Bull. Nov. Comp. Center, Comp. Science, 9 (1998), 15–21

Discrete cellular neural networks for image processing

O. Bandman, S. Pudov, and A. Selikhov

A discrete time version of Cellular–Neural Network (DCNN) is a paradigm of neural networks that has the realistic perspective to be implemented in VLSI. An investigation of DCNN capabilities for image processing is attempted. It is done through detailed study of two typical concrete problems: 1) associative storing, retrieving and restoring written literals, 2) image differentiation and integration (constructing contours, isolines, shadows). A general theoretical background is given in brief, and computer simulations results are presented.

Introduction

Cellular–Neural Network (CNN) was introduced in [1] as an analog computing system. It is believed to be best suited for solving tasks, characterized by the locality of information needed by a certain cell to determine its future state. Most of the efforts have been directed to apply CNN for modeling gas and hydro dynamic which is usually represented by partial differential equations. A discrete version of CNN has also appeared [2], and, naturally, it was intended to solve tasks which traditionally belong to discrete domain. Nevertheless, the state of the art is at such a level that there is no complete answer on the questions: what are the capabilities of discrete cellular-neural networks in image processing, what is the dependence of solution quality of time and space complexity of DCNN, what may be the cost of hardware implementation? In this paper a partial answer to above questions is attempted through detailed study of two image processing problems:

1) associative retrieval of stored patterns (cellular version of the Hopfield neural network [3]) and
2) differentiation and integration of images [4] (contour extraction, isolines construction).

The above problems represent typical versions of DCNN. The first version is a discrete Hopfield's neural network with regular connection structure. Connection weight values depend on patterns to be stored and are determined by special learning procedure. The second CNN version is closer to the cellular automaton. It is completely homogeneous, since all cells have

not only identical connection structure, but also identical set of weight values, depending on the processing algorithm.

The paper contains five sections. In the second one main definitions and theoretical propositions are given. The third section is devoted to cellular-neural associative memory (CNAM), in it learning and retrieving algorithms are proposed and simulation results of recognizing distorted literal are presented. The fourth section contains an approach to the design of image processing algorithms and shows pictures obtained by simulation. The paper terminates with a Conclusion and References.

1. Formal representation of DCNN

A DCNN is a model of computation, which is performed over an array of *cells*, (sometimes referred to as *pixels*), each cell being characterized by a *state* and a *name*. States are from a finite set of numbers (representing colors), names are cell positions in the array. The array may be thought of as a network of cells, each cell being connected with a restricted amount of neighboring ones. Each connection is assigned by a real number (*connection weight*), whose value reflects the strength of two cells interaction. The image to be processed is given as an initial array. Computation is a synchronous iterative procedure. At each step every cell computes a threshold function of its neighbor cell states and changes (or does not change) its state according to function value. The computation stops, when all cell states remain unchanged, indicating that the system is in a stable state and the result of the problem is obtained. The main difficulty in DCNN synthesis is to provide the equivalence between stable states and solution results for any initial array state. It is achieved by determining proper connection weights which is done by synthesis or learning techniques.

Formally, a DCNN is defined by three notions: $\langle \boldsymbol{C}, \boldsymbol{W}, \boldsymbol{\Phi} \rangle$. Here $\boldsymbol{C}$ is a rectangular array of m rows and n columns of cells, represented by pairs $(a, (ij))$, where a is from an alphabet, (ij) is a cell name. $\boldsymbol{W} = \{W_{(ij)}\}$ is a set of *weight vectors* of the form $W_{ij} = (w_1, \ldots, w_q)$. w_k being the weight of the connection of the cell named (ij) with its k-th neighbor. $\boldsymbol{\Phi}$ is the set of instructions according to which DCNN operates. Cell neighborhood is defined by a *template*

$$T(ij) = \{\varphi_1(ij), \ldots, \varphi_q(ij)\}. \tag{1}$$

where $\varphi_k(ij)$ is a *naming function*, identifying the name of the k-th neighbor of the cell $(a, (ij))$ (Figure 1). An array, whose names are from $T(ij)$ and states are variables (see Figure 1), is called a *floating subarray*

$$S(ij) = \{(x_1, \varphi_1(ij)), \ldots, (x_q, \varphi_q(ij)\}. \tag{2}$$

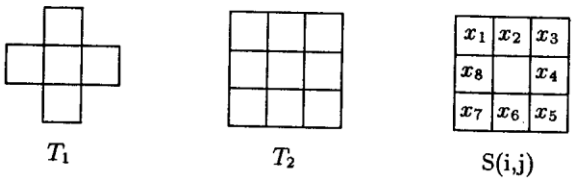

Figure 1. Spatial representation of templates T_1 and T_2, and a floating subarray generated by T_2

The value of $S(ij)$ when applied to a certain array C^l is a subarray of C^l, obtained by substituting the state and the name of $(a_k, \varphi_k(ij)) \in C^l$ for x_k and $\varphi_k(ij)$ in (2). In what follows it is more convenient to define cell neighborhood notions in vectorial form, $X_{ij} = (x_1, \ldots, x_q)$ and $Q_{ij} = (a_1, \ldots, a_q)$ being referred to as *variable neighborhood* and *state neighborhood* vectors, respectively. DCNN operates according to the following procedure.

DCNN operating procedure. Let $C(t)$ be the array at t-th step. Then:

- All cells in $C(t)$ compute the function

$$f(X_{ij}(t), W_{ij}(t)) = \begin{cases} b & \text{if } X_{ij}(t) * W_{ij}(t) > 0, \\ c & \text{otherwise,} \end{cases} \tag{3}$$

  and change their states as follows

$$a_{ij}(t+1) = f(X_{ij}(t), W_{ij}(t)). \tag{4}$$

  In (3) "$*$" denotes scalar multiplication, b, c belong to state alphabet.

- If $C(t+1) = C(t)$, then $C(t) = \Phi(C(0))$ is the result of the computation.

Determination of connection weights (DCNN synthesis procedure) proceeds from the requirement for wanted result to meet a network stability condition, which is expressed by the minimum of the *Liapunov function* [3],

$$E = -\sum_{ij} a_{ij}(W_{ij} * Q_{ij}), \tag{5}$$

Moreover, the strategy for constructing synthesis procedure should be grounded on the quest to provide two following dynamic properties of DCNN:

1) starting from any $C(0)$ the computing procedure should reach a stable state in a finite number of steps *termination property*,

2) the number of unwanted stable states should be minimal (minimum of *spurious states*).

Naturally, synthesis procedure strongly depends on the task DCNN should perform, what is shown in the following sections.

2. Storing and retrieving patterns of literals

Though the cellular algorithm for storing and retrieving distorted patterns, referred to as CNAM, is a cellular version of the well-known Hopfield's associative memory [3], no learning method elaborated for it, may be used for CNAM. There are two reasons. Firstly, they are based on solving matrix equations which is not efficient when the weight matrix is sparce. Secondly, the cellular learning algorithm is preferable, since it may be realized in a CNAM to be learned. Reasoning from this we have proposed a learning algorithm [5], based on Rosenblatt's concept of linear separability. The algorithm is as follows.

CNAM learning algorithm. Let $\boldsymbol{P} = P^0, \ldots, P^{L-1}$ be a sequence formed of patterns to be stored in CNAM.

Step 1. A sequence $\boldsymbol{P}' = P^1, P^2, \ldots$, is formed by repeating $\boldsymbol{P}$ many times and introducing a unique top indexing $t = 1, \ldots, (L-1), L, L+1, \ldots, L-1$ iterations dealing with all P^k from $\boldsymbol{P}$ being called a macroiteration.

Step 2. Initial values of vector W_{ij} components are chosen arbitrarily, a learning parameter α_{ij}^0 and a macroiteration counter τ are set to the given number U.

Step 3. A macroiteration starts. The value $\mu_{ij}^{\tau}(t)$ is set to ∞, then **Step 4** is executed for the whole macroiteration.

Step 4. Weight vectors W_{ij} for each (ij) are updated as follows:

$$W_{ij}^{t+1} = \begin{cases} W_{ij}^t, & \text{if } a_{ij}^t(Q_{ij}^t * W_{ij}^t) > \alpha_{ij}^t, \\ W_{ij}^t + a_{ij}^t Q_{ij}^t & \text{otherwise,} \end{cases} \tag{6}$$

$$\mu_{ij}^{\tau}(t+1) = \min\{(\mu_{ij}^{\tau}(t), a_{ij}^t(Q_{ij}^t * W_{ij}^t)\}. \tag{7}$$

Step 5. If $\tau = 0$ and no W_{ij} has been changed during the macroiteration, then the computation stops. Otherwise, if any W_{ij} is changed, then go to **Step 3**, if $\tau > 0$, then

$$\alpha_{ij}^{\tau-1} = \max(\alpha_{ij}^{\tau}, \mu_{ij}^{\tau}(t)). \tag{8}$$

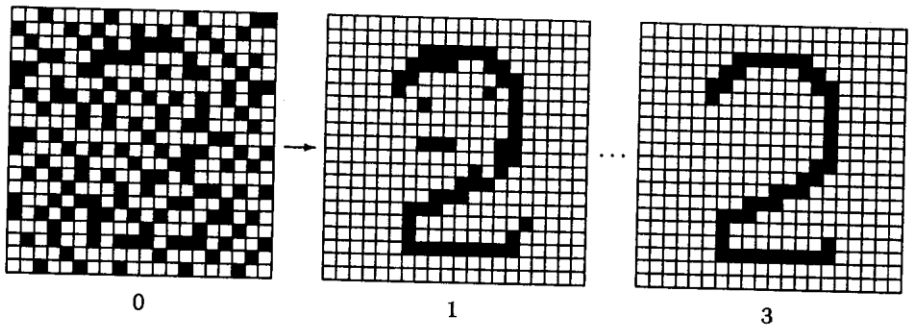

Figure 2. The process of retrieving the stored figure "2" when a distorted pattern is an input

In the above algorithm it is assumed that selfconnection weights $w_{ii} = 0$ for all cells. Nevertheless, theoretical analysis and simulation results show that selfconnection influence is essential. Proceeding from the requirements of minimizing a number of unwanted stable states the following expression for optimal selfconnection values is obtained:

$$w_{ii} = \min_{K}(W_{ij} * a_{ij}Q_{ij}^{K}), \tag{9}$$

The above learning and retrieving algorithms were applied to design and study a number of CNAMs, storing latin and russian literals and figures (Figure 2). The computations were simulated with the help of a special program package, called Animated Language Tools (ALT) [6], which makes run cellular algorithms in quasiparallel mode and provides facilities for observing and controlling computation process in any detail length level. Here are some results of experimental simulations, which allow to assess CNAM capabilities. The array with 20×20 bistable cells and $q = 5 \times 5$ template size has been learned to store up to 50 literals. Such an array restored 60–70 % of 1-distortions in input patterns (k-distortion of a pattern differs from it in k cell states). Introducing selfconnections according to (9) increased this figure up to 90 %. When storing 10 literals, restoration of all 1-distortions are surely guaranteed.

3. Image differentiation

A wide class of image processing tasks are based on marking the borders of areas, characterized by certain parameters, or, inversely, on coloring over the domains of cells having given properties. Such problems are referred to as differentiation and integration of images. These tasks are local by nature and, hence, all cells should perform one and the same operation,

i. e., for all cell weight vectors $W = (w_0, \ldots, w_q)$ are identical, (w_0 being a selfconnection weight).

The image Im to be processed is represented in discrete form as a cellular array, cell states representing colors (coded by natural numbers). A set of cells with identical states forms a pattern, so, that $Im = \{P_1, \ldots, P_m\}$. A pattern which is considered to be a background is formed by cells in the state denoted as a_0. The operation of discrete differentiation is the determination of difference in states of neighboring cells.

One of the most simple task in this class is the extraction of pattern contours. Two types of contours are of interest. The first is called a *dense* contour. It is defined as a subset $K_l \subseteq P_l$ of cells having in their neighborhoods formed by a template $T_1(ij)$ (Figure 1) at least one cell with a state $a_k \neq a_l$, a_l being the state of cells in P_l. The second type of contours referred to as a *minimal* one and is denoted as $K_l' \subseteq P_l$ is defined in the same way, but with the neighborhood generated by $T_2(ij)$. The extraction algorithm is based on changing states a_l into a_0 in those cells of the patterns P_l, which do not belong to K_l (K_l'). This is done in all cells synchronously according to procedure given in Section 2, where floating subarrays are generated by T_1 (T_2), the weight vectors are $W_1 = (8, -1, -1, -1, -1, -1, -1, -1)$, $W_2 = (4, -1, -1, -1)$ and the constants in (3) are $b = a_l$, $c = a_0$. It is easy to show that for any initial image the computation terminates in one iteration.

In real world problems images to be processed are accessible in bitmap format. So, the first stage of processing is to transform this information into cellular array representation. Then, to apply a DCNN algorithm to an array, it, sometimes, should be preliminary processed. And, the most difficult task is to determine template size and weight vectors. There are no formal synthesis methods for that. Usually, this is done by specifying the problem and trying to compose a system of inequalities with weights as arguments, or apply neural networks learning procedure. Hence, the use of

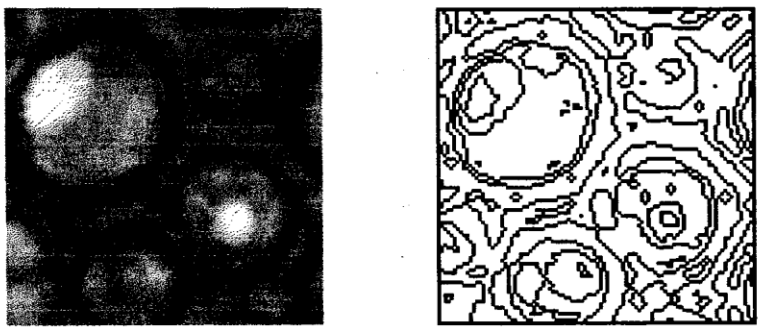

Figure 3. Image of a surface in bitmap format and the result of isolines construction

simulation tools when designing DCNN algorithm is absolutely necessary. For our investigations we elaborated our own program package called ALT (Animating Language Tools), which combines graphical and textual forms of cellular algorithm representation and allows to watch and control the computation process at any level of detail length. ALT has a wide range of facilities to perform all preliminary processing, as well as the algorithm itself in a quasiparallel mode.

As an example, the result of simulation isoline construction is shown in Figure 3. The algorithm consists of three procedures: 1) transforming the image from bitmap to ALT representation, 2) performing a preliminary discretisation of the image reducing the state alphabet to the set isolines labels, 3) executing contour extraction algorithm.

4. Conclusion

A wide class of image processing algorithms may be represented and solved based on DCNN model. In the paper a formalism is presented to describe image processing algorithms, and two typical algorithms (pattern retrieving and contour extraction) are investigated. It is shown theoretically and by simulation that DCNN is a very comprehensive and efficient model to design cellular image processing algorithms and special purpose processor architecture.

References

[1] L.O. Chua and L. Yang, *Cellular neural networks: theory and application*, IEEE Trans. Circuits and Systems, **CAS-35**, 1988, 1257–1290.

[2] H. Harrer and J.A. Nossek, *Discrete-time cellular neural networks*, International Journal of Circuit Theory Applications, **20**, 1993, 453–460.

[3] J.J. Hopfield and D.W. Tank, *Computing with neural circuits*, A Model. Science, **233**, 1986, 625–640.

[4] A.V. Selikhov, *Cellular algorithm for isoline extraction from a 2D Image*, Joint Bulletin of the Novosibirsk Computer Center and the Institute of Informatics Systems, series: Computer Science, Issue 6, 1997, 91–104.

[5] S. Pudov, *Learning of cellular neural associative memory*, Avtomteriya, 1997, No. 2, 107–120.

[6] Yu. Pogudin, *Simulation of fine-grained parallel algorithms with the ALT system*, Proceedings of First International Workshop on Distributed Interactive Simulation and Real Time Applications, 9–10 Jan., 1997, Eilat. IEEE Computer Society, 1997, 22–27.