LEARNING ALGORITHMS FOR CELLULAR NEURAL NETWORKS

Bahram Mirzai, Zhenlan Cheng and George S. Moschytz

Signal and Information Processing Laboratory
Swiss Federal Institute of Technology
Zurich, Switzerland
mirzai@isi.ee.ethz.ch

ABSTRACT

A learning algorithm based on the decomposition of the A-template into symmetric and anti-symmetric parts is introduced. The performance of the algorithm is investigated in particular for coupled CNNs exhibiting diffusion-like and propagating behavior.

1. INTRODUCTION

Cellular neural networks (CNNs) are examples of recurrent networks defined by the following system of differential equations

$$\frac{dx_{ij}(t)}{dt} = -x_{ij}(t) + \sum_{mn\in\mathcal{N}ij} a_{mn}y_{mn}(t) + \sum_{mn\in\mathcal{N}ij} b_{mn}u_{mn} + I\ ,$$

where $\mathcal{N}_{ij}$ denotes the neighborhood of the ij-th cell for $1 \le i \le M$, $1 \le j \le N$ and $y = (|x+1| - |x-1|)/2$. The state, input and output of a cell are defined by x_{ij}, u_{ij} and y_{ij}, respectively. We assume a nearest neighborhood CNN. The output at an equilibrium point, when one exists, is denoted by y^*_{ij}. The parameters of a CNN are gathered into the so-called A-template, the B-template and the bias I.

In view of learning algorithms, since a CNN is a recurrent neural network, one can apply the learning algorithms known for this class of networks [1], such as the recurrent back-propagation algorithm (RBA) by replacing the non-linear function sat$(\cdot)$ by a smoother similar one [2]. Algorithms like RBA are based on the minimization of a cost function E that compares the output of the network y^*_{ij} with the desired output d_{ij}:

$$\mathrm{E} = \frac{1}{2} \sum_{1\le i\le M, 1\le j\le N} (y^*_{ij} - d_{ij})^2\ . \tag{1}$$

Comparisons of this kind inherently neglect the evolution of a cell from its initial state to the final state. However, the structure of a template set depends crucially on the kind of dynamics involved, and vice-versa, e.g., symmetric templates are normally assumed for diffusion-type applications. Hence a learning algorithm only based on (1) may not be capable of tracing a desired set of parameters, if it is not provided with information of this kind.

A different approach may be taken by minimizing the cost function

$$\mathrm{E} = \frac{1}{2}\int_0^T \sum_{1\le i\le M, 1\le j\le N} \big(x_{ij}(t) - d_{ij}(t)\big)^2 dt\ . \tag{2}$$

Although this approach takes into account the temporal evolution of the system, it would also not be an appropriate one in the context of CNNs. In fact, its definition requires a set of desired trajectories $d_{ij}(t)$ which, in turn, implicitly assumes a template set of the task to be trained for. Moreover, the algorithms based on (2) require larger computational storage capabilities than RBA [1,2].

In this paper we introduce a class of learning algorithms that seeks to minimize (1), on the one hand, and to take into account the dynamic nature of the particular task, on the other. The algorithm considered here was originally motivated by [3]. We first introduce a modified version of the learning algorithm described in [3] and extend it to more involved applications. Simulations are provided for a number of coupled CNNs. For later convenience, we adopt the following notation and apply the notion of "weight" to the parameters of a template set

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}\ .$$

2. A BACKPROPAGATION-LIKE LEARNING ALGORITHM

Let the error $e_{ij}[k]$ of the cell $\mathcal{C}_{ij}$ at the iteration step k be defined by

$$e_{ij}[k] = \frac{1}{2}\big(d_{ij} - y^*_{ij}[k]\big)\ . \tag{3}$$

Analogously to the perceptron learning algorithm, define an updating of the weights according to

$$\begin{aligned} a_{mn}[k+1] &= a_{mn}[k] + \eta\Delta a_{mn}[k] \\ b_{mn}[k+1] &= b_{mn}[k] + \eta\Delta b_{mn}[k] \\ I[k+1] &= I[k] + \eta\Delta I[k] \end{aligned} \tag{4}$$

with

$$\begin{aligned} \Delta a_{mn}[k] &= \begin{cases} 0 & \text{if } m=n=2, \\ \dfrac{1}{MN}\displaystyle\sum_{1\le i\le M, 1\le j\le N} e_{ij}[k]\, y^*_{i+m-2\,j+n-2}[k] & \text{else} \end{cases} \\ \Delta b_{mn}[k] &= \frac{1}{MN}\sum_{1\le i\le M, 1\le j\le N} e_{ij}[k]\, u_{i+m-2\,j+n-2}[k] \\ \Delta I[k] &= \frac{1}{MN}\sum_{1\le i\le M, 1\le j\le N} e_{ij}[k]\ , \end{aligned} \tag{5}$$

where $m,n \in \{1, 2, 3\}$, and the learning rate $\eta > 0$. By (5), at a given time step k, the change in a template parameter or the

0-7803-4455-3/98/$10.00 (c) 1998 IEEE

bias is obtained by taking the product of the error and the corresponding "input", and then averaging over all cells. In particular, a constant input of unity is assigned to the bias.

The algorithm (4) differs in two ways from the original one proposed in [3]. First, we do not update the center entry a_{22} but rather set it initially to some fixed value $a_{22}[0] \geq 1$. This is motivated by the fact that usually one is interested in *bipolar* outputs. Moreover, an update of a_{22} according to

$$a_{22}[k+1] = a_{22}[k] + \eta \frac{1}{MN} \sum_{1 \leq i \leq M, 1 \leq j \leq N} e_{ij}[k] y^*_{ij}[k] , \quad (6)$$

as proposed in [3], may lead to inconsistencies, since (6) updates a_{22} only in one direction, namely it either increases a_{22} by some *positive* fraction of η or leaves it unchanged. Other approaches may be applied to include a_{22} in the training procedure as well [4]. Second, we have introduced an update of the bias I which is a required parameter for many applications.

Beside the analogy of (4) to the perceptron learning algorithm, (4) can be motivated by the following observation. A cell $\mathcal{C}_{ij}$ contributes to the update of weights only when $e_{ij} \neq 0$. In the case of $e_{ij} = 1$, y^*_{ij} is 1 and it should become -1. Therefore, to update, say a_{mn}, we seek to decrease a_{mn} by η for those cells $\mathcal{C}_{mn}$ in the neighborhood $\mathcal{N}_{ij}$ with $y^*_{mn} = 1$ and increase it by η for those with $y^*_{mn} = -1$. Averaging over all cells then provides the amount by which updating is done. Similar reasoning applies to the case of $e_{ij} = -1$.

In contrast to the back-propagation algorithm, the algorithm (4), as a whole, cannot be derived as a gradient descent algorithm by minimizing the cost function (1). However, if the CNN were only to operate in its uncoupled mode, then an update of the B-template by (4) would correspond to the back-propagation algorithm obtained by minimizing (1) with respect to the weights. Furthermore, we note that (4) corresponds to "batch" training, meaning that we first average over the partial updates of cells due to the corresponding input/output patterns, and then perform the updating of the weights. This is in contrast to an "on-line" training, where an update of the weights is performed after the presentation of each input/output pattern.

As previously mentioned, a shortcoming of the algorithm (4) is that it is based only on the information contained in the comparison of the output y^*_{ij} with the desired output d_{ij} and that it neglects the temporal evolution of the system. In some instances the output alone may not provide sufficient information to obtain a desired template set. Particularly for those tasks that exhibit propagating solutions, such as connected component detection or shadowing, the algorithm (4) tends to fail. In the following section we seek to extend this algorithm in a direction that allows us to partially overcome this shortcoming.

3. THE CENTER OF MASS LEARNING ALGORITHM

To overcome the limitations of the algorithm (4) we propose a modified version which differs from (4) in the update of the A and B-templates. The bias will still be updated as in (4). For reasons to follow, we will denote the new algorithm as the *center of mass algorithm* (CMA).

The update of the B-template is essentially done according to (4) with the only difference that we now require the updating to respect the symmetry of the B-template. In other words, if the nature of a task suggests a B-template of the form

$$B = \begin{bmatrix} 0 & b & 0 \\ b & b_c & b \\ 0 & b & 0 \end{bmatrix} ,$$

then it is updated by

$$\Delta B[k] = \begin{bmatrix} 0 & \Delta b[k] & 0 \\ \Delta b[k] & \Delta b_{22}[k] & \Delta b[k] \\ 0 & \Delta b[k] & 0 \end{bmatrix} ,$$

with

$$\Delta b[k] = \frac{\Delta b_{12}[k] + \Delta b_{21}[k] + \Delta b_{23}[k] + \Delta b_{32}[k]}{4}$$

and $\Delta b_{mn}[k]$ as given in (5).

To update the A-template, we consider it as being composed of three parts $A = A_c + A_s + A_a$, with

$$A_s = \frac{1}{2} \begin{pmatrix} a_{11}+a_{33} & a_{12}+a_{32} & a_{13}+a_{31} \\ a_{21}+a_{23} & 0 & a_{21}+a_{23} \\ a_{13}+a_{31} & a_{12}+a_{32} & a_{11}+a_{33} \end{pmatrix} ,$$

$$A_a = \frac{1}{2} \begin{pmatrix} a_{11}-a_{33} & a_{12}-a_{32} & a_{13}-a_{31} \\ a_{21}-a_{23} & 0 & a_{23}-a_{21} \\ a_{31}-a_{13} & a_{32}-a_{12} & a_{33}-a_{11} \end{pmatrix} ,$$

and $A_c = A - A_s - A_a$. The *symmetric* part A_s is then considered to account for the local and diffusion-like dynamics and the *anti-symmetric* part A_a for the global and propagation-like dynamics. Correspondingly, we apply different updating procedures to A_s and A_a.

The symmetric part A_s is updated as $A_s[k+1] = A_s[k] + \eta \Delta A_s[k]$, with

$$\Delta A_s = \frac{1}{2} \begin{pmatrix} \Delta a_{11}+\Delta a_{33} & \Delta a_{12}+\Delta a_{32} & \Delta a_{13}+\Delta a_{31} \\ \Delta a_{21}+\Delta a_{23} & 0 & \Delta a_{21}+\Delta a_{23} \\ \Delta a_{13}+\Delta a_{31} & \Delta a_{12}+\Delta a_{32} & \Delta a_{11}+\Delta a_{33} \end{pmatrix} ,$$

where Δa_{mn} is given by (5) and for compactness we have left out the iteration step $[k]$. Furthermore, the updating is assumed to respect the initial symmetry of A_s.

To update the anti-symmetric part A_a, we first define the notion of the *center of mass*. The center of mass of a 2-dimensional grid with positive masses m_{ij} assigned to vertices along an axis l is defined by

$$r_l = \frac{1}{M_{\text{total}}} \sum_{1 \leq i \leq M, 1 \leq j \leq N} D(l)_{ij} \, m_{ij} , \quad (7)$$

where $D(l)_{ij}$ is the distance of $\mathcal{C}_{ij}$ from l and

$$M_{\text{total}} = \sum_{1 \leq i \leq M, 1 \leq j \leq N} m_{ij} .$$

is the total mass.

We now assign to each cell on the CNN grid a "mass" m_{ij} defined by

$$m_{ij} = \frac{1 + y^*_{ij}}{2} .$$

Note that $0 \leq m_{ij} \leq 1$, where the lower bound is obtained for a white cell and the upper bound for a black cell. Consider now the centers of mass with respect to the axes indicated in Fig. 1:

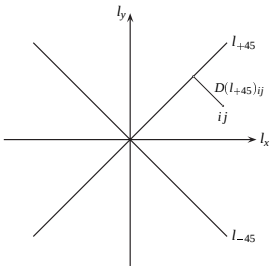


Figure 1: *Center of Mass Axes*

$$
\begin{aligned}
r_x &= \frac{1}{M_{\text{total}}} \sum_{1\le i\le M,\,1\le j\le N} j\,m_{ij} \\
r_y &= \frac{1}{M_{\text{total}}} \sum_{1\le i\le M,\,1\le j\le N} i\,m_{ij} \\
r_{45} &= \frac{1}{M_{\text{total}}} \frac{\sqrt{2}}{2} \sum_{1\le i\le M,\,1\le j\le N} |i-j|\,m_{ij} \\
r_{-45} &= \frac{1}{M_{\text{total}}} \frac{\sqrt{2}}{2} \sum_{1\le i\le M,\,1\le j\le N} (i+j)\,m_{ij}\ .
\end{aligned}
\tag{8}
$$

The center of mass r_{45} is obtained by calculating the distance $D(l_{45})_{ij}$ of the cell C_{ij} from the corresponding axis l_{45}:

$$
D(l_{45})_{ij} = \sqrt{(\frac{i+j}{2}-j)^2+(\frac{i+j}{2}-i)^2} = \frac{\sqrt{2}}{2}|i-j|\ .
$$

Similarly, r_{-45} can be obtained.

The update of the anti-symmetric part A_a can now be formulated as $A_a[k+1] = A_a[k] + \eta \Delta A_a[k]$, where

$$
\Delta A_a[k] = \begin{pmatrix} \Delta_{-45} & \Delta_y & \Delta_{45} \\ \Delta_x & 0 & -\Delta_x \\ -\Delta_{45} & -\Delta_y & -\Delta_{-45} \end{pmatrix}\ , \tag{9}
$$

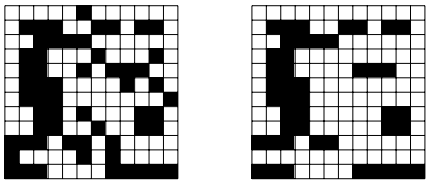
and

$$
\begin{aligned}
\Delta_x &= r_x^d[k] - r_x^{y^*}[k] \qquad & \Delta_{45} &= r_{45}^d[k] - r_{45}^{y^*}[k] \\
\Delta_y &= r_y^d[k] - r_y^{y^*}[k] \qquad & \Delta_{-45} &= r_{-45}^d[k] - r_{-45}^{y^*}[k]\ .
\end{aligned}
\tag{10}
$$

The superscripts d and y^* indicate the centers of mass of the desired output and the CNN output, respectively. By (10), the update of the anti-symmetric part A_a becomes small as soon as the centers of mass of the desired image and of the CNN output along the considered axes almost coincide. In other words, CMA seeks to update the weights in a direction that brings the centers of mass of y^* and d closer to each other.

4. SIMULATION RESULTS

In the following we investigate the performance of the CMA for a number of tasks. In view of an analog implementation, the search space for template parameters is confined to $[-4, 4]$, i.e., if during the training procedure a template parameter leaves the specified region, it is reset to the corresponding boundary value, e.g., 4.2 is reset to 4.0. Parameters are randomly initialized in $[-4, 4]$ and the initialization is assumed to obey the symmetry of a template set, e.g., if the A-template is assumed to be anti-symmetric, so will be its initialization. The boundary of the CNN is set to -1. Adaptation is terminated after 60 steps if no desired template set is yet obtained. The convergence times given below are to be understood in relation to the given input/output images used for the training; other sets of images may result in other convergence times.



(a) Input (b) Output

Figure 2: *Training Set*

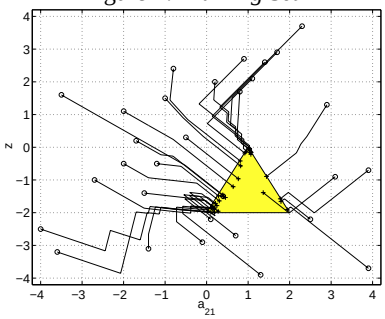


Figure 3: *Template Parameter Trajectories*

4.1. Horizontal Line Detection (HLD)

Assume the template set $A = [a_{21}\ \ a_{22}\ \ a_{23}]$, $I = z$. In a first approach, we take the A-template to be symmetric ($a_{21} = a_{23}$) and apply the CMA to train a_{21} and z. The center entry is fixed at $a_{22} = 3$. The A-template is updated such that its symmetry is preserved, i.e.,

$$
\Delta A = [\frac{\Delta a_{21} + \Delta a_{23}}{2}\ \ 0\ \ \frac{\Delta a_{21} + \Delta a_{23}}{2}]\ .
$$

The network is trained with the input/output images given in Fig. 2. We consider 30 runs with the learning rate being set to $\eta = 2$. The CMA converges in all 30 instances to template sets performing HLD correctly. Fig. 3 shows the trajectories of the weights with their initial and final values indicated by circles ($\circ$) and plus signs ($+$), respectively. The filled polygon in Fig. 3 indicates the region of correct operation [5].

To investigate the dependency of the convergence time as a function of η, we run the algorithm with various learning rates. An increase in the learning rate results in a decrease of the mean convergence time, Fig. 4.

In a second approach, we set $a_{22} = 3$ and updated the remaining parameters of the A-template and the bias according to (4), where the updating is not necessarily symmetric. For $\eta = 2$ we obtain convergence within 60 iterations in all runs except two. The average number of iterations is 15.67.

4.2. Shadowing (SH)

By the nature of the task, we assume an asymmetric A-template and train it by means of the input/output images given in Fig. 5. First, we consider the template set $A = [0\ \ 2\ \ a_{23}]$, $I = z$.

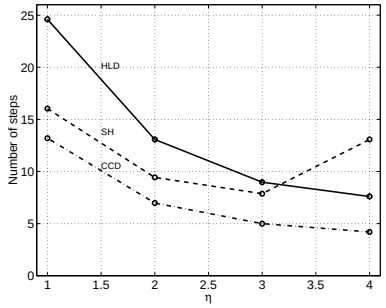
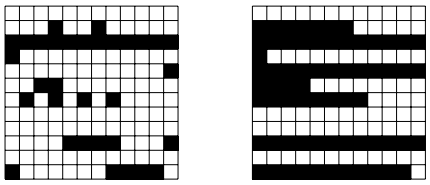


Figure 4: *Convergence Time versus Learning Rate*



(a) Input (b) Output

Figure 5: *Training Set*

By [5], the filled polygon in Fig. 6 denotes the desired region of the parameters. The A-template is trained according to $\Delta A = [0 \;\; 0 \;\; -\Delta_x]$ and the bias according to (5) with $\eta = 2$. As shown in Fig. 6 the algorithm converges to the desired region for all 30 runs. Furthermore, we investigate the average convergence time of the 30 runs as a function of η. The results are depicted in Fig. 4. In the case of $\eta = 4$, due to an overly high learning rate, the updates are more likely to shift the weights passed the desired region to the undesired part, hence the increase in the convergence time.

As a second approach, we assume the asymmetric template set $A = [a_{12} \;\; 2 \;\; a_{23}]$, $I = z$ and update the A-template according to

$$\Delta A = [\frac{\Delta a_{21} + \Delta a_{23}}{2} + \Delta_x \;\; 0 \;\; \frac{\Delta a_{21} + \Delta a_{23}}{2} - \Delta_x] \, .$$

The bias is updated as before and $\eta = 2$. The algorithm converges for all 30 runs with an average convergence time of 19.23 steps.

4.3. Connected Component Detection (CCD)

The nature of the task suggests a template set with a nearest neighbor anti-symmetric A-template $A = [a_{21} \;\; a_{22} \;\; a_{23}]$, where $a_{23} = -a_{21}$. We, further, assume a bias term $I = z$

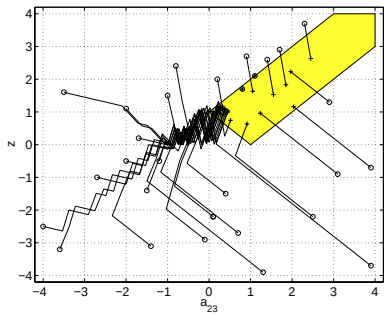
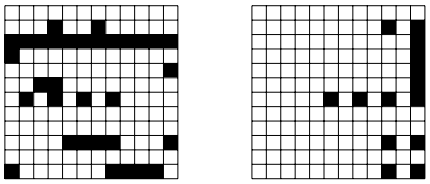


Figure 6: *Weight Trajectories*



(a) Input (b) Output

Figure 7: *Training Set*

and set $a_{22} = 3$. The A-template is updated according to $\Delta A = [\Delta_x \;\; 0 \;\; -\Delta_x]$.

Simulations are performed with input/output images given in Fig. 7 for different values of η as shown in Fig. 4. The algorithm converges for all values of η and for all 30 runs.

A simpler search problem is obtained by excluding the bias term, but maintaining the anti-symmetry, i.e., $A = [a_{21} \; 3 \; -a_{21}]$. The region for the CCD operation [5] is then determined by $a_{21} > 1$. The CMA is applied to train a_{21} with $\eta = 2$. Convergence is obtained for all 30 runs with a mean convergence time of 2.93 steps. Note that in this example, the probability of the random initialization already performing the correct task is 60%.

5. CONCLUSIONS

A new algorithm based on the decomposition of the A-template into symmetric and anti-symmetric parts is introduced. The updating procedure takes the particular dynamics implied by each part into account. Symmetric and anti-symmetric updates, the latter based on the evolution of the center of mass, are used to train the weights of the symmetric and anti-symmetric parts, respectively. The performance of the algorithm is investigated for a number of tasks requiring coupled CNNs.

6. REFERENCES

[1] John Hertz, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, 1991, ISBN 0-201-51560-1.

[2] Josef A. Nossek, "Design and Learning with Cellular Neural Networks," in *IEEE International Workshop on Cellular Neural Networks and their Applications*, Rome, Dec. 1994, pp. 137–146.

[3] C. Güzeliş and S. Karamahmut, "Recurrent Perceptron Learning Algorithm for Completely Stable Cellular Neural Networks," in *IEEE International Workshop on Cellular Neural Networks and their Applications*, Rome, Dec. 1994, pp. 177–182.

[4] Martin Hänggi and George S. Moschytz, "Genetic Optimization of Cellular Neural Networks," in *IEEE International Conference on Evolutionary Computation*, Anchorage, 1998, *accepted for publication.*

[5] Bahram Mirzai, Drahoslav Lím, and George S. Moschytz, "Robust CNN Templates: Theory and Simulations," in *IEEE International Workshop on Cellular Neural Networks and their Applications*, Seville, June 1996, pp. 393–398.