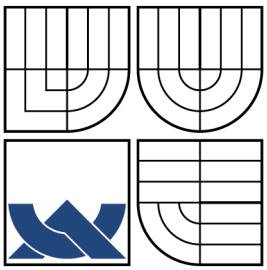
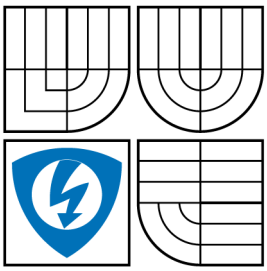


BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

PARALLEL COMPUTING AND NEURAL NETWORKS IN BEHAVIORAL MODELING

BEHAVIORÁLNÍ MODELOVÁNÍ POMOCÍ PARALELNÍCH VÝPOČTŮ A NEURONOVÝCH SÍTÍ

SHORT VERSION OF PHD THESIS
TEZE DISERTAČNÍ PRÁCE

AUTHOR
AUTOR PRÁCE
Ing. JITKA SVOBODOVÁ

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. ZBYNĚK LUKEŠ, Ph.D.

BRNO 2012

ABSTRACT

This thesis is focused on methods for the aircraft equipment modeling. The first part provides a brief overview of classical system modeling approaches used for system description, identification, and modeling. Then adaptive, fuzzy and hybrid methods used mainly for black-box system modeling are introduced. Aim of the thesis is to develop an algorithm for identification and modeling of a general system, which can be nonlinear, dynamic and complex. Multiple inputs and multiple outputs of model are assumed.

The main part of the thesis introduces a new method which falls into the hybrid systems. It combines fuzzy approach with parametrically defined rules and general regression neural network. Firstly, the fundamentals of simple general regression neural network and its smoothness parameter determination are presented. Secondly, the general regression neural network with the fuzzy rules is introduced. Third part of the thesis is focused on the parallel computing, one of the main objectives. Block diagram for parallel computing in Matlab and CUDA is provided, as well as the basic structure of CUDA source code.

Finally, the method is verified on data obtained from the measurement of a miniaturized aircraft model using the antenna outside the aircraft and the probe inside the fuselage of the aircraft model. The validation of the method is done using mean squared error and compared to mean squared error of corresponding model performed using the multilayer neural network with backpropagation learning and Levenberg-Marquardt algorithm.

KEYWORDS

fuzzy general regression neural network, neuro-fuzzy, system identification

ABSTRAKT

Tato disertační práce se zabývá metodami modelování elektronického zařízení letadel. První část je stručným přehledem klasických metod modelování systémů a adaptivních, fuzzy a hybridních metod používaných převážně k black-box modelování.

V hlavní části práce je rozebrána metoda, která patří mezi hybridní systémy, protože kombinuje fuzzy systém s parametricky definovanými pravidly a regresní neuronovou síť. Nejprve je zmíněn základní princip regresní sítě a způsob určení jejího parametru strmosti, dále se kapitola zabývá zavedením fuzzy pravidel do této sítě. Třetí část je zaměřena na jeden z hlavních bodů práce, paralelní výpočty. Výsledný algoritmus je navržen pro paralelní zpracování a implementován v CUDA a Matlabu.

V závěru práce je metoda ověřena na datech získaných z měření zmenšeného modelu letadla. Ověření je provedeno pomocí střední kvadratické odchylky a srovnáním s odpovídajícím modelem vytvořeným pomocí vícevrstvé neuronové sítě trénované zpětným šířením chyby s algoritmem Levenberg-Marquardt.

KLÍČOVÁ SLOVA

fuzzy regresní neuronová síť, neuro-fuzzy, identifikace systémů

SVOBODOVÁ, Jitka *Parallel Computing and Neural Networks in Behavioral Modeling*: short version of phd thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Radio Electronics, 2012. 32 p. Supervised by Ing. Zbyněk Lukeš, Ph.D.

CONTENTS

1 INTRODUCTION

1.1 INTRODUCTION

The modeling takes part in complex devices design. Well designed mathematical model provides experiments similar to those with the real object while the case of breakdown is avoided. There is also no need of real object, so the costs of preliminary experiments are lower. The behavioral modeling is aimed to estimate a simplified mathematical black box model of a system based on measurement of input and output signals of the system. The resultant model is useful for prediction or simulation of the model output while the inner structure of the original system is usually unknown. Existing methods differ from each other according to the current systems properties, but general model is assumed as nonlinear, stochastic and dynamical, to be as close as possible to the real system.

These formerly intractable problems are being solved since the 18^{th} century when the probability and curve fitting approaches were presented. The algorithms are simple and effective, but they fit the data reliably only in the model validity range and in cases where the known dataset is large enough thus the model parameters can be estimated. Some of them fit the measured data exactly, so these methods were not suitable for real systems corrupted by noise. Later, these techniques were improved by their combination or by implementation of prediction and correction factors.

Nowadays, these algorithms are still modified and used, but many of them deal with a small degree of freedom and poor generalization ability. Today's research is focused mainly on hybrid methods, as the negative properties of the individual approaches can be suppressed when the method is well designed. The main disadvantage is the computational demand which can be solved using the parallel computing, that is available even for today's multicore personal computers, graphic cards and, of course, supercomputers. The nonlinear system modeling is an objective of many recent articles. Most common methods treated are neural networks, evolutionary algorithms, fuzzy systems, support vector machines and their various combinations.

The authors of other papers combine previously mentioned methods with Wiener Theory, wavelets, projections and other various methods. Almost all the publications are oriented to SISO (single input single output) models and many presented methods are demonstrated on simple or easily predictable functions (i.e. sine function), and they usually are not valid in case of more complex function representing the real system, or their implementation is not possible because of the hardware demands.

The aim of this thesis is to find a robust algorithm to identify and simulate the system of the aircraft equipment (fuel level transmitters, inputs and outputs of electronic blocks, supply systems, etc.) or to solve the electromagnetic compatibility issues. This equipment is considered as the input/output model. High precision and reliability of the model are required while the system is nonlinear and dynamical by nature, so the nonlinearity and system dynamics cannot be omitted or significantly simplified.

The black box nonparametric model is based only on measurement data with no prior information about the system structure. It should be able to simulate accurately also the data out of range of the data available from the measurements. One of the points to consider is also the training set of data which is corrupted by noise because it comes from real measurements. Thus the model is supposed to have good generalization ability while the good precision is retained, and it should be applicable in both time and frequency domain. Aim to satisfy these requirements leads to the use of hybrid model designed to be run in parallel. The need of multiple outputs of the model should be also considered.

1.2 STATE OF THE ART

1.2.1 Adaptive and fuzzy methods

Fuzzy modeling

The complex system might be defined imprecisely, which is a problem for both classical parametric and nonparametric approaches. Fuzzy modeling allows¨the use of imprecise description usually given by a joint function or verbal description of the relationship between signals in a model. Basic principle of the most widely used Takagi-Sugeno model is to divide the whole solution space into small areas and their linearization using the local linear models. The fuzzy rule base is a collection of statements about the relationship between measured variables and it is a fundamental of fuzzy model creation. The variables describing the system are called regressors and they are associated to a number of attributes (possible cases) with their levels of membership. The membership function $\mu_A(\varphi)$ assumes values between 0 and 1. If set φ has r attributes $A_i, i = 1, \dots r$, the membership function would be

$$\sum_{i=1}^{r} \mu_{A_i}(\varphi) = 1, \forall \varphi. \tag{1.1}$$

If a rule base is complete, it covers all possible combinations of attributes A_i. This is satisfied when the number of rules equals to the product of the number of the attributes associated to each regressor:

$$\sum_{j=1}^{p} \prod_{i=1}^{d} \mu_{A_{j,i}}(\varphi_i) = 1, \tag{1.2}$$

or

$$\mu_{A_{j*}} = \prod_{i=1}^{d} \mu_{A_{j,I}}(\varphi_i). \tag{1.3}$$

If the rules are defined as linguistic, in the fuzzification phase, the inputs are transferred to the degrees of membership which are combined according to the fuzzy operators in the aggregation phase. Subsequently, in the activation phase the output activations of the rules are calculated. After the defuzzification, the output estimation $\hat{y}$ is obtained.

The most commonly used Takagi-Sugeno model is defined by fuzzy rules:

$$\begin{aligned} R_1 &: IF\, u_1 = A_{11}\, AND \dots AND\, u_p = A_{1p}\, THEN\, y = f_1(u_1, u_2, \dots, u_p) \\ R_2 &: IF\, u_2 = A_{21}\, AND \dots AND\, u_p = A_{2p}\, THEN\, y = f_2(u_1, u_2, \dots, u_p) \\ &\vdots \\ R_i &: IF\, u_i = A_{i1}\, AND \dots AND\, u_p = A_{ip}\, THEN\, y = f_i(u_1, u_2, \dots, u_p) \end{aligned} \tag{1.4}$$

Zero and first order Takagi-Sugeno fuzzy system are mostly used for the nonlinear system modeling. The zero order means that the

functions $f_i(\cdot,)$ are constants. For the first order of Takagi-Sugeno fuzzy model, the consequent is a linear function of the inputs:

$$y = w_{i0} + w_{i1}u_1 + w_{i2}u_2 + \dots + w_{ip}u_p, \tag{1.5}$$

and the the output is evaluated as:

$$\hat{y} = \frac{\sum_{i=1}^{M} f_i(\overline{u})\, \mu_i(\overline{u})}{\sum_{i=1}^{M} \mu_i(\overline{u})}. \tag{1.6}$$

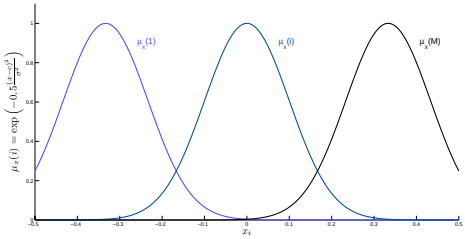


Fig. 1.1: Example of membership functions: Gaussian

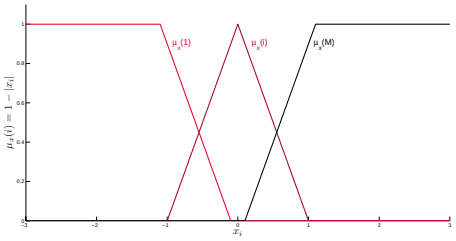


Fig. 1.2: Example of membership functions

Neural networks

The neural networks are an efficient tool for the black box modeling or simulating the nonlinear systems because their universal approximation abilities and learning capabilities. They are also able to model very large, complex, nonlinear and multidimensional structures. The neural networks used because of their capability to adapt, generalize and for their universality. The main disadvantage can be the long time that is necessary for learning, so they are useful when the system cannot be treated by analytical or numerical solution. In this section, the three frequently used types of networks are presented.

Multilayer Perceptron The multilayer perceptron (MLP) is one of the basic neural network architectures. The fundamental formulation of the network feedforward function describes the dependency of the estimated output on the inputs and weights in accordance with the function of individual elemental neurons [6, 8].

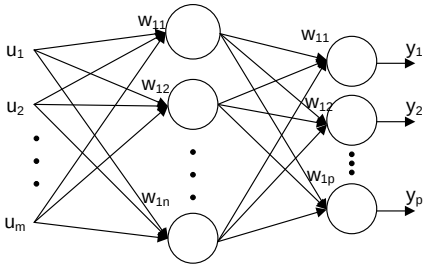


Fig. 1.3: Multilayer neural network

$$\hat{y} = \sum_{i=0}^{n} w_i \Phi_i \left(\sum_{j=0}^{m} w_{ij} u_j \right), \tag{1.7}$$

where m is the number of inputs, n is the number of hidden layers neurons. w is the weight vector or matrix, Φ is the function of the neuron (usually the same within the layer) and u is a vector of inputs.

This network architecture is trained using the backpropagation algorithm, which is based on the generalization of delta rule. Thus the function of neurons is usually sigmoidal, tangential, for the hidden layers, and linear for the output layer, respectively. The neuron function has to be differentiable. [6]

This method is simple, but not very effective, so its properties are often improved by reguralization methods. For example, the Bayesian regularization [6] is suitable for improving the generalization ability and suppressing noise in training data. The accuracy is very high and the sensitivity to noise is low, so this type of network is suitable for the system identification. However, there are some restrictions, as well. The training dataset has to be large enough, the learning speed is slow and the topology optimization methods are computationally intensive.

Radial Basis Neural Network The radial basis network has a different structure than the multilayer perceptron. The number of layers is always the same: input layer, one hidden layer and summation layer. Input layer consists of one neuron for each pattern. In this layer, the range of the values is evaluated by subtracting the median and dividing by the interquartile range. The interconnection between input and hidden layer is provided by connection of each neuron of the input layer to each hidden layer neuron. The number of neurons in hidden layer is determined by the learning. Each neuron with radial basis function is placed with its center on a point in solution space. The number of dimensions depends on the number of input variables and the width of RBF is given by learning. In the third layer, summation layer, the values from hidden layer are multiplied by the weights and the network output is given by their sum. A bias value of 1.0 is multiplied by a weight W_0.

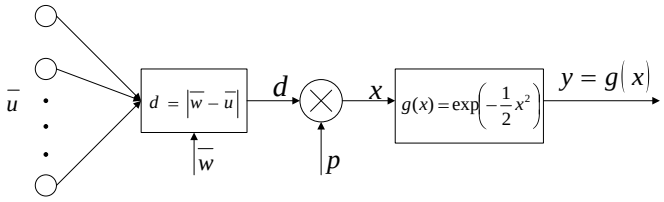


Fig. 1.4: The basic principle of the radial basis network

The radial principle consists of the input $\overline{u}$ and center vector $\overline{w}$ distance computation with respect to the p parameter. Then the distance x is transformed by the activation function $g(x)$, which is usually the Gaussian function:

$$g(x) = \exp\left(-\frac{1}{2}x^2\right). \tag{1.8}$$

The distance is evaluated from the hidden layer neuron parameters:

$$x = |\overline{u} - \overline{w}|\, p = \sqrt{(\overline{u} - \overline{w})^T\, p\, (\overline{u} - \overline{w})}, \tag{1.9}$$

and the output is computed as:

$$\hat{y} = \sum_{i=0}^{M} w_i g_i(x) \cdot (|u - w_i|\, p_i). \tag{1.10}$$

The RBF network training is more complicated then the multilayer network training, usually the outer layer parameters are estimated by the least squares after the hidden layer parameter determination. [6]

General Regression Neural Network The general regression neural network (GRNN) belongs to the probabilistic neural network category. It is supposed to perform regression for the continuous target variable. The accuracy is often better for the GRNN than for the multilayer network and one of the main advantages is insensitivity to the outliers. The network is suitable for the multidimensional data evaluation. Each training sample is treated as the mean of normal distribution, so the output can be estimated as:

$$\hat{y} = \frac{\sum_{i=1}^{N} y(i)\,\Phi_i}{\sum_{i=1}^{N} \Phi_i}, \tag{1.11}$$

$$\Phi_i = \exp\left(-\frac{1}{2}\frac{|\overline{x} - \overline{x}(i)|^2}{\sigma^2}\right), \tag{1.12}$$

where the standard deviation σ determines the smoothness, the number of basis functions Φ corresponds to the number of data samples. [4]

The GRNN behaves similarly as a look-up table [2], so it belongs to the memory-based network class. Thus the necessary training dataset is smaller than the dataset needed to train the optimization-based networks (for example the multilayer network). The major drawback of the GRNN is a need of many basis functions (their number corresponds with the number of training samples), so this method is very computationally intensive and the parallel approach should be considered.

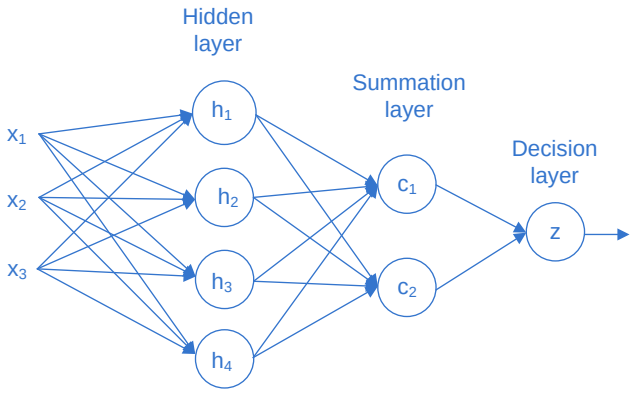


Fig. 1.5: General regression neural network

The figure 1.5 shows the general structure of general regression neural network. The first layer neurons represent the probability density function and thus they determine how the input fits the pattern unit. The number of neurons in this layer corresponds to the number of training patterns. The second layer is called the summation layer. The values coming to one of this layer neurons are weighted using Y_i, which is a corresponding value of training output. Output of this neuron is used as a denominator for the next layer. Another neuron of the summation layer has all the inputs weighted with one. In the last layer, the two values are divided and the output is evaluated.

Evolutionary algorithms

In recent articles, genetic programming and genetic algorithms are mentioned as methods for nonlinear system modeling. These approaches are not very commonly used themselves, but some authors use them in a combination with one of previously mentioned approaches (mostly the classical ones). Both methods create the initial population in their first step. Subsequently, the fitness function is evaluated. The selection of the fitness function pattern is usually made using any other method. The genetic operators used in genetic algorithms are selection, crossover and mutation. Genetic programming uses only the selection operator. In [28], the model is assumed as a NARMAX model. Both the model and the time-delay are encoded using the two bits binary numbers. The population of initial individuals is generated randomly. The fitness function is evaluated using the equation:

$$f = \frac{1}{1+\sum_{t-1}^{N} |y(t)-\hat{y}(t)|} - \omega \frac{l}{100} \tag{1.13}$$

where N is the number of input/output data, l is the model length and ω is a penalty parameter. This equation determines the ratio between error given by $y(t)-\hat{y}(t)$ and complexity of the model. The selection is done by roulette selection (with elitism) according to the fitness, then the crossover and mutation operators are used. The whole procedure is repeated for a given number of generations.

Hybrid algorithms

Today's publications tend to use various combinations of the known methods of system modeling. Their aim is common and obvious: to improve the properties of known approaches and design a new one which is able to deal with the nonlinearity, to be precise enough and to have good approximation ability. From the classical approaches, the authors usually choose the ARMAX or the NARMAX model which is then combined with an evolutionary algorithm or a neural network.

Neuro-fuzzy model with LOLIMOT algorithm Basic idea of this neuro-fuzzy model is to replace the nonlinear function $y(x)$ with scalar product of local linear function vectors $\hat{y}_i(x$ and the validity function vector $\Phi_i(x)$.

$$\hat{y} = \sum_{i=1}^{M} \hat{y}_i \Phi_i(x) = \sum_{i=1}^{M} (w_{i0} + w_{i1}u_1 + w_{i2}u_2 + \ldots + w_{ip}u_p)\,\Phi_i(x), \tag{1.14}$$

where $x = (u_1, u_2, \ldots, u_p)$ is the system input vector, $w = (w_{i0}, w_{i1}u_1, w_{i2}u_2, \ldots, w_{ip}u_p)$ is local linear model (LLM) parameter vector, $\Phi_i(x)$ is the validity function of i^{th} LLM dependent on the system inputs, x. M is the number of LLM. The validity functions are determined as normalized Gaussian functions:

$$\Phi_i(x) = \frac{\mu_i(x)}{\sum_{j=1}^{M} \mu_j(x)}, \tag{1.15}$$

where

$$\mu_i(x) = \exp\left(-\frac{1}{2}\left(\frac{(u_1-c_{i1})^2}{\sigma_{i1}^2} + \frac{(u_2-c_{i2})^2}{\sigma_{i2}^2} + \ldots + \frac{(u_p-c_{ip})^2}{\sigma_{ip}^2}\right)\right) \tag{1.16}$$

is the membership function (MSF) with the centers in c_{ij} and the standard deviations:

$$\sigma_{ij} = k_\sigma \Delta_{ij}, \tag{1.17}$$

where $Delta_{ij}$ is the size of i^{th} subarea in j^{th} dimension and k_σ the proportion of the subarea width and standard deviation. The identification algorithm is known an local linear model tree (LOLIMOT) [15]. The measured data area is divided with orthogonal cuts into the subplanes to aproximate the system in current subarea. For the initial model, the measured data area boundary is designated. The whole area is mentioned as a single linear model and the validity function $\Phi_1(x) = 1$ for the initial conditions. The linear model coefficients are estimated and the initial number of linear models is set to 1. The error function I_i is evaluated for every LLM for $i = 1, 2, \ldots, M$. LLM with the highest error value I_i is indicated with l.

$$I_i = \sum_{j=1}^{N} (\hat{y}_j - y_j)^2 \Phi_i(x_j) \tag{1.18}$$

The division cut is performed for all the dimensions and the MSF μ_i is generated for every subarea. The validity function Φ_i of the original LLM is determined and the parameters of the new LLMs are estimated. The error function value is computed:

$$I = \sum_{i=1}^{M} \sum_{j=1}^{N} (\hat{y}_j - y_j)^2 \Phi_i(x_j) \tag{1.19}$$

The best division cut is kept and M is increased by 1. The whole algorithms (except the initial step) is repeated until the ending criteria are met.

General Regression Neuro-Fuzzy Model The general regression network is an adjustment of the radial basis network and it belongs to a group of probabilistic, memory-based neural networks. Use of this network provides a high precision solution if the solution space is significantly larger than the training set. The neuro-fuzzy approach based on this kind of network allows better multidimensional data treatment. Exact results of this approach have not been published, so after the implementation, the statistical analysis and benchmark testing is to be performed.

One of the possible GRNN neuro-fuzzy architectures is derived from the equations 1.3 and 1.11. This form is the simplest one derived directly from the equations for the fuzzy systems and general regression neural network and it is supposed to be changed in accordance to the results to fit the model requirements [4].

The architecture of GRNN consists of five sequentially connected layers. First hidden layer is composed of blocks with n fuzzy basis functions in each and realizes fuzzification of the input variables vector. Second hidden layer implements aggregation of membership levels that are computed in first layer, and consists of the multiplication blocks. The layer of synaptic weights forms the third layer. Fourth layer consists of two summation units and computes the sums of output signals from the second and third layers. The output layer, the summation layer, gives the output signal.

The network architecture is simple and it comes out from the theory of neuro-fuzzy [2]. Like other neuro-fuzzy systems, this network has five layers. In the first layer, the input data is fuzzified, their degrees of membership are calculated. The second layer represents the aggregation phase where the fuzzy operators are applied. In the activation phase (third layer), the weights are defined. These three hidden layers are indicated with blue color in Fig. 1.6, because the can be run in parallel. This is one of the advantages in comparison with the multilayer network, most of which has to be run sequentially. The fourth layer performs an accumulation, where all the rules are joined together. In Fig. 1.6, this phase is done by summation of the fuzzy part outputs. The last layer represents a centre of gravity method

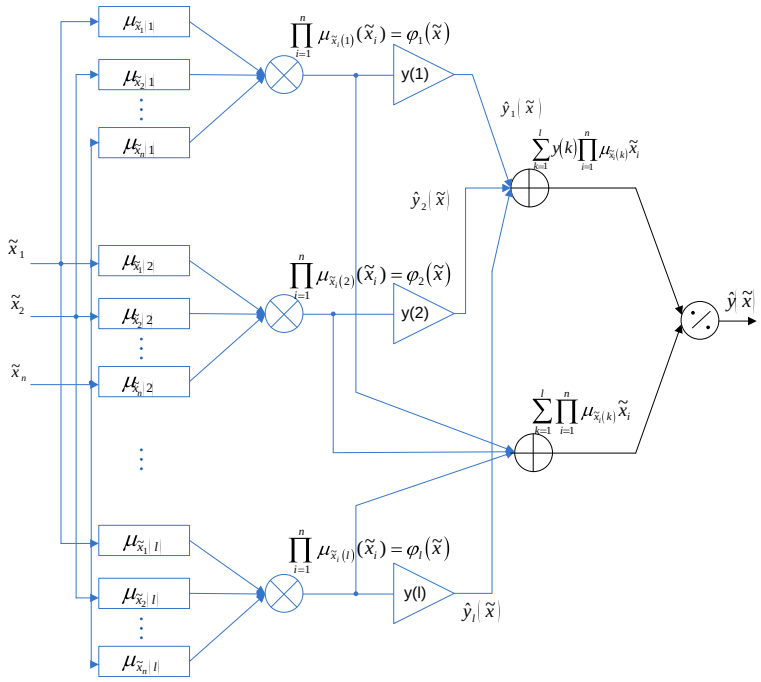


Fig. 1.6: Neuro-fuzzy network

[2, 5] necessary for a crisp value extraction from a fuzzy set. Then the $\hat{y}(\tilde{x})$ estimation is achieved [4].

The nonlinear system problem is generally described by the equation 1.20.

$$y(k) = F(x(k)), \tag{1.20}$$

where $y(k)$ and $x(k)$are output and input, respectively, and the $F(\ldots)$ is an unknown nonlinear operator. The learning dataset has to be in form of $\{x^*(k), y^*(k)\}$, $k = 1, 2, \ldots n$. The system response $y(k)$ can be estimated using input signal x so that $\hat{y}(k)$ is taken from [4].

$$\hat{y}(x) = \frac{\sum_{k=1}^{n} y^*(k)\,\varphi(D(k))}{\sum_{k=1}^{n} \varphi(D(k))}, \tag{1.21}$$

where $D(k)$ is an Euclidean distance between x and $x^*(k)$, $\varphi(k)$ is a GRNN kernel function.

$$D^2(k) = \sum_{i=1}^{j} \left(\frac{x_i(k) - x_i^*(k)}{\sigma(k)} \right)^2. \tag{1.22}$$

In the fuzzification layer, the membership functions $\mu_1, \mu_2, \ldots \mu_n$ are derived for each vector x_i. The centers of the Gaussian membership functions correspond to the individual training patterns, so their number is equivalent.

The learning samples are normalized, so their value is adjusted using 1.23:

$$x_i(k) = \frac{x_i^*(k) - x_i^{*min}}{x_i^{*max} - x_i^{*min}} - 0.5, \tag{1.23}$$

$$x_i^*(k) = (x_i^*(k) + 0.5) \cdot \left(x_i^{*max} - x_i^{*min} \right) + x_i^{*min}. \tag{1.24}$$

The determination of the fuzzy rule is to be done in the further work using a statistical analysis of the various posibilities of numbers of kernel functions and their shape.

1.3 AIM OF THE THESIS

Aim of the thesis is to develop an algorithm for identification and modeling of a general system, which can be nonlinear, dynamic and very complex. The resultant model is a black box which is able to deal generally with multiple or multidimensional input and output. Since the model is assumed to be nonlinear, the statistic and curve fitting methods themselves are unsuitable. The neural networks provide a method with many degrees of freedom while the high precision of the model is achieved. The issue is to find appropriate type and structure of network which is able to learn relatively fast, have good generalization abilities and high precision.

Some approaches based on the backpropagation and radial basis neural networks have been published, but the most promising method should be based on the probabilistic or general regression neural network because of its ability of sufficient learning from the small learning datasets. Combination of this approach and the fuzzy modeling should provide better generalization ability for multidimensional nonlinear system. The general regression neuro-fuzzy model is not widely used or published because of its computational intensity. The solution for this issue is parallel computing which should be used as much as possible.

The main objectives:

- Design of neural and neuro-fuzzy model which model should be able to simulate a nonlinear system, with multidimensional output data of the electronic blocks or the electromagnetic compatibility measurements and with the data corrupted by noise. System model design with these requirements is usually intractable using the classical parametric or nonparametric approaches. Simple neural models are not precise enough or have problems with the multidimensionality of data and the lack of training data.
- Verification of the model on a real data, disussing the precision, reliability and generalization ability and comparison with other methods.
- Use of parallel computing: modification of existing algorithms to be run in multiple threads. The use of neural network for such system is computationally intensive. Especially, the probabilistic neural network training is time consuming, so the parallel computing should be used as much as possible.

2 GENERAL REGRESSION NEURO-FUZZY MODEL

2.1 GENERAL REGRESSION NEURAL NETWORK

The idea of general regression neural networks is based on the probability theory, especially Bayes strategy and Parzen windows, so the probability density function (PDF) of the training data is one of the crucial elements to be evaluated. Fundamental of this method is to reconstruct the underlying function given by the training samples. The function is assumed as joint and smooth. The best reconstruction with minimum variance is given by the conditional mean value [5]. The joint input-ouput distribution function is represented by following equation:

$$\hat{f}(x) = E\langle z(x) \,||\, x\rangle = \frac{\int_{-\infty}^{\infty} z(x)\, p(x,z)\, dz}{\int_{-\infty}^{\infty} p(x,z)\, dz} \tag{2.1}$$

Each training sample is represented by its own Gaussian distribution, which is centered at the corresponding training sample. The density function is the sum of Gaussian distributions over whole training dataset. The meaning of the parameters is shown in the figure 2.1. Kernel functions can be also reciprocal, rectangular or triangular, but the most commonly used is Gaussian because the advantages of normal distribution [**?**].

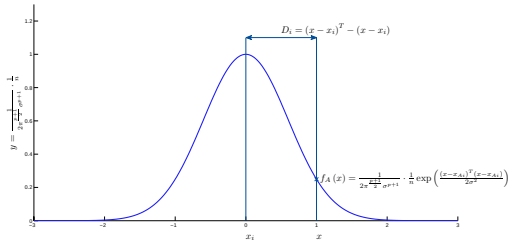


Fig. 2.1: GRNN probability density function

D_i is the distance between the training sample and the modeled value. This is evaluated for every sample of the training dataset. When the distance is computed and evaluated using the Gaussian function, $f_A(x)$, the sum over whole training dataset provides the resultant output, y. The detailed structure of network is illustrated by 2.4. Each training sample x_i represents one mean of a distribution, the normal distribution in this case. The Gaussian function is computed in the first hidden layer, a pattern layer, after the distance evaluation. The D_i defines how much the position of the modeled sample x corresponds with the training sample x_i. If D_i is zero, the equation $\exp\left(-\frac{D_i^2}{\sigma^2}\right)$ is one, that means the point being modeled is exactly represented by this training sample. Before the training data are put on the network inputs, they are often normalized fit the range $\langle 0,1\rangle$ or $\langle -0.5, 0.5\rangle$. The distance is evaluated as Euclidean distance. In [4],[5] , the model is treated as SISO, so the distance is in one-dimensional space:

$$D^2(k) = \sum_{i=1}^{n} \left(\frac{x(k) - X_i(k)}{\sigma(k)} \right)^2, \tag{2.2}$$

where X_i is the current training input, x is the point being modeled, and σ is the smoothness parameter. In [5], the standard deviation, σ is called a smoothness parameter. The higher is σ the wider is possible representation of the modeled point by the training sample. An

optimization of the smoothness parameter is usually done. The narrow shape of Gaussian curve results in overfitting to the training samples, too wide curve leads to oversmoothing of the function.

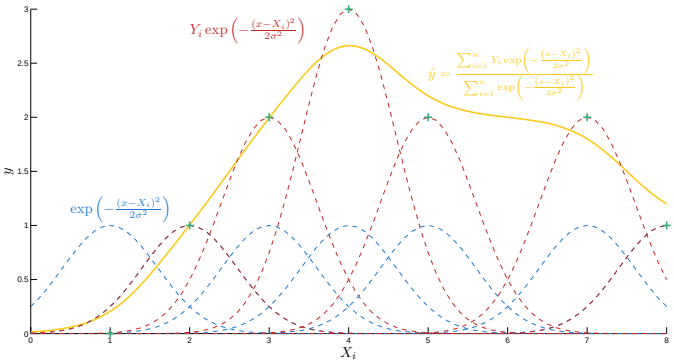


Fig. 2.2: Signal pass through the GRNN

Signals going from the first hidden layer to the second one (the summation layer) are passed directly to the first neuron and weighted with the corresponding values of training outputs, Y_i. In these two neurons, their input values are summed to get two scalar numbers, a numerator and a denominator for the output value.

$$\hat{y} = \frac{\sum_{i=1}^{n} Y_i \exp\left(-\frac{(x-X_i)^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{(x-X_i)^2}{2\sigma^2}\right)} \tag{2.3}$$

Training of this network consists of tuning the widths of kernel using any of statistical or evolutionary methods. The resultant outputs are independent on the initial conditions and training procedure; in the case of the same training data they remain the same as in previous attempts. The main drawback of this method is on the other hand of the advantage: good generalization ability due to the smoothing causes that the highest modeled value is always smaller than the trained one and vice versa: the smallest modeled value is always higher than the trained one. The method presented in this chapter is intended only for SISO model, so it has to be changed and extended for the MISO or MIMO system or the system with multimensional inputs and/or outputs. This is discussed in a chapter 2.3.

2.2 SIGMA DETERMINATION

The smoothness parameter determines the width and slope of the neurons functions and this is the only parameter to be adapted. The other ones are given by the training patterns. The mean squared error and the smoothness are two contradictory parameters which can be defined empirically, but their optimization should be performed to obtain better results.

When the value σ is too high, the generalization ability is too high and the MSE between the training data and the estimate of underlying function is significant, as illustrated with the red line in 2.3. The higher values of σ are useful when the data is noisy or when it contains several significantly outstanding values. These are omitted successfully. However, the value of

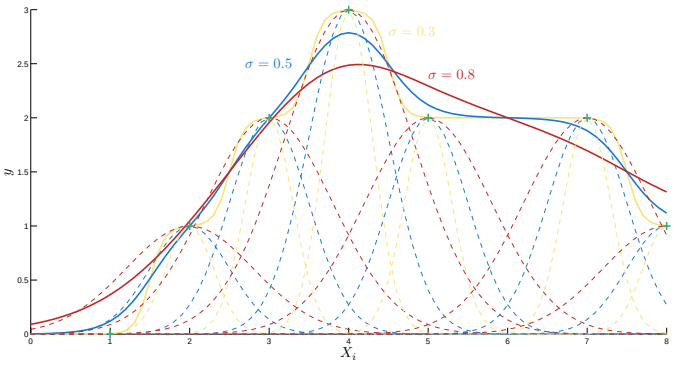


Fig. 2.3: Dependence of generalization ability on the smoothness parameter

σ should not usually exceed 1, because then the abilities of function approximation are lost. The evaluation of equation 2.3 for $\sigma = \infty$ is as follows:

$$\lim_{\sigma\to\infty} \frac{\sum_{i=1}^{n} Y_i \exp\left(-\frac{(x-X_i)^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{(x-X_i)^2}{2\sigma^2}\right)} = \frac{Y_i \lim_{\sigma\to\infty} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)}{\lim_{\sigma\to 0} \exp\left(-\frac{D_i^2}{2\sigma^2}\right)} = \frac{\sum_{i=1}^{n} Y_i}{n} \tag{2.4}$$

The resultant expression is the quotient of the sum of all the training output values and the number of training values, which represents the average value of the training samples.

If the value of σ is too small, the output function fits the training outputs exactly and the influence of neighboring functions is restrained. It cannot be considered as the underlying function of input training data, because there is a lack of generalization and approximation (yellow line in 2.3). The false inflection points can occur and devalue the resultant underlying function.

For the σ value determination in GRNN, the holdout method is usually used because of its simplicity. The training patterns are divided into two groups, $\frac{1}{3}$ of the training dataset is set as testing and the rest is the training data. After the network training, the MSE is computed using the testing data and kept. This process is repeated for a given number of passes with different division of the dataset (with less training data than in the previous run). Whole process is repeated for many different values of σ. The run with the smallest overall MSE value is picked and its σ is used for the whole network.

2.3 GENERAL REGRESSION NEURO-FUZZY NETWORK FOR MIMO MODEL

The methods mentioned above have to be modified and generalized to be used with the MIMO or MISO model. The main weakness of the GRNN model published in [4] and most of the GRNN articles is the presumption of single input and single output. This approach brings more general algorithm based on simple mathematical equations, which can be suitably interpreted to form the desired model. The block diagram of modified GRNN is shown in the figure 2.4. The red neuron layers represent the neural network part and the black ones represent the fuzzy modification. The pattern neurons from GRNN remain almost the same, but the equation 2.2 has to be generalized to perform the evaluation of n-dimensional inputs. The distance is

computed using the equation for the Euclidean distance [**?**]:

$$D_i = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{(X_{ij}-x_j)^2}{2\sigma_j^2}} = \sqrt{\sum_{i=1}^{n}\left[\frac{(X_{i1}-x_1)^2}{2\sigma_1^2}+\frac{(X_{i2}-x_2)^2}{2\sigma_2^2}+\ldots+\frac{(X_{in}-x_n)^2}{2\sigma_n^2}\right]} \tag{2.5}$$

where n is the total number of training sets, m is the number of the elements in each set, X_{ij} is the actual input being modeled. The output of the pattern layer, y_{i1}, is fuzzified in the next layer:

$$y_{i1} = \exp\left(-D_i^2\right), \tag{2.6}$$

y_{i1} stands for i^{th} output of first layer and σ is the smoothness parameter.

$$\overline{y_{i2}} = y_{i1}(1), y_{i1}(2), \ldots, y_{i1}(m), \tag{2.7}$$

where y_{i2} indicates i^{th} output of second layer, m is the number of fuzzy rules. This number is always odd, usually 3 or 5. The number and parameters of these rules are discussed below. The third layer goes back to the GRNN idea and provides the summation of the previous layer outputs in both neurons of these layer. In addition, one of these two neurons multiplies the signal by the training output corresponding to the actual training input, Y_i. The output of this layer, y_{3a}, is the numerator for the last layer and the other one, y_{3b}, is the denominator. The resultant ratio y is also the output of whole network and the estimate of resultant point.

$$\begin{aligned} y_{3a} &= \sum_{i=1}^{n}\sum_{j=1}^{m} y_{i2j} \times Y_i, \\ y_{3b} &= \sum_{i=1}^{n}\sum_{j=1}^{m} y_{i2j}, \end{aligned} \tag{2.8}$$

where j goes through the fuzzy rules of i^{th} training pattern and the results of all the partial results from the training patterns are summed up to get a scalar value.

$$y = \frac{y_{3a}}{y_{3b}}. \tag{2.9}$$

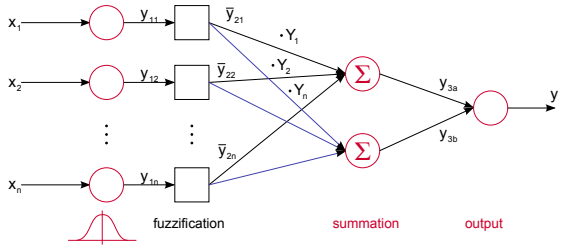


Fig. 2.4: Fuzzy GRNN block diagram

The fuzzy rule in the second layer is important part of the solution. The signal coming from the first layer is given by n scalar values, where n corresponds with the number of training patterns. The output of this layer, $\overline{y_{i2}}$, is a vector of m elements, where m corresponds with the number of the fuzzy rules. The circle seems to be the most suitable pattern for generating the fuzzy rules around the training pattern. As shown in the figure 2.5, the height

of the individual functions, k, distance between the concentric circles, on which the fuzzy-rule centers are located, r, and a number of the rules on each circle have to be determined.

The first one can be either given empirically, but better results are obtained, when the heights are given by any optimization method. The k value should be the same for all the rules of same r . Other possibilities are not tested within the scope of this thesis. The distance between the circles can be derived from the value of σ, they should correspond and they can be optimized in the case of inappropriate results. In this case, the number of fuzzy rules on the individual circles was determined to be 8. The fuzzy-rule centers are computed using following equations:

$$x = a + r \cdot \cos\theta, \tag{2.10}$$

$$y = b + r \cdot \sin\theta, \tag{2.11}$$

where x and y are the center coordinates, a and b are the shift parameters (mostly both zero) and θ is the angle (in radians) determined by the number of rules on the circle ($\theta = \frac{2\pi}{m}$). For more dimensions, these equations are replaced by the equations for a sphere or a hypersphere [9]:

$$\begin{aligned}
x_1 &= r\cos\theta_1 \\
x_2 &= r\sin\theta_1\cos\theta_2 \\
x_3 &= r\sin\theta_1\sin\theta_2\cos\theta_3 \\
&\vdots \\
x_{n-1} &= r\sin\theta_1\ldots\sin\theta_{n-2}\cos\theta_{n-1} \\
x_n &= r\sin\theta_1\ldots\sin\theta_{n-2}\sin\theta_{n-1}
\end{aligned}$$

The higher the number of concentric circles is, the complexity of fuzzy rules number determination grows because of the increasing radius. The length of the circle grows with 2π multiplication of r, so the number of rules has to be higher in the outer layers to prevent from the lack of intersections between the rules. This topic is an opportunity for further discussion and analysis of the nonsymmetric rule base. In this thesis, the maximum of two circles is used, because the higher number causes too smooth underlying function and evokes the idea of decreasing sigma, but it does not provide satisfactory results. The parameters optimization is performed using the well-proved mean squared error method (MSE):

$$MSE(y) = E\left[\left(\overline{Y_i} - y_i\right)^2\right], \tag{2.12}$$

where Y_i is a training sample and y_i is corresponding estimated value.

In this case, the fuzzy rule is defined as:

$$\begin{aligned}
R_1 &: IF\,\mu_1 = A_{11}\,AND\ldots AND\,\mu_n = A_{1n}\,THEN\,y = f_1(\mu_1,\mu_2,\ldots,\mu_n) \\
R_2 &: IF\,\mu_2 = A_{21}\,AND\ldots AND\,\mu_n = A_{2n}\,THEN\,y = f_2(\mu_1,\mu_2,\ldots,\mu_n) \\
R_3 &: IF\,\mu_3 = A_{31}\,AND\ldots AND\,\mu_n = A_{3n}\,THEN\,y = f_3(\mu_1,\mu_2,\ldots,\mu_n)
\end{aligned} \tag{2.13}$$

A_{11} to A_{31} are three fuzzy rules symmetrically distributed around each training pattern as shown in the figure 2.4. This approach provides more degrees of freedom due to the parameters to be optimized: the smoothness parameter σ, the height of the Gaussian curve, k, and the distance between the individual fuzzy rules. The figure 2.4 shows the fuzzy rules with the

centers organized in three concentric circles. The inner (green) one corresponds with the current training pattern, the others are defined to extend the possibilities of the neural network. The blue line in the left figure presents a cut though the functions which is illustrated in the figure on the right. The figure 2.5 shows the possible pattern function used for each training input.

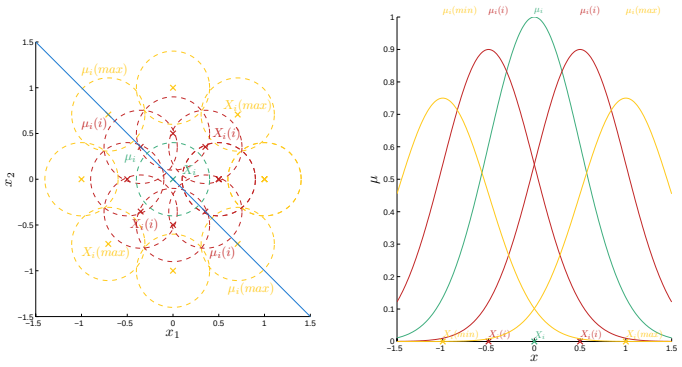


Fig. 2.5: Fuzzification of the neuron functions

The method mentioned above is suitable for the SISO or MISO system, but in many measurements, multiple outputs are obtained, for example a modulus and an argument. The presented network is able to evaluate multiple outputs with only small modification. The summation layer numerator input in 2.4, y_{i3a}, is computed using all the training inputs and outputs:

$$y_{3ak} = \sum_{i=1}^{n} \sum_{i=1}^{m} y_{i2j} \times Y_{ik} \tag{2.14}$$

analogously, the individual outputs are evaluated in modified equation 2.9:

$$y_k = \frac{y_{3ak}}{y_{3b}}. \tag{2.15}$$

The evaluation of these two equations, 2.14 and 2.15, is performed over all the training inputs for every training output. In principle, that can be interpreted as an own FGRNN network for every output.

2.4 SUMMARY

The general regression neuro-fuzzy network is aimed for a general MIMO system modeling. The original contribution of this approach is in the extension of currently used networks with the fuzzy rule and defining the fuzzy-GRNN for multiple inputs and multiple outputs. The use of fuzzy rules causes the use of more degrees of freedom, while the precisely set parameters of neural network guarantee the precision. The general regression neural network is one of memory-based networks, so it is suitable for batch-mode training. This is very useful in real-time modeling. The memory-based networks are also useful when the training dataset is poor and the modeled dataset is significantly larger. The distance between the training and modeled data is evaluated using the Euclidean distance, because of the simplicity and effectivity of this method. For N inputs, the distance is computed in N-dimensional space. For M outputs, M networks have to be trained and then the outputs can be merged. The sigma parameter was determined using the holdout method, where the dataset is divided into training and testing parts multiple times. Each time the training part is smaller and testing part larger. After the

network training, the MSE is computed. This is repeated for several values of sigma and the one with smallest MSE is picked to be used for the network. The main weakness of former use of fuzzy GRNN was presumption of single input, single output system and the demonstration on simple and easily predictable functions. The network presented in this thesis is able to solve multiple inputs, multiple outputs issue and the results are acceptable even for complex functions. The output data of the network always fall into the range of the training data, so the maximum value of the network output can not be higher than the maximum value of the training dataset. Analogously, the minimum output value can not be less than the minimum training value. This is advantageous, when there are some unwanted outliers in the training or modeling data. A normalization is not necessary, but it can bring clarity into the data during the programming.

3 PARALLEL APPROACH

The use of parallel computing is one of the main objectives of this thesis. The main idea is targeted to modeling of complex systems with multiple or multidimensional inputs and outputs, so a large dataset is assumed. The algorithms is designed to be run in parallel. A sequential approach is also possible, but then the problem with computational capabilities arises. All the algorithms designed as parallel can be theoretically converted into sequential, but it is not always possible to do it conversely. Many algorithms are of iterational nature, so they are inconvertible to parallel.

In the figure 3.1, the parallel part of the algorithm is highlighted by red rectangle. All the Euclidean distances and their exponentials can be evaluated in multiple threads. This approach is advantageous in case of batch learning and when the amount of training data is of the tens to thousands order. The training algorithm is considered as parallel, but it can be run also in sequential form for small amount of data.

The simplest and most easily accessible way to implement the algorithm in parallel, is the multicore computation on a local computer. Data parallelism is used for distributing data across multiple computing nodes. This single process, multiple data method causes that each processor gets the different part of the data and performs the same instructions as the others. In this case, the Euclidean distance, the Gaussian function and the fuzzification are performed simultaneously in independent threads for different data.

The fuzzy GRNN was implemented to be run in parallel on the multicore cluster in Matlab with shared memory. The training data was placed in the shared memory area (as a distributed variable on the server) and the process was performed on the individual processors. After releasing the resultant data from the server, the rest of calculations (summation and division) was done sequentially on a local computer.

3.1 PARALLEL COMPUTING IN MATLAB

The parallel computing in Matlab is based on data parallelism, which is focused on the distributed nature of data. This single program, multiple data approach is executed on distributed memory computed architecture consisting of multiple independent computing nodes. Each of these nodes can cooperate with other nodes via passing and receiving routines for messages. In this case, serial parts of the program are computed on the local computer, while the parallel ones are sent to the cluster machine using a synchronous message passing system, so the sender has to wait for receiver to transfer the message, and vice versa. The cluster machine is a shared-memory machine, consisting of multiple CPUs with the access to the shared memory space, so the messages are passed by sending them to the shared memory.[10]

3.2 COMPUTING ON THE GRAPHICS PROCESSING UNIT

For this neural network, the GPU is very useful solution. Each block representing the Euclidean distance, Gaussian function and fuzzification can be evaluated using one kernel in a texture. Kernel can be thought as the body of loop.

The advantage of computing using floating-point operations is natural for the GPU. Graphic card is intended for highly parallel and computationally intensive computation to be able to do the graphic rendering reliably and promptly. Therefore, more transistors are intended for data processing than data caching and flow control. Like in other data-parallel computation schemes, the same program is computed in multiple threads using different data with high ratio of arithmetic operations to memory operations. The program is partitioned into subprograms which can be solved independently, each of them in its own thread. These subproblems

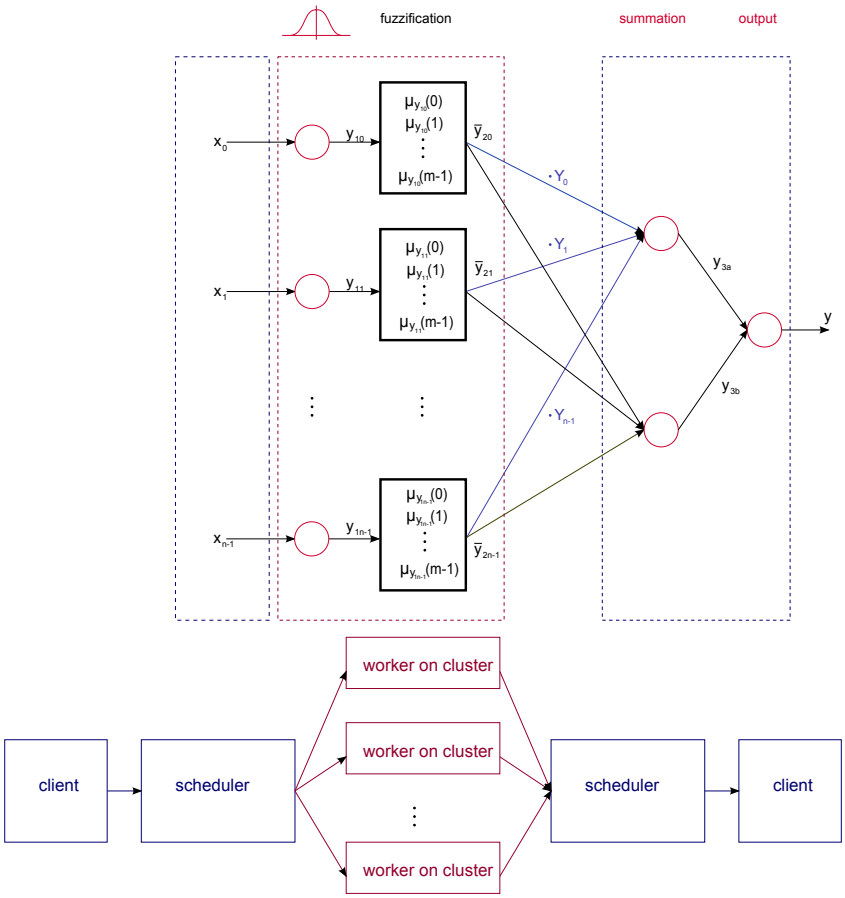


Fig. 3.1: 1D thread blocks of GRNN distributed in a 1D grid

then form a less fine subproblem which is solved by means of the block of threads. Each block of threads is schedules on one of the GPU multiprocessors [8].

The grid model can be suitably structured in accordance to corresponding algorithm. When the evaluated data are one-, two- or three-dimensional, the grid of blocks can imitate the structure of data. Then the grid looks like a vector, matrix or volume of threads, respectively.

In CUDA, there are special functions, called kernels. When, called, kernel is run N times in parallel in N threads with different data. General syntax of kernel definition and call is illustrated in listing 3.1. The kernel is defined using the ___global___ declaration (listing 3.1, line 4). The function call (3.1, line 19) specifies the number of threads. Each thread is labeled using a unique thread ID, which is accessible within kernel using threadIdx variable. [?] [?] [?]

Listing 3.1: Array of Pointers to Strings

```
1 #include <stdio.h>
2 ...
3
4 __global__ void fuzzy_distance (float x, float Xi, float sigma,
    int n_fuzzy) //modeled pattern, i-th trained pattern, kernel
    width, number of fuzzy functions
5 {
```

```
    ...
    //function for evaluation of first two hidden layers:
       fuzzification and distance
}

int main()
{
    //inputs initialization
    ...

        dim3 num_blocks (1, n) //number of blocks in the grid; n
           corresponds to the number of trained patterns
        dim3 threads_per_block (1, n_fuzzy) //number of threads
           in each blocks

        //kernel invocation; grid of num_blocks blocks by
           threads_per_block threads
        fuzzy_distance <<< num_blocks, threads_per_block >>> (x,
           Xi, sigma, n_fuzzy);

        ...
}
```

3.3 SUMMARY

The parallel approach enables the algorithm to be run fast and, in the case of large data, it is necessary for the computation. When the amount of the data is large, there are problems with the placing the data into the memory and also the computational time can rise that the computation becomes impossible. The fuzzy GRNN is designed to have the first three layers run in parallel. The sequential run of the same algorithm is always possible, but it is limited to small datasets. The parallelism is also suitable for the use in real time applications, as the computational time decreases. These applications require to get the results as soon as possible, so the ideal delay between stimulus and result should be zero. The proper use of parallel method and the memory-based network can reduce the computational time to minimum (of the milliseconds order). In this section, two different approaches are presented: the message passing approach provided by Matlab functions and the GPU computing using CUDA. The performance and computational time of these implementations cannot be compared properly, because each of the approaches was executed on different computer. Generally, the CUDA approach is more useful for larger datasets than the message passing approach provided by Matlab, because of the availability of more multiprocessors.

4 VERIFICATION

4.1 DATA ACHIEVEMENT

The data necessary for network training are obtained by the automated measurement. The miniaturized model of aircraft (EV 55) is illuminated by electromagnetic wave having various incident angles (0° to 360°, step 15°) and frequencies (1 to 2 GHz, step 5 MHz). Measurements of a 'shielding effectiveness' given by the electric field intensity measured using the probe inside the aircraft fuselage are used for neural network training. Assuming the nonlinearity of the measurement in the hand, the neural network is able to estimate the transfer function of aircraft fuselage for chosen incident angle and to suppress noise in the training data. [31]

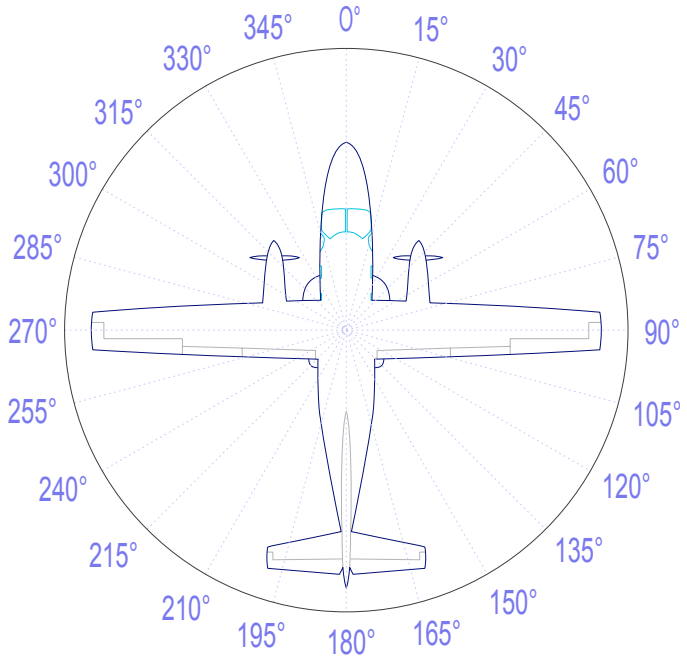


Fig. 4.1: Measurement on the miniaturized model of aircraft EV 55.

4.2 VALIDATION

Since no independent dataset is available, there is no possibility to provide the external validation. Thus the validation of the predictive ability of the model is performed using the cross validation. The measured data is divided into two parts: training and validation data. This is performed several times with different sizes of training and validation datasets. The training performance is evaluated using the root mean squared error (RMSE) [11].

The RMSE values are listed in the table below. These error values are computed for the normalized output values, so the value of 1 would correspond to 100% error. The error values are relatively high, this is caused by the multidimensionality of the solved issue. If the error values comparable along multiple dimensions are desired, they have to be divided by the number of dimensions. When the training and testing dataset are of the same size, the network error for GRNN is nearly 13%, while for MLP, it reaches 32%. When the training dataset size grows, the GRNN error significantly decreases below 10%, while the MLP error decreases slightly to 22% for the training dataset fifty times larger than testing dataset.

ratio RMSE	**1:2**	**1:5**	**1:10**	**1:20**	**1:50**
GRNN	0.1298	0.0658	0.0804	0.0620	0.0576
MLP	.3235	.3414	.3166	.2683	0.2240

4.3 RESULTS

The network training is done using non-uniformly distributed data randomly chosen from the measured dataset and the testing dataset is two times larger than the training dataset. Both of these conditions should cause problems for the multilayer perceptron. The training dataset has two inputs and one output given by measurement. The presence of noise is expected, it can be thought as white noise. The normalization of input data is necessary only for the multilayer perceptron, but it is better to do it also for FGRNN, not for the evaluation itself, but rather for clarity from the programmer view. When the maximum value corresponds to one, the precision of evaluation sometimes can be better.

The fuzzy-GRNN is compared with the MLP with Levenberg-Marquardt algorithm [6]. As expected, the training set of 4824 patterns (24 angles for each of 201 frequencies) is too small for sufficient MLP training and modeling of 359 angles for 201 frequencies. This is caused by gradient nature of the method, which is vulnerable by falling into local minima (it is partially treated by the use of the Levenberg-Marquardt algorithm [6]). But the main disadvantage is, that the training data for the multilayer perceptron is almost never large enough.

The smoothness parameter was set using the holdout method (section 2.2) to 0.68*maximum value of the output. There was higher mean squared error sum for other parameter values, although individual local part could have smaller value of error.

The MLP cannot be used for this case, while the FGRNN has no problems with the regression. It is caused by the optimization-based nature of multilayer perceptron and memory-based and probability nature of GRNN supplemented by the fuzzy rules. The general regression neural network itself should have acceptable results, but the fuzzy rules bring more of generalization ability, so the network is more resistant to outliers and noise. The use of GRNN guarantees the precision.

Figures 4.2 and 4.3 show the resultant output in comparison to original measured data for multilayer perceptron and fuzzy-GRNN, respectively. Due to the use of fuzzy rules, the network has better generalization ability. The use of general regression network causes better precision and generalization ability then for the multilayer perceptron when the training dataset is relatively small. The figure 4.2 shows that the multilayer perceptron had problems with learning. Some of the training patterns are exactly copied while other ones are completely diverse and there are no generalization and regression abilities. The FGRNN has good both generalization and regression abilities as shown in the figure 4.3.

The two network models are compared using the radiation pattern from figure 4.2 and 4.3. In both figures, the 'x' marks represent the testing data which are obtained from measurement and different from training data. The range of values corresponds with the training dataset. Blue curve represents the network output data from the multilayer perceptron. This network usually has good regression ability and poor generalization ability, but in this case, the regression ability is negatively affected by the size of training dataset.

The figure 4.3 shows the same items for the fuzzy general regression neural network. Since the GRNN is closer to the probabilistic and memory based ones, the outputs of this network are completely different from the multilayer perceptron model outputs. The curve is smoother and it omits the extreme values, so the generalization ability is balanced with the regression.

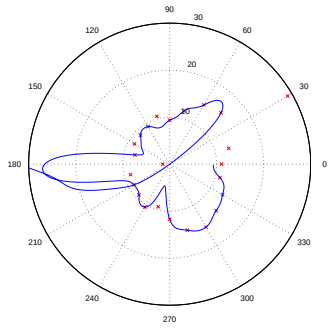


Fig. 4.2: Comparison of measured data and network output computed using MLP

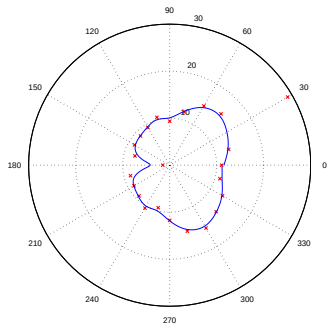


Fig. 4.3: Comparison of measured data and network output computed using FGRNN

This property would be useful when suppressing noise, because the outliers are suppressed, but the shape is very close to the real shape of the curve.

The network output value is defined by the most probable occurrence of the dependent variable value (the outputs of neural network) with respect to the independent variable value (inputs of neural network). On the other hand, the network model does not omit the steep variations in the modeled function, so the precision is better than the precision of the MLP model. For the MLP approach, the antinomy of the two requirements, the precision and generalization, significant. For the GRNN, the requirements are better balanced.

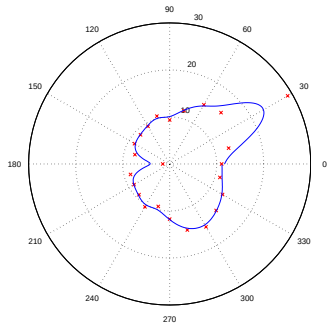


Fig. 4.4: Comparison of measured data with a significant ¨outlier and network output computed using FGRNN

The figure 4.4 shows neural network output for the same data as figure 4.3, but one of the training dataset elements is modified to be a significant outlier. The resultant curve is closer to real data without the sporadic incorrect values.

4.4 SUMMARY

The network was trained and tested using the data obtained from the automated measurement on the miniaturized EV55 aircraft model. Since no independent data are available from this measurement, the validation is performed five times using the cross-validation with different training and testing dataset sizes. The FGRNN results are compared to the results obtained from the MLP with the Levenberg-Marquardt and backpropagation algorithms. The multilayer neural network training is supported by genetic algorithm, which provides better training and avoids falling into the local minima. The FGRNN is trained only once, because the training re-run does not bring different results.

The performance of both networks is compared using root mean square error. When the training and testing dataset are of the same size, the network error for GRNN is nearly 13%, while for MLP, it reaches 32%. When the training dataset size grows, the GRNN error significantly decreases below 10%, while the MLP error decreases slightly to 22% for the training dataset fifty times larger than testing dataset. The better results of FGRNN are given mainly by the precision of designed network, which is in contradiction with generalization ability. Elimination of significant outliers results in higher RMSE, although it is desired.

5 CONCLUSION

The overview of widely used system identification and modeling methods is presented. Classical approaches based on spectral and correlation analysis or the parameter estimation are insufficient for the behavioral modeling of the aircraft equipment because they are not supposed to solve an unknown nonlinear system. The most suitable approach is the neuro-fuzzy memory-based which is designed to behave as an input/output model, or a black box model. The training dataset for the network learning is acquired by measurements of real inputs and outputs of the system. The model should be able to process generally multidimensional data corrupted by noise with good generalization and high precision.

The first of the main objectives of the thesis was to design the neural network which is able to compute the model of general nonlinear system. The resultant approach uses the general regression neural network (GRNN) with additional fuzzy rules. This kind of network is suitable for modeling because of its sufficient generalization ability and precision. The originality of this approach consists of the use of this method for the MIMO model, which requires multidimensional approach to this network. Previously published methods were designed only for SISO model, so the neuro-fuzzy networks were not able to deal with more than one dimension of data. In addition, the most frequently used type of neural network was the multilayer perceptron, which is popular because of its simplicity, but it is suitable for modeling only with relatively large training dataset. More sufficient type of neural networks for this issue is the memory-based neural network, for example the GRNN. Detalied derivation of the multidimensional approach of this network, determination of the fuzzy rules and computation of the smoothness parameter using the holdout method, are presented in chapter 2.1.

The network is designed to be run in parallel. The algorithm can be converted to sequential, but it would fail in case of larger datasets, because of the lack of memory or it would become incomputable because of the long evaluation time. Thus the sequential run of this algorithm is recommended only for the small datasets of up to tens of the training patterns. The fuzzification layer and the hidden layer are evaluated in parallel. The decision layer and the summation layer are evaluated sequentially, because parallel approach is impossible. Running the algorithm in multiple threads also allows the usage of the method for the batch learning in real time applications. The two approaches: the message passing in Matlab and the GPU using CUDA are discussed in chapter 3. These two methods are available for the majority of today's computers. They cannot be compared properly, because each of the approaches was executed on different computers.

The third main part of this thesis, the verification of the model, is performed using the data obtained from the automated measurement on the miniaturized EV55 aircraft model, illuminated by electromagnetic wave having various incident angles and frequencies. Measurements of a 'shielding effectiveness' given by the electric field intensity measured using the probe inside the aircraft fuselage are used for neural network training. In total, there are 4824 values: 24 angles by 201 frequencies. The cross-validation is performed five times, the testing and training data had five different ratios: 1:2, 1:5, 1:10, 1:20 and 1:50. The cross-validation of the model is chosen, because no independent data are available for the verification.

The result obtained from FGRNN is compared with the one from optimized MLP with backpropagation learning algorithm with Levenberg-Marquardt method. The root mean squared error of the MLP decreases significantly with large training dataset, the FGRNN error is smaller and decreases slightly with increasing number of training data. The better results of FGRNN are given mainly by the precision of designed memory-based network, which is in contradiction with generalization ability. Elimination of significant outliers results in higher RMSE, although it is desired.

BIBLIOGRAPHY

[1] LJUNG, L. *System identification: Theory for the user.* Prentice Hall, London, 1999.

[2] NELLES, O. *Nonlinear system identification: From classical approaches to neural network and fuzzy models.* Springer, Berlin, 2001.

[3] SÖDERSTRÖM, T; STOICA, P. *System identification.* Prentice Hall, Uppsala, 2001.

[4] BODYANSKIY, Y.; TESLENKO, N. *General regression neuro-fuzzy network for identifiaction of nonstationary plants.* In International Journal "Information Technologies and Knowledge" Vol.2 / 2008. p. 136-142.

[5] SPECHT, D. *A General Regression Neural Network*IEEE Transactions on Neural Networks, vol. 2, p. 568-576, 1991.

[6] HAYKIN, S. *Neural networks: A comprehensive foundation.* Pearson Education, Ontario, 1999.

[7] GUPTA, M.; JIN, L.; HOMMA, N. *Static and dynamic neural networks: from fundamentals to advanced theory.* Wiley, New Jersey, 2003.

[8] NVIDIA Corp. *NVIDIA CUDA C Programming guide.*NVIDIA Corporation, 2006.

[9] WEEKS, J. R. *The Shape of Space: how to visualize surfaces and three-dimensional manifolds. 2nd edition* Marcel Dekker, Inc., New York, 2002.

[10] MathWorks, Inc.*Parallel Computing Toolbox User's Guide*, Natick, 2011.

[11] JOERGENSEN, B. GOEGEBEUR, Y. Prediction and validation, http://statmaster.sdu.dk/courses/ST02, [online]. [cit. 10. 8. 2012]

[12] CANETE, J. F.; GONZALEZ-PEREZ, S.; SAZ-OROSCO, P. *Software tools for system identification and control using neural networks in process engineering.* World Academy of Science, Engineering and Technology 47, 2008. p 59-63.

[13] KIM, I.; FOK, S.; FREGENE, K.; LEE, D.; OH, T.; WANG, D. W. L. *Neural network-based system identification and controller synthesis for an industrial sewing machine.* International Journal of Control, Automation, and Systems Vol. 2, No. 1, March 2004. p. 83-91.

[14] FEKIH, A. *Neural networks based system identification techniques for model based faultdetection of nonlinear systems.* International Journal of Innovative Computing, Information and Control Volume 3, Number 5, October 2007. p. 1073-1085.

[15] BABUŠKA, R.; VERBRUGGEN, H. *Neuro-fuzzy methods for nonlinear system identification.* Annual Reviews in Control 27, 2003. p. 73–85.

[16] SENG, T. L.; KHALID, M.; YUSOF, R.; OMATU, S. *Adaptive neuro-fuzzy control system by RBF and GRNN neural networks.* Journal of Intelligent and Robotic Systems Vol. 23, Kluwer Academic, 1998. p. 267-289.

[17] NOSSAIR, Z. B.; MADKOUR, A. A.; AWADALLA, M. A.; ABDULHADY, M. M. *System identification using intelligent algorithms.* 13th International Conference on AEROSPACE SCIENCES & AVIATION TECHNOLOGY, ASAT- 13, May 26 – 28, 2009.

[18] NOCEDAL, J.; WRIGHT, S. J. *Numerical optimization.* Springer, New York, 1999.

[19] ANTONIOU, A.; LU, W. *Practical optimization: Algorithms and engineering applications.* Springer, New York, 2007.

[20] HAATAJA, K. *Sumean logiikan ja neuroverkkojen työkaluselvitys.* VTT Elektroniikka, Espoo 1998.

[21] BABUŠKA, R., VERBRUGGEN, H. *Neuro-fuzzy methods for nonlinear system identification*Annual Reviews in Control 27, 2003.

[22] JOELIANTO, E., RAHMAT, B. *Adaptive Neuro Fuzzy Inference System (ANFIS) with Error Backpropagation Algorithm using Mapping Function*International journal of artificial intelligence A08, 2008.

[23] SENG, T., MARZUKI, K., RUBIAH, Y., OMATU, S. *Adaptive neuro-fuzzy control system by RBF and GRNN neural networks*Journal of Intelligent and robotic systems, vol. 23, pp 267-289, 1998.

[24] JIMENEZ, J., GIRON-SIERRA, J., INSAURRALDE, C., SEMINARIO, M. *A simulation of aircraft fuel management system*Simulation modelling practice and theory 15, 2007.

[25] BAUER, M. *General Regression Neural Network, GRNN- A Neural Network for Technical Use*Diploma thesis, University of Wisconsin-Madison

[26] ŠTEFAN, M. *Identifkace nelineárních systémů pomocí neurofuzzy modelů*Bulletin of Applied Mechanics 2, p.121-131 (2005).

[27] HAYASHI, K., KAWADA, K., YAMAMOTO, T. *Evolutionary modeling of a process system*ICROS-SICE International Joint Conference 2009, Fukuoka, Japan, 2009.

[28] BOYEON, K., PARK, K. *A study on the modeling of nonlinear system using genetic programming*18th annual conference of the IEEE Engineering in medicine and biology society, Amsterdam, 1996.

[29] FRÝZA, T.; SVOBODOVÁ, J.; ADAMEC, F.; MARŠÁLEK, R.; PROKOPEC, J. *Overview of Parallel Platforms for Common High Performance Computing.* Radioengineering, 2012.

[30] FRÝZA, T.; SVOBODOVÁ, J.; MARŠÁLEK, R.; PROKOPEC, J. *Multiplatform Approaches and Tools for Parallel Computing in Signal Processing Domain.* In Proceedings of The Seventh International Conference on Digital Telecommunications ICDT2012. 1. Chamonix, France: IARIA, 2012.

[31] SVOBODOVÁ, J.; KOUDELKA, V.; RAIDA, Z. *Aircraft equipment modeling using neural networks. In Proceedings of 2011 International Conference on Electromagnetics in Advanced Applications.* Torino, Italy: COREP, 2011.

[32] SVOBODOVÁ, J. *Parallel Computing in Microwave Structure Optimization.* In Metz, Francie: Supélec, Metz, 2010.

CURRICULUM VITAE

Personal	
Name	**Ing. Jitka Svobodová**
Born	November 30, 1984 in Olomouc
Address	1. máje 32, 783 35 Horka nad Moraovu, Czech Republic
Contact	svobodovic@gmail.com
Education	
2009 – 2012	Doctor of Philosophy (PhD) Brno University of Technology (Department of Radio Electronics)
2007 – 2009	Master's degree (MSc) – inženýr (Ing.) Brno University of Technology (Department of Biomedical Engineering) Thesis: Neural networks and evolutionary algorithms
2004 – 2007	Bachelor's degree (BSc) – bakalář (Bc.) Brno University of Technology (Department of Radio Electronics) Thesis: Influence of physical load to electrical activity of muscles
Internships	
08/2008 - 12/2008	Erasmus study stay University of Kuopio, Finland
Courses	
07/2010	International Travelling Summer School on Microwaves and Lightwaves Supélec, Metz (France)
02/2012	Winter School of Parallel Computing CINECA, Bologna (Italy)
07/2012	GPU Programming using CUDA HLRS, Stuttgart (Germany)