INTERNATIONAL JOURNAL OF CIRCUIT THEORY AND APPLICATIONS, VOL. 24, 15–24 (1996)

DESIGN AND LEARNING WITH CELLULAR NEURAL NETWORKS†

JOSEF A. NOSSEK

Institute for Network Theory and Circuit Design, Technical University of Munich, Munich, Germany

SUMMARY

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 can 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 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 from an engineering point of view.

1. INTRODUCTION

To find a set of parameters (coefficients, synaptic weights) so that a network performs according to a given task is one of the core problems in the field of neural networks. This is of course also true in the case of continuous-time and discrete-time cellular neural networks (CT-CNNs and DT-CNNs), where the local and translationally invariant interconnections are put together in so-called templates.

The methods utilized to find such templates can be roughly grouped into three areas, the borders between which are not always clear and sharp. In the following an attempt will be made to distinguish between the three approaches. The rest of this contribution will elaborate on learning and design, while the third approach, mapping, will not be dealt with in detail.

Learning

(Supervised) learning (see e.g. Reference 1) is solely based on a given set of pairs of input and desired output patterns. No intuition is used to prescribe any internal representation or trajectories. The aim is to find a parameter vector performing the desired global mapping from input and initial state to the corresponding desired output. Therefore it is also sometimes called global learning.

Design

Design with CNNs (see e.g. Reference 2) can have many different faces, e.g. programming the network to have some desired fixed points (or other attractors) or to evolve along a prescribed trajectory from a given initial condition to some desired fixed point under control of a given input or specifying some desired local dynamics.

Mapping

This last group (see e.g. References 3 and 4) summarizes situations where the CNNs are used to simulate or mimic a certain physical, chemical or biological phenomenon based on a set of equations (e.g. partial

† Part of this research has been reported in the Proceedings of the 1994 IEEE International Workshop on Cellular Neural Networks and Their Applications held in Rome.

CCC 0098–9886/96/010015–10
© 1996 by John Wiley & Sons, Ltd.

Received 15 January 1995
Revised 6 April 1995

differential equations (PDEs)). Here the discretization of space (and time) plays a central role in arriving at a set of ODEs (or difference equations), which can be easily mapped onto a CT-CNN (or DT-CNN).[5] Of utmost importance is to carry out the discretization such that invariants of the original phenomenon are still invariants of the discrete approximation. The mapping onto the CNN architecture is straightforward.

System equations

The equations for each cell c of a CT-CNN are

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

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 neighbourhood $\mathcal{N}_r$ determines the connection weight.

Outline

The first useful templates have been derived by design in analogy to known image-processing algorithms, while the first systematic approach for the design of CT-CNNs was aimed at programming desired fixed points[6] (see subsection 2.1). This technique has later been adapted to the discrete-time case in Reference 7 (see subsection 2.2) and it requires the *a priori* knowledge of the trajectories. Modified versions of recurrent backpropagation and backpropagation-through-time have been developed[8] to make sure that the CT-CNN will not only have the desired fixed point but also 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 3[9] 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 of how many samples (input–output pairs) are necessary for reliable generalization. In Reference 10 an upper bound on the sample size is derived by applying the probably approximately correct (PAC) learning theory to DT-CNNs.

In a companion paper[11] various applications of global learning algorithms for DT-CNNs are described in some detail.

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 4).[12] This is already a step towards taking into account the hardware constraints at the design or learning stage. The approach in Reference 13 even proposes the use of modified network equations for the actual behaviour 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,[14] are first broken down into individual tasks by the intuition and experience of the designer and then dealt with as in single-layer CNNs above.

2. DESIGN

2.1. *Designing fixed points*

In this subsection, the issue of designing fixed points 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, equation (3), really is given by the desired y^∞, which is equivalent to

$$\left.\begin{matrix} x_c^\infty \geqslant +1 & \text{if } y_c^\infty = +1 \\ x_c^\infty \leqslant -1 & \text{if } y_c^\infty = -1 \end{matrix}\right\} \Leftrightarrow y_c^\infty\left(\sum_{d\in\mathcal{N}_r(c)} a_{d-c} y_d^\infty - \sum_{d\in\mathcal{N}_r(c)} b_{d-c} u_d + i_c\right) \geqslant 1 \tag{4}$$

for each cell c of the network $(c = 1, \ldots, M)$. In general one has $L > 1$ desired fixed points $y^{\infty[l]}$ along with some input patterns $u^{[l]}(l = 1, \ldots, L)$. For each pair of training patterns one obtains the system of affine inequalities, equation (4), for the unknowns a, b and i.[6] This can now be solved by many methods, e.g.the relaxation method in Reference 15, the perceptron algorithm,[16] Rosenblatt's algorithm[17] and the AdaTron algorithm,[18] to mention just 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 the above mentioned algorithms can be adapted to incorporate these additional equality constraints.[6,19]

Simply replacing equation (4) by

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

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

In both cases (equations (4) and (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 Reference 20 a step towards taking into account initial conditions is made but this approach works reliably only if the transients are simply monotonic.

2.2. *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 non-linearity. 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 subsection 2.1 can be used, though one has to be willing and able to prescribe a sensible trajectory $\mathbf{u}, \mathbf{y}(0), \ldots, \mathbf{y}(T)$. From the recursion equation (2) the following set of inequalities can be derived for each time step $k = 0,, \ldots, T-1$:

$$\left.\begin{matrix} x_c(k_\infty) \geqslant 0 & \text{if } y_c(k+1) = +1 \\ x_c(k_\infty) < 0 & \text{if } y_c(k+1) = -1 \end{matrix}\right\} \Leftrightarrow y_c(k+1)\left(\sum_{d\in\mathcal{N}_r(c)} a_{d-c} y_d(k) + \sum_{d\in\mathcal{N}_r(c)} b_{d-c} u_d + i_c\right) > 0. \tag{6}$$

Again more than one trajectory can be prescribed, and one can replace the inequality '>0' in the above equation by '$\geqslant R$' to ensure some kind of robustness of the solution.[7] 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

Table I. Template coefficients for edge detection

$$A = \begin{bmatrix} -1{\cdot}29 & 2{\cdot}58 & -1{\cdot}29 & 2{\cdot}58 & -1{\cdot}29 \\ 2{\cdot}58 & 0{\cdot}83 & 1{\cdot}58 & 0{\cdot}83 & 2{\cdot}58 \\ -1{\cdot}29 & 1{\cdot}58 & 8{\cdot}67 & 1{\cdot}58 & -1{\cdot}29 \\ 2{\cdot}58 & 0{\cdot}83 & 1{\cdot}58 & 0{\cdot}83 & 2{\cdot}58 \\ -1{\cdot}29 & 2{\cdot}58 & -1{\cdot}29 & 2{\cdot}58 & -1{\cdot}29 \end{bmatrix}, \quad B = \frac{1}{0{\cdot}9}\begin{bmatrix} 0 & -3 & -1 & -3 & 0 \\ -3 & 2 & 3 & 2 & -3 \\ -1 & 3 & 8 & 3 & -1 \\ -3 & 2 & 3 & 3 & -3 \\ 0 & -3 & -1 & -3 & 0 \end{bmatrix}, \quad I = -1$$

$\mathcal{L} := \{\mathbf{p} \in \mathbb{R}^N\colon\ \mathbf{p}^T\mathbf{v}^{[l]} > R \geq 0;\ \mathbf{v}^{[l]} \in \mathbb{R}^N \text{ for } 1 \leq l \leq L\}$ 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 4.

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

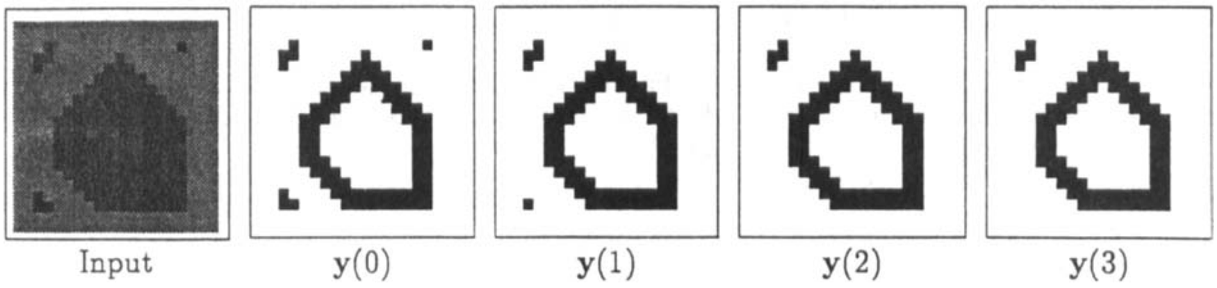

Figure 1. Learning samples for edge detection

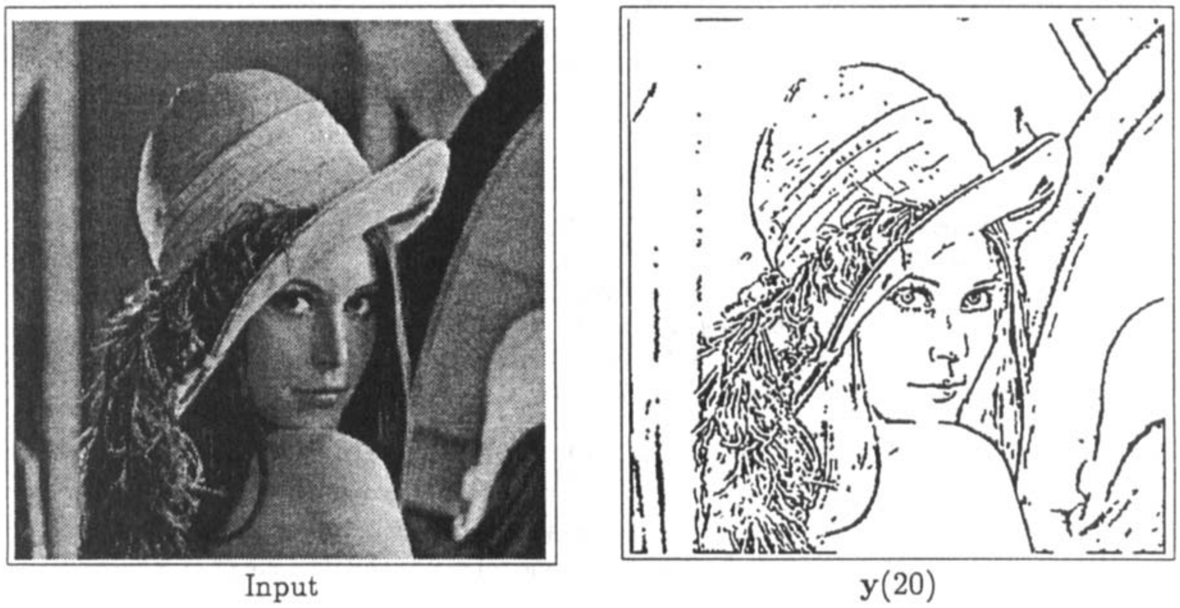

Figure 2. Edge detection on 'Lena' image

2.3. Gradient-based methods for designing CT-CNNs with prescribed trajectories

The design of fixed points, however, does not guarantee the correct behaviour 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 $(l = 1, \dots, L)$.

A common way of learning in neural networks is to define an error measure or cost function of the fixed points and the desired outputs (*recurrent backpropagation*[21]) or in general of the trajectory of the system and the desired trajectory (*backpropagation-through-time*[22]):

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

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 $l = 1, \dots, L$.

Owing to the piecewise linear output function, it is better to define the error as a function of the states instead of the output.[8] With the following function with a parameter R,

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

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

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

In Figure 3 this state-based error function is visualized in contrast with a conventional one for the special case of robustness $R = 0$ and $k = 1$. The conventional error measure (e.g. $e = |y_c - d_c|$) would lead to a constant error (and therefore vanishing gradient) as soon as a cell c was in the wrong saturation region, which is obviously not the case for the error measure (9).

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

Recurrent backpropagation (RBP)[21] 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

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

where F_c is the right-hand side of equation (1). $\lambda \in \mathbb{R}^M$ 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.

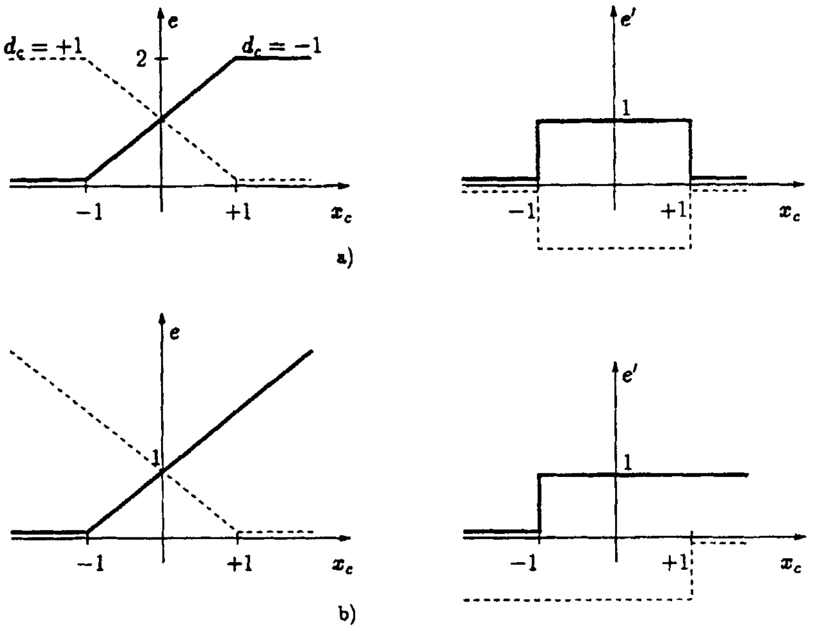

Figure 3. (a) Conventional and (b) state-based error measure and its derivative

It is possible to implement, in addition to the CT-CNN itself, the associated dynamical system (equation (12)) also in hardware. By properly separating the time constants of the CNN and (equation (12)), a fully adaptive (learning) CNN can be constructed.

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),[22,23] not only fixed points but also prescribed trajectories can be learned. The gradient of the state-based error can be simplified as

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

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 backwards in time, since the boundary value of λ_c is known at the terminal time T:

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

Depending on the choice of e_1 and e_2, BTT can be used to follow a prescribed trajectory $d_c^{[t]}(t)$ 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. Although this is true, the state-based versions of RBP and BTT described here are much better in this respect when compared with their output-based counterparts.[8]

For both algorithms, versions applicable to DT-CNNs are also available[24] provided that their threshold non-linearity is replaced by a continuously valued one.

3. 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 Reference 25, it can be concluded that global learning for DT-CNNs belongs to the class of NP-complete problems.[9]

All 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 behaviours: 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

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

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

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

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

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 the input image $\mathbf{u}^{[l]}$ is 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.

Owing to the inherently non-linear behaviour of a DT-CNN cell (caused by the sign function in equation (2)), the objective function $o(\mathbf{p})$ has some unpleasant properties: it consists of multidimensional plateaux with constant value and abrupt boundaries between the plateaux. Thus gradients of the objective function are either zero (on the plateaux) or undefined (at the boundaries) and classical optimization methods using gradient information are not applicable.

Still, various 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[26]and using a combination of *Rosenbrock's* method and the *Simplex* method.[24]

In another approach the sign-type non-linearity in equation (2) is replaced by a sigmoidal non-linearity 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 function becomes similar to the sign-type non-linearity.[27]

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* algorithms have been applied to this problem.[9]

Genetic algorithms have also been tried in the global learning problem, both with CT-CNNs and DT-CNNs.[24,28] 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 their 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 are 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 Reference 11 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.

4. ROBUST DESIGN ISSUES

As already mentioned before, the trajectory of a DT-CNN (Subsection 2.2) and the fixed points of a CT-CNN (subsection 2.1) 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 2.2, 2.3 and 3). 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 $\mathbb{R}^N$ for a solution $\mathbf{p} \in \mathcal{L} := \{\mathbf{p} \in \mathbb{R}^N\colon \mathbf{p}^{\mathrm{T}}\mathbf{v}^{[l]} \geq 0;\ \mathbf{v}^{[l]} \in \mathbb{R}^N \text{ for } 1 \leq l \leq L\}$ (see Section 2.2).[12] For example, the relative robustness in weight space, $r_w(\mathbf{p})$, with respect to the Euclidean norm $\|\cdot\|$ is defined as the solution of

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

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

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

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:[18]

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

Since the objective function is very simple and the constraints are affine, it is 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 $\tilde{\mathbf{v}}^{[l]} = \mathbf{v}^{[l]}/\mathbf{v}^{[l]}\|$, the Wolfe dual can be written as

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

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 (20) is obtained from a solution $\mathbf{x}^*$ of (21) by $\mathbf{p}^* = \sum_i x_i^* \tilde{\mathbf{v}}^{[i]}$. The so-called AdaTron algorithm[18] is one implementation of these ideas.

A solution which has been robustified in accordance with the above-described concept will be most insensitive both to 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.

5. CONCLUSIONS

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

ACKNOWLEDGEMENTS

The author would like to thank first and foremost his doctoral students Hubert Harrer, Holger Magnussen, Peter Nachbar, Georgios Papoutsis, Andreas Schuler, Gerhard Seiler, Wolfgang Utschick and Fan Zou, who did all the work making this survey paper possible. He is also grateful to Professors Leon O. Chua and Tamás Roska for stimulating and encouraging discussions.

REFERENCES

1. P. Szolgay, I. Kispál and T. Kozek, 'An experimental system for optical detection of layout errors of printed circuit boards using learned CNN templates', *Proc. Second IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Munich, October 1992, pp. 203–209.
2. L. O. Chua and P. Thiran, 'An analytic method for designing simple cellular neural networks', *IEEE Trans. Circuits and Systems*, **CAS-38**, 1332–1341 (1991).
3. J. Henseler and P. J. Braspenning, 'Membrain: a cellular neural network model based on a vibrating membrane', *Int. j. cir. theor. appl.*, *20*, 483–496 (1992).
4. P. Szolgay, G. Voros and G. Eross, 'On the applications of the cellular neural network paradigm in mechanical vibrating systems', *IEEE Trans. Circuits and Systems O*, **CAS-40**, 222–227 (1993).
5. T.Roska, D.Wolf, T.Kozek and R.Tetzlaff, 'Solving partial differential equations by CNN', *Proc. 11th Eur. Conf. on Circuit Theory and Design*, Davos, August 1993, pp.1477–1482.
6. F.Zou, S.Schwarz, and J.A. Nossek, 'Cellular neural network design using a learning algorithm,' *Proc.IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Budapest, December 1990, pp. 73–81.
7. H.Harrer, J.A. Nossek and F.Zou, 'A learning algorithm for discrete-time cellular neural networks', *IJCNN'91 Proc.*, Singapore, November 1991, pp.717–722.
8. A.Schuler, P.Nachbar and J.Nossek, 'State-based backpropagation-through-time for CNNs', in *Proc. 11th Eur. Conf. on Circuit Theory and Design*, Davos, August, 1993, pp.33–38.
9. 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, CA, 1993.
10. W.Utschick and J.A. Nossek, 'Computational learning theory applied to discrete-time cellular neural networks', *Proc.Third IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Rome, December 1994, 159–164.
11. H.Magnussen, A.Kellner and J.A. Nossek, 'A collection of applications for discrete-time cellular neural networks', submitted to IEEE Transactions on Circuits and Systems, Part I.
12. P.Nachbar, J.A. Nossek and J.Strobl, 'The generalized AdaTron algorithm', *Proc. IEEE Int. Symp. on Circuits and Systems*, Chicago, IL, 1993, Vol. 4, pp. 2152–2156.
13. A.J. Schuler, M.Brabec, D.Schubel and J.A. Nossek, 'Hardware-oriented learning for cellular neural networks', *Proc.Third IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Rome, December 1994, 183–188.
14. T.Roska and L.O. Chua, 'The CNN universal machine: an analogic array computer', *IEEE Trans. Circuits and Systems II*, **CAS-40**, 163–173 (1993).
15. S.Agmon, 'The relaxation method for linear inequalities', *Can. J. Math.*, **6**, 382–392, (1954).
16. M.Minsky and S.Papert, *Perceptrons—An Introduction to Computational Geometry* (*Expanded Edition*), 3rd edn, MIT Press, Boston, MA, 1988.
17. F.Rosenblatt, *Principles of Neurodynamics*, Spartan, New York, 1962.
18. J.K. Anlauf and M.Biehl, 'The AdaTron: an adaptive perceptron algorithm', *Europhys. Lett.*, **10**, 687–692 (1989).
19. P.Nachbar, A.J. Schuler, T.Füssl, J.A. Nossek and L.O. Chua, 'Robustness of attractor networks and an improved convex corner detector', *Proc.Second IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Munich, October 1992, pp.240–245.
20. F.Zou, 'Cellular neural networks: stability, dynamics and design methods', *Ph.D. Thesis*, Technical University of Munich, Munich, 1992.
21. F.J. Pineda, 'Generalization of back-propagation to recurrent neural networks', *Phys. Rev. Lett.*, **59**, 2229–2232 (1987).
22. B.A. Pearlmutter, 'Learning state space trajectories in recurrent neural networks', *Neural Comput.*, **1**, 263–269 (1989).

23. S.Miesbach, 'Efficient gradient computation for continuous and discrete time-dependent neural networks', *Proc. IEEE Int. Symp. on Circuits and Systems*, Singapore, June 1991, pp.2337–2342.
24. H.Magnussen, 'Discrete-time cellular neural networks: theory and global learning algorithms', *Ph.D. Thesis,* Technical University of Munich, 1994.
25. J.S. Judd, *Neural Network Design and the Complexity of Learning*, MIT Press, Boston, NA, 1990.
26. H.Magnussen and J.A. Nossek, 'Towards a learning algorithm for discrete-time cellular neural networks', *Proc.Second IEEE Int. Workshop on Cellular Neural Networks and Their Applications*, Munich, October 1992, pp.80–85.
27. H.Magnussen, G.Papoutsis and J.A. Nossek, 'Continuation-based learning algorithm for discrete-time cellular neural networks', Proc.Third IEEE Int. Workshop on Cellular Neural Networks and Their Applications, Rome, December 1994, 171–176.
28. T.Kozek, T.Roska and L.O. Chua, 'Genetic algorithm for CNN template learning', *IEEE Trans. Circuits and Systems I,* **CAS-40**, 392–402 (1993).
29. L.O. Chua and L.Yang, 'Cellular neural networks: theory,' *IEEE Trans. Circuits and Systems*, **CAS-35**, 1257–1272 (1988).