Zjavka *Mathematical Sciences* 2013, 7:33
http://www.iaumath.com/content/7/1/33

Mathematical Sciences
a SpringerOpen Journal

ORIGINAL RESEARCH Open Access

Approximation of multi-parametric functions using the differential polynomial neural network

Ladislav Zjavka

Abstract

Unknown data relations can describe a lot of complex systems through a partial differential equation solution of a multi-parametric function approximation. Common artificial neural network techniques of a pattern classification or function approximation in general are based on whole-pattern similarity relations of trained and tested data samples. It applies input variables of only absolute interval values, which may cause problems by far various training and testing data ranges. Differential polynomial neural network is a new type of neural network developed by the author, which constructs and resolves an unknown general partial differential equation, describing a system model of dependent variables. It creates a sum of fractional polynomial terms, defining partial mutual derivative changes of input variables combinations. This type of regression is based on learned generalized data relations. It might improve dynamic system models a standard time-series prediction, as the character of relative data allows to apply a wider range of input interval values than defined by the trained data. Also the characteristics of differential equation solutions facilitate a great variety of model forms.

Keywords: Polynomial neural network, Data relations, Partial differential equation construction, Multi-parametric function approximation, Sum derivative term

Introduction

Solving differential equations are able to define models for a variety of pattern recognition [1] and primarily function approximation problems, applying genetic programming techniques [2] or an artificial neural network (ANN) construction [3]. A common ANN operating principle is based on entire similarity relations of new presented input patterns with the trained ones. A principal lack of its functionality in general is a disability of an input pattern data relation generalization. It utilizes only input variables of absolute interval values, which are not able to describe a wider range of applied data. The ANN generalization from the training data set may be difficult or problematic if the model has not been trained with inputs in the range covered testing data [4]. If the data involve relations, which may become stronger or weaker character, the neural network model should generalize it to be applied also onto different interval values. Differential polynomial neural network (D-PNN) is a new neural network type, which creates and resolves an unknown partial differential equation (DE) of a multi-parametric function approximation. A DE is replaced producing sum of fractional polynomial derivative terms, forming a system model of dependent variables. In contrast with the ANN approach, each neuron of the D-PNN can direct take part in the network total output calculation. Analogous to the ANN function approximation (and pattern identification), the study tried to create a neural network, which function estimation (or pattern recognition) is based on any dependent data relations. In a case a function approximation the output of the neural network is a functional value. In the case a pattern identification its response should be the same to all input vectors, which variables keep up the trained dependencies, no matter what values they become. However the principle of both types is the same, analogous to the ANN approach [5].

$$y = a_0 + \sum_{i=1}^{m} a_i x_i + \sum_{i=1}^{m}\sum_{j=1}^{m} a_{ij} x_i x_j + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m} a_{ijk} x_i x_j x_k + \ldots \qquad (1)$$

Correspondence: lzjavka@gmail.com
VŠB-Technical University of Ostrava Centre of Excellence IT4innovations, Ostrava, Czech republic

m – number of variables $A(a_1, a_2, \ldots, a_m)$, … - vectors of parameters

$X(x_1, x_2, \ldots, x_m)$ - vector of input variables

D-PNN's block skeleton is formed by the GMDH (Group Method of Data Handling) polynomial neural network, which was created by a Ukrainian scientist Aleksey Ivakhnenko in 1968, when the back-propagation technique was not known yet [6]. General connection between input and output variables is possible to express by the Volterra functional series, a discrete analogue of which is Kolmogorov-Gabor polynomial (1). This polynomial can approximate any stationary random sequence of observations and can be computed by either adaptive methods or system of Gaussian normal equations [7]. GMDH decomposes the complexity of a process into many simpler relationships each described by low order polynomials (2) for every pair of the input values.

$$y' = a_0 + a_1x_i + a_2x_j + a_3x_ix_j + a_4{x_i}^2 + a_5{x_j}^2 \tag{2}$$

Partial differential equation construction

The basic idea of the D-PNN is to create and resolve a generally true partial differential equation (3), which is not known in advance and can describe a system of dependent variables, with a special type of fractional multi-parametric polynomials (4), i.e. sum derivative terms.

$$a + \sum_{i=1}^{n} b_i \frac{\partial u}{\partial x_i} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \ldots = 0$$
$$u = \sum_{k=1}^{\infty} u_k \tag{3}$$

$u = f(x_1, x_2, \ldots, x_n)$ – searched function of all input variables

a, $B(b_1, b_2, \ldots, b_n)$, $C(c_{11}, c_{12},\ldots)$ – polynomial parameters

Elementary methods of DE solutions express the solution in special elementary functions – polynomials (e.g. Bessel's functions, Fourier's or power series). Numerical integration of differential equation solutions is based on using:

- rational integral functions
- trigonometric series

Partial DE terms are formed by the adapted application of the method of integral analogues, which replaces mathematical operators and symbols of a DE by ratio of corresponding values. Derivatives are replaced by their integral analogues, i.e. derivative operators are removed and simultaneously all operators are replaced by similarly or proportion signs in equations, all vectors are replaced by their absolute values [8]. However there should be possible to form sum derivative terms replacing a general partial DE (3) by using different math techniques, e.g. wave series and others.

$$y_i = \frac{(a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_{n+1}x_1x_2 + \ldots)^{m+1/n}}{b_0 + b_1x_1 + \ldots}$$
$$= \frac{\partial^m f(x_1, x_2, \ldots, x_n)}{\partial x_1 \partial x_2 \ldots \partial x_m} \qquad Y = \sum_{i=1}^{\infty} y_i = 0 \tag{4}$$

n – combination degree of n-input variable polynomial of numerator

m – combination degree of denominator w_t – weights of terms

The fractional polynomials (4), defining partial relations of n-input variables, represent summation derivative terms (neurons) of a DE. The numerator of eq. (4) is a complete n-variable polynomial, which realizes a new partial function u of formula (3). The denominator of eq. (4) is a derivative part, which gives a partial mutual change of some input variables combination. It arose from the partial derivation of the complete n-variable polynomial in respect to competent combination variables. Root functions of numerator (4) take the polynomials into a correspondent combination degree but needn't be used at all if are not necessary. They may be adapted to enable the D-PNN to generate an adequate range of desired output values.

Multi-parametric function approximation

D-PNN can approximate a multi-parametric function through a general partial sum DE solution (3). Consider first only linear data relations, which describe the DE, e.g. a simple sum function $y_t = x_1 + x_2$ (however it could be any linear function). The network with 2 inputs, forming 1 functional output value $y = f(x_1, x_2)$ should approximate the true function y_t by replacing sum derivative terms of the DE (5). It consists of only 1 block of 2 neurons, terms of both derivative variables x_1 and x_2 (Figure 1).

$$y = w_1 \frac{a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2}{b_0 + b_1x_1} \tag{5}$$
$$+ w_2 \frac{a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2}{b_0 + b_1x_2}$$

D-PNN can be trained with only very small data set (6 samples), involving a wide range of input values *<5,500>*. Figure 2 shows approximation errors (y-axis) of the trained network, i.e. differences of the true and estimated function, to random input vectors with dependent variables. Thus X-axis represents the ideal function.

Output errors can result from some disproportional dependent random vector values (Figure 2), which D-PNN was

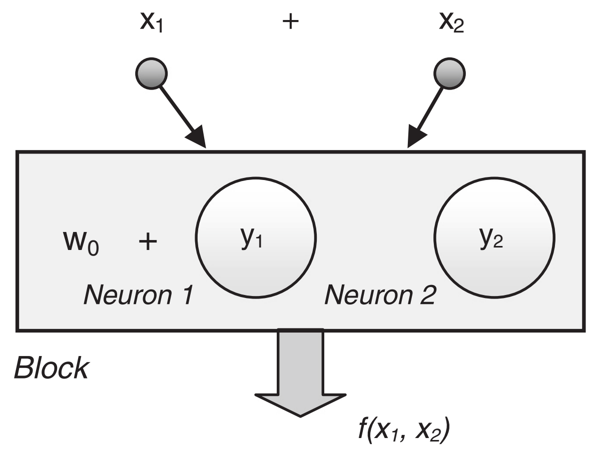

Figure 1 1-block D-PNN.

not trained with, e. g. *360 = 358 + 2*. Figure 2 shows only the results of the *2*-variable 2D-function $f(x_1,x_2)= x_1 + x_2$. The testing random output functions $f(x_1,x_2)$ displayed on *x*-axis (Figure 2) exceed the maximal trained sum value *500*, while the approximation error increases just slowly.

If the number of input variables is increased to 3, the DE composition can apply polynomials of higher combination degree (=3), which results in raising amount of sum derivative terms. The 3-variable D-PNN for linear true function approximation (e.g. $y_t = x_1 + x_2 + x_3$) can contain again 1 block of 6 neurons, DE terms of all 1 and 2-combination derivative variables of the complete DE, e.g. (6)(7).

$$y_1 = w_1 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_2 + \ldots + a_7x_1x_2x_3)^{2/3}}{b_0 + b_1x_1} \tag{6}$$

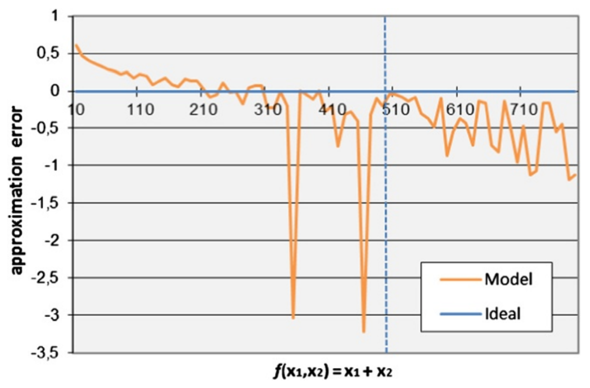

Figure 2 Approximation of the 2-variable function.

$$y_4 = w_2 \frac{a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_2 + \ldots + a_7x_1x_2x_3}{b_0 + b_1x_1 + b_2x_1 + b_3x_1x_2} \tag{7}$$

The training data set of the 3-variable function required an extension (in comparison 2-variables) to enable the D-PNN to get with a desired approximation error. The parameter optimization may apply a proper difference evolution algorithm (EA), supplied with sufficient random mutations to prevent a parameter adjustment convergence before a desired error reaching [9]. Not every experiment succeeds in a functional model.

Multi-layered backward D-PNN

Multi-layered D-PNN, consisting of blocks of neurons, forms composite polynomial functions (functions of functions) (8) in each next hidden layer. Each block contains a single polynomial (without derivative part) forming its output, entranced into the next hidden layer (Figure 3). Neurons don't affect the block output but are applied just directly as the sum derivative terms of a total output calculation (DE composition). The blocks of the 2nd and following hidden layers also form additional extended neurons, i.e. composite terms (CT), which define derivatives of composite functions, applying reverse outputs and inputs of back connected blocks of previous layers. These partial derivatives in respect to variables of previous layers are

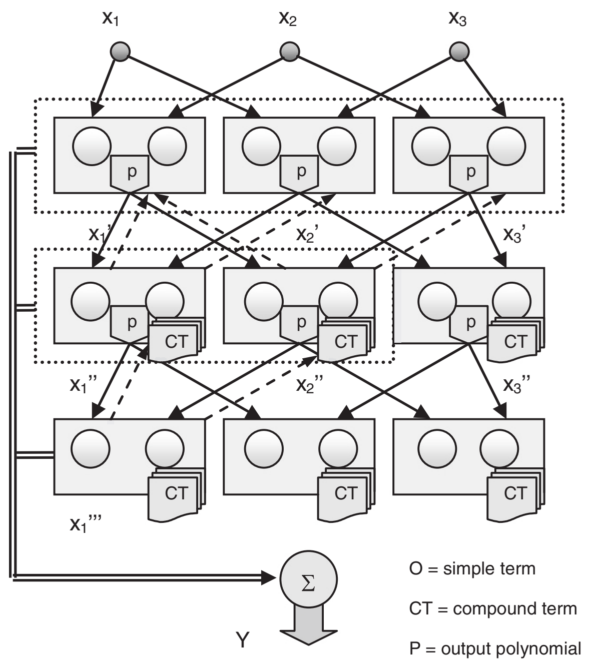

Figure 3 The 3-variable multi-layered 2-combination D-PNN. Back-connections of the 3rd layer 1st block.

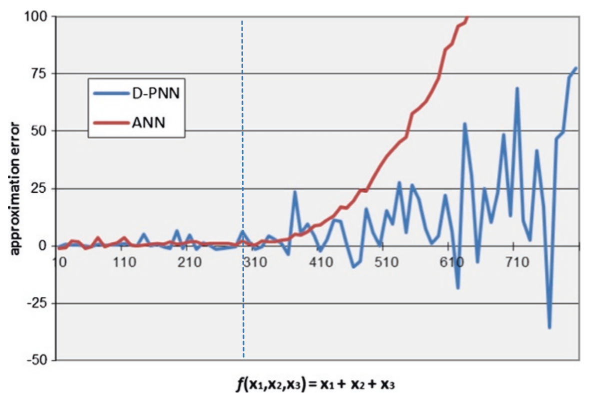

Figure 4 Compare approximation of the 3-variable linear function.

calculated according to the composite function derivation rules (9) (10) and formed by products of partial derivatives of outer and inner functions [10].

$$y_i = \phi_i(X) = \phi_i(x_1, x_2, \ldots, x_n) \quad i = 1, \ldots, m \tag{8}$$

$$\begin{aligned} F(x_1, x_2, \ldots, x_n) &= f(y_1, y_2, \ldots, y_m) \\ &= f(\phi_1(X), \phi_2(X), \ldots, \phi_m(X)) \end{aligned} \tag{9}$$

$$\frac{\partial F}{\partial x_k} = \sum_{i=1}^{m} \frac{\partial f(y_1, y_2, \ldots, y_m)}{\partial y_i} \cdot \frac{\partial \phi_i(X)}{\partial x_k} \quad k = 1, \ldots, n \tag{10}$$

The 1st block of the last (3rd) hidden layer (Figure 3) forms 2 neurons of its own input variables as 2 simple terms (11) of the DE (3). It creates also 4 compound terms of the 2nd (previous) hidden layer, using reverse outputs and inputs of 2 bound blocks in respect to 4 derivative variables (12). As couples of variables of the inner functions can differ from each other, their partial derivations are *0* and so the sum of formula (10) will consist only of 1 term. Thus each neuron of the D-PNN represents a DE term. Likewise compound terms can be created in respect to the 1st hidden layer variables e.g. (13). The 3 back-joint blocks form 8 CT of the DE and this can be well performed by a recursive algorithm.

$$y_1^1 = w_1 \frac{\left(a_0 + a_1 x_1^{,,} a_2 x_2^{,,} + a_3 x_1^{,,} x_2^{,,}\right)^{4/7}}{b_0 + b_1 x_1^{,,}} = w_1 \frac{\left({}^1x_1^{,,,}\right)^{4/7}}{b_0 + b_1 x_1^{,,}} \tag{11}$$

$$y_3^1 = w_3 \frac{\left({}^3x_1^{,,,}\right)^{2/3}}{x_2^{,,}} \Big/ \frac{\left(x_1^{,,}\right)^{2/3}}{b_0 + b_1 x_1^{,}} \tag{12}$$

$$y_7^1 = w_7 \frac{{}^7x_1^{,,,}}{x_2^{,,}} \Big/ \frac{x_1^{,,}}{x_2^{,}} \Big/ \frac{x_1^{,}}{b_0 + b_1 x_1} \tag{13}$$

D-PNN should create a functional value around the desired output. As the input vector variables can take a wide range of values (Figure 2), the combination polynomials produce big output values. Therefore the multiplication (10) was replaced by division operator in fractions of compound terms (11)(12)(13) without an negative effect, reducing the combination degree of composite term polynomials each previous joint layer. Without this modification the root exponents of CT fractions would require an adjustment. The numerator exponents are adapted to the current layer calculation, as the

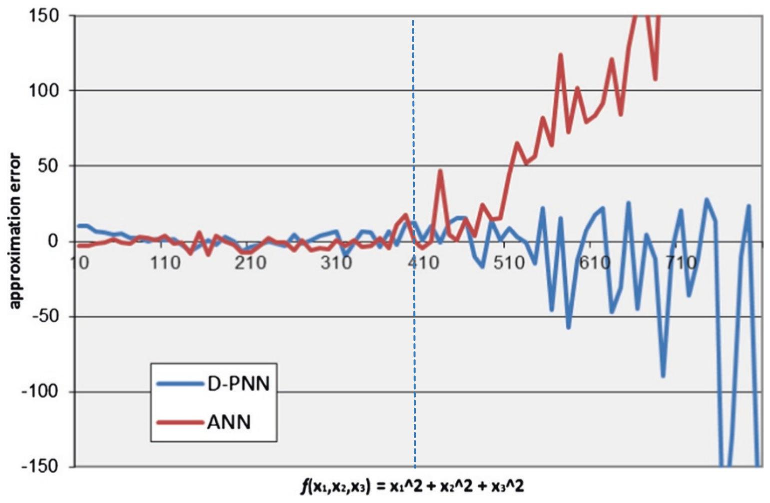

Figure 5 Compare approximation $f(x_1,x_2,x_3) = x_1^2+x_2^2+x_3^2$.

combination degree of polynomials doubles each following hidden layer (11)(12)(13).

Each neuron has an adjustable term weight w_i but not everyone may participate in the total network output calculation (DE composition). The selection of optimal neuron combination can perform easily a proper genetic algorithm (GA) [11]. Parameters of polynomials are represented by real numbers, which random initial values are generated from the interval *<0.5, 1.5>*. They are adjusted with simultaneous GA best-fit neuron combination search in the initial phase of the DE composition. There would be welcome to apply an adequate gradient steepest descent method [12], in conjunction with the EA [13]. D-PNN can be trained with only small input–output data set likewise the GMDH polynomial neural network does [14]. D-PNN's total output Y is the arithmetic mean of all active neuron output values (14) to prevent the neuron amount to influence it.

$$Y = \frac{\sum_{i=1}^{k} y_i}{k} \qquad k = actual\ amount\ of\ active\ neurons \tag{14}$$

Experiments

The presented 3-variable multi-layered D-PNN (Figure 3) is able to approximate any linear function e.g. simple sum $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$. The D-PNN and ANN comparison processes 12 fixed same training data samples. The progress curves are typical of all following experiments (benchmarks) (Figure 4). The approximation accuracy of both methods is co-equal on the trained interval values *<10,300>,* however the ANN approximation ability rapidly falls outside of this range, while the D-PNN alternate errors grow just slowly. The ANN with 2-hidden layers of neurons applied the sigmoidal activation function and the standard back-propagation algorithm. The D-PNN output has typically a wave-like behavior as it model is composed of sum DE terms.

Approximation of non-linear functions requires the extension of the D-PNN block and neuron polynomials (11)(12)(13) with square power variables. Polynomials are the same as applied by the GMDH algorithm (2). Competent square power (16) and combination (17) derivatives form additional sum terms of the 2nd order partial DE (15). The compound neurons of these derivatives are also formed according to the composite function derivative rules [10].

$$F\left(x_1, x_2, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \frac{\partial^2 u}{\partial x_2^2}\right) = 0 \tag{15}$$

where $F(x_1, x_2, u, p, q, r, s, t)$ is a function of 8 variables

$$y_{10} = w_{10}\frac{a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2 + a_5x_1x_2}{b_0 + b_1x_1 + b_2x_1^2} = \frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} \tag{16}$$

$$y_{12} = w_{12}\frac{a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2 + a_5x_1x_2}{b_0 + b_1x_1 + b_2x_2 + b_3x_1x_2} = \frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} \tag{17}$$

Figures 5 and 6 show the D-PNN and ANN compare approximation of some benchmarks - growing non-linear

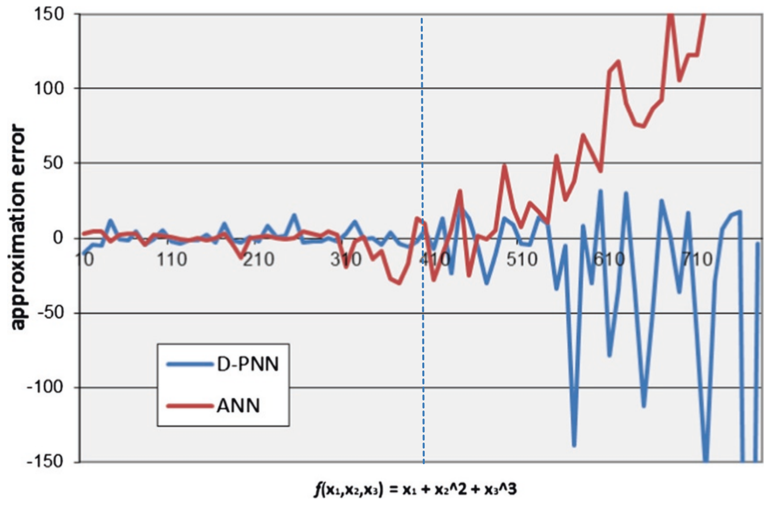

Figure 6 Compare approximation $f(x_1,x_2,x_3) = x_1+x_2^2+x_3^3$.

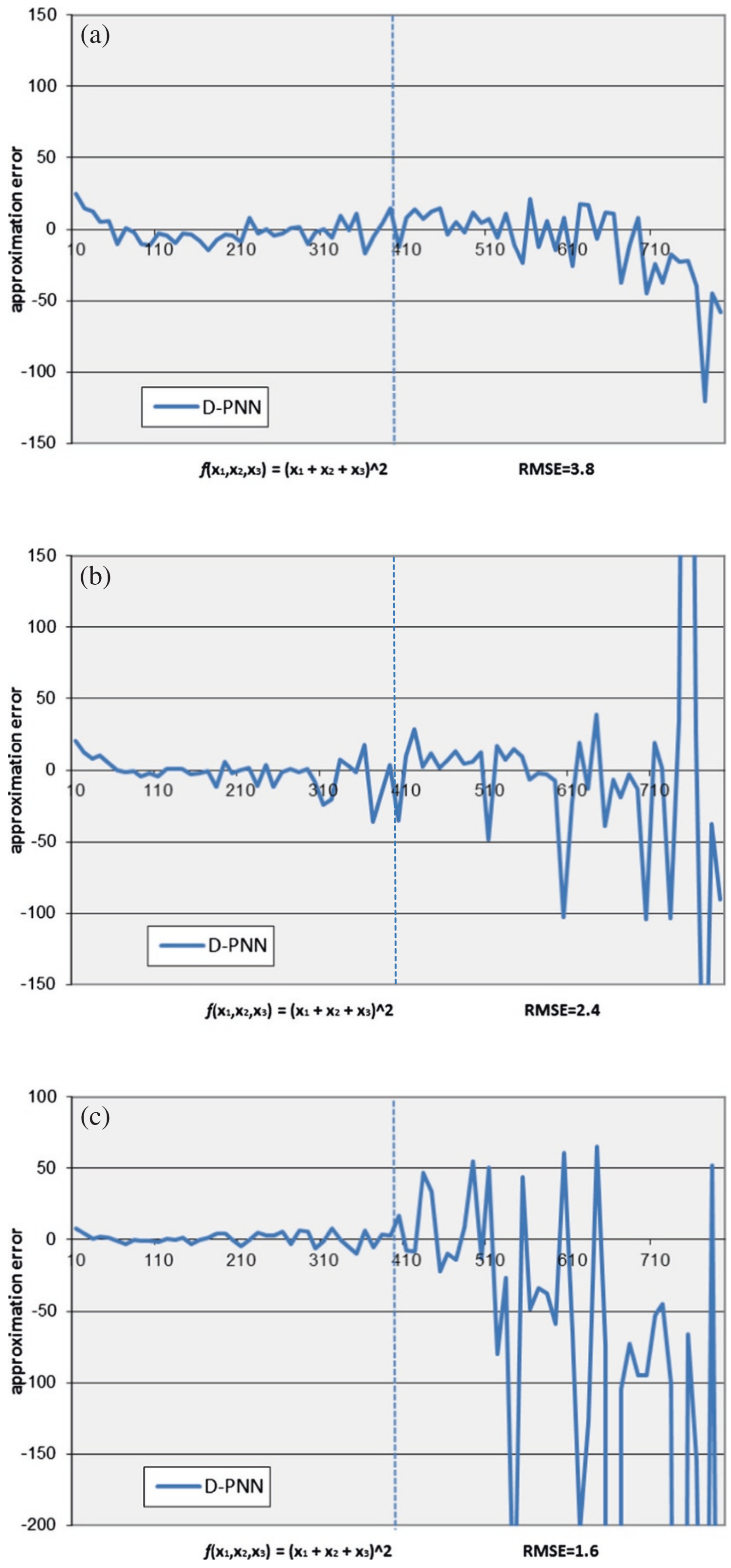

Figure 7 Models of the function $f(x_1,x_2,x_3) = (x_1+x_2+x_3)^2$ with decreasing training errors (a, b, c).

functions. The 24 training data samples were randomly generated by benchmark functions from the interval <*10,400*> for both network models. The parameter and weight adjustment of both methods appeared heavy time-consuming and have not succeed any experiment. The optimal number of the D-PNN's derivative neurons of the non-linear benchmark models was around *100*. Experiments with other benchmarks (e.g. $x_2^2+x_3^3+x_3^4$) result in similar outcome graphs. The D-PNN wavelet output can comprise a wider range of testing data interval values, which was not trained, than ANN. Both method inaccuracies intensify on untrained interval values, as the benchmarks involve power functions. Presented experiments verified the neural network capability to approximate any multi-parametric function, though the operating principles of both techniques differ essentially.

Figure 7a-c show D-PNN models of different final training root mean square errors (RMSE). RMSE decrease results in a lower generalization on untrained data interval values (Figure 7c). Vice versa the greater RMSE evoke the model is valid for a wider data range, while the inaccuracies of the training data interval are overvalued (Figure 7a). This effect became evident of all experiments. As a result the D-PNN should not be trained to a minimal achievable error value to get with the optimal generalization of testing data. The applied incomplete adjustment and selective methods require improvements, which could yield better results. The presented D-PNN operating principle differs by far from other common neural network applied techniques. Benchmark results enable to suppose the D-PNN will succeed forming complex models, which can be defined in the form of multi-parametric functions.

Conclusion

D-PNN is a new type of neural network, which function approximation and dependence of variables identification is based on a generalization of data relations. It does not utilize absolute values of variables but relative ones, which can better describe a wide range of input data interval values. D-PNN constructs a general partial differential equation, which defines a system model of dependent variables, applying integral fractional polynomial sum terms. An acceptable implementation of the sum derivative terms is the principal part of a partial DE substitution. Artificial neural network pattern identification and function approximation models are simpler techniques, based only on whole-pattern similarity relations. A real data example might solve weather forecasts of 1 locality, e.g. static pressure values prediction applying some trained data relations of few nearby localities of surrounding areas. Phases of a constant time interval, of this very complex system could define input vectors of training data set. Estimated multi-parametric function values, i.e. next system states of a selected locality of a time delay form desired network outputs. D-PNN could create better long-time models based on a partial sum DE solution, than a standard ANN time-series prediction (based on entire pattern definitions too).

Competing interests
The author declares that they have no competing interests.

Author's contributions
The author designed a new type of neural network based on GMDH polynomial neural network, which forms its skeleton structure. The proposed new neural network generates a sum of relative derivative polynomial terms as a general differential equation model description. It replaces and resolves the sum partial differential equation as an approximation of an unknown multi-parametric function defined by several discrete point observations. The network was tested with some benchmark functions and compared with artificial neural network models. Real data application models will follow the test experiments.

Acknowledgements
This paper has been elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 supported by Operational Programme'Research and Development for Innovations' funded by Structural Funds of the European Union and state budget of the Czech Republic and supported by the Ministry of Industry and Trade of the Czech Republic, under the grant no. FR-TI1/420, and by SGS, VŠB – Technical University of Ostrava, Czech Republic, under the grant No. SP2012/58.

Received: 31 May 2012 Accepted: 21 June 2013
Published: 24 Jul 2013

References
1. Zhou, B, Xiao-Li, Y, Liu, R, Wei, W: Image segmentation with partial differential equations. Information Technology Journal **9**(5), 1049–1052 (2010)
2. Iba, H: Inference of differential equation models by genetic programming, pp. 4453–4468. Information Sciences, Volume 178, Issue 23, 1 December 2008, Pages (2008)
3. Tsoulos, I, Gavrilis, D, Glavas, E: Solving differential equations with constructed neural networks Neurocomputing, Volume: 72. Issues **10–12**, 2385–2391 (2009)
4. Giles, CL: Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference. Machine Learning **44**, 161–183 (2001)
5. Zjavka, L: Recognition of Generalized Patterns by a Differential Polynomial Neural Network. ETASR - Engineering, Technology & Applied Science Research **2**(1), 167–172 (2012)
6. Ivakhnenko, AG: Polynomial theory of complex systems. IEEE Transactions on systems, Vol. SMC-1, No.4 (1971)
7. Nikolaev, NY, Iba, H: Adaptive Learning of Polynomial Networks. Springer, New York (2006)
8. Kuneš, J, Vavroch, O, Franta, V: Fundamentals of modeling. SNTL Praha, in Czech (1989)
9. Das, S, Abraham, A, Konar, A: Particle swarm optimization and Differential evolution algorithms. Studies in Computational Intelligence 116, 1–38, Springer-Verlag Berlin (2008)
10. Kluvánek, I, Mišík, L, Svec, M, Matematics, I: SNTL Bratislava. , in Slovak (1966)
11. Obitko, M: Genetic algorithms. Hochshule fur Technik und Wirtschaft Dresden (1998). [Online] Available: http://www.obitko.com/tutorials/genetic-algorithms/
12. Nikolaev, NY, Iba, H: Polynomial harmonic GMDH learning networks for time series modeling. Neural Networks **16**, 1527–1540 (2003). Science Direct
13. Zjavka, L: Construction and adjustment of differential polynomial neural network, pp. 40–50. Journal of Engineering and Computer Innovations Vol. 2 Num. 3, March 2011, Academic Journals (2011)
14. Galkin, I: Polynomial neural networks. Materials for UML 91. 531 Data mining course, University Mass Lowell. http://ulcar.uml.edu/~iag/CS/Polynomial-NN.html

10.1186/2251-7456-7-33
Cite this article as: Zjavka: **Approximation of multi-parametric functions using the differential polynomial neural network.** *Mathematical Sciences* 2013, 7:33