Genetic Multivariate Polynomials: An Alternative Tool to Neural Networks

Angel Fernando Kuri-Morales[1] and Federico Juárez-Almaraz[2]

[1] Instituto Tecnológico Autónomo de México
[2] Universidad Nacional Autónoma de México
Río Hondo No.1, México 01000, D.F.
akuri@itam.mx

Abstract. One of the basic problems of applied mathematics is to find a synthetic expression (model) which captures the essence of a system given a (necessarily) finite sample which reflects selected characteristics. When the model considers several independent variables its mathematical treatment may become burdensome or even downright impossible from a practical standpoint. In this paper we explore the utilization of an efficient genetic algorithm to select the "best" subset of multivariate monomials out of a full polynomial of the form $F(v_1,...,v_n) = \sum_{i_1=0}^{g_1} ... \sum_{i_n}^{g_n} c_{i_1...i_n} v_1^{i_1} ... v_n^{i_n}$ (where g_i denotes the maximum desired degree for the i-th independent variable). This regression problem has been tackled with success using neural networks (NN). However, the "black box" characteristic of such models is frequently cited as a major drawback. We show that it is possible to find a polynomial model for an arbitrary set of data. From selected practical cases we argue that, despite the restrictions of a polynomial basis, our Genetic Multivariate Polynomials (GMP) compete with the NN approach without the mentioned limitation. We show how to treat constrained functions as unconstrained ones using GMPs.

1 Introduction

One of the basic goals of the scientific endeavor is to (try to) identify patterns in apparently chaotic data given a (necessarily) finite sample which reflects selected characteristics in the system under study. In this paper we explore the utilization of an efficient genetic algorithm to select the "best" subset of multivariate monomials out of a full polynomial. Such multivariate regression problem has been tackled with success using neural networks (NN) whose "black box" nature is frequently cited as a major drawback. We show that it is possible to find a polynomial model for an arbitrary set of data and give evidence that, despite the restrictions of a polynomial basis, our Genetic Multivariate Polynomials (GMPs) compete with the NN approach without the mentioned "black box" limitation.

1.1 Statistical Systems

Statistical systems are a relatively modern approach to automated machine learning (AML). They rely on the overall analysis of data representing the behavior of the system. No previous knowledge about the system is assumed and, indeed, they do achieve AML with a certain amount of success depending on how one measures it.

1.1.1 Neural Networks

Perhaps the most representative systems in this category are the so-called neural networks (NN). The basic idea is that simple computing elements (which we will refer to as "units"), individually displaying little computing power, when arranged in richly interconnected networks, may embody the essence of the system they model. The term "neuron" arises from suggestive analogies where the units purportedly simulate the behavior of the neuron of a living being. Every connecting path between units has an associated weight. It is in these weights that knowledge, tacitly (as opposed to the explicit rules of the classical AI approach) is stored in a "trained" network. In supervised mode the NN is "shown" the data repeatedly and, via an iterative algorithm, it modifies the initial (typically random) value of the weights so that the NN's outputs replicate the known ones for every element in the data. NNs have evolved from the initial animal-neuron-inspired approach into sophisticated entities in which units are determined by their mathematical properties.

The statistical nature of the learning process has been given solid theoretical foundation by the work of many researchers, outstanding that of Vapnik [VV95]. It has been proven that a feedforward strongly interconnected network of units (perceptrons) constitutes a universal approximator [SH99]. Furthermore, analogous NNs are able to represent the data in the best possible way given a set of data [BB92]. Notice that the proper selection of the data is not an issue here; data is assumed to have been properly selected (a fact which we will take for granted in what follows). In conclusion, NNs are able to extract knowledge, given an arbitrary set of data, fully and optimally. However, a drawback of NNs is that the process by which they arrive at their conclusions is not explicit and, upon presentation of a larger (possibly richer) set of data the learning process has to be repeated or, at best, continued from the previous one. Nevertheless, the NN methodology yields a tool which is able to tackle complex multivariate regression effectively.

1.1.2 Multivariate Polynomials

An obvious alternative is to attempt such regression appealing to a functional representation (such as the one in (1)) where the known response of the system to a set of input stimulae is expressed explicitly.

$$F(v_1,\dots,v_n)=\sum_{i_1=0}^{g_1}\dots\sum_{i_n}^{g_n} c_{i_1\dots i_n} v_1^{i_1}\dots v_n^{i_n} \qquad (1)$$

In (1) v_i corresponds to the *i-th* independent variable and g_i is the highest allowed power for v_i. In order to find $F(v_1,\dots,v_n)$ one must device a method to approximate the data in a typically overdetermined system for a given metric. We must also overcome the curse of dimensionality inherent to this approach[1]. In what follows we give a

[1] For instance, consider a problem where n=10 and $g_1=g_2=\dots=g_n=4$. The number of coefficients in (1) is easily calculated as $C = 5^{10} = 9{,}765{,}625$ which implies that we must have, at least, those many elements in our sample.

method which allows us to solve both problems. In part 2 we expound the method. In part 3 we make a comparison of a representative set of problems tackled with GMPs and NNs. In part 4 we offer our conclusions.

2 Genetic Multivariate Polynomials

To approximate the data vectors we have chosen the minimax or L_∞ norm for reasons that will become apparent in what follows. In L_∞ one seeks an F(x) that minimizes ε_θ, where $\varepsilon_\theta = \max |F(\mathbf{v}_i) - d_i|$; $\mathbf{v}_i$ denotes the *i-th* independent variable vector and d_i the *i-th* desired output. The original data set is found in matrix **O** of dimensions $(n+1)\times s$; where *n* denotes the number of independent variables and *s* the number of elements in the sample. In order to find the approximator of (1) we map the vectors of **O** to a higher dimensional space yielding matrix **V** of dimensions $p \times s$, where $p = \Pi_{i=1}^{n}(1+g_i)$.

2.1 Minimax Approximation to a Set of Size *m*

To illustrate minimax approximation we arbitrarily select a submatrix of **V** of size $m\times m$ (call it **V'**), where $m=p+1$; then, we solve the system of (2).

$$\begin{bmatrix} \eta_1 & (v_1^0 \ldots v_n^0)_1 & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_1 \\ \eta_2 & (v_1^0 \ldots v_n^0)_2 & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_2 \\ \ldots & \ldots & \ldots & \ldots \\ \eta_m & (v_1^0 \ldots v_n^0)_m & \ldots & (v_1^{g_1} \ldots v_n^{g_n})_m \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ c_1 \\ \ldots \\ c_m \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \ldots \\ d_m \end{bmatrix} \quad (2)$$

Denoting the approximation error for the *i-th* vector as ε_i we may define $\varepsilon_i = \eta_i \varepsilon_\theta$; clearly, $\eta_i \varepsilon_i \le \varepsilon_\theta$. We also denote the elements of row *i*, column *j* of (2) as δ_{ij} and the *i-th* cofactor of the first column as κ_i. From Cramer's rule, we immediately have:

$$\varepsilon_\theta = \frac{\begin{vmatrix} d_1 & \ldots & \delta_{1m} \\ \ldots & \ldots & \ldots \\ d_m & \ldots & \delta_{mm} \end{vmatrix}}{\eta_1 \kappa_1 + \ldots + \eta_m \kappa_m} \quad (3)$$

To minimize ε_θ we have to maximize the denominator of (3). 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 equal absolute size. On the other hand, achieving (b) simply means that we must set the signs of the η_i's to those of the cofactors. Making σ_i=sign(κ_i) system (2) is simply re-written as

$$\begin{bmatrix} \sigma_1 & \dots & \delta_{1m} \\ \dots & \dots & \dots \\ \sigma_m & \dots & \delta_{mm} \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ \dots \\ c_m \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_m \end{bmatrix} \tag{4}$$

Once having all the elements in (4) it suffices to solve this system to obtain both the value of ε_θ an the coefficients $c_1,...,c_m$ which best fit the elements of **V'** in the minimax sense. To find the minimax coefficients for **V** we apply the next algorithm.

2.2 Exchange Algorithm

1. Set i $\leftarrow$ 1.
2. Select an arbitrary set (of size *m*) of rows of matrix **V**; this set is called M_i.
3. Determine the signs of the ε_i which maximize the denominator of (3).
4. Solve the system of (4). Denote the resulting polynomial by P_i.
5. Calculate the value of $\varepsilon_\phi = \max(|P_i - d_i|)\ \forall\ \mathbf{v}_i \notin M_i$.
6. If $\varepsilon_\phi \le \varepsilon_\theta$ end the algorithm; the coefficients of P_i are those of the polynomial which best approximates **V** in the minimax sense.
7. Set i $\leftarrow$ i+1.
8. Exchange the row corresponding to ε_ϕ for the one in M_i which preserves its 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 (4) satisfy Haar's condition while, on the other hand, the cost of its execution (in FLOPs) is of O(m^6). There are implementation issues which allow to apply this algorithm even in the absence of Haar's condition and which reduce its cost to O(m^2). The interested reader is referred to [KG02].

2.3 Genetic Algorithm

The basic reason to choose a minimax norm is that the method outlined above is not dependent on the origin of the elements in **V**. We decided them to be the monomials of a full polynomial. But it makes no difference to the exchange algorithm whether the $\mathbf{v}_i$ are gotten from a set of monomials or they are elements of arbitrary data vectors. This is important because, as stated above, the number of monomials and coefficients in (2) grows geometrically. One way to avoid the problem of such coefficient explosion is to define a priori the number (say μ) of desired monomials of the approximant and then to properly select which of the *p* possible ones these will be. There are $\binom{p}{\mu}$ possible combinations of monomials and even for modest values of *p* and μ and exhaustive search is out of the question. This optimization problem may be tackled using a genetic algorithm (GA), as follows.

The genome is a binary string of size *p*. Every bit in it represents a monomial. If the bit is '1' it means that the corresponding monomial remains while if it is a '0' it means that such monomial is not to be considered. All one has to ensure is that the number of 1's is equal to μ . Assume, for example, that **v** = (v_1, v_2, v_3) and that g_1=1, g_2=2, g_3=2; if $\mu = 6$ the genome 110000101010000001 corresponds to the polynomial in (5).

$$P(v_1, v_2, v_3) = c_{000} + c_{001}v_3 + c_{020}v_2^2 + c_{022}v_2^2v_3^2 + c_{112}v_1v_2v_3^2 + c_{122}v_1v_2^2v_3^2 \tag{5}$$

It is well known that any elitist GA will converge to a global optimum [GR94]. It has also been shown that a variation of GA called Vasconcelos Genetic Algorithm (VGA) shows superior behavior on a wide range of functions [AK00]. VGA uses a) Deterministic parenthetical selection, b) Annular crossover, c) Uniform mutation [KV98]. All results reported are based on VGA's application.

Therefore, the initial population of the GA is generated randomly. It consists of a set of binary strings of length *p* in which there are only μ 1's. Then the GA's operators are applied as usual. The fitness function is the minimax fitness 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 best approximant for the original data set.

3 Neural Networks and GMPs

As we already pointed out, NNs have been proven to be able to synthesize the knowledge contained in an arbitrary set of data. Particularly, when the units are the well known perceptrons [SH99], any continuous function may be approximated by a three layer NN, such as the one shown in figure 1.

In figure 1 we show a NN with 6 input variables and one output variable, i.e., one dependent variable and 6 independent ones. The **b** neuron is the so-called bias and its input is canonically set to +1. It is easy to see that there are w = 33 ($6 \times 4 + 4 \times 1 + 4 + 1$) weights in this network. The number of neurons in the input and output neurons is determined by the number of input and output variables respectively. The number of neurons in the hidden layer (H) was estimated from the heuristic rule of equation (6).

$$H \approx \frac{S - 3O}{3(I + O + 1)} \tag{6}$$

Here, S is the number of elements in the data sample; I and O are the number of input and output neurons, respectively. What equation (6) says is that the number of weights should equal, roughly, 1/3 of the size of the sample. With these convention we tackled the problem of approximating a set of constrained functions of which a small fraction is shown in table 1.

In every case, we sampled the independent variables randomly and selected those values which complied with the constraints. Equalities were treated as closely bounded inequalities. For example, the first constraint of function 6 was actually transformed into: $x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 \geq 9.9999$ and $x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 \leq 10.0001$. The samples represented the actual values of interest of every one of the functions and the resulting NN, in fact, constitutes an alternate non-constrained version of the original one.

Table 1. A Set of Constrained Functions

No	Function	Constraints
1	$(x_1 - x_2)^2 + (x_2 - x_3)^4 + 1$	$x_1 + x_1 x_2^2 + x_3^4 = 3$
2	$x_1^2 + 4x_2^2$	$\frac{3}{5}x_1 + \frac{4}{5}x_2 \geq \frac{13}{5}$
3	$9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2$ $+ 2x_2^2 + x_3^2 + 2x_1 x_2 + 2x_1 x_3$	$x_1 \geq 0; x_2 \geq 0; x_3 \geq 0$ $-x1 - x2 - x3 + 3 \geq 0$
4	$1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1 x_2 - x_1 x_3$	$x_1^2 + x_2^2 - x_3^2 - 25 = 0$ $8x_1 + 14x_2 + 7x_3 - 56 = 0$ $x_i \geq 0 \quad i = 1,2,3$
5	$100(x2 - x_1^2) + (1 - x_1^2)^2 + 90(x_4 - x_3^2)^2$ $+ (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x4 - 1)^2]$ $+ 19.8(x_2 - 1)(x_4 - 1) + 1$	$-10 \leq x_i \leq +10$
6	$\exp(x_1 x_2 x_3 x_4 x_5)$	$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10$ $x_2 x_3 - 5x_4 x_5 = 0$ $x_1^3 + x_2^3 = -1$ $-2.3 \leq x_i \leq +2.3 \quad i = 1,2$ $-3.2 \leq x_i \leq +3.2 \quad i = 3,4,5$

Our thesis may be resumed as follows:

a) The domain of a constrained function may be sampled in such a way that the resulting sample represents adequately the domain of a constrained function.

b) Any set of data may be re-expressed as a trained NN.

c) If a GMP is able to duplicate the workings of a NN it is possible to work with the resulting algebraic expression.

d) The optimization process can be performed on the polynomial with traditional calculus' tools.

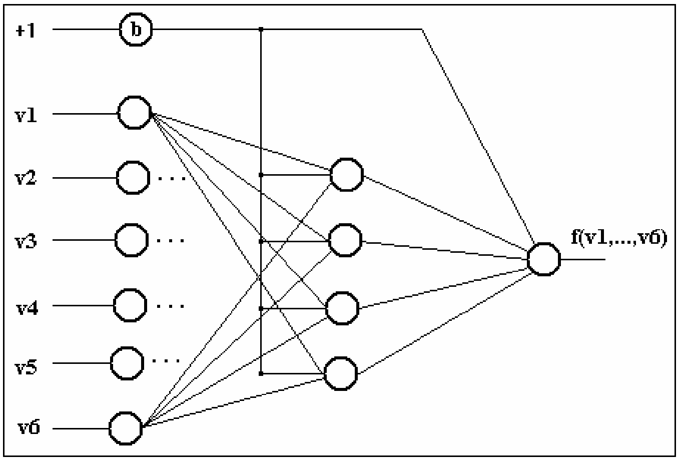

Fig. 1. Three-layered Perceptron Network

3.1 Experiments

Data was divided in two sets: a training set and a test set. The training set encompasses 80% of the data; the test set consists of the remaining 20%. Both NNs and GMPs were trained using the training set. Then both methods were tested for performance on the test set, which they had not previously "seen". The number of weights for the NNs were calculated from (6); the number of monomials in the GMP was set accordingly. In the following table we show the actual errors found from the trained NNs and GMPs for the selected functions. Eight types of error were compiled: a) Maximum training error for NNs and GMPs; b) RMS training error for NNs and GMPs; c) Maximum test error for NNs and GMPs; d) RMS test error for NNs and GMPs.

Table 2. Error Comparison for Selected Functions (NN and GMP)

	Maximum Training Error		RMS Training Error		Maximum Test Error		RMS Test Error	
	NN	GMP	NN	GMP	NN	GMP	NN	GMP
Function								
1	0.0915	0.0745	0.0318	0.0338	0.0668	0.0600	0.0456	0.0400
2	0.5419	0.1593	0.1015	0.0978	0.3310	0.2319	0.0906	0.0393
3	0.0955	0.0919	0.0218	0.0465	0.1381	0.1861	0.0354	0.0618
4	0.2069	0.0724	0.0742	0.0466	0.1872	0.0960	0.0731	0.0480
5	0.2654	0.2148	0.0616	0.0849	0.3695	0.2589	0.1337	0.1249
6	0.0014	0.0003	0.0002	0.0001	0.0328	0.1805	0.0070	0.0280

4 Conclusions

Table 2 shows the remarkable performance of NNs and GMPs for this set of problems. For instance, the RMS test error was always of O(0.1) which directly bears on the generalization properties of the model. NNs behavior was expected but GMP's was not as obvious: with two exceptions, GMPs showed better generalization capabilities than their neural counterparts.

That maximum errors were smaller for GMPs may be explained easily, since the norm focuses on their minimization. That, in the majority of cases, GMPs RMS errors were comparable was not so clear, particularly since the number of monomials and weights were the same. In the perceptron networks the underlying functions (based on a sigmoidal transformation of the local induced field) are much more complex and, in principle, richer than linear combinations of monomials. However, as attested by the results, the VGA does a fine job in finding the best such combinations.

The polynomial expression shows explicitly which powers of the independent variables bear on the behavior of the function and to what extent. It also allows for simple algebraic manipulation of the different terms. For instance, finding the partial derivatives with respect to any of the input variables is trivial and allows for the simple analysis of the function's behavior.

On the other hand, given the reliable representation of the original data, the method suggests a general algorithm to tackle constrained optimization problems as follows:

a) Sample the feasible domain of the constrained function
b) Synthesize the function appealing to a GMP
c) Optimize utilizing traditional algebraic or numerical tools.

We do not claim that the optimization process proposed herein will be able to deliver a global optimum. However, in general, it will certainly approach one or more (depending on the starting VGA's population) local optima. These may be utilized to refine the search using other techniques.

Finally, we would like to emphasize the fact that GMPs are not limited to use simple monomials as units. Other basis are applicable and it only remains to see whether the extra computational cost implied in more complex units yields cost effective results.

References

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

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

[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.

[KG02] Kuri, A., Galaviz, J., "Algoritmos Genéticos", Fondo de Cultura Económica, México, 2002, pp. 165-181.

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

[AK00] Kuri, A., “A Methodology for the Statistical Characterization of Genetic Algorithms”, Lectures Notes in Artificial Intelligence No 2313, pp. 79-89, Coello, C., Albornoz, A., Sucar, L., Cairó, O., (eds.), Springer Verlag, April 2000.

[KV98] Kuri, A., Villegas, C., “A Universal Genetic Algorithm for Constrained Optimization”, *EUFIT '98, 6th European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1998.