



## Global Learning Algorithms for Discrete-Time Cellular Neural Networks

Holger Magnussen 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** – Two learning algorithms for Discrete-Time Cellular Neural Networks (DTCNNs) are proposed, which do not require the a priori knowledge of the output trajectory of the network. A cost function is defined, which is minimized by Direct Search optimization methods and Simulated Annealing. Applications of the algorithms are presented in a companion paper.

### 1 Introduction

The Discrete-Time Cellular Neural Network (DTCNN) is a discrete-time, nonlinear, first-order dynamical system consisting of  $M$  identical cells on a 1D or 2D cell grid.  $M$  input ports are placed on an input grid with identical dimensions. The DTCNN was introduced in [1] as a discrete-time version of the continuous-time Cellular Neural Network (CNN) [2]. It is defined by a state equation and an output equation

$$\begin{aligned}x_{\mu}(t+1) &= \sum_{\lambda \in \mathcal{N}(\mu)} a_{\lambda-\mu} y_{\lambda}(t) + \sum_{\lambda \in \mathcal{N}(\mu)} b_{\lambda-\mu} u_{\lambda}(t) + i \\y_{\mu}(t) &= \text{SGN}(x_{\mu}(t)) := \begin{cases} +1 & \text{for } x_{\mu}(t) \geq 0 \\ -1 & \text{for } x_{\mu}(t) < 0 \end{cases},\end{aligned}\tag{1}$$

and the initial state  $y_{\mu}(0) = y_{\mu,0}$ .  $u_{\mu}(t)$ ,  $y_{\mu}(t)$ , and  $x_{\mu}(t)$  are the input signal at port  $\mu$ , the output signal and the state of cell  $\mu$ , respectively, at time-step  $t$ .  $t$  is a non-negative integer corresponding to the time-step.  $\mathbf{a} = (a_{\nu})$  and  $\mathbf{b} = (b_{\nu})$  are the feedback and feedforward connections between the cells and the inputs and the cells. Due to the translational invariance of the network parameters,  $\mathbf{a}$  and  $\mathbf{b}$  are also called *template coefficients*.  $i$  is the cell bias. Let  $\mathbf{p}$  denote the *parameter vector*, which contains the template coefficients  $\mathbf{a}$ ,  $\mathbf{b}$  and the bias  $i$ .  $\mathcal{N}(\mu)$  is the *neighborhood* of cell  $\mu$  on the cell grid. The use of the neighborhoods in the summations implies that the connections between the cells are only local. In most cases, neighborhood sizes are  $r = 1$  or  $r = 2$ . The cell grid is surrounded by  $r$  layers of *dummy cells*.

Due to its discontinuous cell nonlinearities and the local interconnection scheme, the DTCNN is difficult to analyze. This is especially true for the learning problem, i.e. the

problem of finding the network parameters, so that the network fulfills a desired task. Most neural network learning problems are inherently difficult [3]. In the case of the DTCNN, the network has to map a set of input images onto corresponding desired output images, which are usually required to be stable. Almost all the proposed learning algorithms for DTCNNs and CNNs (with the exception of [4]) are *design* algorithms rather than *learning* algorithms [5], [6], [7]. They work either by just setting the fixed points of the system, or they require that the trajectory of the network is known for each time-step. This last condition is not satisfied for many practical learning problems. Therefore, two “global” learning algorithms are proposed in this work, which extract the information on the network parameters from a given set of input images and the desired output images. The trajectory of the network is thus designed by the learning algorithms. Learning is done by transforming the learning problem into a more general optimization problem. Applications of the two learning algorithms are presented in a companion paper [8].

## 2 Objective Function

Let  $\mathcal{P} \ni \mathbf{p}$  denote the *parameter space*. We define an *objective function*  $o(\mathbf{p})$  (also called *cost function*), which is a mapping  $o : \mathcal{P} \rightarrow [0, 1]$ . The objective function measures the deviation of the actual output behavior of the network from the desired output behavior. We apply optimization algorithms to the objective function in order to find a parameter vector  $\mathbf{p}$ , so that the network performs the desired mapping as well as possible. Therefore, the problem of learning is mapped onto a general, nonlinear optimization problem.

Let  $\mathbf{y}^{[l]}(t, \mathbf{p})$  denote the vector of network output signals, when input signals  $\mathbf{u}^{[l]}(t)$  are fed into the network (including an associated initial state) and the network is operated with a parameter vector  $\mathbf{p}$ . Let  $\mathbf{d}^{[l]}$  denote the vector of desired cell output vectors associated with the input signals  $\mathbf{u}^{[l]}(t)$ . The distance measure  $\Delta^{[l]}(\mathbf{p})$  quantifies the deviation of the actual network response from the desired response for item  $l$

$$\Delta^{[l]}(\mathbf{p}) := \frac{1}{4} \sum_{t=1}^T \alpha^{[l]}(t) \left[ \sum_{\mu=1}^M \omega_{\mu} \left( y_{\mu}^{[l]}(t, \mathbf{p}) - d_{\mu}^{[l]} \right)^2 \right]. \quad (2)$$

$\omega_{\mu} \in [0, 1]$  and  $\alpha^{[l]}(t) \in [0, 1]$  are weighting factors used for attaching different importance to different cells of the network and for weighting the contributions to the objective function at different time-steps, respectively. Usually, the network behavior during the transient is of minor importance. Let  $T_{\text{tran}}^{[l]}$  and  $T_{\text{cyc}}^{[l]}$  be the transient length and cycle length of the network, respectively, when the input signals  $\mathbf{u}^{[l]}(t)$  are fed into the network. Since the DTCNN is a discrete-time system with binary output values,  $T_{\text{tran}}^{[l]}$  and  $T_{\text{cyc}}^{[l]}$  can be determined easily during the operation of the network. Oscillatory behavior is very common in DTCNNs, and therefore a reasonable strategy to set the  $\alpha^{[l]}(t)$  in practice is

$$\alpha^{[l]}(t) = \begin{cases} 0 & \text{for } 0 \leq t \leq T_{\text{tran}}^{[l]} \\ (T_{\text{cyc}}^{[l]})^{-1} & \text{for } T_{\text{tran}}^{[l]} < t \leq T_{\text{tran}}^{[l]} + T_{\text{cyc}}^{[l]} \end{cases}. \quad (3)$$

If oscillations cannot be tolerated, then it is possible to define  $\Delta^{[l]}(\mathbf{p}) = 1$  whenever  $T_{\text{cyc}}^{[l]} > 1$ , thereby punishing oscillatory behavior. Let  $L$  be the number of (input/desired output) pairs in the training set. Then the objective function  $o(\mathbf{p})$  is defined by

$$o(\mathbf{p}) := \sum_{l=1}^L \Omega_l \cdot \Delta^{[l]}(\mathbf{p}). \quad (4)$$

The coefficients  $\Omega_l \in [0, 1]$  are again weighting factors used for attaching different importance to different items  $l$ .

The global dynamics of the DTCNN are uniquely determined by the local mapping at cell level as given in Eq. 1. Since each cell of the DTCNN is a standard perceptron and the network parameters are translationally invariant, it follows from the existing theory on Threshold Logic that the parameter space  $\mathcal{P}$  of the DTCNN is segmented into a large, but finite number of convex cones [9]. Let the length of the parameter vector  $\mathbf{p}$  be  $N + 1$ . Then, the number of convex cones grows with  $O(2^N/N!)$ . Each of the convex cones corresponds to a possibly different global mapping of the DTCNN, and thus the behavior of the DTCNN and therefore the value of the objective function is constant for parameter vectors  $\mathbf{p}$  from one convex cone. At the boundaries between the convex cones, the behavior of the network and the objective function can change abruptly, and therefore the gradient  $\nabla_{\mathbf{p}}\alpha(\mathbf{p})$  is either zero or undefined. Since the number of possible network behaviors is finite, the problem of minimizing the objective function  $\alpha(\mathbf{p})$  belongs to the class of *combinatorial optimization* problems [10].

### 3 Algorithms

Due to the lack of gradient information, standard optimization methods like steepest-descent, conjugate-gradient, or quasi-Newton methods [11] cannot be applied. Exact, non-polynomial classes of algorithms used for the solution of many combinatorial optimization problems like *Partial Enumeration Schemes* (e.g. Branch-and-Bound methods) or *Polyhedral methods* (e.g. Cutting-Plane algorithms) [10] are also not applicable, because they require the existence of certain reasonably tight lower bounds on the value of the objective function for certain subsets of the search space. However, there are optimization algorithms that can deal with difficult objective functions like Eq. 4.

#### 3.1 HYBRID Algorithm

The HYBRID algorithm is a heuristically motivated algorithm, which relies on the experimental observation that the macro-structure of the error surface of the objective function is reasonably well-behaved and "smooth". We use *Direct Search* methods [12] (optimization methods not requiring gradient information) for the minimization of Eq. 4. These algorithms are generally less efficient from a computational point of view when compared to algorithms using first or higher order gradient information. A number of algorithms were tested, including the *Simplex* algorithm, *Rosenbrock's* method, the *Pattern Search* method, the *DSC* method, and a random search method [12]. It turned out that a combination of the Simplex algorithm (the modified version by Nelder and Mead) and Rosenbrock's method is the best alternative for the minimization of the objective function Eq. 4. Both methods are given in Appendix A.

The Simplex algorithm works quite well in the beginning of the optimization process, i.e. when the objective function is still high. This is due to the fact that the Simplex method scans large areas of the parameter space with relatively few evaluations of the objective function. If the algorithm has descended into narrow valleys on the error surface, the performance of the method deteriorates. In those cases, when the current point has a lower objective function value than most of the surrounding areas in the parameter space, Rosenbrock's method performs much better, because it relies on a local search of the neighborhood of the current point.

Thus for the HYBRID method, the Simplex method is used, until the objective function value reaches a certain lower bound. Then, the minimization is continued by Rosenbrock's method, using the best result of the Simplex method as the starting point for Rosenbrock's method. The optimal value for the switching point is problem-dependent, and it has to be found experimentally. However, the exact value of the switching point is not very critical. The HYBRID method has been applied successfully to several difficult learning problems, for which no solutions have been found before by design methods.

### 3.2 Simulated Annealing

Simulated Annealing (SA) methods have been introduced in 1983 as a flexible tool for difficult combinatorial optimization problems [13], and since then they have been applied successfully to many different practical optimization problems in various fields. The algorithm is computationally expensive, but it is very easy to implement, and many researchers report that it almost constantly comes up with approximate solutions of good quality. The SA algorithm is mainly specified by the choice of a *cooling schedule*, which determines, how the *system temperature*, an internal parameter, is changed. See [14] for a good and recent review of SA theory and cooling schedules.

SA methods were used to minimize the objective function Eq. 4. The algorithm is given in Appendix A. In absence of a more suitable method, the abstract states required for the SA algorithm were simply mapped onto the parameter vector  $p$  (non-injective mapping). Different cooling schedules were tested experimentally. It turned out that the quality of the final result depends mainly on the total number of function evaluations, not so much on the choice of a specific cooling schedule. SA methods come up with good solutions more constantly than the HYBRID algorithm, but they are worse in terms of execution time.

## 4 Conclusions

An objective function was defined, which is used to reformulate the learning problem for DTCNNs as a general optimization problem. The objective function was minimized by the HYBRID method, which is a combination of two Direct Search methods, and Simulated Annealing methods. By using this strategy, much more complex trajectories are feasible.

In a companion paper [8], both algorithms are used to find the template coefficients for a DTCNN in Texture Classification, Texture Segmentation, and Text Segmentation applications. The results underline the usefulness of the proposed learning methods.

### A PIDGIN ALGOL Description of the Algorithms

The SIMPLEX algorithm, Rosenbrock's method, and the Simulated Annealing algorithm are given in a PIDGIN ALGOL description [10]. In all cases,  $o_x$  is the objective function value corresponding to the parameter vector  $p_x$ . The function Gram-Schmidt denotes an orthonormalization process (starting with the first argument).

```
begin Simplex algorithm; /*  $\alpha=1.1, \beta=2, \gamma=0.6, \Delta\alpha_{\min}=0.005, \Delta_{\min}=10^{-6}$  */
  initialize_simplex(p[0], ..., p[N+1]); /* regular Simplex */
  do /* main loop */
    label_1 : relabel_vertices(p[0], ..., p[N+1]); /* then:  $o_{[0]} \leq \dots \leq o_{[N+1]}$  */
    if  $o_{[N+1]} - o_{[0]} < \Delta\alpha_{\min}$  /* best/worst vertex too close ? */
      then begin randomize(p[1], ..., p[N+1]); goto label_1; end
```

```

Pc = 1/(N+1) * (P[0] + P[1] + ... + P[N]); /* centroid */
if ||Pc - P[N+1]|| < Δmin /* simplex collapses ? */
  then begin randomize(p[N+1]); goto label_1; end
Pref = Pc + α(Pc - P[N+1]); /* reflected vertex */
if o_ref > o[0] and o_ref < o[N]
  then P[N+1] = Pref; /* accept reflection */
  else if o_ref ≤ o[0]
    then begin /* expanded vertex */
      Pexp = Pc + β(Pref - Pc);
      if o_exp ≤ o[0] then P[N+1] = Pexp; else P[N+1] = Pref;
    end
    else begin /* contracted vertex */
      if o_ref ≥ o[N+1]
        then Pcon = Pc + γ(P[N+1] - Pc);
        else Pcon = Pc + γ(Pref - Pc);
      if o_con < o_ref then P[N+1] = Pcon;
      else for ν = 1 to N + 1 do P[ν] = 1/2(P[ν] + P[0]);
    end
  end
until(termination_criterion);
end Simplex algorithm;

begin Rosenbrock's method; /* α=3, β=-0.5, δ=10-2 */
initialize(p_opt); randomize(r_0, ..., r_N); Gram-Schmidt(r_0, ..., r_N);
for ν = 0 to N do δ_ν = δ;
do /* main loop */
  Pcur = Popt;
  for ν = 0 to N do begin Δ_ν = 0; η_ν = 0; end
do /* exploratory phase */
  {i_0, ..., i_N} = rand.perm({0, ..., N}); /*create random permutation*/
  for ν = 0 to N do begin
    Pexp = Pcur + δ_i_ν · r_i_ν;
    if o_exp ≤ o_cur
      then begin /* success */
        Pcur = Pexp; Δ_i_ν = Δ_i_ν + δ_i_ν; δ_i_ν = α · δ_i_ν;
        if η_i_ν = 0 then η_i_ν = 1; /* record success */
      end
      else begin /* failure */
        Pexp = Pcur; δ_i_ν = β · δ_i_ν;
        if η_i_ν = 1 then η_i_ν = 2; /* record failure */
      end
    end
  end
end
until(η_ν = 2 ∀ ν = 0, ..., N);

/* reorthonormalization phase */
s_0 = Pexp - Popt; /* total progress from the exploratory phase */
Popt = Pexp; for ν = 1 to N do s_ν = s_{ν-1} - Δ_{ν-1} · r_{ν-1};
for ν = 0 to N do r_ν = s_ν; Gram-Schmidt(r_0, ..., r_N);
until(termination_criterion);
end Rosenbrock's method;

```

```

begin SimulatedAnnealing;
  initialize_state(scur); oopt = 1; T = T0;
  do /* outer loop */
    do /* METROPOLIS or inner loop */
      snew = generate_state(scur); /* generate new state */
      if onew < oopt then sopt = snew; /* save optimum state */
      if accept(onew, ocur, T) then scur = snew; /* accept new state */
    until(inner-loop criterion false);
    T = update_T(T); /* decrease temperature */
  until(outer-loop criterion false);
end SimulatedAnnealing;

```

## 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] S. Judd, "Learning in networks is hard," in *IEEE First International Conference on Neural Networks*, vol. 2, pp. 685–692, 1987.
- [4] 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.
- [5] P. Nachbar, *Robuster Entwurf von Neuronalen Netzen*. PhD thesis, Technical University Munich, Munich, Germany, Dec. 1993.
- [6] G. Seiler, *Grundlagen des Entwurfs Zellularer Neuronaler Netze*. PhD thesis, Technical University Munich, Munich, Germany, Feb. 1993.
- [7] 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.
- [8] A. Kellner, H. Magnussen, and J. A. Nossek, "Texture classification, texture segmentation and text segmentation with discrete-time cellular neural networks (*to appear*)," in *Proc. Third IEEE International Workshop on Cellular Neural Networks and their Applications CNNA-94*, (Rome, Italy), Dec. 1994.
- [9] 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.
- [10] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1. ed., 1982.
- [11] R. Fletcher, *Practical Methods of Optimization*. John Wiley and Sons, 2 ed., 1987.
- [12] W. Swann, "Direct search methods," in *Numerical Methods for Unconstrained Optimization* (W. Murray, ed.), ch. 2, pp. 13–28, Academic Press, 1972.
- [13] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
- [14] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica*, vol. 6, no. 3, pp. 302–345, 1991.