CNNA-94 Third IEEE International Workshop on Cellular Neural Networks and their Applications
Rome, Italy, December 18-21, 1994

Design and Learning with Cellular Neural Networks

Josef A. Nossek
Institute for Network Theory and Circuit Design,
Technical University Munich, Munich, Germany
e-mail: nossek@nws.e-technik.tu-muenchen.de

Abstract – *The template coefficients (weights) of a CNN, which will give a desired performance, can either be found by design or by learning. "By design" means, that the desired function to be performed could be translated into a set of local dynamic rules, while "by learning" is based exclusively on pairs of input and corresponding output signals, the relationship of which may be by far too complicated for the explicit formulation of local rules. An overview of design and learning methods applicable to CNNs, which sometimes are not clearly distinguishable, will be given here. Both technological constraints imposed by specific hardware implementation and practical constraints caused by the specific application and system embedding are influencing design and learning.*

1 Introduction

Since their introduction in 1988 [1] the design of both continuous-time and discrete-time cellular neural networks (CT-CNNs and DT-CNNs) has been an interesting research topic. The aim is to find a set of parameters (coefficients, synaptic weights), which in the case of locally connected translationally invariant CNNs are usually called templates, so that the network performs according to a given task. The equation for each cell c of CT-CNN is as follows:

$$\begin{aligned} \dot{x}_c &= -x_c + \sum_{d\in\mathcal{N}_r(c)} a_{d-c} y_d + \sum_{d\in\mathcal{N}_r(c)} b_{d-c} u_d + i_c\,, \\ y_c &= f(x_c)\,, \\ f(x) &:= \tfrac{1}{2}(|x+1| - |x-1|)\,, \end{aligned} \tag{1}$$

while for a DT-CNN

$$\begin{aligned} x_c(k) &= \sum_{d\in\mathcal{N}_r(c)} a_{d-c} y_d(k) + \sum_{d\in\mathcal{N}_r(c)} b_{d-c} u_d + i_c\,, \\ y_c(k) &= f(x_c(k-1))\,, \\ f(x) &:= \operatorname{sgn}(x)\,. \end{aligned} \tag{2}$$

The symbolic notation a_{d-c} and b_{d-c} of the feedback and control coefficients indicates that only the relative position of cells within a neighborhood $\mathcal{N}_r$ determines the connection weight.

The first useful templates have been derived in analogy to known image processing algorithms, while the first systematic approach for the design of CT-CNNs was aiming at programming desired fixed points [2] (see Section 2). This technique has later been adapted to the discrete-time case in [3] (see Section 3), and it requires the a-priori knowledge of the trajectories. Modified versions of recurrent backpropagation and backpropagation-through-time have been developed [4] to make sure, that the CT-CNN will not only have the desired fixed point, but evolve from a given initial condition (e.g. input image) into the corresponding fixed point (output image) along a desired trajectory. While all the aforementioned techniques require the intuition of an experienced designer in choosing proper training patterns and specifying the local dynamics, the approach described in Section 5 [5] for DT-CNNs leaves the choice of the trajectories to an optimization procedure. It is therefore the only (global) learning procedure in the strict sense.

For such global learning approaches the question arises, how many samples (input-output pairs) are necessary for reliable generalization. In [6], an upper bound on the sample size is derived by applying the probably approximately correct (PAC) learning theory to DT-CNNs.

Finally the optimization of the nominal parameters of a CNN, which has been designed with one of the previous procedures, with respect to parameter tolerances as well as pattern disturbances is treated (Section 6) [7]. This is already a step towards taking into account the hardware constraints at the design or learning stage. The approach in [8] is even proposing the use of modified network equations for the actual behavior of a simplified CNN hardware.

Multilayer CNNs, where a sequence of operations (various virtual layers) is carried out on one programmable physical layer taking advantage of in-place computations [9], are first broken down into individual tasks by the intuition and the experience of the designer and then being dealt with as in single layer CNNs above.

2 Designing Fixed Points

In this section, the issue of designing fixpoints x^∞ of a CT-CNN, specified by the corresponding output $y^\infty = f(x^\infty)$ in the saturation region, is discussed. Given an output in the saturation region $|y_c^\infty| = 1$ and a fixed input u_c, the corresponding state must be given by

$$x_c^\infty := \sum_{d\in\mathcal{N}_r(c)} a_{d-c}y_d^\infty + \sum_{d\in\mathcal{N}_r(c)} b_{d-c}u_d + i_c \tag{3}$$

since the derivative has to vanish. One still has to make sure that the output of the cell c Eq. 3 really is given by the desired y^∞, which is equivalent to

$$\left.\begin{array}{l} x_c^\infty \geq +1 \quad \text{if} \quad y_c^\infty = +1 \\ x_c^\infty \leq -1 \quad \text{if} \quad y_c^\infty = -1 \end{array}\right\} \Longleftrightarrow y_c^\infty \left(\sum_{d\in\mathcal{N}_r(c)} a_{d-c}y_d^\infty + \sum_{d\in\mathcal{N}_r(c)} b_{d-c}u_d + i_c \right) \geq 1 \tag{4}$$

for each cell c of the network ($c = 1, \cdots, M$). In general, one has $L > 1$ desired fixed points $y^{\infty[l]}$ along with some input patterns $u^{[l]}$ ($l = 1, \cdots, L$). For each pair of training patterns, one obtains the system of affine inequalities Eq. 4 for the unknowns a, b, and i [2]. This can now be solved by many methods, e.g. the relaxation method in [10], the perceptron algorithm [11], Rosenblatt's algorithm [12] and the AdaTron algorithm [13], just to mention a few. For each algorithm there is a convergence theorem stating that, if a solution exists, the algorithm finds a solution. In some applications (e.g. image processing), rotationally invariant or isotropic templates are needed. All of the above mentioned algorithms can be adapted to incorporate these additional equality constraints [2], [14].

Simply replacing Eq. 4 by

$$\left.\begin{array}{l} x_c(k_\infty) \geq 0 \quad \text{if} \quad y_c(k_\infty) = +1 \\ x_c(k_\infty) < 0 \quad \text{if} \quad y_c(k_\infty) = -1 \end{array}\right\} \Longleftrightarrow$$
$$\Longleftrightarrow y_c(k_\infty)\left(\sum_{d\in\mathcal{N}_r(c)} a_{d-c}y_d(k_\infty) + \sum_{d\in\mathcal{N}_r(c)} b_{d-c}u_d + i_c\right) > 0 \tag{5}$$

with some k_∞ large enough for the network to settle at a fixed point will give the inequalities to program the fixed points of a DT-CNN.

In both cases (Eq. 4 and Eq. 5), the initial condition $x(0)$ or $y(0)$ is not involved in the learning of fixed points. Therefore, no control of the basins of attraction of these fixed points is provided. In [15], a step towards taking into account initial conditions is made, but this approach works reliably only, if the transients are simply monotonic.

3 Design of DT-CNNs with Prescribed Trajectories

Gradient-based methods are not applicable to DT-CNNs, since error gradients do not exist everywhere in the space of the network parameters. The reason for this is the hard threshold function used as the nonlinearity. The advantage is, that the transition from $\mathbf{y}(k)$ to $\mathbf{y}(k+1)$ can be described by linear inequalities. Hence the methods described in Section 2 can be used, though one has to be willing and able to prescribe a sensible trajectory $\mathbf{u}, \mathbf{y}(0), \cdots, \mathbf{y}(T)$. From the recursion Eq. 2, the following set of inequalities can be derived for each time step $k = 0, \cdots, T-1$:

$$\left.\begin{array}{l} x_c(k) \geq 0 \;\text{ if }\; y_c(k+1) = +1 \\ x_c(k) < 0 \;\text{ if }\; y_c(k+1) = -1 \end{array}\right\} \Longleftrightarrow$$
$$\Longleftrightarrow y_c(k+1)\left(\sum_{d\in\mathcal{N}_r(c)} a_{d-c}y_d(k) + \sum_{d\in\mathcal{N}_r(c)} b_{d-c}u_d + i_c\right) > 0\,. \tag{6}$$

Again, more than one trajectory can be prescribed, and one can replace the inequality "> 0" in the above equation by "$\geq R$" to ensure some kind of robustness of the solution [3]. This does not change the solvability of the system since by appropriately scaling a solution of the original system one obtains a solution of the new system. This reflects the fact that the space of solutions of a general system of inequalities $\mathcal{L} := \left\{\mathbf{p} \in \mathbb{R}^N : \mathbf{p}^t\mathbf{v}^{[l]} > R \geq 0; \mathbf{v}^{[l]} \in \mathbb{R}^N \text{ for } 1 \leq l \leq L\right\}$ is a convex cone. By increasing the value of R, the vertex of the cone is moved away from the origin. A precise definition of the robustness of a solution $\mathbf{p} \in \mathcal{L}$ and how the most robust solution is obtained will be discussed in Section 6.

An example for the application of Eq. 6 for extracting the edges of an image and simultaneously suppressing the noise is given in the following. It is remarkable, how simple the learning samples (Fig. 1) are, and how well this works for quite general images (Fig. 2).

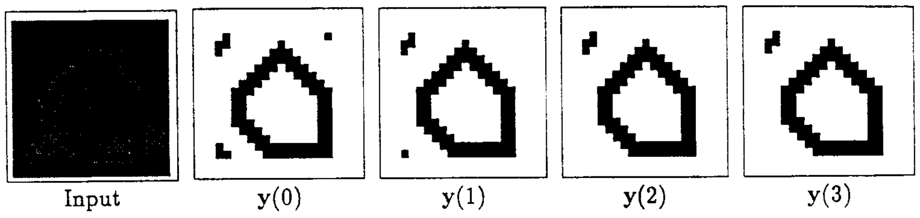

Figure 1: Learning Samples for Edge Detection

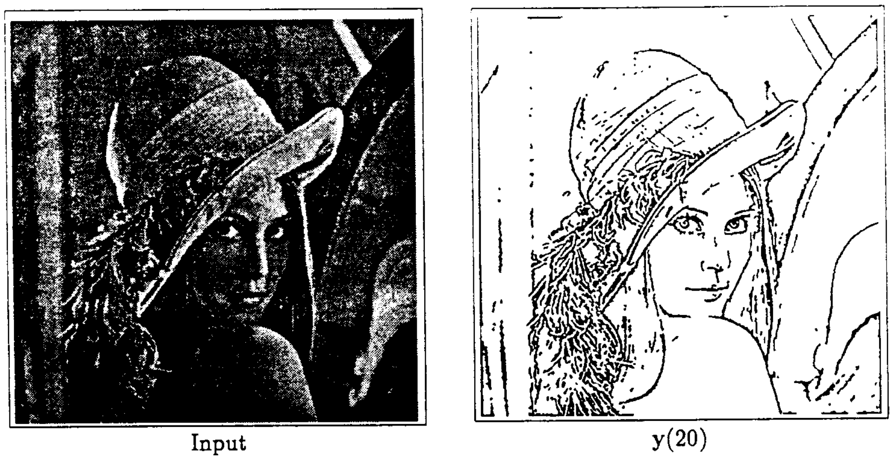

Figure 2: Edge Detection on "Lena" image

4 Gradient Based Methods for Learning CT-CNNs with Prescribed Trajectories

The design of fixed points, however, does not guarantee the correct behavior of the dynamical system, since the initial states do not necessarily lie in the basins of attraction of the correct fixed points. It is thus necessary to find a parameter vector $\mathbf{p} = (a, b, i)$ such that the output of the CNN equals the desired output $\mathbf{d}^{[l]}(\infty)$ starting with a given initial state $\mathbf{x}^{[l]}(0)$ and input $\mathbf{u}^{[l]}$ for all training patterns $(l = 1, \ldots, L)$.

A common way for learning in neural networks is to define an error measure or cost function of the fixed points and the desired outputs (*Recurrent Backpropagation* [16]) or in general of the trajectory of the system and the desired trajectory (*Backpropagation-Through-Time* [17]).

$$E(p) = \sum_{l=1}^{L} \sum_{c=1}^{M} E_c(x_c^{[l]}, d_c^{[l]}) \,, \tag{7}$$

The gradient of this error with respect to the weights can then be used to descend to a local minimum of the error.

$$\frac{\partial E}{\partial p} = \sum_{l=1}^{L}\sum_{c=1}^{M} \frac{\partial E_c}{\partial p}(x_c^{[l]}, d_c^{[l]}) \,. \tag{8}$$

For the sake of notational simplicity, we will omit the index $[l]$, since the gradient is simply summed over all learning samples $l = 1, \dots, L$.

Due to the piecewise linear output function, it is better to define the error as a function of the states instead of the output [4]. With the following function with a parameter R

$$e(v|R) = \begin{cases} \frac{1}{k}|v-(1+R)|^k \,, & \text{if } v < +(1+R)\,; \\ 0\,, & \text{else }\,, \end{cases} \tag{9}$$

the state-based distance and the partial derivative are given by

$$E_c(x_c, d_c) = e(x_c d_c|R)\,, \quad \frac{\partial E_c}{\partial p} = e'(x_c d_c|R) d_c \frac{\partial x_c}{\partial p}\,. \tag{10}$$

The error of a cell is zero, whenever a cell is in the proper saturation region of the output function having at least a distance of R to the boundary of this region.

Recurrent Backpropagation (RBP) [16] is a generalization of the well-known Backpropagation algorithm to learn the fixed points of recurrent neural networks. The error is taken at the fixed points, assuming a fixed point is reached:

$$E_c(p) = e(x_c(\infty)d_c|R)\,, \tag{11}$$

and the equations for RBP read:

$$\frac{\partial E_c}{\partial p} = \lambda_c \frac{\partial F_c}{\partial p}\bigg|_{t\to\infty}\,, \quad \dot{\lambda}_c = -\lambda_c + \sum_{d\in\mathcal{N}_c} a_{c-d} f'(x_c(\infty))\lambda_d + e'(x_c(\infty)d_c|R)d_c\,, \tag{12}$$

where F_c is the right-hand side of Eq. 1. $\lambda \in \mathrm{IR}^M$ is an "error signal" vector, which is computed from the associated dynamical system, with any initial condition $\lambda_c(0)$. Thereby, the ODEs for λ (the associated dynamical system) are simply introduced to avoid a matrix inversion, which would be necessary otherwise. If the algorithm succeeds in finding a suitable parameter vector, not only the fixed points of the dynamical system are learned, but also the trajectories from the given initial states to the desired fixed points.

The problem with RBP is that the algorithm breaks down, if the CNN becomes unstable during some step of the learning procedure. To avoid this dilemma, Backpropagation-Through-Time has been introduced.

With *Backpropagation-through-Time (BTT)* [17],[18], not only fixed points, but also prescribed trajectories can be learned. The gradient of the state-based error can be simplified

$$E_c(p) = e_1(x_c(T)d_c|R)\int_0^T e_2(x_c(t)d_c|R)dt\,, \qquad \frac{\partial E_c}{\partial p} = \int_0^T \lambda_c \frac{\partial F_c}{\partial p}\, dt \tag{13}$$

using the associated dynamical system

$$\dot{\lambda}_c = \lambda_c - \sum_{d\in\mathcal{N}_c} a_{c-d} f'(x_c(t))\lambda_d - e_1(x_c(T)d_c|R)e_2'(x_c(t)d_c|R)d_c\,, \tag{14}$$

which has to be integrated backward in time, since the boundary value of λ_c is known at the terminal time T:

$$\lambda_c(T) = e_1'(x_c(T)d_c|R)d_c \int_0^T e_2(x_c(t)d_c|R)dt\,. \tag{15}$$

Depending on the choice of e_1 and e_2, BTT can be used to follow a prescribed trajectory $d_c^{[l]}(t)$, or to gain information from the trajectory to find a parameter vector, for which the system converges in a given time T to the desired output.

One problem in common with all gradient-based learning algorithms is that only local minima of the error surfaces are found. Therefore, the result depends on the selected initial parameter. This is true, although the state-based versions of RBP and BTT, which are described here, are much better in this respect when compared with their output-based counterparts [4].

For both algorithms, versions applicable to DT-CNNs are also available [19], provided that their threshold nonlinearity is replaced by a continuously-valued one.

5 Global Learning for DT-CNNs

In *global* learning algorithms, the task, which has to be learned by the network, is defined by a set of input images (training patterns) and the corresponding desired output images of the network. The input images are inputs for the whole network as opposed to local cell input patterns in *local* learning algorithms. The global learning algorithm is used to find the network parameters for this task, which implies that the algorithm itself designs the trajectory. Thus much more complicated trajectories are obtainable, and more complicated tasks can be implemented by the network. Unfortunately, global learning algorithms are computationally expensive. Following from the results in [20], it can be concluded that global learning for DT-CNNs belongs to the class of NP-complete problems [5].

All different variants of global learning algorithms are based on the idea that an objective function (cost function) is defined, which measures how well the network maps a set of input images onto the desired output images. Learning is thus achieved by minimizing the cost function.

DT-CNNs have two stable output behaviors: either they run into a stable fixed point, or they perform stable limit cycles (oscillations). In many applications, oscillations cannot be tolerated, and thus they have to be punished by the objective function.

Let $\mathbf{p}$ be the parameter vector, which contains the template coefficients of the DT-CNN. A distance measure $\Delta^{[l]}(\mathbf{p})$ and the cost function $o(\mathbf{p})$ are defined as follows:

$$\Delta^{[l]}(\mathbf{p}) = \begin{cases} \frac{1}{4}\sum_{c=1}^{M} \omega_c \cdot (y_{c,\mathbf{p}}^{[l]}(\infty) - d_c^{[l]})^2 & \text{for stable output fixed points} \\ 1 & \text{for stable limit cycles} \end{cases} \tag{16}$$

$$o(\mathbf{p}) = \sum_{l=1}^{L} \Omega_l \Delta^{[l]}(\mathbf{p}) \tag{17}$$

The $\omega_c \in [0,1]$ and $\Omega_c \in [0,1]$ are weighting factors, which obey

$$\sum_{c=1}^{M} \omega_c = 1 \quad \text{and} \quad \sum_{l=1}^{L} \Omega_l = 1\,.$$

L is the number of training patterns, and M is the number of cells in the network. $y_{c,\mathbf{p}}^{[l]}(\infty)$ denotes the output of cell c, when input image $\mathbf{u}^{[l]}$ was fed into the network, and the network has reached a stable fixed point. $\mathbf{d}^{[l]}$ is the corresponding desired output image of the network.

In some applications, moderate oscillations can actually be tolerated. In this case, it makes sense to use a modified distance measure $\tilde{\Delta}^{[l]}(\mathbf{p})$, in which the distances between the actual and the desired output image are averaged over one period of the limit cycle.

Due to the inherently nonlinear behavior of a DT-CNN cell (caused by the SGN function in Eq. 2), the objective function $o(\mathbf{p})$ has some unpleasant properties: It consists of multi-dimensional plateaus with constant value and abrupt boundaries between the plateaus. Thus gradients of the objective function are either zero (on the plateaus) or undefined (at the boundaries), and classical optimization methods using gradient information are not applicable.

Still, different ways seem feasible to solve the problem. One approach is to use optimization methods, which do not require gradient information, to minimize the objective function $o(\mathbf{p})$. This has been done using *alternate variable* methods [21] and using a combination of *Rosenbrock's* method and the *Simplex* method [19].

In another approach, the SGN-type nonlinearity in Eq. 2 is replaced by a sigmoidal nonlinearity with variable gain. In this case, the system becomes a (continuously-valued) discrete-time dynamical system, where gradients are well-defined and classical optimization algorithms can be applied. The idea is to use *Continuation* methods, i.e. to start with a low gain of the sigmoidal function and find the minimum for the objective function in that case. Then the gain is increased by a small amount, and the objective function is minimized again, using the result of the last optimization as the starting point. This scheme is repeated until the gain is very high, and thus the sigmoidal functions becomes similar to the SGN-type nonlinearity [22].

A third method is based on the observation that, even if the continuously-valued template coefficients suggest otherwise, the underlying optimization problem has a finite state space and thus can be treated as a combinatorial optimization problem. *Simulated Annealing* type algorithms have been applied to this problem [5].

Genetic algorithms have also been tried in the global learning problem, both with CT-CNNs and DT-CNNs [19], [23]. The results have been mixed, and it was at least pointed out that the coding of the coefficients for these algorithms is an open problem, which is decisive for the success.

All the above methods can be used to minimize the objective function, but extended experiments suggest that Simulated Annealing is the most robust tool, and that it can find good solutions even in difficult cases. It has to be mentioned, though, that Simulated Annealing algorithms are expensive in terms of computational requirements. Global Learning algorithms are no replacement for local learning algorithms, but an important complement to solve learning problems for DT-CNNs. It has to be mentioned that, as with most learning algorithms for neural networks, the selection of the right training patterns is a crucial problem.

In [24] some interesting examples are given, which are quite complex and certainly beyond the capability of local learning algorithms. The above global learning algorithms are quite successful there and open up interesting, practically relevant areas of application for DT-CNNs.

6 Robust Design Issues

As already mentioned before, the trajectory of a DT-CNN (Section 3) and the fixed points of a CT-CNN (Section 2) can be described by affine inequalities. The trajectory, as well as the fixed points, can be designed by intuition or an appropriate learning algorithm (see Sections 3, 4, and 5). In any case, it is desirable to obtain templates which are robust against noise or deviations from their nominal values. It is possible to define several notions of robustness with respect to arbitrary q-norms on IR^N for a solution $\mathbf{p} \in \mathcal{L} := \left\{\mathbf{p} \in \mathrm{IR}^N : \mathbf{p}^t\mathbf{v}^{[l]} \geq 0; \mathbf{v}^{[l]} \in \mathrm{IR}^N \text{ for } 1 \leq l \leq L\right\}$ (see Section 3) [7]. For example, the relative robustness in weight space $r_w(\mathbf{p})$ with respect to the Euclidean norm $\|\bullet\|$ is defined as the solution of

$$\max r \text{ subject to } \forall \Delta\mathbf{p} \in \mathrm{IR}^N : \|\Delta\mathbf{p}\| = r\|\mathbf{p}\| \text{ implies } (\mathbf{p} + \Delta\mathbf{p}) \in \mathcal{L} \ . \tag{18}$$

It can be shown that $r_w(\mathbf{p})$ is the minimal distance of the vector $\mathbf{p}/\|\mathbf{p}\|$ to the planes defined by the "patterns" $\mathbf{v}^{[l]}$. The most robust solution $\mathbf{p}^*$ is therefore obtained by solving

$$\max_{\mathbf{p}} r_w(\mathbf{p}) = \max_{\mathbf{p}} \min_{l=1,\cdots,L} \frac{\mathbf{p}^t\mathbf{v}^{[l]}}{\|\mathbf{p}\|\|\mathbf{v}^{[l]}\|} \ . \tag{19}$$

Obviously the solution is not unique, since an arbitrary positive scaling does not influence the robustness. Therefore, one can add the additional constraint $\|\mathbf{p}\| = 1$ to the optimization problem. It can be shown that, if the problem is solvable, the objective and the constraints can be interchanged, resulting in an equivalent quadratic programming problem with linear inequality constraints [13]:

$$\min \|\mathbf{p}\| \text{ subject to } \mathbf{p}^t\mathbf{v}^{[l]} \geq \|\mathbf{v}^{[l]}\| \text{ for } l = 1, \cdots, L \, . \tag{20}$$

Since the objective function is very simple and the constraints are affine, it possible to obtain an explicit expression for the dual function ϕ provided by Lagrangian duality, which in this case is called the Wolfe dual. Denoting by $\bar{\mathbf{v}}^{[i]} = \mathbf{v}^{[i]}/\|\mathbf{v}^{[i]}\|$, the Wolfe dual can be written as

$$\max_{\mathbf{x}} \left(-\frac{1}{2} \sum_{ij=1}^{L} x_i \left(\bar{v}^{[i]t}\bar{v}^{[j]}\right) x_j + \sum_{i=1} x_i \right) \text{ subject to } x_j \geq 0 \text{ for } j = 1, \cdots, L \, . \tag{21}$$

Any gradient method can now be applied, and only minor modifications are necessary in order to satisfy the constraints, since they are very simple. The solution $\mathbf{p}^*$ of the original problem Eq. 20 is obtained from a solution $\mathbf{x}^*$ of by Eq. 21 $p^* = \sum_i x_i^* \bar{\mathbf{v}}^{[i]}$. The socalled AdaTron algorithm [13] is one implementation of these ideas.

A solution, which has been robustified in accordance with the above described concept, will be most insensitive to both the tolerances of the weights of the CNN due to an imperfect hardware implementation and to disturbances in the input vectors (images) to be processed.

7 Hardware Oriented Learning

The gradient-based methods described in Section 4 rely on a system description of the following form

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{p}) \,, \tag{22}$$

where ideally the right hand side of $\mathbf{F}$ in Eq. 22 is given by the right hand side of Eq. 1. Any real implementation will deviate from that. But as long as an accurate description of the real circuit is available, it can be used in the learning procedures of Section 4, since they are using a quite general $\mathbf{F}$. This way, important simplifications in hardware are possible [4], [25], which are taken into account in the design stage.

8 Conclusions

The systematic steps towards design and learning with CNNs provide powerful techniques to find the template coefficients (synaptic weights) to perform a desired task. In addition, it also opens up the world of learning of general artificial neural networks to the VLSI-oriented world of CNNs.

References

[1] L. O. Chua and L. Yang, "Cellular neural networks: Theory," *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1257–1272, Oct. 1988.

[2] F. Zou, S. Schwarz, and J. A. Nossek, "Cellular neural network design using a learning algorithm," in *Proc. International Workshop on Cellular Neural Networks and their Applications CNNA-90*, (Budapest, Hungary), pp. 73–81, Dec. 1990.

[3] H. Harrer, J. A. Nossek, and F. Zou, "A learning algorithm for Discrete-Time Cellular Neural Networks," in *IJCNN'91 Proc.*, (Singapore), pp. 717–722, Nov. 1991.

[4] A. Schuler, P. Nachbar, and J. Nossek, "State-based backpropagation-through-time for CNNs," in *Proceedings of the 11th European Conference on Circuit Theory and Design, ECCTD93*, (Davos, Switzerland), pp. 33–38, 1993.

[5] H. Magnussen, J. A. Nossek, and L. O. Chua, "The learning problem for Discrete-Time Cellular Neural Networks as a combinatorial optimization problem," Tech. Rep. UCB/ERL M93/88, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, USA, 1993.

[6] W. Utschick and J. A. Nossek, "Computational learning theory applied to Discrete-Time Cellular Neural Networks," in *Proc. Third IEEE International Workshop on Cellular Neural Networks and their Applications CNNA-94*, (Rome, Italy), Dec. 1994.

[7] P. Nachbar, J. A. Nossek, and J. Strobl, "The generalized AdaTron algorithm," in *Proc. of the International Symposium on Circuits and Systems*, vol. 4, (Chicago, Ill., USA), pp. 2152–2156, 1993.

[8] A. J. Schuler, M. Brabec, D. Schubel, and J. A. Nossek, "Hardware-oriented learning for Cellular Neural Networks," in *Proc. Third IEEE International Workshop on Cellular Neural Networks and their Applications CNNA-94*, (Rome, Italy), Dec. 1994.

[9] T. Roska and L. O. Chua, "The CNN universal machine: an analogic array computer," *IEEE Transactions on Circuits and Systems, II: Analog and Digital Signal Processing*, vol. 40, pp. 163–173, Mar. 1993.

[10] S. Agmon, "The relaxation method for linear inequalities," *Canadian Journal of Mathematics*, vol. 6, pp. 382–392, 1954.

[11] M. Minsky and S. Papert, *Perceptrons - An Introduction to Computational Geometry (Expanded Edition)*. MIT Press, 3 ed., 1988.

[12] F. Rosenblatt, *Principles of Neurodynamics*. Spartan, New York, 1962.

[13] J. K. Anlauf and M. Biehl, "The AdaTron: an adaptive perceptron algorithm," *Europhys. Lett.*, vol. 10, no. 7, pp. 687–692, 1989.

[14] P. Nachbar, A. J. Schuler, T. Füssl, J. A. Nossek, and L. O. Chua, "Robustness of attractor networks and an improved convex corner detector," in *Proc. Second International Workshop on Cellular Neural Networks and their Applications CNNA'92*, (Munich, Germany), pp. 240–245, Oct. 1992.

[15] F. Zou, *Cellular Neural Networks: Stability, Dynamics and Design Methods*. PhD thesis, Technical University Munich, Munich, Germany, Dec. 1992.

[16] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, pp. 2229–2232, Nov. 1987.

[17] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, pp. 263–269, 1989.

[18] S. Miesbach, "Efficient gradient computation for continuous and discrete time-dependent neural networks," in *ISCAS 1991*, (Singapore), pp. 2337–2342, June 1991.

[19] H. Magnussen, *Discrete-Time Cellular Neural Networks: Theory and Global Learning Algorithms*. PhD thesis, Technical University Munich, Munich, Germany, Dec. 1994.

[20] J. S. Judd, *Neural Network Design and the Complexity of Learning*. MIT Press, 1990.

[21] H. Magnussen and J. A. Nossek, "Towards a learning algorithm for Discrete-Time Cellular Neural Networks," in *Proc. Second International Workshop on Cellular Neural Networks and their Applications CNNA'92*, (Munich, Germany), pp. 80–85, Oct. 1992.

[22] H. Magnussen, G. Papoutsis, and J. A. Nossek, "Continuation-based learning algorithm for discrete-time cellular neural networks," in *Proc. Third IEEE International Workshop on Cellular Neural Networks and their Applications CNNA-94*, (Rome, Italy), Dec. 1994.

[23] T. Kozek, T. Roska, and L. O. Chua, "Genetic algorithm for CNN template learning," *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications*, vol. 40, pp. 392–402, June 1993.

[24] A. Kellner, H. Magnussen, and J. A. Nossek, "Texture classification, texture segmentation and text segmentation with discrete-time cellular neural networks," in *Proc. Third IEEE International Workshop on Cellular Neural Networks and their Applications CNNA-94*, (Rome, Italy), Dec. 1994.

[25] S. Espejo, *Redes Neuronales Celulares: Modelado Y Diseño Monolítico*. PhD thesis, Universidad De Sevilla, Spain, 1994.