Translating Feedforward Neural Nets to SOM-like Maps

Berend Jan van der Zwaag[1], Lambert Spaanenburg[2], and Kees Slump[1]
[1] Systems and Signals, Dept. of Electrical Engineering, University of Twente
P.O.Box 217, 7500 AE Enschede, the Netherlands
E-mail: b.j.vanderzwaag@utwente.nl
[2] Dept. of Information Technology, Lund University
P.O.Box 118, 22100 Lund, Sweden

Abstract— **A major disadvantage of feedforward neural networks is still the difficulty to gain insight into their internal functionality. This is much less the case for, e.g., nets that are trained unsupervised, such as Kohonen's self-organizing feature maps (SOMs). These offer a direct view into the stored knowledge, as their internal knowledge is stored in the same format as the input data that was used for training or is used for evaluation. This paper discusses a mathematical transformation of a feed-forward network into a SOM-like structure such that its internal knowledge can be visually interpreted. This is particularly applicable to networks trained in the general classification problem domain.**

Keywords— **Neural networks, rule extraction, self-organizing maps, feature maps, character recognition.**

I. Introduction

Since the early development of artificial neural networks, but especially in the past 10 years, researchers have tried to analyze them to provide insight into their behavior. For certain applications and in certain problem domains this has been successful. In particular in decision making systems and other systems that can easily be expressed in sets of rules, great advances have been made by the development of so-called rule extraction methods [1]. Neural network systems with relatively few inputs can sometimes be analyzed by means of a sensitivity analysis [2], which is a nonparametric statistical analysis technique.

However, most neural network systems are so high-dimensional that an extracted rule base would become too large to be easily interpreted, or so nonlinear that a sensitivity analysis would only be valid for a small part of the input space. For this reason, we propose domain-specific neural network analysis methods that utilize domain-specific base functions [6] that are easy to interpret by the user and that can even be used to optimize neural network systems. An analysis in terms of base functions may also make clear how to (re)construct a superior system using those base functions, thus using the neural network as a construction advisor.

In this paper we will describe the analysis of feedforward neural networks in the application field of classification problems, by translating the trained network into a feature map such as used in self-organizing feature maps (SOMs). The structural composition by a set of basic functions that have a known translation into the functionality of a SOM, allows transforming the overall structure into a feature map. The gained benefit is the transparency of the stored knowledge from the resulting SOM.

II. Analysis of Neural Networks

In general, artificial neural networks with unsupervised training merely reorganize the input space, so analyzing them after training becomes fairly simple: an investigation into the reorganized input space reveals how the network has restructured the input space.

Analyzing neural networks trained under supervision is far more complicated, for input and output spaces are usually in different domains (e.g., a character recognition system has an image as input, and a character class as output), whereas in the unsupervised case, input and output spaces are basically the same, although they are organized in different ways.

The idea of describing a trained neural network in terms of basic domain-specific functions was introduced and presented in earlier publications [6], [7]. For many problems in certain domains, such as linguistics and decision theory, the common, domain-dependent base functions could be chosen to be *if–then* rules or decision trees, in which case the analysis reduces to rule extraction. Table I lists a few more problem domains where neural networks have been successfully applied. For each of these domains, candidate base functions are presented.

For one such domain, i.e., digital image processing, we have shown [5] how our analytical method can overcome the impracticality of more common knowledge extraction methods. The neural network edge detector was described as a set of gradient filter components, giving easy insight into the behavior of the neural network as an edge detector, and allowing simple comparison with other edge detectors which can in the same way be described in terms of gradient filter components.

TABLE I
Some application domains with potential base functions.

application domain	potential base functions
signal processing (1-D)	basic operational filters
digital image processing (2-D)	differential operators
general classification problems	feature map prototypes (compare Kohonen's self-organizing feature map)
decision theory	if-then rules (fuzzy or not) (i.e., rule extraction as a special case of the proposed method)
control theory	basic control operators

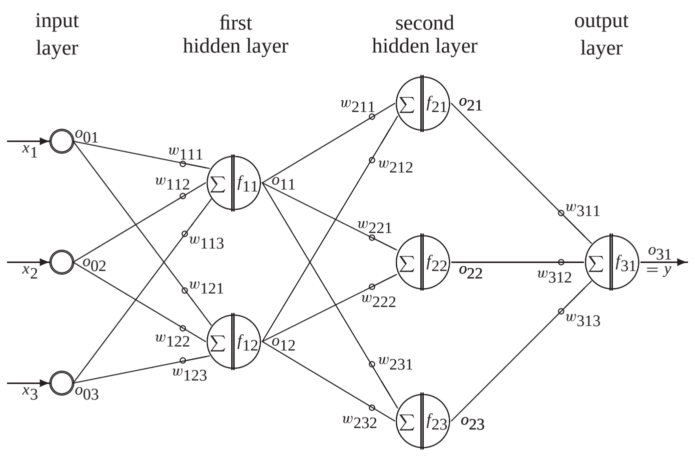

Fig. 1
EXAMPLE OF A FULLY LAYER-TO-LAYER-CONNECTED FEEDFORWARD NEURAL NETWORK.

In general, the analysis consists in describing the internal functionality of the neural network in terms of basic domain functions, functions that can be considered basic in the application domain of the neural network. This means that users who may not be familiar with artificial neural networks, but who are familiar with basic functions that are often used in their problem domain, can gain insight in the way the neural network solves their problem. For such users, this is often an important factor in deciding to apply artificial neural networks to a problem that may be difficult to solve otherwise.

We have shown that, for the 2-D image processing domain, it is feasible to translate trained feed-forward neural networks into sets of differential operators of various orders and with various angles of operation [8]. Such operators can in turn be readily identified in a SOM. This significantly improves the comprehension of the knowledge that is stored in the networks.

In the general domain of classification problems, the prototyping of the classes in the input space can be regarded as a suitable basic function. Such a prototyping is characteristic for self-organizing maps, where prototypes are mapped onto a topology-preserving feature map. So, if one would translate a feed-forward network into a SOM-like network, a comprehensible description of the neural network's functionality would be accomplished. Though the transformation is not straightforward, it holds the promise to be computable in limited time, where the alternative of straight rule extraction falls short.

III. FEEDFORWARD NEURAL NETWORKS

Feedforward–error-back-propagation neural networks [3] are the most commonly used neural networks that are trained with supervised learning methods. They consist of several layers of neurons, as illustrated in Figure 1. Actually, the first layer, or input layer, does not contain any neurons, but merely distributes the input signals to the first layer that does consist of neurons, the first hidden layer. The output signals from the first hidden layer are then forwarded to the next layer, and so on. The final hidden layer's output signals are passed on to the output layer, which, after processing its input signals, produces the output of the network.

The feedforward neural network shown in Figure 1 has three inputs, two hidden layers with two and three neurons respectively, and one output. The threshold unit and its connections to all neurons are not shown in the figure. For clarity, the superscript P in the signal names representing the pattern offered to the network is not shown either. In general, feedforward neural networks consist of the following layers:

the input layer, layer 0, consisting of N_0 inputs x_i^P, with $i = 1, \ldots, N_0$ and P being the pattern that is currently offered to the network. The input signals are then distributed as $o_0 i^P$ from the input layer to the next layer,

the first hidden layer, layer 1, consisting of N_1 hidden neurons. The outputs o_{0i}^P, $i = 1, \ldots, N_1$, of these neurons are connected to the inputs of the next layer,

...

the ***k*****th hidden layer,** layer k, consisting of N_k hidden neurons, $k = 1, \ldots, K - 1$, where K is the number of neuron layers. In a fully layer-to-layer-connected network, each neuron in this layer has N_{k-1} inputs, $k = 1, \ldots, K-1$. The input signals $o_{k-1,j}^P$, $j = 1, \ldots, N_{k-1}$ to this layer are distributed along the connections from the previous layer; input signal $o_{k-1,j}^P$, $j = 1, \ldots, N_{k-1}$

traveling along the connection between unit j in the previous layer and unit i in the present layer is multiplied by the connection weight w_{kij}, $k = 1, \ldots, K-1$, $i = 1, \ldots, N_k$, $j = 1, \ldots, N_{k-1}$. The weighted sum of input to neuron i is then processed by the neuron's activation function f_{ki} and the resulting output signal o_{ki}^P of this neuron, along with the output signals from the other neurons in this layer, is forwarded to the next layer,

...

the last hidden layer, layer $K-1$, consisting of N_{K-1} hidden neurons. The output signals $o_{K-1,i}^P$, $i = 1, \ldots, N_{K-1}$, from this layer are forwarded to the final layer,

the output layer, layer K, consisting of N_K output neurons. The output signals o_{Ki}^P, $i = 1, \ldots, N_K$, from this layer are also the outputs y_i^P, of the whole network, when pattern P is offered to the network.

Every layer k, except the output layer, furthermore contains a threshold unit, which has a constant output $o_{k0}^P = -1$, $k = 0, \ldots, K-1$, that is connected via connection weights $w_{k+1,i,0}$, $k = 0, \ldots, K-1$, to neurons i in the subsequent layer $k+1$.

Thus, a hidden neuron i in layer k has a total of $n_{k-1}(+1)$ inputs $o_{k-1,j}^P$, a total of $n_k + 1$ weights w_{kij}, and one output o_{ki}^P. An output neuron i in layer K has a total of $n_{K-1}(+1)$ inputs $o_{K-1,j}^P$, a total of $n_K + 1$ weights w_{Kij}, and one output $o_{Ki}^P = y_i^P$, which is also one of the outputs of the network as a whole.

The output of a neuron i in layer k is calculated with an activation function f_{ki}. Often, the same activation is used for all neurons in the entire network, or at least in all neurons within the same layer, but this is not a requirement. Its shape usually varies from threshold functions (Figure 2) and piecewise linear functions (Figure 3) to smooth nonlinear functions such as the logarithmic sigmoid function shown in Figure 4, which is most frequently used as activation function.

IV. A Classification Task

In order to illustrate our analysis method applied to a neural network classifier, we have trained a feedforward neural network for the classification of a set of characters from the Latin alphabet. The set that we used is available from the Matlab[1] Neural Network Toolbox. It is shown in Figure 5.

The feedforward back-propagation network that we used had 35 inputs, fully connected to one hidden layer containing 10 hidden neurons, which were again fully connected to the output layer holding 26 output neurons. The

[1] ©The Mathworks, Inc.

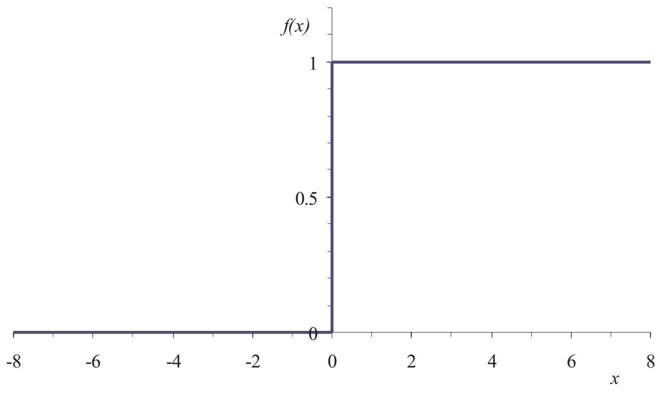

Fig. 2
A threshold activation function.

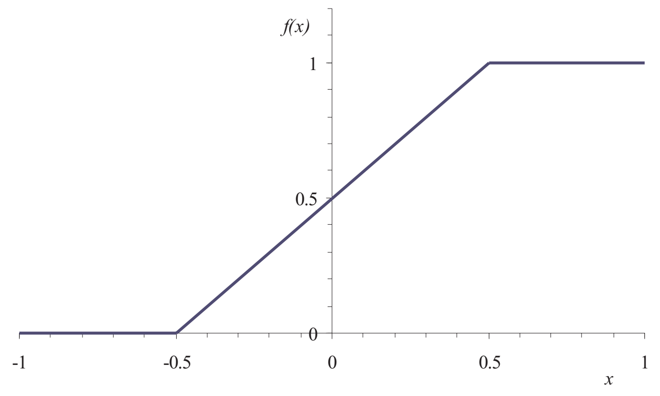

Fig. 3
A piecewise linear activation function.

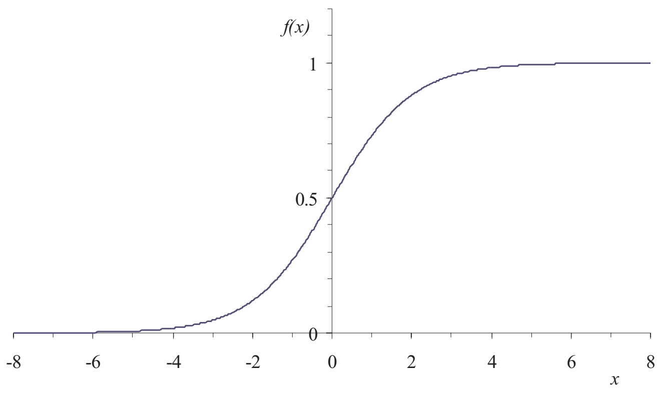

Fig. 4
A sigmoid activation function, $f(x) = \frac{1}{1+e^{-x}}$.

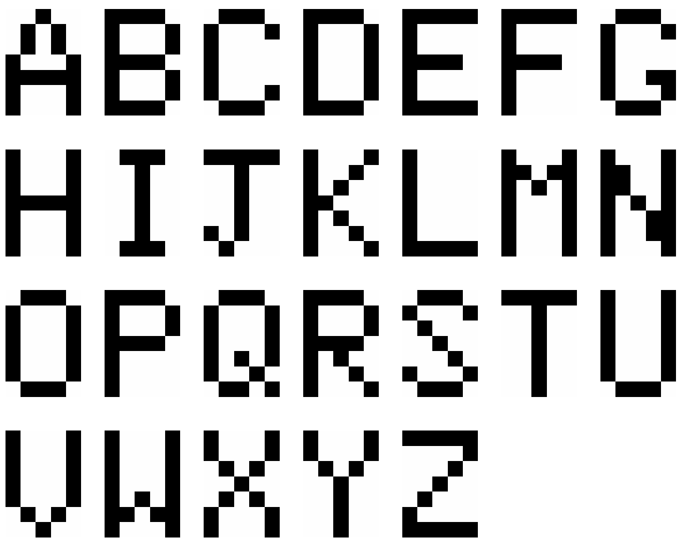

Fig. 5
A SET OF TWENTY-SIX 5×7 CHARACTERS.

neurons in the hidden and output layers all used the logarithmic sigmoid shown in Figure 4 as their activation function. During training, clean as well as noisy images were offered to the network inputs, and for each character image, the target vector for the output layer contained a one for the output neuron corresponding to the position of the character in the alphabet and zeros otherwise. After training, all training patterns were correctly classified by the network.

V. DETERMINING PROTOTYPES

One of the graphical representation methods occasionally seen in literature is basically a graphical representation of the synaptic weights of the network, in particular those between the input and hidden layers. This is usually based on the idea that character features are stored in the network's synaptic weights and that these features can be identified by visual inspection of the weights, ordered in the same way as the input vectors. We can also do this for our network, resulting in the "feature map" presented in Figure 6. This figure shows the synaptic weights in matrix form for all ten hidden neurons, as well as the synaptic weights between the hidden and output layers, organized per hidden unit.

Unfortunately, the visualization of the synaptic weights in the way shown in Figure 6 does not give insight into the knowledge stored in the network. Therefore, we propose to determine feature vectors, or prototypes, from the neural network. In this sense, we analyze the neural network in terms of basic domain-dependent functions, which in this case we choose to be class prototypes.

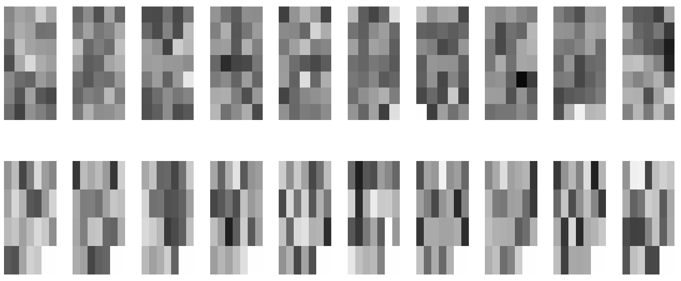

Fig. 6
GRAPHICAL REPRESENTATION OF THE SYNAPTIC WEIGHTS INTO (*top*) AND FROM (*bottom*) THE HIDDEN NODES.

Whereas determining prototypes from a Kohonen self-organizing map is easily achieved by directly reading the feature vectors stored in the neurons, this is not straightforward when dealing with feedforward networks. We will now describe the method we used to determine prototypes in two main steps.

A. Determining Internal Prototypes

Suppose we want to determine a prototype for class C, being the class for the letter "C". We know that the ideal output of the network is then

$$\mathbf{y} = (0, 0, 1, 0, 0, 0, \ldots, 0, 0, 0, 0). \tag{1}$$

From Section III, we also know that the network output is given by

$$\mathbf{y} = f(\mathbf{W}_2\mathbf{o}_1) \quad \text{or} \quad y_i = f(\sum_j w_{2ij} o_{1j}), \tag{2}$$

with activation function

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}}. \tag{3}$$

In order to ensure ideal classification of the prototype for the third class, C, we must maximize $f(\sum_j w_{2ij} o_{1j})$ for $i = 3$ and minimize the same for $i \neq 3$. With (3), this would mean that we are looking for $\sum_j w_{2ij} o_{1j} \to \infty$ and $\sum_j w_{2ij} o_{1j} \to -\infty$, respectively. This is not realizable, as the o_{1j} are bounded between 0 and 1.

However, what we can do is the following. We can define an internal prototype $\mathbf{p}_C^{\mathrm{h}}$ for class C by defining $p_{Ci}^{\mathrm{h}} = 1$ if $w_{2Ci} > 0$ and $p_{Ci}^{\mathrm{h}} = 0$ otherwise. This hidden layer output prototype will ensure maximum output for output class C and hopefully minimum output for all other classes.

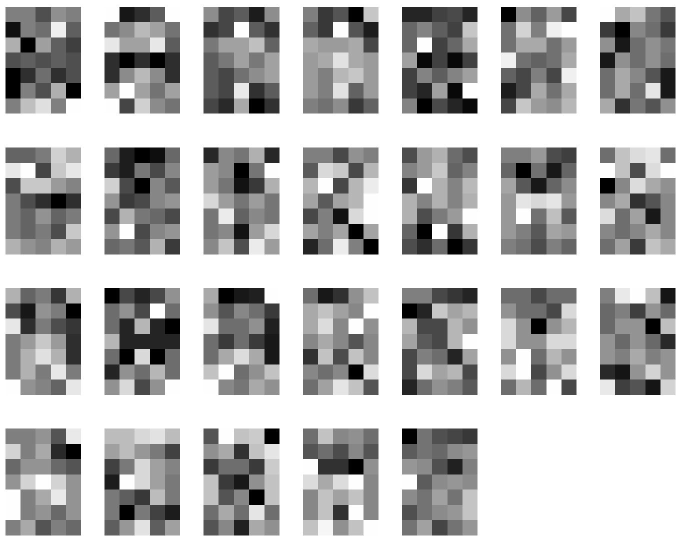

Fig. 7
PROTOTYPES OF ALL 26 CLASSES.

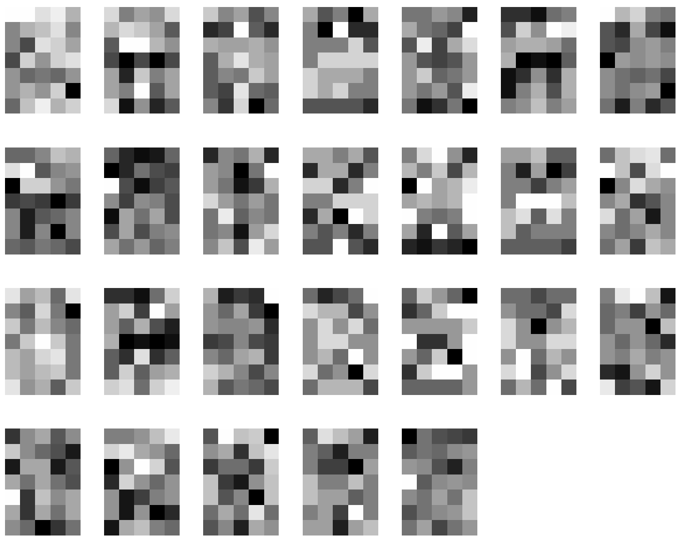

Fig. 8
IMPROVED PROTOTYPES OF ALL 26 CLASSES.

B. Determining Class Prototypes

The next step is to name the internal prototype $\mathbf{p}_C^{\mathrm{h}}$ the ideal hidden layer output for class C. Preliminary experiments have resulted in the following recipe for determining the final prototype $\mathbf{p}_C$ for class C:

$$\mathbf{p}_C = (K_1 - \Sigma \mathbf{p}_C^{\mathrm{h}})\mathbf{p}_C^{\mathrm{h}}\mathbf{B}_{\mathbf{W}_1} - \Sigma \mathbf{p}_C^{\mathrm{h}}(1 - \mathbf{p}_C^{\mathrm{h}})\mathbf{B}_{\mathbf{W}_1}, \quad (4)$$

where K_1 is the number of hidden neurons and $\mathbf{B}_{\mathbf{W}_1}$ is a Boolean variable with value 1 for those j for which $w_{1ij} <$ 0 and with value 0 for all other j.

VI. RESULTS

Applied to all classes of our feedforward network, we get the resulting prototypes shown in Figure 7. Even though the similarities with the original training patterns are weak (yet still visible, e.g., at classes “K”, “O”, “U” or “X”), the network classifies 24 out of the 26 prototypes correctly (most of them with confidence over 90%) when offered to the network. The misclassifications occur with the derived prototype of a “B”, which is misclassified as a “Q” and the derived prototype of a “W”, which is misclassified as a “G”.

The inaccuracy of the derived prototypes is partially caused by the noninvertability of the network function due to the bounds on the neuron outputs. For a small network as the one in our experiments, a more accurate internal prototyping can be achieved by calculating the network output over the whole range of possible hidden layer outputs, only allowing the hidden layer outputs to be either 1 or 0. For a network with 10 hidden neurons, this means an evaluation of $2^10 = 1024$ cases, which is still feasible. The internal prototypes can then be determined by choosing the hidden layer outputs with the highest distinction in the network output, i.e., for which the difference between the output value for the target class and the highest output value for any other class is maximal.

The new internal prototypes can then again be used for the derivation of the final prototypes, using (4). The graphical result (Figure 8) is similar to that of Figure 7, but the network performance when the new prototypes are offered to the neural network has improved. All 26 prototypes are now correctly classified by the network. It is expected that more improvements in our prototype method are possible. This will require more research.

VII. CONCLUSIONS AND RECOMMENDATIONS

We have trained neural networks to classify characters and analyzed them in terms of class prototypes. From the results displayed and described in the previous sections it is clear that it is indeed feasible to describe the trained neural networks in terms of basic functions from the classification domain, although improvements can still be expected.

The description with class prototypes gives an impression of the class knowledge stored in the synaptic weights of the neural network as a classifier, even though the prototypes seem not very likely as input patterns for this particular classification problem.

In general, the analysis method that we have developed consists in describing the internal functionality of the neural network in terms of basic domain functions, functions that can be considered basic in the application domain of the neural network. This means that users who may not be familiar with artificial neural networks, but who are familiar with basic functions that are often used in their problem domain, can gain insight in the way the neural network solves their problem. For such users, this is often an important factor in deciding to apply artificial neural networks to a problem that may be difficult to solve otherwise.

To improve the algorithms for the extraction of prototypes from trained feedforward neural network classifiers, and to enhance the interpretation of the derived prototypes, we recommend a deeper investigation. Other explorations could also be recommended, such as the investigating the possibility of translating self-organizing maps into feedforward networks, for example for hardware considerations.

REFERENCES

[1] M.W. Craven and J.W. Shavlik (1994), "Using sampling and queries to extract rules from trained neural networks," in: *Machine Learning: Proceedings of the Eleventh International Conference*, San Francisco, CA.

[2] S. Hashem (1992), "Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions," in: *Proceedings of the 1992 International Joint Conference on Neural Networks*, IEEE Press, Piscataway, NJ, USA, vol. 1, pp. 419-424.

[3] D.E. Rumelhart, J.L. McClelland, et al. (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol 1: *Foundations*, Cambridge, MA: MIT Press.

[4] L. Spaanenburg, R. Alberts, C.H. Slump, and B.J. vanderZwaag (2003), "Natural learning of neural networks by reconfiguration," in: A. Rodriguez-Vazquez, D. Abbott, and R. Carmona (eds.), *Bioengineered and Bioinspired Systems* (Gran Canaria, Spain, 19-21 May), *Proceedings of SPIE* vol. 5119, pp. 273-284.

[5] B.J. van der Zwaag and C. Slump (2002), "Analysis of neural networks for edge detection," in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 28-29 Nov.), pp. 580-586.

[6] B.J. van der Zwaag, L. Spaanenburg, and C. Slump (2002), "Analysis of neural networks in terms of domain functions," in: *Proceedings IEEE Benelux Signal Processing Symposium SPS-2002* (Leuven, Belgium, 21-22 March), pp. 237-240.

[7] B.J. van der Zwaag, C. Slump, and L. Spaanenburg (2002), "Process identification through modular neural networks and rule extraction," in: D. Ruan, P. D'hondt, and E.E. Kerre (eds.), *Computational Intelligent Systems for Applied Research: Proceedings of the 5th International FLINS Conference* (Ghent, Belgium, 16-18 Sep.), World Scientific, Singapore, pp. 268-277.

[8] B.J. van der Zwaag, K. Slump, and L. Spaanenburg (2003), "Extracting knowledge from supervised neural networks in image processing," Chapter 5 in R. Jain, A. Abraham, C. Faucher, and B.J. van der Zwaag (eds.) (2003), *Innovations in Knowledge Engineering*, Adelaide, South Australia: Advanced Knowledge International, ISBN 0-9751004-0-8.