

Design of Neural Networks for Time Series Prediction Using Case-Initialized Genetic Algorithms

Ricardo Bastos Cavalcante Prudêncio, Teresa Bernarda Ludermit

Centro de Informática, Universidade Federal de Pernambuco
Caixa Postal 7851 - CEP 50732-970 - Recife (PE) - Brasil
[rbcp, tbl]@cin.ufpe.br

Abstract

One of the major objectives of time series analysis is the design of time series models, used to support the decision-making in several application domains. Among the existing time series models, we highlight the Artificial Neural Networks (ANNs), which offer greater computational power than the classical linear models. However, as a drawback, the performance of ANNs is more vulnerable to wrong design decisions. One of the main difficulties of ANN's design is the selection of an adequate network's architecture. In this work, we propose the use of Case-Initialized Genetic Algorithms to help in the ANN's design. We maintain a case base in which each case associates a time series to a well-succeeded neural network used to predict it. Given a new time series, the most similar cases are retrieved and their solutions are inserted in the initial population of the Genetic Algorithms (GAs). Next, the GAs are executed and the best generated neural model is returned. In the undergone tests, the Case-Initialized GAs presented a better generalization performance than the GAs with random initialization. We expect that the results will be improved as more cases are inserted in the base.

1 Introduction

A time series is a realization of a process or phenomenon varying in time. Time series analysis is an inductive process that, from an observed time series, is capable of inferring general characteristics of the phenomenon which generated the series. Among the objectives of time series analysis, we highlight the design of time series prediction models. These models can be used to support the decision-making in several application domains, such as finance, industry, management, among others.

Some temporal phenomena can be conceptually modeled by the characteristics of the physical entities which influence on it. However, when there is not enough information available, the use of black-box models can be a good alternative. Among them, we highlight the Box-Jenkins [1] models and the Artificial

Neural Networks [2]. The latter approach is computationally more powerful, however the design of these networks is, in general, more complex and sensitive to wrong decisions.

In this work, we propose the use of Case-Initialized Genetic Algorithms (CIGAs) [3] in the design of neural networks for time series prediction problems. These algorithms are similar to the traditional Genetic Algorithms (GAs) [4], however here the first GA's population is generated from well-succeeded solutions used in problems similar to the one being tackled. Hence, the experience in solving past problems is used to solve new ones. The GAs have already been successfully used in the design of neural networks [5] [6]. As such, the case-initialization of Genetic Algorithms is a promising improvement in the traditional use of GAs for this problem.

We implemented a case base in which each case associates a time series to a well-succeeded neural network used to predict it. The neural network models deployed were the NARX e NARMAX networks [7], which will be briefly discussed in following section. The case base currently counts on 47 cases that are indexed and retrieved based on the serial autocorrelations, which reveal time dependencies in the series.

In the undergone tests, we compare the Case-Initialized GAs to GAs with random initialization. Both procedures were used to define neural models for three different time series. The Case-Initialized GAs generated neural networks with better generalization performance for the three time series used.

The case base is continuously being augmented and we expect that the results of the case-initialization will be better as the number of cases increases.

In section 2, we present concepts regarding time series models. Section 3 brings the proposed methodology for the design of neural networks. In section 4, we present implementation details of the initial prototype, and the tests and preliminary results can be found in section 5. Finally, we present the conclusion and future work in section 6.

2 Time Series Models

As said above, the analysis of a time series aims to identify its characteristics and main properties. Based on that, prediction models can be constructed and used to predict the process or phenomenon represented by the series under analysis. These prediction models can be deployed in a diversity of tasks, such as planning and control.

One kind of prediction model, called conceptual model, identifies the physical variables that significantly influence the phenomenon, and relates these variables by a parametric formula. Although a conceptual model provides a realistic interpretation of a phenomenon under analysis, it is not always possible conceptually describe very complex phenomena.

In the absence of physical insights about the domain, an alternative approach is the use of black-box models [7]. They model a time series through a function with adjustable parameters using as input the current and some past values of the series. Each class of black-box models deploys a basic set of functions which should be flexible enough to adequately describe the largest possible number of series.

One of the most widespread classes of black-box models for time series prediction is that developed by Box and Jenkins [1]. They model a time series through linear functions with few parameters. As they are linear, these models have a very limited computational power.

An alternative approach which implements non-linear models is via the use of Artificial Neural Networks (ANNs) [2]. They present a higher computational power when compared to the linear models since they are capable of modeling non-linear phenomena. Nevertheless, they are more vulnerable to the overfitting and local minima problems.

The NARX (Non-linear AutoRegressive model with eXogenous variables) network, described by equation 1, predicts a time series Z at time t using as regressors the last p values of an external variable U and the last p values of the series itself. The non-linear function f represents a feedforward network architecture and its weights. The input layer is usually known as the time-window.

$$(1) \quad Z(t) = f(U(t-1)...U(t-p) ... Z(t-1),..., Z(t-p)) + e(t)$$

The NARMAX (Non-linear AutoRegressive Moving Average model with eXogenous variables) networks predict a series using the same inputs of the NARX model plus the last q values of the prediction error, which form a context layer. This layer is supported by a recurrent connection from the output node. The NARMAX model can be describes by the equation 2.

$$(2) \quad Z(t) = f(U(t-1)...U(t-p), ..., Z(t-1), ..., Z(t-p), ..., e(t-1), ..., e(t-q)) + e(t)$$

We can see in figure 1(a) an example of a NARX network with time-window of length 2, and in 1(b) we present an example of a NARMAX network with both time-window and context layer of length 2.

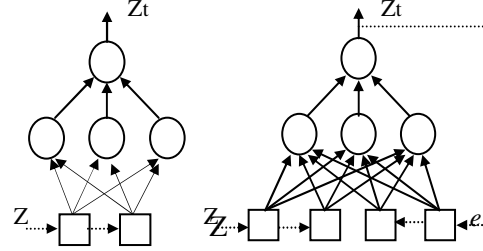


Figure 1: (a) NARX and (b) NARMAX networks

2.1 Design of Time Series Models

The design of time series models consists of three steps - identification, estimation and evaluation - briefly presented below.

Identification: In the Box-Jenkins models, this step determines the regressors, that is, how many past values of the series and how many past prediction errors will be used in the prediction. One of the most deployed tools in the identification of linear models is the autocorrelation analysis. The autocorrelation of order k measures the dependence between the values of the process at time t and at time $t-k$ and it can be estimated by the serial autocorrelations according to the equation:

$$(3) \quad r(k) = \frac{1}{N} \sum_{t=1}^N (Z(t) - \mu) * (Z(t-k) - \mu)$$

where N is the number of series's values and μ is the mean of the series.

In order to determine whether a given model is adequate to a series, we must compare a possible theoretical behavior of the model's autocorrelations to the behavior of the serial autocorrelations. The model will be chosen if these behaviors are similar (see [1] for details).

In Neural Networks, the identification step consists of defining the regressors plus the network's architecture. This task is more difficult than the identification of linear models since an inadequate architecture's choice may compromise the performance of the neural network. A small architecture may not be enough to model a given series, and a big architecture

may lead to overfitting and may also increase the number of local minima.

One approach that can be used in the identification of ANNs is to define the regressors based on the linear identification and then determine the best possible network's architecture for these regressors, either experimentally or using an automatic technique. A problem with this approach is that it is difficult to define the theoretical autocorrelation's behavior of complex models, such as the ANNs.

An alternative approach to identify neural networks is via the use of Genetic Algorithms [5]. In fact, the design of neural networks can be seen as a search problem and, hence, the application of a traditional search and optimization algorithms, such as GAs, is very adequate. In [6], the authors mention several characteristics of the search space of networks that motivate the use of GAs, among them, a non-differentiable, deceptive and multimodal surface. Another advantage of GAs is that, instead of treating each parameter of the network in isolation, they are able to define, at same time, several ANN's parameters, performing a global optimization in the search space of parameters. In [8], for example, GAs were successfully used to define the input variables, the number of hidden nodes, the activation function, and the learning parameters of a network for predicting a time series.

Estimation: After the identification of a model for the series under analysis, the estimation step determines the values of the model's adjustable parameters in order to minimize the prediction error. In Box-Jenkins models, this task consists of the application of a simple linear regression technique.

In the ANNs, this step corresponds to the training process, i.e. the learning of the ANN's weights. The ANN learning algorithms usually use a gradient-based technique [9]. In general, the estimation of linear models is faster and simpler than in ANNs due to a small quantity of adjustable parameters. Besides time efficiency, other issues must be addressed during ANNs training. First of all, a long training phase may lead to overfitting. Besides, a good learning algorithm must be deployed, in order to avoid local minima.

Evaluation: The evaluation step concerns the analysis of the prediction errors. A model is usually evaluated by the sum or average of the squared errors generated by the model, which must be as small as possible. Other desired characteristics of the prediction errors are randomness and normality. Clearly, each domain of application has its specific requirements, which must be used to evaluate the results generated by the model.

3 Case-Initialized GAs for ANN Design

As discussed in the previous section, ANNs have a strong computational power, however an adequate use of these models depends upon their design. Here, the identification step plays a crucial role.

The work presented here proposes a methodology to automate the design of neural networks based on the use of Case-Initialized Genetic Algorithms [3] during the identification step. The case-initialization of GAs, proposed to improve their performance, consists of generating the first GA's population from well-succeeded solutions to problems which are similar to the current one. The inspiration of this technique comes from the fact that similar problems have similar search spaces and, therefore, good solutions to a particular problem can provide information about the search space for similar problems. The case-initialization enables us to use the acquired experience in solving past problems to solve new ones.

The case-initialization, that shares some ideas with the Case-Based Reasoning methodology [10] [11], was successfully deployed in [3] to a particular problem, showing feasibility.

Although the focus of our work is the design of neural networks for time series prediction, our methodology can be deployed for different classes of problems, such as classification problems. Figure 2 depicts the proposed methodology.

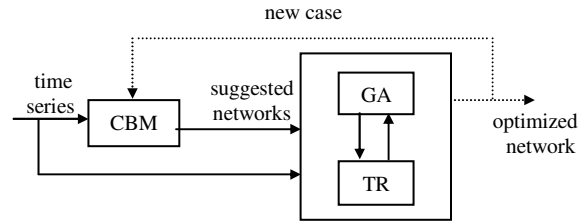


Figure 2: Proposed Methodology

The CBM module receives as input the problem being treated and retrieves a predefined number of cases, selected on the basis of their similarity to the input problem. Following, the Genetic Algorithm (GA) module inserts the networks associated to the retrieved series in the GA's initial population. Each network is trained by the Training module (TR), responsible for performing the learning of the network's weights. The output network will be the best one generated by the GA. Following, a new case may be created and inserted in the base associating the current series to the optimized network. The new cases are available for future use, in order to suggest more adequate networks for modeling other time series.

In what follows, we present some details about each of these modules.

3.1 CBM Module

This module maintains a case base in which each case associates a time series to a well-succeeded network used to predict it. The most important tasks performed by this module are retrieving similar cases from the base and inserting new cases in the base. In order to perform the first task, a similarity measure between time series must be specified, as well as an insertion criterion must be used to decide when a new case may be inserted in the base.

3.2 GA Module

This module implements a GA to determine an optimized network used for predicting an input time series.

Initially, a population of chromosomes is generated either randomly from the search space of networks or from the networks returned by the CBM module. Each chromosome represents a codification of an ANN. In order to evaluate the fitness function, each chromosome is translated into a neural network, which is then trained by the TR module (figure 3). Based on the training results, a fitness value is associated to each chromosome. The best chromosomes will be select to compound the next generation and the others will be discarded. This process runs for a predefined number of generations and the best generated chromosome is returned as the optimized network.

The most important points to define here are: the chromosome representation, the fitness function and the genetic operators. These points intrinsically depend on the type of neural networks chosen as time series models.

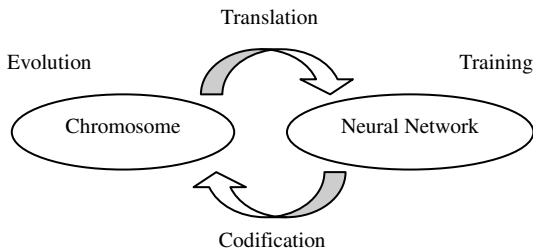


Figure 3: Optimization Scheme

3.3 TR Module

This module implements the training process, i.e., the estimation of the network's weights. It receives as input the definition of a neural network and a time series, returning the trained weights and an evaluation of the training process.

Among the points to define here, we quote: the training algorithm, the transformations, stopping criteria and performance measures. An ideal training

process should avoid local minima and overfitting problems, with a reasonable amount of computational effort.

4 Prototype

In this section, we present details about the implemented prototype. The models used for time series prediction were the NARX and NARMAX networks, described in section 2.1. In these models, the following three parameters are optimized: time-window length, context layer length and number of hidden nodes.

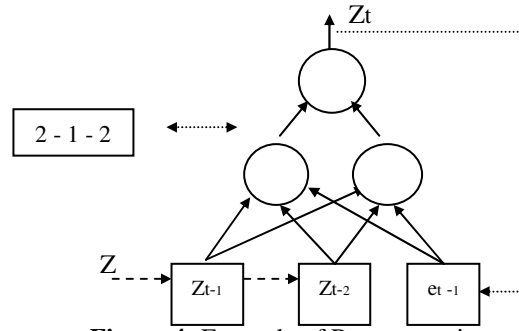


Figure 4: Example of Representation

In the GA module, each network is represented by a vector storing the real values of the parameters to be optimized. As genetic operators, so far we have only implemented a mutation operator which increases or decreases the current values in one unity with the same probability. This operator is the same for the three covered network's parameters.

In the TR module, the networks are trained using the Levenberg-Marquardt algorithm [12] because it is, in general, faster than the Backpropagation [9]. When a time series is received as input, it is equally divided into three sets: training, validation and test. The validation set is used to avoid overfitting on the training set. The Mean Squared Error (MSE) on the validation set was used to evaluate the training process, as well as the GA fitness function.

The similarity measure implemented considers the similarity between the autocorrelations of the series. As said before, it is not straightforward to use the serial correlations in the ANN's identification due to the difficulty in determining their theoretical behavior in ANNs. However, with the help of a case base, we are able to know what ANN was successfully used when the serial autocorrelation presented a similar behavior to the current one.

The case base was initially created with 47 cases. To generate the cases, we chose 47 time series and applied GAs to define the adequate ANN to each series. In those executions, the number of chromosomes per

generation was set to 4 and the number of generations per execution was set to 5. Therefore, for each time series, 20 architectures were defined and trained, and the one with the lowest validation error was returned as the final architecture. The mutation rate was set to 0.4. The time-window length and the context-layer length were initially assigned to values within the interval [0;12], and the number of the hidden nodes was constrained to the interval [1;5].

The entire prototype was implemented in Matlab 5.0. Both the NARX and NARMAX networks, and the Levenberg-Marquardt algorithm were implemented using the Nnsysid (Neural Network System Identification) toolbox [13].

5 Tests and Preliminary Results

To evaluate the performance of our prototype, we choose 3 time series and defined neural networks for each one deploying 2 different techniques: (1) using case-initialized GAs; and (2) randomly initializing the GAs. Each of these options was executed 5 times for each series, and the average MSE of the training, validation and error sets are shown in tables 1, 2 and 3. The GAs' parameters and the exploited search space were the same as the ones used during the creation of the cases (see section 4).

	AVG MSE Training	AVG MSE Validation	AVG MSE Test
Case	126131,15	186623,70	149631,48
Random	104408,96	198346,50	160808,85

Table 1: Average Errors – Time Series 1

	AVG MSE Training	AVG MSE Validation	AVG MSE Test
Case	14102,08	41050,44	85318,92
Random	11508,00	42436,19	96287,59

Table 2: Average Errors – Time Series 2

	AVG MSE Training	AVG MSE Validation	AVG MSE Test
Case	1355,28	1559,11	1359,75
Random	1474,61	1686,90	1411,46

Table 3: Average Errors – Time Series 3

We opted to use the validation error as fitness function because it estimates the generalization performance of a network. For the three analyzed time series, we observed that the use of the case-

initialization showed a gain in the validation errors. The good generalization performance of the networks generated by Case-Initialized GAs was confirmed by the lower errors for the test sets.

These results, however preliminary, encourage us to increase the number of cases in the base. We expect that the choice of architectures from the case base will become better as more cases are inserted.

6 Conclusion e Future Work

In this paper, we approached the problem of neural network design for time series prediction. We proposed a methodology for designing neural networks using the technique of Case-Initialized Genetic Algorithms.

In the initial prototype, the methodology was used to define the time-window, context layer and hidden layer lengths of the NARX e NARMAX networks. To form the initial case base, neural models were defined for 47 time series using random initialized GAs.

Tests were undergone to define the networks for three new time series. The CIGAs were compared to GAs with random initialization and a gain with the case-initialization was observed in the validation and test sets for these series.

The case base currently counts on 47 cases, however it is being continuously augmented. In future work, new results will be presented with the augmented base.

As we have said, the proposed methodology can be adapted to other problems also treated by neural networks, such as classification problems. This is an issue to be faced in the future.

References

- [1] G. E. Box, G. M. Jenkins & G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, third edition (Englewood Cliffs, NJ: Prentice Hall, 1994).
- [2] G. Dorffner, Neural Networks for Time Series Processing, *Neural Network World*, 6(4), 1996, 447-468.
- [3] S. Louis & J. Johnson, Robustness of Case-Initialized Genetic Algorithms, 1999, on-line accessed on July, 11 2001, <http://citeseer.nj.nec.com/92649.html>.
- [4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley, 1989).
- [5] K. Balakrishnan & V. Honavar, Evolutionary Design of Neural Architectures: Preliminary Taxonomy and Guide to Literature, *Technical Report*

CS TR95-01, Department of Computer Science, Iowa State University, 1995.

[6] X. Yao, Evolutionary Artificial Neural Networks, In *Encyclopedia of Computer Science and Technology*, 33, 137-170 (New York, NY: Marcel Dekker Inc., 1995).

[7] J. Sjöberg, H. Hjalmarsson & L. Ljung, Neural Networks in System Identification, 1994, on-line, accessed on July, 11 2001 <http://citeseer.nj.nec.com/sjoberg94neural.html>.

[8] J. Hakkarainen, A. Jumperanen, J. Kyngas & J. Kyyro, An Evolutionary Approach to Neural Network Design Applied to Sunspot Prediction, 1996, on-line, accessed on July, 11 2001, <http://citeseer.nj.nec.com/>.

[9] R. Battiti, First and Second-Order Methods for Learning Between Steepest Descent and Newton's Method, *Neural Computation*, 4, 1992, 141-166.

[10] A. Aadmot & E. Plaza, Case-based reasoning: foundational issues, methodological variations and system approaches, *AI Communications*, 7, 1994, 39-59.

[11] J. Kolodner, *Case-based Reasoning* (San Matteo, CA: Morgan Kaufmann, 1993).

[12] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, *SIAM J. Applied Mathematics*, 11, 1963, 431-441.

[13] M. Norgaard, Neural Network Based System Identification Toolbox Version 1.1 For Use with Matlab, *Technical Report 97-E-851*, Department of Automation, Technical University of Denmark, 1991.