ISTANBUL UNIVERSITY –
JOURNAL OF ELECTRICAL & ELECTRONICS ENGINEERING
YEAR : 2003 (719-726)
VOLUME : 3
NUMBER : 1

MODULAR AND HIERARCHICAL SPECIALIZATION IN NEURAL NETWORKS

A.J.W.M. ten BERG[1,2] and L. SPAANENBURG[1]

[1] Lund University, Department of Information Technology, P.O. Box 118, S-22100 Lund, Sweden.
Email: lspaanenburg@ieee.org , Tel: +46-46-22 24931, Fax: +46-46-22 24714

[2] currently with Philips Research Laboratories, Eindhoven, The Netherlands

ABSTRACT

Modularity and hierarchy are fundamental notions in structured system design. By subdividing a large and unstructured problem into smaller and tractable chunks, design automation becomes possible. In this paper we discuss the use of modularity and hierarchy for functional specialization during the development of neural networks. We study the behavioral differences and requirements for back-propagation training of feed-forward networks. Further we illustrate that a deliberate mix of hierarchically imposed evaluation functions will improve network accuracy and learning speed.

Keywords: *Neural Networks, Modularity, Hierarchy, Structure Transformation, Function Specialization*

I. INTRODUCTION

Compositional techniques are at the fundament of any complex design task. Where complex neural networks are plagued by learning problems, composition styles can help to increase the capacity and accuracy of neural networks **0**, **0**. Though providing better performance, the fundamental difficulty in network training **0** remains a major concern and several studies have been made for solutions to this problem **0**, **0**. Typically, literature defines three categories of compositions of a neural network **[1]**; ensembles, modules and hierarchies.

Ensembles consist of a set of autonomous networks in which each network solves the complete problem. *Data engineering* typically derives the individual networks. By applying different subsets of the training set to the different networks in the ensemble, they generalize in different ways. A voting mechanism then reaches a final, overall decision: a mechanism well known in other engineering disciplines as well. This improves accuracy and robustness.

Received Date : 25.09.2002
Accepted Date: 25.12.2002

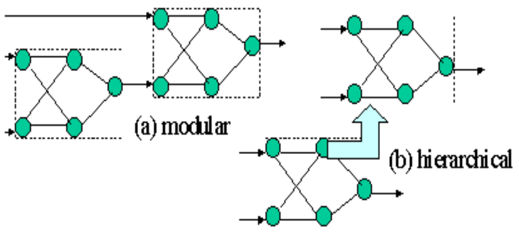

Figure 1. Some neural network types.

Modular networks typically consist of a network of interconnected neural networks **0** or modules that each solve a sub-problem, see 0(a). For this approach extensions to the standard back-propagation learning algorithm are necessary. For example techniques to schedule the learning of the modules to prevent unlearning, as proposed in **0**.

Hierarchical networks go one step further in the compositional sense. Each node in a neural network may be again a neural network, or a specialized function, as shown in 0(b). So a node's evaluation function is implemented by a neural network, while preserving the weights on its inputs from the upper layer.
Both the modular and hierarchical approach focus on the structure of the overall network, this in contrast to the ensemble approach. Where data engineering suffices to support ensemble engineering, for modular and hierarchical networks also *structure engineering* should be defined. Both data-engineering and structure engineering are elements of a more generic *knowledge engineering discipline*.

For the introduction of the problem we look at the function |sin(x)| * exp(-x) as appearing in an experiment in the non-destructive test of stitch welds **[2]**. There are some clear discontinuities in this function, that make it hard to learn on a network using a sigmoid transfer. It is widely claimed that neural networks need a differentiable _ and therefore continuous _ transfer. A similar observation seems valid for the function to be approximated. Nevertheless functional modularity proves to be able to approximate the target. Therefore the question arises: is there a structure transformation that links a learnable network to one that is more efficiently executable?

2.STRUCTURE TRANSFORMATION

Structure transformations are defined from two basic operations on nodes:

- The composition of several nodes into a single node
- The decomposition of a single node into a (sub-) network of nodes.

Structure transformations require that the behavior of the structure before and after the transformation is identical: *function preservation*.

In general, structure transformations are possible for function networks. This implies the definition of composition and decomposition rules for the nodes in such networks. So, functional networks are compositional, of course under certain conditions.

Neural networks, as they are, do not support compositionality. The question is now to formulate the conditions that enable compositionality for the general case of a modular neural network. If such conditions cannot be found, then the modular network needs to be considered as a function network, in which each function is to be considered as a specification of a neural module. Since a function is a specification of a module, no formal manipulation of the structure of the modular neural network will exist. Here, the trained modular network is considered as an implementation of the module's function.

The main consequence of this is that the neural modules need to be re-trained from scratch after each manipulation. This is feasible, but induces much training overhead. So, composition rules or conditions are essential for neural networks to prevent (or minimize) re-training components in a modular network.

In this paper, we study this problem by considering only feed-forward neural networks with one hidden layer and a sigmoid (or its piece-wise linear functional equivalent) evaluation function for its neurons. Both for composition as for decomposition we can distinguish between a parallel and a cascade (de-) composition.

2.1 Parallel (de-) composition

The transformation whereby a network is split into two or more different networks with the same inputs, while overall covering the original function is called decomposition.. The reverse is called a composition. Presume that both composing parts are implemented with neural networks, is this composition then feasible? Yes, since the networks can be merged in a very simple way as shown in 0. In this case all the layers are simply merged, and duplicate input nodes can be merged into single nodes.

The same holds also for the decomposition, in which neurons in the hidden layer may need to be duplicated over both nodes. In 0c some arcs can be added to complete the module to a fully connected network. However the weights on these additional arcs should be zero. The new module has again a single hidden layer. So the composition of modules in a parallel way is transparent and therefore feasible. Also the parallel decomposition is feasible, but copies of input and hidden nodes must be made.

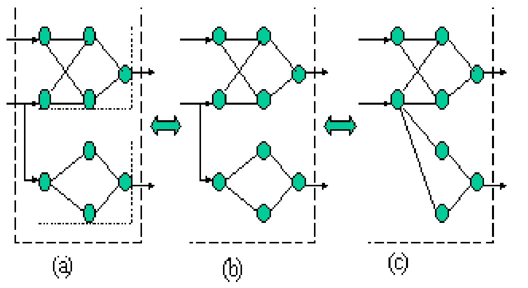

Figure 2. Parallel (de-) composition

2.2 Cascade (de-) composition

In the cascade (de-) composition, some of the outputs of the internal nodes are input to the other internal node, together constituting the overall node. However, a cascade modular network has also *internal* arcs, this in contrast to parallel modular networks.

In case neural networks implement both solid-border nodes a complication occurs. The cascade of both neural networks cannot be easily reconstructed into a neural network with one hidden layer. This is caused by the non-linear (sigmoid) evaluation function in the neurons. It will be impossible to reconstruct the weights on all arcs in the modular network after decomposition.

The resulting network in (b), has a number of peculiarities. First, it has not the same number of hidden layers for all inputs to each of the outputs, since some paths from inputs to some output are longer than others. For the upper inputs, four hidden layers exist, for the lower input only one. However, these layers can be merged into one again. The real issue occurs for decomposition. Due to the non-linear evaluation function in the hidden nodes, the merged hidden nodes cannot be disassembled easily into several hidden layers. So, composition is possible, but decomposition into a cascade network is prohibited by the non-linear evaluation function.

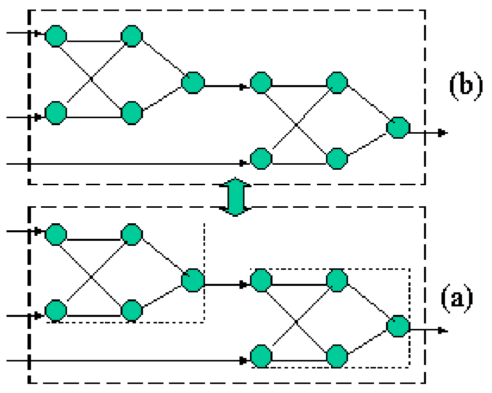

Figure 3. Cascade (de-) composition

3. FUNCTION SPECIALIZATION

The obvious direction for a solution is to investigate relaxation of the sigmoid function. An overview of the historical progress, both in separation as in coverage, is depicted. Research on modifications of the evaluation function **[3]** has been done in order to investigate in simplifications for implementation purposes. It is shown in [4] that the evaluation function of the neurons is not always used on its full domain, but in most cases only in a subset of it, called the active domain. By this, a linearized evaluation function as valid in the neuron's active domain is sufficient to guarantee the accurate overall network functionality.

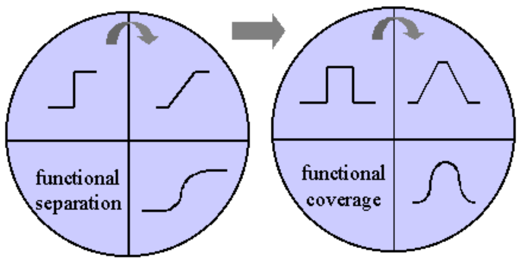

Figure 4. Historical Progress

Next, this type of behavior transformations might help in case of certain cascade transformations, but as shown in [4], the assumption on the evaluation function's domain subset is not valid for all networks. This validity is expressed in **[3]** by a 'redundancy' measure.

This type of transformations is not really suited for compositionality, since the behavior is also modified. Concluding, this type of behavioral transformations is not sufficiently general to enable cascade decompositions. However, it shows that partial linear evaluation functions are quite powerful in neural networks, without loss of accuracy.

Our central assumption is that the sigmoid transfer function *f* can be interpreted as a rounding function on a piece-wise linear function. From [4] we know it as:

$$out_j = \begin{cases} 0 & if \quad in_j \le -b_j/a_j \\ 1 & if \quad in_j \ge (1-b_j)/a_j \\ a_j * in_j + b_j & else \end{cases}$$

where in_j represents sum_j (1)

This piece-wise linear function removes the non-linearities from the network and therefore allows compositionality for cascade topologies, where the elements can be shown to be again networks without need for re-learning after a decomposition.

The property that allows re-computing the weights in decomposition from a composed node is exactly found in the pieces from which the composed function is built. When computing a weight for each line-piece in the composed function it is always possible to find a decomposition for the function. Unfortunately, the original nodes and weights can not be recovered, but it is possible at least to decompose a network into different networks again. That is, networks with one hidden layer.

Assigning a hidden node to each line-piece in the function can easily derive the number of hidden neurons. The direction of each line-piece leads then to the weight(s) for that node. So, piece-wise linear evaluation allows the decomposition of a single network into cascade networks without the need to re-learn both component networks.

4. PIECE-WISE LINEAR EVALUATION

In **[4]** piece-wise linear evaluation for neural networks was presented without comparison with the sigmoid evaluation. It provides a dedicated and little general approach to training and design of the network architecture.

In this paper we compare the alternatives since we want to show the generic feasibility of piece-wise linear evaluation. Analyzing its learning behavior and the accuracy that can be obtained with this type of evaluation can show its feasibility.

As example function to be learned we choose the weighted sine function f(x)=abs(sine(x)) × exp(-x) **[2]**, because it is non-trivial and consists of some sub-functions that can easily be separated into modules. This function is interesting, since it applies the *abs* function that introduces discontinuities on the multiples of $\pi/2$. Learning this type of discontinuity is well known to be difficult for the standard sigmoid neuron evaluation. Furthermore, output normalization of *f*(x) is superfluous.

For all cases a relative low *learning rate* of 0.2 with a *momentum* of 0.3 was applied. The *learning rate* needed to be low for learning convergence of the piece-wise linear evaluation function. For the sigmoid also a value of 0.6 was fine for learning, but makes the comparison invalid.

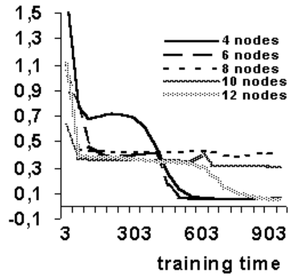

Figure 5. Non-normalized error for learning with hidden node range for sigmoid evaluation

In 0 we have displayed the learning error of a sigmoid evaluation for several hidden neuron counts. There are some apparent learning problems. This may be explained by the typical character of the example function, with its discontinuities at π/2 multiples, that are difficult for the sigmoid, which can be observed in the late descent from the 0.5 error value downwards.

The piece-wise linear function is expected to learn the discontinue behavior on the x=π/2 multiples. The piece-wise linear function was modeled such that it came as close as possible to the sigmoid function. This is done in order to keep the error and weight values close to those of the sigmoid function and prevents training issues.

While learning how large the impact is of the 'smoothing' effect of the sigmoid, especially next to the borders between the different line-pieces. It turns out that the maximal difference at any point between the sigmoid and the piece-wise linear function is minimal when *a*=0.5/2.6 and *b*=0.5 in (1).

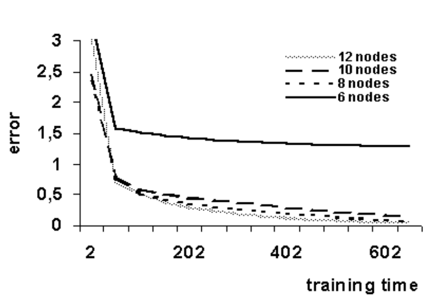

Figure 6. Error for learning with hidden node range for piece-wise linear evaluation

It is easy to see that the linear evaluation learns more stable than the sigmoid. The main difference is the dependence of the learning accuracy on the hidden neuron count. But, when the accuracy is sufficient, the learning will cause no problems. Apparently, the piece-wise linear evaluation is better equipped to handle the discontinuities in the function to be trained.

In other words, the linear evaluation allows an additional degree of freedom in signal approximation. Raising the amount of hidden neurons improves the approximation accuracy, something the sigmoid evaluation is unable to do. A typical result is shown in 0. Where the sigmoid is always suffering from the bias-variance problem, we are less dependent in the case of a piece-wise linear approximation. For instance, the best accuracy for a 15 node network was found to be 5.0 E-2 and for 25 nodes to be 4.2 E-2. This signals a demand for many hidden neurons, that can be structurally improved by mixing sigmoid and linear approximation.

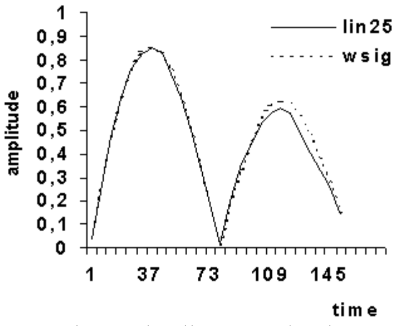

Figure 7 . Piece-wise linear evaluation accuracy for 25 hidden nodes

A number of experiments have been performed to proof the concepts. The relative learning speeds of each composition should provide information on hierarchical network learning, or more specific, provide information on learning with specialized evaluation functions. For reference purposes, we have first checked on the difference between

- Piece-wise linear only, denoted by 'lin'.
- Sigmoid only, denoted by 'sig'.

Applying the *sine* or *exp* function in the hidden nodes makes little sense, since the *sine* and *exp* functions are very specific and thus prohibits learning to an acceptable level of accuracy.

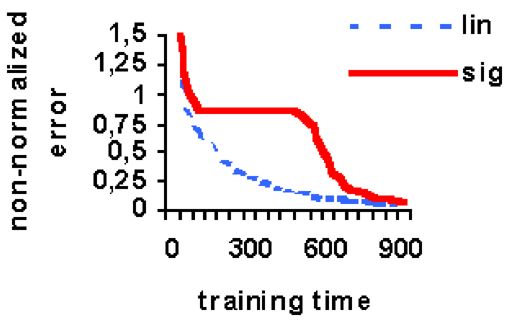

Figure 8. Comparison

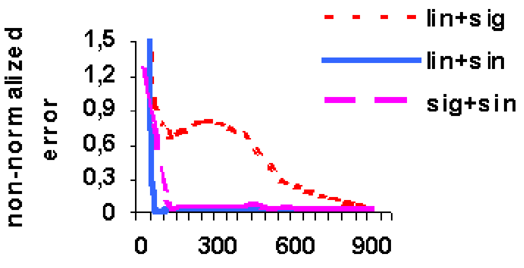

Figure 9. Effect of Modularity

The results are shown in 0. Overall the linear evaluation learns better than the sigmoid, as the sigmoid has difficulties to learn the discontinuities at the x=π/2 multiples. This causes local extrema in the error domain, that become apparent through the difficult learning during the first 500 epochs in 0.

5. STRUCTURING THE PROBLEM

Next, we have applied several combinations of evaluation functions to compare the effect of different specializations:

- 'lin+sig' -Linear and sigmoid, on a 50/50 base
- 'lin+sin' - Linear and sine, on a 50/50 base.
- 'sig+sin' - Sigmoid and sine, on a 50/50 base.

As shows, the *sine* evaluation adds enormously to the learning speed, in combination with a linear evaluation. However, instability occurs in the early phases of the training process, where the symmetry of the evaluation function easily causes a large amount of learning indecision. This results in such large errors, that we have simply omitted these data from the figure.

We see from the 'lin+sig' curve, that the mixture of linear and sigmoid evaluation will be worse than pure linear but still better than pure sigmoid. Typically, the 'lin+sin' and 'sig+sin' training is less predictable than the sigmoid or linear-only learning. It happens quite often that a local optimum determines the best solution found, but with far from an acceptable accuracy. This did not happen for the 'sig+lin' and the 'lin' training.

0 compares two hierarchical combinations 'lin+sin' and 'sig+sin', with two modular networks, both with *sine* and *exp* as specialized modules, the first with a linear evaluation (the 'ext-lin' curve), the second with a sigmoid evaluation (the 'ext-sig' curve). All modules deliver their output to the input of the network to be trained.

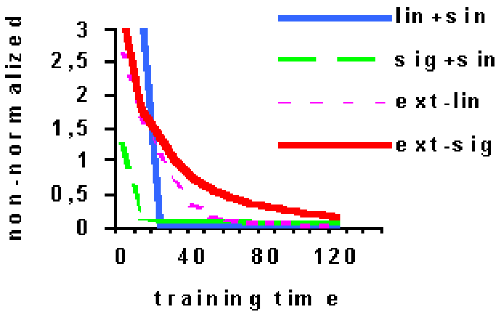

Figure 10. Effect of Hierarchy

Remarkable in 0 is that the modular approach (trains only the second stage network) learns much more smoothly than the hierarchical network. The hierarchical networks learn faster, but in case of the linear-sine combination very large error values in the early stages of the training occur. Again we observe that both hierarchical combinations 'lin+sin' and 'sig+sin' learn unpredictable. In several training runs these combinations converged into a local optimum with unacceptable error levels.

The second part of the experiment compares modularity and hierarchy. This is done by comparison of the specialized evaluation functions with the simple modular approach as described in figure 2. This experiment was done for:

- both the sine and exp functions in the first stage.
- sine in the first stage with x input to the second.
- exp in the first stage with x input to the second.
-

So, in all cases only the network in the second stage needs to be learned, thus ruling out learning scheduling issues in our experiment. Of course, on a larger scale, learning scheduling will still be required.

It is clear that the *exp* function provides the network in the second stage with relatively little information, due to its slow learning. Nevertheless the evaluation function has some influence, as the linear evaluation learns clearly faster.

The *sine* function clearly provides more information, which results in faster learning. In this case, the choice of evaluation function does not have much impact.

Overall what we observe from these experiments is the improved learning by adding functional specialization to the network. Also the linear function may be considered as specialization, due to its ability to fit the discontinuities. This is of importance as linear functions are amenable to linear transformation: function conserving rewriting rules of neural structures that do not require post-learning.

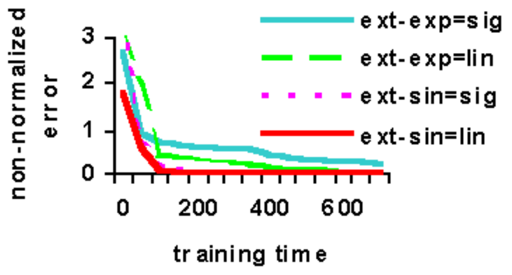

Figure 11. Effect of Compositionality

6. CONCLUSIONS

In this paper we have presented the concept of hierarchical networks which is based on functional specialization. We have described the implications of functional specialization for learning in either a modular or a hierarchical network, illustrated by a simple experimental "proof of concept". Given the results obtained, it is clear that in general functional specialization pays off and increases learning speed. Secondly, modularity provides a smoother learning curve, whereas hierarchy more easily ends up in a fast learning, with the observed larger chance of getting stuck into some local optimum. Further the lower *alfa* value than for pure sigmoid networks are more than compensated by the learning speed increase that is obtained. We have also shown by these experiments that structure engineering is essential to improve neural network performance additional to data engineering.

In contrast to the popular belief, there does not seem to be a clear preference in terms of performance. A piece-wise linear evaluation function in the nodes provides at least the same level of accuracy as a sigmoid evaluation. This is however at the cost of more hidden nodes than are needed in sigmoid based networks. Next to this conclusion, the results also show that for functions with certain types of discontinuities, piece-wise linear outperforms the sigmoid evaluation, since accuracy scales with the number of hidden nodes for the piece-wise linear evaluation. This supports a need for designated specialization of the evaluation functions in neural networks.

This result is relevant for a wide variety of signal processing applications. Next to the example featured here, this includes also a number of adaptive correction filters as for instance in digital decimation for oversampled A/D converters.

But, the main result of our experiments is that compositionality of modular neural networks is possible under the condition of piece-wise linear evaluation. Piece-wise linear evaluation provides the ability to re-compute the weights for hidden nodes in a decomposition from a composite node. This ability enables cascade compositionality, which is sufficient for compositionality in general as we showed. By this result, structure manipulation becomes the key to knowledge engineering for ANN implementations.

REFERENCES

[1] G. Auda and M. Kamel, “Modular neural Networks: A survey”, *Int. Journal of Neural Systems*, Vol. 9, No. 2, (April 1999), pp. 129-151.
[2] T. Caelli, L.Guan and W.Wan, “Modularity in Neural Computin”, *Proceedings of the IEEE* 87, No.9, September 1999, pp. 1497-1518.
[3] E. Barakova, “Learning Reliability: a study on indecisiveness in sample selection” *PhD thesis, Rijksuniversiteit Groningen*, ISBN 90-367-09873, 1999.
[4] L. Spaanenburg, W.J. Jansen, J.A.G. Nijhuis, “Over Multiple Rule-blocks to Modular Net”, Proceedings Euromicro’97, (Budapest, 1997), pp. 698-705, 1997.
[5] L.Spaanenburg, “Knowledge Fusion in Modular Neural Network”, *Proceedings of the NC2000*, Berlin, 2000, pp. 356-362.
[1] [6] A. Sharkey, “Multi-Net Systems” in “Combining Artificial Neural Nets” Ed. A Sharkey, Springer-Verlag, London, ISBN 1-85233-004-X.,1999.
[2] [7] M.H. ter Brugge et al., “On the representation of data for optimal learning”, Proceedings ICNN’95 Perth, pp. 3180-3184, 1995.
[3] [8] H. Keegstra et alieni, “Exploiting Network Redundancy for Low-Cost Neural Network Realization”, *Digest ICNN’96*,Washington, USA, pp. 951-955, 1996.
[4] [9] M. Staley, “Learning with Piece-Wise Linear Networks”, *Int. Journal of Neural Systems*, Vol. 6, No. 1, pp. 43-59, March 1995.

Ad ten Berg: Ad ten Berg obtained a MSc. (Ingenieur) in Electronic engineering with computer science specialisation at the University of Twente in 1983 on the topic of compiler-generation for reconfigurable computer architectures. From 1983 to 1988 he held several positions in industry in the field of CAD for IC-design. In 1988 he joined the University of Twente as assistant professor in the Computer Science faculty. His field of expertise was research in the area of design methodologies for computer architectures. Among others he studied synthesis methodologies and different modelling techniques among which dataflow modelling and relational algebra. Also research was done to the supporting optimisation algorithms, focussing on genetic algorithms in combination with neural networks. In 1995 he joined Philips Research where he headed the Synthesis department of the Electronic Design and Tools group. This group was responsible for several of the internal digital design tools developed in Philips. From 2001 onwards he heads the research group Digital Design and Test at the “Natuurkundig Laboratorium” of Philips Research. His current interests are digital design, transformational design methods, system-level design and neural networks. The engineering of neural networks has his special interest**.**

Lambert Spaanenburg: Lambert Spaanenburg received the M.Sc. degree in Electrical Engineering from Delft University in 1972. He worked in various positions at Twente University (Enschede, The Netherlands) and consulted to industry in the area of microelectronics. During this period he also spent a sabbatical leave with the VENUS group at Siemens Research Laboratories (Muenchen-Perlach, Germany) on software and hardware design frames, which subsequently became part of his Ph.-D. thesis. In 1988 he joined the Institute for Microelectronics Stuttgart (Vaihingen, Germany) to manage the Signal Processing Department. Here, novel software and hardware functions for automotive control were developed and demonstrated. This line of research was continued when he moved in 1993 to Rijksuniversiteit Groningen (Groningen, The Netherlands) to found the Chair in Technical Computing Science. Next to heading the IWI Cluster on Systems Technology, he started the Research & Consultancy Center IMPACTS, the predecessor of Dacolian. Of late, he started the Intelligent Systems on Silicon group at Lund University (Lund, Sweden). Spaanenburg has published more than 160 refereed articles and holds 5 patents. He founded the journal Integration and still is editor of the VLSI Design Journal and of the Journal of Intelligent and Fuzzy Systems.