CORE
Metadata, citation and similar papers at core.ac.uk

Optimization of Self-Organizing Polynomial Neural Networks

Author:

Ivan Marić
Rudjer Boskovic Institute
Bijenicka 54
10000 Zagreb
Croatia

Email: ivan.maric@irb.hr
Phone: +385-1-4561191

Abstract

The main disadvantage of self-organizing polynomial neural networks (SOPNN) automatically structured and trained by the group method of data handling (GMDH) algorithm is a partial optimization of model weights as the GMDH algorithm optimizes only the weights of the topmost (output) node. In order to estimate to what extent the approximation accuracy of the obtained model can be improved the particle swarm optimization (PSO) has been used for the optimization of weights of all node-polynomials. Since the PSO is generally computationally expensive and time consuming a more efficient Levenberg-Marquardt (LM) algorithm is adapted for the optimization of the SOPNN. After it has been optimized by the LM algorithm the SOPNN outperformed the corresponding models based on artificial neural networks (ANN) and support vector method (SVM). The research is based on the meta-modeling of the thermodynamic effects in fluid flow measurements with time-constraints. The outstanding characteristics of the optimized SOPNN models are also demonstrated in learning the recurrence relations of multiple superimposed oscillations (MSO).

Keywords

Polynomial neural networks
GMDH
Levenberg-Marquardt algorithm
Particle swarm optimization
Time series modeling

1. Introduction

Approximation of complex multidimensional systems by SOPNN, also known as the GMDH polynomial neural networks (PNN), was introduced by Ivakhnenko, (1971). The SOPNN are constructed by combining the low order polynomials into multi layered polynomial structures where the coefficients of the low-order polynomials (generally 2-dimensional 2^{nd} order polynomials) are obtained by polynomial regression (Chapra & Canale, 1998) with the aim to minimize the approximation error. GMDH models may achieve reasonable approximation accuracy at low complexity and are simple to implement in digital computers (Maric & Ivek, 2011). The GMDH is resistant to over-fitting since it uses separate data sets for regression and for model selection. When applied to real time compensation of nonlinear behavior, the self-organizing nature of GMDH may eliminate the complicated structural modeling and parameterization, common to conventional modeling strategies (Iwasaki, Takei & Matsui, 2003).

The performance of the SOPNN is generally evaluated by a single parameter measure (Witten & Eibe, 2005), typically by the least square error, which minimizes the model approximation error rather than its complexity. When building the models for time-constrained applications the constraints can be efficiently embedded into the model selection metrics (Maric & Ivek, 2011). It was shown (Maric & Ivek, 2011) that the raw SOPNN models (GMDH PNN) are inferior to multilayer perceptron (MLP) when considering the accuracy with respect to the complexity. It was also concluded (Maric & Ivek, 2011) that SVM is not appropriate for building the low complexity models for time-constrained applications.

The SOPNN node-polynomial weights, after calculated by the regression, remain unchanged during the rest of the training process resulting in sub-optimal SOPNN models. The accuracy and the prediction of models may be improved significantly when trained by the genetic programming and back-propagation (BP). It was shown (Nikolaev & Iba, 2003) that the population-based search technique, relying on the genetic programming and the BP algorithm, enables to identify the networks with good training as well as generalization performances. The BP improves the accuracy of the model but it is known to often get stuck in local minima. The idea of this paper is to adapt a more robust procedure for the optimization of the SOPNN relation with respect to its weights.

The PSO is a nature inspired algorithm, which enables the optimization of model weights by simulating the flight of a bird flock (Eberhart & Kennedy, 1995). Since the PSO is simple to implement it has been used in our experiments for the estimation of the approximation abilities of raw SOPNN models. After the PSO improved significantly the approximation accuracy of various SOPNN models a more complex Levenberg-Marquardt (LM) algorithm (Levenberg, 1944; Marquardt, 1963) has been adapted for the optimization of model weights. Although widely used for the optimization of ANN, the use of the LM algorithm for the optimization of weights of the SOPNN to the best of my knowledge has not been reported in the literature. The LM algorithm converges many times faster than the PSO and increases the approximation accuracy of the SOPNN model substantially. This paper describes the adaptation of PSO and LM algorithm for the optimization of SOPNN and demonstrates how the approximation accuracy of the original GMDH model can be significantly improved after optimizing its weights.

In the following section, the GMDH, PSO and the LM algorithm are described. The PSO and the LM algorithm are adapted for the optimization of SOPNN weights. Section 3 describes the procedure for the estimation of the execution time for SOPNN and MLP in time constrained applications. A procedure for the compensation of thermodynamic effects in flow rate measurements is summarized in section 4 and the results of the simulations of the flow rate error compensation procedure by the surrogate models are given in section 5. Finally in section 6, the outstanding performances of the SOPNN are demonstrated on MSO task that has been widely studied in echo state networks (ESN) literature.

2. GMDH, PSO and LM algorithm

2.1. GMDH algorithm

The GMDH algorithm (Ivakhnenko, 1971) constructs the models by combining the low-order polynomials into multi layered polynomial networks. Fig. 1 illustrates a complete two-layer feed-forward SOPNN representing a 3-dimensional system, where $p_{\lambda,i}$ denotes a low-order and low-dimensional polynomial corresponding to the i^{th} node of the layer λ and x_i represents the i^{th} independent variable.

First order and second order two-dimensional polynomials are preferred as their cascading does not increase rapidly the order of overall polynomial relation and because they are fast to process. In this paper we will restrict our attention to a complete second-order two-dimensional polynomial

$$p_{\lambda i} = a_{\lambda i1} + a_{\lambda i2} z_{\lambda i1} + a_{\lambda i3} z_{\lambda i2} + a_{\lambda i4} z_{\lambda i1}^2 + a_{\lambda i5} z_{\lambda i2}^2 + a_{\lambda i6} z_{\lambda i1} z_{\lambda i2}\,, \tag{1}$$

where $z_{\lambda i1}$ and $z_{\lambda i2}$ represent the input variables and $a_{\lambda i1},...,a_{\lambda i6}$ are the corresponding weights (coefficients) obtained by the polynomial regression. Note that $z_{\lambda i1}$ and $z_{\lambda i2}$ can be any combination of two different variables from lower layers including the independent variables (x_i) and the derived regression polynomials ($p_{\lambda i}$). For example, the input variables for the polynomial $p_{2,6}$ from Fig. 1 are the polynomial $p_{1,1}$ from the first layer and the independent variable x_3.

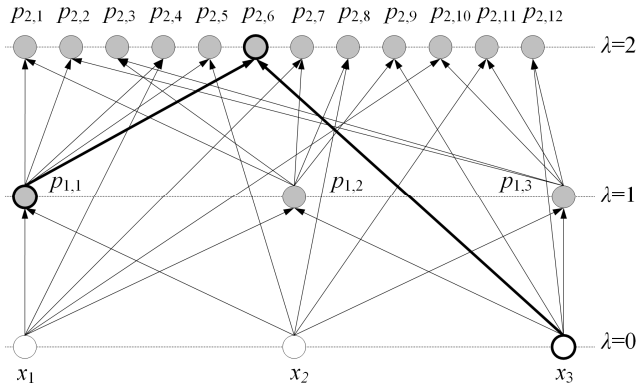


Fig. 1.

The GMDH algorithm assumes two independent data sets: a training set of M samples

$$\mathcal{D}_t = \{(\mathbf{x}_{ti}, y_{ti})\}_{i=1}^{M}\,, \tag{2}$$

where each sample consists of a data vector $\mathbf{x}_{ti} = (x_{ti1}, x_{ti2},..., x_{tiK})$, $x_{ti} \in \mathcal{R}^K$, and the corresponding dependent variable $y_{ti} \in \mathcal{R}$, as well a validation data set of N samples

$$\mathcal{D}_v = \{(\mathbf{x}_{vi}, y_{vi})\}_{i=1}^{N} \tag{3}$$

with data vector $\mathbf{x}_{vi} = (x_{vi1}, x_{vi2},..., x_{viK})$, $x_{vi} \in \mathcal{R}^K$, and the corresponding dependent variable $y_{vi} \in \mathcal{R}$. The algorithm uses the training data set (2) to fit the coefficients of the regression polynomial (1) and the validation set (3) to verify the approximation error of the polynomial. The polynomials are then ranked according to some predefined metrics (Maric & Ivek, 2011).

In order to calculate the coefficients, $a_{\lambda i1},...,a_{\lambda i6}$, in (1) by the polynomial regression, a set of 6 simultaneous linear equations

$$\left\{ \frac{\partial}{\partial a_{\lambda ik}} \sum_{m=1}^{M} \left(y_{tm} - p_{\lambda i}^{tm} \right)^2 = 0 \right\}_{k=1}^{6} \tag{4}$$

must be solved, where M is a total number of training samples, y_{tm} is the m^{th} sample value of the dependent variable from the training data set (2) and $p_{\lambda i}^{tm}$ is the value of the i^{th} polynomial at layer λ corresponding to the m^{th} data vector from the training data set (t). In our implementation of the GMDH algorithm the above set of linear equations is solved by the Gauss elimination method using forward elimination, back substitution, and pivoting (Chapra & Canale,1998).

As illustrated in Fig. 1. the total number of possible nodes in each layer is increasing rapidly by the increase of the layer number and can be easily calculated by the following simple iterative equation

$N_{\lambda+1} = \frac{N_\lambda \cdot (N_\lambda - 1)}{2} + N_\lambda \sum_{i=0}^{\lambda-1} N_i$, where λ=0,1,… denotes the corresponding layer, N_λ is the total number of nodes at layer λ and the second term in the equation equals zero for λ=0. To prevent the combinatorial explosion the maximum number of nodes retained per layer is generally limited. The nodes retained at lower layers are combined multiple times to produce the nodes at higher layers. To speedup the algorithm the retained polynomials may be tabulated.

2.2. Particle Swarm Optimization of SOPNN Weights

Let $\mathbf{a} \in \Re^N$ be the vector, $\{a_i\}$, i=1,...,N, in N-dimensional space. Let the SOPNN polynomial relation $P(\mathbf{x},\mathbf{a})$ be the particle whose position in the space is defined by the vector **a**. Particle swarm optimization (Eberhart & Kennedy, 1995) minimizes the nonlinear fitness function:

$$e(\mathbf{a}) = \sum_{k=1}^{K} [y_k - P(\mathbf{x}_k, \mathbf{a})]^2 \tag{5}$$

where K is the total number of training vectors, $\mathbf{x}_k$ denotes the k^{th} M-dimensional input training vector, y_k=$y(\mathbf{x}_k)$ is the corresponding training value and **a** denotes N-dimensional vector representing the coefficients of all nodes of the SOPNN polynomial $P(\mathbf{x}_k,\mathbf{a})$. Before starting the PSO the position **a** and the velocity **v** of each particle are initialized by:

$$a_{ki}^0 = L_i + r_{ki}|U_i - L_i|, k = 1,...,K, i = 1,...,N \tag{6}$$

and

$$v_{ki}^0 = (2s_{ki} - 1) \cdot |U_i - L_i|, k = 1,...,K, i = 1,...,N \tag{7}$$

where a_{ki}^0 and v_{ki}^0 are the corresponding i^{th} component of the k^{th} particle's initial position and velocity, L_i and U_i are the corresponding lower and upper boundaries for the i^{th} dimension of the search space, and r_{ki} and s_{ki} denote the random numbers from the range [0,1]. For each particle k, the best particle position A_k and the best fitness value E_k are initialized by setting them equal to the initial position and the initial fitness value of the particle, respectively. The pointer to the particle α having the best fitness value, E_α, of all particles is also initialized. According to (Shi & Eberhart, 1998) the particles are manipulated iteratively by using the following equations:

$$v_{ki}^{j+1} = w \cdot v_{ki}^j + c_1 \cdot R_{ki} \cdot (A_{ki} - a_{ki}^j) + c_2 S_{ki} (A_{\alpha i} - a_{ki}^j), k = 1,...,K, i = 1,...,N \tag{8}$$

and

$$a_{ki}^{j+1} = a_{ki}^j + v_{ki}^{j+1}, k = 1,...,K, i = 1,...,N \text{ ,} \tag{9}$$

where v_{ki}^j and a_{ki}^j denote the corresponding i^{th} component of the k^{th} particle's velocity and position in j^{th} iteration, w denotes the inertia weight (0.8<w<1.2), c_1 and c_2 are constants and R_{ki} and S_{ki} are random numbers in the range [0,1].

2.3. LM Optimization of SOPNN Weights

2.3.1. Adaptation of LM algorithm to SOPNN

The optimization problem can be formulated in the following way: given the set of K samples each consisting of N–dimensional input data vector **x** and the dependent variable y, $\{(x_{1k}, x_{2k},...,x_{Nk}, y_k), k = 1,...,K\}$, minimize the nonlinear fitness function (5) by optimizing all the weights $\{a_1,...,a_M\}$ of the SOPNN polynomial relation P. In each iteration step the LM algorithm calculates the increment vector $\boldsymbol{\delta}$, $\delta_1,...,\delta_M$, and estimates the new weight

vector $\mathbf{a}=\mathbf{a}+\boldsymbol{\delta}$, $a_1+\delta_1,...,a_M+\delta_M$, in order to decrease the value of the fitness function (5). To calculate the increments, i.e. to estimate the improved polynomial weight vector ($\mathbf{a}=\mathbf{a}+\boldsymbol{\delta}$) for the next iteration, the polynomial P is calculated by

$$P(\mathbf{x}_k,\mathbf{a}+\boldsymbol{\delta}) \approx P(\mathbf{x}_k,\mathbf{a}) + \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_1}\cdot\delta_1 + ... + \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_M}\cdot\delta_M \ , \tag{10}$$

where the polynomial $P(\mathbf{x}_k,\mathbf{a}+\boldsymbol{\delta})$ denotes the new approximation of the dependent variable y. After substituting $\mathbf{a}+\boldsymbol{\delta}$ for $\mathbf{a}$ and (10) for $P(\mathbf{x}_k,\mathbf{a})$ in (5) we obtain:

$$e(\mathbf{a}+\boldsymbol{\delta}) = \sum_{k=1}^{K}\left(y_k - P(\mathbf{x}_k,\mathbf{a}) - \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_1}\cdot\delta_1 - ... - \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_M}\cdot\delta_M \right)^2 \tag{11}$$

By setting the partial derivatives of (11) with respect to increments δ_j, j=1,...,M, equal to zero, a set of M simultaneous linear equations with M unknowns ($\delta_1,...,\delta_M$) is obtained

$$\left\{ \sum_{k=1}^{K}\left[\frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_j}\cdot\left(y_k - P(\mathbf{x}_k,\mathbf{a}) - \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_1}\cdot\delta_1 - ... - \frac{\partial P(\mathbf{x}_k,\mathbf{a})}{\partial a_M}\cdot\delta_M \right)\right] = 0 \right\}_{j=1}^{M} \tag{12}$$

or in vector notation:

$$(\mathbf{J}^T\mathbf{J})\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{P}(\mathbf{a})] \tag{13}$$

where $\mathbf{J}$ denotes the Jacobian matrix i.e. the first order partial derivatives of $\mathbf{P}$ with respect to $\mathbf{a}$. After introducing an adjustable nonnegative damping factor γ and the diagonal matrix of $\mathbf{J}^T\mathbf{J}$ we obtain the Levenberg-Marquardt equation

$$(\mathbf{J}^T\mathbf{J} + \gamma\cdot diag(\mathbf{J}^T\mathbf{J}))\boldsymbol{\delta} = \mathbf{J}^T[\mathbf{y} - \mathbf{P}(\mathbf{a})], \tag{14}$$

summarizing the set of M linear equations with M unknowns

$$\begin{aligned}
&(1+\gamma)\delta_1\sum_{k=1}^{K}\left(\frac{\partial P_k}{\partial a_1}\right)^2 + \delta_2\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_1}\frac{\partial P_k}{\partial a_2} + ... + \delta_M\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_1}\frac{\partial P_k}{\partial a_M} = \sum_{k=1}^{K}\frac{\partial P_k}{\partial a_1}(y_k - P_k)\\
&\delta_1\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_1}\frac{\partial P_k}{\partial a_2} + (1+\gamma)\delta_2\sum_{k=1}^{K}\left(\frac{\partial P_k}{\partial a_2}\right)^2 + ... + \delta_M\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_2}\frac{\partial P_k}{\partial a_M} = \sum_{k=1}^{K}\frac{\partial P_k}{\partial a_2}(y_k - P_k)\\
&\qquad\qquad\vdots\\
&\delta_1\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_1}\frac{\partial P_k}{\partial a_M} + \delta_2\sum_{k=1}^{K}\frac{\partial P_k}{\partial a_2}\frac{\partial P_k}{\partial a_M} + ... + (1+\gamma)\delta_M\sum_{k=1}^{K}\left(\frac{\partial P_k}{\partial a_M}\right)^2 = \sum_{k=1}^{K}\frac{\partial P_k}{\partial a_M}(y_k - P_k)
\end{aligned} \tag{15}$$

where P_k denotes the SOPNN polynomial relation P ($\mathbf{x}_k$,$\mathbf{a}$).

2.3.2. Calculation of partial derivatives

As can be seen from Fig. 1, each layer contains the corresponding number of nodes. The i^{th} node in the layer λ, $n(\lambda, i)$, λ =1,2,..., is characterized by the corresponding weights $\{a_{\lambda i1},...,a_{\lambda i6}\}$ of the basic polynomial (1), where the inputs $z_{\lambda i1}$ and $z_{\lambda i2}$ denote either the polynomials from lower layers or the independent variables. In our algorithm the polynomials and the corresponding partial derivatives are calculated recursively.

Partial derivatives of the polynomial $P_{\lambda i}$ with respect to the weights of its topmost node $n(\lambda, i)$ are:

$\frac{\partial P_{\lambda i}}{\partial a_{\lambda i1}} = 1$, $\frac{\partial P_{\lambda i}}{\partial a_{\lambda i2}} = z_{\lambda i1}$, $\frac{\partial P_{\lambda i}}{\partial a_{\lambda i3}} = z_{\lambda i2}$, $\frac{\partial P_{\lambda i}}{\partial a_{\lambda i4}} = z_{\lambda i1}^2$, $\frac{\partial P_{\lambda i}}{\partial a_{\lambda i5}} = z_{\lambda i2}^2$ and $\frac{\partial P_{\lambda i}}{\partial a_{\lambda i6}} = z_{\lambda i1} z_{\lambda i2}$, where $z_{\lambda i1}$ and $z_{\lambda i2}$ are the inputs to node $n(\lambda, i)$.

Partial derivatives of the polynomial $P_{\lambda i}$ with respect to j^{th} weight of any lower layer node $n(x, y)$ are calculated by:

$\frac{\partial P_{\lambda i}}{\partial a_{xyj}} = \frac{\partial P_{\lambda i}}{\partial P_{xy}} \cdot \frac{\partial P_{xy}}{\partial a_{xyj}}$, where P_{xy} is the polynomial with the topmost node $n(x, y)$ having the corresponding weights a_{xyj}, j=1,…,6, and $\frac{\partial P_{\lambda i}}{\partial P_{xy}}$ is the partial derivative of the polynomial with topmost node $n(\lambda, i)$ with respect to the polynomial with topmost node $n(x, y)$. Note that partial derivative $\frac{\partial P_{\lambda i}}{\partial P_{xy}}$ is calculated by a chain rule over all existing connections between the nodes $n(\lambda, i)$ and $n(x, y)$.

The calculation of partial derivatives by chain rule is illustrated for the hypothetical example of the SOPNN shown in Fig. 2. For example, the polynomial P_{34} in Fig. 2 can be calculated by the following concatenated polynomial relations $P_{34}(P_{23}(P_{12}(x_1, x_2, \mathbf{a}_{12}), P_{14}(x_2, x_3, \mathbf{a}_{14}), \mathbf{a}_{23}), P_{25}(P_{14}(x_2, x_3, \mathbf{a}_{14}), x_4, \mathbf{a}_{25}), \mathbf{a}_{34})$, where $P_{\lambda i}$ denotes the polynomial with the topmost node $n(\lambda, i)$ and $\mathbf{a}_{\lambda i}$ is the $n(\lambda, i)$-th node weight vector, with the corresponding weights $a_{\lambda ij}$, j=1,…,6. The partial derivative of the polynomial P_{34} with respect to the weights of the topmost node of the polynomial P_{14} from Fig. 2 is:

$$\frac{\partial P_{34}}{\partial a_{14\,j}} = \left(\frac{\partial P_{34}}{\partial P_{23}} \cdot \frac{\partial P_{23}}{\partial P_{14}} + \frac{\partial P_{34}}{\partial P_{25}} \cdot \frac{\partial P_{25}}{\partial P_{14}} \right) \cdot \frac{\partial P_{14}}{\partial a_{14\,j}}, \; j = 1,...,6,$$

where

$$\frac{\partial P_{34}}{\partial P_{23}} = a_{342} + 2a_{344} P_{23} + a_{346} P_{25},$$

$$\frac{\partial P_{34}}{\partial P_{25}} = a_{343} + 2a_{345} P_{25} + a_{346} P_{23},$$

$$\frac{\partial P_{23}}{\partial P_{14}} = a_{233} + 2a_{235} P_{14} + a_{236} P_{12},$$

$$\frac{\partial P_{25}}{\partial P_{14}} = a_{252} + 2a_{254} P_{14} + a_{256} x_4,$$

$\frac{\partial P_{14}}{\partial a_{141}} = 1$, $\frac{\partial P_{14}}{\partial a_{142}} = x_2$, $\frac{\partial P_{14}}{\partial a_{143}} = x_3$, $\frac{\partial P_{14}}{\partial a_{144}} = x_2^{\,2}$, $\frac{\partial P_{14}}{\partial a_{145}} = x_3^{\,2}$ and $\frac{\partial P_{14}}{\partial a_{146}} = x_2 x_3$, with $a_{14\,j}$, $a_{23\,j}$, $a_{25\,j}$ and $a_{34\,j}$ denoting the j^{th} weight of the corresponding topmost node of the polynomials P_{14}, P_{23}, P_{25} and P_{34}, respectively.

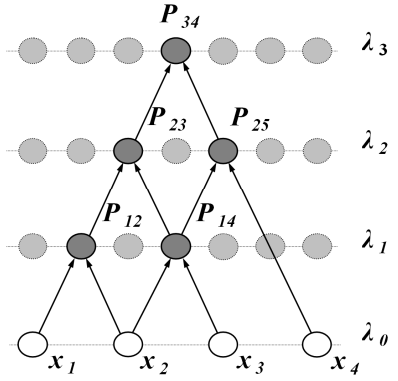


Fig. 2.

2.3.3. LM algorithm in pseudo code

In order to speedup the minimization of the fitness function the nonnegative damping factor γ is adjusted at the beginning of each optimization cycle. The algorithm starts with the best polynomial obtained from GMDH algorithm $P(\mathbf{x}_k,\mathbf{a})$ and calculates the corresponding error $e_0=e(\mathbf{a})$ by using (5), sets the maximum allowed damping factor γ_M, maximum number of iterations i_M and initializes the damping factor $\gamma=\gamma_0$ and the coefficient $v>1$. The following pseudo code illustrates the simplified flow diagram of the LM algorithm (Marquardt, 1963):

Begin with: $i=0$, $\gamma=\gamma_0$, $e_0=e(\mathbf{a})$ Eq. (5)
1. Set $\gamma_1=\gamma$
2. Calculate $\boldsymbol{\delta}_1$, and $e_1=e(\mathbf{a}+\boldsymbol{\delta}_1)$; Eq. (15) and (11)
3. Set $\gamma_2=\gamma/v$
4. Calculate $\boldsymbol{\delta}_2$ and $e_2=e(\mathbf{a}+\boldsymbol{\delta}_2)$; Eq. (15), and (11)
5. If $e_1<e_0<e_2$ Then $\gamma=\gamma_1$, $e_0=e_1$, $\mathbf{a}=\mathbf{a}+\boldsymbol{\delta}_1$ and GoTo 12.
6. If $e_2<e_0<e_1$ Then $\gamma=\gamma_2$, $e_0=e_2$, $\mathbf{a}=\mathbf{a}+\boldsymbol{\delta}_2$ and GoTo 12.
7. If $e_0<e_1$ and $e_0<e_2$ Then $\gamma=\gamma_1\cdot v$,
8. If $\gamma>\gamma_M$ Then GoTo 13.
9. $\gamma_1=\gamma$
10. Calculate $\boldsymbol{\delta}_1$, and $e_1=e(\mathbf{a}+\boldsymbol{\delta}_1)$; Eq. (15) and (11)
11. If $e_1<e_0$ Then $\gamma=\gamma_1$, $e_0=e_1$, $\mathbf{a}=\mathbf{a}+\boldsymbol{\delta}_1$
12. $i=i+1$, If $i\le i_M$ Then GoTo 1.
13. End

The idea of the LM algorithm is very simple but its adaptation to SOPNN is somewhat complicated by the recursive calculation of the polynomials and its derivatives.

3. SOPNN and MLP models for time-constrained applications

In certain cases the execution of the procedures based on "first principles" may be unfeasible in real time. In these cases, it is reasonable to construct the corresponding meta-model (Jin, Chen & Simpson, 2000) by maximizing the approximation accuracy for the given computational complexity. To derive a meta-model from the original high-complexity model by machine learning technique it is necessary to generate sufficient training and validation examples from the original model. SVM (Vapnik, Golowich & Smola, 1997) and ANN (Ferrari & Stengel, 2005) can be efficiently used for the approximation of multidimensional nonlinear functions.

In measurement and control applications (Maric & Ivek, 2011) it is often necessary to maximize the approximation accuracy for the given computational complexity. The computational complexity can be measured by the corresponding execution time (ET). Our models are tailored for the flow computer based on low computing power microcomputer (8-bit/16-MHz) with implemented floating point (FP) subroutines for single precision addition, multiplication, division and the exponential function. In this section we will briefly summarize the procedure for the estimation of the ET of SOPNN and MLP models in low computing power systems (Maric & Ivek, 2011).

3.1. Computational Complexity of SOPNN

The ET of the SOPNN is defined by the total number of polynomial nodes and the maximum ET of the basic low order polynomial (1). The maximum ET of the SOPNN (Maric & Ivek, 2011) can be estimated by

$$T_{SOPNN} \le N\cdot\left(N_{add}\cdot T_{add}+N_{mul}\cdot T_{mul}\right) \tag{16}$$

where N denotes the total number of polynomial nodes, while N_{add} and N_{mul} denote the corresponding total number of FP additions and FP multiplications, necessary to calculate the basic polynomial (1), and T_{add} = 50 μs and T_{mul} = 150 μs are the corresponding maximum execution times of the FP addition and FP multiplication. If we rewrite the polynomial (1) by Horner's rule, its calculation is reduced to N_{add} = 5 FP additions and N_{mul} = 5 FP multiplications i.e.

$$p_{\lambda i}=a_{\lambda i1}+z_{\lambda i1}\left(a_{\lambda i2}+a_{\lambda i4}z_{\lambda i1}+a_{\lambda i6}z_{\lambda i2}\right)+z_{\lambda i2}\left(a_{\lambda i3}+a_{\lambda i5}z_{\lambda i2}\right).$$

3.2. Computational Complexity of MLP

We trained the feed-forward MLP consisting of one output neuron and M-1 neurons in the hidden layer, each neuron employing the sigmoid activation function. The MLP scheme is shown in Fig. 3. The maximum ET of the calculation procedure for the simple MLP with *N* inputs, one output neuron and M-1 neurons in the hidden layer can be estimated by (Maric & Ivek, 2011)

$$T_{mlp} \le (M-1)(N+1)(T_{add}+T_{mul})+M(T_{\exp}+T_{add}+T_{div}) \tag{17}$$

where T_{exp}, and T_{div} denote the maximum execution time for the FP exponential function and FP division, respectively. The maximum ETs for the above FP operations in the flow computer are: T_{exp} = 3470 μs and T_{div} = 430 μs.

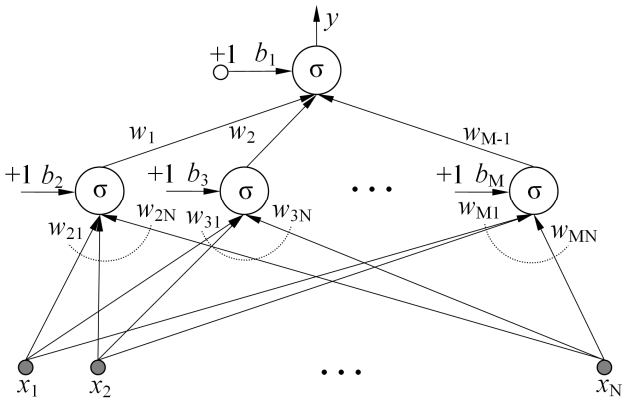


Fig. 3.

4. Compensation of flow rate error

We investigated the combined effect of the Joule-Thomson (JT) coefficient and the isentropic exponent of a natural gas on the accuracy of flow rate measurements based on differential devices (Maric & Ivek, 2010). The measurement of the flow rate of a natural gas (ISO-20765-1, 2005) flowing in a pipeline through the orifice plate with corner taps (ISO-5167-2, 2003) is illustrated in Fig. 4.

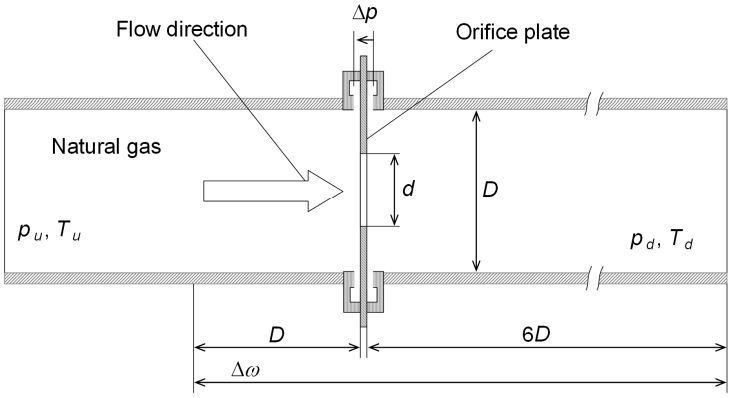


Fig. 4.

A detailed description of the flow rate equation with the corresponding iterative computation scheme is given in (ISO-5167-2, 2003). The calculation of the natural gas flow rate depends on multiple parameters:

$$q_u = q(P_u, T_u, \Delta p, \rho_u, \gamma_u, \kappa_u, D, d) \tag{18}$$

where q_u, ρ_u, γ_u and κ_u represent the corresponding mass flow rate, density, viscosity and the isentropic exponent calculated at upstream pressure P_u and temperature T_u, while ΔP, D and d denote the differential pressure across the orifice plate, internal diameter of the pipe and the orifice, respectively. In case of upstream pressure and downstream temperature measurement, as suggested by (ISO-5167-2, 2003), the flow rate (18), changes to:

$$q_d = q(P_u, T_d, \Delta p, \rho_d, \gamma_d, \kappa_d, D, d) \tag{19}$$

where q_d, ρ_d, γ_d and κ_d denote the corresponding mass flow rate, density, viscosity and the isentropic exponent calculated in "*downstream conditions*" i.e. at the upstream pressure p_u and the downstream temperature T_d. For certain natural gas compositions and operating conditions the flow rate q_d may differ significantly from q_u and the corresponding compensation for the temperature drop effects, due to JT expansion, may be necessary in order to preserve the requested measurement accuracy (Maric & Ivek, 2010). Precise compensation of the flow rate error needs double calculation of the properties of a natural gas and the flow rate what makes the compensation unfeasible in real time in low computing power embedded systems. To reduce the computational burden we aim to derive a low-complexity flow rate correction factor model that will enable direct compensation of the flow rate error caused by the measurement of the downstream temperature. The correction factor model has to be simple enough in order to be executable in real time and accurate enough to ensure the requested measurement accuracy.

The flow rate correction factor K can be obtained precisely by dividing the true flow rate q_u calculated in the upstream conditions, (18), by the flow rate q_d calculated in the "downstream conditions" (19):

$$K = \frac{q_u}{q_d}. \tag{20}$$

For the given correction factor (20), the flow rate at upstream pressure and temperature (18) can be calculated directly from the flow rate computed in "*downstream conditions*" (19), i.e. $q_u = K \cdot q_d$. The relative flow rate measurement error E_r can be estimated by comparing the uncompensated (q_d) and precisely calculated (q_u) flow rate i.e. $E_r = (q_d - q_u)/q_u$. Our objective is to derive the surrogate model of the flow rate correction factor. Given the surrogate model (K_{sm}) for the flow rate correction factor (20), the true flow rate q_u can be approximated by: $q_{sm} = K_{sm} \cdot q_d$, where q_{sm} denotes the flow rate corrected by the surrogate model (K_{sm}) of the real correction factor K. The relative measurement error E_{sm} can be estimated by comparing the approximate (q_{sm}) and the precisely calculated (q_u) flow rate i.e.

$$E_{sm} = (q_{sm} - q_u)/q_u \tag{21}$$

The correction is modeled by SOPNN and MLP, having equivalent computational complexities, and the corresponding approximation errors are compared. Note that natural gas properties, like density, isentropic exponent and viscosity, need to be calculated and are involving the natural gas characterization parameters in the flow rate calculation, like molar fractions of the components, superior calorific value and relative density. The procedures for the correction of the flow rate error and for the estimation of optimal set of input parameters (Table 1) needed for the correction factor modeling are elaborated in (Maric & Ivek, 2010).

Table 1
Optimal Set of input Parameters for Natural Gas Flow Rate Correction Factor Modelling

Index	Parameter description	Range of application
0	X_{CO2} - mole fraction of carbon dioxide	$0 \le X_{CO2} \le 0.20$
1	X_{H2} - mole fraction of hydrogen	$0 \le X_{H2} \le 0.10$
2	p - absolute pressure in MPa	$0 < p \le 12$
3	T - temperature in K	$263 \le T \le 368$
4	Δp - differential pressure in MPa	$0 \le \Delta p \le 0.25p$
5	ρ - density in kg/m^3	intermediate result
6	ρ_r - relative density	$9.55 \le \rho_r \le 0.80$
7	H_S - superior calorific value in MJ/m^3	$30 \le H_S \le 45$
8	β - orifice to pipe diameter ratio: d/D	$0.1 \le \beta \le 0.75$

The next section describes the correction factor modeling and the results of the simulation of the flow rate error.

5. Flow rate error compensation modeling

For the purpose of meta-modeling, the precise calculation of the natural gas flow rate correction factor is implemented in software and run on a digital computer. The training data set, validation data set and 10 test data sets, each consisting of 20000 samples of correction factor, were randomly sampled across the entire space of application (Table 1) in accordance with (Maric & Ivek, 2010). The ranges of application are shown in Table 1. The maximum tolerable ET and the maximum acceptable root relative squared error (RRSE) (Maric & Ivek, 2011) of the correction factor surrogate model in our flow computer prototype are limited to 30 ms and 5%, respectively. For the given maximum ET of the model, maximum ET of the FP operations, and 9 input parameters, the maximum acceptable number of SOPNN nodes and MLP neurons can be calculated by (16) and (17), respectively. In order to build the models with equivalent ET not exceeding 30 ms, the MLP (Fig. 3) is limited to 5 neurons with the maximum ET, $T_{mlp} \leq 27.75$ ms, estimated by (17). The complexity of the corresponding SOPNN model (Fig. 5) is limited to maximum 27 polynomial nodes with the maximum ET, $T_{SOPNN} \leq 27.00$ ms, estimated by (16).

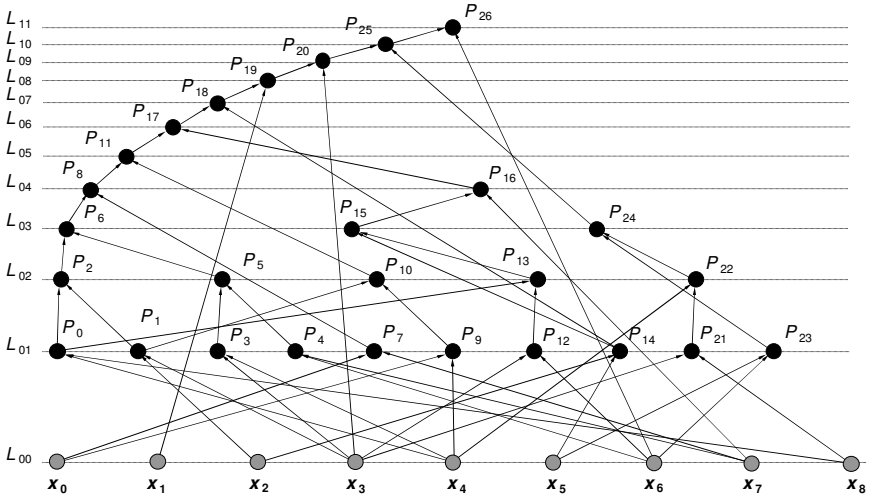


Fig. 5.

The approximation accuracy of the derived models are tested on 10 randomly generated test data sets for the flow rate correction factor, each consisting of 20000 samples, and the results are shown in Fig. 6. The corresponding mean RRSE and standard deviation are given in Table 2. The results obtained from 10 independent test data sets clearly indicate that the models are not over-fitted.

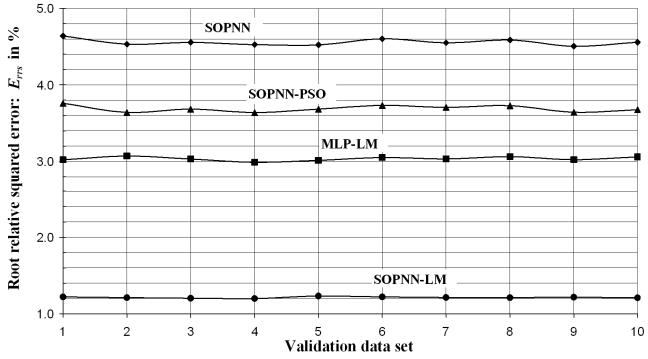


Fig. 6.

Table 2
Average RRSE of the correction factor for 10 test data sets when approximated by MLP and SOPNN models

	Root relative squared error: E_{rrs} in %			
	MLP-LM	***SOPNN***	***SOPNN-PSO***	***SOPNN-LM***
Mean value	3.036	4.558	3.690	1.214
Standard deviation	0.02525	0.04099	0.04158	0.00968

The MLP-LP in Fig. 6 and Table 2 denotes the feed forward MLP (Fig. 3) with 4+1 neurons trained by the LM algorithm. The SOPNN represents the 27-node self-organized model (Fig. 5) trained by the GMDH algorithm using the compound measure $E_{CE} = c_w \cdot (E_{rrs}/E_{rrs0})^2 + (1-c_w)\cdot(T_{exe}/T_{exe0})^2$ for model selection (Maric & Ivek, 2011), which combines the constraints upon model approximation error ($E_{rrs} \leq E_{rrs0}$) and ET ($T_{exe} \leq T_{exe0}$) by the weighting coefficient ($0 \leq c_w \leq 1$). SOPNN-PSO and SOPNN-LM represent the same SOPNN model optimized by the PSO and the LM, respectively. From Fig. 6 and Table 2 it can be seen that MLP trained by the LM algorithm (MLP-LM) achieves about 33% lower root relative squared error than the non-optimized SOPNN model having approximately the same complexity. Optimization of the SOPNN model by PSO (SOPNN-PSO) is time consuming. The procedure is computationally intensive due to the high dimensional search space (27*6=162 weights) requesting large number of particles and iterations. The unknown lower and upper boundaries of the search space are complicating the procedure additionally. To overcome the problem the optimization has been divided into epochs each consisting of 2000 generations. At the beginning of each epoch the boundaries are fixed around the position of the best particle from the previous epoch in the following way: $L_i=A_i(1-\mu)$, $U_i=A_i(1+\mu)$, where A_i denotes the position of the best particle in the i^{th} dimension, L_i and U_i denote the lower and the upper boundary of the i^{th} dimension of the search space and μ is a positive number. The μ has been varied from 0.01 to 1.5 and the best results, in this particular application, are obtained for μ=1.2. The SOPNN-PSO improves the SOPNN accuracy for about 20%. Since the PSO shows very slow but constant improvement of the model, the LM algorithm for the optimization of the SOPNN weights has been implemented in order to speed-up the convergence.

The SOPNN-LM converges rapidly. It decreased the SOPNN error almost four times (73.3%) and outperformed all other models. It achieved 60% better accuracy than the corresponding MLP-LM. In each cycle of the LM algorithm, the damping coefficient has been automatically adjusted for maximum decrease of the RRSE and the model converges rapidly to its optimum weights. The same effect of the LM optimization on model error has been observed for models at all layers. Fig. 7 illustrates the average error in logarithmic scale obtained for the eight best ranked models from each layer, before (SOPNN) and after they have been optimized by the LM algorithm (SOPNN-LM).

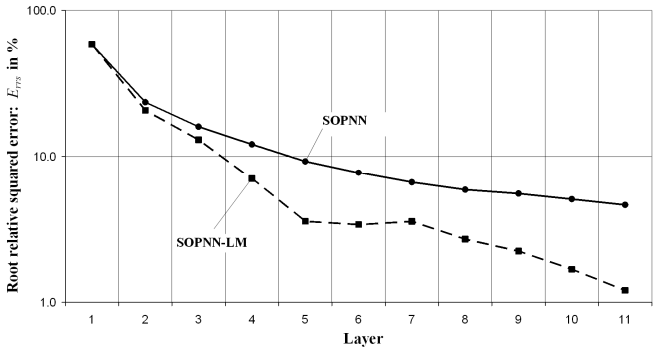


Fig. 7.

The results show that the SOPNN polynomial relation offer substantially better approximation abilities than it can be concluded from the characteristics of raw models obtained directly from the GMDH algorithm. We have obtained similar results by modeling the procedures for the calculation of various thermodynamic properties of natural gas including molar heat capacities, speed of sound, etc.

The model generated by the GMDH algorithm is generally far from optimal since the weights of each node-polynomial, after calculated by the regression, remain unchanged throughout the rest of the learning process. The GMDH algorithm is trying to maximize the approximation accuracy of the model by embedding the system internal dependencies into the model structure. To achieve this it performs the regression of the polynomial

weights of the uppermost node only. In this way the GMDH algorithm controlled by the imposed metrics produces the model that generally achieves the structure, the weights and the approximation accuracy that can be considered as a very good starting point for further optimization. Next Section demonstrates the outstanding ability of the SOPNN to model the polynomial-type recurrence relations.

6. Multiple Superimposed Oscillations Modeling

In this section we discuss the learning of the recurrence relations from time series by using SOPNN. We analyze the performances of SOPNN on multiple superimposed oscillations (MSO) task that has already been attempted with varying degrees of success by using ANN and SVM (Xue, Yang & Haykin, 2007; Schmidhuber, Wierstra, Gagliolo & Gomez, 2007; Holzmann & Hauser, 2010; Ceperic, Gielen & Baric, 2012). The following multiple sinusoids are modeled:

$$y_2 = \sin(0.2n) + \sin(0.311n), \tag{22}$$

$$y_3 = y_2 + \sin(0.42n), \tag{23}$$

$$y_4 = y_3 + \sin(0.51n), \tag{24}$$

$$y_5 = y_4 + \sin(0.74n), \tag{25}$$

where n=1,…,700. Note that the frequencies of the sinusoids are not integer multiples of each other. As described in (Ceperic, Gielen & Baric, 2012) the first 400 samples (n=1,...,400) are used to train the model, while the rest of data (n=401,...,700) is used to test the model. The data is generated in double floating point (FP) number precision. We have limited the complexity of our SOPNN to maximum 20 nodes (N), where each node-polynomial can be calculated by only 5 FP additions and 5 FP multiplications. In the worst case the maximum total number of 100 FP additions and 100 FP multiplications would be necessary to calculate the SOPNN output, which is far below the computational complexities of the corresponding models described in the above mentioned references. As in (Ceperic, Gielen & Baric, 2012), we limited the maximum dimension (D) of the input space to 50 what corresponds to maximum 50 delayed output samples in the recurrent configuration.

6.1. Comparison with other approaches

Table 3 shows the mean square error (MSE) and the normalized root mean square error (NRMSE) (Holzmann & Hauser, 2010) on the test data set, obtained by the non-optimized GMDH model (SOPNN) and by the same model after it has been optimized by the proposed LM algorithm (SOPNN-LM). In Table 3, D denotes the dimensionality, i.e. the corresponding total number of delayed output signals, and N denotes the total number of nodes in SOPNN. From Table 3 it can be seen that the dimension (D) of the input space and the number of nodes (N) of the SOPNN are increasing with the number of sinusoids in the MSO in order to preserve high prediction accuracy.

Table 3
Mean square error in the prediction of the next sample of the MSO test data set (y_2, y_3, y_4, y_5; n=401,...,700) by the corresponding non-optimized (SOPNN) and optimized (SOPNN-LM) N-node, D-dimensional SOPNN model generated by the GMDH algorithm using the corresponding training data set (y_2, y_3, y_4, y_5; n=1,...,400)

			SOPNN		SOPNN-LM	
MSO	***D***	***N***	**MSE**	**NRMSE**	**MSE**	**NRMSE**
y_2	4	3	1.28E-04	9.16E-03	3.07E-27	5.53E-14
y_3	10	8	1.16E-05	2.75E-03	3.89E-25	5.04E-13
y_4	16	13	6.47E-05	5.52E-03	2.28E-24	1.04E-12
y_5	50	16	2.66E-04	1.01E-02	6.06E-25	4.84E-13

Xue, Yang & Haykin, 2007, use four reservoirs each with 100 neurons to approximate the two sine problem (22) with “decoupled echo state network with lateral inhibition”. The MSE they obtained on test data set was

$3 \cdot 10^{-4}$ and is almost 23 orders of magnitude higher than the MSE obtained by the corresponding y_2 SOPNN-LM model. Also, a huge complexity of their model is incomparable to low complexity of our 3-node, 4-dimensional y_2 SOPNN-LM model (Table 3). Schmidhuber, Wierstra, Gagliolo & Gomez, 2007, use PI-EVOLINO and EVOKE networks to model the MSO data sets. They reported the following normalized root mean square errors: y_2: NRMSE=4.15E-3, y_3: NRMSE=8.04E-3, y_4: NRMSE=1.01E-1 y_5: NRMSE=1.66E-1. The NRMSE errors they reported are at least 9 orders of magnitude worse than the results obtained by the optimized SOPNN-LM (Table 3). Also, the generalization results they obtained for two sinusoids y_2 by the EVOKE networks, which are based on SVM, are even much worse. The huge complexity of their models cannot be compared with the simplicity of SOPNN models. Ceperic, Gielen & Baric, 2012, use recurrent sparse support vector regression machines trained by active learning in time-domain combined with the optimization of SVM hyper parameters to model the MSO data sets. They reported the following errors:

- y_2: MSE=3.57E-8, NRMSE=1.88E-4, using 13 SV;
- y_3: MSE=2.33E-6, NRMSE=1.23E-3, using 15 SV;
- y_4: MSE=1.75E-5, NRMSE=2.86E-3, using 15 SV;
- y_5: MSE=9.16E-5, NRMSE=5.94E-3, using 15 SV,

which are considerably better than the errors reported in (Xue, Yang & Haykin, 2007) and (Schmidhuber, Wierstra, Gagliolo & Gomez, 2007) but still the orders of magnitude worse than the errors obtained by the corresponding optimized SOPNN-LM model shown in Table 3. Also the computational complexity (Maric & Ivek, 2011) of the SVM model with 13 or 15 support vectors is much higher than the complexity of the corresponding SOPNN-LM model with 3, 8, 13 or 16 2-dimensional, 2nd order node-polynomials (1).

Holzmann & Hauser, 2010, achieved very good results using echo state networks (ESN) with arbitrary infinite impulse response filter neurons and a delay&sum readout. They used a reservoir of 100 specialized neurons tuned to different frequencies, which are able to generate a superposition of sinusoids. They did not report the results in numerical form but from graphical presentation of the NRMSE in logarithmic scale it can be seen that their errors for y_2 (NRMSE>2E-9), y_3 (NRMSE>2E-7), y_4 (NRMSE>1E-5) and y_5 (NRMSE>5E-5) are more than 4 orders of magnitude higher than the corresponding SOPNN-LM errors (Table 3). Again, the huge complexity of ESN models cannot be compared with the simplicity of the corresponding SOPNN.

6.2. Generalization abilities of SOPNN

Table 4 shows the MSE and the NRMSE for the next sample prediction obtained on various combinations of sinusoids from y_5 MSO by using the same model (Fig. 8) generated and optimized on y_5 training data. Note that the dimension of each test data set in Table 4 is equal to 50 as determined by the model (SOPNN-LM-y_5), which is built and optimized by using 50-dimensional training data set obtained from the first 400 samples (n=1,...,400) of y_5 MSO (25). The output from the 16th node-polynomial P_{15} in Fig. 8 is the next predicted output (x_{50}) calculated from 50 delayed outputs $x_0,\ldots,x_{49}$, where x_0 denotes the oldest and x_{49} the most recent output from 50-samples window. In the next step the outputs $x_1,...,x_{50}$ are used to predict the output x_{51} and so on. From Fig. 8 it can be seen that the SOPNN model uses only 12 out of 50 delayed outputs to predict the next output. As can be seen from Table 4, the same SOPNN-LM-y_5 model is able to accurately predict any subset of sinusoids from y_5 MSO (25). It is extremely superior to any model tailored to a specific MSO by various ANN and SVM approaches.

In order to demonstrate another superior characteristic of SOPNN, let us consider for example the following MSO signal obtained after changing the amplitudes and phase shifts of all the corresponding sinusoids of y_5 MSO (25) e.g.:

$$\begin{aligned} y_{5a} = {} & 1.2\sin(1.3+0.2n) - 1.8\sin(0.7-0.311n) - \\ & 2.1\sin(2.1+0.42n) + 1.35\sin(-1+0.51n) - . \\ & 2.8\sin(0.32-0.74n) \end{aligned} \tag{26}$$

The next sample prediction errors MSE=1.91E-22 and NRMSE=4.45E-12 have been obtained by the same SOPNN-LM-y_5 model after predicting the data set (y_{5a}, n=401,...,700) arranged as 50 delayed output samples in a recurrent configuration. Fig. 9 illustrates the modeled values for y_{5a} (26) and the corresponding model error for the first 300 steps calculated by the SOPNN-LM-y_5 (Fig. 8).

Table 4
MSE and RMSE in the prediction of the next sample of the MSO and single sinusoid test data obtained by the same 50-Dimensional SOPNN generated and optimized on y_5 training data

Test data set	**SOPNN-LM-y_5**	
MSO	**MSE**	**NRMSE**
y_2	5.31E-26	2.30E-13
y_3	8.16E-26	2.31E-13
y_4	9.76E-26	2.14E-13
y_5	6.06E-25	4.84E-13
sin(0.2n)	5.76E-26	3.39E-13
sin(0.311n)	5.09E-26	3.21E-13
sin(0.42n)	4.63E-26	3.04E-13
sin(0.51n)	4.56E-26	3.01E-13
sin(0.74n)	2.56E-26	2.26E-13

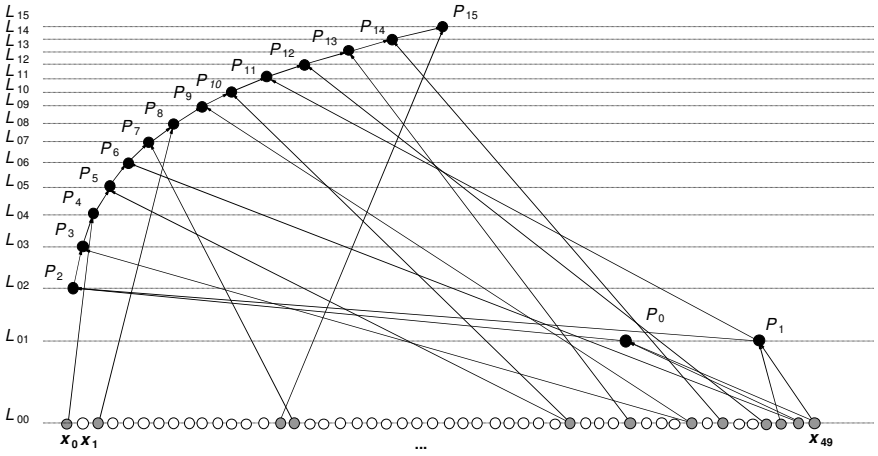
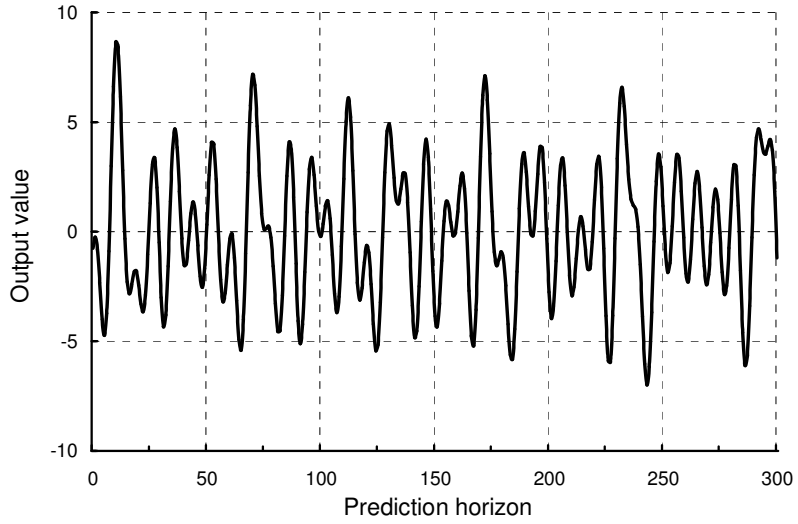
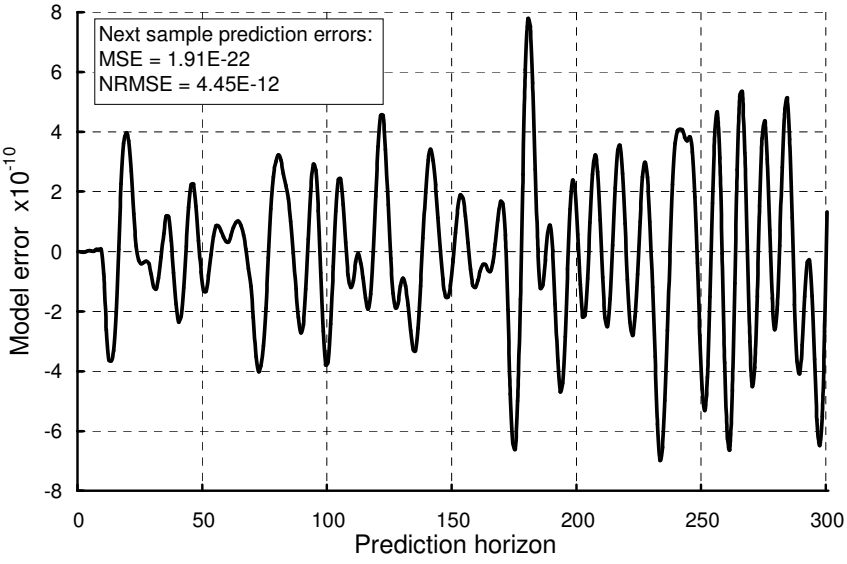


Fig. 8.



(a)



(b)

Fig. 9.

Fig. 9 shows the results of the prediction of y_{5a} test data by SOPNN-LM-y_5 model starting with y_{5a} data vector, which consists of 50 delayed outputs preceding the output sample n=401. Fig. 9(a) shows the prediction of y_{5a} test data by SOPNN-LM-y5 for the prediction horizon from 0 to 300 samples. Fig. 9(b) shows the corresponding prediction error for the same prediction horizon. The model preserves high accuracy even if changing the amplitudes and phases of the sinusoids from which the y_5 MSO signal is composed. Note that maximum absolute prediction error did not exceed $8 \cdot 10^{-10}$ in the whole prediction horizon.

Fig. 10 shows the absolute model error in logarithmic scale when predicting test data for y_5 MSO (25) and y_{5a} MSO (26) by the same SOPNN-LM-y_5 model in the prediction horizon from 0 to 3000 samples.

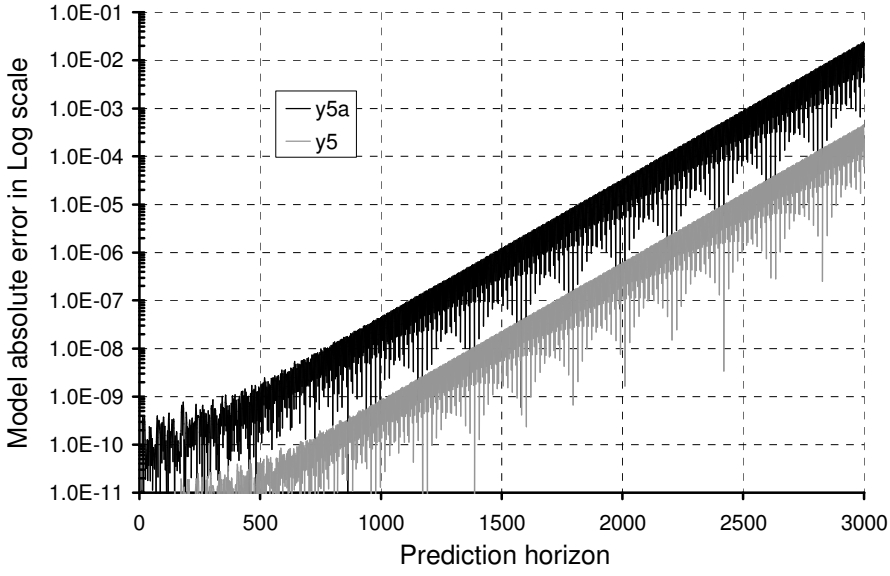


Fig. 10.

From Fig. 10 it can be seen that the prediction error is increasing exponentially when increasing the prediction horizon. When using SOPNN-LM-y5 to predict the y_{5a} MSO test data (black line), the corresponding maximum errors are more than 50 times higher than the errors obtained on y_5 MSO test data (gray line). But, as can be concluded from Fig. 9 and Fig. 10, the y_{5a} MSO prediction error still remains extremely low in a wide prediction horizon. Similar error characteristics are obtained by SOPNN-LM-y5 model when predicting any subset of sinusoids from y_5 MSO.

From Table 4 and Figs. 9 and 10 it can be seen that unlike any other above mentioned approach the SOPNN, optimized by the LM algorithm, has extraordinary generalization abilities. It can accurately predict in a wide prediction horizon any subset of sinusoids from the MSO signal it has been trained for, even if their amplitudes and phases are significantly changed. The same equations can be also used to accurately predict any sinusoid or a subset of superimposed sinusoids from y_5 MSO (25). The equations and the coefficients of the SOPNN-LM-y5 model (Fig. 8) are given in Appendix A for testing purposes.

7. Conclusion

The paper presents an efficient adaptation of the Levenberg-Marquardt algorithm for the optimization of the SOPNN. The paper points out that LM algorithm makes the SOPNN very competitive for the approximation of complex systems and procedures, particularly in real-time applications, since high approximation accuracy is generally achieved with low complexity models. In computationally intensive real-time applications the complex procedures may be replaced by simplified SOPNN surrogates and thus become feasible in real-time. The paper particularly emphasizes a high accuracy/complexity ratio of the optimized model and the simplicity of its implementation in software.

The LM algorithm has been tested by modeling the computationally intensive procedure for the correction of the flow rate error. It was shown that the approximation characteristics of the original SOPNN can be improved substantially when optimizing the model weights by the LM algorithm. The SOPNN outperformed the corresponding MLP model when both optimized by the LM algorithm and having approximately equal computational complexities. Similar results have been obtained by modeling the procedures for the calculation of various thermodynamic properties of a natural gas.

The SOPNN proves to be extremely efficient in learning polynomial-type recurrence relations from time series. When optimized by the LM algorithm the SOPNN outperformed the ANN and the SVM in MSO modeling. When modeling a MSO recurrence relation the optimized SOPNN displays outstanding generalization ability, uncommon to ANN and SVM models, and can accurately predict any subset of superimposed sinusoids from the MSO signal it has been trained for. The model prediction error remains very low even if changing the amplitudes and phases of the superimposed sine waves. SOPNN is also able to learn with extreme accuracy the recurrence

relations of multiple products of sine waves, modulated MSO, damped MSO, etc. The ability to learn the recurrence relation and to accurately predict the future values from past known samples opens the possibilities of using the SOPNN in modeling the dynamic behavior of complex systems.

Appendix A

The concatenated polynomial relation of the SOPNN model from Fig. 8

$$P = P_{15}(P_{14}(P_{13}(P_{12}(P_{11}(P_{10}(P_9(P_8(P_7(P_6(P_5(P_4(P_3(P_2(P_0(x_{48},x_{49}),P_1(x_{47},x_{49})),x_{41}),x_0),x_{33}),x_{48}),x_{15}),x_2),x_{41}),x_{33}),P_1(x_{47},x_{49})),x_{46}),x_{37}),x_{43}),x_{14})$$

can be computed by 16 basic polynomials,

$P_0(x_{48},x_{49})$, $P_1(x_{47},x_{49})$, $P_2(P_0,P_1)$, $P_3(P_2,P_{41})$, $P_4(P_3,x_1)$, $P_5(P_4,x_{33})$, $P_6(P_5,x_{48})$, $P_7(P_6,x_{15})$, $P_8(P_7,x_2)$, $P_9(P_8,x_{41})$, $P_{10}(P_9,x_{33})$, $P_{11}(P_{10},P_1)$, $P_{12}(P_{11},x_{46})$, $P_{13}(P_{12},x_{37})$, $P_{14}(P_{13},x_{43})$, $P_{15}(P_{14},x_{14})$,

where the basic polynomials are calculated by

$$P_i(z_1, z_2) = a_{i,0} + a_{i,1}z_1 + a_{i,2}z_2 + a_{i,3}z_1^2 + a_{i,4}z_2^2 + a_{i,5}z_1z_2$$

using the following optimized double precision coefficients:

$a_{0,0}$=2.4089806255330348e-001, $a_{0,1}$=-2.0609933517875878e-002, $a_{0,2}$=1.7531109297004226e+000,
$a_{0,3}$=1.8381037377004718e-009, $a_{0,4}$=4.6419038970178617e-005, $a_{0,5}$= -8.6782752584535839e-008,

$a_{1,0}$=-6.6581136153327869e-001, $a_{1,1}$=-2.5039829582659459e-009, $a_{1,2}$=1.1149843617468462e+000,
$a_{1,3}$=-6.7607152698616724e-012, $a_{1,4}$=2.9528090425706862e-005, $a_{1,5}$=-1.5739033560547226e-011,

$a_{2,0}$=-7.8345182851489101e-002, $a_{2,1}$=3.0222771938049342e+000, $a_{2,2}$=-1.4659393406572994e+000,
$a_{2,3}$=3.1719446468444557e-001, $a_{2,4}$=7.8358643543900208e-001, $a_{2,5}$=-9.9744717755725765e-001,

$a_{3,0}$=-7.1836925226834172e-002, $a_{3,1}$=8.2254766389549738e-001, $a_{3,2}$=-5.8661286884162590e-002,
$a_{3,3}$=-9.4478934614005779e-007, $a_{3,4}$=1.0203172272677782e-004, $a_{3,5}$=-7.1603355247232822e-012,

$a_{4,0}$=-7.8622868604762711e-002, $a_{4,1}$=8.3924346238730552e-001, $a_{4,2}$=-2.3551250558386455e-001,
$a_{4,3}$=5.8200253943802276e-008, $a_{4,4}$=4.5880465656650697e-009, $a_{4,5}$=-3.2699243420244513e-008,

$a_{5,0}$=-5.7663794534583354e-002, $a_{5,1}$=8.5302808485752646e-001, $a_{5,2}$=-1.7417718227724835e-001,
$a_{5,3}$=-7.0354372488410098e-008, $a_{5,4}$=5.3723532859500584e-005, $a_{5,5}$=-8.4611612366864986e-011,

$a_{6,0}$=-1.0373663629806662e-001, $a_{6,1}$=8.5125981079556934e-001, $a_{6,2}$=-6.4996174285114006e-001,
$a_{6,3}$=4.8155545589541646e-010, $a_{6,4}$=-6.7537943733023871e-005, $a_{6,5}$=-1.1056387926085760e-009,

$a_{7,0}$=-8.2497971373195644e-002, $a_{7,1}$=8.9229466216913467e-001, $a_{7,2}$=1.7609360498148038e-002,
$a_{7,3}$=5.2983902284172561e-008, $a_{7,4}$=2.1005689852310062e-011, $a_{7,5}$=2.1265022979134083e-009,

$a_{8,0}$=-7.9273328703511364e-002, $a_{8,1}$=8.9884511364949637e-001, $a_{8,2}$=6.0847463081777652e-002,
$a_{8,3}$=1.1340108864742532e-007, $a_{8,4}$=7.9841230450439977e-010, $a_{8,5}$=2.3588069735832587e-008,

$a_{9,0}$=-6.4817529180249148e-002, $a_{9,1}$=8.9631687643991975e-001, $a_{9,2}$=-3.5335719981528518e-001,
$a_{9,3}$=-1.9340241083767072e-007, $a_{9,4}$=-4.4699594154393525e-005, $a_{9,5}$=9.3368507730513902e-011,

$a_{10,0}$=-8.6117614268338985e-002, $a_{10,1}$=9.4290247056516585e-001, $a_{10,2}$=3.2597051000195628e-003,
$a_{10,3}$=1.0750613301384445e-009, $a_{10,4}$=-3.0999033115579099e-005, $a_{10,5}$=6.4343070771435519e-012,

$a_{11,0}$=-3.0294545534514258e-002, $a_{11,1}$=1.0343523735701383e+000, $a_{11,2}$=1.5009384616591143e-001,
$a_{11,3}$=-1.5912006880327145e-009, $a_{11,4}$=1.6831199002889785e-004, $a_{11,5}$=-1.4626868034716834e-010,

$a_{12,0}$=-4.5010467101454295e-002, $a_{12,1}$=9.4652096952271103e-001, $a_{12,2}$=-3.7223918066824557e-001,
$a_{12,3}$=-2.0759958560079309e-009, $a_{12,4}$=-3.9017659043448758e-010, $a_{12,5}$=1.9822967708575749e-009,

$a_{13,0}$=-4.3242875776603580e-002, $a_{13,1}$=9.9731718722116014e-001, $a_{13,2}$=6.0688857337153193e-002,
$a_{13,3}$=-3.4003929112323072e-009, $a_{13,4}$=-2.2978021794518529e-011, $a_{13,5}$=-7.5532778919304302e-010,

$a_{14,0}$=-3.3716883954380848e-002, $a_{14,1}$=1.0001258995629190e+000, $a_{14,2}$=3.7500419296080839e-001,
$a_{14,3}$=1.7889822638343193e-009, $a_{14,4}$=-6.2592651157127744e-010, $a_{14,5}$=-3.3383808879078198e-009,

$a_{15,0}$=-3.3337185503942837e-002, $a_{15,1}$=1.0819274725734265e+000, $a_{15,2}$=3.3496218434058894e-002,
$a_{15,3}$=4.8130435297607735e-009, $a_{15,4}$=-1.9617688051299804e-014, $a_{15,5}$=-1.2826297843293008e-013.

The variables x_0, …,x_{49} correspond to the delayed output samples in recurrent configuration where x_0 denotes the oldest and x_{49} the most recent sample as described in Section 6.

Acknowledgements

This work has been supported by the Croatian Ministry of Science, Education, and Sports through the project "Computational Intelligence Methods in Measurement Systems" – No. 098-0982560-2565.

References

Ceperic, V., Gielen, G., & Baric, A. (2012), Recurrent sparse support vector regression machines trained by active learning in the time-domain, *Expert systems with applications*, vol. 39, 10933-10942.

Chapra, S. C., & and Canale R. P. (1998), *Numerical Methods for Engineers*, 3rd ed., McGraw-Hill, New York.

Eberhart, R. C., & Kennedy, J. (1995), A new Optimizer Using Particles Swarm Theory, *Proceedings of the Sixt International Symposium on Micro Machine and Human Science* (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43.

Ferrari, S., & Stengel, R. F. (2005), Smooth function approximation using neural 756 networks, *IEEE Transactions on Neural Networks*, vol. 16, no. 1, 24–38.

Holzmann, G., & Hauser, H. (2010), Echo state networks with filter neurons and a delay&sum readout, *Neural Networks*, vol. 23, 244-256.

ISO-20765-1. (2005), *Natural gas – Calculation of thermodynamic properties - Part1: Gas phase properties for transmission and distribution applications*, Ref. No. ISO-20765-1:2005.

ISO-5167-2. (2003), *Measurement of fluid flow by means of pressure differential devices inserted in circular-cross section conduits running full - Part 2: Orifice plates*, Ref. No. ISO-5167-2:2003(E).

Ivakhnenko, A. G. (1971), Polynomial Theory of Complex Systems, *IEEE Transactions on Systems Man, and Cybernetics*, Vol. SMC-1, No.4, 364-378.

Iwasaki, M., Takei, H., & Matsui, N. (2003), GMDH-Based Modeling and Feedforward Compensation for Nonlinear Friction in Table Drive Systems, *IEEE Transactions on Industrial Electronics*, Vol. 50, No. 6, 1172-1178.

Jin, R., Chen, W., & Simpson, T. W. (2000), Comparative studies of metamodeling techniques under multiple modeling criteria**,** *AIAA-2000-4801, AIAA/USAF/ NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 8th, Long Beach, CA.

Levenberg, K. (1944), A method for the solution of certain problems in least squares, *Quarterly of Applied Mathematics*, Vol. 2, 164–168.

Maric, I., & Ivek, I. (2010), Compensation for Joule-Thomson effect in flow rate measurements by GMDH polynomial, *Flow Measurement and Instrumentation*, vol. 21, no. 1, 134-142.

Maric, I., & Ivek, I. (2011), Self-Organizing Polynomial Networks for Time-Constrained Applications, *IEEE Transactions on Industrial Electronics,* vol. 58, no. 5, 2019-2029.

Marquardt, D. (1963), An algorithm for least-squares estimation of nonlinear parameters, *SIAM Journal on Applied Mathematics*, Vol. 11, 431–441.

Nikolaev, N. Y., & Iba, H. (2003), Learning Polynomial Feedforward Neural Networks by Genetic Programming and Backpropagation, *IEEE Transactions on Neural Networks*, Vol. 14, No. 2, 337-350.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007), Training recurrent networks by evolino. *Neural Computation*, 19, 757–779.

Shi, Y. H., & Eberhart, R. C. (1998), A Modified Particle Swarm Optimizer, *Proceedings of International Conference on Evolutionary Computation*, IEEE WCCI, Anchorage, Alaska, 69-73.

Vapnik, V., Golowich, S. E., & Smola, A. J. (1997), Support vector method for function approximation, regression estimation and signal processing, Advances in Neural Information Processing Systems 9, Mit Press, Cambridge, 281-287,.

Witten, I. H., & Eibe, F. (2005), *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, ISBN: 9780120884070, Morgan Kaufmann Publishers.

Xue, Y., Yang, L., & Haykin, S. (2007), Decoupled echo state networks with lateral inhibition, *Neural Networks*, vol. 20, 365-376.

Figure captions

Fig. 1. Illustration of SOPNN construction.

Fig. 2. Hypothetical example of SOPNN.

Fig. 3. Feed forward MLP scheme.

Fig. 4. A schematic diagram of the natural gas flow rate measurement using an orifice plate with corner taps.

Fig. 5. SOPNN model of flow-rate correction factor satisfying the constraint on maximum ET.

Fig. 6. Root relative squared error for MLP and SOPNN models measured on 10 randomly generated test data sets.

Fig. 7. Average RRSE calculated before and after optimization by the LM algorithm of the eight best SOPNN models from each layer

Fig. 8. SOPNN modeled and optimized by using (y_5, n=1,...,400) 50-dimensional training data set. The output from P_{15} is calculated from 50 delayed outputs $x_0,\ldots,x_{49}$, and represents the next predicted output x_{50}. Note that x_0 is the oldest and x_{49} is the most recent output from 50-sample window.

Fig. 9. Illustration of the modeled values and the model error for the first 300 test data samples (y_{5a}, n=401,…,700) obtained by the model SOPNN-LM-y_5 (Fig. 8) with 50 delayed outputs in recurrent configuration: (a) modeled values for y_{5a}, (b) model error for y_{5a}. Note that SOPNN-LM-y_5 is learned on y_5 (25) training data and used to predict y_{5a} (26) test data.

Fig. 10. Illustration of the absolute prediction error in Log scale for the first 3000 test data samples of y_5 and y_{5a}, (n=401,…,3400), obtained by the model SOPNN-LM-y_5 with 50 delayed outputs in recurrent configuration.