1Q is the same as C1 but binary quantization followed by some simple evolutionary optimization was now employed.
- C2 corresponds to $n = 9$, i.e., the input space was linearly extended. The input vector is formed now as $\mathbf{u} = [u_1, u_2, u_3, u_4, u_5, u_1 - u_2, u_1 - u_3, u_1 - u_4, u_1 - u_5]$. There is also no quantization employed.
- C2Q is similar to C2, except that binary quantization followed by some simple evolutionary optimization was employed.
- C3 is similar to C2, except that one more input was added to form a ten-dimensional input vector: $\mathbf{u} = [u_1, u_2, u_3, u_4, u_5, u_1 - u_2, u_1 - u_3, u_1 - u_4, u_1 - u_5, u_2 - u_3]$.
- C3Q is similar to C3, except that binary quantization followed by some simple evolutionary optimization was employed.

Table II summarizes the results on the test data sets.

Although the problem is quite different in complexity and number of samples, the same remarks as for the IRIS problem stand. The tendency of the network with binary coefficients to converge to the best possible result (in our case 19.2%) if appropriate dimensionality expansion is performed over the input space, is again confirmed. As shown for the case C3Q, 1024 bits suffice to encode the knowledge associated with the PHONEME problem with a performance loss of only 1.7% compared to the case of using continuous gene coefficients.

C. Nonlinear Expansion of the Input Space

In the above examples it was shown that a linear expansion of the input space before using the simplicial neural network

TABLE II
THE MISCLASSIFICATION PERCENTAGE FOR THE PHONEME PROBLEM AND THE SIX CASES DISCUSSED ABOVE

	C1	C2	C3	C1Q	C2Q	C3Q
Class 1:	19.2%	20.13%	19.95%	29.35%	22.61%	20.91%

can significantly improve the quality of the classification, especially when combined with a one-bit quantization of the gene coefficients. However, there are situations where a linear expansion of the type considered above cannot be applied. Such situations correspond to problems characterized by a single input. For example, let us consider the case of *the simplicial neural cell* trained to learn the function $f(u_1) = \sin(u_1)$ with $u_1 \in [0, 2\pi]$.

In such cases, the preprocessing method in [18] can be applied. This method is based on a very simple nonlinear transform inspired from the multinested nonlinearity employed in (1). This method of expanding the input space assumes that instead of applying the unique input to the simplicial neural circuit (which will result in a very poor performance since only two gene parameters will be available to learn the problem) one will generate m additional inputs using the following scheme:

$$u_k = 1 - |2u_{k-1} - 1|, \quad k = 2, \ldots, m. \tag{4}$$

Typically, the performance of the simplicial neural cell improves when m is increased up to a value m_{opt}. For $m > m_{\text{opt}}$ no significant improvement in performance is observed. Using the above observation one can easily determine m_{opt} for a given problem. The same idea of nonlinear preprocessing can be also applied to problems with $n > 1$. Using this method with $m_{\text{opt}} = 3$ for the PHONEME problem, a misclassification error of only 17% was obtained, a similar result to the one reported in [17] for a multilayer perceptron, and better than what has been obtained for the simplicial neuron with linear preprocessing (Table II).

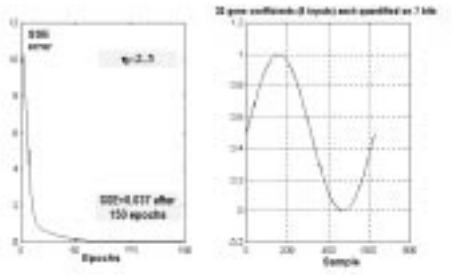
For the SIN problem considered above with $m_{\text{opt}} = 5$, Fig. 14 clearly indicates the convergence of the LMS learning algorithm toward a very good solution characterized by a very small sum of square error $(\text{SSE} = 0.037)$. The whole problem is therefore "learned" in the 32 RAM coefficients and further investigations indicate that the above approximation error is maintained even if each coefficient is quantified with at least six bits.

For comparison, the case of $m = 3$ is presented in Fig. 14(b). In this case, the SSE error is much larger, of about $\text{SSE} = 1$.

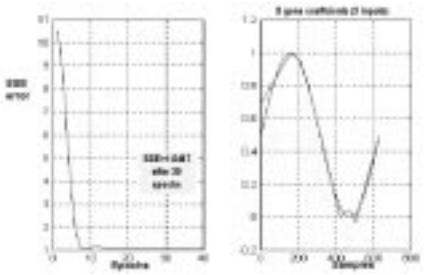
VII. HARDWARE IMPLEMENTATION EFFICIENCY

In a hardware realization of a neural system the goal is to achieve a proper functionality while minimizing the area of silicon (by reducing the number of electronic devices) and the power consumption. The easiness of programmability and reconfigurability is also an important issue.

To evaluate the efficiency of our circuit realization against other solutions, let us consider the problem of square scratch removal with a direct one bit quantization (see Section V-A) and a RAM-based implementation of the simplicial neuron. In



(a)



(b)

Fig. 14. The simplicial cell with nonlinear expansion of the input space from one to five inputs used to approximate the SIN function. (a) The decrease of the SSE during the learning process. (b) Desired and approximated functions—observe their overlapping which indicates a very good approximation. 32 coefficients in the simplicial neurons' RAM suffice to represent an input–output mapping corresponding to the 620 samples of the SIN function.

order to achieve the same level of performance a standard MLP was trained and tuned. A 9:15:1 structure results, i.e., 15 neurons on the hidden layer and a total of $150+16 = 166$ synapses. In order to achieve programmability, the mixed-signal VLSI solution in [21] can be used to implement the MLP. Since each synapse has to be programmed with five bits, a RAM of at least $166 \times 5 = 803$ bits is required to store the weights. In addition, the VLSI implementation of the MLP in [21] would require 166 synaptic circuits (each containing 24 MOS transistors) and 16 neuron units. It is obvious from Table III that the simplicial neuron require almost 63% of the RAM cells and less than 2.5% of the CMOS transistors required by the programmable MLP in [21] (assuming that ten CMOS transistors suffice to implement a comparator in the simplicial neuron).

Moreover, our solution is much more flexible since any new application may require only the loading of the RAM with a new string of 512 bits. Instead, the MLP in [21] requires additional wiring circuitry to accommodate a variable number of hidden nodes as required by a certain application. Also, while the RAM storage in the MLP is distributed in many registers of five bits each, one single compact RAM is used by the simplicial neuron thus making the silicon use more effective. Similar comparisons stand for any other of the problems considered, leading always to a much more efficient implementation for the simplicial neuron architecture. In fact, although the requirement of RAM cells is quite similar (as expected, since the same amount of knowledge should be stored), the simplicial neuron architecture leads to a dramatic reduction in the number of devices otherwise associated to the synapses and neurons of the equivalent MLP's or other type of neural network.

TABLE III
A HARDWARE EFFICIENCY COMPARISON BETWEEN THE SIMPLICIAL NEURON IMPLEMENTATION AND A MIXED-SIGNAL MLP REALIZATION IN [21]

	MLP realization [21]	Simplicial cell	Devices in the simplicial neuron (% of the MLP)
RAM bits	803	512	63%
Transistors	$\cong$ 4016	$\cong$ 102	2.5%
Flexibility and reconfigurability	Good	Very good	

VIII. CONCLUDING REMARKS

In this paper, a novel neural architecture exploiting the simplicial subdivision proposed in [3] for PWL approximation of nonlinear circuits was introduced. An efficient mixed-signal circuit implementation was proposed and its capabilities for binary and continuous nonlinear filtering tasks and classification problems were demonstrated. The complexity of the circuit does not significantly exceed the complexity of a binary RAM or its analog implementation via a nested Adaline [6] (in the latter case, with an implementation complexity of only $O(n)$), while the resulting architecture was proved highly effective in solving complex nonlinear signal processing tasks.

An interesting feature of the simplicial neural cell is that it allows a binary quantization and storage of its coefficients (also called a gene) without major loss in the performance, the result being a highly compact yet efficient structure. Although the gene is *binary* (i.e., equivalent to the gene of an arbitrary Boolean input function) *the compressed simplicial neural cell* was proved to perform various intelligent signal processing tasks including nonlinear filtering and classification, with a quality that closely resembles those obtained using much more sophisticated architectures.

Further research will focus on improving the algorithms for optimizing the binary genes for the compressed simplicial cells. Other topics of interest include the investigation of a wider palette of signal processing applications in multimedia and intelligent signal processing.

The features of the novel simplicial neural cell makes it well suited for compact and low power implementations in mixed signal technologies (e.g., the CMOS technologies used for fabricating high-density DRAMs). Its applications are in the area of portable multimedia technology, smart mobile phones, portable digital assistants, and other portable devices incorporating intelligent signal processing functions. An interesting perspective is the possibility to build high-frequency and high-density cells by implementing the simplicial neural cell in room operating nanotechnologies based on resonant tunneling diodes (RTDs). Successful realization of both RAM cells and comparators, the main components of our *simplicial neural cells*, were already reported in the literature [22]–[24].

REFERENCES

[1] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
[2] H. W. Kuhn, "Simplicial approximation of fixed points," in *Proc. Nat. Academy Sci. USA*, vol. 61, 1968, pp. 1238–1242.
[3] M. Chien and E. Kuh, "Solving nonlinear resistive networks using piecewise-linear analysis and simplicial subdivision," *IEEE Trans. Circits Syst. I*, vol. CAS-24, pp. 305–317, June 1977.

[4] L. O. Chua and S. M. Kang, "Section-wise piecewise-linear functions: Canonical representation, properties, and applications," *Proc. IEEE*, vol. 65, no. 6, pp. 915–929, 1977.
[5] L. O. Chua, *CNN: A Paradigm for Complexity*. Singapore: World Scientific, 1998.
[6] R. Dogaru and L. O. Chua, "Universal CNN cells," *Int. J. Bifurcation Chaos*, vol. 9, pp. 1–48, Jan. 1999.
[7] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Englewood Cliffs: Prentice-Hall, 1985.
[8] R. Dogaru, L. O. Chua, and K. R. Crounse, "Pyramidal cells: A novel class of adaptive coupling cells and their applications for cellular neural networks," *IEEE Trans. Circuits Syst. I*, vol. 45, pp. 1077–1090, Oct. 1998.
[9] R. Dogaru, L. O. Chua, and M. Haenggi, "A compact universal cellular neural network cell based on resonant tunnelling diodes: Circuit design, model and functional capabilities," in *Proc. Int. Workshop Cellular Neural Networks Applicat.*, May 2000, pp. 183–188.
[10] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 326–334, June 1965.
[11] R. Dogaru, P. Julián, and L. O. Chua, "A robust and efficient universal CNN cell circuit using simplicial neuro-fuzzy inferences for fast image processing," in *Proc. ISCAS 2001 (IEEE Symp. Circuits Syst.)*, Sydney, Australia, May 2001, pp. 493–496.
[12] P. Julián, R. Dogaru, and L. O. Chua, "A piecewise-linear simplicial coupling cell for CNN gray-level image processing," in *Proc. ISCAS 2001 (IEEE Symp. Circuits Syst.)*, Sydney, Australia, May 2001, pp. 109–112.
[13] C.-T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Upper Saddle River, NJ: Prentice-Hall, 1996.
[14] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1995.
[15] K. R. Castleman, *Digital Image Processing*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1996.
[16] R. Maheshwari, S. P. Rao, and E. G. Poonach. FPGA implementation of median filter. presented at Proc. 10th Int. Conf. VLSI Design: VLSI in Multimedia Applications. [Online]. Available: http://www.computer.org/proceedings/vlsid/7755/77 550 523.pdf
[17] "Enhanced Learning for Evolutive Neural Architecture," Databases, Benchmarks, ftp://ftp.dice.ucl.ac.be/pub/neural-nets/ELENA/, 1995.
[18] R. Dogaru, M. Alangiu, M. Rychetsky, and M. Glesner, "Perceptrons revisited: The addition of a nonmonotone recursion greatly enhances their representation and classification properties," in *Proc. IJCNN'99 (Int. Joint Conf. Neural Networks*, Washington, DC, July 10–16, 1999, pp. 862–867.
[19] P. Julián, A. Desages, and O. Agamennoni, "High level canonical piecewise linear representation using a simplicial partition," *IEEE Trans. Circuits Syst. I*, vol. 46, pp. 463–480, Apr. 1999.
[20] P. Julián, A. Desages, and B. D'Amico, "Orthonormal high level canonical piecewise linear functions with applications to model reduction," *IEEE Trans. Circuits Syst. I*, vol. 47, pp. 702–712, May 2000.
[21] G. C. Cardarilli, C. D'Alessandro, P. Marinucci, and F. Bordoni, "VLSI implementation of a modular and programable neural architecture," in *Proc. MicroNeuro'94 4th Int. Conf. Microelectron. Neural Networks Fuzzy Syst.*, Turin, Italy, Sept. 1994, pp. 218–225.
[22] J. P. A. van der Wagt, "Tunneling based SRAM," *Proc. IEEE*, vol. 87, no. 4, pp. 571–595, Apr. 1999.
[23] A. Seabaugh, B. Brar, T. Broekaert, F. Morris, P. van der Wagt, and G. Frazier, "Resonant-tunneling mixed-signal circuit technology," *Solid-State Electron.*, vol. 43d, pp. 1355–1365, 1999.
[24] J. P. A. van der Wagt, "Tunnelling-based SRAM," *Nanotechnol.*, vol. 10, pp. 174–186, 1999.

Radu Dogaru (S'95–M'98) was born in Constantza, Romania, in 1963. He received the M.S. and Ph.D. degrees in electrical engineering both from the Polytechnic University of Bucharest, Bucharest, Romania, in 1987 and 1996, respectively. He received the M.S. degree from the same university in computer aided circuits design in 1994, concluding a two-year postgraduate school organized under the European TEMPUS program.

In 1990, he joined the Department of Applied Electronics and Information Engineering of the Polytechnic University of Bucharest, Romania as a Teaching Assistant. He became an Assistant Professor in 1996, and Associate Professor in 1999, and Professor in 2001. He was a Fulbright Visiting Scholar at the University of California at Berkeley, Berkeley, the Nonlinear Electronics Laboratory from 1996 to 1998. From 1999 to 2000, he visited the same laboratory as a Research Scholar. He did several research stages at I.N.P., Grenoble, France, in 1994 and at the Institute of Microelectronic Systems, the Technical University of Darmstadt, Darmstadt, Germany, in 1995, 1996, 1998, and 2000. With the latter team, he recently started a joint research program funded by a Volkswagen Stiftung grant. His scientific interests include nonlinear and intelligent systems, bio-inspired computing architectures, nonlinear signal processing, complex adaptive systems, cellular neural networks, emergent computation, and neural architectures for micro and nanotechnologies.

Dr. Dogaru was a corecipient of the Romanian Academy of Sciences "Tudor Tanasescu" Award in 1997.

Pedro Julian was born in Bahia Blanca, Argentina, on June 28, 1970. He received the Ingeniero Electronico degree in 1994 and the Ph.D. degree in Control de Sistemas in 1999, both from the Universidad Nacional del Sur.

From 2000 to 2002, he was a Postdoctoral Fellow in the Nonlinear Electronics Laboratory of the University of California at Berkeley, Berkeley. Currently, he is a Postdoctoral Fellow at the Johns Hopkins University, Baltimore, MD. His research interests include system and circuit theory and design, including numerical algorithms.

Dr. Julian received a five-year fellowship for doctoral studies and a two-year fellowship for postdoctoral studies, both from CONICET.

Leon O. Chua (S'60–M'62–SM'70–F'74–LF'02) received the S.M. degree from the Massachusetts Institute of Technology (MIT), Cambridge, in 1961 and the Ph.D. degree from the University of Illinois, Urbana, in 1964.

He is presently a Professor of Electrical Engineering and Computer Sciences at the University of California at Berkeley, Berkeley. He has been a Consultant to various electronic industries in the areas of nonlinear network analysis, modeling, and computer-aided design. He is the author of *Introduction to Nonlinear Network Theory* (New York: McGraw-Hill, 1969), and a coauthor of the books *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques* (Englewood Cliffs, NJ: Prentice-Hall, 1975), *Linear and Nonlinear Circuits* (New York: McGraw-Hill, 1987), and *Practical Numerical Algorithms for Chaotic Systems* (New York: Springer-Verlag, 1989). He has published many research papers in the area of nonlinear networks and systems. His research interests include general nonlinear network and system theory.

Dr. Chua served as Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS from 1973 to 1975 and as President of the IEEE Society on Circuits and Systems in 1976. He is presently the Editor of the *International Journal of Bifurcation and Chaos* and a Deputy Editor of the *International Journal of Circuit Theory and Applications*. He is also the recipient of many awards, including the 1967 IEEE Browder J. Thompson Memorial Prize Award, the 1973 IEEE W.R.G. Baker Prize Award, the 1974 Frederick Emmons Terman Award, the 1976 Miller Research Professorship from the Miller Institute, the 1982 Senior Visiting Fellowship at Cambridge University, England, the 1982/1983 Alexander von Humboldt Senior U.S. Scientist Award at the Technical University of Munich, W. Germany, the 1983/1984 Visiting U.S. Scientist Award at Waseda University, Tokyo, from the Japan Society for Promotion of Science, the IEEE Centennial Medal in 1985, the 1985 Myril B. Reed Best Paper Prize, both the 1985 and 1989 IEEE Guillemin-Cauer Prizes, the Professor Invite International Award at the University of Paris-Sud from the French Ministry of Education in Fall 1986, and the 1993 Technical Achievement Award by the IEEE Circuits and Systems Society. He received the IEEE 1995 Van Valkenburg Award. He was also awarded a Doctor Honoris Causa degree from the Ecole Polytechnique Federale-Lausanne, Switzerland, in 1983, an Honorary Doctorate from the University of Tokushima, Japan, in 1984, an Honorary Doctorate at the Technical University of Dresden, Germany, in 1992, a Doctor Honoris Causa degree from the University of Santiago de Compostela, Spain, in 1995, and a Doctor Honoris Causa degree from the University of Frankfurt, Germany, in 1996.

Manfred Glesner (M'93–SM'99–F'00) received the diploma degree from Saarland University, Saarbrucken, Germany, in 1969, and the Ph.D. degree from the same university in 1975. His doctoral research was in the application of nonlinear optimization techniques in computer-aided design of electronic circuits.

From 1975 to 1981, he was a Lecturer at Saarland University, in the areas of electonic CAD and control. In 1981, he was appointed Associate Professor for electrical engineering at Darmstadt University of Technology, Darmstadt, Germany. In 1989, he was appointed Full Professor for microelectronic system design. His current interests include advanced design tools for microelectronic circuits, VLSI digital signal processing, and innovative system applications of microelectronics.

Dr. Glesner is a member of several technical societies and he is active in organizing international conferences. With the EU-based TEMPUS initiative, he built up several microelectronic design centers in Eastern Europe. He received two doctoral degrees (honoris causa) from Tallinn Technical University, Estonia, and Bucharest Polytechnical University, Romania.