RECTIFICATION NEURAL NETWORKS: A NOVEL ADAPTIVE ARCHITECTURE AND ITS APPLICATION FOR IMPLEMENTING THE LOCAL LOGIC OF CELLULAR NEURAL NETWORKS

by

Radu Dogaru and Leon O. Chua

Memorandum No. UCB/ERL M98/4

15 January 1998

COVER PAGE

RECTIFICATION NEURAL NETWORKS: A NOVEL ADAPTIVE ARCHITECTURE AND ITS APPLICATION FOR IMPLEMENTING THE LOCAL LOGIC OF CELLULAR NEURAL NETWORKS

by

Radu Dogaru and Leon O. Chua

Memorandum No. UCB/ERL M98/4

15 January 1998

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Rectification Neural Networks: A Novel Adaptive Architecture and its Application for Implementing the Local Logic of Cellular Neural Networks

Radu Dogaru and Leon. O. Chua
Electronic Research Laboratory and
Department of Electrical Engineering and Computer Sciences,
University of California Berkeley,
258 M Cory Hall, Berkeley, CA-94720 1770
E-mail: radu_d@fred.eecs.berkeley.edu

Abstract

This paper introduces Rectification Neural Networks (RNNs), a novel adaptive structure composed of layers of simple linear perceptrons connected through simple rectification devices. The learning algorithm is constructive, i.e. it adds new synaptic and rectification units until the specific problem is learned. To demonstrate the capabilities of the novel architecture, the design of local logic for Cellular Neural Networks was considered as a particular application. It is proved that any Boolean function with any number of inputs can be implemented using the RNN. The RNN in the form presented in this paper can be used as an efficient and fast design alternative to the classic digital systems. However, the RNN architecture is not restricted to binary inputs and it can cover the area of classification problems as well.

1. Introduction

Rectification, one of the basic non-linear phenomena, is observed in different physical mediums and has also found a lot of applications, particularly in the field of electronics. Among other nonlinear operators, including the widely used sigmoid, the rectification operator, mathematically denoted as $r1(x) = (x + |x|)/2$ or as $r2(x) = |x|$ has the simplest implementation in either analog or digital technologies. For example, a simple PN junction or a CMOS transistor having its gate connected with the source can act as a current rectifier, thus implementing the *r1* operator. In digital systems, *r2* rectification is obtained by simply ignoring the sign bit (assuming the use of a sign plus magnitude code). It is thus natural to consider such non-linear functions from the perspective of designing adaptive neural networks with convenient implementation. However, little or no interest was shown for such structures in the artificial neural networks community; most of the "classic" neural network architectures [7] employ smooth nonlinearities, such as the sigmoid. The reason may be related to the differentiability of the absolute value function, such functions being regarded by many scientists as being not well-suited, from the perspective of gradient search. On the other hand, in [2] it was shown that an elegant formula, which is essentially based on the absolute value function, can be used as a canonical representation for any piece-wise linear (PWL) function of one variable given through a set of input-output pairs. Successive improvements of this technique were published in [3],[8]. However, it is difficult if not impossible to compute analytically the coefficients for such structures, particularly in the case of a large number of inputs. The only adaptive versions of the PWL-canonical representation reported to date [9],[10] are restricted to a particular class of problems, i.e. signal processing, and they deal with relatively simple PWL structures. In this paper, we introduce a novel approach for designing adaptive structures based on rectification, namely the RNN structure. Its basic building block is a single-nested, piece-wise linear adaptive structure (here referred to as *PWL2cell*), for which the gradient descent was employed simultaneously with an annealing scheme. This cell and its training algorithm are described in section 2. The PWL2cell has excellent capabilities, being proved that it can implement any arbitrary Boolean function with up to 4 inputs and a significantly large number of Boolean functions with more than 4 inputs. Among them, a 4 to 1 multiplexer, which is the key element in building the RNN structure is introduced in section 3. The RNN architecture is able to adjust its complexity to the complexity of the particular function to be implemented. Moreover, the existence of the solution is guaranteed and the learning time was found smaller when compared with other solutions recently developed for the design of local logic [5],[14]. Several such applications are presented in Section 4. Among them, the RNN implementation of the Conway's "Life" Boolean function [1], responsible for producing a universally computational medium in a CNN [6] was found to be the simplest reported up to date.

2. The single-nested PWL adaptive cell (PWL2cell)

In this section, the structure and the learning law for a novel adaptive module, called a PWL2 cell, are presented. The PWL2cell is the basic building block which allows for the easy construction of a RNN structure able to implement any desired Boolean function with any number of inputs.

In what follows it is convenient to consider a differentiable approximation of the standard rectification unit: $abs(x,\varepsilon) = \sqrt{x^2+\varepsilon}$, $\varepsilon > 0$. Indeed, it is obvious that $\lim_{\varepsilon\to 0} abs(x,\varepsilon) = |x|$. For simplicity of writing, we will alternatively use the notation $\langle x \rangle = abs(x,\varepsilon)$. The approximated rectifier $abs(x,\varepsilon)$ offers two advantages:

a) It is differentiable at the origin and its derivative is $\frac{d}{dx} abs(x,\varepsilon) = sign(x,\varepsilon) = x/\sqrt{x^2+\varepsilon}$.

b) It allows the employment of an "annealing" scheme during learning. The use of this scheme allows for a faster convergence towards the global optimum. According to this scheme, at the beginning of the learning process, a relatively large value for ε is considered (usually $\varepsilon(0) = 0.2$). After each training epoch, the value of ε, which may be regarded as having the meaning of a temperature, will be decreased in such a way that at the end of the learning process it will be close to 0. For example, the following simple annealing schedule was found to be efficient: $\varepsilon(t) = \varepsilon(0)\cdot[1 - t/(T+1)]$, where t is the index of the actual training epoch and T is the number of epochs which should be fixed at the beginning of the learning process.

2.1. The structure of the PWL2cell

The following expressions describes the structure of a PWL2cell for an n- dimensional input vector $\mathbf{u} = [u_1, u_2, .. u_n]$:

$$pwl2cell(u_1,..u_n) = \mathbf{sgn}(pwl2(u_1, u_2,..u_n, \mathbf{W})), \tag{1}$$

where,

$$\begin{aligned} pwl2(\mathbf{u},\mathbf{W}) = prc_1(\mathbf{u},\mathbf{w}^0) - 2\langle prc_{10}(\mathbf{u},\mathbf{w}^{10}) - 4\langle prc_{100}(\mathbf{u},\mathbf{w}^{100})\rangle + 4\langle prc_{101}(\mathbf{u},\mathbf{w}^{101})\rangle\rangle + \\ + 2\langle prc_{11}(\mathbf{u},\mathbf{w}^{11}) - 4\langle prc_{110}(\mathbf{u},\mathbf{w}^{110})\rangle + 4\langle prc_{111}(\mathbf{u},\mathbf{w}^{111})\rangle\rangle \end{aligned} \tag{2}$$

and where

$$prc_j(\mathbf{u},\mathbf{w}^j) = w_0^j + \sum_{i=1}^{n} u_i w_i^j \tag{3}$$

is the defining expression of a linear perceptron.
The expression (1) is employed during the retrieval process while (2) it is employed during the training phase, following the idea of training the Adaline [13]. Observe, that structurally, the PWL2cell is composed by at most 7 perceptrons, each of them defined by a set of $n+1$ synaptic weights, where n is the number of inputs. The binary index of each perceptron indicates its position within the whole structure. For convenience, in what follows, we will use the decimal instead of the binary notation for indices; For example $prc_3(.)$ is equivalent to $prc_{11}(.)$. We should stress that the structure described by (2) is maximal. Indeed, one can consider that up to six of the perceptrons have 0-valued weights (i.e. they are removed). For some particular problem, the performance may be the same with a reduced number of units than with the whole set of 7 units. This is why, in training the PWL2cell, the following strategy is implemented:

1. Consider only prc_1 ; Train the PWL2cell (in this case a linear perceptron)
2. If the performance is OK, STOP; Else train again the PWL2cell with prc_2 and prc_3 added.
3. If the performance is OK, STOP; Else train again PWL2cell with prc_4, prc_5, prc_6, prc_7 added.

Hence, stopped after step 1, the PWL2cell is a linear structure, after step 2 a simple (not nested) PWL structure and after step 3, a single nested PWL structure. In what follows, the learning will be considered for the most general case (step 3) but the same algorithm can be applied for any of the other steps by simply removing the lines associated with updating the weights of perceptrons that are not considered as belonging to the structure.

2.2. The training algorithm of the PWL2cell

In the case of batch training, for the SSE error criterion one should minimize the goal function $E(t)=\frac{1}{2}\sum_{q=1}^{ns}\left[d^q - pwl2(\mathbf{u}^q, \mathbf{W(t)})\right]^2$ during each training epoch t. In this expression, ns is the number of available training patterns, q is the pattern index and d^q is the desired output associated with pattern q. According to the principle of gradient descent, and since (2) is a differentiable function, one can now derive the weight update rules for each of the perceptrons. Let us abbreviate with u_i^q the input i associated with the pattern q and $e^q = d^q - pwl2(\mathbf{u}^q, \mathbf{w(t)})$ the actual output error obtained during the presentation of the pair input-desired output associated with pattern q. The following update rules were derived:

Percepton prc_1: $\Delta w_i^1 = \overline{\eta}\cdot\sum_{q=1}^{ns} e^q u_i^q$ (4)

Perceptrons prc_2 and prc_3: $\Delta w_i^2 = -\overline{\eta}\sum_{q=1}^{ns} e^q u_i^q sign\left(prc_2(\mathbf{u}^q, \mathbf{w}^2), \varepsilon\right)$ (5)

$$\Delta w_i^3 = \overline{\eta}\sum_{q=1}^{ns} e^q u_i^q sign\left(prc_3(\mathbf{u}^q, \mathbf{w}^3), \varepsilon\right) \tag{6}$$

Perceptrons $prc_4, prc_5, prc_6, prc_7$:

$$s^2 = sign\left(prc_2(\mathbf{u}^q, \mathbf{w}^2) - 4\langle prc_4(\mathbf{u}^q, \mathbf{w}^4)\rangle + 4\langle prc_5(\mathbf{u}^q, \mathbf{w}^5)\rangle, \varepsilon\right) \tag{7}$$

$$\Delta w_i^4 = \overline{\eta}\sum_{q=1}^{ns} e^q u_i^q s^2 sign\left(prc_4(\mathbf{u}^q, \mathbf{w}^4), \varepsilon\right) \tag{8}$$

$$\Delta w_i^5 = -\overline{\eta}\sum_{q=1}^{ns} e^q u_i^q s^2 sign\left(prc_5(\mathbf{u}^q, \mathbf{w}^5), \varepsilon\right) \tag{9}$$

$$s^3 = sign\left(prc_3(\mathbf{u}^q, \mathbf{w}^3) - 4\langle prc_6(\mathbf{u}^q, \mathbf{w}^6)\rangle + 4\langle prc_7(\mathbf{u}^q, \mathbf{w}^7)\rangle, \varepsilon\right) \tag{10}$$

$$\Delta w_i^6 = -\overline{\eta}\sum_{q=1}^{ns} e^q u_i^q s^3 sign\left(prc_6(\mathbf{u}^q, \mathbf{w}^6), \varepsilon\right) \tag{11}$$

$$\Delta w_i^7 = \overline{\eta}\sum_{q=1}^{ns} e^q u_i^q s^3 sign\left(prc_7(\mathbf{u}^q, \mathbf{w}^7), \varepsilon\right) \tag{12}$$

where $\overline{\eta} = \eta / ns$

The inputs associated with the bias weights are considered constant, $u_0^q = 1$. The following ranges of the training parameters, were found by simulations as the best choices in order to provide convergence towards a near global optimum in the parameter space: Linear structure: $\eta \in [0.3, 0.7]$ and $T \in [10, 100]$; Simple (not nested) PWL2cell: $\eta = [0.1, 0.3]$, $T \in [100, 500]$; Single nested PWL2cell: $\eta \in [0.02, 0.05]$, $T \in [500, 1000]$.

The initialization of the weights is done with random small values (a magnitude of 0.001 was considered in practice).

2.3. Pruning and optimization of weights

Additional pruning and optimization of the resulting set of weight parameters can be done, using the method described below. This method is essentially based on exploiting the hard non-linearity of the sign function in (1) so that the resulting weights can be represented as integers and some of them will become 0. In order to have a robust solution (i.e. to accept tolerances for weights) one has also to check that $\min\limits_{q=0,..ns-1}\left[abs(pwl2(\mathbf{u}^q,\mathbf{W}))\right] > tol$ where tol is an arbitrary tolerance threshold. In practice, the value of tol is imposed by the implementation technology. The larger the tol the better the robustness, but at the expense of a larger variation domain for the weights. The following algorithm ensures the generation of a robust PWL2cell realization with integer weight parameters:

```
1. res=1 ;
2 Compute a quantized matrix of weight parameters: $qw_i^p = \lfloor w_i^p \cdot res \rfloor$ [1], $i = 0,..n; p = 1,..7$
3. Check if the performances of the PWL2cell with quantified parameters changed;
    3.1. If YES, then res=res+1 ; go to step 2 ;
     ELSE , then check the robustness criterion: $\min\limits_{q=0,..ns-1}\left[abs(pwl2(\mathbf{u}^q,\mathbf{QW}))\right] > tol$
        3.1.1. If YES, then STOP; $\mathbf{QW} = \begin{bmatrix} qw_0^1 & .. & qw_n^1 \\ .. & .. & .. \\ qw_0^7 & .. & qw_n^7 \end{bmatrix}$ represents a good set of
               parameters ensuring both functionality and robustness as well as
               the simplicity of implementing integer synaptic coefficients.
               ELSE, then res=res+1; go to step 2.
```

2.4. Realization of Boolean logic functions with the adaptive PWL2cell

When $ns = 2^n$, the defining truth table of a Boolean function can be considered as training set. The specific Boolean function is now encoded by the binary vector: $\mathbf{d} = \left[d^0,...d^q,..d^{ns-1}\right]$ while the set of input patterns is unchanged for any Boolean function. In this case, any input vector $\mathbf{u}^q$ is the binary representation of q; For example, if $q=13$, for $n=4$ inputs, $\overline{13_{10}} = \overline{1101_2}$ and thus $\mathbf{u}^{13} = [1,1,-1,1]$. Observe that, our convention for representing binary numbers is bipolar , i.e. $d^q \in \{-1,1\}$ as well as any of the inputs.

In order to test the capability of the PWL2cell to implement Boolean functions for different number of inputs, a set of 1000 arbitrary Boolean functions were randomly

[1] $\lfloor x \rfloor$ denotes here the rounding operator, i.e. the result is the closest integer to the real value x.

generated and for each of them the PWL2cell was trained. For the case *n=3* , the entire set of 256 Boolean functions was considered. The percentage of Boolean functions correctly learned is presented in Table 1 for each of the following three cases: (PWL0) Linear perceptron (all perceptrons except 1, removed); (PWL1) no nesting (perceptrons 4-7 removed); (PWL2) The maximal structure of PWL2cell.

Table 1: Percentage of correctly learned Boolean functions

inputs	PWL0	PWL1	PWL2
3	40.62%	100%	100%
4	2.87%	80.2%	100%
5	less than 0.1%	13.8%	88%
6	less than 0.1%	less than 0.1%	2%

It is important to observe that the complete PWL2 structure is able to implement any arbitrary Boolean function with less than 5 inputs. Moreover, if the Boolean function has only 3 inputs, it was found that a reduced PWL1 structure having only one absolute value term (and thus only two perceptrons), can represent 254 of the 256 possible Boolean functions. The only 2 exceptions, requiring the maximal PWL1 structure with 3 perceptrons, were represented by the "parity" function and its negate. Even if, apparently, for larger numbers of inputs (e.g. *n=6*) the PWL2cell seems to be inefficient, it was found that the probability to represent a meaningful Boolean function is considerably higher than the probability to represent arbitrary, randomly chosen Boolean functions. Here, by meaningful we mean Boolean functions which are the result of a human process of thinking or arise from modelling some real world processes. Such examples include: the PARITY function; the MUX41 function and the LIFE function. As it will be shown in section 4, all of them were found to have very simple PWL1 or PWL2 implementations even for numbers of inputs larger than 5. In what follows, the PWL2 implementation of MUX41 will be considered since this is a basic building block for the RNN.

The defining table of the MUX41 function is presented in Table 2: This Boolean function has 6 inputs (u0, u1, u2, u3, c1 and c0) but it can be better specified as a 4 inputs - to - 1 output multiplexer. From this perspective, its output copies one of the four inputs u0, u1, u2 or u3 depending on the particular configuration of the two control inputs *c0* and *c1* .

Table 2: The definition of MUX41

output	u0	u1	u2	u3
c1	-1	-1	1	1
c0	-1	1	-1	1

The PWL2cell structure which implements the MUX41 Boolean function is presented in Table 3.

Table 3: The realization of MUX41 with a PWL2cell

perceptron	bias	c0	c1	u0	u1	u2	u3
w^1	2	-1	3	5	5	5	5
w^2	-8	0	13	-3	3	-3	3
w^3	0	-12	0	-3	-3	3	3
w^5	0	3	0	-1	1	1	-1

Observe that only 4 perceptrons (23 non zero synapses) and 3 rectification units are needed. For comparison, the representation of the same function with [14] requires 8 perceptrons (55 non zero synapses) and 7 2-inputs XOR operators. The method presented in [5] generates a structure with 9 perceptrons and 8 logic operators with 2 inputs. According to Table 3 and the definition (1) of the PWL2cell, one can express the multiplexer function as a simple piece-wise linear expression which can be, for example, implemented with a few lines in MATLAB:

```
function y=mux41(c1,c0,u0,u1,u2,u3)
prc1=2-c0+3*c1+5*(u0+u1+u2+u3); prc2=-8+13*c1+3*(-u0+u1-u2+u3);    (13)
prc3=-12*c0+3*(-u0-u1+u2+u3); prc5=3*c0-u0+u1+u2-u3;
y=sign(prc1-2*abs(prc2+4*abs(prc5))+2*abs(prc3));
```

Using current mode techniques [11], this expression can be implemented in CMOS technology with less transistors than required by its digital counterpart (i.e. where MUX41 is implemented as a combination of basic digital gates).

3. Rectification Neural Networks

Using the PWL2cell together with its particular case, the MUX41 cell, a rectification neural network structure able to implement any given Boolean function with any number of inputs can be constructed: The algorithm is simple and it is based on the "divide and conquer" principle which can be applied to Boolean functions using the multiplexer MUX41;

1. Train a PWL2cell to solve the given Boolean function with *n* inputs (called *B* for convenience)

2. If training ended OK then STOP;
   Else, consider the two most significant input bits in the truth table defining the function *B* as the control inputs of a MUX41 cell, and the output of the MUX41 cell as the output associated with the function *B*. Thus, each of the 4 "*u*" inputs of the MUX41 cell will correspond to one of the 4 sub-functions B_0, B_1, B_2, B_3 with *n-2* inputs each;
   For example, $B = \underbrace{1\,0}_{B_0}\underbrace{1\,1}_{B_1}\underbrace{0\,1}_{B_2}\underbrace{0\,0}_{B_3}$ is a Boolean function with 3 inputs, decomposed into 4 functions B_0, B_1, B_2, B_3 with 1 input, using the MUX41 cell.

3. For each of the sub-functions, repeat the procedure starting with Step 1.

Observe, that by successive division into 4 sub-functions with the number of inputs reduced by 2, even if the initial Boolean function is too complex to be represented by a PWL2cell structure (the learning fails), its implementation complexity is decreased until the point where it can be represented by a PWL2cell. Here we should recall the results presented in Table 3 showing that statistically any function with 4 inputs has a PWL2cell realization and that by simple enumeration, the whole set of 256 Boolean functions was found to be realizable with a PWL1 structure. Hence it is obvious that the complexity of the RNN structure adapts to the complexity of the Boolean functions to be represented and any arbitrary function with any number of inputs has a realization with a RNN structure. It was found for example, that some useful local logic functions for Cellular Neural Networks (CNN) [4] can be implemented with a very low complexity as the RNN structure. For example, the function associated with the Conway's game of life [1] can be represented with a single reduced PWL2cell with only two perceptrons, even if it has 9 inputs (see section 4).

4. Applications of the RNNs.

The RNN network can be viewed as an alternative method for digital design and implementation. The design using the RNN is quite simple, in fact completely automated, since it consists only in applying the RNN construction algorithm and the training algorithm in each phase where a PWL2cell is required. Moreover, it produces an optimal structure, adapted to the complexity of the Boolean function to be implemented. In some sense, the result of this design is similar to the result of the much more elaborate process of generating optimal digital structures using Veitch-Karnaugh diagrams or other similar methods. However, instead of the basic logic gates, the building blocks of the RNN structures are rectifiers and modules which are able to compute weighted sums. Such modules can be easily implemented in cheap CMOS technology using the current mode approach. This allows for a significant reduction of the hardware requirements, particularly for those functions with many inputs which are in fact not so complex. Most of the local Boolean functions of CNNs describing specific image processing tasks fall in this category, so the RNN method can be considered a valuable alternative to methods proposed in [5] and [14] since the RNN structure can be easily be mapped into the CNN-Universal Machine [15] framework. Moreover, since the inputs of the RNN structure are *not* restricted to binary inputs, specific tasks for neural networks such as pattern classification can be also considered. In fact, preliminary tests with the PWL2cell showed that it can solve most of the classification benchmarks with performances which are close to the best performances reported in [12]. However, the extension to more complex, RNN-like structures in order to increase the performance, cannot be done in this case using MUX41 cells but rather considering additional degrees of nesting. In the end, let us consider several examples of CNN local logic Boolean function design and implementation, using the concepts presented above.

1. The PARITY function; It can be easily checked that the function $y_{par2} = 1 - |u_0 + u_1|$ implements the truth table of the XOR function (where 0 was replaced by -1). This result was found by training the PWL2cell; what is appealing in this case is the fact that the output function is implemented without the need of the hard

limiter sgn(x). Thus, such a piece-wise linear formula for implementing a parity function with any number of inputs can be easily derived; For example, the parity function with 3 inputs can be represented as: $y_{par4} = 1 - \left| u_0 + 1 - |u_1 + u_2| \right|$. Another interesting aspect is that the implementation of the 2-input PARITY function, considered from the analog input-output perspective when $u_0 = u_1$ is nothing other than a "tent map", which was found recently as a very convenient non-linear map for generating discrete-time chaotic signals with cryptographic properties [16].

2. The LIFE function; It was first mentioned in the context of Cellular Automata by Conway [1] and it is described by a set of logic rules instead of a truth table. It was proved that the cellular medium resulted by connecting standard CNN cells through this local Boolean function, has the property of universal computation [6]. It can be easily checked that the following simple PWL2cell: $y_{life} = \mathbf{sgn}\left[3 + u_c - |5 + 2u_\Sigma|\right]$ implements the same local logic as the function described by Conway. Here, u_Σ means the sum of all outputs of the neighbour cells and u_c the output of the actual cell. Compared with other implementations reported in the CNN literature [6],[14] it is the simplest implementation of Conway's Life and it opens an interesting perspective for building universally computational mediums at low cost and high density of cells.

Conclusions

A novel approach for the design and implementation of Boolean logic was described. It is essentially based on employing adaptation in piece-wise linear structures which have a very simple analog implementation using only rectifiers and weighted summation. It is able to implement any desired Boolean function with a structure called Rectification Neural Network (RNN) which adapts its complexity to the complexity of the Boolean function to be implemented. The basic cell of this network, here referred to as a PWL2cell also has potential application in solving pattern classification tasks. Several examples demonstrates that for most of the local Boolean functions defining meaningful tasks, the RNN approach leads to very simple implementations, in most cases requiring less hardware than their implementation using the standard digital approach. Moreover, the design technique is quite simple and can be easily automated. Preliminary tests of the PWL2cell with benchmark classification problems [12] showed good performance if we consider that it is a structure restricted to maximum 7 perceptrons. Further research will focus on the possibility of building a structure similar to the RNN for improving the performance when the tasks are analog pattern classification or signal processing.

Acknowledgement

The work is supported by the Office of Naval Research under grant No. N00014-96-1-0753. The first author was partially supported by a Fulbright grant. We express our thanks to Ken Crounse for reading the manuscript and making valuable suggestions.

References

[1] E. Berlekamp, J.H. Conway, and R.K. Guy, *Winning Ways for your Mathematical Plays,* New York: Academic, 1982, Vol. 2, chapter 2, pp. 817-850.

[2] L. O. Chua, S. M. Kang, "Section-wise piecewise-linear functions: canonical representation, properties, and applications", in *Proceedings of the IEEE,* Vol. 65, No. 6, pp. 915-929, June 1977.

[3] L. O. Chua, and A-C. Deng, "Canonical piecewise-linear modeling", *IEEE Tr. on Circuits and Systems,* Vol. CAS-33, No. 5, May 1986, pp. 511-525.

[4] L. O. Chua, T. Roska, "The CNN Paradigm", in *IEEE Tr. on Circuits and Systems- I,* Vol. 40, No. 3, pp. 147- 156, March 1993.

[5] K.R. Crounse, E.L. Fung and L.O. Chua, "Efficient implementation of neighbourhood logic for cellular automata via the cellular neural network universal machine", ", *IEEE Tr. on Circuits and Systems - I,* Vol.44, No.4, April 1997, pp. 355-361.

[6] K. R. Crounse and L. O. Chua, "The CNN universal machine is as universal as a Turing machine", *IEEE Tr. on Circuits and Systems - I,* Vol. 43, Apr. 1996, pp. 353-355.

[7] M.H. Hassoun, "Fundamentals of Artificial Neural Networks", 1995, MIT Press

[8] C. Kahlert, and L. O. Chua, "A generalized canonical piecewise-linear representation", *IEEE Tr. on Circuits and Systems,* Vol.37, No. 3, March 1990, pp. 373-383.

[9] J-N. Lin, and R. Unbehauen, "Adaptive nonlinear digital filter with canonical piecewise-linear structure", *IEEE Trans. on Circuits and Systems,* Vol. 37, No. 3, March 1990, pp. 347-353.

[10] N.Plaziac, C.T. Ledinh, J-P. Adoul, "PWL Nonlinear adaptive filter via RLS and NLMS algorithms", *IEEE Transactions on Signal Processing,* Vol. 45, No. 5, May 1997, pp. 1364-1367.

[11] *Analogue IC design: the current mode approach,* Edited by C. Toumazou, F.J. Lidgey and D.G. Haigh, IEE circuits and systems series; 2, London, June 1990.

[12] ***, ELENA-95 "Enhanced Learning for Evolutive Neural Architecture", technical report ESPRIT-6891, (Benchmarks and Databases), available from Internet at *http://www.dice.ucl.ac.be/neural-nets/ELENA/ELENA.html* , 1995.

[13] B. Widrow, and M.E. Hoff Jr., "Adaptive switching circuits", in *IRE Western Electric Show and Convention Record,* part 4, pp. 96-104, 1960.

[14] L. Nemes, L.O. Chua and T. Roska, "Decomposition of linearly non-separable Boolean functions and their implementation on the CNN universal machine", submitted for publication, Special Issue on CNN, *Int. J. Cir. Theory and Applications*, 1997.

[15] T. Roska and L.O. Chua, "The CNN universal machine: an analogic array computer", *IEEE Transctions on Circuits and Systems II,* Vol. 40, pp. 163-173.

[16] T. Hobutsu, Y. Nishio, I. Sasase, and S. Mori, "A secret key criptosystem by iterating a chaotic map", *Proceedings of Eurocrypt '91,* pp. 127-140, 1991.