Recognition of Generalized Patterns by a Differential Polynomial Neural Network

Ladislav Zjavka
Faculty of Management Science and Informatics
University of Žilina
Žilina, Slovakia
lzjavka@gmail.com

Abstract — **A lot of problems involve unknown data relations, identification of which can serve as a generalization of their qualities. Relative values of variables are applied in this case, and not the absolute values, which can better make use of data properties in a wide range of the validity. This resembles more to the functionality of the brain, which seems to generalize relations of variables too, than a common pattern classification. Differential polynomial neural network is a new type of neural network designed by the author, which constructs and approximates an unknown differential equation of dependent variables using special type of root multi-parametric polynomials. It creates fractional partial differential terms, describing mutual derivative changes of some variables, likewise the differential equation does. Particular polynomials catch relations of given combinations of input variables. This type of identification is not based on a whole-pattern similarity, but only to the learned hidden generalized relations of variables.**

Keywords - polynomial neural network; dependence of variables identification; differential equation approximation; rational integral function

I. INTRODUCTION

The principal disadvantage of the artificial neural network (ANN) identification in general is the disability of input pattern generalization. ANNs can learn to classify any input patterns but utilize only the absolute values of variables. However, the latter may differ significantly while their relations may be the same. That is why ANNs are able to correctly recognize only similar or incomplete patterns compared to the train set. If the input considered is e.g. a shape moved or sized in the input matrix, the neural network identification will fail. An approach to look at the input vector of variables not as a "pattern" but as a dependent bound point set of N-dimensional space could be attempted. A neural network, which would be able to learn and identify any unknown data relations, is to contain a multi-parametric polynomial functions to catch partial dependence of given inputs. Its response would be the same to all patterns (sets) which variables are performed with the trained dependence, regardless of the actual values [9]. Biological neural cell seems to apply a similar principle. Its dendrites collect electrical signals coming from other neurons. But unlike the artificial neuron, some of the signals already interact in single branches (dendrites) of a neural cell (see Figure 1), likewise the multiplied variables of a multi-parametric polynomial do.

Parameters of polynomial terms can represent the synopsis of the cell dendrites. These weighted combinations are summed in the body cell and transformed into relative values using time-delayed dynamic periodic activation functions (the activated neural cell generates series of time-delayed output pulses, in response to its input signals). Axon passes electrical pulse signals on to dendrites of other neural or effector cells [1]. The period of this function depends on some input variables and seems to represent the derivative part of a partial term of a differential equation composition. Differential polynomial neural network (D-PNN) constructs and tries to approximate an unknown differential equation describing relations of input variables that are not entirely patterns. It forms its output as a generalization of input patterns similar to the ones utilized by the human brain. It creates a structural model of any unknown relationships of input variables description. D-PNN is based on GMDH (Group Method of Data Handling) polynomial neural network, which was created by the Ukrainian scientist Aleksey Ivakhnenko in 1968, when the back-propagation technique was not known yet. He attempted to decompose the complexity of a process into many simpler relationships each described by a low order 2-variable polynomial processing function of a single neuron [2].

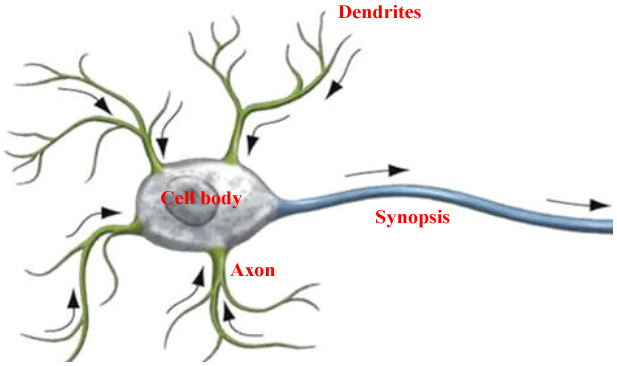

Fig. 1. A biological neural cell

II. DIFFERENTIAL POLYNOMIAL NEURAL NETWORK

The basic idea of the D-PNN is to create and approximate a differential equation (DE) (3), which is not known in advance [3], with a special type of root (power) fractional multi-parametric polynomials (5).

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

$u(x_1, x_2, \ldots, x_n)$ - searched function of all input variables
$a, B(b_1, b_2, \ldots, b_n), C(c_{11}, c_{12}, \ldots)$ - parameters

Fourier's method of partial DE solution searches the solution in a form of the product of 2 functions, of which at least 1 depends only on 1 variable. A partial derivation of function $z(x, y)$ of 2 input variables x, y can be expressed by (4) [4].

$$\frac{\partial z(x,y)}{\partial x} = f_1(x) \cdot f_2(z) \tag{4}$$

Elementary methods of a differential equation solution express the solution in special elementary functions – polynomials (e.g. Bessel's functions, Fourier's power series). Numerical integration of differential equations is based on their approximation through:

- rational integral functions
- trigonometric series

The 1st, and more simple way, has been selected, using the method of integral analogues, which replaces mathematical operators and symbols in DE by the ratio of corresponding variables. Derivatives are replaced by the integral analogues, i.e. derivative operators are removed and simultaneously all operators are replaced by similarly or proportion marks in equations, all vectors are replaced by their absolute values. Dimensional terms are divided by some others, which results in searched non-dimensional likeness criterions [5].

$$y_i = \frac{(a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + a_{n+1} x_1 x_2 + \ldots)^{1/n}}{(b_0 + b_1 x_1 + \ldots)^{1/m}} = \frac{\partial^m f(x_1, x_2, \ldots, x_n)}{\partial x_1 \partial x_2 \ldots \partial x_m} \tag{5}$$

n – combination degree of n-input variable polynomial of numerator
m – combination degree of denominator ($m<n$)

$$Y = w_0 + w_1 \frac{a_0 + \ldots + a_k x_1 x_2 + \ldots}{b_0 + b_i x_1 + \ldots} + \ldots + w_t \frac{a_0 + \ldots + a_p x_i x_j + \ldots + a_q x_i x_j x_k + \ldots}{b_0 + b_i x_1 + \ldots + b_r x_u x_v + \ldots} + \ldots = 0 \tag{6}$$

w_t – weights of terms

The fractional polynomials (5), which can describe a partial dependence of n-input variables of each neuron, are applied as terms of the DE (6) composition. They partly create an unknown multi-parametric non-linear function, which codes relations of input variables. The numerator of (5) is a polynomial of complete n-input combination degree of a single neuron and realizes a new function z of formula (4). The denominator of (5) is a derivative part, which gives a partial mutual change of some neuron input variables and its polynomial combination degree m is less then n. It arose from the partial derivation of the complete n-variable polynomial by competent variable(s).

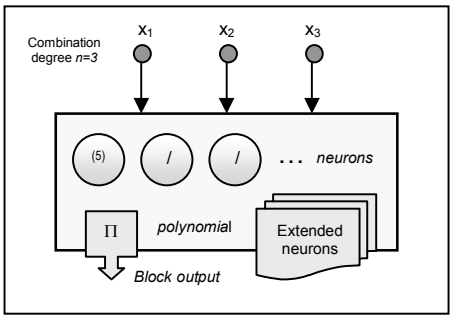

Fig. 2. A block of differential neurons

Each layer of the D-PNN consists of blocks, which contain derivative neurons, one for each fractional polynomial (5), defining the partial derivative dependent change of some input variables. A block also contains additional extended neurons (EN), which form compound functions (15) applying previous layer block outputs. Each block contains a single polynomial (without derivative part), which forms its output entrance into the next hidden layer (Figure 2.). Neurons don't affect the block output but are applied only for the total output calculation (DE composition). Each neuron has 2 vectors of adjustable parameters **a**, **b** and each block contains 1 vector of adjustable parameters of the output polynomial. The root functions of denominators (5) are lower than n, according to the combination degree, which take the polynomials of neurons into competent power degree. They can be replaced by power functions of denominators. Inputs of constant combination degree ($n=2,3,\ldots$) forming particular combinations of variables, enter each block, where they are substituted into polynomials (Figure 3.). It is necessary to adjust not only the polynomial parameters, but also the D-PNN's structure. This means some neurons in terms of role of the DE are to be left out.

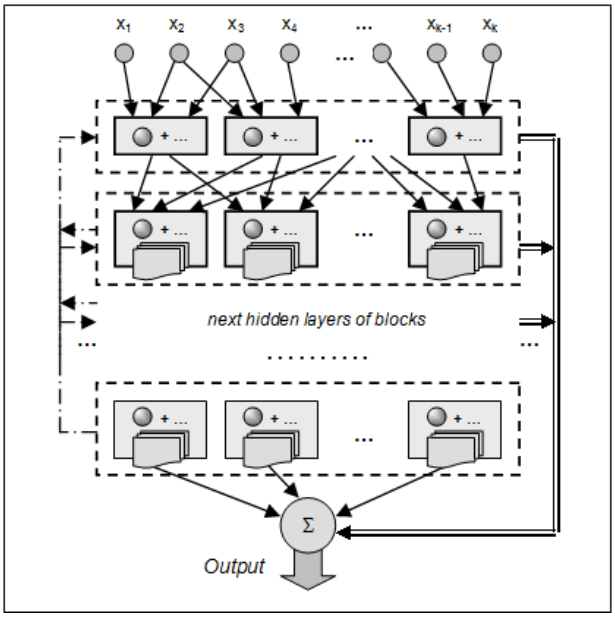

Fig. 3. Differential polynomial neural network

III. IDENTIFICATION OF SIMPLE LINEAR DEPENDENCIES

Consider a very simple dependence of 2-input variables, which multiplicity is constant (e.g. =2). D-PNN will contain only 1 block of 2 polynomial neurons (7)(8) as terms of DE (Figure 4.). As the input variables don't change constantly, it is necessary to add both terms (fractional polynomial of derivative variable x_1 and x_2) in the DE (block). D-PNN will learn this relation easily according to samples of the training data set by means of genetic and evolution algorithm (GA) [7].

$$y_1 = w_1 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2)^{1/2}}{b_0 + b_1x_1} \quad (7)$$

$$y_2 = w_2 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2)^{1/2}}{b_0 + b_1x_2} \quad (8)$$

Consider a more complicated linear dependence, where 2 variables depend on a 3rd. For example sum of the first 2 variables equals the 3rd variable ($x_1 + x_2 = x_3$). The complete DE (for derivatives 1 and 2-combinations of block) consists of 6 terms (neurons) but only 3 of them will be enough for derivative terms x_3 (9), x_1x_3 (10), x_2x_3 (11). If other terms (neurons) are added, the D-PNN will work amiss (see Figure 5). Two-variable combination polynomials of numerators (7)(8) can be also applied, which could improve the D-PNN functionality and increase the number of the DE terms. This 3-variable dependence is described by more complicated exponential functions. The D-PNN as well is charged by the possible 2-sided dependent change of input variables. For example *1+9=10* is the same sum as *9+1=10*. The principal phase of its adjustment resides in eliminating of some neurons (in terms of the DE).

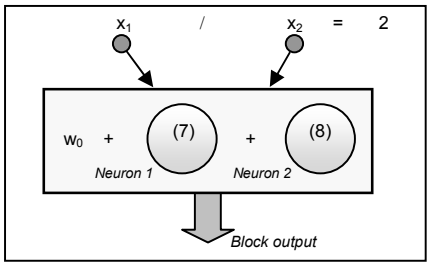

Fig. 4. Identification of a constant quotient of 2 variables ($x_1 = 2x_2$)

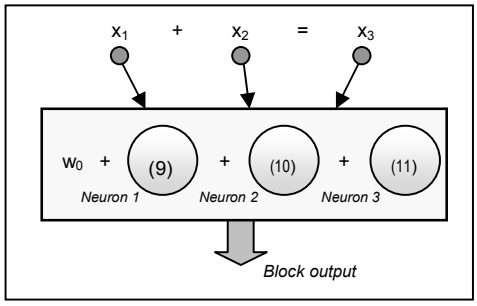

Fig. 5. Identification of the sum dependence

$$y_1 = w_1 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_2 + \ldots + a_7x_1x_2x_3)^{1/3}}{b_0 + b_1x_3} \quad (9)$$

$$y_2 = w_2 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_2 + \ldots + a_7x_1x_2x_3)^{1/3}}{(b_0 + b_1x_1 + b_2x_3 + b_3x_1x_3)^{1/2}} \quad (10)$$

$$y_3 = w_3 \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_1x_2 + \ldots + a_7x_1x_2x_3)^{1/3}}{(b_0 + b_1x_2 + b_2x_3 + b_3x_2x_3)^{1/2}} \quad (11)$$

Multi-layered D-PNN creates compound polynomial functions. Main exponential functions of higher layers "carry" some secondary functions of previous layers, describing the partial relations of its variables. From mathematical point of view the 1st hidden layer forms the inner functions, which substitute the input variables of 2nd hidden layer neuron and block polynomials - the outer functions. Provided this assumption we are able to calculate the partial derivatives of compound functions by variables of previous layers as DE terms (14), from the inner functions (12) of an outer function (13). These compound DE terms are formed as products of partial derivatives of main and inner functions (15) [6].

$$y_i = \varphi_i(X) == \varphi_i(x_1, x_2, \ldots, x_n) \qquad i=1, \ldots, m \quad (12)$$

$$F(x_1, x_2, \ldots, x_n) = f(y_1, y_2, \ldots, y_m) = f(\phi_1(X), \phi_2(X), \ldots, \phi_m(X)) \quad (13)$$

$$\left.\begin{aligned} \frac{\partial F}{\partial x_1} &= \frac{\partial f}{\partial y_1}\cdot\frac{\partial \varphi_1}{\partial x_1} + \frac{\partial f}{\partial y_2}\cdot\frac{\partial \varphi_2}{\partial x_1} + \ldots + \frac{\partial f}{\partial y_m}\cdot\frac{\partial \varphi_m}{\partial x_1} \\ \frac{\partial F}{\partial x_2} &= \frac{\partial f}{\partial y_1}\cdot\frac{\partial \varphi_1}{\partial x_2} + \frac{\partial f}{\partial y_2}\cdot\frac{\partial \varphi_2}{\partial x_2} + \ldots + \frac{\partial f}{\partial y_m}\cdot\frac{\partial \varphi_m}{\partial x_2} \\ &\cdots\cdots\cdots \\ \frac{\partial F}{\partial x_n} &= \frac{\partial f}{\partial y_1}\cdot\frac{\partial \varphi_1}{\partial x_n} + \frac{\partial f}{\partial y_2}\cdot\frac{\partial \varphi_2}{\partial x_n} + \ldots + \frac{\partial f}{\partial y_m}\cdot\frac{\partial \varphi_m}{\partial x_n} \end{aligned}\right\} \quad (14)$$

$$\frac{\partial F}{\partial x_k} = \sum_{i=1}^{m} \frac{\partial f(y_1, y_2, \ldots, y_m)}{\partial y_i} \cdot \frac{\partial \phi_i(X)}{\partial x_k} \qquad k=1, \ldots, n \quad (15)$$

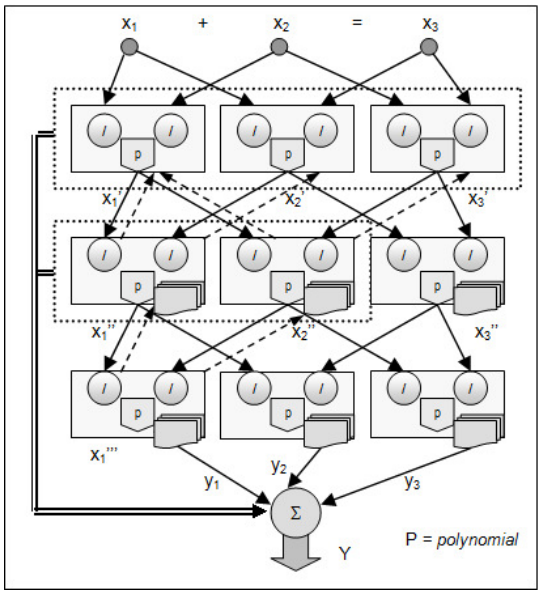

Fig. 6. Identification of the 3-variable dependence with 2-combination blocks

Each block of the D-PNN forms partial DE terms utilizing its basic and extended neurons. Single adjustable polynomial (P in Figure 6.) without derivative part creates the block output (applying in the next hidden layer) but the neurons are applied only for the total DE composition. The blocks of the 2nd and the following hidden layers create compound terms (CT) of the DE using their additional extended neurons, outputs and inputs of back connected blocks of previous layers. Consider for instance the 1st block of the last hidden layer, which takes its own neurons as 2 basic terms (16) of the DE (6). Subsequently it creates 4 extended terms of the 2nd (previous) hidden layer variables, using reverse output polynomials and inputs of 2 bound blocks. It creates 4 fractional compound terms of the DE for 4 derivative input variables of previous hidden layer using derivations of compound and inner functions (17). As couples of variables of the inner functions ϕ_1 *(x1, x2)* and ϕ_1 *(x3, x4)* differ from each other, their partial derivations are = 0 and so the sum (15) will consist only of 1 term.

$$y_1^1 = w_1 \frac{\left(a_0 + a_1 x_1'' a_2 x_2'' + a_3 x_1'' x_2''\right)^{1/2}}{2 \cdot \left(b_0 + b_1 x_1''\right)} = w_1 \frac{\left({}^1x_1\right)^{1/2}}{2 \cdot \left(b_0 + b_1 x_1''\right)} \tag{16}$$

$$y_3^1 = w_3 \frac{({}^3x_1''')^{1/2}}{x_2''} \cdot \frac{(x_1'')^{1/2}}{2 \cdot \left(c_0 + c_1 x_1'\right)} \tag{17}$$

$$y_7^1 = w_7 \frac{({}^7x_1''')^{1/2}}{x_2''} \cdot \frac{(x_1'')^{1/2}}{x_2'} \cdot \frac{(x_1')^{1/2}}{2 \cdot \left(d_0 + d_1 x_1\right)} \tag{18}$$

The previous layer block reverse outputs are used to create necessary partial derivations of the outer and inner functions (of polynomials) of differential neurons (17). Likewise compound terms can be created for the 1st hidden layer (18). The 3 linked blocks forming 8 terms of the DE were attached to the presently adjusted block. This can be performed well by a recursive algorithm. It was not every term that was used in the complete DE; some of them were necessarily left out. This indicates "0" or "1" in the neurons of blocks and is ease to use them as genes of GA. Parameters of polynomials are represented by real numbers. A chromosome is a sequence of their values, which can be easy mutated. The D-PNN's total output *Y* is the sum of all active partial DE term values according to (19), which the present active amount *a* can be built in.

$$Y = \frac{\sum_{i=1}^{k} y_i}{a} \qquad k = \textit{total amount of DE terms} \tag{19}$$

It can be seen, that the 3-variable D-PNN (Figure 6.) substantially consists of 3 overlaying "wedge" networks (WN), each going back out from the blocks of the last hidden layer and gradually attaching to the derivative variables of previous layers. The D-PNN of the 4 dependent input variables using 2-combination blocks will have totally 6 blocks of all input combination couples in the 1st hidden layer. The number of combinations for all variables increases enormously each next hidden layer. This could be solved by applying WNs, as only some of the blocks are created and used. The total amount of D-PNN's hidden layers could equal at least to the number of input variables (i.e. 4), as it must be able to create each combination of which and to reach back all derivative variables of the 1st layer. So WNs of the 1st hidden layer will involve min. 4 random blocks, consequently in the 2nd layer will contain min. 3 blocks, etc. This way the number of all WN blocks decreases each next hidden layer until is reached just 1 block. D-PNN will have several overlaying WNs partly in the layers again. Some WN layers overlay each other and so the blocks can be used several times by different WNs (Figure 7.). The blocks of the 2nd and following hidden layers can be reconnected and this could compensate missing combination blocks. The connections of the complete 1st hidden layer blocks are fixed. Likewise the previous 3-variable D-PNN type does, it can construct the partial fractional terms of the DE from back-connected blocks of previous layers. All WN blocks attach back gradually the derivative variables of previous layers. The searching space contains a great amount of local error solutions, which GA can finish easily. This problem is caused by a lot of possible combinations of block inputs and composed DE terms (only some of them may be employed), which selection is a critical phase of the D-PNN's construction, besides the simultaneous parameter adjustment [10].

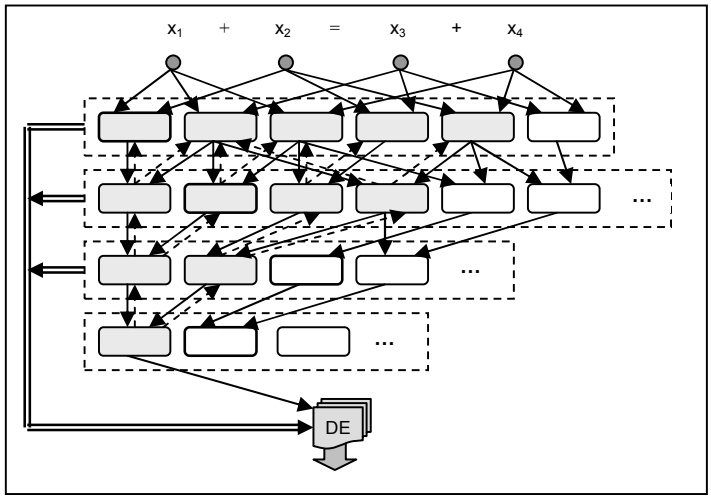

Fig. 7. "Wedge" networks of the 4-variable 2-combination D-PNN

The D-PNN of the 6 dependent input variables using 2-combination blocks will have totally 15 blocks of all input combination couples in the 1st hidden layer and 6 hidden layers. However in experiment with right triangles (Figure 9 and Figure 10.) it could be sufficient with 4 hidden layers again, because there is the maximum of 4-variable dependence to identify.

IV. EXPERIMENTS

4-variable D-PNN is able to identify row/column or diagonal dependence of chess pieces (Figure 8.). Input vector is formed by their *x, y* positions (*row, column*). If the white rook checks the black bishop their *x* or *y* positions equal and this can D-PNN learn to identify. Another relation occurs if the black bishop checks the white rook, the sum or difference of their *x* and *y*-positions are equal $A_x+A_y=B_x+B_y$ or $A_x-A_y=B_x-B_y$. Table 1 and Table 2 show network responses to dependent and

independent variables of input vectors. Training data set can consist of following 5 data samples *x* and *y*-positions of the chess pieces (the chessboard was enlarged) [9]:

{1+22=20+3}, {16+1=2+15}, {34+3=30+7}, {60+30=42+48}, {5+25=2+28}, {2+5=4+3}

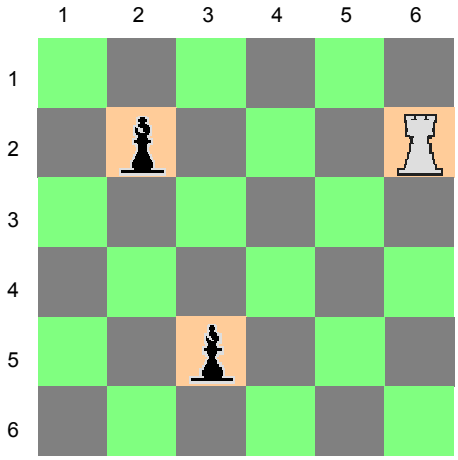

Fig. 8. Relations of chess pieces

TABLE I. RESPONSES TO RANDOM INPUT VECTORS WITH 4 DEPENDENT VARIABLES

Input vector	Output
31 + 46 = 11 + 66	0.9654
18 + 57 = 24 + 51	1.0273
46 + 30 = 25 + 51	1.0018
58 + 44 = 17 + 85	1.0006
14 + 25 = 1 + 38	0.9882
4 + 23 = 8 + 19	1.0031

TABLE II. RESPONSES TO RANDOM INDEPENDENT INPUT VECTORS OF 4 VARIABLES

Input vector	Output	Input vector	Output
46 + 55 = 3 + 108 (-10)	1.2041	25 + 45 = 51 + 9 (+10)	0.8461
11 + 55 < 27 + 49	1.1865	57 + 54 > 43 + 58	0.9564
56 + 49 < 5 + 110	1.1213	15 + 42 > 12 + 35	0.9179
8 + 29 < 21 + 26	1.1587	49 + 47 > 52 + 34	0.9162
45 + 34 < 2 + 87	1.1917	31 + 32 > 37 + 16	0.8704
3 + 39 < 26 + 26	1.4803	45 + 56 > 30 + 61	0.9515
35 + 9 < 18 + 36	1.2038	27 + 58 > 45 + 30	0.9228
29 + 6 < 37 + 8	1.1653	44 + 34 > 36 + 32	0.9267
60 + 43 < 41 + 72	1.0526	39 + 25 > 30 + 24	0.9174
5 + 40 < 40 + 15	1.2457	59 + 39 > 51 + 37	0.9414

A separating plane could be noticed, detached from the relative "classes", which have the same characteristic (if the sum of the 1st couple is less then it should be, the output is less then the desired round and other hand round). D-PNN can be trained only with small input-output data samples (likewise the GMDH polynomial neural network does) to learn any dependence [8].

6-variable D-PNN can learn to identify (generalize) a changeable shape (e.g. triangle) regardless of its size or position in the input matrix (Figure 9.). Input vector of the D-PNN is formed by *x, y* (*row, column*) coordinates of the 3 triangle apexes *A, B, C*. The dependence of points *A, C* is diagonal, *A, B* vertical and *B, C* horizontal. As there are simultaneously occurred 3 types of point relations, it is necessary to increase the number of blocks of D-PNN's hidden layers. Training data set can consist of following 5 data samples – right equal-based triangles :

{6,44,11,44,11,49}, {22,13,32,13,32,23}, {10,30,25,30,25,45}, {3,20,23,20,23,40}, {25,50,50,50,50,75}

Testing random right triangles must keep the apexes *A, B, C* dependent to be correctly recognized by D-PNN. Table 3 shows responses of trained network to dependent right triangles. Table 4. applies only vertical deformations (+ and –) of right triangles in *C* apexes (Fig.10. down), to be shown a transparent separating plane, detaching the relative triangles.

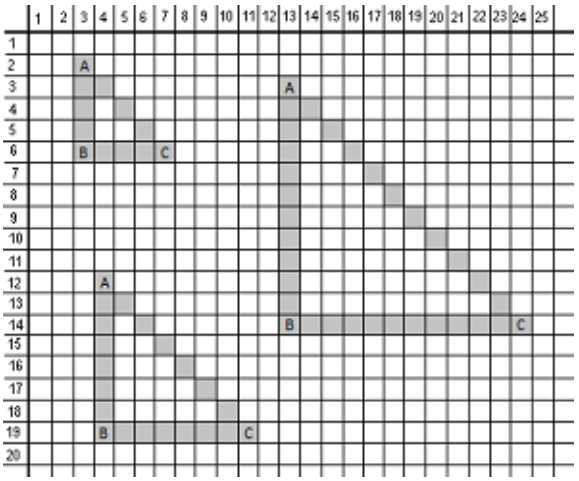

Fig. 9. Dependent right equal-sided triangle shapes

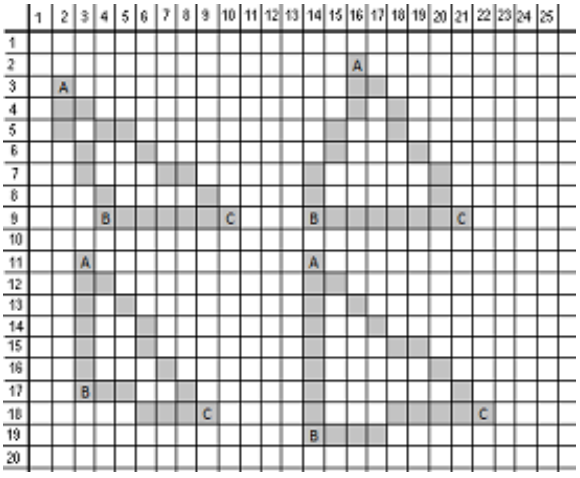

Fig. 10. Independent deformed right triangles in apexes A and C

TABLE III. RESPONSES TO RANDOM INPUT VECTORS WITH 6 DEPENDENT VARIABLES (RIGHT TRIANGLES)

Input vector (A_x,A_y B_x,B_y C_x,C_y)	Output
34,44 60,44 60,70	1.0142
18,5 35,5 35,22	0.9692
15,39 35,39 35,59	0.9932
16,51 58,51 58,93	1.0095
29,51 45,51 45,67	1.0227
21,5 29,5 29,13	1.067

TABLE IV. RESPONSES TO RANDOM INDEPENDENT INPUT VECTORS OF 6 VARIABLES (DEFORMED RIGHT TRIANGLES IN C APEXES)

Input vector (A B C)	Output	Input vector (A B C)	Output
13,36 42,36 33,65	0.8750	24,15 34,15 37,25	1.0713
21,27 59,27 47,65	0.8803	4,11 40,11 52,47	1.2376
9,10 34,10 26,35	0.8605	48,40 56,40 58,48	1.1255
12,43 25,43 21,56	0.9096	32,22 57,22 65,47	1.0885
35,10 57,10 50,32	0.9523	18,28 25,28 27,35	1.0558
2,12 28,12 20,38	0.9037	5,19 51,19 53,25	1.1906
6,43 42,43 30,79	0.8806	6,6 36,6 46,36	1.1850
16,35 48,35 38,67	0.8766	20,13 34,13 38,27	1.0477
13,48 16,48 15,51	0.9835	18,45 30,45 34,57	1.0857
10,8 49,8 36,47	0.9372	18,38 62,38 76,82	1.1325

V. DISCUSSION

Only linear dependencies of variables have been assumed for simplicity in the examples presented. If there is an occurrence of a non linear dependence of the input data, the square power exponent variables would extend combination polynomials of neurons and applied also as competent derivative terms (20). The denominators of (20)(21) result from the partial derivatives of the complete DE term polynomials of numerators. The root square (or power) functions are likely not involved into fractions (21), if the differences of values of input variables are not too big (in case a real data model) as occurred in presented examples.

$$y_i = \frac{(a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2 + a_5x_1x_2 + a_6x_1^2x_2 + a_7x_1x_2^2 + a_8x_1^2x_2^2)^{1/4}}{(b_0 + b_1x_1 + b_2x_2 + b_5x_1x_2)^{1/2}} = \frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2} \quad (20)$$

$$y_j = \frac{a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2 + a_5x_1x_2}{b_0 + b_1x_2 + b_2x_2^2} = \frac{\partial^2 f(x_1, x_2)}{\partial x_1^2} \quad (21)$$

According to (12)-(15) it is possible to define higher degree partial derivations of 2-variable compound function $F(x,y)=f(u,v)$ (22) [6]. As the variables of the D-PNN's inner functions $u=\phi(x_1,y_1)$ and $\psi=(x_2,y_2)$ are different the 2nd , 3rd and 5th terms of eq. (23) are = *0* (as the partial derivation of ψ by x_1 is = *0*). Likewise the terms of (24) and (25) do [6].

$$F(x,y) = f(u,v) = f[\varphi(x,y), \psi(x,y)] \qquad u=\phi(x,y)\ v=\psi(x,y) \quad (22)$$

$$\frac{\partial^2 F}{\partial x^2} = \frac{\partial^2 f}{\partial u^2}\left(\frac{\partial \varphi}{\partial x}\right)^2 + 2\frac{\partial^2 f}{\partial u \partial v}\frac{\partial \varphi}{\partial x}\cdot\frac{\partial \psi}{\partial x} + \frac{\partial^2 f}{\partial v^2}\left(\frac{\partial \psi}{\partial x}\right)^2 + \frac{\partial f}{\partial u}\cdot\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial f}{\partial v}\cdot\frac{\partial^2 \psi}{\partial x^2} \quad (23)$$

$$\frac{\partial^2 F}{\partial y^2} = \frac{\partial^2 f}{\partial u^2}\left(\frac{\partial \varphi}{\partial y}\right)^2 + 2\frac{\partial^2 f}{\partial u \partial v}\frac{\partial \varphi}{\partial y}\cdot\frac{\partial \psi}{\partial y} + \frac{\partial^2 f}{\partial v^2}\left(\frac{\partial \psi}{\partial y}\right)^2 + \frac{\partial f}{\partial u}\cdot\frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial f}{\partial v}\cdot\frac{\partial^2 \psi}{\partial y^2} \quad (24)$$

$$\frac{\partial^2 F}{\partial x \partial y} = \frac{\partial^2 f}{\partial u^2}\cdot\frac{\partial \varphi}{\partial x}\cdot\frac{\partial \varphi}{\partial y} + \frac{\partial^2 f}{\partial u \partial v}\left(\frac{\partial \psi}{\partial x}\frac{\partial \varphi}{\partial y} + \frac{\partial \varphi}{\partial x}\frac{\partial \psi}{\partial y}\right) + \frac{\partial^2 f}{\partial v^2}\cdot\frac{\partial \psi}{\partial x}\cdot\frac{\partial \psi}{\partial y} + \frac{\partial f}{\partial u}\cdot\frac{\partial^2 \varphi}{\partial x \partial y} + \frac{\partial f}{\partial v}\cdot\frac{\partial^2 \psi}{\partial x \partial y} \quad (25)$$

A real data example might solve the weather forecast based on some trained data relations, which are used for calculating the next state of a system. Let's take several types of variables (e.g. pressure, damp, temperature) partly describing states of this very complex system. The input vector of the D-PNN is formed by values of these variables in defined matrix coordinates of a meteorological map. The training data set includes definite states of a time interval and desired network outputs. The output could mark the weather forecast as "rainfalls"= 1, "cloudy" = 2, "sunshine" = 3. There can naturally arise possible transient states (e.g. 1.4). The output is computed for 1 locality of the map and could also predict the atmospheric pressure or another quantity.

VI. CONCLUSION

Artificial neural networks in general respond to related patterns with a similar output. They identify input patterns on the bases of their relationship. Likewise, the identification of unknown dependencies of the data variables could also be considered. This could be regarded as a pattern of abstraction, similar to that utilized by the human brain, which applies the approximation with time-delayed periodic activation functions of biological neurons in high dynamic system of behavior. D-PNN is a new type of neural network, which performs identification based on any unknown generalized relations of input variables. D-PNN forms its functional output as a composition of differential equation terms (which describe a system of dependent variables) from rational integral functions. The problem of the multi-layered D-PNN construction reside creates every partial combination term for a complete DE in utilizing some fixed low combination degrees (2, 3), while the amount of variables is as a rule higher.

REFERENCES

[1] Ľ. Beňušková, Neuron and brain. Cognitive sciences, Calligram Bratislava, 2002 (in Slovak).

[2] A.G. Ivakhnenko, "Polynomial theory of complex systems", IEEE Transactions on systems, Vol. SMC-1, No. 4, pp. 364-378, 1971.

[3] J. Hronec, Differential equations II., SAV Bratislava, 1958 (in Slovak).

[4] R. Rychnovský, J. Výborná, Partial differential equations and some of their solutions, Publ. SNTL Praha, 1970 (in Czech).

[5] J. Kuneš, O. Vavroch, V. Franta, Principles of modelling, SNTL Praha, 1989 (in Czech).

[6] I. Kluvánek, L. Mišík, M. Švec, Matematics I., II., SNTL Bratislava, 1966 (in Slovak).

[7] S. Das, A. Abraham, A. Konar, "Particle swarm ptimization and differential evolution algorithms: Technical snalysis, applications and hybridization perspectives", Computer and Information Science, Vol. 38, pp. 1-38, 2008.

[8] B. B. Misra, S. Dehuri, P.K. Dash, G. Panda, "A reduced and comprehensible polynomial neural network for classification", Pattern recognition letters, Vol. 29, No. 12, pp. 1705-1715, 2008.

[9] L. Zjavka, "Generalization of patterns by identification with polynomial neural network", Journal of Electrical Engineering, Vol. 61, No. 2, pp. 120-124, 2010

[10] L. Zjavka, "Construction and adjustment of differential polynomial neural network", Journal of Engineering and Computer Innovations, Vol. 2, No. 3, pp. 40-50, 2011