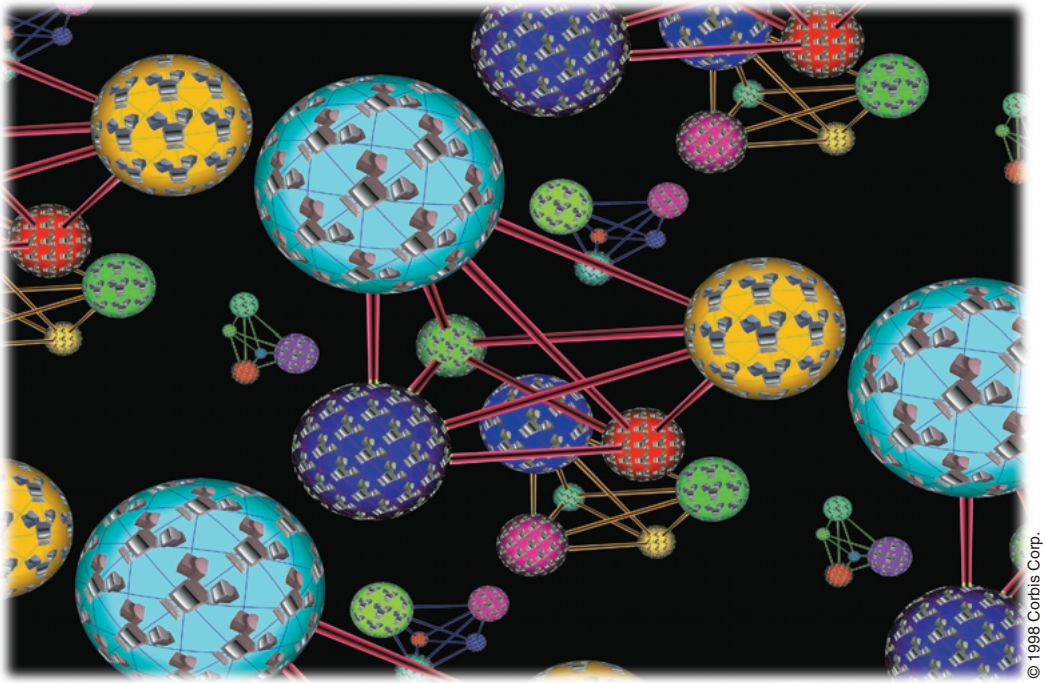
Accuracy versus Complexity in RBF Neural Networks

Cesare Alippi, Vincenzo Piuri, and Fabio Scotti



Time to market, rapid prototyping, and the design of complex embedded systems can only be handled by system-level design [1]. The application is characterized at a very high level of abstraction, and a compiler transforms a coarse description of the application into a finer description where implementation constraints drive the search toward the most appropriate solution. Here, we apply the methodology in radial basis function (RBF) neural networks to detect wear and deformation of railway tracks by automatic video inspection. We show how to design a virtual sensor within an image-processing application. We use "intelligent" systems composed of neural networks to implement virtual instruments for embedded systems.

Neural Networks

Neural networks can be seen as a complex system comprising several, but not necessarily identical, subsystems—the neurons. Very-large-scale integration (VLSI) implementation requires that each component of the system reduces power consumption and complexity to lower computational load and reduce the silicon area. A topologically small network implies the implementation of fewer neurons [22], [23]. Modularity, at the neuron level, allows treatment of the neuron or an ensemble of neurons as atomic units. Such units can be then implemented in one of three ways:

- Parallel solutions from replicating the basic neuron macrocell;

1094-6969/01/$10.00©2001IEEE

- Time-multiplexed architecture, where the basic neuron macrocell is used repeatedly during each time step (as in sequential code execution); and
- Hardware implemented in FPGAs and reconfigured on the fly during code execution [8].

Altogether, the minimal and the modular properties of such a solution allow compact and modular description of the model, which can feed into a hardware/software codesign compiler for an effective code partitioning and hardware synthesis [7].

In this application, we use RBF networks to maximize accuracy under the constraint of minimum complexity [9]. Such topologies are universal function approximators, they can model any real function under loosely constrained hypotheses, provided a sufficiently large set, Z_N, of N measured (*input, output*) pairs is given [10].

An RBF network is a three-layered feed-forward topology (Fig. 1). The hidden neurons outputs, n_r, derive from applying a nonlinear function, or a radial basis function e.g., a Gaussian-like function, to the difference between the input vector and the class centers. A linear output neuron computes the weighted sum of the hidden neurons' outputs. In other words, we can interpret the RBF as a function expansion in a number of radial function bases. The neural network has the coefficient set $\Theta = [\lambda_0,\ldots,\lambda_{n_r},\bar{c}_1,\ldots,\bar{c}_{n_r}]$, which adapts during the training phase to model the approximating function [10], [11].

Accuracy versus Complexity

A tradeoff function cannot give a closed form compromise between accuracy and network complexity. A graph of the unknown function can give the compromise experimentally. In designing an RBF network, we must dimension the number of hidden units, n_r, and the spread of the radial basis function, σ, and then configure Θ. The modularity constraints might require consideration of the same spread function for all the hidden neurons; in such a case, the same σ characterizes all hidden units. The resulting neural network function y, therefore, becomes $y = f_r(\Theta,\sigma,n_r,\bar{x})$.

Training is based on increasing the hidden units to monitor the topological complexity of the neural networks [11], [12]. Additional hidden neurons are incrementally inserted into the neural topology until $J_A(\Theta,Z_N)$ is no longer useful for the given application. A very simple, but effective, training algorithm matches a training input vector to the hidden unit centers and configures the output weights with the classic linear least mean squared procedure. The process iterates until the validation error, evaluated with respect to a cross-validation technique, is below a desired value.

Interestingly, σ can be interpreted as an additional parameter, controlling both the RBF accuracy and its complexity. In fact, the larger σ spreads, the smoother y is. In general, this implies that fewer neurons are required to approximate the given function.

To evaluate the tradeoff between accuracy and complexity, we finally generate the function $A = A(n_r,\sigma)$ by points, with a graph in n_r and σ. Then, we introduce an equivalence relationship on accuracy by considering the set, S_ε, of all neural networks whose loss in accuracy is below a given ε. ε provides a reference point for the accuracy of the best neural networks. We can formalize the concept as follows:

Definition: A network is ε-equivalent when its accuracy performance is within an ε neighborhood of the one provided by the best neural network.

By fixing a loss in performance with ε, we have a set of ε-equivalent neural networks all solving the application. Once ε is fixed, the cardinality of S_ε represents a measure of the space solutions and hence, the difficulty in solving the con-

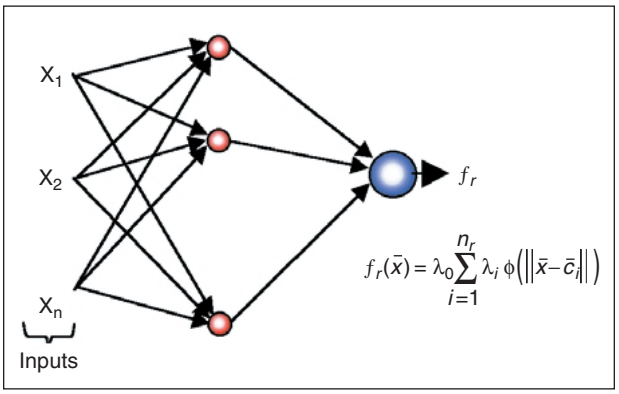


Fig. 1. A radial basis function (RBF) neural network.

```
Methodology (implementation_constraints, accuracy_constraints)
    {

    Create suitable training and validation set
    for NeuronNumber = 1: MaxNeuronNumber {
        For Spread = SpreadMin : SpreadMax {
                Unit (training_constrain are satisfied) {
                    Add one neuron
                    Train the network
                    Test the resulting network
                }
        }
    }
    select networks in the ε-performance set
                /*    wideness of the area selected gives an intuitive
                            idea of the solution space    /*
    if (no networks satisfy accuracy_constraints) {
            there is no solution;
            relax accuracy_constraints;
            restart Methodology;
            }
    }
    else{
    pick the network with
        (lower NeuronNumber AND best accuracy);
    choose the Network with best implementable spread;
    }
}
```

Fig. 2. Pseudo code for the accuracy versus complexity methodology.

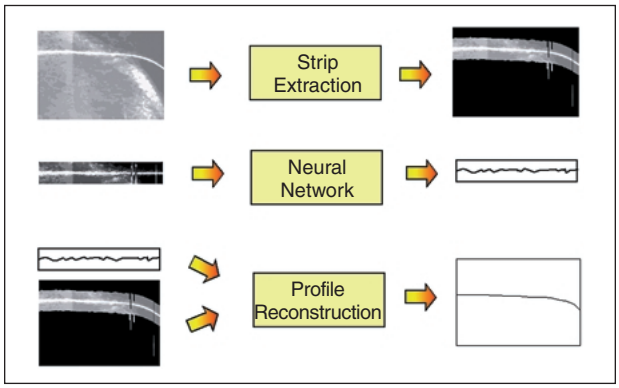


Fig. 3. The processing system for railway tracks.

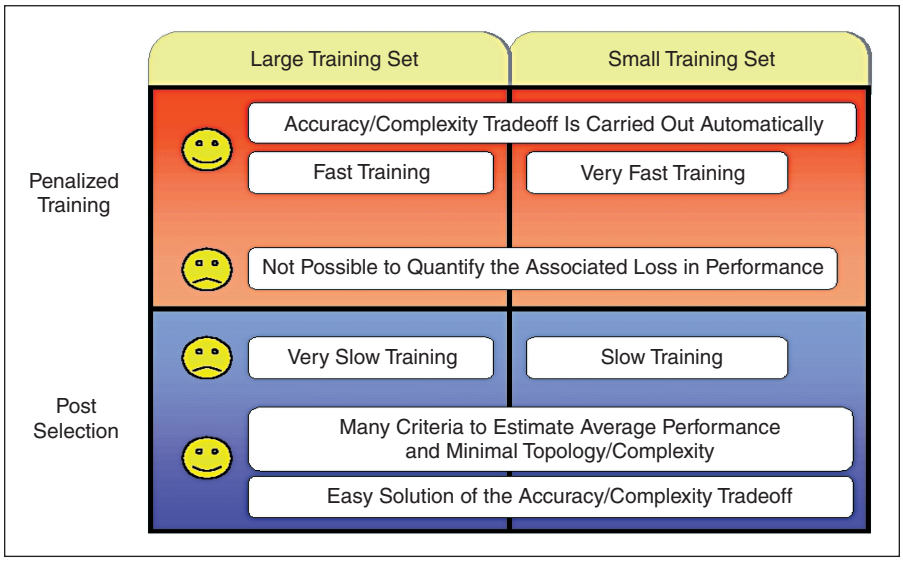


Fig. 4. Online versus off-line constraints integration: advantages and drawbacks.

straint problem. The complexity constraint immediately follows by noting that such networks have, a priori, a different topological complexity. Finally, model selection will select the smallest network, solving the application within the ε equivalence. Fig. 2 provides pseudo code that describes the selection of a network. If the smallest network does not solve the complexity constraint (defined by a boundary on n_r), we must enlarge S_ε by increasing ε.

Railway Wear and Deformation

We studied the wear and deformation of railway tracks by illuminating the track with a laser beam and retrieving the image with a CCD camera [24]. This operation must be carried out on a train traveling at 180 km/h. This speed imposes a real-time constraint on the processing time of 10 ms for an image frame. The feature extraction algorithm requires prefiltering to extract a strip of the image containing the laser-enlightened profile and an RBF neural network to extract the profile in the desired subpixel accuracy.

The neural net receives a vector of the image containing the laser signal. Reflection, saturation of the CCD's pixels, vibration, and the electronic noise of the CCD distort the ideal Gaussian shape of the laser signal, thus making the determination of its maximum value quite difficult. Basically, the RBF function behaves as a pattern-matching filter, which identifies the most likely position of the profile. For each frame, the processing system inputs one column of the image at a time and outputs the vertical coordinate of the reconstructed profile, as depicted in Fig. 3.

The neural net is the crucial part of the processing system because it requires half of the computational time. Software realization of the RBF on a digital signal processor requires the network dimension to be kept at the minimum to satisfy the real-time constraint.

Constraints can be oriented toward either application or implementation. Application constraints relate to features such as a good approximating ability or a smooth solution [3]. Implementation constraints seek minima (the smallest neural network solving the task), robustness (a graceful loss in performance when the network is affected by perturbations), and fault tolerance [4]-[6].

The RBF can be software with code running on a host machine, realized with dedicated hardware (e.g., analog, digital, optical), or mixed, as it happens in hardware/software codesign where the application is partitioned between hardware and software [7]. For instance, if we consider a digital VLSI implementation, a topologically simpler network means less silicon area, a lower power consumption, and faster computation.

Integrating Constraints

Constraints integrate at the system level by following either of two philosophies:

- Online integration: constraints integrate directly during the training phase.
- Off-line integration: constraints integrate after the training phase is terminated and the model is configured.

Online integration adds a penalty term to the training cost function to drive the training procedure towards those Θs that attempt to satisfy the constraints. For instance, if $J_A(\Theta, Z_N)$ is the training function (e.g., a mean squared error loss function [9], [12]) whose optimization focuses only on system accuracy, then $P(\Theta, Z_N)$ is the function integrating the constraint and γ is a penalty term weighting the two contributions. The modified training function becomes

$$J_A(\Theta, Z_N) = J_A(\Theta, Z_N) + \gamma P(\Theta, Z_N).$$

When several constraints need to be considered, $P(\Theta, Z_N)$ and γ become a vector or a complex function.

Training, using $P(\Theta, Z_N)$, biases the solution toward functions with some desirable characteristics (e.g., weight decay for penalizing large weights, number of elementary cells penalizations, high curvature penalizations, extended Tikhonov regularizers for improving fault tolerance) [3], [5], [13], [14].

Penalty terms trade off between accuracy and constraints at a high level of abstraction. This method has some drawbacks:

- $P(\Theta, Z_N)$ biases neural networks that diverge from the optimum by considering only accuracy.
- You cannot quantify the associated loss in performance nor the advantages introduced by $P(\Theta, Z_N)$ with respect to the desired feature.
- It is difficult to tune γs; only few theoretical results are available e.g., see [13] and [14].

Off-line integration tackles the issue after the training procedure has been completed. Constraints are then tested directly by the model. Examples of off-line integration are neuron pruning and unnecessary weights removal, which improves accuracy and integrates smoothness requirements. Network augmentation replicates network topology to improve fault tolerance and reduce the network complexity [15]-[21]. The drawbacks of this approach are:

- Different models must be trained before any selection starts.
- The balance between different constraints with respect to the loss in accuracy requires a tradeoff function based on suitable thresholds.

When the number of data, N, is limited, all data should be used in training the model, but cross-validation techniques cannot be applied to quantify the model accuracy. In such cases, model selection criteria must provide a relative measure for accuracy [14]. Fig. 4 gives a graphical description of constraint integration versus the number of training data and the accuracy selection criteria.

With off-line integration, the compromise between accuracy and constraints occurs after training by inspecting and ranking the accuracy and the level of constraints achieved by various models. Model selection, with respect to accuracy, and constraints integration are, therefore, independent tasks. The user first optimizes the neural architectures, then tests and trades off accuracy and constraints to obtain the best neural network for the application. This is the key for automatic synthesis, where the nonexpert has to tune few, and possibly no, parameters.

We use the following methodology to trade off between accuracy and complexity:

We indicate with a color scale the performance level of the network (a darker color implies better accuracy) in the accuracy function $A = A(n_r, \sigma)$, which is plotted in Fig. 5. The color bar denotes the standard deviation of the output error in pixels. The blank region refers to unacceptable RBF networks because their accuracy in identifying the correct profile of the railway tracks was not at the subpixel level.

If the goal is accuracy, then the selection procedure is extremely simple and suggests the RBF with the best accuracy. Conversely, if computational complexity is important, we must fix ε and identify the S_ε regions. Fig. 6 shows two S_ε sets characterized by $\varepsilon = 0.025$ and $\varepsilon = 0.04$. By increasing ε, thereby tolerating a larger error, the S_ε area grows significantly. Consequently, we can expect a large margin of topological complexity. In particular, for case $\varepsilon = 0.04$, we have the RBF neural networks solving the application with between eight and 16 hidden units. This depends on σ, which is a relevant parameter in solving the compromise between accuracy and complexity.

The smallest neural networks solving the tradeoff correspond to the left border. In particular, the minimal networks satisfying $\varepsilon = 0.025$ and $\varepsilon = 0.04$ accuracy constraints require 12 and eight hidden neurons, respectively.

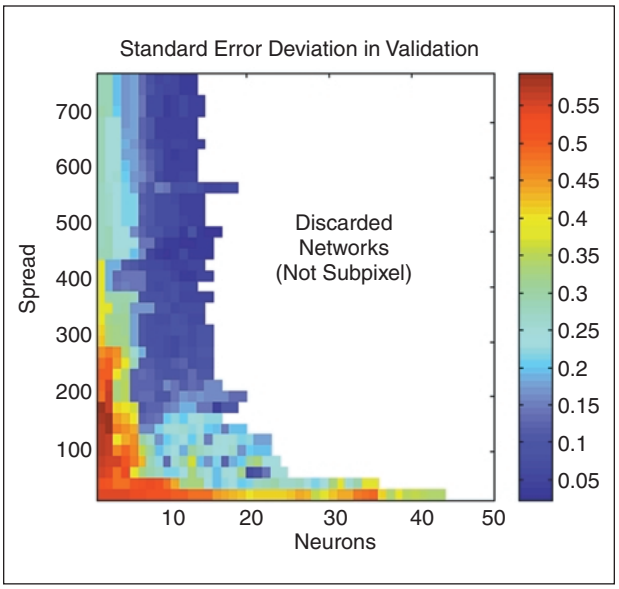


Fig. 5. The accuracy function $A = A(n_r, \sigma)$.

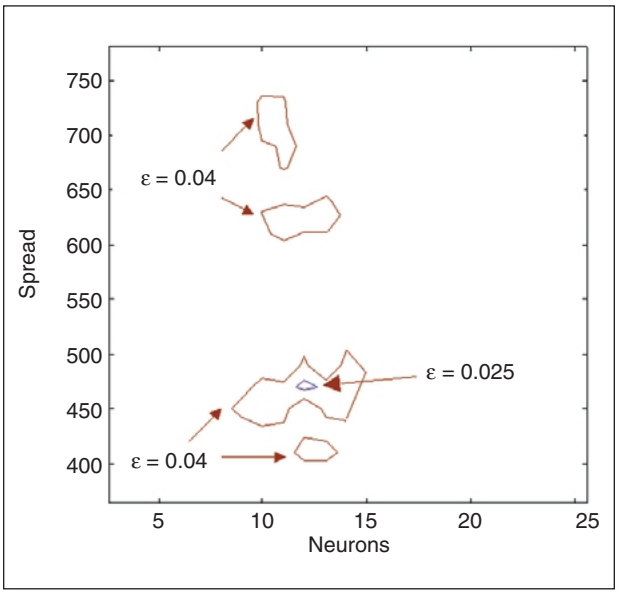


Fig. 6. Two S_ε sets.

Summary

We have introduced a methodology for solving the tradeoff between accuracy and complexity in complex virtual systems directly at the system level. Such methodology can be inserted in an application-level compiler for transforming a high-level description of the application into a lower level. This can then be fed into a hardware/software codesign compiler for final system implementation. Off-line integration of constraints relaxes model accuracy by introducing the concept of neural networks equivalent according to accuracy. The complexity criterion help select the smallest neural network topology within this set. The methodology is the key element for an effective high-level synthesis where few or no application tuning parameters need to be set by the designer. Indirectly, the methodology supports a system level integration of the requirement for low-power consumption, a particularly appealing constraint for embedded system design.

References

[1] C. Alippi and V. Piuri, "Topological minimisation of multi-layered feed-forward neural networks by spectral decomposition," in *Proc. 1992 Inter. Joint Conf. Neural Networks*, Beijing, China, pp. 805-810, 1992.

[2] C. Alippi, E. Casagrande, V. Piuri, and F. Scotti, "Composite real-time image processing for railways track profile measurement," *IEEE Trans. Instrum. Meas.*, vol. 49, pp. 559-564, June 2000.

[3] C. Alippi, S. Ferrari, V. Piuri, M. Sami, and F. Scotti, "New trends in intelligent system design for embedded and measurement applications," *IEEE Instrum. Meas. Mag.*, vol. 2, pp. 36-44, June 1999.

[4] J.A. Anderson, *An Introduction to Neural Networks*. Cambridge, MA: MIT, 1995.

[5] G. De Micheli and M.G. Sami, Eds., *Hardware/Software Codesign*, (NASO ASI Series), vol. 310. Norwell, MA: Kluwer Academic, 1995.

[6] P.J. Edwards and A.F. Murray, "Towards optimally distributed computation," *Neural Computation*, vol. 10, pp. 987-1005, 1998.

[7] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Computation*, vol. 7, pp. 219-269, 1995.

[8] I. Guyon, V. Vapnik, B. Boser, L. Bottou, and S. Solla, "Structural risk minimization for character recognition," presented at Advances in Neural Processing Systems (NIPS), Vail, CO, 1992.

[9] B. Hassibi and D.G. Stork, "Second order derivative for network pruning: Optimal brain surgeon," presented at Advances in Neural Processing Systems (NIPS), Vail, CO, 1993.

[10] A. Krogh and J.A. Hertz, "A simple weight decay can improve generalisation," in *Proc. Advances in Neural Processing Systems (NIPS)*, Vail, CO, pp. 950-957, 1992.

[11] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal brain damage," *Proc. Advances in Neural Processing Systems (NIPS)*, Vail, CO, pp. 598-605, 1990.

[12] J.A. Leonard, M.A. Kramer, and L.H. Ungar, "Using radial basis function to approximate a function end its error bounds," *IEEE Trans. Neural Networks*, vol. 3, pp. 624-627, July 1992.

[13] A.U. Levin, T.K. Leen, and J.E. Moody, "Fast pruning using principal components," in *Proc. Advances in Neural Processing Systems (NIPS)*, Vail, CO, pp.35-42, 1993.

[14] M.J.L. Orr, "Regularisation in the selection of radial basis function centres," *Neural Computation*, vol. 7, pp. 606-623, 1995

[15] W. Pedrycz, *Computational Intelligence, an Introduction*. Boca Raton, FL: CRC Press, 1998.

[16] D. Phatak and I. Koren, "Complete and partial fault tolerance of feed forwards neural nets," *IEEE Trans. Neural Networks*, vol. 6, pp. 446-456, March 1995.

[17] G. Seber and C. Wild, *Nonlinear Regression*. New York: Wiley, 1989.

[18] Y. Tohma and Y. Koyanagi, "Fault-tolerant design of neural networks for solving optimisation problems," *IEEE Trans. Comput.*, vol. 45, pp.1450-1455, 1996.

[19] A.S. Weigend and D.E. Rumelhart, "The effective dimension of the space of hidden units," in *Proc. 1991 Int. Joint Conf. Neural Networks*, Singapore, pp.2069-2074, 1991.

[20] "Configurable computing," *IEEE Computer Mag.*, vol. 33, April 2000.

[21] "Design tools for embedded systems," *IEEE Design & Test Mag.*, vol. 17, 2000.

[22] IEEE Annual Workshop on VLSI: System Level Design, Orlando, Florida, April 1998.

[23] *IEEE Trans. Circuits Syst.*, special issue on neural networks, vol. 36, May 1989.

Cesare Alippi obtained his Dr. Ing. degree in electronic engineering summa cum laude in 1990 and the Ph.D. in computer engineering in 1995, both from the Politecnico di Milano of Milano, Italy. Currently, he is an Associate Professor in Information Processing Systems at the Politecnico di Milano. His interests include neural networks, genetic algorithms, signal and image processing, and VLIW architectures. His research results have been published in more than 70 technical papers in international journals and conference proceedings.

Vincenzo Piuri obtained his Ph.D. in computer engineering in 1989, from the Politecnico di Milano of Milano, Italy. Between 1992 and September 2000, he was Associate Professor in Operating Systems at Politecnico di Milano. Since October 2000, he has been a Full Professor in computer engineering at the University of Milano. His research interests include distributed and parallel computing systems, computer arithmetic, application-specific processing architectures, digital signal processing architectures, fault tolerance, and neural networks. The original results of his research have been published in more than 150 papers in book chapters, international journals, and proceedings of international conferences. He has been the Associate Editor of the *IEEE Transactions on Instrumentation and Measurement* since 1998.

Fabio Scotti obtained his master's degree in electronic engineering in 1998 from the Politecnico di Milano of Milano, Italy. Currently, he is working toward his Ph.D. in computer science at the same university. His research interests include signal processing and neural technologies for image processing and embedded systems.