Evolving modular networks with genetic algorithms: application to nonlinear time series

A.S. Cofino,1 J.M. Gutierrez1 and M.L. Ivanissevich2

(1) Department of Applied Mathematics and Computational Sciences, University of Cantabria, Spain
(2) Department of Mathematics and Informatics, Universidad Nacional de la Patagonia Austral, Argentina

Abstract: A key problem of modular neural networks is finding the optimal aggregation of the different subtasks (or modules) of the problem at hand. Functional networks provide a partial solution to this problem, since the inter-module topology is obtained from domain knowledge (functional relationships and symmetries). However, the learning process may be too restrictive in some situations, since the resulting modules (functional units) are assumed to be linear combinations of selected families of functions. In this paper, we present a non-parametric learning approach for functional networks using feedforward neural networks for approximating the functional modules of the resulting architecture; we also introduce a genetic algorithm for finding the optimal intra-module topology (the appropriate balance of neurons for the different modules according to the complexity of their respective tasks). Some benchmark examples from nonlinear time-series prediction are used to illustrate the performance of the algorithm for finding optimal modular network architectures for specific problems.

Keywords: neural networks, functional networks, genetic algorithms, time series, chaos

1. Introduction

During the last two decades feedforward neural networks (FFNNs) have become a very popular modelling and forecasting technique (Hertz et al., 1991) and have been successfully applied to many real-world problems for approximating arbitrary input–output mappings (universal approximation property (Cybenko, 1989)). The key disadvantage of these networks is their rigid structure of fully connected layers with many degrees of freedom that may overfit the data, train slowly or converge to local minima.

In recent years, several attempts for obtaining flexible models have been developed using the idea of modularity (hybrid neural systems (Wermter & Sun, 2000), modular neural networks (Happel & Murre, 1994), mixtures of experts (Jacobs et al., 1991) etc.). The concept of modularity is linked to the notion of local computation: ‘each module is an independent system and interacts with others within a whole architecture in order to perform a given task’. Thus, modularity tends to create some structure within the topology to specialize the performance of each module. In the case of modular neural networks these ideas lead to sparse networks beyond the fully connected topology, thus requiring a smaller number of weights. Nowadays, modular neural networks are increasingly popular models for dealing with complex problems that can be decomposed into a number of subtasks.

The local structure and the complexity reduction sustained by modularity have been shown to overcome some of the problems of fully connected FFNNs (Boers et al., 1993). However, in order to have meaningful and efficient models, each module has to perform an interpretable and relevant function according to the mathematical or physical properties of the system (domain knowledge). Unfortunately, it is unclear how best to design such a modular topology based either on the data or on some available domain knowledge. For example, given the trivial modular network shown in Figure 1(a) (an FFNN consisting of four modules: an input layer, two hidden layers and an output layer), several nontrivial modular networks can be easily obtained by splitting up some layers into sub-layers, thereby reducing the number of weights (one such modular network is shown in Figure 1(b)). However, there is no general procedure to design an optimal modular structure for a given problem.

This problem was partially addressed with the development of functional networks, which combine domain qualitative knowledge and data of the problem at hand, providing optimal, simple and soundness functional structures (see Castillo (1998), Castillo et al. (1999) and Castillo and Gutierrez (2000) for an introduction to functional networks). The main advantage of functional networks is that the network topology is derived using...
available properties and information of the problem (domain knowledge); thus, the resulting network model is not a general purpose blackbox, but is specially designed for the problem at hand. On the other hand, the main shortcoming of functional networks is the parametric procedure used to fit each of the resulting functional units, which requires additional information to supply convenient functional families. Unfortunately, in most real-world applications we lack such information and a non-parametric methodology is usually required for determining the functional units from data.

In recent work, Cofiño and Gutierrez (2001) introduced modular functional–neural networks (MFNNs), a hybrid technique which combines functional and neural networks to avoid the above shortcomings. Functional networks are used to infer the optimal functional topology for a given problem, whereas neural networks are used for training the resulting functional modules from data. The resulting modular neural networks were shown to outperform standard functional and neural networks in some benchmark problems of nonlinear time-series prediction. Unfortunately, not every problem can be simplified using functional networks and therefore each particular case needs to be analyzed separately. For instance, as we shall see later, the MFNN shown in Figure 1(c) is the optimal modular structure for the class of associative operators; this network was obtained by applying the MFNN methodology and contains four functional modules (denoted by the four boxes in the figure); one of them is a known function (the additive operator) and the three remaining modules are approximated using FFNNs with a predetermined topology (as in standard neural networks, a trial and error process is required in order to adjust the number of hidden neurons for the problem at hand).

The complexity of each of the functional modules depends on each specific problem (each particular associative operator) and therefore finding the optimal balance of hidden neurons among the different modules (the intra-module structure) is a hard data-driven optimization problem. Genetic algorithms have been widely applied as optimization techniques for complex objective functions, including the automatic generation of neural network architectures (Vonk et al., 1997) and training the resulting models (van Rooij & Johnson, 1996). In this paper, we present a genetic algorithm for searching the optimal neuronal configuration of the functional units in an

Figure 1: (a) A fully connected feedforward 2:4:4:1 network; (b) a particular modular neural network; (c) optimal MFNN for associative operators.

Expert Systems, September 2004, Vol. 21, No. 4 ________________________________________________________________ 209
MFNN in order to best approximate a given problem with a meaningful and simple model. In this case, the genetic algorithm only solves a combinatorial problem, since the modular topology is given by the functional network. To this aim, a fitness function is defined to keep the model as simple as possible, rewarding the quality of the fit to data and penalizing the network complexity. Optimal models are found as a compromise between the two factors.

The performance of this hybrid genetic algorithm is analyzed considering some benchmark examples from time-series prediction. Optimal modular structures are obtained after a few cycles of the genetic algorithm, providing important information about the functional form of the different problems analyzed.

In Section 2 we briefly introduce the functional network paradigm and describe hybrid MFNNs. In Section 3 we describe the application of genetic algorithms to find optimal networks. In Section 4 the resulting models are applied to a nontrivial problem in nonlinear time-series prediction, and some comparative results are presented to illustrate this new technique. Finally, conclusions and some further remarks are given in Section 5.

2. MFNNs

In this section we briefly describe two interesting types of MFNNs considered in this paper to illustrate the proposed methodology: the associative and the separable models (see Castillo et al. (1999) for a complete description of these models).

2.1. The associative MFNN

The associative functional network is the optimal functional structure for the so-called general associativity models, which generalize the class of models that combine the separate contribution of independent variables. The network shown in Figure 2(a) is the graphical representation of the functional equation which characterizes these models:

\[
F_4(x, y) = F_3(F_1(x, z), F_2(y, z))
\]

Apart from the input, output and hidden layers of processing units, the functional network also includes a layer of intermediate units, which do not perform any computation but only store intermediate information and may force the outputs of different processing units to be equal. For instance, in Figure 2(a) the output unit \( u \) is connected to the processing units \( F_1 \) and \( F_3 \), so their outputs must be coincident, as shown in (1).

The above functional equation imposes some constraints in the neuron functions \( F_1, \ldots, F_4 \), which may lead to a simplification of the functional structure (see Aczél (1966) for a detailed description of this process). For this problem, the simplest functional expression resulting from solving the functional equation is

\[
u = F_4(x, y) = F_3(F_1(x, z), F_2(y, z)) = f_3^{-1}[f_1(x) + f_2(y)]
\]

The above equation defines the functional network in Figure 2(b). Note how the functional complexity is greatly reduced, since now all the unknown functions are univariate. The beauty of the functional network method is that the topology is not arbitrarily chosen as a blackbox but is the result of the available qualitative knowledge of the problem exploited in the framework of functional equations.

The problem of teaching the above functional network now reduces to estimating the neuron functions \( f_1, f_2 \) and \( f_3 \) from the available data \( (x_i, y_i, u_i), i = 1, \ldots, n \). If we use fully connected FFNNs to learn the above functions, then we obtain the particular type of modular neural network shown in Figure 3. Therefore, the parameters can be estimated from data using, for example, modular backpropagation (Ballard, 1987).

2.2. The separable MFNN

Another interesting family of functional networks with many applications are the so-called separable functional networks. These models arise in several practical problems in civil engineering, computer-aided design etc., where we
require the output \( z = z(x, y) \) to satisfy the following two conditions.

1. For any value of \( y \), say \( y_0 \), the output must be a linear combination of a given set of functions \( F = \{ f_1(x), \ldots, f_n(x) \} \) with coefficients \( G = \{ g_1(y_0), \ldots, g_n(y_0) \} \).
2. For any value of \( x \), say \( x_0 \), the output must be a linear combination of a given set of functions \( K = \{ k_1(y), \ldots, k_m(y) \} \) with coefficients \( H = \{ h_1(x_0), \ldots, h_m(x_0) \} \).

These properties lead to the functional equation

\[
z = z(x, y) = \sum_{i=1}^{n} g_i(y) f_i(x) = \sum_{j=1}^{m} h_j(x) k_j(y)
\]  

(3)

which generalizes the class of models which combine the separate contribution of each of the independent variables \( z = z(x, y) = f(x) g(y) \).

Thus, equation (3) represents the coincidence of both advantageous representations, and allows us to determine the form of the coefficient functions \( G \) and \( H \) when the sets \( F \) and \( K \) are given. The functional equation (3) can be simplified leading to the final topology shown in Figure 4(a).

\[
z = z(x, y) = \sum_{i=1}^{k} c_i f_i(x) g_i(y)
\]

(4)

where \( c_i \) are constants (parameters of the model). Figure 4(b) shows the corresponding modular network, obtained by considering an FFNN to train each of the functional units in the above functional network.

3. Optimal model selection with genetic algorithms

Genetic algorithms and evolutive algorithms have received a lot of attention regarding their potential as optimization techniques for complex objective functions (e.g. non-differentiable or discontinuous functions). These techniques have been applied to numerous difficult problems of optimization, particularly in engineering and applied sciences (a complete discussion of genetic algorithms can be found in the books by Goldberg (1989) and Michalewicz (1999)). Genetic algorithms work with a population of individuals which are iteratively adapted towards the optimum by means of a random process of selection, recombination and mutation. During this process, a fitness function measures the quality of the population, and selection favors those individuals of higher quality.

Different genetic techniques have recently been applied both for optimizing the weights of neural networks and for pruning their topology, discarding useless connections (see van Rooij et al., 1997; Castillo et al., 2000; and references therein); they have also been applied for designing modular neural networks (Happel & Murre, 1992). Recently, the convenience of combining evolutive algorithms with hill-climbing methods for global optimization has been reported (Renders & Bersini, 1994). In this paper, we shall consider this approach for searching the optimal neuronal configuration of the functional units in an MFNN. To this aim, a fitness function is defined to keep the model as simple as possible, rewarding the quality of the fit to data and penalizing the functional complexity. A gradient-descent method is used for evaluating the fitness of a single individual (a functional network with a specific configuration for the hidden neurons) and the minimum description length criterion (Rissanen, 1989) is used for measuring its complexity.

In this section we present a brief description of the components of a genetic algorithm and analyze the specific implementations adopted for our problem.

3.1. A convenient coding of individuals

The first step in a genetic algorithm is to choose a convenient coding of individuals (modular networks), as a vector of binary, or decimal, real numbers. In our case,
Given a specific topology (an associative or separable network), a modular network is defined by the number of neurons \( n_i \) of the first and second hidden layers of each module \( i \). Thus, each individual is represented by a vector of integer numbers of the form

\[
m = (n_{i_1}, n_{i_2}, \ldots, n_{j_1}, n_{j_2})
\]

Each couple \( n_{i_1}, n_{i_2} \) in (5) corresponds to module \( i \) and indicates the number of neurons in each of the two hidden layers (note that we use a two-hidden-layer FFNN throughout the paper). The notation \( n_{i_1} : 0 \) or \( 0 : n_{i_2} \) is used to denote a neural network component with a single hidden layer, and \( 0 : 0 \) denotes the absence of the corresponding module. This notation plays an important role, especially for separable networks, since the required number of modules depends on the problem complexity and therefore it should be adaptively obtained.

For instance the codings \((0, 1; 3, 2; 0, 0)\) and \((2, 1; 1, 0; 0, 0)\) represent the networks shown in Figures 5(a) and 5(b), for associative and separable models, respectively (these networks are the optimal architectures obtained in Section 4 for two different time-series problems).

3.2. A fitness function to be maximized

Once the topology of a given individual \( m \) is known, a hill-climbing method (back-propagation for modular neural networks) is used for evaluating the quality of the fit to data (errors \( e_i = (x_i - \hat{x}_i)^2 \) between the observed \( x_i \) and the neural estimated data \( \hat{x}_i \)). On the other hand, the minimum description length criterion (Rissanen, 1989) is used for characterizing the network complexity. In our case, the model complexity is given by the weights of the modular neural network resulting from the training process. This results in the following ‘description length measure’ for a given individual \( m \):

\[
DL(m) = DL_p + DL_e = \sum_{i=1}^{u} \log_2(w_i) - \frac{u \log n}{2} - \frac{n}{2} \log \left( \frac{1}{n} \sum_{j=1}^{n} e_j \right)
\]

where \( u \) denotes the total number of weights \( w_i \), which are estimated using a set of data of size \( n \) (according to Rissanen (1989), each weight can be encoded using \( \log_2(w_i) + \log_2(n)/2 \) bits). Then, the opposite of the description length function \( f(m) = -DL(m) \) is used as the fitness function to be maximized.

One of the problems of this fitness function is that the gradient-descent algorithm may lead to different training errors for \( DL_e \) for different initial weights (there are multiple local minima). One possibility for taking this problem into account is training the network several times (starting from different random initial weights) and averaging the resulting errors (this will also increase the computational cost of the algorithm). In this paper we train each modular network five different times and use the median of the resulting errors to avoid outliers.

3.3. Recombination, mutation and inversion operators

Genetic operators are used to combine individual characteristics and to maintain the population diversity. We use three operators: crossover, mutation and inversion. A two-point crossover is used to exchange complete modules (chosen randomly with equal probability) between two individuals. For instance, given the parents

\[
\text{parent-1 : (2, 2; 3, 2; 0, 3)}
\]

\[
\text{parent-2 : (0, 2; 1, 1; 0, 1)}
\]
after recombination (e.g. two-point 2–3 crossover) we get the offspring

offspring-1 : (2, 2; 1,1; 0,1)
offspring-2 : (0, 2; 3,2; 0,3)

where modules between the second and third are exchanged in the parents. The value $p_{\text{cross}}$ determines the probability of each of the individuals to be included in the recombination pool.

Mutation works by increasing, or decreasing, the number of neuron functions of a module’s layer selected at random; this operator is applied with probability $p_{\text{mut}}$. Note that the above convention for zeros allows us to simplify the number of modules when the neurons are decreased to zero.

Simple inversion (a unary operator like mutation) builds offspring by choosing two positions at random within an individual and permuting the order of the modules (it is applied with probability $p_{\text{invert}}$).

### 3.4. A reproduction mechanism

The reproduction mechanism is applied to a given population $\text{pop}(t)$ (the initial population $\text{pop}(0)$ is generated at random). It selects a new intermediate population, $\text{pop}^\ast(t + 1)$, applying different probabilistic rules which reward best adapted individuals (those with higher fitness) from $\text{pop}(t)$ (e.g. stochastic universal sampling using the fitness of individuals as sample probabilities). Then, the intermediate population undergoes a specific recombination and mutation process (described in the previous section), resulting in a new population $\text{pop}(t + 1)$ ready to feed a new cycle. The size of the population is denoted $p_{\text{size}}$ and is kept constant through the process.

Standard genetic reproduction techniques have been designed for working with both a large population size and a large number of iterations for theoretical results about global optimum convergence to hold. In our case, the size of the population and the number of iterations are limited by the computational cost of evaluating the fitness of individuals (i.e. training the functional network with the hill-climbing feedforward method). Hence, more specific reproduction mechanisms, such as the modGA (see Michalewicz, 1999), are required to deal with the above restrictions. Suppose a population $\text{pop}(t)$ consists of $p_{\text{size}}$ individuals; then the modGA consists of the following steps.

- Select $r$ individuals to reproduce from $\text{pop}(t)$. Some of these parents will undergo crossover ($p_{\text{cross}}$), some others mutation ($p_{\text{mut}}$) and the rest inversion ($p_{\text{invert}}$).
- Insert the resulting $r$ offspring into $\text{pop}(t + 1)$.
- Select $p_{\text{size}} - r$ individuals from $\text{pop}(t)$ and copy them to $\text{pop}(t + 1)$.

The term ‘select’ in the above process refers to any standard genetic selection mechanism (e.g., stochastic universal sampling). In the modGA we can regulate changes in the population using the parameter $r$, which balances the selective
pressure and the population diversity, changing the proportion of individuals for crossover, mutation and inversion.

4. Experimental results: time-series analysis

Neural networks have been successfully applied to model chaotic time series. For example, Stern showed that a multilayer perceptron trained with the back-propagation algorithm outperforms standard autoregressive models to approximate a chaotic time series (Stern, 1996). Moreover, some chaotic time series are currently used as benchmarks for different modelling and prediction techniques. In this section we apply modular functional networks to this problem, considering two popular chaotic maps: the Lozi map and the Burger map.

4.1. The non-differentiable Lozi map

With the aim of illustrating the performance of the above genetic algorithm, we consider the Lozi map (Misiurewicz, 1980):

\[ x_n = 1 - 1.7|x_{n-1}| + 0.5x_{n-2} \]  

We have considered a 500-point time series obtained with the initial conditions \( x_0 = 0.5 \) and \( x_1 = 0.7 \). In this example we use the first 300 points of the above time series to train the functional network and the next 200 points to test the resulting models. The error used in the DL-p component of the fitness function (6) would be the test error. In this case we consider the associative modular network shown in Figure 3 and use a modGA version of the genetic algorithm to find the optimal neuronal configuration (intra-module topology) of the different functional units. Several experiments with different values of the parameters were performed, leading to different behaviors (note that in this paper we do not code the parameters of the algorithm into the genetic algorithm). Table 1 shows the information of an illustrative example. In this case, the population included only 10 individuals.

Note that an efficient modular network for the Lozi map (7) would concentrate most of the complexity (number of neurons) in the second module – associated with the absolute value function – whereas the first and third modules are simple linear functions. From Table 1 we can see how a population of 10 modular networks is iteratively evolved to more efficient modular structures for approximating the Lozi map. For instance, the best adapted individuals in the first three iterations are [0, 1; 2, 0; 1, 1], [2, 2; 2, 3; 1, 2] and [0, 1; 3, 1; 1, 1], with fitness 195.3, 278.1 and

Table 1: Population of MFNNs resulting from three generations (cycles) of the modGA with \( p_{\text{size}} = 10 \), \( r = 7 \), \( p_{\text{cross}} = 0.55 \), \( p_{\text{mut}} = 0.3 \), and \( p_{\text{invert}} = 0.15 \) trained using 300 points of data from the Lozi map.

<table>
<thead>
<tr>
<th>Generation</th>
<th>Network</th>
<th>Parameters</th>
<th>RMSE</th>
<th>Fitness</th>
<th>DL-p</th>
<th>DL-c</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[1 0, 2 1, 0 1]</td>
<td>17</td>
<td>0.111</td>
<td>179.79</td>
<td>39.14</td>
<td>-218.93</td>
</tr>
<tr>
<td>1</td>
<td>[1 2, 1 0, 0 0]</td>
<td>15</td>
<td>0.270</td>
<td>96.171</td>
<td>34.53</td>
<td>-130.71</td>
</tr>
<tr>
<td>1</td>
<td>[1 1, 0 2, 0 0]</td>
<td>15</td>
<td>0.108</td>
<td>187.13</td>
<td>34.53</td>
<td>-221.67</td>
</tr>
<tr>
<td>1</td>
<td>[1 2, 1 0, 1 0]</td>
<td>17</td>
<td>0.271</td>
<td>91.25</td>
<td>39.14</td>
<td>-130.39</td>
</tr>
<tr>
<td>1</td>
<td>[1 0, 1 1, 1 1]</td>
<td>16</td>
<td>0.481</td>
<td>36.20</td>
<td>36.84</td>
<td>-73.04</td>
</tr>
<tr>
<td>1</td>
<td>[1 0, 1 0, 1 2]</td>
<td>17</td>
<td>0.447</td>
<td>41.27</td>
<td>39.14</td>
<td>-80.41</td>
</tr>
<tr>
<td>1</td>
<td>[1 1, 2 0, 2 0]</td>
<td>18</td>
<td>0.193</td>
<td>123.03</td>
<td>41.44</td>
<td>-164.49</td>
</tr>
<tr>
<td>1</td>
<td>[2 2, 1 2, 1 2]</td>
<td>31</td>
<td>0.163</td>
<td>109.78</td>
<td>71.38</td>
<td>-181.16</td>
</tr>
<tr>
<td>1</td>
<td>[0 1, 2 0, 1 1]</td>
<td>16</td>
<td>0.098</td>
<td>195.34</td>
<td>36.84</td>
<td>-232.18</td>
</tr>
<tr>
<td>1</td>
<td>[2 1, 0 0, 1 1]</td>
<td>17</td>
<td>0.415</td>
<td>48.64</td>
<td>39.14</td>
<td>-87.78</td>
</tr>
<tr>
<td>1</td>
<td>[0 1, 2 0, 1 1]</td>
<td>16</td>
<td>0.117</td>
<td>177.71</td>
<td>36.84</td>
<td>-214.55</td>
</tr>
<tr>
<td>2</td>
<td>[1 0, 2 1, 0 1]</td>
<td>17</td>
<td>0.107</td>
<td>188.56</td>
<td>34.53</td>
<td>-223.10</td>
</tr>
<tr>
<td>2</td>
<td>[0 2, 1 0, 1 0]</td>
<td>19</td>
<td>0.105</td>
<td>185.86</td>
<td>39.14</td>
<td>-225.00</td>
</tr>
<tr>
<td>2</td>
<td>[1 1, 1 0, 1 2]</td>
<td>16</td>
<td>0.179</td>
<td>134.98</td>
<td>36.84</td>
<td>-171.83</td>
</tr>
<tr>
<td>2</td>
<td>[2 1, 1 0, 1 1]</td>
<td>19</td>
<td>0.269</td>
<td>87.52</td>
<td>43.74</td>
<td>-131.27</td>
</tr>
<tr>
<td>2</td>
<td>[1 2, 0 0, 0 0]</td>
<td>13</td>
<td>0.456</td>
<td>48.53</td>
<td>29.93</td>
<td>-78.46</td>
</tr>
<tr>
<td>2</td>
<td>[1 1, 2 0, 0 1]</td>
<td>16</td>
<td>0.112</td>
<td>181.33</td>
<td>36.84</td>
<td>-218.17</td>
</tr>
<tr>
<td>2</td>
<td>[2 2, 2 3, 1 2]</td>
<td>39</td>
<td>0.025</td>
<td>278.11</td>
<td>89.80</td>
<td>-367.91</td>
</tr>
<tr>
<td>2</td>
<td>[1 1, 0 2, 0 0]</td>
<td>15</td>
<td>0.107</td>
<td>188.56</td>
<td>34.53</td>
<td>-223.10</td>
</tr>
<tr>
<td>3</td>
<td>[0 1, 3 1, 1 1]</td>
<td>22</td>
<td>0.052</td>
<td>243.48</td>
<td>50.65</td>
<td>-294.13</td>
</tr>
<tr>
<td>4</td>
<td>[1 1, 3 1, 0 0]</td>
<td>20</td>
<td>0.051</td>
<td>249.73</td>
<td>46.05</td>
<td>-295.79</td>
</tr>
<tr>
<td>8</td>
<td>[0 0, 3 1, 1 1]</td>
<td>20</td>
<td>0.050</td>
<td>250.30</td>
<td>46.05</td>
<td>-296.35</td>
</tr>
<tr>
<td>14</td>
<td>[3 1, 4 1, 0 1]</td>
<td>31</td>
<td>0.004</td>
<td>430.83</td>
<td>71.38</td>
<td>-502.21</td>
</tr>
<tr>
<td>16</td>
<td>[0 1, 3 2, 0 0]</td>
<td>23</td>
<td>0.006</td>
<td>438.03</td>
<td>52.95</td>
<td>-490.99</td>
</tr>
</tbody>
</table>
As in the previous example we considered a time series which includes double interactions between the variables. (Whitehead & MacDonald, 1984)

As a second example, we consider the Burger map given by dealing with small populations. modGA which requires only a few generation cycles at hand. This example illustrates the efficiency of the information about the functional structure of the problem compromise between error and simplicity, thus providing that this model is not the one with lowest fit error, but a the structure underlying the data (see Figure 5(a)). Note works with large complexity in the second module. In third generation the genetic algorithm favors those net-generation is an appropriate model. Moreover after the We can see that even the model obtained in the third 243.4, respectively (note that the algorithm is not elitist). We can see that even the model obtained in the third generation is an appropriate model. Moreover after the third generation the genetic algorithm favors those networks with large complexity in the second module. In generation 16 the model obtained is perfectly adapted to the structure underlying the data (see Figure 5(a)). Note that this model is not the one with lowest fit error, but a compromise between error and simplicity, thus providing information about the functional structure of the problem at hand. This example illustrates the efficiency of the modGA which requires only a few generation cycles dealing with small populations.

4.2. The Burger map

As a second example, we consider the Burger map given by (Whitehead & MacDonald, 1984)

\[ x_n = (1.8 - x_{n-2}^2)x_{n-1} \]  

(8)

which includes double interactions between the variables. As in the previous example we considered a time series consisting of 500 points (300 for training and 200 for test) obtained for the initial conditions \( x_0 = 0.2 \) and \( x_1 = 0.3 \).

We consider the separable modular network shown in Figure 4(b) and proceed as we did in the previous example for obtaining an appropriate modular model for this problem, using a modGA version of the genetic algorithm. We also performed several experiments, obtaining a good performance of the algorithm with the parameters shown in Table 2. From this table we can study the evolution of the populations of 10 modular networks obtained in three different generation steps of the algorithm. The best model resulting after 10 generations ((2,1; 1,0; 1,0; 0,0) (shown in Figure 4(b) and proceed as we did in the previous example for obtaining an appropriate modular model for this problem, using a modGA version of the genetic algorithm. We also performed several experiments, obtaining a good performance of the algorithm with the parameters shown in Table 2. From this table we can study the evolution of the populations of 10 modular networks obtained in three different generation steps of the algorithm. The best model resulting after 10 generations ((2,1; 1,0; 1,0; 0,0) (shown in Figure 5(b)) is a simple model representing the functional structure underlying the time series (8) and providing a reasonable error.

5. Conclusion and further remarks

In this paper we have presented a genetic algorithm for searching an optimal modular network structure for a given problem. The resulting modular networks are
hybrids of functional and neural networks. A nice feature of the genetic algorithm is that it penalizes the network complexity while rewarding low fitting errors. Thus, simple models with reasonable generalization capabilities can be obtained for a given problem. To illustrate the performance of the algorithm, simple examples from time-series analysis are analyzed. The resulting modular networks capture the essential functional structure of the models underlying the time series. In the near future we shall analyze the performance of this algorithm when applied to real-world problems.

We implemented the genetic evolution of the modular architectures using Matlab and the Matlab Neural Network Toolbox (Demuth & Beale, 1998).

References


WHITEHEAD, R.R. and N. MACDONALD (1984) Chaotic mapping techniques in machine learning and nonlinear systems, on which he has published several books and papers (personales.unican.es/gutierjm). He is currently working on the application of these methods in meteorology.

M.L. Ivanissevich

Maria L. Ivanissevich is currently a regular associate professor in the Department of Exact and Natural Sciences at the University of Patagonia Austral, Argentina. She is a Master in Computer Science and her research interests include genetic algorithms and evolutive strategies in search, optimization and fractal image processing.