[4] L. Kocarev and T. D. Stojanovski, "A model for secret-key cryptography using chaotic synchronization," in *Proc. Int. Symp. Information Theory Applications*, Sydney, Australia, 1994.

[5] L. Kocarev and T. Stojanovski, "On chaotic synchronization and secure communications," *IEICE Trans. Electron. Commun. Comput. Sci.*, vol. E78-A, no. 9, pp. 1142–1147, 1995.

[6] H. Zhou and X.-T. Ling, "Problems with the chaotic inverse system encryption approach," *IEEE Trans. Circuits Syst. I*, vol. 44, pp. 268–271, Mar. 1997.

[7] H. Zhou, X.-T. Ling, and J. Yu, "Secure communication via one-dimensional chaotic inverse systems," in *Proc. IEEE Int. Symp. Circuits Systems*, 1997.

Multiplierless Digital Learning Algorithm for Cellular Neural Networks

Renzo Perfetti and Giovanni Costantini

***Abstract*—A new learning algorithm is proposed for space-varying cellular neural networks, used to implement associative memories. The algorithm exhibits some peculiar features which make it very attractive: the finite precision of connection weights is automatically taken into account as a design constraint; no multiplication is needed for weight computation; learning can be implemented in fixed point digital hardware or simulated on a digital computer without numerical errors.**

***Index Terms*—Associative memories, cellular neural networks, learning.**

I. Introduction

This paper is concerned with the design of associative memories using space-varying cellular neural networks (CNNs). The problem is to compute the weights of the cell-dependent templates in order to store binary information in a distributed and content-retrievable fashion. Several design techniques have been proposed in the literature to perform this task [1]–[8]. These methods differ in several aspects e.g., capacity, error correction capability, connectivity, additive learning, computational complexity.

An important issue is the effect of finite precision when implementing these memories using analog hardware. The limited precision of analog weights introduces errors in the nominal connection matrix and these errors can make unstable some of the otherwise stored patterns. The algorithm proposed in this Brief was developed to take into account, just in the learning phase, the limited precision of analog hardware implementations of CNNs. The computed weights have the required precision, so the synthesized network will be void of sensitivity problems.

As side benefits, the proposed algorithm exhibits some peculiar features which make it very attractive from a computational viewpoint: no multiplication is needed to compute the weights; the algorithm can be implemented in fixed point digital hardware or simulated on a digital computer without numerical errors.

Manuscript received January 4, 2000; revised October 4, 2000. This paper was recommended by Associate Editor P. Szolgay.

R. Perfetti is with the Department of Electronic and Information Engineering, University of Perugia, Perugia, Italy.

G. Costantini is with the Department of Electronic Engineering, University of Rome "Tor Vergata," Rome, Italy.

Publisher Item Identifier S 1057-7122(01)03845-4.

The paper is organized as follows. In Section II, some theoretical results are shortly summarized. In Section III, the algorithm is outlined. In Section IV, some simulations results are presented concerning the convergence of the algorithm and the performance of the synthesized CNN. Section V presents an example of digital hardware implementation of the algorithm. Finally, Section VI contains a comparison with different existing methods.

II. Review of Theory

We assume a CNN model defined on a rectangular $(M \times N)$ cell grid. $C(i, j)$ denotes the cell at the intersection of row i and column j. $N_r(i, j)$ denotes the neighborhood of cell $C(i, j)$, with radius r. To simplify the statements and the notation, wraparound connections will be assumed. This is equivalent to assume the CNN arranged on a torus. The state equation of the CNN is as follows:

$$\dot{x}_{ij} = -x_{ij} + \sum_{C(k,\ell) \in N_r(i,j)} \mathbf{A}_{ij},\, k\ell f(x_{k\ell}) \qquad (1)$$

where x_{ij} is the state of cell $C(i, j)$. $A_{ij,\, k\ell}$ denotes the connection weight .from cell $C(k, \ell)$) to cell $C(i, j)$. The transfer function $f()$ is the usual piecewise-linear function: $f(x) = 0.5\ (|x+1| - |x-1|)$. The proposed method is based on the following theoretical results, which are summarized for the reader's convenience.

In the saturation regions, necessary and sufficient conditions for the existence of an equilibrium point $\mathbf{x}$ are [9]

$$\sum_{C(k,\ell) \in N_r(i,j)} \mathbf{A}_i j,\, k\ell f(x_{ij}) f(x_{k\ell}) > 1 \qquad (2)$$

for every i and j.

In each saturation region there is at most one equilibrium point. Equilibrium points in saturation regions are asymptotically stable. Hence there is a one-to-one correspondence between stable equilibrium points in saturation regions and the corresponding output bipolar patterns, $\mathbf{y} = f(\mathbf{x})$, which will be called *stable output patterns*. Moreover, if $A_{ij,\, ij} \geq 1$ for every i and j, then there are stable equilibrium points only in the saturation regions. Hence, only bipolar outputs can be observed in practice [9]. Finally, if $A_{ij,\, ij} \leq 1$ for every i and j, then two distinct stable output patterns must differ in more than one pixel per neighborhood [8]. This last property entails a local error correction capability that allows to recall a stored pattern, from a noisy version of it, provided that the errors are almost uniformly distributed on the cell array [8].

According to the above properties, we assume $A_{ij,\, ij} = 1$ for every i, j. So, taking into account that $(y_{ij})^2 = 1$ for $y_{ij} = \pm 1$, conditions (2) become

$$\sum_{C(k,\ell) \in \underline{N}_r(i,j)} \mathbf{A}_{ij,\, k\ell}\, y_{ij}\, y_{k\ell} > 0 \qquad (3)$$

where $\underline{N}_r(i, j) = N_r(i, j) - C(i, j)$, and $y_{ij} = f(x_{ij})$.

The design problem of the CNN associative memory is formulated as follows.

The L bipolar patterns to be stored are: $\mathbf{y}^{(1)} \ldots \mathbf{y}^{(L)}$. Assume $A_{ij,\, ij} = 1$ for every i, j. Then, find the connection weights,

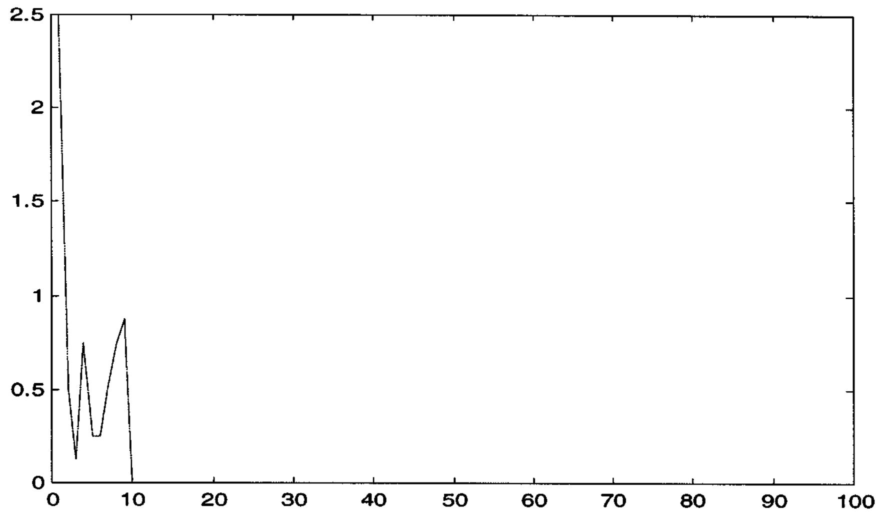


Fig. 1. $J(t)$ versus t for Example 1 in the text.

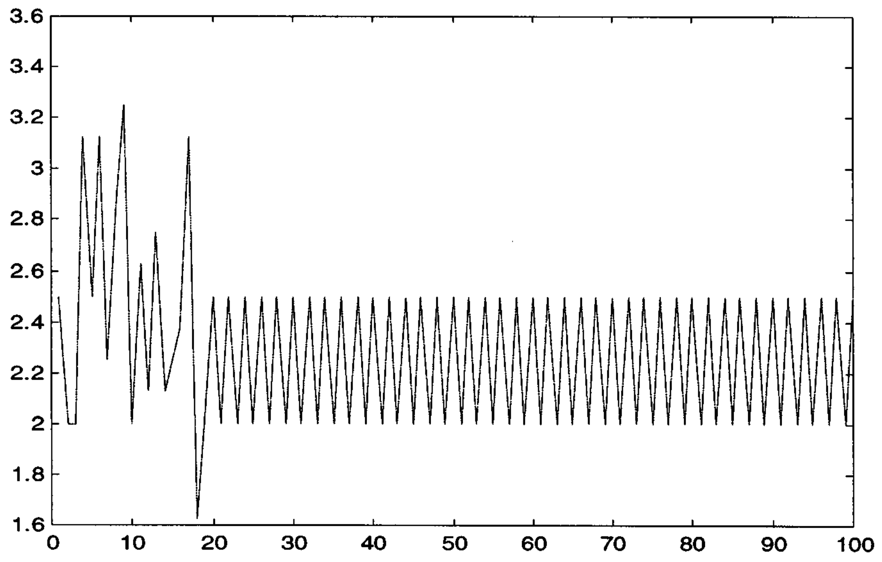


Fig. 2. $J(t)$ versus t for Example 2 in the text.

$A_{ij,k\ell}$, $k\ell \neq ij$, $i = 1, \ldots, M$, $j = 1, \ldots, N$, satisfying the following set of constraints:

$$\sum_{C(k,\ell)\in \underline{N}_r(i,j)} \mathbf{A}_{ij,k\ell}\, y_{ij}^{(m)} y_{k\ell}^{(m)} \geq \delta > 0 \tag{4}$$

$i = 1, \ldots, M$, $j = 1, \ldots, N$, $m = 1, \ldots, L$.

δ represents a margin of stability for the stored patterns.

III. Proposed Algorithm

The proposed algorithm is based on a penalty function approach to solve the set of constraints (4). Let consider one of the inequalities (4). If $A_{ij,k\ell}$ and the product $y_{ij}^{(m)} y_{k\ell}^{(m)}$ have the same sign, the corresponding term $A_{ij,k\ell}\, y_{ij}^{(m)} y_{k\ell}^{(m)}$ gives a positive contribution to the left hand of (4). Otherwise, if the signs are opposite, the contribution is negative. Since the sum must be positive in order to satisfy the design constraint, the weights $A_{ij,k\ell}$ should be chosen with the same sign as that of the product $y_{ij}^{(m)} y_{k\ell}^{(m)}$. This is possible only if $L = 1$.

TABLE I
Storage Probability Versus δ

δ	Storage probability
0	0.80
0.25	0.77
0.5	0.75

If $L > 1$, $A_{ij,k\ell}$ must satisfy L simultaneous constraints with different products $y_{ij}^{(m)} y_{k\ell}^{(m)}$. However we can try to update the weights according to the following rule: for each constraint being violated, increase the weight $A_{ij,k\ell}$ if $y_{ij}^{(m)} y_{k\ell}^{(m)} = 1$, decrease the weight $A_{ij,k\ell}$ if $y_{ij}^{(m)} y_{k\ell}^{(m)} = -1$.

The following algorithm can be used to update the weights according to the above rule:

Let

$$A_{ij,k\ell}(0) = 0, \qquad \text{for every } i,\ j,\ k,\ \ell.$$

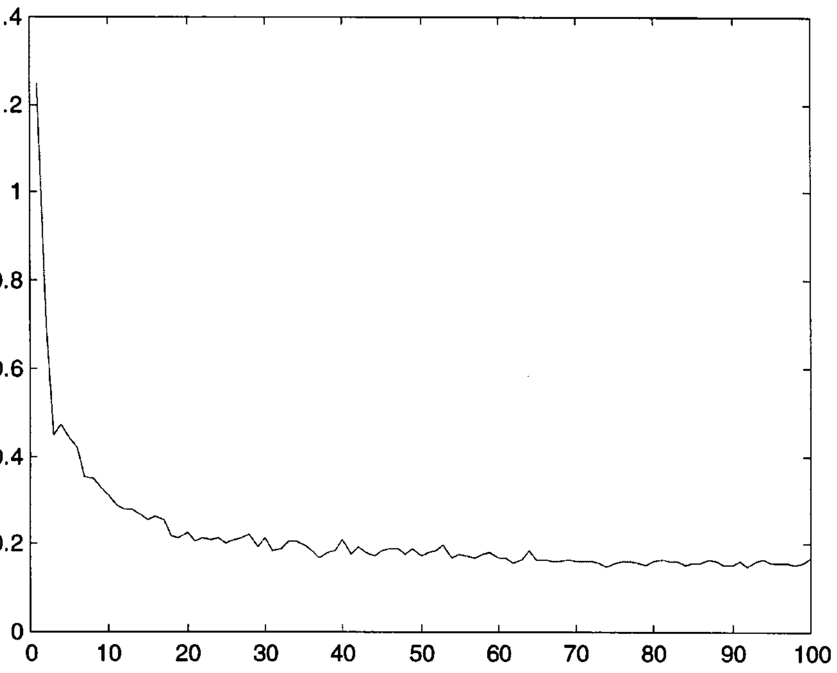


Fig. 3. $J(t)$ averaged over 100 experiments in Example 3.

For $t = 0, 1, 2, \ldots,$ compute:

$$P\left(\Delta_{ij}^{(m)}(t)\right), \quad i = 1, \ldots, M;$$
$$j = 1, \ldots, N; \quad m = 1, \ldots, L,$$

where

$$\Delta_{ij}^{(m)}(t) = \sum_{C(k,\ell)\in \underline{N}_r(i,j)} \mathbf{A}_{ij,k\ell}(t)\, y_{ij}^{(m)} y_{k\ell}^{(m)} - \delta \tag{5}$$

and

$$P(x) = 0, \text{ for } x \geq 0, \quad P(x) = 1, \text{ for } x < 0; \tag{6}$$

then update the weights as follows:

$$A_{ij,k\ell}(t+1) = A_{ij,k\ell}(t) + \alpha \sum_{m=1}^{L} y_{ij}^{(m)} y_{k\ell}^{(m)} P\left(\Delta_{ij}^{(m)}(t)\right) \quad k\ell \in \underline{N}_r(i,j). \tag{7}$$

Note, that $P(x)$ is a flag function of the constraint violation. It depends on the weights at iteration t. $\alpha > 0$ is a learning rate.

As explained in [10], asymptotic convergence of this type of algorithm to a solution of (4) is not guaranteed, since it can approach a limit cycle in the solution space. However, by choosing α sufficiently small, it is possible to force the sequence $A_{ij,k\ell}(t), A_{ij,k\ell}(t+1), \ldots$ to stay arbitrarily close to a correct solution of (4). Then, the presence of a margin δ in (4) guarantees the storage of the desired patterns.

Let us examine some properties of the proposed algorithm.

1) From expressions (5) and (7) it follows that *the algorithm can be implemented without multiplications.*
2) Since each term of the sum in (7) can be $+1$, -1 or zero, starting from $A_{ij,k\ell}(0) = 0$, the weights $A_{ij,k\ell}(t)$ are integer multiples of α, for every t. Assuming $\alpha = 2^{-B}$, where B is a positive integer, *all the weights (at each iteration) have a fixed point digital representation.*
3) The required precision is 2^{-B} i.e., equal to the learning rate α. Note, that more than $B+1$ bits could be required to represent the weights, since the magnitude of the weights can be greater than one. If the weights and δ are in magnitude less than one, $B+1$ bits are sufficient to represent them, and then to implement the algorithm using digital hardware.
4) The algorithm can be implemented or simulated on a digital hardware, without numerical errors, provided that a sufficient number of bits is used. In fact no rounding or truncation of the weights is necessary. Moreover, since only additions are needed, no arithmetic errors are generated if overflows are prevented.

IV. Experimental Results

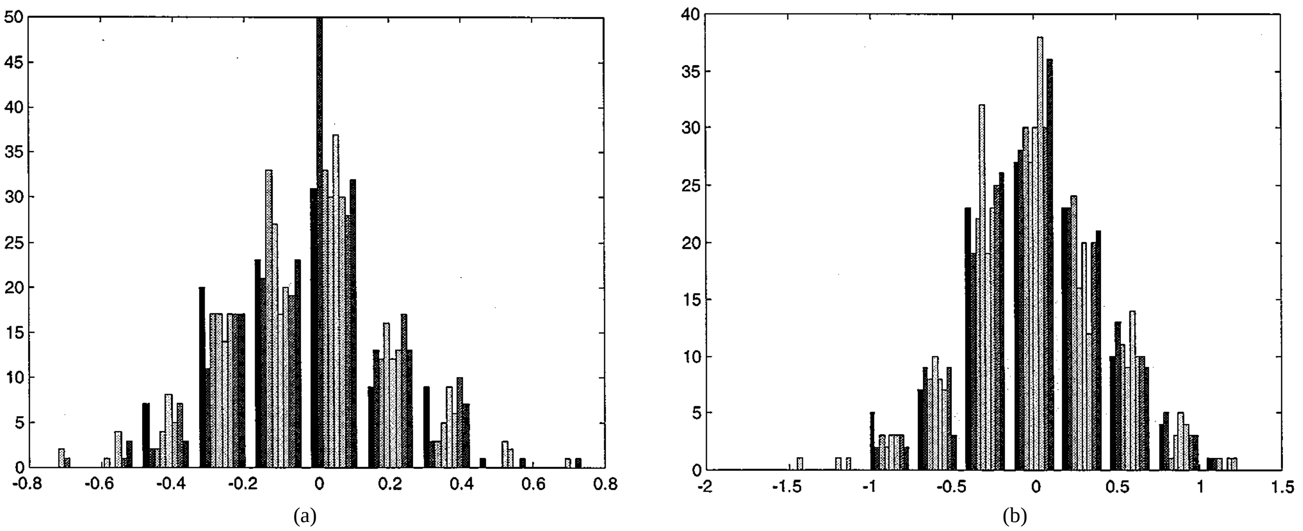
Computer simulations have been performed to ascertain the behavior of the proposed learning algorithm.

Example 1: Since no symmetry is assumed for the weights, the constraints (4) for a cell are independent from the weights of the other cells. So, we first examine a single cell to understand the behavior of the proposed algorithm. The neighborhood is 3×3 (nine weights). For a single cell, we must consider sub-patterns of nine components. Assume the following ten sub-patterns to be stored:

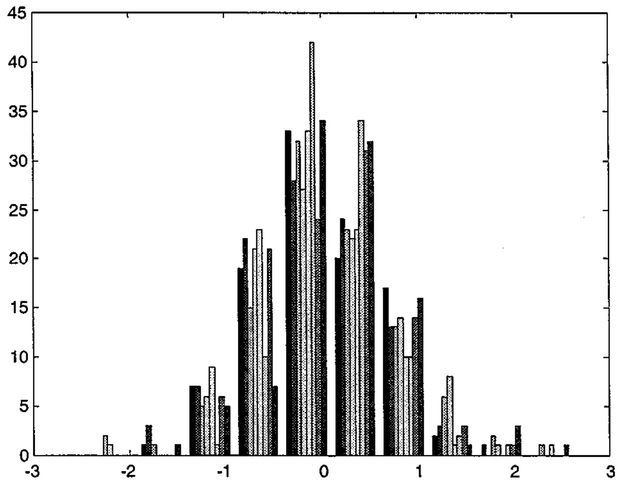
$$\begin{aligned}
y^{(1)} &= [\;\;1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\; -1 \;\; -1 \;\; -1] \\
y^{(2)} &= [-1 \;\; -1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\;\;\; 1] \\
y^{(3)} &= [\;\;1 \;\; -1 \;\; -1 \;\; -1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\;\;\; 1] \\
y^{(4)} &= [\;\;1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1] \\
y^{(5)} &= [-1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\; -1] \\
y^{(6)} &= [-1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1] \\
y^{(7)} &= [\;\;1 \;\;\;\; 1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1] \\
y^{(8)} &= [-1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\; -1 \;\; -1 \;\; -1] \\
y^{(9)} &= [\;\;1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1 \;\;\;\; 1] \\
y^{(10)} &= [\;\;1 \;\;\;\; 1 \;\;\;\; 1 \;\; -1 \;\; -1 \;\; -1 \;\;\;\; 1 \;\;\;\; 1 \;\; -1].
\end{aligned}$$

The following values have been used: $\delta = 0.25$; $\alpha = 0.125 = 2^{-3} (B = 3)$. Fig. 1 shows the behavior of the error function defined as follows:

$$J(t) \doteq -\sum_{m=1}^{L} \mathbf{P}\left(\Delta_{i,j}^{(m)}(t)\right) \Delta_{i,j}^{(m)}(t).$$



(a) (b)



(c)

Fig. 4. Distribution of weight values in Example 3. (a) $\delta = 0$. (b) $\delta = 0.25$; (c) $\delta = 0.5$.

The error function $J(t)$ converges to zero. The final weights are

$$0.75 \quad -0.75 \quad -0.5 \quad 0.25 \quad 0.5 \quad 0.5 \quad 0 \quad 0.5.$$

The self feedback is not shown, being fixed at one. All the weights are less than one in magnitude, so four bits are sufficient to represent the weights. In correspondence to these weights, the patterns are stored as stable states of the cell.

Example 2: The sub-patterns to be stored are

$$\begin{aligned}
y^{(1)} &= [\ 1 \quad 1 \quad -1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1]\\
y^{(2)} &= [-1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1 \quad 1]\\
y^{(3)} &= [-1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]\\
y^{(4)} &= [\ 1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad -1 \quad 1 \quad 1]\\
y^{(5)} &= [-1 \quad 1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1]\\
y^{(6)} &= [-1 \quad 1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1]\\
y^{(7)} &= [\ 1 \quad 1 \quad 1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1]\\
y^{(8)} &= [-1 \quad -1 \quad 1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1 \quad -1]\\
y^{(9)} &= [-1 \quad 1 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad 1]\\
y^{(10)} &= [-1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1]
\end{aligned}$$

with the same parameters of Example 1. Fig. 2 shows the behavior of $J(t)$. It does not converge to zero but exhibits a steady-state oscillation between 2 and 2.5. The nonzero steady-state value of $J(t)$ corresponds to the fact that some patterns are not stored. Indeed, patterns

TABLE II
STORAGE PROBABILITY VERSUS L

L	Storage probability
5	0.96
8	0.86
10	0.77
15	0.26

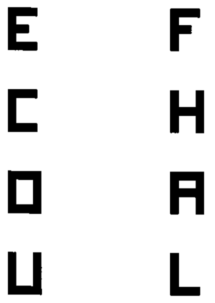
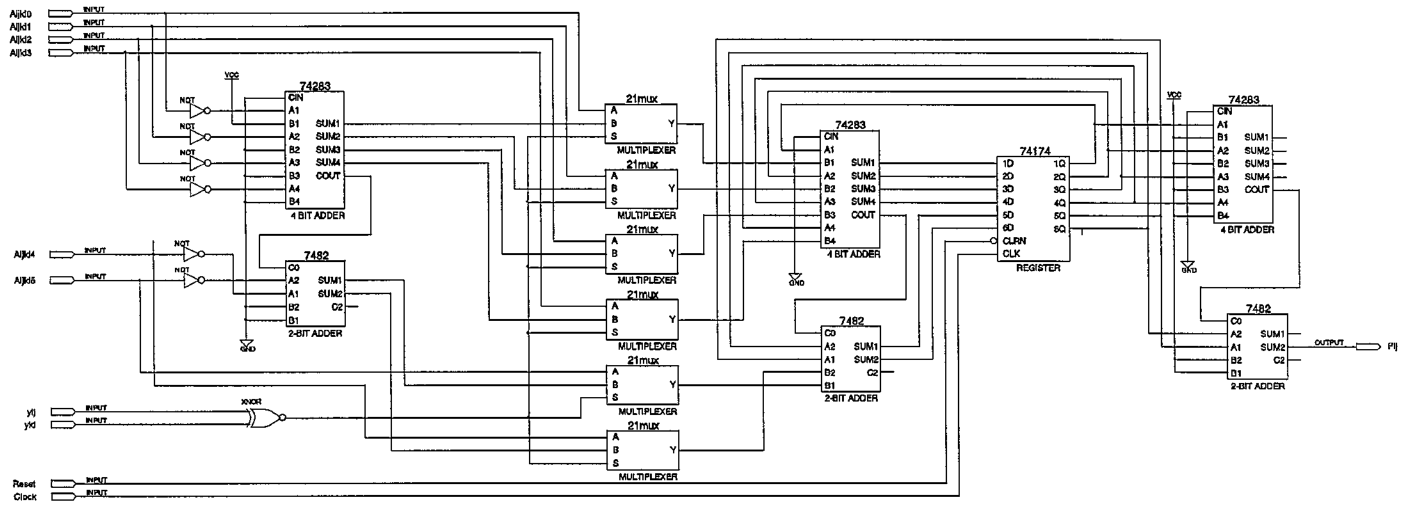
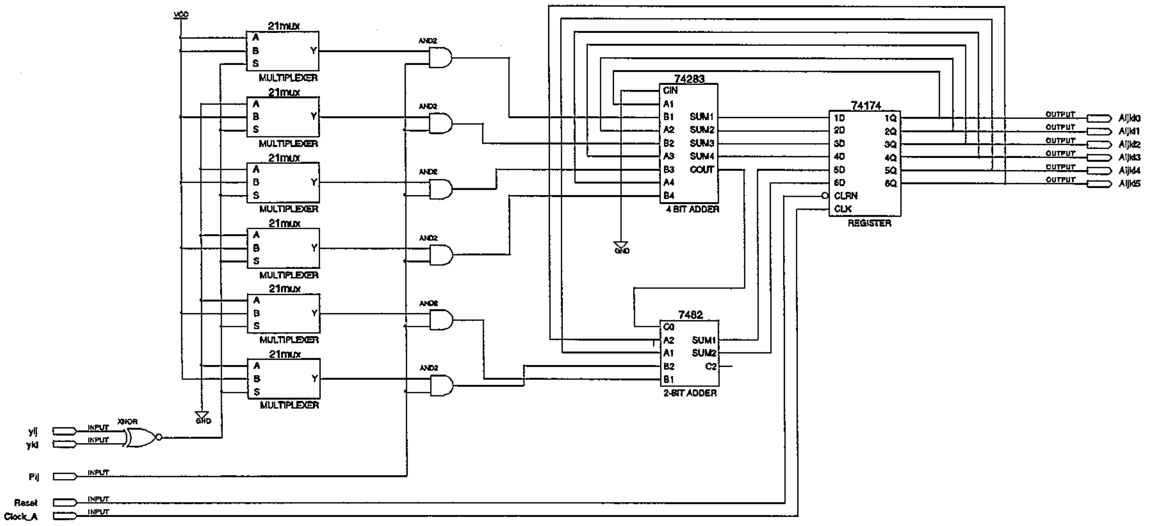


Fig. 5. Patterns stored in Example 4.



(a)



(b)

Fig. 6. (a) Circuit used to compute $P(\Delta_{ij}^{(m)})$. (b) Circuit used to implement (7) in the text.

$\mathbf{y}^{(2)}\mathbf{y}^{(5)}\mathbf{y}^{(9)}$ are not stable states of the cell. This behavior is not significantly changed if we decrease α. With $\alpha = 2^{-4}(B = 4)$, $J(t)$ converges to the constant value 1.5 and patterns $\mathbf{y}^{(2)}\mathbf{y}^{(6)}\mathbf{y}^{(9)}$ are not stored. With $\alpha = 2^{-7}$, $J(t)$ converges smoothly to 1.5, but patterns $\mathbf{y}^{(6)}\mathbf{y}^{(8)}\mathbf{y}^{(9)}$ are not stored.

Example 3: To test the general behavior of the proposed algorithm, 300 independent tests were performed for a CNN with $r = 1$ and $L = 10$ random patterns to be stored. The following parameters were used: $\alpha = 2^{-4}$(0.0625),$\delta = 0.0$, 0.25 and 0.5. The storage probability per cell in the three cases is shown in Table I. Fig. 3 shows the behavior of $J(t)$ averaged over 100 experiments, for $\delta = 0.25$. In Fig. 4 the corresponding distributions of the weight values are shown (100 experiments, i.e., 800 weights). The weight distribution is approximately Gaussian with zero mean. Increasing δ the standard deviation increases, and there is a significant probability to get weights greater than one in magnitude. Hence, more than $B+1$ bits would be required to represent them. Table II shows the storage probability for $\delta = 0.25$ and $L = 5$, 8, 10, 15.

Finally, the same test has been performed for a CNN with $r = 2(5\times 5$ neighborhood). Assuming $\delta = 0.25$, $\alpha = 0.0625$, and $L = 30$, the storage probability per cell was 0.94.

According to Tables I and II, we can say that the capacity obtained by the proposed method, in a statistical sense, is roughly equal to the number of connections per cell. This result is in accordance with the measure of capacity proposed in [11], and is comparable with the simulation results presented in [6]–[8].

Example 4: As a final example, we consider the problem of storing the eight patterns shown in Fig. 5 on a 12×12 CNN with $r = 1$. In Fig. 5, -1 represents white, $+1$ represents black. The proposed algorithm was used with $\alpha = \delta = 0.125$. All the patterns were stored as stable states of the network after 4 iterations. Then, the CNN with the computed weights was simulated to ascertain the error correction capability. For each of the eight stored patterns, 50 trials were performed. In each trial, the initial state of the network is obtained by randomly reversing the pixels of the corresponding stored pattern, with probability 0.15 (15% of pixels are reversed). The probability of error in the recall process depends on the pattern: it varies from 12% ("L," "F") to 24% ("A," "H"). The probability of error is not negligible due to the similarity among patterns. Indeed, the CNN dynamics converges in most cases to one of the eight stored patterns. The probability to get a spurious state is very low.

V. Hardware Implementation

The proposed learning algorithm was implemented on digital hardware using the EPLD EPM7256ATC100-7 by ALTERA, with 40 MHz clock.

The algorithm described in Section III can be decomposed in two stages. The first one corresponding to the computation of the penalty flags $P(\Delta_{ij}^{(m)})$ [eqns. (5) and (6)]. The second one corresponding to the weights updating (7). We used two distinct circuits for these two stages of computation.

The first circuit is shown in Fig. 6(a). Each component of a pattern is represented with 1/0 corresponding respectively to $+1/-1$. The weights are represented using two's complement fixed-point binary format with six bits: one sign bit, two bits for the integer part, and three bits for the fractional part ($\alpha = 2^{-3}$). The product $y_{ij}^{(m)} y_{k\ell}^{(m)}$ in (5) is implemented with the XOR function. The result (S) is $S = 0$ if $y_{ij}^{(m)} y_{k\ell}^{(m)} = -1$, $S = 1$ otherwise. The value of S controls the sign of $A_{ij,\,k\ell}$. Using a multiplexer, the value of $A_{ij,\,k\ell}$ or its two's complement are selected. The two's complement of $A_{ij,\,k\ell}$ is obtained by reversing the bits and adding one. The output of the multiplexer is the input of an 4+2 bit adders (74 283, 7482) which implement expression (5). Using eight clock cycles, the eight weights of the cell $C(i,\,j)$ are added (or subtracted) to the content of an accumulation register (74 174), initially zeroed. Finally, the value $\delta = 2^{-3}$ is subtracted from the result using a second adder. The sign bit of the result now represents the flag $P(\Delta_{ij}^{(m)})$. The same computation is repeated for $m = 1,\,\ldots,\,L$, and the computed flags are stored in a flag register.

Then, the second stage (7) is executed [Fig. 6(b)]. The XOR is performed to compute the sign of $y_{ij}^{(m)} y_{k\ell}^{(m)}$. The result (S) controls the output of a multiplexer whose inputs are 2^{-B} and its complement. The output of the multiplexer is used as input of an adder $(74\,283 + 7482)$ which implements expression (7), adding (or subtracting) the L terms to the content of the accumulation register (74 174). The input to the adder is zeroed, using six AND ports, when $P(\Delta_{ij}^{(m)}) = 0$. Initially, the accumulator contains $A_{ij,\,k\ell}$ (old); at the end of the computation it contains $A_{ij,\,k\ell}$ (new).

We tested the hardware implementation using the Example 4 above, with the same values for α and δ. The computation of each $P(\Delta_{ij}^{(m)})$ requires eight clock cycles (3×3 neighborhood, self-feedback fixed). The weight adaptation requires L clock cycles for each weight. Hence, the updating of the weights for a cell requires $8L + 8L = 16L$ cycles. In our example $L = 8$, then 128 cycles. The learning algorithm converges in four iterations, so the computation requires 128×4×25 ns =12.8 μs. We used the same circuits in Fig. 6 for the whole CNN, with a total computation time of 144×12.8=1843 μs =1.843 ms. Using two circuits for each cell we could complete the learning in 12.8 μs!

VI. Discussion

It is useful to compare the present method to similar existing techniques for associative memories design. The present method is an evolution of that proposed in [2] to design Hopfield neural networks, and then revisited and applied to BSB neural networks [5]. The main differences are: the continuous-time dynamics of the network, the local connectivity, the absence of symmetries, and above all the digital representation (finite precision) of weights. As a consequence of this last variant, convergence cannot be proved as in [2], [5].

In [7] the authors propose the same method to design CNNs with analog weights. The present approach differ from [7] in two aspects: the digital representation of weights, and the value of $A_{ij,\,ij}$, here fixed at one. As concerns this last point, we note that if $A_{ij,\,ij}$ is allowed to increase, inequalities (4) always have a solution, for every value of δ. In [7] the authors use $A_{ij,\,ij}$ as a design parameter to guarantee the storage of the desired patterns. However, increasing the self-feedback entails an increment in the number of spurious states, and then a reduction of error correction capability [7]. So, a trade-off must be pursued between these two conflicting aspects. We preferred to fix the value of the self-feedback in order to guarantee saturated outputs, and to keep at a minimum the number of spurious stable states, i.e., $A_{ij,\,ij} = 1$. In fact, only if $A_{ij,\,ij} \geq 1$ all the stable states of the CNN are into the saturation regions, namely with outputs $+1$ and -1. Otherwise the CNN dynamics could converge to stable states with nonsaturated outputs, that are necessarily spurious states. Adopting this strategy, storage capacity is the main issue, since the given patterns are not necessarily stored, and the error correction is the better we can get with the given assumptions. However, the proposed algorithm can easily be modified to include different values of self-feedback, or symmetric weights, preserving the advantages in computation.

References

[1] S. Tan, J. Hao, and J. Vandewalle, "Cellular neural networks as a model of associative memories," in *Proc. 1st Int. Workshop Cellular Neural Networks*, Budapest, Hungary, 1990, pp. 26–35.

[2] R. Perfetti, "A neural network to design neural network," *IEEE Trans. Circuits Syst.*, vol. CAS-38, pp. 1099–1103, Sept. 1991.

[3] G. Martinelli and R. Perfetti, "Associative memory design using space-varying cellular neural networks," in *Proc. 2nd Int. Workshop Cellular Neural Networks*, Munich, Germany, 1992, pp. 117–122.

[4] D. Liu and A. N. Michel, "Sparsely interconnected neural networks for associative memories with applications to cellular neural networks," *IEEE Trans. Circuits Syst. II*, vol. 41, pp. 295–307, 1994.

[5] R. Perfetti, "A synthesis procedure for brain-state-in-a-box neural networks," *IEEE Trans. Neural Networks*, vol. 6, pp. 1071–1080, May 1995.

[6] M. Brucoli, L. Carnimeo, and G. Grassi, "A global approach to the design of discrete-time cellular neural networks for associative memories," *Int. J. Circuit Theory Appl.*, vol. 24, pp. 489–510, 1996.

[7] P. Szolgay, I. Szatmari, and K. Laszlo, "A fast fixed point learning method to implement associative memory on CNNs," *IEEE Trans. Circuits Syst. I*, vol. 44, pp. 362–366, Apr. 1997.

[8] R. Perfetti, "Dual-mode space-varying self-designing cellular neural networks for associative memory," *IEEE Trans. Circuits Syst. I*, vol. 46, pp. 1281–1285, Oct. 1999.

[9] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 1257–1272, Oct. 1988.

[10] A. Rodriguez-Vasquez, R. Dominguez-Castro, A. Rueda, J. L. Huertas, and E. Sanchez-Sinencio, "Nonlinear switched capacitor neural networks for optimization problems," *IEEE Trans. Circuits Syst.*, vol. CAS-37, pp. 384–397, Mar. 1990.

[11] M. Bo, G. Martinelli, and R. Perfetti, "Estimating the storage capacity of space-varying cellular neural networks," *Electron. Lett.*, vol. 30, no. 20, pp. 1691–1693, 1994.