IEEE TRANSACTIONS ON

NEURAL NETWORKS

A PUBLICATION OF THE IEEE NEURAL NETWORKS SOCIETY

MAY 2003 VOLUME 14 NUMBER 3 ITNNEP (ISSN 1045-9227)

(Contents Continued on Back Cover)

IEEE

Neural Associative Memory Storing Gray-Coded Gray-Scale Images

Giovanni Costantini, Daniele Casali, and Renzo Perfetti

***Abstract*—In this paper, we present a neural associative memory storing gray-scale images. The proposed approach is based on a suitable decomposition of the gray-scale image into gray-coded binary images, stored in brain-state-in-a-box-type binary neural networks. Both learning and recall can be implemented by parallel computation, with time saving. The learning algorithm, used to store the binary images, guarantees asymptotic stability of the stored patterns, low computational cost, and control of the weights precision. Some design examples and computer simulations are presented to show the effectiveness of the proposed method.**

***Index Terms*—Associative memories, brain-state-in-a-box (BSB) neural networks, gray-scale images, neural networks with finite precision weights.**

I. Introduction

THE realization of binary associative memories by recurrent neural networks has been widely explored [1]–[3]. One of the most promising area of application of associative memories is that of image recognition in presence of noise. The design goal is to recognize a noisy image, even if it differs from the original one in any pixel, as the human eye does. An image with n pixels and L gray levels can be represented using $R = \log_2(L)$ bits for each pixel. It can be stored using a binary neural network with nR neurons; however, the number of interconnections is very large, i.e., n^2R^2.

A second approach is based on a multilevel activation function with L plateaus, in place of two as in the usual sigmoidal function [4], [5]. The resulting neural network presents stable equilibria with multivalued components, corresponding to the different gray levels. The number of neurons is n; the number of interconnections is n^2. For networks with multilevel sigmoidal functions, sophisticated but heavy design methods have been proposed.

A third approach is based on complex-valued neural networks [6], [7]. The neuron state can assume one of L complex values, with unit magnitude and different phases, regularly spaced between 0 and 2π. Each phase angle corresponds to a different gray level of the image pixel. The number of neurons is n; the number of interconnections is n^2. For networks with complex-valued neurons, only the simple Hebb rule is available, at the best of our knowledge. It is well known that Hebb rule gives poor performance in the case of binary images, as concerns both storage capacity and noise suppression.

A different approach, investigated in this paper, consists in the decomposition of the image into R binary images, stored using R independent *binary* neural networks. The total number of interconnections is n^2R, but each independent network has only n^2 interconnections. The R networks can be implemented via parallel hardware, both for learning and recall, with considerable saving in time. This approach retains the two main advantages of binary neural networks: the robustness of the steady state solutions, with respect to noise and to inaccuracies of implementation; the availability of efficient and simple design methods.

Several neural models have been proposed in the literature to realize binary associative memories. Among these, the brain-state-in-a-box (BSB) neural network is frequently used [8]–[12]. The connection weights of BSB neural networks can be computed by solving a set of linear constraints. This approach was developed by different researchers, using iterative algorithms [10], analog "designer" networks [11], or semidefinite programs for linear matrix inequalities [12]. In this paper, we use the algorithm proposed in [13], which faces an important, rarely addressed, problem, i.e., the precision required to implement the neural network using digital hardware. To this end, we compute weights with controlled precision, so that the digital hardware implementation of the associative memory will exhibit the same storage and retrieval performance of simulations.

This paper is organized as follows. The BSB neural model and the proposed learning algorithm, are shortly summarized in Section II. In Section III, the decomposition of gray-scale images into binary ones is discussed. In Section IV, the algorithm used to generate random gray-scale images for simulations, is outlined. Section V presents some design examples and simulation results. Some comments in Section VI conclude the paper.

II. BSB Model and Learning Algorithm

To implement a binary associative memory, we use the BSB neural model described by the following difference equation:

$$\mathbf{x}(k+1) = \mathbf{g}[\mathbf{x}(k) + \alpha \mathbf{W}\,\mathbf{x}(k)] \qquad k = 0, 1, 2 \ldots . \tag{1}$$

$\mathbf{x}(k) = [x_i(k)] \in [-1, +1]^n$, is the state vector at time k. n is the number of neurons. $\mathbf{W} = [w_{ij}] \in \Re^{nxn}$ is the weight matrix. $\mathbf{g}$ is a vector valued function, whose ith component is defined as

$$\begin{aligned} (\mathbf{g}(\mathbf{y}))_i &= 1, && \text{if } y_i \geq 1 \\ (\mathbf{g}(\mathbf{y}))_i &= y_i, && \text{if } -1 < y_i < +1 \\ (\mathbf{g}(\mathbf{y}))_i &= -1, && \text{if } y_i \leq -1. \end{aligned} \tag{2}$$

Several analysis results are available for the above neural model. In the following, we review only the most significant; the proof can be found in [10].

Manuscript received March 7, 2002; revised October 21, 2002.

G. Costantini and D. Casali are with the Department of Electronic Engineering, University of Rome "Tor Vergata," Rome, Italy.

R. Perfetti is with the Dipartimento di Ingegneria Elettronica e dell'Informazione, Università di Perugia, 06125 Perugia, Italy (e-mail: perfetti@diei.unipg.it).

Digital Object Identifier 10.1109/TNN.2003.810596

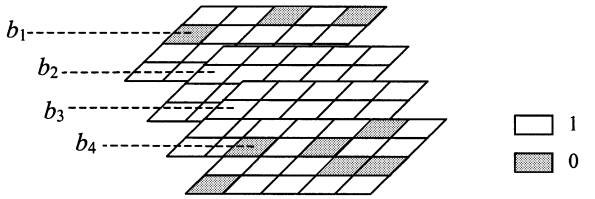

Fig. 1. Proposed architecture in the case of 16 gray levels.

Property 1

Let $w_{ii} \geq 0$ for $i = 1, \ldots, n$. Then, only the vertices of $[-1, +1]^n$ can be asymptotically stable equilibria of system (1).

As a consequence, if $\mathbf{W}$ has nonnegative diagonal terms, only binary steady-state solutions can be observed.

Property 2

Let $\xi \in B^n$, $B = \{-1, +1\}$. ξ is an asymptotically stable equilibrium point of system (1) iff

$$\sum_{j=1}^{n} w_{ij}\xi_i\xi_j > 0, \qquad i = 1, \ldots, n. \tag{3}$$

Constraints (3) represent existence and stability conditions for a given binary equilibrium point ξ. They can be used as design constraints, by solving (3) with respect to the weights, for a given set of desired binary equilibrium points $\xi^1 \ldots . \xi^M$.

Property 3

Let $w_{ii} = 0$, for $i = 1, \ldots, n$. Assume that $\xi \in B^n$ is an asymptotically stable equilibrium point of system (1). Then, none of the vectors $\xi' \in B^n$ at Hamming distance one from ξ, is an equilibrium point.

A zero-diagonal connection matrix guarantees the absence of two or more equilibria at Hamming distance one, i.e., in close proximity. Even if a trajectory starting at ξ' not necessarily converges to ξ, the condition $w_{ii} = 0, i = 1, \ldots, n$, is a prerequisite to obtain large basins of attraction for the stored patterns [10]. Moreover, setting to zero the diagonal entries of $\mathbf{W}$ we avoid the trivial solution $w_{ii} > \sum_{\substack{j=1 \\ j \neq i}}^{n} |w_{ij}|$ (all the vertices of $[-1, +1]^n$ would be asymptotically stable equilibria [8]).

The design of a binary associative memory, based on model (1), can be formulated as follows. Find the connection matrix $\mathbf{W}$ so that:

- a given set of binary vectors $\xi^{(1)} \ldots . \xi^{(M)} \in B^n$ represent as many asymptotically stable equilibria of system (1);
- the attractivity of the desired equilibria is as large as possible;
- the number of not desired stable equilibria is as small as possible.

Taking into account the properties above, the design can be formulated as a constraint satisfaction problem: find $\mathbf{W}$ such that

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij}\xi_i^{(m)}\xi_j^{(m)} \geq \delta > 0 \quad i = 1, \ldots ., n; \; m = 1, \ldots, M. \tag{4}$$

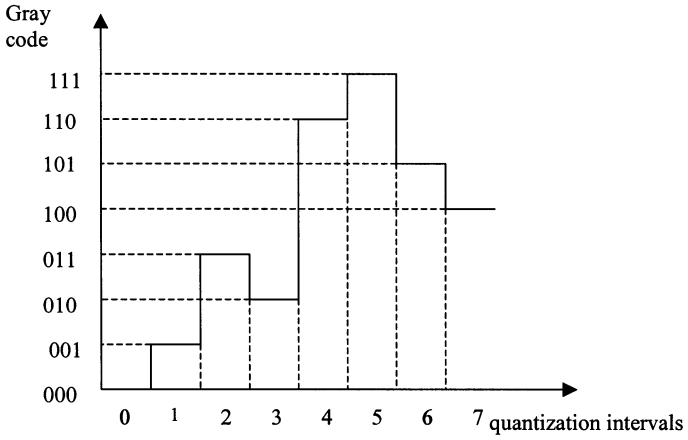

Fig. 2. Gray coding.

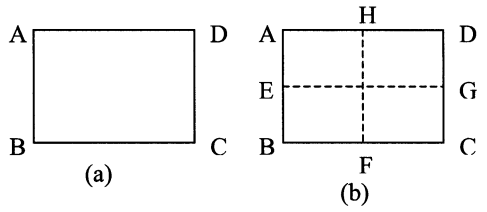

Fig. 3. Description of the RMD algorithm.

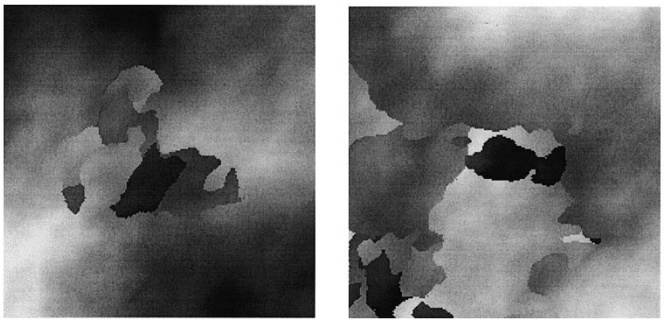

Fig. 4. Two images generated by the modified RMD algorithm.

In order to compute the weights satisfying constraints (4), we use the following iteration [13]:

$$w_{ij}(t+1) = w_{ij}(t)\xi \sum_{m=1}^{M} \xi_i^{(m)}\zeta_j^{(m)} P(\Delta_i^{(m)}(t))$$
$$t = 0, 1, 2, \ldots . \qquad i, j = 1, \ldots, n, \; i \neq j \tag{5}$$

where

$$\eta > 0$$
$$P(x) = 1, \text{ if } x < 0, P(x) = 0, \text{ if } x \geq 0.$$
$$\Delta_i^{(m)} = \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij}\xi_i^{(m)}\xi_j^{(m)} - \delta. \tag{6}$$

Each term of the sums in (5) can be ± 1 or zero. As a consequence, the learning algorithm (5) has the following properties.

1) Only additions are required for its implementation.
2) Starting from $w_{ij}(0) = 0$, the weights can be represented as $w_{ij}(t) = \pm\eta N_{ij}(t)$, where $N_{ij}(t)$ is a positive integer.

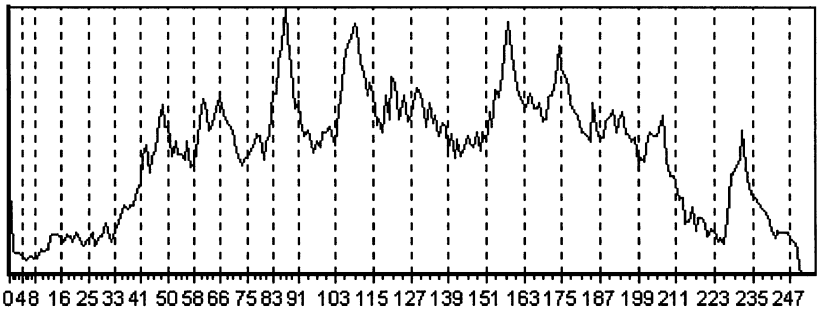

Fig. 5. Gray-level distribution for image in Fig. 4(a).

Hence, all the weights (at each iteration) have finite precision. The required number of bits is $\log_2(N_{\max})+1$, where $N_{\max}$ is the maximum value of N_{ij}.

3) The algorithm can be implemented or simulated on a digital hardware, without numerical errors, since no rounding or truncation is required to represent the weights. A digital implementation of the algorithm is described in [13].

Asymptotic convergence of (5) to a solution of (4) is not guaranteed, since the iteration can approach a limit cycle in the solution space [13]. In our experiments, the algorithm is stopped when all the terms $\Delta_i^{(m)}$ become nonnegative, for every i and m. If this condition is not reached within a given number of iterations, we say that the desired patterns cannot be stored with the stability margin δ. By choosing η sufficiently small, we obtain satisfactory performance, as it will be shown in Section V.

III. Decomposition of Gray Scale Images

Let consider an image with n pixels and L gray levels. Each pixel can be represented by R bits, $b_1 \ldots . b_R$, being $R = \log_2 L$. So the image can be decomposed into R binary images, each with n pixels. Each binary image can be stored into a binary associative memory, called *layer*, designed as explained in Section II. The recall process will recover a stored binary pattern in each layer. By combining the binary components in each layer, we can reconstruct the original image. In Fig. 1 the case $L = 16$ is shown. Each pixel is represented by $R = 4$ bits. Four neural networks are used, each storing one bit for each pixel. The four networks are not coupled, so a full parallel implementation is possible, both for computation of the weights, and for the recall process.

The coding strategy is of crucial importance. The usual binary-weighted coding entails a high sensitivity to additive Gaussian noise. Additive Gaussian noise with zero mean, gives a high probability of jumping from a quantization interval to an adjacent one. However, this jump could correspond to the reversing of several bits. For example, moving from the integer 3 to the integer 4, all the bits change ($011 \rightarrow 100$). As a consequence, gaussian noise amplifies the Hamming distance between the stored pattern and its noisy version, in each layer.

To circumvent this problem, we used the reflected-binary or *Gray code*, which has the property that only one bit can change, moving from one quantization interval to an adjacent one. The input-output relation for a three-bit Gray coding is shown in Fig. 2.

Now, moving from integer 3 to integer 4, only one bit changes ($010 \rightarrow 110$). Using the Gray code, zero-mean additive gaussian noise results in the minimal Hamming distance, in each layer, between the stored pattern and its noisy version. As a consequence, the probability of correct recall is improved.

Fig. 6. Image used to test the RMD algorithm.

IV. Random Midpoint Displacement Algorithm

To test the proposed associative memory design for gray-scale images, we need a reliable method to generate random images, whose gray levels distribution is similar to that of real-life images. To this end, we modified the random midpoint displacement (RMD) algorithm invented by Mulvey [14], widely used in different fileds to generate fractal pseudorandom images, called *plasmas*.

The RMD can be outlined as follows.

1) Consider a rectangle with the same size as the image to be generated [Fig. 3(a)]. Pick at random the gray levels of the four pixels A,B,C and D.
2) Divide the rectangle into four sub-rectangles, as shown in Fig. 3(b). The intensity of pixel E is computed as the mean intensity value between pixels A and B, plus a small signed random value, proportional to the size of segment AB. The intensity of the remaining pixels F, G, and H is computed the same way. Finally, the central pixel intensity is obtained by adding a random value to the mean of the four intensities E, F, G, and H.
3) Iterate Step 2 for each subrectangle, until the subrectangles have the pixel size.

The images generated by this algorithm share some useful features. In particular, nearby pixels have similar intensities, and the mean value of the intensity is varying over the image. The images generated by the RMD have a smooth appearance, but using a threshold we can create objects with definite contours. For example, if all the intensities below a given threshold are set to zero, we obtain black objects on a smooth background.

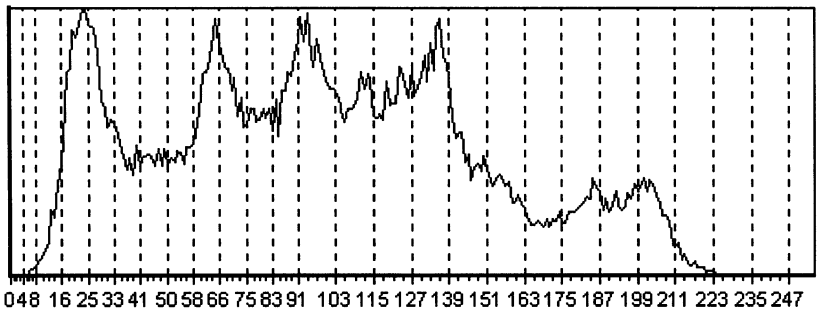

Fig. 7. Gray-level distribution for image in Fig. 6.

The intensity distribution of real-life images is characterized by some sharp peaks, with irregular size and position. To generate images with this property, we modified the RMD algorithm. As said above, using a threshold we can get black objects. We used this objects as "holes" through which we can see a second *plasma* image below the first, generated with the same algorithm. Using different thresholds, we obtain several holes which can be filled with different *plasma* images. Using this procedure we can closely approximate a practical image histogram.

Two examples of images generated by the modified RMD algorithm, with 256 gray levels, are shown in Fig. 4. Fig. 5 shows the histogram corresponding to Fig. 4(a). For sake of comparison, Fig. 7 shows the histogram of the image in Fig. 6.

V. Experimental Results

Some design examples are presented to show the effectiveness of the proposed method. In all the following examples we assume $\alpha = 1$ in (1) and $\eta = 0.1$ in (5).

A. Example 1

In this example we try to store 50 gray-scale images of size 25×25, with $L = 16$ gray levels ($R = 4$ bits). The images are stored using four neural networks with $(25)^2 = 625$ neurons each.

The images are generated by the modified RMD algorithm, described in Section IV. Table I summarizes the results of this experiment. The first column shows the value of δ. The second column represents the maximum magnitude of the connection weights $|w_{ij}|_{\max}$. The number of bits needed to represent the weights is shown in the third column (see Section II). The last column shows the number of iterations needed for the convergence of the learning algorithm. In all the cases the 50 images were stored correctly (all the constraints were satisfied).

Increasing δ, more iterations are required for convergence, and an increasing number of bits is required to represent the weights (Table I). Hence, the learning time and the space (memory) required by the network, increase with δ. However, the increased weight precision, improves the recall, as it will be shown in the following examples. So, the choice of δ is a trade-off between network complexity and noise suppression capacity.

B. Example 2

The design objective is to store 32 images with 50×50 pixels, $L = 16$ gray levels, and then recall them starting from corrupted versions. The images were randomly generated by the modified RMD algorithm described above, and stored using four neural networks with 2500 neurons each. We used two different values for δ,i.e., 100 and 500; storage of the 32 images is accomplished in both cases. Then, we tried to recall the stored images, starting from a noisy version. Noisy initial states were generated by adding zero mean Gaussian noise, with standard deviation $\sigma = 1.5$, to each pixel of the stored images. We observed an improvement of error correction while increasing δ. Using $\delta = 100$, the stored images were correctly retrieved only in a few cases. Using $\delta = 500$ we obtained a probability of correct recall of about 98%.

TABLE I
Results of Example 1

δ	Maximum weight magnitude	Number of bits	Number of iterations
1	9.5	8	19
10	10	8	22
20	11.7	8	24
30	11.6	8	27
40	12.6	8	32
50	13.9	9	31
60	16.2	9	35
70	16.2	9	38
80	17.7	9	47
90	19	9	50
100	20.5	9	58
200	34.1	10	94
300	47.9	10	131

C. Example 3

The design objective is to store two images with 200×200 pixels and $L = 16$ gray levels. The images are *lenna* and *stefan*, shown in Fig. 8. Due to computer memory limitations, we partition each image into 16 parts, each of size 50×50. This way we obtain 32 images 50×50 which can be stored using four neural networks with 2500 neurons each. We used three different values of δ,i.e., 100, 250, 800; storage is accomplished with the values outlined in Table II.

Then, we tried to recall the stored images starting from a corrupted version. Noisy initial states were generated by adding to each pixel of the stored images, a zero mean Gaussian noise, with standard deviation σ. Some examples of noisy images are shown in Fig. 9.

Fig. 8. Images stored in example 3.

TABLE II
RESULTS OF EXAMPLE 3

δ	Maximum weight magnitude	Number of bits	Number of iterations
100	23.9	9	119
250	38.5	10	146
800	88.3	11	433

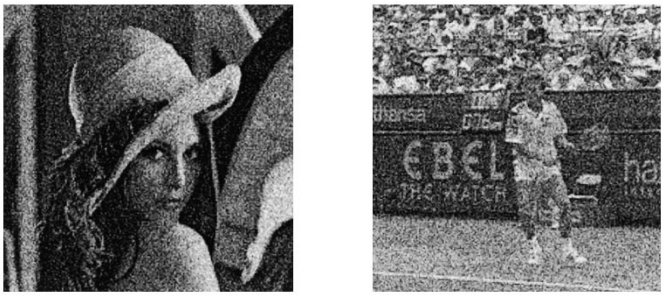

Fig. 9. Noisy versions of images in Fig. 8, with $\sigma = 1.7$.

The recall results are summarized below.

- $\delta = 100$, $\sigma = 1$. The four networks reached a stable state within a maximum of 56 iterations. Only 24 out of 32 images were correctly recalled.
- $\delta = 250$, $\sigma = 1$. The four networks reached a stable state within a maximum of seven iterations. All the 32 images were correctly recalled. By recombining them we obtain the two full-size images in Fig. 8.
- $\delta = 250$, $\sigma = 1.7$. The four networks reached a stable state within a maximum of 47 iterations. Only 14 images were correctly recalled.
- $\delta = 800$, $\sigma = 1.7$. The four networks reached a stable state within a maximum of ten iterations. All the 32 images were correctly recalled. By recombining them we obtain the two full-size images in Fig. 8.

VI. COMMENTS AND CONCLUSION

A neural architecture, storing gray-scale images, has been proposed. The design examples show a reliable image retrieval in presence of zero-mean additive Gaussian noise, for images with 16 gray levels. Simulation results, not included here, show a worse performance in the case of images with 256 gray levels.

Different methods have been proposed in the literature to store multivalued images into a neural associative memory. However, only few experimental results, on real images, are available, since these approaches have been investigated mainly from a theoretical viewpoint.

In the case of complex-valued networks, an estimation of storage capacity is available, but only for networks designed by the Hebb rule. The capacity depends on the ratio p/n, where p is the number of stored images. The storage probability, derived in [7], is comparable to our experimental results. However, the ratio p/n used in our experiments is quite low, due to computer memory limitations and computation time. An investigation on the capacity should be carried out with higher p/n ratios.

The main advantages of the proposed decomposition approach, with respect to existing methods, can be summarized as follows. Learning is easier, since several efficient and robust methods are available for binary neural networks; even some methods developed for bidirectional associative memories, as that proposed in [15], [16], could be adapted to this scope. The R networks evolve in parallel; so, a parallel implementation is possible both for storage and retrieval, giving the same learning and recall speed of a binary neural network with n neurons.

REFERENCES

[1] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones, "Distinctive features, categorical perception and probability learning: Some applications of a neural model," *Psych. Rev.*, no. 84, pp. 413–451, 1977.

[2] Y. Kamp and M. Hasler, *Recursive Neural Networks for Associative Memory*. Chichester, U.K.: Wiley, 1990.

[3] J. M. Zurada, *Introduction to Artificial Neural Systems*. St. Paul, MN: West, 1992.

[4] J. Si and A. N. Michel, "Analysis and syntehsis of discrete-time neural networks with multilevel threshold functions," in *Proc. ISCAS*, 1991, pp. 1461–1464.

[5] J. M. Zurada, I. Cloete, and E. van der Poel, "Generalized Hopfield networks for associative memories with multi-valued stable states," *Neurocomputing*, vol. 13, pp. 135–149, 1996.

[6] N. N. Aizenberg and I. N. Aizenberg, "CNN based on multi-valued neuron as a model of associative memory for grey-scale images," *Proc. IEEE Int. Workshop Cellular Neural Networks Applications*, pp. 36–41, 1992.

[7] S. Jankowski, A. Lozowski, and J. M. Zurada, "Complex-valued multistate neural associative memory," *IEEE Trans. Neural Networks*, vol. 7, pp. 1491–1496, Nov. 1996.

[8] S. Hui and S. H. Zak, "Dynamical analysis of the brain-state-in-a-box (BSB) neural models," *IEEE Trans. Neural Networks*, vol. 3, pp. 86–100, May 1992.

[9] W. E. Lillo, D. C. Miller, S. Hui, and S. H. Zak, "Synthesis of brain-state-in-a-box (BSB) based associative memories," *IEEE Trans. Neural Networks*, pp. 730–737, Sept. 1994.

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

[11] H. Y. Chan and S. H. Zak, "On neural networks that design neural associative memories," *IEEE Trans. Neural Networks*, vol. 8, pp. 360–372, Mar. 1997.

[12] J. Park, H. Cho, and D. Park, "On the design of BSB neural associative memories using semidefinite programming," *Neural Comput.*, no. 11, pp. 1985–1994, 1999.

[13] R. Perfetti and G. Costantini, "Multiplierless digital learning algorithm for cellular neural networks," *IEEE Trans. Circuits Syst. I*, vol. 48, pp. 630–635, May 2001.

[14] W.-C. Lau, A. Erramilli, J. L. Wang, and W. Willinger, "Self-similar traffic generation: The random midpoint displacement algorithm and its properties," in *IEEE Int. Conf. Communications, 'Gateway to Globalization'*, vol. 1, Seattle, 1995, pp. 466–472.

[15] Y. Wu and D. A. Pados, "A feedforward bidirectional associative memory," *IEEE Trans. Neural Networks*, vol. 11, pp. 859–866, July 2000.

[16] J. Park, C.-H. Kwon, and D. Park, "An optimization-based design procedure for asymmetric bidirectional associative memories," *IEEE Trans. Neural Networks*, vol. 12, pp. 169–170, Jan. 2001.