FINITE ITERATION DT-CNN – NEW DESIGN AND OPERATING PRINCIPLES

C. Merkwirth

Computational Biology and
Applied Algorithmics Group
Max-Planck-Institut für Informatik
66123 Saarbrücken, Germany
e-mail: `cmerk@mpi-sb.mpg.de`

J. Bröcker, M. Ogorzałek, J. Wichard

Department of Electrical Engineering
AGH University of Science and Technology
al. Mickiewicza 30
30-059 Kraków, Poland
e-mail: `maciej@agh.edu.pl`

ABSTRACT

In this paper we propose to use the Discrete-Time Cellular Neural Network (DT-CNN) in a finite iterate mode. In such a mode of operation no special requirements on template stability properties are needed. We propose a constructive back propagation based algorithm for template design. For a given number of iterations we can find optimal sequence of templates for a given problem to be solved. Our novel approach is demonstrated by design of a digit recognition DT-CNN.

1. INTRODUCTION

When first introduced by Chua and Yang [1] Cellular Neural Networks were considered as a large array of simple processing elements – continuous time dynamical systems – characterized by only local connectivity. The local connections defined by so called templates were fixed throughout the network and kept constant when performing a single operation. Further a discrete time version of the CNN principle has been introduced (DT-CNN) [2]. In both proposed types of networks the underlying principle is that computation is done by system dynamics, i.e. the response of the network to given stimuli is defined by the limit behavior in time – convergence of all responses to some fixed point [3]. The achieved steady state is predefined by the connection templates and by the the network input and initial conditions.

Here we propose to use the DT-CNN as a “finite iteration” computing device. The network will perform a fixed number of iterations in contrast to the standard convergence approach. We propose also to use a different template design strategy the main feature of which is to use for each iteration a different template. It should be stressed that in the finite iterate approach we can drop the requirement that templates guarantee convergence of the network to a fixed point – no requirements on stability, symmetry or any other template properties are needed [4].

We propose also a constructive approach to design the templates – a learning strategy. For the finite iterate DT-CNN this is based on optimization and use of advanced back propagation techniques. Fixing the number of iterates as a design parameter we can run the optimization algorithm which will find an optimal sequence of templates minimizing a suitably chosen cost function of the considered problem (eg. pattern recognition).

2. THE CNN TOPOLOGY

The states of the DT-CNN cells y_{ij}^t evolve for iterations $t = 0, \dots, T-1$ according to:

$$\begin{aligned} x_{ij}^{t+1} &= \sum_{l,m=-\frac{K-1}{2}}^{\frac{K-1}{2}} \Big(A_{l,m}^t y_{i+l,j+m}^t \\ &\qquad + B_{l,m}^t u_{i+l,j+m}\Big) + b^t \\ y_{ij}^{t+1} &= \sigma(x_{ij}^{t+1}), \end{aligned} \tag{1}$$

wherein the u_{ij} denote the input pattern and $\sigma(x_{ij})$ is the activation function, further discussed in subsection 2.1. If we use different template weights (A^t, B^t and offsets b^t) in each iteration t, we can massively improve the performance of the DT-CNN in classification tasks. In this case, the total number P of adjustable scalar parameters can be calculated by $P = T(2K^2 + 1) - K^2$, where T is the number of *template layers*, and K is the number of rows of the quadratic template matrices. For eg $K = 5$, this results in 51 weights per template layer. A^0 is not used since $y_{ij}^0 = 0$ by initialization.

2.1. The activation function

The activation function $\sigma(x)$ depicted in Figure 1 is a piecewise linear function that does not fully saturate for input

Parts of this work are supported by the Research Training Network *COSYC of SENS* No. HPRN-CT-2000-00158 within the 5th Framework Program of the EU. We would like to thank the people of the AGH University of Science and Technology for their support.

values x far from the origin [5]. This feature ensures a still nonvanishing derivative of the training error with respect to very large template entries. A training algorithm exploiting derivative information can thus escape from such usually quite suboptimal solutions. In contrast, networks having a saturating sigmoid activation function may cause the training algorithm to get stuck at such places, for the gradient almost vanishes for large arguments. Since this feature is only relevant for the training, the slope far from the origin can be set to converge to zero during the backpropagation epochs.

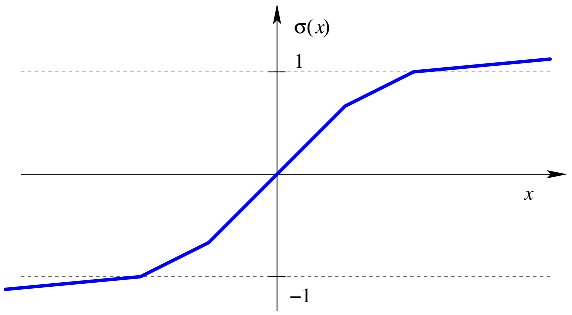

Fig. 1. A piecewise linear activation function is used as nonlinearity. Note that the function does not saturate at ± 1, but the slope decreases to 0.01. The singularity of the derivative at isolated points does not adversely affect a gradient based training algorithm.

3. THE DT-CNN CLASSIFIER

This paper is concerned with using a DT-CNN as a classifier. In a binary decision problem we present an input pattern u to the receptive field of the DT-CNN, we want to get a decision whether this input pattern belongs to a specific class or not. Since the DT-CNN is a spatial processing engine, we have to convert the spatial output y_{ij}^T into one output value suitable as decision variable. This is done by simply averaging these 256 states y_{ij}^T and comparing the result to a certain threshold (usually zero).

In the case of *Handwritten Digit Recognition* the task is to identify handwritten digits (from zero to nine) from a 16 by 16 pixel grayscale image. This obviously is a classification problem with *ten* instead of two classes. This multiclass classification problem can be converted into 10 binary classification problems to be solved independently as follows. The first classifier has to distinguish the zeros from the rest, the second the ones from the rest etc. All ten classifiers are trained on the same input patterns. The desired outputs presented to the first classifier are $+1$ for input patterns showing a zero, otherwise -1. For the second classifier the desired outputs are $+1$ for observations for input patterns showing a one, and so on.

4. TRAINING OF DT-CNN CLASSIFIERS

The training of DT-CNN classifiers is based on minimisation of a function henceforth called the *training error*. The training error depends on the training data and the network templates and is minimized with respect to the latter. More specifically, let $(z^i, u^i), i = 1 \dots N$ be the training data consisting of input output pairs. Further, denote by $f(u^i; A, B, b)$ the output of DT-CNN classifier, where A, B and b stand for the entity of the templates. The training error then has the form

$$E(A,B,b) := \sum_{i=1}^{N} \lambda(z_i - f(u^i; A, B, b)). \tag{2}$$

The *loss function* λ measures the deviation of the CNN output from the desired value z^i. In optimisation usually a quadratic loss function is used, basically due to the simplicity of the resulting derivatives. We however advocate the use of an *ϵ–insensitive absolute loss function*. The advantages of this strategy are subject of the next subsection. Further subjections are concerned with the details of the optimisation procedure. The training of networks with shared weights requires an augmented error function. Although a shared weight network may be initialized with equal templates in each layer and trained as such, we found this procedure to be quite unstable. A better way to train such networks is to initialize them with different weights and include the following penalty term

$$D(A,B,b) := \sum_{t=2}^{T} \lambda(A^{t-1} - A^t) + \lambda(B^{t-1} - B^t) + \lambda(b^{t-1} - b^t)$$

into the training error. Here λ is to be understood as applied componentwise and then summed. The total error to be minimized is the convex sum

$$TE(A,B,b) := (1-\gamma)E(A,B,b) + \gamma D(A,B,b).$$

4.1. The ϵ–insensitive loss function

We use an *ϵ–insensitive absolute loss function* $\lambda(\xi)$ to calculate the training error as plotted in Figure 2. The output of the DT-CNN has zero loss and gradient if it lies inside the ϵ–margin of the desired output. This forces the algorithm to learn the misclassified training patterns instead of adjusting the weigths with gradient steps of already correctly classified patterns. Training patterns that cannot be correctly classified have only a linear contribution to the loss. This provides an apropriate tradeoff between tolerating outliers and penelizing classification errors. In our case, we use $\epsilon = 0.9$ which seems to be very high, but a smaller ϵ degrades the classification performance. The use of ϵ–insensitive loss functions in the context of learning problems is described in [6].

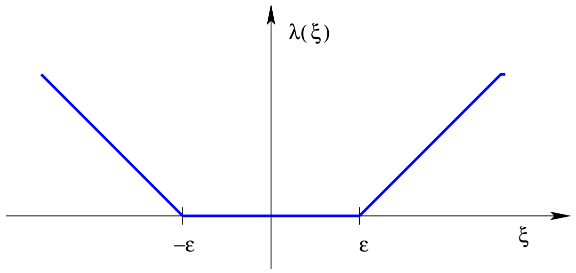

Fig. 2. The ϵ-insensitive loss function used for the training of the DT-CNNs. Though the derivative of the loss function is not continuous, it can be used in the backpropagation training without problems.

4.2. Stochastic gradient descent and back propagation

The method of training a Neural Network considered in this paper is known as *stochastic gradient descent* or *online learning*. The templates are updated recursively according to the rule

$$\begin{aligned} A_n &= A_{n-1} - \delta A_n, \\ B_n &= B_{n-1} - \delta B_n, \\ b_n &= b_{n-1} - \delta b_n, \end{aligned} \qquad (3)$$

where $n = 1 \ldots N$, the number of training samples. The update δA_n depends on the nth training sample only. Namely, we set

$$\begin{aligned} \delta A_n &= \mu \frac{\partial}{\partial A} \lambda(z_n - f(u^n; A_{n-1}, B_{n-1}, b_{n-1})), \\ \delta B_n &= \mu \frac{\partial}{\partial B} \lambda(z_n - f(u^n; A_{n-1}, B_{n-1}, b_{n-1})), \\ \delta b_n &= \mu \frac{\partial}{\partial b} \lambda(z_n - f(u^n; A_{n-1}, B_{n-1}, b_{n-1})), \end{aligned}$$

Thus, in contrast to batch learning, where gradients are accumulated over all samples of the data set for the *same template setting* before a gradient step is performed, a tiny gradient step is done here after each observation has been processed. A sweep through the whole data set is called *epoch*. The global stepsize $\mu = 0.0007$ is exponentially decreased every five epochs by a factor of 0.8. For every epoch the order in which the samples of the training data set are processed is randomized. In all simulations below, the number of training epochs is fixed to 50.

Note that, in contrast to batch learning, the algorithm will not terminate even in a global minimum. Nonetheless, it may prvide good solutions: The parameters classifying the first sample correctly usually provide a reasonable guess for the second sample, and, after being updated again, will provide a good guess for the third sample and so on. In a linear problem, online learning provides the optimal solution in one epoch.[1] The algorithm does not terminate as well, but more epochs will yield spurious results (overfitting). In a nonlinear setup more epochs but with smaller steps are required. The stochastic gradient descent furthermore does not suffer from getting stuck in local minima.

A detailed discussion of the advantages of stochastic gradient descent over batch learning can be found in [7].

In the training of shared weight networks, the second term in the error function has to be taken into account. The update rule for the templates in online learning in this case is

$$\begin{aligned} \delta A_n &= \mu(1-\gamma) \frac{\partial}{\partial A} \lambda(z_n - f(u^n; A_{n-1}, B_{n-1}, b_{n-1})) \\ &\quad + \gamma \frac{1}{P} \frac{\partial}{\partial A} D(A, B, b), \end{aligned}$$

and similarly for B and b, where P is the number of parameters.

4.3. Shrinking

A reduction in training time can be achieved by using a heuristics called *shrinking*. The evaluation of training samples that are easy to classify is postponed, while the algorithm concentrates on samples that are harder to classify. In detail this means that if a training observation repeatedly yields a vanishing classification error (i.e. is inside the ϵ–interval of the loss function), it is disclosed from the training set for a certain number of epochs determined by the following rules. For each sample, store two counters C_1 and C_2. Initially, both counters are equal to one. At the beginning of any epoch, decrement C_1 by one. If C_1 becomes zero, evaluate the sample. If the error turns out to be inside the ϵ-insensitive margin, increase C_2 by one and then reset C_1 to the value of C_2. However, if an observation is outside the ϵ-insensitive margin, reset both C_1 and C_2 to 1. Then start the new epoch.

4.4. Building classifier ensembles

A common way to improve the performance of classifiers is ensemble building (see [8] and the references therein). This is done by training several (6-16) DT-CNNs on randomly choosen subsets of the training data where the training starts with random weight initializations. To compute the output of the ensemble for one input pattern, the output decision variables of all DT-CNNs belonging to that ensemble are averaged. The averaging is not done on the template weights. Please note that we have to construct one ensemble for each of the ten binary classification tasks.

[1] This however requires a more sophisticated choice of the stepsize μ, namely the *Kalman gain*

5. DIGIT RECOGNITION – RESULTS

The ZIP Code Data Set[2] consists of normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service. The images, originally binary and of different sizes and orientations, have been deslanted and size normalized, resulting in 16 by 16 grayscale images (see [7]). There are 7291 training observations and 2007 test observations, distributed as follows: The 2009 samples of the ZIP Code test set are neither used for training nor to derive stopping criteria or for model selection for classifier ensembling. The patterns are presented to the training algorithm as pairs together with the desired output, i.e. the class label out of 0,1,. . . ,9.

Digit	0	1	2	3	4	5	6	7	8	9
Train	1194	1005	731	658	652	556	664	645	542	644
Test	359	264	198	166	200	160	170	147	166	177

Table 1. Distribution of classes in the ZIP Code data sets

5.1. Classification Performance

Computational demand did not allow for a systematic check of all possible combinations of the template size K and the number of layers T. So $K = 5$ was used for all simulations (see table 2). With standard nearest-neighbor coupling $K = 3$ we could not achieve an adequate classification performance, for template size $K = 7$ we observed training and test errors starting to disagree (overfitting).

From Table 2 one can observe that varying T from one to ten layers, the classification rate increases significantly. For more template layers the rate saturates around 96.5%. For comparison, we trained a polynomial SVM classifier(see [9])of degree 4 with $C = 100$ on the ZIP Code training set and applied it to the test set.

T	1	2	3	4	5	6	8	10	14	18	24	SVM
Rate [%]	79.7	91.6	94.9	95.9	96.0	96.1	95.6	96.5	96.6	96.2	96.6	95.4

Table 2. Classification rates of final classifiers on the ZIP Code test set versus number of template layers T. Note that uniform random guessing yields a trivial classification rate of 10%.

6. CONCLUSIONS

This article discusses a new class of Discrete-Time Cellular Neural Networks which operate in the finite iterate regime. This new approach does not rely on convergence and stability of the network dynamics but uses the CNN as a one-step computing engine. The templates are designed using a minimization approach. As an application example we have confirmed excellent performance of a DT–CNN classifier for digit recognition. The results on a database of handwritten digits outperform a polynomial Support Vector classifier. A main drawback of the method is the higher computational effort for training ensembles of DT-CNN classifiers than for SVM classifiers. Nonetheless, the authors hope that a high speed hardware implementation can be realized based on the templates determined by the proposed method.

7. REFERENCES

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

[2] H. Harrer and J.A. Nossek, "Discrete-time cellular neural networks," *Int. J. Circuit Theory and Applications*, vol. 20, pp. 453–467, 1992.

[3] A. Kilinc S. Arik and F. Acar Savaci, "Global asymtotic stability of discrete-time cellular neural networks," in *Proc. of IEEE Int. Workshop on Cellular Neural Networks and their Appl.*, 1998, pp. 52–55.

[4] Á. Zárandy, "The art of CNN template design," *Int. J. Circuit Theory and Applications - Special Issue: Theory, Design and Applications of Cellular Neural Networks*, vol. 17, pp. 5–24, 1999.

[5] Y. LeCun et al, "Efficient BackProp," in *Neural Networks: Tricks of the trade*, G. Orr and K. Mller, Eds., vol. 1524 of *Lecture Notes in Computer Science*, pp. 9–50. Springer Verlag, 1998.

[6] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, New York, 1999.

[7] Y. LeCun et al, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] C. Merkwirth, M. Ogorzałek, and J.D. Wichard, "Stochastic gradient descent training of ensembles of DT-CNN classifiers for digit recognition," in *Proc. of the ECCTD*. Kraków, Poland, 2003, pp. 337–341.

[9] C. C. Chang and C.J. Lin, "Libsvm - A library for support vector machines," 2001. `www.csie.ntu.edu.tw/~cjlin/libsvm`

[2] The ZIP Code data set can be obtained from `www-stat-class.stanford.edu/~tibs/ElemStatLearn`