Finite Iteration DT-CNN with Stationary Templates

*J. Wichard**, M. Ogorzałek**, C. Merkwirth***, J. Bröcker****
* AGH University of Science and Technology, Kraków, Poland
** Max-Planck-Institut für Informatik, Saarbrücken, Germany
*** London School of Economics, London, UK

ABSTRACT: In this study we introduce a training algorithm for the design of CNN templates for the recently proposed DT-CNN in finite iterate mode that fulfill the standard requirement of stationarity. We present simulation results concerning the classification performance of such a classifier on a data set of hand-written digits for different numbers of iterations.

1 Introduction

The Cellular Neural Networks introduced by Chua and Yang [1] 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. A discrete time version of the CNN principle has been introduced (DT-CNN) [2]. Recently we have proposed a new mode of operation of DT-CNN — namely the finite iteration DT-CNN [3]. In the proposed approach the network was used as a one-step-forward computing engine with templates changing from step to step. Efficient procedures were proposed for template design and optimum selection of number of time steps. In this paper we introduce the design of a finite iterate network but with time invariant (stationary) templates. This is achieved by applying a penalty function to the template coefficients in the design optimization procedure. We show that by careful design one can achieve excellent results comparable with the time variant version in the handwritten digit recognition problem.

2 The CNN Topology

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

$$x_{ij}^{t+1} = \sum_{l,m=-\frac{K-1}{2}}^{\frac{K-1}{2}} (A_{l,m} y_{i+l,j+m}^t + B_{l,m} u_{i+l,j+m}) + b \quad (1)$$

$$y_{ij}^{t+1} = \sigma(x_{ij}^{t+1}), \quad (2)$$

wherein the u_{ij} denote the input pattern and $\sigma(x_{ij})$ is the activation function, further discussed in subsection 2.1. In the case of stationary CNN, we use the same templates (A, B and offsets b) in each iteration. In this case, the number of adjustable parameters depends only on the range of the local connectivity. If K is the number of rows of the quadratic template matrix we have K^2 adjustable weights.

2.1 The activation function

The activation function $\sigma(x)$ is a piecewise linear function that does not fully saturate for input values x far from the origin [3]. This feature ensures a still non vanishing 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.

2.2 The DT-CNN classifier

The general classification task is to decide for a given object, whether this object belongs to a specific class or not (binary decision). We use the DT-CNN as a classifier for Handwritten Digit Recognition. The task is to identify handwritten digits (from zero to nine) from a 16 by 16 pixel gray scale image. This is a classification problem with ten different classes. This multi-class 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. In the binary decision problem we present an input pattern u (the 256 values from the gray scale image) to the receptive field of the DT-CNN and we 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).

3 Training of DT-CNN Classifiers

The training of the DT-CNN classifiers is based on a stochastic gradient descend, which is a common method for training neural networks. In order to cope with the particular structure of the DT-CNN we have to introduce several improvements that make the training feasible.
In general the training consists of minimizing the average loss function (also called training error) with respect to the network templates. More specifically, let $(z^i, u^i), i = 1 \dots N$ be the training data consisting of input output pairs. We assume the templates (A_t, B_t, b_t) of layer t to be organized in the parameter vector $P_t = (P_t^1 \dots P_t^p)$ of length p. We denote the network output by $f(u^i, P)$ and the average loss function has the form

$$E(P, \epsilon) := \sum_{i=1}^{N} \lambda_\epsilon^1 (z_i - f(u^i, P)). \tag{3}$$

The loss function λ_ϵ^1 measures the deviation of the CNN output from the desired value z^i. In optimization usually a quadratic loss function is used, basically due to the simplicity

of the resulting derivatives. We also advocate the use of an ϵ–insensitive absolute loss function. The advantages of this strategy are described in section 3.1.
We further introduce a penalty term $D(P)$ that measures the deviation of the templates between the different layers

$$D(P,\epsilon) := \frac{1}{Tp}\sum_{t=1}^{T}\sum_{i=1}^{p}\lambda_\epsilon^2(P_{t-1}^p, P_t^p). \tag{4}$$

We decrease the ϵ–margin in the loss function λ_ϵ^2 iteratively during the training process. In the beginning the templates are initiated with different weights and they are allowed to differ inside the ϵ–margin without contributing to the loss. This strategy is necessary to scan the space of all possible templates extensively.
The resulting average loss function to be minimized is the convex sum

$$TE(D) := (1-\gamma)E(P,\epsilon_E) + \gamma D(P,\epsilon_D). \tag{5}$$

3.1 The ϵ–insensitive loss function

We use two types of ϵ–insensitive loss functions to calculate the training error. The ϵ–insensitive absolute loss function is defined as

$$\lambda_\epsilon^1(\xi) := \begin{cases} 0 & : \xi \le \epsilon \\ |\xi - \epsilon| & : \xi > \epsilon. \end{cases} \tag{6}$$

It is used to calculate the loss contribution of the miss-classified examples. 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 weights 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 appropriate trade-off between tolerating outliers and penalizing 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 [4]. The ϵ–insensitive squared loss function denotes as

$$\lambda_\epsilon^2(\xi) := \begin{cases} 0 & : \xi \le \epsilon \\ (\xi - \epsilon)^2 & : \xi > \epsilon. \end{cases} \tag{7}$$

We apply this to the deviation of the template weights in the different layers. During the training process the ϵ–margin is iteratively reduced to zero.

3.2 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

$$P_n = P_{n-1} - \delta P_n, \tag{8}$$

where $n = 1 \ldots N$, the number of training samples. The update δP_n depends on the nth training sample only and is calculated according to the minimization of the average loss function in equ. 5. 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.
For practical reasons we split this update in two parts and perform two gradient steps separately $\delta P_n = \delta P_n^1 + \delta P_n^2$. The first gradient step is related to the loss function in equ. 3 and is given by

$$\delta P_n^1 = \mu(1-\gamma)\frac{\partial}{\partial P}\lambda_{\epsilon_E}^1(z_n - f(u^n; P_{n-1}), \tag{9}$$

with $\epsilon_E = 0.9$. The initial step size is $\mu_{init} = 0.0008$ and it is multiplied after each training epoch with the factor $(\frac{\mu_{end}}{\mu_{init}})^{1/N}$, wherein $N = 1000$ is the total number of training epochs and $\mu_{end} = 0.0002$. The factor γ is also multiplied after each epoch with $(\frac{\gamma_{end}}{\gamma_{init}})^{1/N}$, wherein $\gamma_{init} = 0.9$ and $\gamma_{end} = 1.0$.
The second gradient step is related to the deviation in the templates and denotes

$$\delta P_n^2 = \frac{\gamma}{Tp}\frac{\partial}{\partial P}\sum_{t=1}^{T}\sum_{i=1}^{p}\lambda_{\epsilon_D}^2(P_{t-1}^p, P_t^p). \tag{10}$$

The ϵ–margin of the loss function is reduced iteratively during the training following the rule $\epsilon_D = 0.01 \cdot (1-\gamma) \cdot \sigma(P)$, where $\sigma(P)$ is the standard deviation of the all values in the parameter vectors $\{P_t\}_{t=0,\ldots,T}$

3.3 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.

3.4 Class frequency equalization

In the case of handwritten digit recognition the DT-CNN has to distinguish a certain digit from all others. Concerning the stochastic gradient descend this leads to a ratio between positive and negative examples of approximately 1:9 – or in other terms – each gradient step of a positive example is followed by nine gradient steps of negative examples. We can significantly improve the learning rate and the classification performance if we balance the ratio between positive and negative examples in the training data. Therefor we enrich the data set with duplicates of positive examples until the ratio is 1:1.

3.5 Building classifier ensembles

A common way to improve the performance of classifiers is ensemble building (see [5] and the references therein). This is done by training several (8-10) DT-CNNs on ran-

Digit	0	1	2	3	4
Rate [%]	95.7	99.4	96.2	96.9	96.5
Digit	5	6	7	8	9
Rate [%]	94.1	99.1	98.4	94.3	92.9

Table 1: Classification rates of final classifiers on the ZIP Code test set. Note that uniform random guessing yields a trivial classification rate of 10%. The average classification rate of the DT-CNN classifier is 96.3 %.

T	1	2	4	6	8	10	12
Rate [%]	79.9	93.5	95.4	95.6	96.3	96.1	95.9

Table 2: Classification rates of final classifiers on the ZIP Code test set versus number of template layers T.

domly chosen 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.

4 Digit Recognition

The ZIP Code Data Set[1] 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 gray-scale images (see [6]). There are 7291 training observations and 2007 test observations. 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.

4.1 Classification Performance

We trained an ensemble of 10 DT-CNN Classifiers with template size $K = 5$ and $T = 8$ layers. The stochastic gradient descent went over 1000 epochs. The classification rates for the resulting DT-CNN classifier are presented in Table 1. The classification rate of the DT-CNN classifier is 96.35 % which is comparable with the results of the non-stationary case [3]. For comparison, we trained a polynomial SVM classifier (see [7]) of degree 4 with $C = 100$ on the ZIP Code training set and applied it to the test set with an classification rate of 95.4%.
We further investigate the performance of the DT-CNN classifiers for different values of T. The results are shown in Table 2. One can observe that varying T from two to twelve layers, the classification rate increases with the number of layers and saturates

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

around 96.0% . We observed this behavior also in the case of non-stationary templates [3].

5 Conclusion

This article discussed the use of stationary DT-CNNs which operate in the finite iterate regime. In such a mode of operation no special requirements on template stability properties are needed. The templates are designed using a minimization approach based on stochastic gradient descend. 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. We have shown that it is possible to design templates for a specific purpose without using any kind of prior knowledge concerning the given task.

Acknowledgment

We would like to thank the AGH University of Science and Technology for the support. 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 and by the Deutsche Forschungsgemeinschaft (DFG) grant LE 491/11-1.

References

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

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

[3] C. Merkwirth, J. Bröcker, M. Ogorzałek, and J. Wichard, "Finite Iteration DT-CNN - New Design and Operation Principles," in *Proceedings of the ISCAS 2004 (in Press)*. Vancouver, Canada, 2004.

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

[5] C. Merkwirth, M. Ogorzałek, and J. Wichard, "Stochastic gradient descent training of ensembles of DT-CNN classifiers for digit recognition," in *Proceedings of the ECCTD*, vol. 2. Kraków, Poland, 2003, pp. 337–341.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: citeseer.nj.nec.com/lecun98gradientbased.html

[7] C. C. Chang and C. Lin, "Libsvm - A library for support vector machines (http://www.csie.ntu.edu.tw/~cjlin/libsvm)," 2001. [Online]. Available: `http://www.csie.ntu.edu.tw/~cjlin/libsvm`