Computing the Weights of Polynomial Cellular Neural Networks Using Quadratic Programming

Anna Rubi-Velez[1], Eduardo Gomez-Ramirez[2], and Giovanni E. Pazienza[3]

[1] Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Quatre Camins 2, 08022 Barcelona (Spain)
[2] LIDETEA, Posgrado e Investigacion, Universidad La Salle, Benjamin Franklin 47, Col. Condesa, 06140 Mexico City (Mexico)
[3] Cellular Sensory and Wave Computing Laboratory MTA-SZTAKI, Budapest (Hungary)
st12961@salle.url.edu,
egr2@ulsa.mx,
gpazienza@sztaki.hu

Abstract. Finding the weights of a Polynomial Cellular Neural/Nonlinear Network performing a given task is not straightforward. Several approaches have been proposed so far, but they are often computationally expensive. Here, we prove that quadratic programming can solve this problem efficiently and effectively in the particular case of a totalistic network. Besides the theoretical treatment, we present several examples in which our method is employed successfully for any complexity index.

Keywords: polynomial cellular neural networks, cellular automata, quadratic programming.

1 Introduction

Since the origin of the Cellular Neural Networks [1] in 1988, there is a natural concept relation with the Cellular Automata (CA) [2]. However, in spite of this relation, there are few papers that describe in a formal way the mathematical relations between them. Recently, it has been shown how nonlinear dynamics can give a new perspective of CA, by proving that Cellular Automata are a particular case of a more general paradigm called Cellular Neural Networks (CNNs) [3]. This approach has shed new light on CA, for which novel concepts – e.g., the index of complexity for one-dimensional CA – have been introduced. As proved in [4], the bridge between Cellular Automata and Cellular Neural Networks is the so-called Universal CNN cell (also known as Generalized CA). Nevertheless, the dynamic behavior of a Universal CNN cell can be also synthesized by means of other nonlinear functions, such as polynomials. On this ground, we put forward Polynomial CNNs (PCNNs), a cellular paradigm that has been already applied to several problems, including the exclusive OR and the Game of Life [5]. The main advantage of this model is that even the simplest implementation, single-layer and space-invariant weights, has the same computational power as a Universal Turing Machine but with the computational cost of the learning using genetic algorithm. In [6] the

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 645–652, 2009.

authors present the relationship between polynomial CNNs and two-dimensional CA providing a formal method to find the weights of the network through a simple procedure based on the properties of polynomials. The problem in this methodology is that is not easy to find the template if the order of the polynomial is more than 3. In this paper a new simple set of equations are presented to find the weights for any order of the polynomial using quadratic programming. With this model it is possible to find minimal solutions using the quadratic metric of the weights. The advantage is that for the specific case of modeling totalistic cellular automata it is not necessary to use genetic algorithm or other computational technique. The structure of the paper is the following: section 2, describes some cellular automata concepts; in section 3, PCNN's are introduced and in section 4, the quadratic programming approach is presented for any order of the PCNN. Finally section 5presents some conclusions.

2 Cellular Automata

Cellular Automata consist of regular uniform lattice of cells assuming a finite number of states; here, we consider two-dimensional CA in which cells are arranged in an eight neighbor rectangular grid and can take only two states. Cells are updated synchronously (discrete-time evolution) and the state of each cell at iteration $n + 1$ depends on the states of the cells in its neighborhood at iteration n. Similarly as before, we consider that the neighborhood of a cell is composed by the cell itself plus its eight nearest neighbors. Therefore, a two-dimensional Cellular Automaton is a discrete-time system with 9 binary inputs and 1 binary output; consequently, the dynamics of CA can be conveniently represented on a truth table containing $2^9 = 512$ rows. In total, there are $2^{2^9} = 2^{512}$ of such tables, also called rules, corresponding to all possible ways to evolve two-dimensional binary CA. We assume without loss of generality that at any fixed time the value of each cell can be either +1 (active cells) or -1 (non-active cells), exactly as in CNNs.

2.1 Totalistic Cellular Automata

The definition of a Totalistic Cellular Automaton is that the next state of a cell depends exclusively on the number of active cells in the neighborhood at the previous state. This means that, the position of the active cells in the neighborhood does not influence the result. The truth tables defining the rules of totalistic CA (see Table 1) have only 10 rows corresponding to all possible combinations of active and non-active cells in the neighborhood: from 0 (no active cell in the neighborhood) to 9 (all cells in the neighborhood are active).

There are only $2^{10} = 1024$ totalistic CA rules, (from $N = 0$ to $N = 1023$). The rule number is given by the following formula:

$$N = \sum_{i=0}^{9} \left(\frac{\beta_i + 1}{2}\right) 2^i \tag{1}$$

where N is the rule number, and the values of β can be retrieved from table 1. Rules of totalistic CA can be conveniently represented in a Cartesian coordinate system. This

Table 1. Truth table for a Totalistic Cellular Automata Rule

Active neighbors (Σ neighbors)	Output	Rule 51
0 (-9)	β_0	1
1 (-7)	β_1	1
2 (-5)	β_2	-1
3 (-3)	β_3	-1
4 (-1)	β_4	1
5 (+1)	β_5	1
6 (+3)	β_6	-1
7 (+5)	β_7	-1
8 (+7)	β_8	-1
9 (+9)	β_9	-1

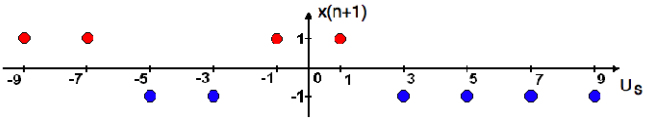

Fig. 1. CA dynamic diagram for totalistic rule 51

original representation was proposed in [6], in which the sum of the nine neighbors of the automaton is on the horizontal axis, and the corresponding output on the vertical axis. Given a rule (or equivalently, a truth table), for each of the 10 input patterns we need to depict a red, in case of a firing pattern; blue, in case of a quenching pattern. For example, the rule corresponding to the truth table in Table 2 is:

$$N = \sum_{i=0}^{9} \left(\frac{\beta_i + 1}{2}\right) 2^i = 1 + 2 + 16 + 32 = 51 \tag{2}$$

and its dynamic diagram is in Fig. 1

3 Polynomial Cellular Neural Network

The Polynomial Cellular Neural Networks (PCNN) was first introduced in [7] and other models can be found in [8]. The general form for PCNN in discrete time is:

$$x(n+1) = A * Y(n) + B * U + z + P_2(u, y) + \cdots + P_m(u, y) \tag{3}$$

where *m* is the order of the PCNN. Considering the previous work [6], it is possible to find through a rigorous method the parameters – degree of the polynomial and the weights of the network – of a one-layer space-invariant Polynomial Cellular Neural Network implementing a totalistic CA rule. When implementing a totalistic CA, the mathematical representation can be simplified thanks to two preliminary considerations about the nature of the problem. First of all, since the output of a CA local rule depends exclusively on the input, also the output of the network has to be a function of the input pattern only. This means that no matrix, except for those convolving the input U and its multiples, can have non-zero elements other than the central one. In other words, in the model of previous equation all matrices convolving the output *Y* and its multiples must have the central element only. For instance,

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 0 \end{bmatrix} = a \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

and, consequently:

$$A * Y = ay_c \tag{5}$$

where the subscript c emphasizes that the only value to take into account is the central one. In totalistic CA all neighbors are considered at once, and hence there is also no reason for making a distinction among the different values of the matrices convolving the input U and its multiplies. For instance,

$$B = \begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix} = b \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6}$$

and, hence the convolution $B * U$ can be simplified as:

$$B * U = b \sum_{(k,l) \in N(i,j)} u_{kl} = bu_s \tag{7}$$

where the subscript s emphasizes that the value u_s results from the sum of all nine neighbors. In conclusion, Eq. 3 can be rewritten as a function of u_s and y_c as follows:

$$x(n+1) = ay_c(n) + bu_s + z + P_2(u_s, y_c) + \cdots + P_m(u_s, y_c) \tag{8}$$

where $u_s \in$ {-9,-7,-5,-3,-1, 1, 3, 5, 7, 9}, which are the values obtainable by summing nine values ± 1, and $y_c \in$ {-1,+1}, by definition of discrete-time PCNN.

3.1 First Order PCNN

The first order PCNN is equivalent to a standard CNN, whose state equation, using the notation just introduced, is:

$$x(n+1) = a_1 y_c(n) + b_1 u_s + z \tag{9}$$

where the subscript 1 added to the network weights a and b means that we consider a first order model.

$$y_\infty = \begin{cases} 1 \Leftrightarrow b_1 u_s + z \geq 0 \\ -1 \Leftrightarrow b_1 u_s + z < 0 \end{cases} \tag{10}$$

From the previous equations, it is easy to see that when $a_1 > 0$ the network is stable, and it converges to the steady state y_∞.

Choosing properly the weights b_1 and z, Eq. 9 describes any line. Therefore, a first degree PCNN can solve only linearly separable problems, or equivalently, perform all linearly separable totalistic CA rules. Note that given a totalistic CA rule N there are infinite ways of setting b_1 and z to obtain it; however, fixed values of b_1 and z define univocally a totalistic CA rule. An example of application of first degree PCNN is illustrated in Fig. 2.

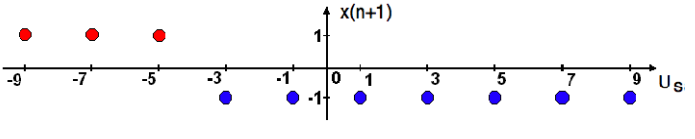

Fig. 2. Rule 7 can be implemented by a first order Polynomial CNNs

3.2 Second Order PCNNs

The generic form for the second degree polynomial $P_2(u_s, y_c)$ is:

$$\begin{aligned} \mathrm{P}_2(u_s, y_c) &= \sum_{i=0}^{2} p_i u_s^i \cdot q_i y_c^{2-i} \\ &= p_0 u_s^0 \cdot q_0 y_c^2 + p_1 u_s \cdot q_1 y_c + p_2 u_s^2 \cdot q_2 y_c^0 \\ &= p_1 q_0 + p_1 q_1 u_s y_c + p_2 q_2 u_s^2 \end{aligned} \tag{11}$$

where in the last step we used the property that $u_s^0 = 1$ and $y_c^i = 1$, for i even, and $y_c^i = y_c$, for i odd. The term $p_0 q_0$ is constant and hence it can be absorbed into the bias z (with abuse of notation, $z + p_0 q_0 = z$) and the expression can be further simplified through the substitutions $p_1 q_1 = a_2$ and $p_2 q_2 = b_2$. Finally, we can add this expression to previous equation obtaining:

$$\begin{aligned} x(n+1) &= a_1 y_c(n) + b_1 u_s + z + a_2 u_s y_c(n) + b_2 u_s^2 \\ &= (a_2 u_s + a_1) y_c(n) + b_2 u_s^2 + b_1 u_s + z \end{aligned} \tag{12}$$

where $a_1, a_2, b_1, b_2, z \in \Re$. If the term $(a_2 u_s + a_1) > 0$, the network is always stable, and it converges to the steady state y_∞

$$y_\infty = \begin{cases} 1 & \Leftrightarrow b_2 u_s^2 + b_1 u_s + z \geq 0 \\ -1 & \Leftrightarrow b_2 u_s^2 + b_1 u_s + z < 0 \end{cases} \tag{13}$$

With this model, the PCNN is capable of implementing any totalistic CA rule whose firing and quenching patterns in the CA dynamic diagram can be separated by two lines. An example of application of second order PCNN is in Fig. 3: this CA dynamical diagram describes the behavior of rule 12.

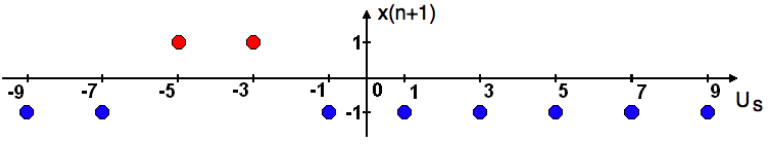

Fig. 3. Rule 12 can be implemented by a second order Polynomial CNNs

3.3 Third Order PCNNs

With the same considerations as before about u_0 and y_i, and through the substitutions, the general equation for the third order PCNN implementing totalistic CA can be described as:

$$\begin{aligned} x(n+1) &= (a_2 u_s + a_1) y_c(n) + b_2 u_s^2 + b_1 u_s + z + \mathrm{P}_3(u_s, y_c) \\ &= (a_3 u_s^2 + a_2 u_s + a_1) y_c(n) + b_3 u_s^3 + b_2 u_s^2 + b_1 u_s + z \end{aligned} \tag{14}$$

Where $a_1, a_2, a_3, b_1, b_2, b_3, z_2 \in \Re$. Considering $a_3 u_s^2 + a_2 u_s + a_1 > 0$, which ensures the stability of the PCNN, the network output can be computed as:

$$y_\infty = \begin{cases} 1 & \Leftrightarrow b_3 u_s^3 + b_2 u_s^2 + b_1 u_s + z \geq 0 \\ -1 & \Leftrightarrow b_3 u_s^3 + b_2 u_s^2 + b_1 u_s + z < 0 \end{cases} \tag{15}$$

which means this kind of network is capable of implementing any totalistic CA rule whose firing and quenching patterns in the CA dynamic diagram can be separated by three lines. An example of application of second degree PCNN is illustrated in Fig. 1: according to the notation introduced before, this CA dynamical map describes the behavior of rule 51.

3.4 M-th Order PCNN and Continuous-Time Model

The procedure illustrated previously can be repeated for higher degree polynomials, obtaining similar results: the PCNN state equation of any order contains a term multiplying $y_c(n)$, controlling the dynamics of the network, and another term which depends only on us and the network weights, determining the steady state of the network. The last term can be used to interpolate the points of the CA dynamic pattern, and the resulting linear system has always a solution since it is Vandermonde-like. Therefore, we can conclude that the complexity index κ of a totalistic CA rule corresponds to the order of the PCNN model implementing it. It can be also noticed that the complexity index in totalistic CA is equal to the number of lines separating firing and quenching patterns in the CA dynamic diagram; hence, the minimum value for κ is 0 (rule 0 and rule 1023) and its maximum value is 9 (rule 341 and rule 682, see Fig. 6).

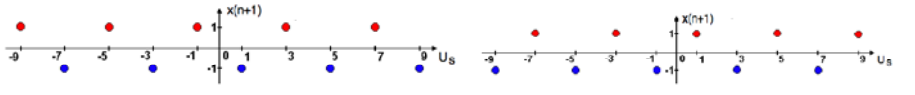

Fig. 4. Totalistic CA rules having complexity index κ = 9: (a) rule 341 and (b) rule 682

4 Computing the Weights Using Quadratic Programming

Defining the set of templates for a second order PCNN as:

$$w^T = \{a_2, a_1, b_2, b_1, z\}$$

Equation (13) can be translated to:

$$y_\infty = \begin{cases} 1 \Rightarrow b_2 \left(u_s^+\right)^2 + b_1 u_s^+ + z \geq 0 \\ -1 \Rightarrow b_2 \left(u_s^-\right)^2 + b_1 u_s^- + z < 0 \end{cases} \tag{16}$$

Where u_s^+ and u_s^- are the values that corresponds to outputs *x(n+1)* 1 and -1 respectively. For example, for rule 7, $u_s^+ = \{-9, -7, -5\}$ and $u_s^- = \{-3, -1, 1, 3, 5, 7, 9\}$.

For the general case it is possible to write the stability condition as:

$$\sum_{i=1}^{m} a_i u_s^{i-1} > 0 \tag{17}$$

and:

$$y_\infty = \begin{cases} 1 \Leftrightarrow \sum_{i=1}^{m} b_i u_s^i + z \geq 0 \rightarrow b_1 u_s^+ + b_2 (u_s^+)^2 + \ldots + b_m (u_s^+)^m + z \geq 0 \\ -1 \Leftrightarrow \sum_{i=1}^{m} b_i u_s^i + z < 0 \rightarrow -b_1 u_s^- - b_2 (u_s^-)^2 - \ldots - b_m (u_s^-)^m - z > 0 \end{cases} \tag{18}$$

Considering the previous equation, it is possible to define the following optimization quadratic programming problem:

$$\min_w w^T H w \tag{19}$$

$$\begin{bmatrix} A \\ A_e \end{bmatrix} w \begin{matrix} > \\ \geq \end{matrix} \begin{bmatrix} d \\ d_e \end{bmatrix} \tag{20}$$

The first order case can be written in matrix form as:

$$\begin{bmatrix} u_s & 1 & 0 & 0 & 0 \\ 0 & 0 & -\left(u_s^-\right)^2 & -u_s^- & -1 \\ 0 & 0 & \left(u_s^+\right)^2 & u_s^+ & 1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ b_2 \\ b_1 \\ z \end{bmatrix} \begin{matrix} > \\ > \\ \geq \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

and the general case as:

$$\begin{bmatrix} u_s^{(m-1)} & u_s^{(m-2)} & \ldots & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & -\left(u_s^-\right)^m & -\left(u_s^-\right)^{(m-1)} & \ldots & -u_s^- & -1 \\ 0 & 0 & \ldots & 0 & \left(u_s^+\right)^m & \left(u_s^+\right)^{(m-1)} & \ldots & u_s^+ & 1 \end{bmatrix} \begin{bmatrix} a_m \\ a_{m-1} \\ \ldots \\ a_1 \\ b_m \\ b_{m-1} \\ \ldots \\ b_1 \\ z \end{bmatrix} \begin{matrix} > \\ > \\ \geq \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{22}$$

With the previous representation is possible to use any algorithm to solve quadratic programming equations. Adapting the previous equation to the quadratic programming function of the Matlab, we use and ε approximation for the first two rows such as: $aw > 0 \rightarrow aw \geq \varepsilon, \forall \varepsilon \rightarrow 0$.

Table 2. Templates obtained using PCNN

Rule	m	ε	Weights
7	1	1e-9	1.0e-008* [0.1000 -0.0500 -0.2500]^T
12	2	1e-9	1.0e-008 *[0 0.1000 -0.0125 -0.1000 -0.1875]^T
51	3	1e-9	1.0e-009 *[0 0 1 -0.1042 -0.5629 0.1042 0.5629]^T
682	9	1e-9	1.0e-009*[0.3354 -0.1693 0.3354 -0.1693 0.3354 -0.1693 0.3354 -0.1693 0.3354 0 0 -0.0020 0 0.0593 0 -0.5079 0 0.9507 -0.5000]^T

5 Conclusions

The computational cost of polynomial cellular neural networks training can be improved using the previous methodology. For these cases is not necessary to use genetic algorithm to find the solutions. It is possible to use some variations of the linear or quadratic programming approach to find solutions in the integer domain or with a specific resolution. Using the results presented in [4] and with the results presented in this work, a similar approach can be obtained for semitotalistic cellular automata rules.

References

1. Chua, L.O., Yang, L.: Cellular neural networks: Theory and Applications. IEEE Trans. Circuits Syst. 35, 1257–1290 (1988)
2. von Neumann, J.: The general and logical theory of automata. In: Jeffress, L.A. (ed.) Cerebral Mechanisms in Behavior–The Hixon Symposium. s.l. John Wiley, Chichester (1951)
3. Chua, L.O., Yook, S., Dogaru, R.: A nonlinear dynamics perspective of Wolfram's new kind of science. Part I: Threshold of complexity. International Journal of Bifurcation and Chaos 12, 2655–2766 (2002)
4. Dogaru, R., Chua, L.O.: Universal CNN cells. International Journal of Bifurcation and Chaos 9(1), 1–48 (1999)
5. Pazienza, G.E., Gomez-Ramirez, E.y., Vilasis-Cardona, X.: Portugal Polynomial discrete time cellular neural networks for implementing the Game of Life.: s.n. In: Proc. International Conference on Artificial Neural Networks, ICANN 2007 (2007)
6. Giovanni, E., Gomez-Ramirez, E.: New properties of 2D Cellular Automata found through Polynomial Cellular Neural Network. In: International Joint Conference on Neural Networks (IJCNN 2009), Pazienza, Atlanta, USA (2009)
7. Schonmeyer, D., Feiden, D., Tetzlaff, R.: Multi-template training for image processing with cellular neural networks. In: Proc. of 2002 7th IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA 2002), Frankfurt, Germany (2002)
8. Pazienza, G., Gomez-Ramirez, E., Vilasis-Cardona, E.: Polynomial discrete time cellular neural networks to solve the XOR problem. In: Proc. 10th International Workshop on Cellular Neural Networks and ther Applications (CNNA 2006), X. Istanbul, Turkey (2006)