Application of a Method Based on Computational Intelligence for the Optimization of Resources Determined from Multivariate Phenomena

Angel Kuri-Morales

Departamento de Computación
Instituto Tecnológico Autónomo de México
Mexico City, Mexico
akuri@itam.mx

Abstract. The optimization of complex systems one of whose variables is time has been attempted in the past but its inherent mathematical complexity makes it hard to tackle with standard methods. In this paper we solve this problem by appealing to two tools of computational intelligence: a) Genetic algorithms (GA) and b) Artificial Neural Networks (NN). We assume that there is a set of data whose intrinsic information is enough to reflect the behavior of the system. We solved the problem by, first, designing a system capable of predicting selected variables from a multivariate environment. For each one of the variables we trained a NN such that the variable at time $t+k$ is expressed as a non-linear combination of a subset of the variables at time t. Having found the forecasted variables we proceeded to optimize their combination such that its cost function is minimized. In our case, the function to minimize expresses the cost of operation of an economic system related to the physical distribution of coins and bills. The cost of transporting, insuring, storing, distributing, etc. such currency is large enough to guarantee the time invested in this study. We discuss the methods, the algorithms used and the results obtained in experiments as of today.

Keywords: Computational intelligence, forecasting, optimization, non-linear modeling.

1 Introduction

The problem of optimizing the future assignment of resources to a dynamic system whose behavior displays a non-chaotic behavior poses two clearly distinct challenges: a) One has to be able to reliably predict the future behavior of the system based on its past history and b) Assuming the purported prediction's reliability, one has to establish the best possible values of the variables involved such that a cost function is minimized. In practical terms this means that, from a table of values of a set of variables $\boldsymbol{V}(t)=v_1(t),\ldots,v_n(t)$ reflecting the previous known values of a system $\boldsymbol{S}(t)=f(\boldsymbol{V}(t))$ at time t with an associated function of cost $C(t) = f(\boldsymbol{V}(t))$ we are to minimize $C(t+k) = f(\boldsymbol{V}(t+k))$ where k represents the time displacement relative to the present time t which we want forecast.

I. Batyrshin and M. González Mendoza (Eds.): MICAI 2012, Part II, LNAI 7630, pp. 292–303, 2013.

To do this, then, we first want to find $V(\tau) = \{v_1(\tau),...,v_n(\tau)\}$ such that

$$v_i(\tau) = f_i(v_1(t),..., v_n(t)) \quad \text{for } i = 1, \ldots, n \tag{1}$$

(where, for convenience, we have made $\tau = t + k$). Second, we would like to minimize $C(\tau)$.

To exemplify, assume that $C(\tau)$ has the polynomial form

$$C(\tau) = c_{0000} + c_{0123} v_2(\tau) v_3(\tau)^2 v_4(\tau)^3 + c_{3211} v_1(\tau)^3 v_2(\tau)^2 v_3(\tau) v_4(\tau) \tag{2}$$

The form of (1) is determined from the conditions of the problem. Generally, it is a combination of the conditions of the dynamics of the system and normally does not have a closed mathematical formulation. This is one of the main reasons behind the choice of a GA as a minimization tool.

Our first aim is to find the functions which yield the approximate values of $v_i(\tau)$ for *i=1, 2, 3, 4*. Such forecasted values (call them $v_i^*(\tau)$) will have an expected approximation error ε_i such that $v_i^*(\tau) = v_i(\tau) \pm \varepsilon_i$. Our second aim is to minimize $C(\tau)$ where, because of the expected approximation errors, the following constraints do hold: $v_i^*(\tau) - \varepsilon_i \le v_i(\tau) \le v_i^*(\tau) + \varepsilon_i$. This minimization process will deliver the values which will yield the forecasted minimum value of $C(\tau)$ taking into consideration possible uncertainties in the forecasted values of the $v_i(t)$.

We have approximated the $v_i(\tau)$ using a set of multi-layer perceptron NNs [1] and minimized $C(\tau)$ using a GA [2]. The unknown values of the model of $v_i(t)$ loosely correspond to the weights in the NN; one NN for each of the $v_i(\tau)$. We denote the function implemented by a NN for $v_i(\tau)$ by

$$v_i^*(\tau) = NN_i(v_1(t),..., v_n(t)) \tag{3}$$

To achieve our goals we must solve a number of issues such as: a) Which variables to consider in our problem, b) How to pre-condition the values of the v_i's, c) How to set the data to achieve the desired forecast, d) What is the architecture of the NN_i (That is, i) The number of layers in the NNs, ii) The number of nodes (*neurons*) in the hidden layer, iii) The activation function in each of the layers, etc.), e) How to measure the NN_i's reliability, f) What GA to apply (and its settings) during the optimization process.

In part 2 we discuss the pre-conditioning of the data. In part 3 we describe the way the NNi's were set, in part 4 we describe the training method for the NN_i, in part 5 we describe the optimization process, in part 6 we describe some experimental results, in part 7 we offer our conclusions.

2 Data Set Up

In what follows we describe the nature of the data with which we worked, the way we set it up to achieve the desired forecasting and the pre-conditioning which was required to increase our probabilities of success. The way in which historical data may be applied to generate reliable forecasting may be better explained by taking an example. To begin with, a correlation matrix was used to detect those variables which exhibited a correlation larger than 90% as shown in figure 1.

1		V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V20	V21
2	V1	1.00	0.35	0.81	0.48	0.75	0.81	0.71	0.87	0.67	0.40	1.00	0.49	0.54	0.54	0.79	0.72	0.32	0.52	0.64	0.79
3	V2	0.35	1.00	0.08	0.13	0.37	0.40	0.41	0.28	0.34	0.29	0.35	0.95	0.30	0.19	0.38	0.21	0.31	0.08	0.33	0.22
4	V3	0.81	0.08	1.00	0.56	0.53	0.66	0.60	0.79	0.46	0.18	0.81	0.10	0.83	0.32	0.56	0.52	0.30	0.55	0.43	0.60
5	V4	0.48	0.13	0.56	1.00	0.46	0.75	0.47	0.69	0.39	0.27	0.48	0.33	0.23	0.17	0.48	0.18	0.21	0.38	0.34	0.51
6	V5	0.75	0.37	0.53	0.46	1.00	0.62	0.43	0.62	0.99	0.90	0.75	0.57	0.08	0.67	1.00	0.78	0.06	0.06	0.98	0.91
7	V6	0.81	0.40	0.66	0.75	0.62	1.00	0.78	0.96	0.52	0.31	0.81	0.55	0.35	0.08	0.65	0.35	0.45	0.67	0.48	0.67
8	V7	0.71	0.41	0.60	0.47	0.43	0.78	1.00	0.77	0.36	0.14	0.71	0.41	0.55	0.01	0.46	0.14	0.85	0.75	0.32	0.37
9	V8	0.87	0.28	0.79	0.69	0.62	0.96	0.77	1.00	0.52	0.26	0.87	0.43	0.53	0.14	0.66	0.41	0.44	0.76	0.48	0.68
10	V9	0.67	0.34	0.46	0.39	0.99	0.52	0.36	0.52	1.00	0.94	0.67	0.53	0.01	0.68	0.98	0.76	0.12	0.03	1.00	0.88
11	V10	0.40	0.29	0.18	0.27	0.90	0.31	0.14	0.26	0.94	1.00	0.40	0.46	0.26	0.61	0.87	0.64	0.28	0.28	0.95	0.76
12	V11	1.00	0.35	0.81	0.48	0.75	0.81	0.71	0.87	0.67	0.40	1.00	0.49	0.54	0.54	0.79	0.72	0.32	0.52	0.64	0.79
13	V12	0.49	0.95	0.10	0.33	0.57	0.55	0.41	0.43	0.53	0.46	0.49	1.00	0.27	0.32	0.57	0.40	0.17	0.05	0.51	0.47
14	V13	0.54	0.30	0.83	0.23	0.08	0.35	0.55	0.53	0.01	0.26	0.54	0.27	1.00	0.03	0.11	0.12	0.53	0.68	0.00	0.11
15	V14	0.54	0.19	0.32	0.17	0.67	0.08	0.01	0.14	0.68	0.61	0.54	0.32	0.03	1.00	0.67	0.88	0.39	0.33	0.70	0.67
16	V15	0.79	0.38	0.56	0.48	1.00	0.65	0.46	0.66	0.98	0.87	0.79	0.57	0.11	0.67	1.00	0.79	0.03	0.10	0.97	0.92
17	V16	0.72	0.21	0.52	0.18	0.78	0.35	0.14	0.41	0.76	0.64	0.72	0.40	0.12	0.88	0.79	1.00	0.33	0.14	0.76	0.85
18	V17	0.32	0.31	0.30	0.21	0.06	0.45	0.85	0.44	0.12	0.28	0.32	0.17	0.53	0.39	0.03	0.33	1.00	0.75	0.14	0.17
19	V18	0.52	0.08	0.55	0.38	0.06	0.67	0.75	0.76	0.03	0.28	0.52	0.05	0.68	0.33	0.10	0.14	0.75	1.00	0.06	0.08
20	V20	0.64	0.33	0.43	0.34	0.98	0.48	0.32	0.48	1.00	0.95	0.64	0.51	0.00	0.70	0.97	0.76	0.14	0.06	1.00	0.86

Fig. 1. Correlation matrix

When this happens only one (it does not matter which and this election is left to the user) is retained reducing the number of independent variables (in this example, from 20 to the 12 shown in figure 2). For simplicity, the values have been re-labeled. Assume, then, that we have such table of values. There we have ordered the variables ($V_1, \ldots, V_{12}$) according to the date of registration. The oldest data are shown first; the newest last. What we would like to do is forecast (in this example) the values of V_{12} for $k=10$. That is, we want to find a function of the form

$$V_{12}(t+10) = f(V_1(t),...,V_{12}(t)) \tag{4}$$

To do this we simply set the values of a (new) dependent variable V_{12} at time $t+10$ as a function of the variables V_i at time t as shown in figure 2. The values of column V_{12} are repeated on $V_{12}(t+10)$ starting at the 12-th row. Notice that all variables have been scaled to the interval [0,1]. This is done to avoid the induction of a variable's spurious importance because of different measurement units. It also has to do with the activation function of the perceptrons we used. We applied a *tanh* function. It is symmetric to the X axis and has maxima at ± 1, as shown in figure 3.

1	MONTH	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V12(t+10)
2	2.00	1.00	0.41	0.15	0.59	0.50	0.05	0.08	0.96	1.00	0.30	0.04	0.07	0.12
3	2.10	0.99	0.40	0.13	0.58	0.49	0.06	0.08	0.94	0.99	0.30	0.04	0.07	0.12
4	2.20	0.99	0.40	0.12	0.56	0.48	0.07	0.07	0.93	0.97	0.30	0.04	0.08	0.13
5	2.30	0.98	0.39	0.10	0.55	0.47	0.07	0.06	0.92	0.96	0.31	0.04	0.08	0.13
6	2.40	0.97	0.39	0.09	0.54	0.46	0.08	0.05	0.90	0.94	0.31	0.04	0.09	0.13
7	2.50	0.96	0.38	0.07	0.52	0.45	0.09	0.04	0.89	0.93	0.31	0.04	0.09	0.14
8	2.60	0.96	0.37	0.06	0.51	0.44	0.09	0.03	0.88	0.91	0.32	0.04	0.10	0.14
9	2.70	0.95	0.37	0.04	0.50	0.42	0.10	0.03	0.86	0.90	0.32	0.04	0.10	0.15
10	2.80	0.94	0.36	0.03	0.48	0.41	0.11	0.02	0.85	0.89	0.32	0.04	0.11	0.15
11	2.90	0.94	0.36	0.01	0.47	0.40	0.11	0.01	0.84	0.87	0.33	0.04	0.11	0.16
12	3.00	0.93	0.35	0.00	0.46	0.39	0.12	0.00	0.82	0.86	0.33	0.04	0.12	0.16
13	3.10	0.92	0.41	0.06	0.45	0.38	0.13	0.06	0.81	0.84	0.34	0.04	0.12	0.16
14	3.20	0.91	0.47	0.12	0.44	0.36	0.14	0.13	0.79	0.82	0.34	0.04	0.13	0.15
15	3.30	0.91	0.53	0.18	0.43	0.34	0.14	0.19	0.78	0.81	0.34	0.04	0.13	0.15
16	3.40	0.90	0.59	0.24	0.42	0.32	0.15	0.26	0.76	0.79	0.35	0.04	0.13	0.14
17	3.50	0.89	0.66	0.29	0.41	0.31	0.16	0.32	0.75	0.77	0.35	0.05	0.14	0.14
18	3.60	0.89	0.72	0.35	0.40	0.29	0.17	0.39	0.73	0.76	0.36	0.05	0.14	0.13
19	3.70	0.88	0.78	0.41	0.39	0.27	0.18	0.45	0.71	0.74	0.36	0.05	0.15	0.13
20	3.80	0.87	0.84	0.47	0.38	0.25	0.18	0.52	0.70	0.72	0.37	0.05	0.15	0.13

Fig. 2. Original scaled data

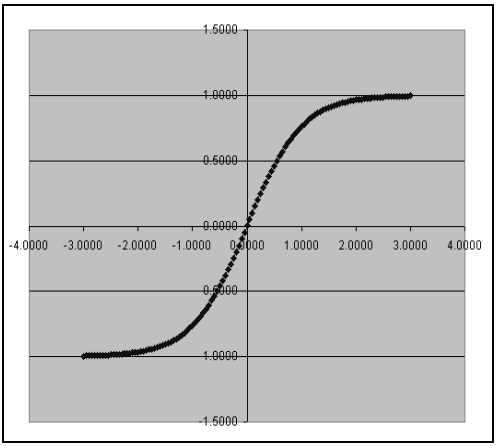

Fig. 3. Hyperbolic Tangent

3 Neural Network Architecture

We now set the NN which will approximate (4) as shown in figure 4. The number of input neurons is determined by the problem and corresponds to the number of input variables. The number of output neurons corresponds to the number of variables to be predicted. Rather than finding a single network for all the desired outputs (in this case 12) we decided to find one NN for each variable thus needing 12 networks. The number of layers (in this case 3) is determined by considering that any three-layered

network is sufficient to approximate a continuous function [3]. We guaranteed such continuous behavior by interpolating with a natural spline [4]. Every neuron in the network consists of an adder whose output is fed to an activation function as shown in figure 5. In our models all activation functions in the input and output layers are linear while the ones in the hidden layer are *tanh*. This configuration has been shown [14] to work properly for regression problems such as this. The number of neurons in the hidden layer was determined from the heuristic

$$H \approx (S - 3O)/[3(I + O + 1)] \tag{5}$$

where H = number of neurons in the hidden layer, S = number of objects in the data set, I/O = number of neurons in the input/output layer. The intuition behind (5) is that the number of connections should be enough for the weights to express the information present in the data but not so many as to deprive the NN of its generalization properties. Therefore, they should be approximately equal to one third the number of elements in the data. From (5) we determined that *H=3*.

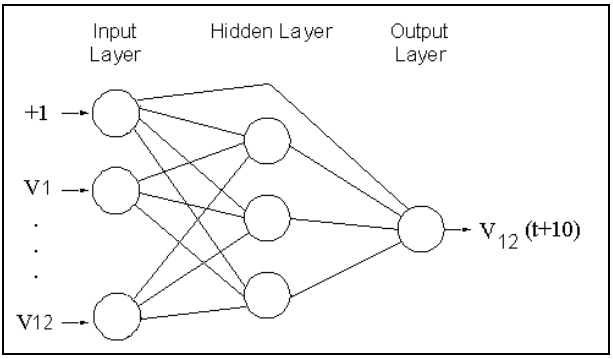

Fig. 4. A Neural Network for $V_{12}(t+10)$

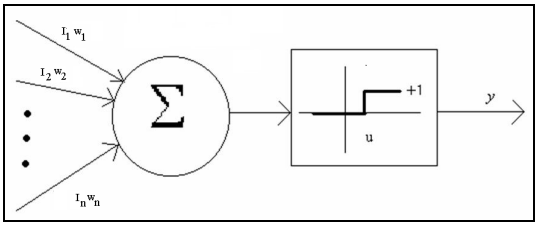

Fig. 5. A Perceptron

4 Training the Neural Networks

To train out NN's we used a GA rather than the classical backpropagation algorithm [5], [6]. The GA we used was not the Simple GA postulated by Holland but the so-called Eclectic GA (EGA) proposed in [7]. Use of the EGA frees the designer of the NN from the specification of the parameters associated to the BPA such as learning rates and

moments. The values of these four parameters (2 for each of the H and O layers) are difficult to determine and have a definite impact on the behavior of the training phase [8]. We represent every weight as a fixed point signed binary number. The structure of the chromosome used to train the NN is shown in figures 6 and 7.

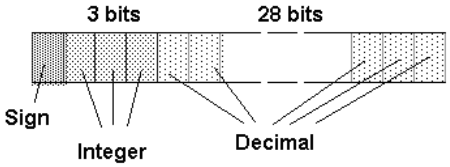

Fig. 6. Representation of one weight

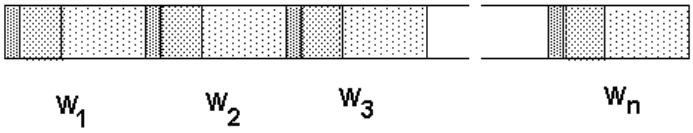

Fig. 7. Full Genome for the Neural Network

Furthermore, by training the NN with EGA we are able to select any measure of error to determine the suitability of the calculated weight. We implemented the following four norms: $L_e = \Sigma 10^{|\varepsilon_i|}$, $L_1 = \Sigma |\varepsilon_i|$, $L_2 = \Sigma \varepsilon_i^2$ and $L_\infty = \max(|\varepsilon_i|)$; where by ε_i we denote the approximation error for the *i-th* object of the sample. Each one of these norms may be used to determine the values of the weights by minimizing the ε_i's relative to the whole data set. On the other hand, BPA, as is well known, is restricted to L_2. The data set was divided into a training set (TRS) and a test set (TSS). TRS consisted of a uniform random sample of 75% of the objects in the data set. TSS consisted of the remaining objects. During the training phase, the NN's weights were evolved with EGA for 500 epochs. For every 25 epochs the NN was tested vs. TSS. The weights corresponding to best error in the test phase were kept. Therefore, the model is the one whose performance is the best for TSS and not for TRS. This strategy ensures that the NN will have the best possible predictive capabilities *outside* the known data. This strategy corresponds to what is usually called cross-validation. Hence, the resulting model is completely described by a) The number of neurons in layers I-H-O, b) The weights of the connections between the layers and c) The kind of activation functions. In figure 8 the model for $V_{12}(t+10)$ is shown. The first line shows the number of neurons in layers $I-H-O$. The last line describes the activation functions for layers $I-H-O$; where 0 means "linear", 1 means "logistic" and 2 stands for "tanh". In this NN there are (I+1)H (39) connections from layer I to layer H; H+1 (4) connections from layer H to layer O. Thus, the first 39 weights shown correspond to the I-H connections; the last four to the H-O connections.

12	3	1			
1.086833	1.311453	0.322658	0.048754	-0.838478	-0.803808
0.053508	0.318320	-0.020897	0.457155	0.516766	-1.034236
-0.287916	0.076564	-0.026285	-0.002845	-0.090761	0.047448
-0.016675	-0.087652	-0.081907	0.045386	-0.089507	-0.163964
0.124563	0.280718	0.209329	-0.025749	-0.039398	0.088745
0.399233	-0.154685	0.285988	-0.261510	-0.101763	-0.280015
-0.128920	0.124525	1.041826	0.511389	-0.398505	0.182359
0.529659					
0	2	0			

Fig. 8. The model for $V_{12}(t+10)$

The models corresponding to variables $V_i(t+10)$ individually take the place of the $v_i(\tau) = f_i(v_1(t),...,v_n(t))$ of equation (1). Notice that, because of cross validation, we are able to determine the approximation error of our models. In figure 9 we compare the actual values of V_{12} and those of NN_{12}. In figure 10 we show the approximation errors in NN_{12}. Notice the scale. The average of the percentage error ε_i was 0.011958 while the largest ε_i was 0.073768. Hence, as anticipated, we have an uncertainty of $\approx \pm 0.074$ on the predicted values. The simplified cost function $C(\tau)$ for this system was determined, from the problem's conditions, to be

$$C(\tau) = 2.35v1(\tau)v3(\tau) + 6.17v5(\tau)v12(\tau) + 3.4v12(\tau)v5(\tau) - 2v1(\tau)v3(\tau)v12(\tau) + 154 \quad (6)$$

Notice that not all of the $v_i(t)$ participate in equation (6) directly. However, they do form part of the functions for $NN_j(t)$ for $j=1,3,5,12$ since $NN_i(t)=f(v_1(t),\ldots,v_{12}(t))$. Accordingly, we determined the models for NN_1, NN_3, NN_5 and NN_{12}.

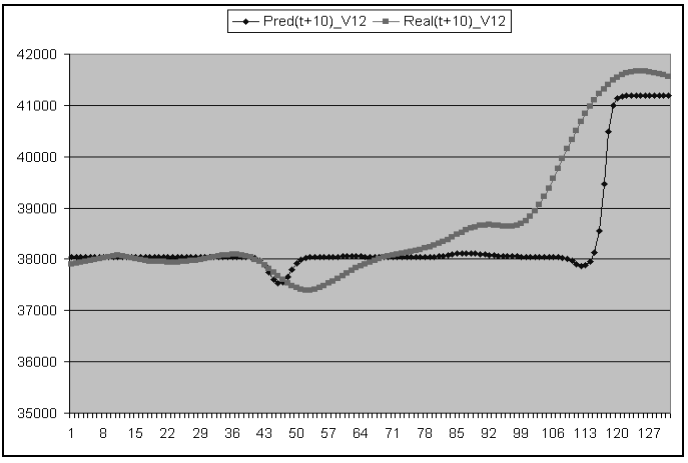

Fig. 9. Real vs. predicted values for $V_{12}(t+10)$

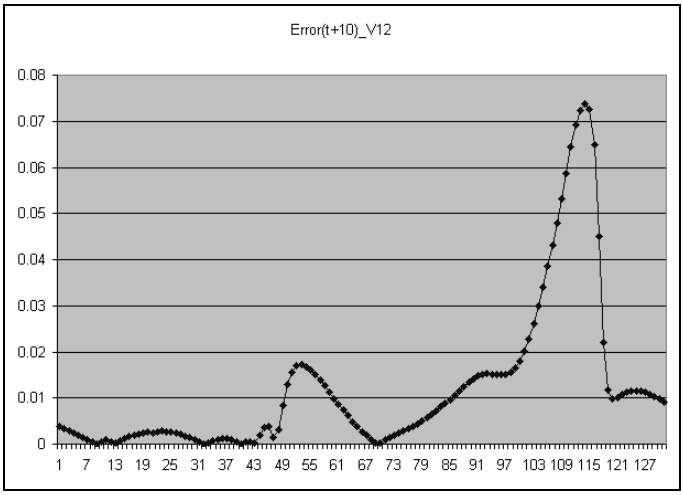

Fig. 10. Error values for $V_{12}(t+10)$

The real vs. predicted values for $V_1(\tau)$, $V_3(\tau)$ and $V_5(\tau)$ are shown in figures 11, 12 and 13 respectively. From the NN models we calculated $\varepsilon 1 = \pm 0.031$, $\varepsilon 3 = \pm 0.072$, $\varepsilon 5 = \pm 0.012$ and, as already pointed out, $\varepsilon 12 \approx \pm 0.074$.

5 Optimizing the Cost Function

Figures 7, 9, 10 and 11 give a graphical feeling of the good fit of the forecasting NNs. And the worst case error, although not necessarily homogeneous for all the values, is always less than 8%. In any case, this possible error, small as it may be, introduces an inherent uncertainty on the values of the target variables V_1, V_3, V_5 and V_{12}. To take into consideration the possible errors we proceeded to minimize equation (6) with the constraints included below and for those values predicted by NN_i.

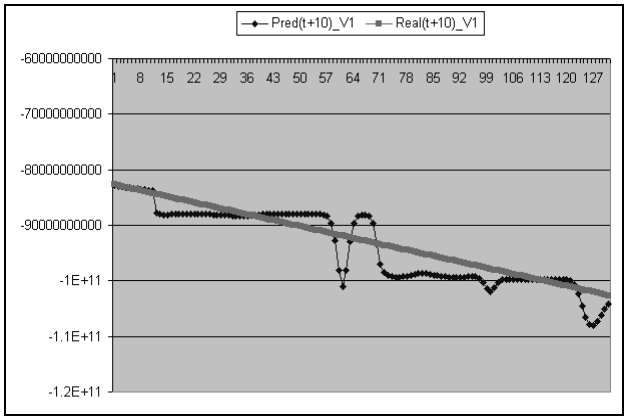

Fig. 11. Real vs. predicted values for $V_1(t+10)$

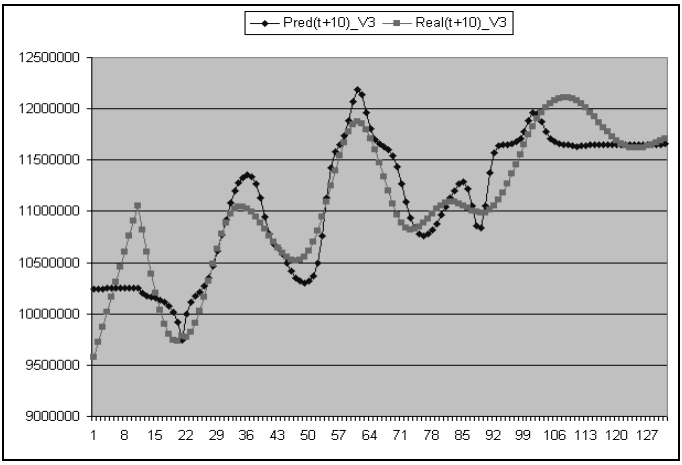

Fig. 12. Real vs. predicted values for V3(t+10)

In table 1 we have included the values of the $V_i(t+1)$ for $i=1,3,5,12$, as per $NN_i(t)$. We also include the ε_i from which the lower and upper values of the predicted variables will be inferred:

$$V_i^*(t+10) - \varepsilon_i \leq V_i(t+10) \leq V_i^*(t+10) + \varepsilon_i$$

Table 1. Forecasted values en percentage errors for $t+10$

i	$V_i^*(t+10)$	ε_i
1	*-1,026.24*	*0.031*
3	*116.975*	*0.072*
5	*1,882.04*	*0.012*
12	*415.70*	*0.074*

From table 1 we may calculate the constraints that apply to the minimization process of equation (6). The full minimization problem, with its restrictions, is shown next.

Minimize

$$C(\tau) = 2.35v1(\tau)v3(\tau) + 6.17v5(\tau)v12(\tau) + 3.4v12(\tau)v5(\tau) - 2v1(\tau)v3(\tau)v12(\tau) + 154 \quad .$$

Subject to

1) $-1{,}058.04 \leq V_1(\tau) \leq -994.42$
2) $108.60 \leq V_2(\tau) \leq 125.40$
3) $1861.4 \leq V_5(\tau) \leq 1906.65$
4) $384.93 \leq V_{12}(\tau) \leq 446.45$

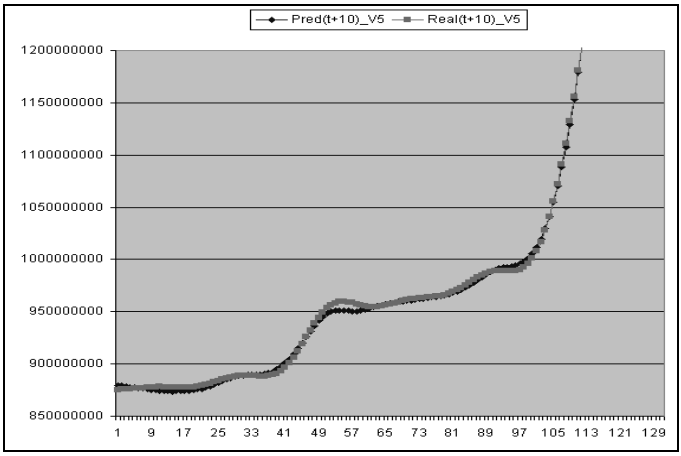

Fig. 13. Real vs. predicted values for V5(t+10)

In order for a GA to tackle constrained problems, it is necessary to define a penalty function [9]. We used the method described in [10]. It consists of defining the penalty function as follows:

$$P(\vec{x}) = \begin{cases} \left[K - \sum_{i=1}^{s} \frac{K}{p} \right] - f(\vec{x}) & s \neq p \\ 0 & otherwise \end{cases}$$

where K is a large constant [$O(10^9)$], p is the number of constraints and s is the number of these which have been satisfied. K's only restriction is that it should be large enough to insure that any non-feasible individual is graded much more poorly than any feasible one. Here the algorithm receives information as to how many constraints have been satisfied but is not otherwise affected by the strategy. Furthermore, to solve the problem we used a fixed point format with I bits for the integer and D bits for the decimal plus one sign bit. Therefore, the size of the chromosome (κ) is

$$\kappa = | Chromosome | = (I + D + 1) \times Number \ \ of \ \ variables \tag{7}$$

With these specifications we set the Genetic Algorithm as follows:

a) Number of individuals → 250
b) Bits for integers → 12
c) Bits for decimals → 32
   $\kappa = (12+32+1) \times 4 = 180$
d) $P_c=0.9$
e) $P_m=0.01$
f) Number of generations → 4,000

For these settings the results were as follows:

$$V_1^*(\tau) = -994.42$$
$$V_3^*(\tau) = +108.625$$
$$V_5^*(\tau) = +1{,}861.43$$
$$V_{12}^*(\tau) = +384.9302$$

for which $C(\tau) \approx 89{,}762{,}858.74$. The reader may verify that these values comply with all the restrictions. Furthermore, the value of $C(\tau)$, which reflects the monthly cost of operation of the system, lies well under the typical average costs of the actual day to day operation of the system.

6 Conclusions

We have shown that a complex dynamic system may be analyzed from its past behavior and, given a known cost function, the relevant variables for its operation may be calculated in advance in order for the system to perform close to the optimum. To do this we found a set of multivariate functions in the form of Multi-Layered Perceptron Networks. Once these were found, we were able to optimize the forecasted function's values by including an uncertainty factor. This factor has to do with the estimated forecasting error. This consideration led us to consider a constrained optimization problem which we tackled using a non-traditional genetic algorithm which has been cited in the literature as superior to other evolutionary alternatives [11]. We stress that, for this method to work adequately, it is very important to observe a series of pre-processing steps. In our case these were: a) Elimination of highly correlated variables, b) Scaling the values of the variables to make them scale independent, c) Optimal interpolation of the known values to enhance the data sample, d) Proper selection of the network's architecture, e) Proper selection of the error measure. Once all these issues have been dealt with, we proceeded to split the data in two samples in order for us to cross-validate the NNs after the training phase. Finally, an agile treatment of the constraints is needed for the GA to find values closer to the optimum.

This is a prime example of computational intelligence put to work; where classical statistical methods are disallowed because of the high number of variables. Likewise, classical approximation methods rarely will be useful in practice because of the inherent high order of the closed approximants and the related numerical instability [12], [13] which often crops up when solving large systems of equations.

Given all of the above considerations, we think that reliable forecasting and optimization is possible given that the data under study do not exhibit chaotic behavior. In the latter case no forecasting algorithm will perform properly. However, for a well behaved problem, such as the one discussed here, we may extract the patterns embedded in the data to our advantage and are able to optimize such forecasted patterns.

References

1. Haykin, S.: Neural Networks. A comprehensive foundation, 2nd edn. Prentice Hall (1999)
2. Back, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
3. MIT Staff, Machine Monitoring and Diagnosis using Knowledge-Based Fuzzy Systems. Data Engine Tutorials and Theory, Management Intelligenter Technologien GmbH, Aachen, Germany (2010)
4. Shampine, L., Allen, R.: Numerical Computing: an introduction, pp. 43–53. W.B. Saunders (1984)
5. Rumelhart, D.E., Hintoen, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (eds.) PDP Research group, Parallel Distributed Processing. Foundations, vol. I. MIT Press (1986)
6. Montana, D.J., Davis, L.D.: Training feedforward Networks using Genetic Algorithms. In: Proceedings of the International Joint Conference on Artificial Intelligence. Morgan Kauffman (1989)
7. Kuri, A.: A universal Eclectic Genetic Algorithm for Constrained Optimization. In: Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT 1998, pp. 518–522 (1998)
8. Yao, X.: A Review of Evolutionary Artificial Neural Networks. International Journal of Intelligent Systems 8, 539–567 (1993)
9. Coello, C.: Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. Computer Methods in Applied Mechanics and Engineering (2001)
10. Kuri-Morales, Á.F., Gutiérrez-García, J.O.: Penalty Function Methods for Constrained Optimization with Genetic Algorithms: A Statistical Analysis. In: Coello Coello, C.A., de Albornoz, Á., Sucar, L.E., Battistutti, O.C. (eds.) MICAI 2002. LNCS (LNAI), vol. 2313, pp. 108–117. Springer, Heidelberg (2002)
11. Kuri-Morales, Á.F.: A Methodology for the Statistical Characterization of Genetic Algorithms. In: Coello Coello, C.A., de Albornoz, Á., Sucar, L.E., Battistutti, O.C. (eds.) MICAI 2002. LNCS (LNAI), vol. 2313, pp. 79–89. Springer, Heidelberg (2002)
12. Scheid, F.: Theory and Problems of Numerical Analysis, 7th edn. McGraw-Hill Book Company (1997)
13. Cheney, E.W.: Introduction to Approximation Theory, ch. 2, pp. 58–64. McGraw-Hill Book Company (1966)
14. Tryba, V., Kiziloglu, B., Thimm, A., Daehn, W.: Bestimmung der Abwasserqualität mit einem Multilayerperceptron-Netz für die Online-Steuerung von Regenüberlaufbecken. In: Anwendersymposium Erlangen 1996, pp. S131–S137. MIT GmbH (1996)