Machine Learning with Genetic Multivariate Polynomials

ANGEL FERNANDO KURI-MORALES
Departamento de Computación
Instituto Tecnológico Autónomo de México
Río Hondo No. 1
México 01000, D.F.
MÉXICO

Abstract. We present an algorithm which is able to adjust a set of data with unknown characteristics. The algorithm is based on the simple notion that a sufficiently large set of examples will adequately embody the essence of a numerically expressible phenomenon, on the one hand, and, on the other, that it is possible to synthesize such essence via a polynomial of the form $F(V_1,...,V_n) = \sum_{I_1=0}^{D_1} \dots \sum_{I_n=0}^{D_n} \mu_{I1...In} C_{I1...In} V_{I1}^{D1} \dots V_{In}^{In}$ where the $\mu_{I1...In}$ may only take the values 0 or 1 depending on whether the corresponding monomial is adequate. In order to determine the adequateness of the monomial in case we resort to a genetic algorithm which minimizes the fitting error of the candidate polynomials. We analyze a set of selected data sets and find the best approximation polynomial. We compare our results with those stemming from multi-layer perceptron networks trained with the well known backpropagation algorithm (BMLPs). We show that our Genetic Mutivariate Plynomials (GMPs) compare favorably with the corresponding BMLPs without the "black box" characteristic of the latter; a frequently cited disadvantage. We also discuss the minimization (genetic) algorithm (GA).

Keywords. Multivariate approximation, multi-layered perceptron networks, genetic algorithms.

1 Introduction

Machine learning has been tackled in the past using (with remarkable success) the so-called Neural Networks (NNs). These statistical algorithms have the undeniable advantage of being able to adjust a set of data with unknown characteristics. Networks of simple computational elements called "perceptrons" have been shown to be able to compute an arbitrary function from a sufficiently large set (sample) of examples (elements). In particular, well behaved (typically continuous) functions are known to be amenable to optimal representation (in terms of statistical learning theory [VV95]) by a multi-layered perceptron network (typically trained with the "backpropagation" optimization algorithm) with no more than three layers (BMLPs) [SH99]. However, two often cited disadvantages of these algorithms are a) The number of neurons in the hidden layer has to be estimated heuristically and b) The resulting BMLP has no explanatory properties (i.e. it looks like a "black box" to the user and does not lend itself to simple rule extraction). To ameliorate these shortcomings, alternative neural network (NN) paradigms have been explored; for instance Radial Basis Function Networks (RBFs) and Support Vector Machines (SVMs). In both of these latter cases the architecture is dictated by the method itself and does not have to be estimated. In both RBFs and SVMs, however, the problem is transferred elsewhere. In RBFs we are faced with having to find the most adequate function centers while in SVMs we have to determine the regularization parameter "C". On the other hand, both RBFs and SVMs still retain the black box characteristic of most NNs. The power of NNs, in general, resides in the proper combination of "simple" computing elements (the "neurons") in a network which is, basically, and embodiment of the "divide-and-conquer" principle. In BMLPs the neurons are called "perceptrons" and, simply put, are summation elements whose output is post-processed by a non-linear (typically sigmoid) function such as $1/(1+e^{-x})$ (the logistic function) or $(e^x - e^{-x})/(e^x + e^{-x})$ (*tanh*: the hyperbolic tangent function)[1]. These functions are usually called the *activation functions*. For further reference, we will index the linear, logistic and tanh functions with the numbers 1, 2 and 3 respectively. Each of the inputs to any given perceptron is weighted in such a way that the concerted interaction of the elements of the NN yields an output which mimics the desired function. In figure 1 we show an example of a BMLP.

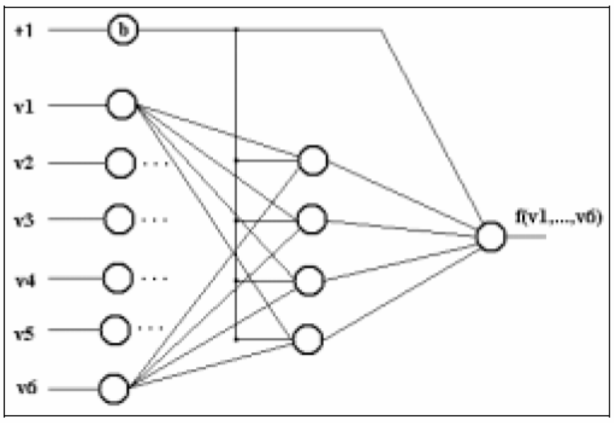

Fig. 1. A Multi-Layered Perceptron Network

This BMLP has 6 inputs, 4 intermediate (hidden) units and one output. In addition to the input neurons corresponding to each of the input variables a so-called bias neuron is included. The bias neuron relieves the designer from having to individually determine the threshold for each neuron. Its input is usually set to "+1". We denote the number of neurons in the input layer with "I", the number of neurons in the hidden layer with "H" and those in the output layer with "O". Therefore, there are C = H(I+O+1)+O connections in any three (as here) layered perceptron network; one weight corresponds to each one of the connections. A BMLP is "given" a set of *n* input values (a vector **X** $=(x_1,...,x_n)$) and a set of output values (a vector **Y** = $(y_1,\ldots,y_p)$[2]). The process of "training" a BMLP consists of finding the C values of the weights such that the NN, given **X**, outputs a value as similar to the original values of **Y** as possible according to a predefined error measure[3]. Furthermore, it is usual to impose the further requirement that the BMLP is trained with the so-called *training set* while minimizing the approximation error for a data set which the NN does not "see" (which we call the *test set*). This last condition is aimed at providing *generalization* properties to the resulting NN: i.e. the trained NN should also synthesize the phenomenon under analysis outside the limited range of the known data. To this effect the original data set is split by randomly leaving out of the original sample a given percentage of elements which constitute the (smaller) test set; the remaining elements constitute the (larger) training set[4]. The process of validating the generalization properties of the NN may be improved upon by trying out different combinations of training-test sets and is usually referred to as *cross-validation*[5].

Finding the weights of an arbitrary BMLP may be seen as a combinatorial problem. Let us take the NN of figure 1, as an example. There we have I = 6, H = 4 and O =1, that is C = 33. If we assume that every weight is expressed with an 80 bit binary number (as in the floating point unit of a typical Pentium processor) we need 33 x 80 = 2640 bits to express the values of the trained NN. To find the very best combination, therefore, we would need to exhaustively try out 2^{2640} combinations. This is a huge number (close to $(10^3)^{264} = 10^{792}$ combinations). To have a feeling of its enormity, consider that the estimated number of elementary particles in the observable universe is 10^{80}; thus, we would need close to 10 times the number of elementary particles in the universe merely to store the possible solutions if we were able to assign one of the possible sets of trained neuron weights to every particle! It is, indeed, a tribute to the efficiency of the backpropagation algorithm that the task of finding an adequate set of weights (**W**) of size C is routinely performed in a modern computer in a few seconds [BB92].

Given the above, therefore, we may see that a properly trained BMLP represents the synthetic representation of the data sample. We may, then, "replace" a large sample set by a) A description of the architecture of the NN (for instance (6,4)), b) A specification of the activation functions (for instance (1, 2, 1)) and c) **W**. In the example, therefore, the vector **F** = [(6,4); (1,2,1); **W**] which consists of 38 numbers will, in principle, adequately synthesize the information of the data sample. Not only that but, if we consider the cross-validation process to be valid, the

[1] Another typical choice is the linear function y = x. Other, more exotic, choices (such as *Gaussian* functions and the like) are possible but we do not consider them here.

[2] In what follows we assume that $|\mathbf{Y}| = 1$.

[3] Typically, the error measure corresponds to the least squares norm.

[4] We assume, hereinafter, that 75% of the simple elements are assigned to the training set and the remaining elements to the test set.

[5] We shall adhere to the simplest form of cross-validation as described above.

same set will represent the observed phenomenon even for data *beyond* the sample.

It seems, then, that having to somehow (typically with heuristics) determine "H" and being unable to establish an explicit expression of the synthesizing function is a small price to pay for such a powerful tool. However, there is an alternative and viable tool (GMP) which is free of both shortcomings and is the main topic of this paper. In what follows we give a brief account of the GMP algorithm's principles and its behavior in a set of selected problems. In part 2 we describe the algorithm; in part 3 we describe experiments allowing us to compare BMLPs with GMPs; in part 4 we discuss their relative characteristics and offer our conclusions. Finally, in section 5 we offer our conclusions and point out to future lines of research.

2 Genetic Multivariate Polynomials

The problem discussed in the introduction may be recast in terms of finding and explicit algebraic expression of the form

$$F(V_1,\dots,V_n) = \sum_{I_1=0}^{D_1} \dots \sum_{I_n=0}^{D_n} C_{I1\dots In} V_{I1}^{D1} \dots V_{In}^{In} \quad (1)$$

In equation (1) the set of coefficients in (1) (which we denote as **C'**) may be considered to roughly replace **W** in **F** with the advantage of yielding an explicit mathematical relationship between the independent variables **X** and the dependent variable **Y**. This approximation problem has been the subject of many studies in the past and it has been pretty much solved except for certain practical issues. First of all, we have to contend with the fact that the cardinality of **C'** (which we denote as γ ') is large enough as to make it unwieldy for all except the simplest cases. Consider the NN of figure 1 once again. The Di's are the largest allowed degrees for the monomials of (1). If we assume that Di = 5 for all i we can easily calculate that $\gamma' = 6^6 = 46{,}656$. This means, first of all, that we need at least 46,656 elements in the sample. Secondly, if we assume that every floating point number uses 8 bytes (a typical representation for double precision numbers in a common high level language) then the simple storage of a hypothetical solution will require 373,248 bytes. Thirdly, and most importantly, the usual numerical methods with which these approximations are normally tackled imply the solution of systems of linear equations of similar order. Such systems inevitably lead to numerical instability which, in fact, renders the said methods ineffectual. Finally, fitting a relatively large set of data under the least squares error measure leads to Hilbert matrices which are known to be particularly sensitive to rounding errors. In other words, even though it is theoretically possible to solve the approximation problem even for large samples, in practice it is both impossible and impractical to do so with the usual methods. If we had a computer of infinite numerical precision and unlimited storage we could solve equation (1) trivially. But, this not being the case, we will replace the approach outlined by equation (1) by another one in which we retain the *form* (an algebraic polynomial) of the solution while removing the need for a large γ ' by introducing a set of constants $\boldsymbol{\mu}$ as shown in equation (2).

$$F(V_1,\dots,V_n) = \sum_{I_1=0}^{D_1} \dots \sum_{I_n=0}^{D_n} \mu_{I1\dots In} C_{I1\dots In} V_{I1}^{D1} \dots V_{In}^{In} \quad (2)$$

The elements of $\boldsymbol{\mu}$ determine whether a given monomial in (2) is to be included in the purported polynomial solution. For this reason we must establish *a priori* the cardinality of $\boldsymbol{\mu}$. Obviously we want to have a manageable number of coefficients (we denote the set of coefficients in (2) as **C** and its cardinality as γ) so that we do not longer need to contend with the problems derived from large storage needs. Assuming we are able to determine the best γ monomials we are still faced with the fact that large samples lead to large unmanageable systems of equations. Therefore, we will replace the least squares (or L_2) norm with the less usual (but well known) minimax (or Chebyshev or L_∞ norm). In so doing we will avoid the need to solve large systems of equations and get around the inherent precision problems. The minimax norm is not as popular as least squares because it is more sensitive to outliers[6] and because the approximation algorithms are slower. In our case, where the sample is assumed to have unknown characteristics, it is impossible to determine what "atypical" means. It would then seem that the first objection is not easy to sustain. The second one is valid but we have developed methods which diminish this inconvenience. In what follows we give a brief account of the principles behind the minimax multivariate approximation algorithm. Once this is done we will discuss the genetic algorithm which allows us to determine **C** for a given γ.

[6] When an element of the sample is outside of the "typical" trend of the whole it is called an "outlier".

2.1 The Ascent Algorithm

The algorithm to approximate the sample under the minimax norm is based on three observations: a) The approximation coefficients of a minimax set *M* of size *m* are uniquely determined by finding the *adequate signs* of the errors for the elements of the set, b) If a larger set *N* of size *n* has elements outside the minimax solution it is always possible to exchange one of the elements in *M* by an element in *N* such that the minimax condition is still met for the new *M* and c) Continuing exchanges will eventually lead to a target set *M* which satisfies the minimax norm for all elements in *N*.

In what follows we denote with F(**X**) the function which is able to fully minimize the error $\varepsilon_\varphi = \max|F_i(\mathbf{X}_i) - y_i|$. $\mathbf{X}_i$ denotes the value of the i-th independent variable vector and y_i the value of the i-th dependent variable. The original data set is found in matrix **X** of dimension $(n+1)\times s$; n is the number of independent variables and s the number of elements in the sample. In order to find the approximator of (2) we map the vectors of **X** to a higher dimensional space yielding matrix **V** of dimensions $p\times s$ where $p = \Pi_{i=1}^{n}(1+d_i)$. Let us arbitrarily select a sub-matrix of **V** of size $m\times m$ (call it **V'**) where m = p+1. Then we solve the following system.

$$\begin{bmatrix} \eta_1 & (v_1^0 \ldots v_n^0)_1 & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_1 \\ \eta_2 & (v_1^0 \ldots v_n^0)_2 & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_2 \\ \ldots & \ldots & \ldots & \ldots \\ \eta_m & (v_1^0 \ldots v_n^0)_m & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_m \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ c_1 \\ \ldots \\ c_m \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \ldots \\ d_m \end{bmatrix} \quad (3)$$

Denoting the approximation error for the i-th vector as ε_i and the largest absolute such error as ε_θ we may define $\varepsilon_i = \eta_i \varepsilon_\theta$; clearly, $\varepsilon_i \eta_i \le \varepsilon_\theta$. We also denote the elements of row i column j of (3) as δ_{ij} and the i-th cofactor of the first column as κ_i. From Cramer's rule we immediately have

$$\varepsilon_\theta = \frac{\begin{vmatrix} d_1 & \ldots & \delta_{1m} \\ \ldots & \ldots & \ldots \\ d_m & \ldots & \delta_{mm} \end{vmatrix}}{\eta_1\kappa_1 + \ldots + \eta_m\kappa_m} \quad (4)$$

To minimize ε_θ we have to maximize the denominator of (4). This is easily achieved by a) Selecting the maximum value of the η_i 's and b) Making the signs of the η_i 's all equal to the signs of the κ_i 's. Obviously the η_i 's are maximized iff $\eta_i = 1$ for i=1,…,m which translates into the well known fact that the minimax fit corresponds to approximation errors of the same absolute size. On the other hand, to achieve (b) we must simply set the signs of η_i 's to those of the cofactors. Making $\sigma_i = \operatorname{sign}(\kappa_i)$, system (3) is simply re-written as

$$\begin{bmatrix} \sigma_1 & \ldots & \delta_{1m} \\ \ldots & \ldots & \ldots \\ \sigma_m & \ldots & \delta_{mm} \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ \ldots \\ c_m \end{bmatrix} = \begin{bmatrix} d_1 \\ \ldots \\ d_m \end{bmatrix} \quad (5)$$

Once having all the elements in (5) it suffices to solve this system to obtain, both, the value of ε_θ and the coefficients $c_1,\ldots,c_m$ which best fit the elements of **X** in the minimax sense. To obtain the coefficients for the whole sample we apply the following algorithm.

2.2 Exchange Algorithm

1. Set $i \leftarrow 1$.
2. Select an arbitrary subset (of size m) of rows of matrix **V**; this set is called M_i.
3. Determine the signs of ε_i which maximize the denominator of (4).
4. Solve the system of (5). Denote the resulting polynomial by P_i.
5. Calculate $\varepsilon_\varphi = \max(|P_i - y_i|) \quad \forall i \notin M_i$
6. If $\varepsilon_\varphi \le \varepsilon_\theta$ end the algorithm; the coefficients of P_i are those which best approximate **V** in the minimax sense.
7. Set $i \leftarrow i+1$.
8. Exchange the row corresponding to ε_θ for the one in M_i which preserves it sign and makes $(\varepsilon_\theta)_{i+1} > (\varepsilon_\theta)_i$.
9. Go to step 4.

□

The exchange algorithm will end as long as the consecutive systems of (5) satisfy Haar's condition while, on the other hand, the cost of its execution (in *flops*) is O(m^6). There are implementation issues which allow us to apply this algorithm even in the absence of Haar's conditions and reduce its costs to $O(m^2)$ the interested reader is referred to [KU99].

2.3 Genetic Algorithm

One of the basic reasons to choose the minimax norm is that the Exchange Algorithm is not dependent on the origin of the elements in **V**. We decided them to be monomials of a full polynomial but it makes no difference to the Exchange Algorithm whether the $\mathbf{v}_i$ are gotten from a set or monomials or they are elements of arbitrary data vectors. This is important because the number of monomials in (3) grows geometrically. One way to avoid the problem of such coefficient explosion is to define a priori γ and then to properly select which of the γ' possible ones these will be. There are $\binom{\gamma'}{\gamma}$ possible combinations of monomials and even for modest values of γ an exhaustive search is out of the question. This optimization problem may be tackled using a genetic algorithm as follows.

The genome is a binary string of size γ'. Every bit in it represents a monomial. If the bit is '1' it means that the corresponding monomial remains while, if it is '0', such monomial is not to be considered. This simple strategy corresponds to determining the values of $\boldsymbol{\mu}$ in equation (2). All one has to ensure is that the number of 1's is equal to $|\boldsymbol{\mu}|$. Assume, for example, that $\mathbf{v} = (v_1, v_2, v_3)$ and that d1 = 1, d2 = 2 and d3 = 2; if $|\boldsymbol{\mu}| = 6$ the genome 110000101010000001 corresponds to the polynomial in (6).

$$P(v1,v2,v3)=C_{000}+C_{001}v3+C_{020}(v2)^2+C_{022}(v2)^2(v3)^2+\ldots$$
$$C_{112}v1v2(v3)^2+C_{122}V1(v2)^2(v3)^2 \qquad (6)$$

It is well known that an elitist GA will converge to a global optimum [GR94]. It has also been shown that a variation called Vasconcelos' GA shows superior behavior on a wide range of functions [KU02]. VGA uses a) Deterministic parenthetical selection, b) Annular crossover, c) Uniform mutation. All results reported here are based on VGA's application (for a detailed discussion see [KV98]).

Therefore, the initial population of the VGA is generated randomly. It consists of a set of strings of length γ' in which there are only γ 1's. Then the GA's operators are applied as usual. The fitness function of the GA is the minimax error as per the exchange algorithm. This error is minimized and, at the end of the process, the polynomial exhibiting the smallest fit error is selected as the approximant for the data sample. This polynomial is called a "Genetic Multivariate Polynomial" (GMP).

The learning machine methodology outlined for the BMLPs in section (1) is, in fact, independent of the learning algorithm and here we replace the NN with a GMP. In the next section we discuss the results of applying this methodology with, both, BMLPs and GMPs.

3 Experiments

In this section we report on different sets of data which were a) Normalized to the interval [0,1] and b) Split into two random partitions accounting for 75%-25% of the data, respectively. We trained, both, a BMLP and a GMP. Then we chose the BMLP/GMP whose behavior was optimal for the test data set, i.e. the one with the best generalization properties. In what follows we describe the different experiments.

3.1 Experiment 1: Time Series

An experimental time series data consisting of a 13×90 matrix was considered. This data correspond to the historical behavior of an airline during an 11 year period. The corresponding graph is shown in figure 2.

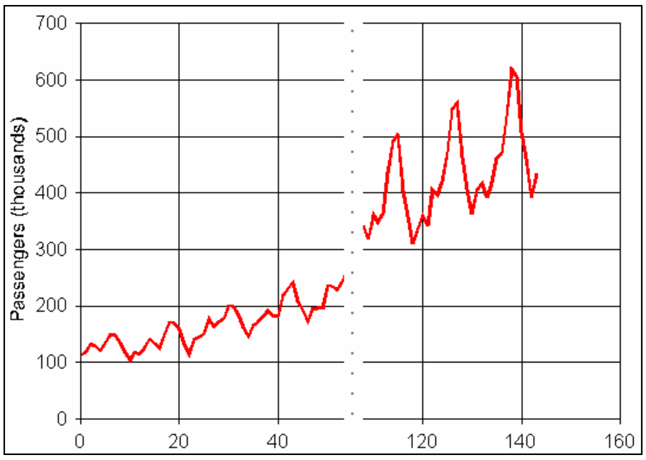

Fig. 2. Graph for time series

3.2 Experiment 2: Neural Network I

We considered a 2-1 architecture for a BMLP. We generated a 3x100 matrix as follows: Columns C1 and C2 were randomly filled in with uniform numbers in [0,1]. Column C3 was calculated by simulating a BMLP whose inputs correspond to the values of C1 and C2 and whose output was calculated by using a logistic activation function for the weights corresponding to those of table 1.

Table 1. Weights of the 2-1 BMLP

1.4142000000
3.1416000000
2.7182800000
0.5002210000
-1.1200000000

3.3 Experiment 3: Neural Network II

We generated a table of dimensions 5x100. The first column (C1) was randomly filled in as above; C2=sin(C1)cos(C1); C3=sin(C2); C4=tan(C2)sin(C3). The dependent variable's values were calculated as in experiment 2 with the weights shown in table 2.

Table 2. Weights for the 4-1 BMLP

0.710677794
0.273139125
0.336020956
0.173518450
0.620503227
0.808716951
0.754668302
0.775027323
0.287502668
0.794020586
0.959432104
0.846120365
0.140885327

The corresponding graph is shown in figure 3.

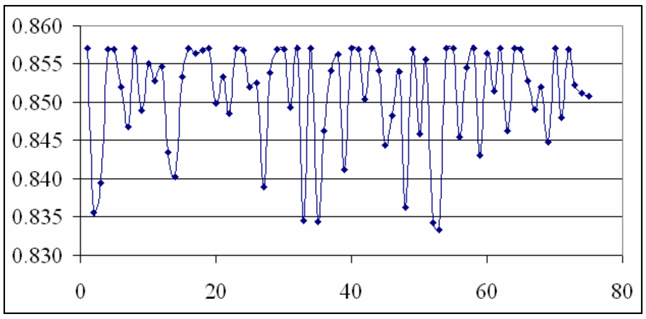

Fig. 3. Graph for 4-1 BMLP

3.4 Experiment 4: Transcendental Function I

We generated a table of dimensions 9x100. The first 4 columns were defined as in experiment 3. The remaining columns (C5-C8) were calculated as follows: C5=sinh(C1); C6=cosh(C1); C7=tanh(C1); C8=sin(C1)cos(C1); C9=$(C1)^2$-2$(C2)^3$+(C2C3C4)-3C5 -C6C7 -2C2(1-C4) -2C5(1-C6) +2C1(1/(C7C8)). The corresponding graph is shown in figure 4.

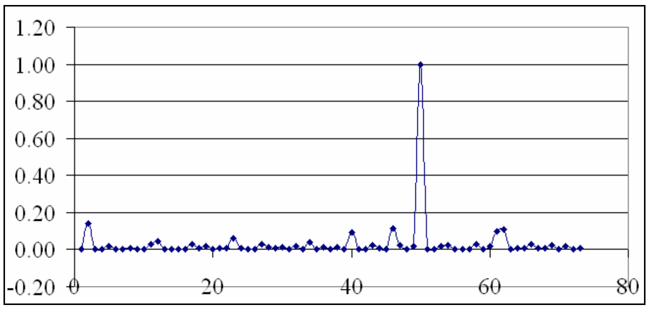

Fig. 4. Graph for Transcendental Function I

3.5 Experiment 5: Transcendental Function II

A table of dimensions 13x100 was generated. The first 9 columns were filled in as in experiment 4. The remaining 5 columns were calculated as follows. C10 = C8/C9; C11 = C8/(C9/C7); C12 = C1+C8C6-C6$(C2)^3$; C13 = sin(C2)cos(C4)tan(C6)/(sin(C6)-1/tan(C8)). The corresponding graph is shown in figure 5.

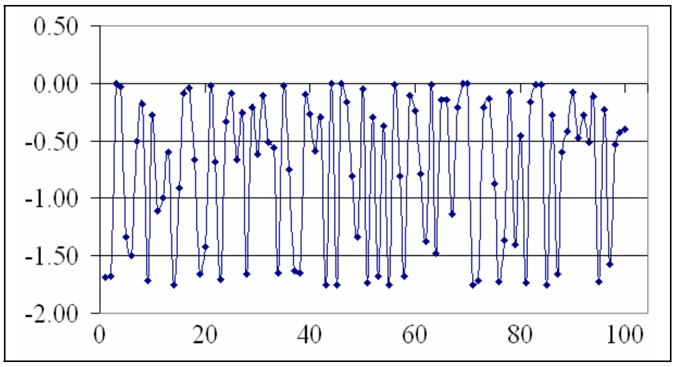

Fig. 5. Graph for Transcendental Function II

For all five functions we trained both a BMLP and a GMP.

Table 3. Coefficients for time series

ε_φ	0.050929454
C001111000010	2.506630455
C001110110111	51.95542862
C000101000011	0.602721425
C010010100110	4.949337075
C101110100010	-11.57360514
C000010010001	-0.039080933
C110111010110	-77.34324491
C100000000000	0.957983801

In table 3 we present an example of the coefficients gotten from training the GMP for the problem in experiment 1. The index of the coefficient denotes the degree of the corresponding monomial. There are 12 indices, one for each independent variable.

In table 4 we show the characteristics of each learning method; in table 5 we show the maximum and RMS errors for the test set in both cases.

Table 4. Basic features for BMLP and GMP

	Independent	BMLP	GMP	
Exp.	Variables	Connections	Terms	Degree
1	12	15	8	1 [8]
2	2	5	6	2 [12]
3	4	13	6	2 [12]
4	8	41	20	2 [40]
5	12	71	20	2 [40]

Notice that the number of terms (γ) for GMPs is of, at most, 20. In contrast, the number of connections in the BMLPs goes up to 71. Also, the maximum allowed degrees for each of the monomials is kept at 2. The number between square parentheses indicates the maximum degree for any given monomial.

Table 5. Errors for BMLP and GMP

	BMLP		GMP	
Exp.	Max Error	RMS Error	Max Error	RMS Error
1	0.130900	0.055800	0.10430000	0.05090000
2	0.004200	0.002200	0.00000309	0.00000110
3	0.000006	0.000003	0.00000011	0.00000006
4	0.167500	0.019290	0.07066679	0.02000086
5	0.008800	0.002200	0.00872117	0.00315788

Notice that errors for GMPs are smaller in all cases. Particularly noteworthy is the case of experiment 4, in which BMLP's maximum error reaches a 16.75%. Looking at figure 4 we see a spike which is responsible for this large NN error. The GMP, on the other hand, does rather well. It was here that we had to increase the Di's to better fit the data. This, in itself, displays the advantage of GMPs over NNs: since we are able to explicitly determine the causes for the behavior of the GMP we are able to modify it accordingly to improve its performance.

4 Conclusions

As noted, the polynomials resulting from the training process yield an explicit algebraic expression for the phenomenon under study. The polynomial expression allows, among other things, a) To perform a sensitivity analysis of the input variables and b) To easily integrate and derive the resulting function. Both of these operations are simply not possible when training *any* kind of NN. Furthermore, even in those cases when the original data obeys complex relations, a multi-variate polynomial closely reflects the behavior of the function. Although polynomial expressions are mathematically limited (for instance, polynomials may not reflect discontinuities) in the analyzed cases such limitations did not arise even in the presence of complex non-linear data. Given the fact that the GMPs are gotten independently of the form of the approximating function, we may try out combinations of more complex basic elements which are not restricted as mentioned. In the future we intend to explore this alternative. It is reasonable to assume that such options will yield even better results.

References

[BB92] Boser, B. E., I.M. Guyon and V. N. Vapnik, "A training algorithm for optimal margin classifiers", Proc. 5th Annual ACM Workshop on Computational Learning Theory, pp. 144–152, 1992.

[GR94] Rudolph, G., "Convergence Analysis of Canonical Genetic Algorithms", IEEE Transactions on Neural Networks, 5(1):96-101, January, 1994.

[KU98] Kuri, A., "A Universal Eclectic Genetic Algorithm for Constrained Optimization", 1998, Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98, pp. 518-522.

[KU99] Kuri, A., A Comprehensive Approach to Genetic Algorithms in Optimization and Learning. Theory and Applications, Vol. 1. Instituto Politécnico Nacional, pp 270, 1999.

[KU02] Kuri, A., A Methodology for the Statistical Characterization of Genetic Algorithms, Proceedings of the II Mexican International Congress on Artificial Intelligence, MICAI 2002, LNAI 2313, pp. 79-88.

[SH99] Haykin, S., "Neural Networks. A Comprehensive foundation", 2nd Edition, Prentice Hall, 1999.

[VV95] Vapnik, V., "The Nature of Statistical Learning Theory", Springer-Verlag, 1995.