An Evolutionary Artificial Neural Network Time Series Forecasting System

Paulo Cortez
paulo@di-ia.uminho.pt

José Machado
jmac@di-ia.uminho.pt

José Neves
jneves@di-ia.uminho.pt

Departamento de Informática
Universidade do Minho
Largo do Paço, 4709 Braga Codex, PORTUGAL
Voice: +351(53)604470 Fax: +351(53)604471

Keywords: Neural Networks, Genetic Algorithms, Time Series.

Abstract

Artificial Neural Networks (ANNs) have the ability of learning and to adapt to new situations by recognizing patterns in previous data. *Time Series (TS)* (observations ordered in time) often present a high degree of noise which difficults forecasting. Using *ANNs* for *Time Series Forecasting (TSF)* may be appealing. However, the main problem with this approach is on the search for the best *ANN* architecture. *Genetic Algorithms (GAs)* are suited for problems of combinatorial nature, where other methods seem to fail. Therefore, an integration of *ANNs* and *GAs* for *TSF*, taking the advantages of both methods, may be appealing. *ANNs* will learn to forecast by back-propagation. Different *ANNs* architectures will give different forecasts, leading to competition. At the end of the evolutionary process the resulting *ANN* is expected to return the best possible forecast. It is asserted that the combined strategy exceeded conventional *TSF* methods on *TS* of high non-linear degree, particularly for long term forecasts.

1 Introduction

With the emerging of open markets competition between enterprises will become more and more tough. Winning means the need of strategic advantages. However, innovating in an uncertain environment can be disastrous. An organization may be interested in predicting the future based on previous data, in terms of the variables that rule its practice. One possibility is the use of *TSF*, based only on previous observations of the same variable, which has the advantage of modeling complex systems as black-boxes [8]. The goal is to predict the system behavior and not how the system works. Organizations normally work with two different kinds of forecasts: short term predictions, few predictions ahead, for current management decisions (eg. dealing with stocks), and long term predictions, for planning (eg. elaborating budgets) [8]. Conventional *TSF* methods base their models on *TS* components such as trend and seasonal effects [11]. Well behaved *TS* are suited for this kind of approach but fails when *TS* present strong noise or non-linear components.

Optimization algorithms, imitating nature, have become popular in both academy and industry. Studies on the nervous system and biological evolution influenced the rise of *ANNs* and *GAs*. *ANNs* have the advantage of learning from previous data in order to respond to new situations, specially when one is dealing with incomplete information or a strong noise component.

In principle *ANNs* can imitate any computable function. This is the main reason why *ANNs* have been applied so successfully in many different areas: expert systems, data filtering, computer vision or economy [5]. *ANNs* seem to be appropriate for classification or approximation problems that are tolerant to some errors, with a lot of available data and where well defined rules can not be applied. *ANNs* seem to be a good strategy for *TSF*. One drawback of using *ANNs* is the time required for the search of the best *ANN*. This is a heavy empirical process, where intuition, heuristic rules or random search are often used.

GAs are suited for combinatorial problems, where the exhaustion of all possible solutions require enormous computational power. *GAs* can heuristically find solutions where other methods seem to fail. Although *GAs* and *ANNs* are both search strategies, empirical studies show that they

Table 1: Activation Functions

Name	Function $f(x)$	Codomain
linear	x	$]-\infty,+\infty[$
sigmoid	$\frac{1}{1+e(-x)}$	$[0,1]$
sigmoid1	$\frac{2}{1+e(-x)}-1$	$[-1,1]$
sigmoid2	$\frac{x}{1+\lvert x\rvert}$	$[-1,1]$
tanh	$tanh(x)$	$[-1,1]$

differ in range [9]. *GAs* seem to perform a superior global search than *ANNs* with back-propagation, although back-propagation will reach a better solution [10].

This work reports on the integration of *ANNs* and *GAs*, a *Genetic Algorithm Neural Network (GANN)* system [9], leading to *TSF*. *ANNs* will learn to forecast by back-propagation and better *ANN* architectures will be selected by evolution.

2 The Artificial Neural Network Architecture

There is no known rule that points out to what kind of *ANN* one must use. The choice is done by looking at the data, to the problem, and using intuition or experience [5][14].

One's approach for *TSF* supposes the use of feedforward *ANNs*. The decision was to use feedforward *ANNs* fully connected, with bias, without shortcut connections, and with one hidden layer, which makes possible to bound at a certain level the *ANN* parameters, in order to reduce the vectorial search space. The initial *ANN* weights were randomly generated within the range of $[\frac{-2}{z};\frac{2}{z}]$ for a node with z inputs. Standard back-propagation [5] was used for the *ANN* training. Table 1 shows the activation functions used.

The *ANN* topology will be represented by $L_i - L_h - L_o$ for an *ANN* with L_i input nodes, L_h hidden nodes and L_o output nodes [14]. In this work L_o is always equal to one. Empirical tests indicated that, with the same data, *ANNs* with one output node gave better forecasts than *ANNs* with n output nodes, even for n prediction steps. The *ANN* receives n previous *TS* values, for n input nodes, and outputs the forecasted value. The training cases were built as follows:

$$\begin{array}{lcl} x_1, x_2, \ldots, x_k & \rightarrow & x_{k+1} \\ x_2, x_3, \ldots, x_{k+1} & \rightarrow & x_{k+2} \\ \ldots & \rightarrow & \ldots \\ x_{p-k}, \ldots, x_{p-1} & \rightarrow & x_p \end{array}$$

where $x_1, \ldots, x_p$ represent the *TS* and k the number of the ANN's inputs. The cases are trained in this order due to the temporal nature of the data. The trained *ANN* can make short or long term predictions:

$$\begin{array}{l} f_{1,p} = output1(x_{p-k}, \ldots, x_p) \\ f_{2,p} = output1(x_{p-k+1}, \ldots, x_p, f_{1,p}) \\ f_{3,p} = output1(x_{p-k+2}, \ldots, x_p, f_{1,p}, f_{2,p}) \\ \ldots \\ f_{n,p} = output1(f_{n-k,p}, \ldots, f_{n-1,p}) \end{array}$$

where $output1(x_1, \ldots, x_k)$ caters for the output function of the *ANN*, and $f_{i,j}$ the forecast in the j period to i periods ahead of j.

One problem that arises with back-propagation training is overfitting: after some training epochs the *ANN* starts to loose generalization. To overcome this situation, early stopping was considered [14]. The training data was divided into the training set (for the *ANN* learning) and the validation set (to test the *ANN* generalization capacity). By default the validation set is 10% of the training data, since the use of the last (recent) values in the training set improves the forecast.

The best *ANN* architecture for short and long term forecasts tends to differ. On long term forecasts *ANNs* seem to be more affected by overfitting. Thus a special validation test, *Specific Mean Square Error (SMSE)*, that takes into consideration the number of forecasts ahead, was implemented:

$$\begin{array}{l} MSE_{n,j} = \frac{(f_{1,j}-x_{j+1})^2+(f_{2,j}-x_{j+2})^2+\ldots+(f_{n,j}-x_{j+n})^2}{n} \\ SMSE_n = \frac{\sum_{j=p-k-n}^{p-n} MSE_{n,j}}{k} \end{array}$$

where p is the last period of the *TS*, k the number of validation cases and n the number of forecasting steps. $MSE_{n,j}$ is the error measure for n forecasting steps from j, and n is an user defined parameter.

3 Combining Genetic Algorithms with Artificial Neural Networks

When using feedforward *ANNs* a number of parameters have to be set before any training: the *ANN* topology, the connections between nodes, the activation function, the training algorithm and its parameters. The search for the best *ANN* goes through a long empirical process, of combinatorial nature. Some rules are used in order to simplify the process: the use of only one hidden layer; choosing a training algorithm that fits to the problem (generally of the back-propagation family); the use of the standard parameters for the algorithm and the use of fully-connected *ANNs* (which avoids the need to define the *ANN's* connections).

To set the remaining of the parameters one can use random search or *GAs*. The last ones are known of being more effective and efficient [6].

GANNs systems optimize *ANNs* in two ways: evolution by the *GA* and learning by back-propagation [10] [7]. The back-propagation learning guides the evolutionary search. Chromosomes (encoded *ANNs*) near optimum will share bit patterns, permitting the *GA* to exploit them by hiperplane sampling. *GANNs* systems use to present serious drawbacks, in particular when long genomes imply the use of large *ANNs* [9]. That is not the situation in this work, once *TSF* may be efficiently modelled by small *ANNs* [4].

When setting the *GAs* parameters one has to compromise between the optimum *ANN* and the computational time required. Therefore, some hard decisions were to be made, namely: the use of a population of 20 individuals; the selection by the classical fitness-based roulette-wheel; one point crossover with a crossover rate of 1 and a mutation rate of 0.02. The random seed for the initial population was 12345. The fitness function uses the *SMSE* for the validation cases: $fitness = \frac{1}{SMSE}$.

The chromosome with the lower *SMSE* will automatically survive for the next generation. This purification process accelerates the search and also warranties that if a optimum *ANN* is reached then it will never be lost. There are 6 (six) factors that affect a good forecast: the number of input nodes, the learning rate, the activation function, the number of hidden nodes, the scaling constant and the random weights initialization seed. Encoding a chromosome: the first three factors, the crticial ones, were put at the left of the chromosome in a direct encoding using base 2 gray codes.

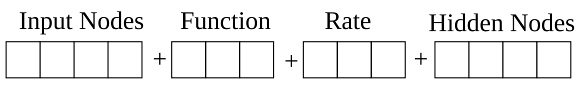

Figure 1: Neural Network Encoding

The number of input and hidden nodes were set within the range $[3, 14]$ allowing a 4 bits binary encoding. The learning rate was encoded in 3 bits taking values from the set $\{0.1, 0.2, \cdots, 0.7, 0.8\}$. The 5 (five) activation functions were encoded in 3 bits, which allows for 3 do not care values, implying that new bits have to be validated upon creation. The scaling constant is set by the user (which gets the highest *TS* value as an indication). The main reason for this is that forecasts near optimum are archived by scaling constants that are near the forecasting values. Therefore, if a series presents a growing trend then the *GA* would find a scaling value very close to the validation data. That would result in forecasts bellow real values. The last parameter (random weights initialization seed) is not encoded since its influence in the forecast is random (by definition).

Table 2: GANN's results for $f_{1,p}$ forecasting

S	Size	T	N	Topology	F	R	MSE
1	100	81	10	$8-8-1$	sigmoid1	0.7	138.5
2	130	106	12	$12-12-1$	sigmoid2	0.1	1656018
3	144	119	12	$9-9-1$	sigmoid	0.7	3037
4	369	339	10	$6-5-1$	sigmoid	0.8	62.4

The whole system was developed using the UNIX operating system, the SICStus Prolog and C languages, with a X-Windows interface, and implemented on a TCP/IP based network of workstations [2][13][12]. A X-Windows interface was built using the SICStus **gmlib** library [1].

Before building a model the system checks for series' randomness. The test uses the autocorrelation coefficient, which gives a measure of the correlation between a series and itself, lagged K periods:

$$r_k = \frac{\sum_{t=1}^{n-k}(Y_t - \overline{Y})(Y_{t+k} - \overline{Y})}{\sum_{t=1}^{n}(Y_t - \overline{Y})}$$

All autocorrelations values should theoretically be zero for a random series. In practice a series is considered to be random if all K lags autocorrelation values are within the range:

$$-1.96 * \frac{1}{\sqrt{n}} \leq r_k \leq 1.96 * \frac{1}{\sqrt{n}}$$

If a series is random them, instead of running the *GA*, the system returns the average of the *TS* (which is the only possible forecast in this case).

4 The System Results

The results for short term forecasting are shown in table 2, where S stands for $Series$. The parameters are: T for the number of series values in the training set and N for the number of forecasts. Optimum *ANN* parameters are: F for the *function* and R for the *learning rate*.

The system results were compared with the Holt-Winters [11] and ARIMA [3] methods (table 3). For series

Table 3: Comparing GANN's MSE results with conventional methods for $f_{1,i}$ forecasting.

S	GANN	ARIMA	Holt-Winters
1	138.5	163.8/256	–
2	1656018	–	1348153
3	3037	–	270
4	62.4	67.2	103.6

1 and 4 the system outperforms the conventional methods (ARIMA and Holt-Winters). In series 2 and 3 the Holt-Winters model performs best results. All the restrictions

Table 4: Comparing the $SMSE_1$ (or MSE) validation test with the $SMSE_n$ test for long term forecasts

S	n	Topology	F	R	p	$MSE_{10,p}$
1	1	$8-8-1$	sigmoid1	0.7	90	286
1	8	$8-10-1$	sigmoid2	0.6	90	235.9
4	1	$6-5-1$	sigmoid	0.8	359	133.7
4	10	$6-14-1$	sigmoid	0.8	359	105.5

Table 5: The $MSE_{10,p}$ for each forecasting method.

S	GANN	ARIMA	Holt-Winters
1	235.9	262.4/278.9	–
4	105.5	–	377.9

imposed on the *GANN* system are probably the main reason for this behavior.

Table 4 shows the validity of the *SMSE* test on long term forecasts. Note that the best *ANNs* for long term forecasts are slightly different than the short term ones. Series 2 and 3 don't appear in this table since the short term system errors were higher than long term ones obtained with conventional methods. Short terms errors are smaller than the long ones, so there was no need to waste computational power. For long term forecasts the system results are even better (see table 5).

5 Conclusions and Future Work

GANN's systems can be very attractive on *TSF* for time series with a high non-linear degree, specially for long term forecasts. However there are some drawbacks: a high computational cost. One advantage of this system is that it works with a minimum of human intervention. For future work one intends to explore different *ANN* architecture, to experiment on other genetic algorithm parameters and operators, and develop and apply software agent-based technology to the *GANN* approach.

References

[1] J. Almgren, S. Anderson, M. Carlsson, L.Floyd, S. Haridi, C. Frisk, H.Nilsson, and J.Sundberg. *SICStus Prolog User's Manual.* Swedish Institute of Computer Science, Sweden., 1993.

[2] O. Belo and J. Neves. A Prolog Implementation of a Distributed Computing Environment for Multi-Agent Systems Based Applications. In *Proceedings of The Third International Conference on Practical Application of Prolog, PAP'95*, Paris, France, 1995.

[3] G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control.* Holden Day, San Francisco, USA, 1976.

[4] P. Cortez, M. Rocha, J. Machado, and J. Neves. A Neural Network Based Forecasting System. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, Western Australia, November 1995.

[5] S. Gallant. *Neural Network Learning and Expert Systems.* MIT Press, Cambridge, USA, 1993.

[6] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., NY, USA, 1989.

[7] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation,*, 3(1):213–233, MIT Pres, 1993. MIT Press.

[8] J. Hanke and A. Reitsch. *A Business Forecasting.* Allyn and Bancon Publishing Company Inc., Massachussetts, USA, 1989.

[9] K. Hiroaki. Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms. Eighth National Conference on Artificial Inteligence Vol. II, pp 788-795, AAAI, MIT Press., 1990.

[10] P. Koenh. Combining Genetic Algorithms and Neural Networks. Thesis for master science degree, University of Tennessee, Knoxville, USA., 1994.

[11] S. Makridakis, S. Weelwright, and V. McGee. *Forecasting: Methods and Applications.* John Wiley & Sons, New York, USA, 1978.

[12] J. Neves, J. Machado, L. Costa, and P. Cortez. A Software Agent Distributed System for Dynamic Load Balancing. In *Proceedings of ESM96, European Simulation Multi-Conference, Modelling and Simulation 1996*, Budapest, Hungary, June 1996.

[13] J. Neves, M. Santos, and V. Alves. An Adaptable and Dynamic Architecture for Distributed Problem Solving Based on the Blackboard Paradigm. In *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*, pages 378–385, Armidale, New South Wales, Australia, 1994.

[14] L. Prechelt. PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. Research Report, Fakultãt fũr Informatik, Universitãt Karlsruhe Germany, 1994.