Time Series Forecasting in a Distributed Environment

Paulo Cortez, José Machado and José Neves
Universidade do Minho, Departamento de Informática
Largo do Paço
4719 Braga Codex, PORTUGAL
Voice : 351 53 604466
Fax: 351 53 604471
E-mail: {paulo.jmac,jneves}@di-ia.uminho.pt

Abstract
The area of Time Series Forecasting (forecasting observations ordered in time) is object of attention, in particular due to its role in the process of decision making. Artificial Neural Networks (ANNs) have been successfully used in pattern recognition, even in Time Series with a high degree of noise, but fail in the attempt to develop a fine architecture on their own. Genetic Algorithms (GAs), on the other hand, proved to be one of the most promising arenas to elaborate on problem solving optimization strategies. Amalgamating both paradigms, one makes the denizens of a new computational world: ANNs and GAs. ANNs will assimilate the pattern series and better ANNs architectures will be attained by evolution.

Keywords: Neural Networks, Genetic Algorithms, Time Series Forecasting.

1 Introduction

Time Series Forecasting (TSF) is of crucial importance in almost any kind of organization. Short term predictions, one or two predictions ahead, are used for current management decisions (eg. dealing with stocks). Middle/long term forecasts are used for planning (eg. elaborating budgets). Indeed, optimization algorithms based on Natural Sciences have been not only object of attention, but exhibited a considerable spread across different disciplines and application domains. In particular, Artificial Neural Networks (ANNs) and Genetic Algorithms (GAs), according to recent developments, either in Neuroscience Theory or in the mechanics of Biological Evolution, have gained the reputation of being among the most popular and important ones.

ANNs have the ability to learn patterns and to respond to new situations based on past experiences, especially where a strong noise component or incomplete data is present [5][12]. This ability is well recognized in the areas where ANNs systems succeeded, namely in the Stock Markets, the Biological Engineering or the Computer Vision ones. GAs, on the other hand, are used to evolve behaved strategies. This is much like the Prisoner’s Dilemma (PD) game, a type of interaction that has been widely studied in Game Theory, Computer Science, Philosophy, Biology, The Political and Social Sciences, just to refer a few [9]. What makes this interesting is that the PD game models situations in which both parties can benefit from playing cooperatively.
Table 1: Activation functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Function $f(x)$</th>
<th>Codomain</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>$x$</td>
<td>$]-\infty, +\infty[\right$</td>
</tr>
<tr>
<td>sigmoid</td>
<td>$\frac{1}{1+e^{-x}}$</td>
<td>$[0,1]$</td>
</tr>
<tr>
<td>sigmoid1</td>
<td>$\frac{1}{1+e^{-x}} - 1$</td>
<td>$[-1,1]$</td>
</tr>
<tr>
<td>sigmoid2</td>
<td>$\frac{x}{1+</td>
<td>x</td>
</tr>
<tr>
<td>tanh</td>
<td>$tanh(x)$</td>
<td>$[-1,1]$</td>
</tr>
</tbody>
</table>

ANNs will learn to forecast based on previous data, trained by backpropagation learning, and better ANNs will be obtained by evolution. The idea is to bound GAs and ANNs rationality. The novelty of the approach is in its straightforwardness, being the bound on rationality measured in terms of computation time (it is assumed that the GAs and ANNs need time for computations, and that the game is repeated for a finite length of time). These assumptions not only balance the game (i.e. create cooperative equilibria), but also enables an ANN (or GA) to control its GA (or ANN) strategy and vice-versa [7] (Figure 1), giving on to a new computational world: GAs and ANNs or the GANNs systems.

![Error vs Computational Costs](image)

Figure 1: Search Speed or a Convergence of Strategies

## 2 The Neural Network Topology

Fully connected feedforward ANNs are used, without shortcut connections but with bias, with one hidden layer. The structure of layers will be represented in the form of $L_u - L_h - L_o$ for an ANN with $L_u$ input nodes, $L_h$ hidden nodes and $L_o$ output nodes. The initial weighting factors (i.e. the adaptive coefficients within the network that determine the intensity of the input signal) are randomize, within the range $\left[\frac{\pm 1}{\sqrt{z}}\right]$ for a node with $z$ inputs [5]. As activation functions are used those of Table 1: linear, sigmoid, sigmoid1, sigmoid2 and tanh. The number of output nodes, $L_o$, is set to one. The main goal is to set the forecasted value as a function of $n$ previous ones, in an ANN with $n$ inputs. In this way it is possible to make short or long term predictions. The training cases are built upon TS. Actually, given the TS as $x_1, \ldots, x_p$ and an ANN with $k$ inputs, one has:

\[
x_1, x_2, \ldots, x_k \rightarrow x_{k+1} \\
x_2, x_3, \ldots, x_{k+1} \rightarrow x_{k+2} \\
\vdots \rightarrow \vdots \\
x_{p-k}, \ldots, x_{p-1} \rightarrow x_p
\]
To avoid problems of overfitting, an early stopping procedure was implemented, implying a splitting of the data case sets, namely: a training set (to assimilate the input patterns) and a validation set (to test the ANNs generalization capability).

Being \( p \) the time period, \( x_1, x_2, \ldots, x_p \) the time series, output the ANN output function, and \( f_{i,j} \) the forecast in the \( j \) period to \( i \) periods ahead of \( j \), \( n \) the number of periods to forecast, it follows:

\[
\begin{align*}
    f_{1,p} &= \text{output}(x_p, \ldots, x_{p}) \\
    f_{2,p} &= \text{output}(x_{p-k+1}, \ldots, x_p, f_{1,p}) \\
    f_{3,p} &= \text{output}(x_{p-k+2}, \ldots, x_p, f_{1,p}, f_{2,p}) \\
    &\vdots \\
    f_{n,p} &= \text{output}(f_{n-k,p}, \ldots, f_{n-1,p})
\end{align*}
\]

Usually the best ANNs for short term forecasts, one period ahead of \( p \), \( (f_{1,p}) \), are different from the best ANNs for long term forecasts, to \( n \) periods ahead of \( p \), \( (f_{1,p}, \ldots, f_{n,p}) \), being the last ones more sensitive to overfitting. On the other hand, the user normally controls the number of the forecasts ahead that he/she wants. Thus the necessity to develop a special validation test, the Specific Mean Square Error (SMSE), that considers the number of the forecasts ahead. The idea was to built a test where the MSE were a particular case. Being \( q \) the last period in the series, \( r \) the number of cases in the validation set, and \( n \) the number of periods to forecast, it follows:

\[
\begin{align*}
    \text{MSE}_{n,j} &= \frac{(f_{i,j} - x_{i+j})^2 + (f_{i,j} - x_{i+2j})^2 + \ldots + (f_{i,j} - x_{i+nj})^2}{n} \\
    \text{SMSE}_n &= \frac{\sum_{j=1}^{n} \text{MSE}_{n,j}}{r}
\end{align*}
\]

where \( \text{MSE}_{n,j} \) gives the error for a long term forecast to \( n \) periods ahead of \( j \).

3 Combining Genetic Algorithms with Neural Networks

GANNs systems optimize ANNs in two ways: evolution by the GA and learning by backpropagation. In one’s system, the ANNs learning guides the evolutionary search [6]. If an encoded ANN is near to the optimum, then by backpropagation learning will reach that optimum, resulting in a good fitness value [8]. Solutions (chromosomes) near the optimum will share patterns of bits, so the GA will be able to explore them by hyperplane sampling.

An empirical fact related to GANNs systems is that they are not appealing to large networks. However, this will not constitute a major drawback once there is some evidence that TSF are modelled by small ANNs [4].

For reasons of optimum solution versus computational demand it was therefore decided to use a population of \( m \) individuals (being in our case \( m \) set to 20), to make chromosome selection by the classical fitness-based roulette-wheel, to use an one point crossover with a crossover rate of 1 and a mutation rate of 0.02. For the initial population the system uses a random seed of 12345. As the fitness function the system uses the SMSE; i.e., \( \text{fitness} = \frac{1}{\text{SMSE}} \).

The chromosome selection is done by a fitness-based process and purification. The chromosome with the lowest SMSE will automatically survive to the next generation. This purification process accelerates the search and warranties that if the optimum ANN is reached, then it will never be lost. The genetic algorithm is stopped after \( t \) generations (in our case \( t \) was set to 40).

After empirical tests it became evident that there are six factors that do affect a good forecast: the number of input nodes, the learning rate, the activation function, the number of hidden nodes, the normalization constant and the random weights initialization seed. The first three have a
much stronger influence than the others, and were put at the left of the chromosome (Figure 2), in a direct encoding using base 2 gray codes (i.e. a binary code, in which any adjacent pair of numbers, in counting order, differ in their digits at one position only, the absolute difference being the value 1).

<table>
<thead>
<tr>
<th>Ninputs</th>
<th>Function</th>
<th>Rate</th>
<th>Nhiddens</th>
</tr>
</thead>
</table>

Figure 2: Neural Network Encoding

The number of input nodes (and the hidden nodes too) were set within the range [3,14] allowing for a binary encoding of 4 bits. There is some evidence that this is the correct making, with a higher range strongly affecting the searching time [4]. The learning rate was encoded in 3 bits using discrete values in the range [0.1, 0.8] with a rank of 0.1. For the activation functions (Table 1) it were used 3 bits, allowing for 3 do not care values (i.e. upon creation new bits must have their values bounded).

Better forecasts are achieved with grades that are near to the forecasting values; i.e. series with a growing trend would lead the GA to find a normalization constant very close to the data in the validation set. That boundary would lead to forecasts below the series values, limiting the future forecasts by the ANN (Figure 3, where q is the last period in the series and $k_{GA}$ is the normalization constant given by the GA). To avoid this situation one has to prompt the user to choose the normalization constant (the system gives the highest value from the series as an indication).

4 The System Architecture

The basis for the system’s architecture is provide by the Linda tuple space model for process communication [11][2]. The overall layer’s structure of the system is shown in Figure 4, which has been implemented on a TCP/IP network with Sun4 UNIX workstations, using SICStus Prolog [1]. A message-passing perspective to parallel computation coupled with the Linda model was used to devise the distributed system. The execution shell provides for user interface, program loading and user goal executions.

The Prolog language glues the ANNs and GAs. Data exchanges are done by data files (Figure 5). The GA was written in the C language. Since the GA refers repeatedly to the same chromosome, a cache was implemented to speed up the process (which was implemented in
Prolog). A true Prolog-C interplay was necessary, with recursive calls from one another. The SICStus Prolog language provided such a mechanisms through dynamic linking.

The system nodes, processes and message flow are shown in Figure 5. Arrows show the flow of messages between processes. Each system node has a node identification and is considered to be autonomous with local control. Local control is effected by the server process that manages control messages and Tuple Space communication.

Figure 4: Overall System Layer Structure

Figure 5: The Distributed System Architecture

The master process guides the ANNs training and collects the results. Each processor receives the system configuration, data series and the ANN parameters. The cases are created and the training begins. Once a SMSE result is obtained, it is sent to the master process and the associated process is halted.

The X-Windows interface (Figure 6), was built using the object oriented facilities of the SICStus Prolog gnu lib library. A flavor of the code is given below:

```prolog
action(forecast) :- Built <= button("Build Model",built),
    Forecast <= button("Forecast",forecast),
    Exit <= button("Exit",exit),
    W <= window("Forecast",
                hbox([space,
                      vbox([ space,
                            hbox([space,Built,space]])]),
```
space,
hbox([space,Forecast,space]),
space,
hbox([space,Exit,space]),
space
],

space])

W => open(460,244),
wait_for_action(W).

Figure 6: The X-Windows Graphical Interface

where "," names conjunction, ":=" is to be read as "if" and "<=" names "<=".

The user is presented with the options: Series, Forecast, Setup and Exit. With Series one
is allowed to perform operations like inserting, loading, saving, listing, etc. With Forecast one
is able to built the forecasting model; i.e., to run the GA and get the optimum ANN, or to
forecast ahead. Setup caters for default system configuration. Exit stands for itself.

When building a model the system checks for series randomness. This test is based in the
autocorrelation coefficient, which gives a measure of the correlation between a series and itself,
lagged \( u \) periods:

\[ r_k = \frac{\sum_{t=1}^{n-u} (X_t - \bar{X})(X_{t+u} - \bar{X})}{\sum_{t=1}^{n} (X_t - \bar{X})} \]

where \( X_t \) stands for the series value for the period \( t \), and \( n \) for the series length. If a series
is random then the correlations for the series values should theoretically be zero, although in
practice one may consider a series to be random if, for all lags, the correlations are within the
range [3]:

\[-1.96 * \frac{1}{\sqrt{n}} \leq r_u \leq 1.96 * \frac{1}{\sqrt{n}}\]

When the series is random the only possible forecast is the series average.
Table 2: GANN results for $f_{1,t}$ forecasting

<table>
<thead>
<tr>
<th>Series</th>
<th>Size</th>
<th>$K$</th>
<th>$T$</th>
<th>$V$</th>
<th>$N$</th>
<th>Topology</th>
<th>$F$</th>
<th>$R$</th>
<th>$G$</th>
<th>$MSE$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>154</td>
<td>81</td>
<td>9</td>
<td>10</td>
<td>$8 - 8 - 1$</td>
<td>sigmoid</td>
<td>0.7</td>
<td>24</td>
<td>138.5</td>
</tr>
<tr>
<td>2</td>
<td>130</td>
<td>5050</td>
<td>106</td>
<td>12</td>
<td>12</td>
<td>$12 - 12 - 1$</td>
<td>sigmoid</td>
<td>0.1</td>
<td>20</td>
<td>1656018</td>
</tr>
<tr>
<td>3</td>
<td>144</td>
<td>620</td>
<td>119</td>
<td>13</td>
<td>12</td>
<td>$9 - 9 - 1$</td>
<td>sigmoid</td>
<td>0.7</td>
<td>14</td>
<td>3037</td>
</tr>
<tr>
<td>4</td>
<td>369</td>
<td>603</td>
<td>339</td>
<td>20</td>
<td>10</td>
<td>$6 - 5 - 1$</td>
<td>sigmoid</td>
<td>0.8</td>
<td>7</td>
<td>62.4</td>
</tr>
</tbody>
</table>

Table 3: $f_{1,t}$ GANN forecast versus the series 1 values

<table>
<thead>
<tr>
<th>Period</th>
<th>91</th>
<th>92</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
<th>97</th>
<th>98</th>
<th>99</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series 1</td>
<td>96</td>
<td>77</td>
<td>59</td>
<td>44</td>
<td>47</td>
<td>30</td>
<td>16</td>
<td>7</td>
<td>37</td>
<td>74</td>
</tr>
<tr>
<td>Forecast</td>
<td>111</td>
<td>85</td>
<td>67</td>
<td>53</td>
<td>40</td>
<td>45</td>
<td>25</td>
<td>15</td>
<td>14</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 2 shows the system’s results for short term forecasting. The series parameters are $K$ for the normalization constant, $T$ for the number of series values in the training set, $V$ for the number of series data in the validation set, and $N$ for the number of forecasts. The optimum ANN parameters are: $F$ for function and $R$ for the learning rate. $G$ stands for the GAs iteration (generation) where the optimum ANN is attained.

Let us consider the forecasts for series 1 given in Table 3. Table 4 shows a comparison of the system’s results, based on the previous forecasts, with the Holt-Winters [10] and ARIMA [3] ones. Analyzing Table 4 for short term forecasting, it follows:

- the GANN forecast for series 1 outperforms that of the ARIMA one. This is an encouraging fact since this is one of the series that is used to support the ARIMA claims;

Table 4: Comparing GANN’s MSE results with conventional methods for $f_{1,t}$ forecasting

<table>
<thead>
<tr>
<th>Series</th>
<th>GANN</th>
<th>ARIMA</th>
<th>Holt-Winters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>138.5</td>
<td>163.8/256</td>
<td>–</td>
</tr>
<tr>
<td>2</td>
<td>1656018</td>
<td>–</td>
<td>1348153</td>
</tr>
<tr>
<td>3</td>
<td>3037</td>
<td>–</td>
<td>270</td>
</tr>
<tr>
<td>4</td>
<td>62.4</td>
<td>67.2</td>
<td>103.6</td>
</tr>
</tbody>
</table>

- series 4 presents a strong trend component. Although in this case the ARIMA model will outperform the Holt-Winters one, the GANN system will outperform both;

- the Holt-Winters model is attractive for series 3, that presents seasonal and trend components. The GANN forecast is outperformed once this is a well behaved series;

- series 2 presents also seasonal and trend components but its variance is not constant, thus increasing non-linearity. The GANN system is outperformed by the Holt-Winters one, although both methods give rise to huge errors. The restrictions imposed on the GANN system are probably the main reason for that behavior.

For long term forecasts, optimum ANNs should be different from those for short term ones. This is evident in Table 5. In fact, the ANNs for long term forecasts are slightly different (this
Table 5: Comparing the $SMSE_1$ (or $MSE$) validation test with the $SMSE_n$ one for long term forecasts

<table>
<thead>
<tr>
<th>Series</th>
<th>$n$</th>
<th>Topology</th>
<th>$F$</th>
<th>$R$</th>
<th>$G$</th>
<th>$p$</th>
<th>$MSE_{10,p}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>8–8–1</td>
<td>0.7</td>
<td>24</td>
<td>90</td>
<td>286</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8–10–1</td>
<td>0.6</td>
<td>6</td>
<td>90</td>
<td>235.9</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>6–5–1</td>
<td>0.8</td>
<td>7</td>
<td>359</td>
<td>133.7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>6–14–1</td>
<td>0.8</td>
<td>9</td>
<td>359</td>
<td>105.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: $MSE_{10,p}$ for each forecasting method

<table>
<thead>
<tr>
<th>Series</th>
<th>GANN</th>
<th>ARIMA</th>
<th>Holt-Winters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>235.9</td>
<td>262.4/278.9</td>
<td>–</td>
</tr>
<tr>
<td>4</td>
<td>105.5</td>
<td>–</td>
<td>377.9</td>
</tr>
</tbody>
</table>

table also shows the validity of the SMSE test). Note that results from series 2 and 3 do not appear in the table. The main reason is that short term system errors are higher than the long term ones obtained with conventional methods. Since short term errors are smaller than the long ones, there is no need to waste computational power.

When comparing with conventional methods, for long term forecasts, the GANN system results are even better (Table 6).

5 Conclusions and Future Work

Cooperative information sharing was a desired, if not a required behavior, in our playing ground. Cooperative equilibria is an inherent characteristic of the model’s soul. Indeed, the common knowledge of the limited computational power available, enables one of the parts (GAs or ANNs) control its counterpart (ANNs or GA) responses. The GANN systems as they are, turn out to be very attractive for TS forecasting with a high non-linear component, especially for long term forecasts. However, some drawbacks are still present, where the high computational cost is not the only one. Indeed, in future work it is intended:

- to use more elaborated training algorithms such as the Quick-Propagation or the R-Prop ones [12][13];
- to experiment on different neural network architectures (eg. use of shortcut connections);
- to turn the GA more efficient by using new parameters or operators;
- to avoid the need of using normalization by some kind of data filtering.

Aknowledgements

The authors wish to thank JNICT for financial support, given under the grant PCSH /C /CJU /924 /95.
References


