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

Continuation-Based Learning Algorithm for Discrete-Time Cellular Neural Networks

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

Abstract – *The SGN-type nonlinearity of a standard Discrete-Time Cellular Neural Network (DTCNN) is replaced by a smooth, sigmoidal nonlinearity with variable gain. Therefore, the resulting dynamical system is fully differentiable. Bounds on gain of the sigmoidal function are given, so that the new, smooth system approximates the standard DTCNN within certain limits. A learning algorithm is proposed, which finds the template parameters for the standard DTCNN by gradually increasing the gain of the sigmoidal function.*

1 Introduction

The Discrete-Time Cellular Neural Network (DTCNN) was introduced in [1] as a discrete-time version of the Cellular Neural Network [2]. In a vector notation, the DTCNN equations can be written as follows

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}\mathbf{y}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{i} \\ \mathbf{y}(t+1) &= \mathbf{s}(\mathbf{x}(t+1)), \quad \mathbf{y}(0) := \mathbf{y}_0 . \end{aligned} \tag{1}$$

Let M_0 be the number of (active and dummy) cells. $\mathbf{u}(t)$, $\mathbf{y}(t)$, and $\mathbf{x}(t)$ are the M_0-vectors of input signals, of cell output signals, and of cell states, respectively. Note that the dummy cells, which surround the array of active cells, are also included in the vectors. The $M_0 \times M_0$ matrices $\mathbf{A}$ and $\mathbf{B}$ contain the feedback and the feedforward template coefficients. Due to the local interconnection structure of the DTCNN and the translational invariance of the template coefficients, the two matrices are only sparse. $\mathbf{i}$ is the M_0-vector of (identical) cell bias values. The vector-valued function $\mathbf{s} : \mathbb{R}^{M_0} \rightarrow \{-1,1\}^{M_0}$ is equivalent to an element-wise application of the sign function $s : \mathbb{R} \rightarrow \{-1,1\}$ ($s(x) = +1$ for $x \geq 0$, and -1 otherwise) to each element of $\mathbf{x}(t)$ of the active cells and to a constant value for the dummy cells. Due to the SGN-type cell nonlinearity $s(x)$, the resulting dynamical system is not differentiable.

Since gradient information is not available, standard learning algorithms for differentiable dynamical systems [3], [4], [5] cannot be applied. Therefore, the SGN-type function $s(x)$ in Eq. 1 is replaced by a smooth, nonlinear, differentiable function $\hat{s}(\beta x)$ with

$\hat{s} : \mathbb{R} \rightarrow [-1, 1]$. $\hat{s}(\beta x)$ is a monotonous function with saturation regions, and it satisfies $\hat{s}(x) = -\hat{s}(-x)$, $\hat{s}(0) = 0$, and $\hat{s}(x \rightarrow \pm\infty) = \pm 1$. A typical choice is $\hat{s}(\beta x) := \tanh(\beta x)$. $\beta > 0$ is an adjustable gain factor. In the limit $\beta \rightarrow \infty$, the smooth function $\hat{s}(\beta x)$ is identical to $\mathrm{SGN}(x)$ for all $x \neq 0$. For large, but finite β, the smooth function will approximate the SGN-function except for small values of $|x|$.

For the rest of this work, all cell states $\mathbf{x}(t)$ and cell outputs $\mathbf{y}(t)$ belonging to the system with the smooth nonlinearity will be distinguished from the standard DTCNN system by a hat "$\hat{*}$". In analogy with Eq. 1, we define

$$\begin{aligned} \hat{\mathbf{x}}(t+1) &= \mathbf{A}\hat{\mathbf{y}}(t) + \mathbf{B}\mathbf{u}(t) + \mathbf{i} \\ \hat{\mathbf{y}}(t+1) &= \hat{\mathbf{s}}(\beta\hat{\mathbf{x}}(t+1))\ , \quad \hat{\mathbf{y}}(0) := \mathbf{y}(0) = \mathbf{y}_0\ . \end{aligned} \tag{2}$$

This system is now fully differentiable with respect to the network parameters.

2 Approximation Properties

If in the original DTCNN from Eq. 1 all cell state values at all time-steps t are nonzero, then the behavior of the original DTCNN can be approximated with arbitrary precision by the smooth system from Eq. 2:

Lemma 1 (Approximation properties of the smooth system): *Let $x_\mu(t) \neq 0$ for all t and all cells μ. Let $0 < \delta < 1$. $\mathbf{a}_\mu$ is the μ-th row of the matrix $\mathbf{A}$. Then*

$$\left| x_\mu(t+1) - \frac{\delta}{2}\mathbf{a}_\mu \mathbf{y}(t) \right| \geq \frac{1}{\beta}\hat{s}^{-1}(1-\delta) + \frac{\delta}{2} \cdot \|\mathbf{a}_\mu\|_1 \qquad \forall\, t \geq 0, \forall\, \mu \tag{3}$$

is a sufficient condition for

$$\|\mathbf{y}(t) - \hat{\mathbf{y}}(t)\|_\infty \leq \delta \qquad \forall\, t \geq 0\ . \tag{4}$$

◇

Proof: by induction; omitted for lack of space □

It is shown in [6] that the parameter space of the standard DTCNN system Eq. 1 is segmented into a very large number of convex cones, in which the behavior of the system is constant. The condition from Eq. 3 adds *transition regions* around the hyperplanes separating the convex cones, where the system Eq. 2 does not necessarily approximate system Eq. 1. This is illustrated for the case of a two-dimensional parameter space in Fig. 1, where the transition regions are shown in grey. The system behavior in the transition regions can be very complicated.

Lemma 1 implies that in the limit for very small δ and very large β, the smooth system Eq. 2 approximates the original DTCNN system Eq. 1 almost everywhere in the parameter space except for points where $|x_\mu(t)| \ll 1$ for some t, μ, i.e. around the hyperplanes separating the convex cones in the parameter space.

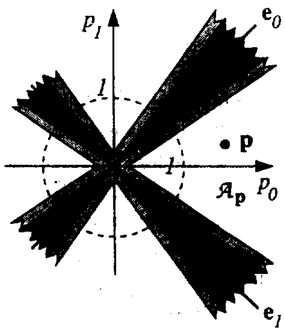

Fig. 1: Transition regions and approximation regions

Note that the SGN-type nonlinearity in a DTCNN hardware realisation is often realized by a high gain amplifier [7]. Therefore, a hardware implementation is in fact a smooth system of the form of Eq. 2. This implies that the template parameters of a hardware implementation will have to obey the condition given in Lemma 1 in order to work properly. This leads to even stricter accuracy conditions on the template parameters than the ones derived in [6].

3 Learning by Continuation-Based Methods

Without loss of generality, we assume that the learning task consists of only one item. An objective function is introduced, which measures the deviation of the actual behavior of the network outputs $\hat{\mathbf{y}}(t)$ from the desired behavior $\hat{\mathbf{d}}(t) \in [-1,1]^{M_0}$ for $1 \leq t \leq T$. Let $\mathbf{p}$ denote the *parameter vector* containing all independent template coefficients and the bias. We then define the objective function $\hat{o}(\mathbf{p})$

$$\hat{o}(\mathbf{p}) := \frac{1}{4M_0}\sum_{t=1}^{T}\alpha(t)\cdot\left\|\hat{\mathbf{y}}(t)-\hat{\mathbf{d}}(t)\right\|_2^2 \qquad \text{with } \alpha \in [0,1]\,,\ \sum_{t=1}^{T}\alpha(t)=1\,. \tag{5}$$

This objective function is differentiable with respect to the template parameters. Gradients can be computed using methods like the Williams/Zipser method [4] or the discrete-time version of the method proposed by Pearlmutter [5]. Note that the network is only operated for a fixed number of T time-steps. The degree of freedom in systems Eq. 1 and Eq. 2 (β is counted as a system parameter as well) is eliminated by using a parameterization for $\mathbf{p}$, which keeps the 2-norm of the parameter vector constant.

From Lemma 1, it can be concluded that a minimization of the objective function Eq. 5 does not make sense for large values of β, because in this case, Eq. 5 is a good approximation of an objective function of system Eq. 1 defined analogously to Eq. 5. In the case of large β, the objective function Eq. 5 has large areas with very low gradient values, and small areas, most notably the ones excluded by Eq. 3, where the gradients can become very large. The minimisation of such an objective function is very difficult from a numerical point of view.

Therefore, the mathematical tool of *Continuation* is applied [8]. Let S^β denote a smooth system of the form Eq. 2 with a gain parameter β and a parameter vector $\mathbf{p}$ satisfying $\|\mathbf{p}\|_2 = 1$. Let $\hat{o}^\beta(\mathbf{p})$ denote the corresponding objective function according to Eq. 2. A PIDGIN ALGOL [9] description of the Continuation method is given in Fig. 2.

```
begin Continuation method;
  randomize($\mathbf{p}_{ini}$);
  for $j = 0$ to $j_{max}$ do
    $\mathbf{p}_{opt}$ = optimize($\hat{o}^{\beta_j}(\mathbf{p}), \mathbf{p}_{ini}$);
    $\mathbf{p}_{ini}$ = new_init($\mathbf{p}_{opt}$);
  end
end Continuation method;
```

Fig. 2: Continuation-based method

The function optimize($\hat{o}^{\beta_j}(\mathbf{p}), \mathbf{p}_{ini}$) minimizes the objective function $\hat{o}^{\beta_j}(\mathbf{p})$ constrained to ($\|\mathbf{p}\|_2 = 1$) using the *ParTan* algorithm, which is an optimisation algorithm especially suited for functions with "narrow valleys" [10]. This minimisation starts from the initial point $\mathbf{p}_{ini}$. It returns the parameter vector $\mathbf{p}_{opt}$ corresponding to a (local) optimum. $\beta_0, \ldots, \beta_{j_{max}}$ is a monotonously increasing series of positive gain factors.

The algorithm works as follows: It starts with a small value β_0 and a random starting point $\mathbf{p}_{ini}$. Let $\mathbf{p}_{opt}$ be the result of the optimisation of $\hat{o}^{\beta_j}(\mathbf{p})$. The new starting point for the optimization of $\hat{o}^{\beta_{j+1}}(\mathbf{p})$ is computed by the function new_init($\mathbf{p}_{opt}$). In most cases, $\mathbf{p}_{\text{ini}} = \mathbf{p}_{\text{opt}}$ is a reasonable strategy, but more elaborated extrapolation methods can be used as well. Then, β is increased from β_j to β_{j+1}. This is repeated, until the smooth system $S^{\beta_{j_{\max}}}$ sufficiently approximates the standard DTCNN system for some large $\beta_{j_{\max}}$. The idea behind the Continuation method is that by increasing β from β_j to β_{j+1}, where $\beta_{j+1} - \beta_j$ should be reasonably small, the location of minimum $\mathbf{p}_{\text{opt}}$ will not move drastically, so that the new optimum is reached in relatively few steps.

The Continuation method can be visualized by a simple example. A one-dimensional DTCNN consisting of 8 cells with only an a-template is set up. This network is supposed to learn the 1D Connected Component Detector task. The objective function of the form (5) is summed over all possible $L = 256$ input patterns. Thus the network has 3 independent parameters. Due to the ($\|\mathbf{p}\|_2 = 1$)-restriction, the parameter vector can be reduced to two dimensions by using a transformation of the form

$$p_1 = \sin v_1; \quad p_2 = \cos v_1 \sin v_2; \quad p_3 = \cos v_1 \cos v_2 \tag{6}$$

with $v_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $v_2 \in [-\pi, \pi]$. Note that this transformation is cyclical in v_2. The objective function $\hat{o}^{\beta}(v_1, v_2)$ is shown in Fig. 3 for different values of β. The objective function value is coded as the greyscale value. Dark regions correspond to low objective function values. Note that the objective function values are scaled differently in the different plots. It can clearly be seen, how the objective function develops from a smooth surface for small β to the plateau-type surface for large β. The location of the minima for the corresponding value of β changes slowly with β. Due to the transformation in Eq. 6, the shape of the convex cones in the parameter space is distorted.

4 Experimental Results

The Continuation-based algorithm described in Section 3 was first applied to easy learning problems with low-dimensional parameter vectors. In most of these cases the algorithm could easily find the expected global minimum of the described objective function $\hat{o}(\mathbf{p})$ without being very sensitive to the specific strategy of increasing the gain parameters β_j.

However, the performance of the method deteriorates for more complicated problems with parameter vectors of a higher dimensionality. This happens, because the optimization algorithm can get stuck in undesirable local minima of the objective function Eq. 5. These local minima lie in the transition regions excluded by Eq. 3, where at least one cell output $\hat{y}_\mu(t)$ can have very small values, so that the signal levels are no longer "quasi-binary". The local minima can be very narrow and very deep, especially for large values of β. They correspond to solutions that cannot be performed by the original DTCNN system from Eq. 1 due to its binary-valued cell outputs. Actually, it can be shown that it is possible to chose a binary-valued desired output $\hat{\mathbf{d}}(t)$, so that the objective function Eq. 5 has a local minimum in a transition region [11].

As an example, Fig. 4 shows $\hat{o}(\mathbf{p})$ (solid line), the objective function of the smooth system according to Eq. 5, and a corresponding objective function $o(\mathbf{p})$ of the standard DTCNN system from Eq. 1 (dotted line). Both functions are shown for parameter vector values taken from a short line segment through a transition region in the parameter space, symbolically denoted by p. Here, β is very large, and the local minimum, which only exists for the smooth system, is clearly visible.

In the case of more complicated problems, the choice of the strategy for increasing β becomes very important. By changing β, the shape of the objective function and the

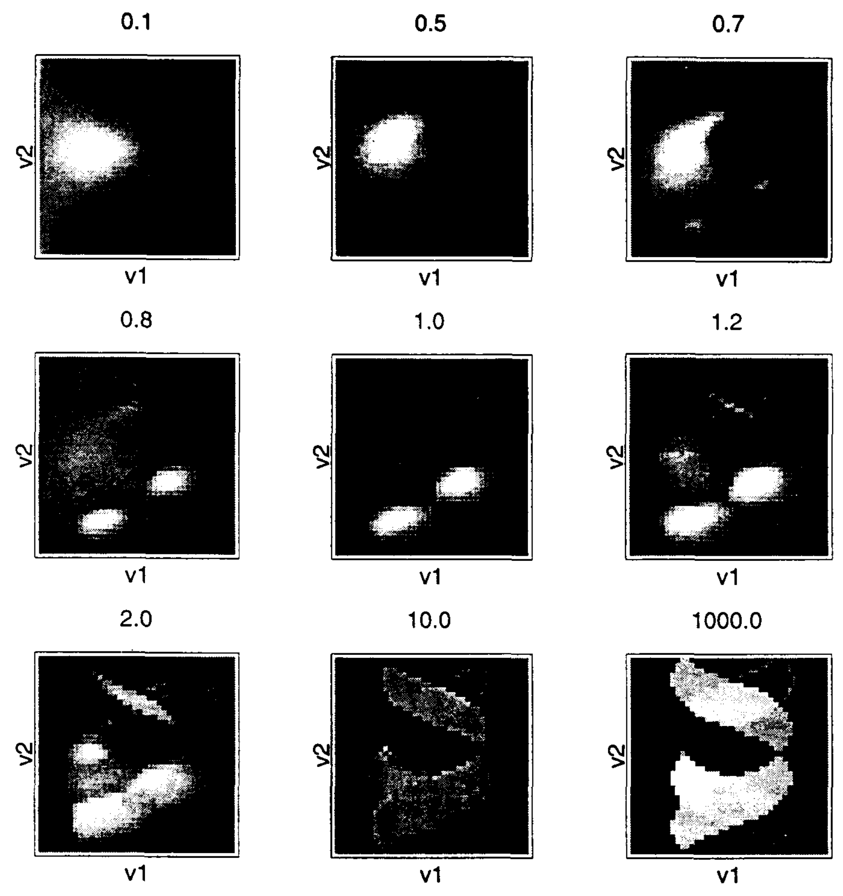

Figure 3: Objective function $\hat{o}^{\beta}(v_1, v_2)$ for the smooth system and the CCD task

location of the minima changes. If β is increased too fast, then the starting point for the optimization of system S^{β_j}, i.e. the result of the optimization of system $S^{\beta_{j-1}}$, is too far away from the actual location of the minimum of system S^{β_j}. On the other hand, if β is increased too slowly, then computation time is wasted, because the function `optimize()` has to be called too many times. In addition, it becomes more likely that the algorithm tracks one of the narrow, undesired minima in the transition regions until $\beta_{\max}$ is reached. This solution is worthless, since it does not reflect the behavior of the standard DTCNN system. For moderate increases of β, the algorithm can "jump out" of the undesired local minima while at the same time it can track the much broader minima corresponding to a mapping performable by the standard DTCNN system.

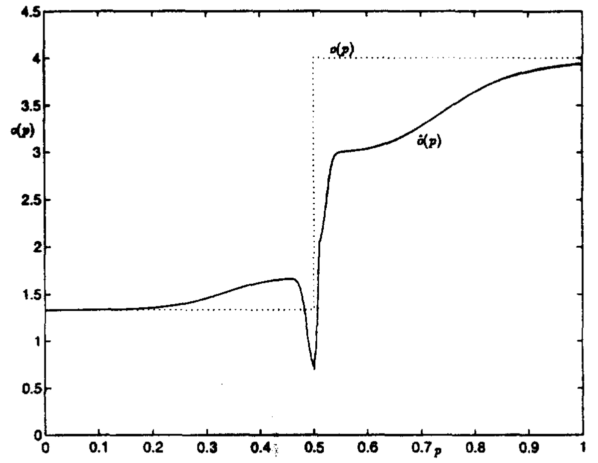

Figure 4: Objective function $\hat{o}(\mathbf{p})$ and $o(\mathbf{p})$ in a transition region

References

[1] H. Harrer and J. A. Nossek, "Discrete-Time Cellular Neural Networks," *International Journal of Circuit Theory and Applications*, vol. 20, pp. 453–467, Sept. 1992.

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

[3] F. J. Pineda, "Dynamics and architecture for neural computation," in *Artificial Neural Networks: Paradigms, Applications and Hardware Implementation* (E. Sánchez-Sinencio and C. Lau, eds.), pp. 58–81, IEEE press, 1992.

[4] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

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

[6] H. Magnussen and J. A. Nossek, "A geometric approach to properties of the discrete-time cellular neural network *(to be published)*," *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications*, 1994.

[7] H. Harrer, J.A. Nossek and R. Stelzl "An analog implementation of Discrete-Time Cellular Neural Networks," *IEEE Transactions on Neural Networks*, vol. 3, pp. 466–476, May 1992.

[8] R. Seydel, "Tutorial on continuation," *Int. J. of Bifurcation and Chaos*, vol. 1, no. 1, pp. 3–11, 1991.

[9] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1. ed., 1982.

[10] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, 1. ed., 1973.

[11] G. Papoutsis, "Implementierung eines Optimierungsalgorithmus als Lernalgorithmus für Zeitdiskrete Zellulare Neuronale Netze *(in german)*." Diplomarbeit, Dec. 1993.