FINANCIAL TIME SERIES FORECASTING USING A HYBRID NEURAL-EVOLUTIVE APPROACH

JUAN J. FLORES\textsuperscript{1}, ROBERTO LOAEZA\textsuperscript{1}, HECTOR RODRIGUEZ\textsuperscript{1}, FEDERICO GONZALEZ\textsuperscript{2}, BEATRIZ FLORES\textsuperscript{3}, ANTONIO TERCEÑO GÓMEZ\textsuperscript{3}

\textsuperscript{1}Division de Estudios de Posgrado, Facultad de Ingeniería Electrica, Universidad Michoacana, Mexico.
\textsuperscript{2}Facultad de Contaduría y Ciencias Administrativas, Universidad Michoacana, Mexico. juanf@umich.mx, roberto.loaeza@gmail.com, hector120713@gmail.com, fsantoyo@umich.mx, bflores@umich.mx
\textsuperscript{3}Facultat de Ciencies Economiques i Empresarials, Universitat Rovira i Virgili- España. antonio.terceno@urv.net

Abstract.
The design of models for time series prediction has found a solid foundation on statistics. Recently, artificial neural networks have been a good choice as approximators to model and forecast time series. Designing a neural network that provides a good approximation is an optimization problem. Given the many parameters to choose from in the design of a neural network, the search space in this design task is enormous. When designing a neural network by hand, scientists can only try a few of them, selecting the best one of the set they tested. In this paper we present a hybrid approach that uses evolutionary computation to produce a complete design of a neural network for modeling and forecasting time series. The resulting models have proven to be better than the ARIMA models produced by a statistical analysis procedure and than hand-made artificial neural networks.

1 INTRODUCTION

The design of models for time series prediction has traditionally been done using statistical methods. In modeling time series, we find the ARIMA (Auto-Regressive Integrated Moving Average), ARMA, and AR, among others [11]. These models are defined in terms of past observations and prediction errors. Statistical techniques like auto-correlation, and partial auto-correlation, help scientists identify which of the past observations and/or errors are significant in the construction of the forecasting models.

In the last decade, artificial neural networks have been used successfully to model and forecast time series. Designing an artificial neural network (ANN) that provides a good approximation is an optimization problem. Given the many parameters to choose from in the design of an ANN, the search space in this design task is enormous. On the other hand, the learning algorithms used to train ANNS are only capable of determining the weights of the synaptic connections, and do not include architectural design issues. So, a scientist in need of an ANN model has to design the network on a trial and error basis. When designing an ANN by hand, scientists can try only a few of them, selecting the best one from the set they tested.

We can approach the optimization task involved in ANN design using evolutionary computation. In this paper we present a hybrid approach that uses evolutionary computation to produce a complete design of an ANN for modeling and forecasting time series. The architecture we use in the forecasting models is a multi-layer perceptron (MLP). We chose to try 3-layer models, which include an input layer, a hidden layer, and an output layer.

After an ANN is designed, it needs to be trained. Training is the process of determining the weights of the synaptic connections for a given architecture (which does not change in the learning process). The most known learning algorithm is back-propagation. Back-propagation takes every example in the
training set, runs the network, and computes the difference between its output and the expected output. The difference is then used to adjust the weights of the network. The process is repeated until convergence, or a maximum number of iterations (epochs) is reached. Unfortunately, back-propagation is a gradient-based optimization algorithm, and as such, opens the possibility of the optimization process to end up in a local maximum.

In this paper we propose to design a forecasting ANN using evolutionary computation in three stages. In the first one, the ANN architecture is designed; the second stage optimizes the weight assignments for the synaptic connections; after a suitable candidate has been determined through the first two stages, the third stage fine tunes the weights of the ANN.

We compared our results with a statistically designed model and a hand-crafted ANN. To compare the forecasting accuracy of the different models, we use the following statistical measures: MSE, MAE and Theil’s U. The best model produced through evolutionary computation has proven to be better than the ARIMA and the hand-made artificial neural network models.

The rest of the paper is organized as follows. Section 2 surveys the state of the art in forecasting with statistical methods and artificial neural networks. Section 3 describes the setting, location, and devices involved in the data acquisition process, used to obtain the wind speed time series used in these experiments. Section 4 proposes the evolutionary computation architecture used in the ANN design. Section 5 discusses the results obtained and compares them with traditional approaches. Finally, Section 6 concludes the work.

2 RELATED WORK

Forecasting techniques assume that the time series, taken from measurements, is the sum of different components and a random error. The goal of most forecasting techniques is to separate and identify those components (trend, cyclical, seasonal, and irregular). Recently, several techniques have been used from the fields of statistics and artificial intelligence [9, 3, 5, 14]. Scientists have even combined them in order to reduce the forecasting error and to produce more accurate predictions [15, 16].

According to the development of studies, it would be useful to conduct an exploration of the entire universe of configurations that form the neural networks and realize if there is an optimal configuration that can reduce the errors found with statistical techniques.

The area of combining Evolutionary Computation and Artificial Neural Networks to produce Neural Systems capable of classifying, predicting, or controlling complex systems has been explore, the proposal presented in this paper makes contributions to the area not present in previous work, therefore advancing human knowledge on the deployment of ANNs. This section contrasts our proposal with related research work, highlighting the differences and the advantage of the proposed methodology, presented here.

Yiau and Liu [13] present a scheme based on evolution programming, emphasizing on evolution ANN’s behaviors. A mixture of other ideas is incorporated in their proposal: mutations are provided by partial training (i.e. a la memetic algorithms) and node splitting. They work, called EPNet evolves architectures and connections weights, while their approach presents a combination of techniques, ours uses pure evolutionary computation.

The work of Abraham [1, 2] presents several differences with respect to our proposal. He uses evolutionary algorithms to determine the network architecture, connection weight, and learning algorithms. Our approach also designs the inputs to the ANN, but does not determine the learning algorithm. That decision relies on the fact that we are training the ANN through evolutionary computation as well. Another difference is that he uses a binary encoding for the weights, while we use real encoding.

Mayor and Schwaiger [8] present a system that evolves ANN in a evolutionary scheme. Low complexity ANNs guides the evolution of ANNs of greater generalization ability. Evolution is achieved by Gas, using error back-propagation to train the networks. The evolutionary processes they
propose consider ANNs of fixed architecture. They also use co-evolution to determine the training data set. The Mackey-Glass benchmark was used to test their results; given that, the benchmark is well known, the inputs to the ANN are fixed.

In summary, our proposal differs from previous works in different aspects. Some of them do not evolve the ANN architecture at all, others evolve it partially. The proposed scheme and representation enable us to design the totality of the ANN architecture. In addition, most of the schemes adopt a hybrid approach, interleaving training (using different learning algorithms) with evolution. Our approach is based on pure evolutionary computation; at the end, though, for the winner ANN, we push it a bit forward using back-propagation [6, 4]. Since we have explored the search space, the winner ANN architecture is expected to reach the local optimum where the GA left it, which most likely will be the global optimum.

3 DATA MEASUREMENT

Experiments were performed with a real time-series formed by a data base taken from the Banco de México (FIRA), measured from the indicator known as “Agregados monetarios y flujo de fondos” of a given activity \( W \). The training set covers the period from January, 1986 to December 1991, while the validation set ranges from January to December, 1992.

Given that the behavior of the measured variable corresponds to a time series (stochastic process), where the uncertainty level caused by white noise, for this kind of processes, it is necessary to find the right forms (models) to perform forecasting and financial decision taking in more efficient and accurate ways [10].

Given the above facts, we propose the hybrid usage of evolutionary computation and artificial neural networks; we consider this combination an efficient, accurate, and powerful scheme applicable to the solution of this kind of problems.

![Financial Time Series](image)

**Fig. 1.** Financial times series taken from Banco de Mexico.

4 EVOLVING ANNS

Given a time series, we need to provide a neural model capable of producing an acceptable prediction of that time series.

In order to define the term acceptable model, i.e. the fitness measure of a given model, there exist statistical measures that allow us to compare two time series, for instance, the Mean Squared Error (MSE), the Mean of the Absolute Value of the Errors (MAE), etc. [11]. Among these measures, we find Theil’s U, which, for an acceptable model, must return a value in the interval \([0.5, 1]\).
An ARIMA model \cite{11} is a statistical model that allows us to model time series, and to predict their behavior. These models have the following form:

\begin{equation}
    y_{t+1} = \sum_{k=0}^{w} a_k y_{t-k} + b_k e_{t-k} + \epsilon_t.
\end{equation}

\begin{equation}
    e_t = y_t - \hat{y}_t.
\end{equation}

Where \( y_t \) represents the measurement at time \( t \) in the time series and \( \hat{y}_t \) is the forecasting produced by the ARIMA model; \( e_t \) represents the effects of random factors; \( w \) is the window width. The window represents how far behind in time we consider measurements as probably important inputs for the ARIMA model. Outside of the window, observations are not taken into account. Using statistical procedures, the numerical value of the coefficients \( a_k, b_k \) and \( \epsilon \) are determined.

In the approach presented in this paper, we are using an Auto-Regressive model (AR), which is a reduced version of ARIMA. The AR model does not consider past forecasting errors as forecasting variables. AR has the following form:

\begin{equation}
    y_{t+1} = \sum_{k=0}^{w} a_k y_{t-k} + \epsilon_t.
\end{equation}

The ANN architecture used for prediction is the Multi-Layer Perceptron (MLP). A MLP, as a universal approximator \cite{6}, can learn any function, given it has enough neurons in the hidden layer. That fact allows the network to capture the different forms of the function to be modeled.

Given an AR model, we can design a MLP capable of reproducing the time series at least as well as the ARIMA model itself. The output of the MLP is always a single neuron, representing the forecasting output, \( \hat{y}_t \). Once the inputs to the MLP are specified, the design process reduces to determine the number of neurons in the hidden layer.

Notice that the learning models for ANNs are designed to determine the weights of the synaptic connections. Those learning models do not consider the design of the network architecture. One way to design the neural network is to perform a statistical analysis to determine what variables are important in the forecasting. Those variables will be considered as the inputs to the ANN.

In this work we intend to design the MLP completely, without the need of any statistical analysis. That is, we design the number of input neurons and what they represent, and the number of hidden neurons (the output neuron will always be the same). The design process includes the determination of the weights of the synaptic connections, without the need of a learning algorithm (v.g. back-propagation). The reason to avoid those learning methods is that since they are gradient-based, they are likely to stop at a local optimum. This fact may make a MLP behave badly, even with an adequate architecture.

The proposal is to use Evolutionary Computation to perform the complete design of the ANN used in forecasting. The scheme involves two nested evolutionary processes followed by a third one. The first one designs the network architecture, while the second (inner) one, once determined the architecture, determines the weights of the synaptic connections. A last evolutionary process refines the weights for the winner network of the previous two processes. The proposed architecture of the hybrid ANN-Evolutionary scheme is shown in Fig. 2.
This evolutionary scheme uses two types of chromosomes. The first one, for the outer evolutionary process, contains a bit vector (Vars), whose size is the window size, followed by an integer (NH). A value of 1 in position k of the bit vector indicates that variable $x_k$ appears as an input variable in the MLP being designed; a 0 indicates that variable is not taken into account in the model. NH indicates the number of neurons included in the MLP’s hidden layer. Fig. 3 shows the structure of this chromosome.

For each individual in the outer evolutionary process, we proceed to the inner evolutionary process. The chromosome of this second process contains a vector of real numbers with NC elements. Let us say NV is the number of 1s appearing in Vars. NC is the number of synaptic connections in the neural model, where $NC = (NV + 1) \times NH$. Fig. 4 shows the structure of the second chromosome.

![Fig. 3. Structure of the chromosome of the outer evolutionary process](image1)

![Fig. 4. Structure of the chromosome of the inner evolutionary process](image2)

Fig. 5 shows an example of an individual belonging to the inner evolutionary process. The chromosome shows the proposed inputs ($x_5$, $x_8$, and $x_{10}$); the network contains 5 neurons in the hidden layer; the remaining real values are the weights of the synaptic connections.

![Fig. 5. ANN Model](image3)
We provide genetic operators for mutation and crossover for both evolutionary processes. Those genetic operators allow populations to evolve and produce optimized solutions.

Once the first two evolutionary processes are performed, we have the best of the inspected models. At that time a third evolutionary process is performed. This third process is similar to the second one, but we allow a larger population, in order to allow the synaptic weights to be refined.

The search space we are exploring and optimizing in the solution of this problem is huge. That made us play with the different parameters in the evolutionary processes and refine them, to be able to explore the search space more efficiently. For instance, the number of ANNs to be explored is very large, and for each designed architecture, the possibilities for the synaptic weights are just too many. Given that, we decided to let the process explore a good number of ANN designs, and for each design try not too many combinations of weights. After that, the winner ANN is further refined. At that moment (the third evolutionary process), we are exploring a single architecture and give it a larger population size, with more generations, and also, a larger chance of mutations.

5 RESULTS

The experiments performed were divided in ANN Architecture Design, ANN Weight Design, and ANN Weight Refinement processes. The ANN Architecture Design process evaluates about 3,250 architectures. Each architecture was evaluated with about 1,640 different combination of weights (ANN Weight Design process). About 5,330,000 evaluations in total.

The ANN Weight Refinement process uses the best Architecture obtained and continues evolving the best combinations of weights. About 80,000 different combination of weights were evaluated.

All experiments were performed using Genetic Algorithms (GA) and Evolutionary Strategies (ES) with Evolvica [7].

The winner ANN was produced using ES, it took about 109 hours and its characteristics are:

- Window width: 18
- Number of inputs: 9
- Number of neurons in the hidden layer: 31
- Number of outputs: 1
- The output was defined as a function of $y_{t-1}$, $y_{t-2}$, $y_{t-6}$, $y_{t-11}$, $y_{t-12}$, $y_{t-13}$, $y_{t-14}$, $y_{t-15}$, and $y_{t-17}$

![Fig. 7. Predicted and observed measurements for the validation set](image)
**Table 1** shows the results obtained for the statistical measures with a naïve model and our approach. From

![Graph of Validation](image1)

**Fig. 7.** Predicted and observed measurements for the validation set

**Table 1** it is clear that the Hybrid model has acceptable statistical errors, lower than those produced with the naïve model. Fig. 6 shows the comparison between the observed data and the predicted ones. In Fig. 6 and Fig. 7 the continuous lines are the observed data and the discontinuous lines are the predicted data.

![Graph of Training](image2)

**Fig. 6.** Predicted and observed measurements for the training set

![Graph of Validation](image3)

**Fig. 7.** Predicted and observed measurements for the validation set

**Table 1.** Fitness measure for the different forecasting models.

<table>
<thead>
<tr>
<th>Method/Accuracy</th>
<th>MAE</th>
<th>MSE</th>
<th>Theil's U</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6. CONCLUSIONS

We presented a hybrid neural-evolutionary methodology to forecast time-series. The methodology is hybrid because an evolutionary computation-based optimization process is used to produce a complete design of a neural network. The produced neural network, as a model, is then used to forecast the time-series.

Experiments were performed with a real time-series formed by a data base taken from the Banco de México (FIRA), measured from the indicator known as “Agregados monetarios y flujo de fondos”. The forecasts produced with the proposed methodology exhibit a better behavior than the previous ones, produced through statistical methods and hand-crafted ANNs.

One of the advantages of the proposed scheme is that the design and training of the ANNs has been fully automated. This implies that the model identification does not require any human intervention. The model identification process involves data manipulation and a highly experienced statistician to do the work. This fact pushes the state of the art in automating the process of producing forecasting models.

Compared to previous work, our approach is purely evolutionary, while others use mixed, mainly combined with back-propagation, which is known to get stuck in local optima. On the direction of model production, the evolutionary process automates the identification of input variables, allowing the user to avoid data pre-treatment and statistical analysis.

The system is fully implemented in Mathematica [12], using Evolvica [7], a Mathematica package developed to perform evolutionary computation.

REFERENCES


Zhang, G.P.: Time series forecasting using a hybrid ARIMA and neural network model.