2005 International Symposium on Nonlinear
Theory and its Applications (NOLTA2005)
Bruges, Belgium, October 18-21, 2005

Can DT-CNN Classifiers outperform SVM?

Christian Merkwirth† and Jochen Bröcker‡ and Jörg Wichard†† and Maciej Ogorzałek†

†Department for Information Technologies
Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University
Reymonta 4, 30-059, Kraków, Poland
‡Centre for the Analysis of Time Series, London School of Economics
Houghton Street, London WC2A 2AE, UK
††Molecular Modeling and Ligand Design, Forschungsinstitut für Molekulare Pharmakologie
Robert-Rössle-Str. 10, 13125 Berlin-Buch, Germany
Email: ChristianMerkwirth@web.de, J.Broecker@lse.ac.uk, JoergWichard@web.de, maciej@agh.edu.pl

Abstract—We show how to train Discrete Time Cellular Neural Networks (DT-CNN) successfully by backpropagation to perform pattern recognition on a data set of handwritten digits. By employing concepts and techniques from Statistical Learning, we can obtain results outperforming that of a polynomial Support Vector Machine (SVM).

1. Introduction

The local processing paradigm of the Cellular Neural Network mimics the processing of visual perception in biological systems and should therefore be well suited for pattern recognition tasks ([1]). We claim that by training several DT-CNNs with a stochastic gradient descent algorithm, we can reach better classification performance on the ZIP code test set than by using SVMs with polynomial kernel (see [2]).

The CNN *template weights* of the individual classifiers are determined by using a backpropagation scheme. We employ an online learning setting, which means that after one observation is processed, a tiny gradient step is performed. In order to reach a high generalization performance, we use several recent developments from the field of Machine Learning:

- ϵ-insensitive absolute loss instead of mean quadratic loss. This brings the backpropagation training of the CNN template weights closer to the optimization criterion used for constructing SVM classifiers (see [3]).
- Ensembling (classifier averaging). By averaging the output of several individually trained classifiers, one can improve the generalization performance of neural network classifiers ([4], [5]).

2. The DT-CNN classifier

Let y^t be a M–by–M spatial pattern, and y^t_{ij} denotes its entries for $i, j = 1 \ldots M$. The states of the DT-CNN cells y^t_{ij} evolve for iterations $t = 0, \ldots, T-1$ according to:

$$\begin{aligned} x^{t+1}_{ij} &= \textstyle\sum_{l,m=-\frac{K-1}{2}}^{\frac{K-1}{2}} (A^t_{l,m} y^t_{i+l,j+m} + B^t_{l,m} u_{i+l,j+m}) + b^t, \\ y^{t+1}_{ij} &= \sigma(x^{t+1}_{ij}), \\ y^0_{ij} &= 0 \end{aligned} \quad (1)$$

where the u_{ij} denote the input pattern and $\sigma(x_{ij})$ is the so–called *activation function*. The matrices (resp. numbers) A^t, B^t and b^t are usually referred to as *template weights*. If we use different template weights (A^t, B^t and offsets b^t) in each iteration t, the DT–CNN is called *non-stationary*. In this case, we get $(2K^2 + 1)$ scalar parameters per iteration step or *template layer*, where K is the number of rows of the quadratic template matrices. K also determines the degree of local connectivity. The total number p of adjustable scalar parameters can be calculated by $p = T(2K^2+1) - K^2$. A^0 is not used since $y^0_{ij} = 0$ by initialization. The number of template weights does not depend on the spatial size M of the CNN receptive field. Elements outside the boundary are treated as zero, which is identical to the value of background pixels.

The classification task is to decide if a given object belongs to a specified class or not (binary decision). To use a DT–CNN for classification (i.e. to decide whether the input pattern belongs to a certain prescribed class or not), we present an input pattern u (e.g. the 256 values from the gray scale image) to the receptive field of the DT-CNN. Then the output pattern has to be transformed into a *decision variable*. This can be done by, e.g. averaging over the entire output y^T_{ij}. By checking if the value of the decision variable z exceeds a certain threshold the decision is taken.

In this paper the specific task is to identify isolated handwritten digits (from zero to nine). This is a classification problem with *ten* different classes which can be converted into 10 binary classification problems [6]. All ten classifiers are trained on the same 7291 input patterns. The desired outputs presented to the first classifier are +1 for observations that have class label 0, otherwise −1. For the second classifier the desired outputs are +1 for observations with class label 1, and so on. To classify a new input

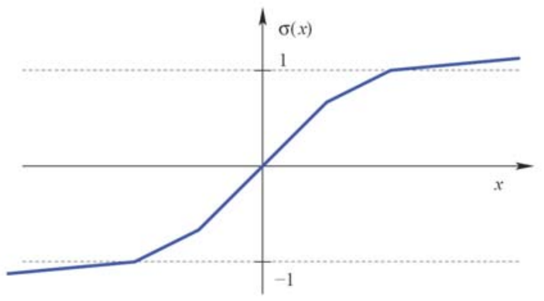

Figure 1: Piecewise linear function used as activation function. The function does not saturate at ±1, but the slope decreases to 0.01.

pattern, it is presented to all ten trained classifiers and the class label belonging to the classifier with highest output value z is chosen as final output label. The activation function $\sigma(x)$ is a piecewise linear sigmoid–like function (see FIg. 1). As this function does not saturate the training algorithm exploiting derivative information can escape solutions featuring very large template weights, which usually are not optimal [7].

3. Training of DT-CNN Classifiers

The main challenge faced when designing DT–CNN's for classification is to find template weights. We will use algorithms and principles from the theory of machine learning. The basic idea is to cast the problem of template design into an optimisation problem by employing a representative set of training data and a suitably chosen measure of performance or *loss*. The whole procedure is referred to as *training*.

Let $(u^k, z^k), k = 1 \dots N$ be the training data consisting of input output pairs. In the example of classifying images of handwritten digits, the $u^k, k = 1 \dots N$ are images of various different handwritten digits of known significance. Upon presentation of an input pattern u^k the DT–CNN yields an output pattern y^T. As mentioned in section 2 this output is averaged, resulting in a real number which can be considered as a function $f(u^k, P)$ of the input pattern u^k and the parameters (template weights) P. Now this value has to be compared to the class label z^k by means of a *loss function*. The loss function measures the deviation of the CNN output from the desired value z^k. In optimization 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 λ^1_ϵ. The advantages of this strategy as well as the functional form of λ^1_ϵ are described in section 3.1. The total loss or training error is simply the loss averaged over the entire training set.

$$E(P, \epsilon) := \sum_{k=1}^{N} \lambda^1_\epsilon(z_k - f(u^k, P)). \tag{2}$$

The training relies on minimization of the training error $E(P, \epsilon)$ with respect to P. Furthermore it depends on ϵ, the significance of which becomes clear in the next section.

3.1. The ϵ–insensitive loss function

The ϵ–insensitive *absolute* loss function (see Figure Figure 2) is defined as

$$\lambda^1_\epsilon(\xi) := \begin{cases} 0 & : \xi \leq \epsilon \\ |\xi - \epsilon| & : \xi > \epsilon. \end{cases} \tag{3}$$

It is used to calculate the loss contribution of the misclassified 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 focus on the misclassified training patterns rather than adjusting the weights by 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 [3].

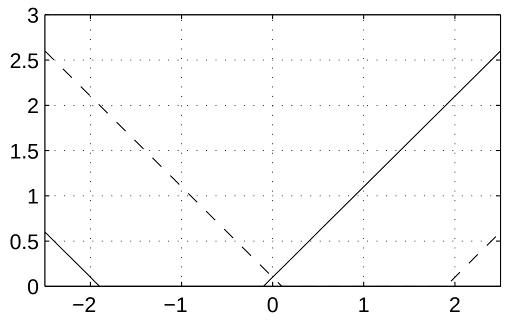

Figure 2: The ϵ-insensitive absolute loss function used for the training of the DT-CNNs.

3.2. Stochastic gradient descent

The method of training of DT-CNN classifier is based on a *stochastic gradient descend* or *online learning*, which is a common method for training neural networks [7]. In order to cope with the particular structure of the DT-CNN we have to introduce several amendments rendering the training feasible.

The gradient of the entire training error (Equ. 2) is a sum of terms of the form

$$\frac{\partial}{\partial P} \lambda^1_\epsilon(z^k - f(u^k; P). \tag{4}$$

The stochastic gradient descent performs a series of very small consecutive steps, determining each step direction from the gradient of an individual summand only. After each step, the new parameter set P is re–inserted into the loss before the next gradient is computed. In other words, we devise an update rule for the parameters of the form

$$P_k = P_{k-1} - \delta P_k, \tag{5}$$

where $k = 1 \dots N$, and N is the number of training samples. The update δP_k depends on the kth training sample only and is given by

$$\delta P_k = \mu \frac{\partial}{\partial P} \lambda^1_\epsilon (z^k - f(u^k; P_{k-1}). \tag{6}$$

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, here a tiny gradient step is carried out after each observation has been processed. A sweep through the whole data set is called *epoch*. The initial step size is $\mu_{\text{init}} = 0.0007$ and it is multiplied after each training epoch with the factor $(\mu_{\text{end}}/\mu_{\text{init}})^{1/N}$, wherein N is the total number of training epochs and $\mu_{\text{end}} = 0.0002$.

3.3. Backpropagation

Backpropagation is an elegant method for efficiently calculating the gradient of an iterated function with respect to its parameters. In the *forward pass* the output of the iterated function is computed accorded to equations 1 and intermediate states x^t_{ij} and y^t_{ij} have to be stored for the *backward pass*. The backward pass traverses the system in reverse order, starting from the last iteration of the forward pass. By applying the chain rule of differentiation, it is possible to obtain the recursion equation 9 for the backward pass that closely resembles the forward equations, though the application of the activation function converts into a multiplication with the derivative of the activation function.

For simplicity, we introduce an auxiliary variable that represents the gradient of the loss function with respect to the CNN states y^t_{ij} :

$$s^t_{ij} := \frac{\partial \lambda^1_\epsilon}{\partial y^t_{ij}} \tag{7}$$

Insertion of initial gradient :

$$s^T_{ij} = -\frac{1}{R^2} \lambda^1_\epsilon (z - f(u, P)) \tag{8}$$

Iteration in reverse order :

$$s^{t-1}_{ij} = \sum_{l,m=-\frac{K-1}{2}}^{\frac{K-1}{2}} A^{t-1}_{lm} s^t_{i-l,j-m} \sigma'(x^t_{i-l,j-m}) \tag{9}$$

From the s^t_{ij} all derivatives with respect to the parameters can be obtained :

$$\frac{\partial E}{\partial A^{t-1}_{lm}} = \sum_{i,j} s^t_{ij} \sigma'(x^t_{ij}) y^{t-1}_{i+l,j+m} \tag{10}$$

$$\frac{\partial E}{\partial B^{t-1}_{lm}} = \sum_{i,j} s^t_{ij} \sigma'(x^t_{ij}) u^{t-1}_{i+l,j+m} \tag{11}$$

$$\frac{\partial E}{\partial b^{t-1}} = \sum_{i,j} s^t_{ij} \sigma'(x^t_{ij}) \tag{12}$$

As can be seen from equations 10-12, only state values y^t_{ij} and derivatives of the activation function $\sigma'(x^t_{ij})$ have to be stored. This results in a space complexity of $O(TM^2)$.

3.4. Classifier Ensembling

A way to improve the performance of classifiers is building classifier ensembles (see [8] and the references therein). We employed a simple *classifier averaging strategy* in which we combine 8 separately trained DT-CNN to an ensemble classifiers. Please note that we have to construct one such ensemble for each of the ten binary classification tasks.

4. Results

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. 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 training data set with duplicates of positive examples until the ratio is 1:1.

For training and testing we used the ZIP Code Data Set[1] which consists of normalized handwritten digits, scanned from envelopes by the U.S. Postal Service. The images have been deslanted and size normalized, resulting in 16 by 16 grayscale images (see [9]). There are 7291 training observations and 2007 test observations, distributed as follows: The patterns are presented to the training algorithm

Digit	0	1	2	3	4
Train	1194	1005	731	658	652
Test	359	264	198	166	200

Digit	5	6	7	8	9
Train	556	664	645	542	644
Test	160	170	147	166	177

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

as input/output pairs with inputs being 16 by 16 matrices containing gray levels coded from 0.0 (background pixels) to 1.0 (foreground pixels). Outputs are class labels out of 0,1,...,9.

Computational demand did not allow to perform a systematic check of all possible combinations of topological

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

parameters K and T. Therefore the template size K employed during the simulations (see table 2) was fixed to 5. With standard nearest-neighbor coupling $K = 3$ we could not achieve an adequate classification performance, for template size $K = 7$ we could observe a beginning gap between training and test errors. From table 2 one can observe that with T from 1 to 10 layers, classification rate increases significantly. At higher numbers of template layers the rate saturates around 96.5%. For comparison, we constructed a polynomial SVM classifier of degree 4 with $C = 100$ (see [2]) on the ZIP Code training set and applied it to the test set.

T	1	2	3	4
Class.rate	79.7%	91.6%	94.9%	95.9%
T	5	6	8	10
Class.rate	96.0%	96.1 %	95.6%	96.5%
T	14	18	24	SVM
Test	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 would yield a trivial classification rate of 10%.

5. Conclusions

This article discusses a backpropagation type algorithm for training CNN classifiers for digit recognition. The results on a database of handwritten digits outperform a polynomial Support Vector classifier. An important aspect here is the usage of an alternative loss function during the training process instead of quadratic error. A main drawback of the method is the higher computational effort for training ensembles of DT-CNN classifiers than constructing SVM classifiers on the same task.

6. Outlook

Recent advances in the treatment of non-vectorial data such as the molecular graph within the Support Vector Machine framework [10] and within the DT-CNN framework [11] allow the construction of classifiers that directly use the molecular structure as input data. We plan to compare both approaches with respect to the classification performance achievable on real-world data sets.

Acknowledgments

The authors would like to thank the inventor of the libSVM, Chih-Jen Lin for stimulating comments and discussions. 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.

References

[1] T. T. Roska and L.O. Chua. The cnn universal machine: an analogic array computer. *IEEE Trans. Circuits and Syst.*, 40(II):163–173, 1993.

[2] C. C. Chang and C.J. Lin. Libsvm - A library for support vector machines (http://www.csie.ntu.edu.tw/˜cjlin/libsvm), 2001.

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

[4] M. P. Perrone and L. N. Cooper. When Networks Disagree: Ensemble Methods for Hybrid Neural Networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall, 1993.

[5] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

[6] C. Hsu, C. Lin, “A comparison of methods for multi-class support vector machines”, Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, 2001

[7] Y. LeCun, L. Bottou, G. Orr, and K.R Müller. Efficient BackProp. In G. Orr and K.R Müller, editors, *Neural Networks: Tricks of the trade*, volume 1524 of *Lecture Notes in Computer Science*, pages 9–50. Springer Verlag, 1998.

[8] C. Merkwirth, M. Ogorzałek, and J.D. Wichard. Stochastic gradient descent training of ensembles of DT-CNN classifiers for digit recognition. In *Proceedings of the European Conference on Circuit Theory and Design ECCTD'03*, volume 2, pages 337–341, Kraków, Poland, 2003. European Circuit Society.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] P. Mahé, N. Ueda, T. Akutsu, J.L. Perret, and J.P. Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *J. Chem. Inf. Model.*, 45(4):939 – 951, 2005.

[11] C. Merkwirth and T. Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *J. Chem. Inf. Model.*, ASAP Web publication, DOI: 10.1021/ci049613b, 2005.