Hybrid neural networks: An evolutionary approach with local search

Eduardo Masato Iyoda and Fernando J. Von Zuben

Department of Computer Engineering and Industrial Automation (DCA), School of Electrical and Computer Engineering (FECC), State University of Campinas (UNICAMP), C.P. 6101, Campinas – SP, CEP 13083-970, Brazil
Tel.: +55 19 3788 3820; Fax: +55 19 3289 1395; E-mail: {emi, vonzuben}@dca.fee.unicamp.br

Abstract: Considering computational algorithms available in the literature, associated with supervised learning in feedforward neural networks, a wide range of distinct approaches can be identified. While the adjustment of the connection weights represents an omnipresent stage, the algorithms differ on three basic aspects: the technique chosen to determine the dimension of the multilayer neural network, the procedure adopted to specify the activation functions, and the kind of composition used to produce the output. Advanced learning algorithms should be developed to simultaneously treat all these aspects during learning, and an evolutionary learning algorithm with local search is proposed here. The essence of this approach is a synergy between genetic algorithms and conjugate gradient optimization, operating on a hybrid neural network architecture. As a consequence, the final neural network is automatically generated, and is characterized to be dedicated and computationally parsimonious.

1. Introduction

Single hidden layer feedforward neural networks have been successfully applied to function approximation problems. The Universal Approximation Theorem guarantees that this kind of structure can approximate any continuous function in compact functional spaces with arbitrary degree of precision [5,10,14]. The most advanced results also give rates of convergence, stipulating how many hidden neurons, with a given activation function (the same for all neurons), should be used to achieve a specific order of approximation [4,6].

Traditional feedforward neural networks usually take all hidden neurons with the same activation function, and the network output is produced by an additive composition of the activation functions. However, the activation function of each hidden neuron can be properly and automatically defined by means of constructive learning using projection pursuit techniques [15, 24,25]. In spite of its greater flexibility, the neural network architectures produced by constructive learning processes still present a strong restriction: additive composition is the only way to combine the possibly distinct activation functions in order to produce the network output. So, we present in this paper a new neural network architecture, called hybrid neural network [16], whose output is produced by a hybrid composition of activation functions that also accepts the multiplicative operator, besides the additive one.

The hybrid neural network can be described as a neural network composed of generalized neurons, which are arranged into a mixture of a layered and a cascade configuration. A generalized neuron differs from the conventional one in two aspects: the level of internal activity can be produced by means of an additive or multiplicative composition of the inputs; and the shape of the activation function may be previously defined or determined during learning.

This paper is organized as follows: in Section 2, the traditional single hidden layer neural network with identical activation functions is presented. Projection pursuit learning is introduced in Section 3. In Section 4, the hybrid neural network architecture is proposed to overcome the limitations presented by the previous models. Learning aspects of the hybrid neural
network and the evolutionary approach are discussed in Sections 5 and 6, respectively. Before presenting some simulation results in Section 8, an entire section is dedicated to compare the three distinct models in qualitative terms. Concluding remarks will be outlined in Section 9.

2. Single hidden layer neural networks with identical activation functions

Learning an input-output bounded mapping, defined in a multidimensional real space, from a set of examples can be regarded as synthesizing an approximation of an unknown function \( g(\cdot) : \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}^r \), where \( \Omega \) is a compact region.

For \( l = 1, \ldots, N \), let \( x_l \in \mathbb{R}^m \) be the column vector of independent variables, \( s_l \in \mathbb{R}^r \) be the column vector of output responses, and \( \epsilon_l \in \mathbb{R}^r \) be an additive stochastic column vector, with zero mean and fixed variance. Then a sampling process given by

\[
s_l = g(x_l) + \epsilon_l, \quad l = 1, \ldots, N,
\]

produces an input matrix \( X \in \mathbb{R}^{N \times m} \) and a corresponding output matrix \( S \in \mathbb{R}^{N \times r} \), with \( x_l^T \) and \( s_l^T \) as their respective rows. The goal is to use the matrices \( X \) and \( S \) to construct the best approximation model \( \hat{g}(\cdot) \), and adopt this model to obtain the estimate \( \hat{s} \), given any \( x \in \Omega \), such that \( \hat{s} = \hat{g}(x) \).

Conventional single hidden layer neural networks, traditionally employed in function approximation problems, can be defined as consisting of an additive weighted composition of the activations of the hidden neurons, plus a bias term. Each activation is also produced by an additive weighted composition of the current input vector \( x = [x_1, \ldots, x_m]^T \), plus a bias term. So, the estimated component \( \hat{s}_k \), \( k = 1, \ldots, r \), of the output vector \( \hat{s} = [\hat{s}_1, \ldots, \hat{s}_r]^T \) takes the form (see Fig. 1):

\[
\hat{s}_k = w_{0k} + \sum_{j=1}^{n} w_{jk} f\left( \sum_{i=1}^{m} v_{ij} x_i + v_{0j} \right), \quad (1)
\]

where:
- \( n \) is the number of hidden neurons;
- \( f \) is the activation function of the hidden neurons;
- \( v_{ij}, \ i \neq 0 \), denotes the hidden layer weight connecting the \( i \)-th input to the \( j \)-th hidden neuron;
- \( v_{0j} \) denotes the bias of the \( j \)-th hidden neuron;
- \( w_{jk}, \ j \neq 0 \), denotes the output layer weight connecting the \( j \)-th hidden neuron to the \( k \)-th output neuron;
- \( w_{0k} \) denotes the bias of the \( k \)-th output neuron.

Notice that this model suffers from a severe limitation: all the hidden neurons have the same activation function \( f(\cdot) \). If the activation function of each hidden neuron can be properly and automatically defined, then better rates of convergence may be achieved. This is exactly the purpose of projection pursuit techniques, which will be reviewed in the next section.

3. Projection Pursuit Learning

Projection Pursuit Learning (PPL) comprises a class of algorithms inspired by statistical techniques based on projection pursuit regression models [8]. Hwang et al. [15] presented a constructive algorithm based on projection pursuit techniques, capable of generating neural networks with a single hidden layer, \( m \) inputs and \( r \) outputs. Further, Von Zuben and Netto [24,25] introduced some improvements into the original algorithm, in order to conciliate the solvability condition [1] and PPL. These modifications led to a computationally efficient and less intensive algorithm.

In PPL, for each unknown multidimensional input-output mapping to be approximated, a pertinent activation function and associated weight connections are determined for each neuron individually, as a function of the learning set. The activation functions can be determined in a parametric (for example, using Hermite polynomials) or non-parametric (for example, using smoothing splines) way. The method starts with a single hidden layer neural network having only one hidden neuron, and adds new hidden neurons until the approximation task is accurately accomplished by an additive weighted composition of the activations of these hidden neurons.

The output of the neural network generated by PPL is given by (see Fig. 2):

\[
\hat{s}_k = w_{0k} + \sum_{j=1}^{n} w_{jk} f_j\left( \sum_{i=1}^{m} v_{ij} x_i + v_{0j} \right), \quad (2)
\]

where \( f_j(\cdot), \ j = 1, \ldots, n \) is the activation function to be defined for the \( j \)-th hidden neuron. Notice that, when the activation functions \( f_j(\cdot), \ j = 1, \ldots, n \), are identical, then Eqs (1) and (2) are completely equivalent. So, the model in Figure 1 is a particular case of the one in Fig. 2.

Now, the supervised learning process not only is responsible for determining optimal values for the connection weights \( v_{ij} (i = 0, \ldots, m; \ j = 1, \ldots, n) \)
Fig. 1. Traditional connectionist model used in function approximation problems. All the hidden neurons have the same activation function $f(\cdot)$. The network has only a single output ($r = 1$) for clarity purposes.

Fig. 2. Model associated with projection pursuit learning (PPL): each hidden neuron can have its own activation function $f_j(\cdot)$, $j = 1, \ldots, n$. Again, we present a network with a single output ($r = 1$).

In spite of its greater flexibility, the PPL model still presents a strong restriction: additive weighted composition is the only way to combine the possibly distinct activation functions. So, we propose in this paper a hybrid neural network architecture that also accepts the multiplicative operator besides the additive one. This hybrid architecture will be presented in the next section.

4. Hybrid neural networks

In this section, we will introduce a hybrid neural network, specially designed to overcome the limitations presented by the architectures discussed in the previous sections. This architecture allows the use of distinct activation functions in the hidden layer, and each output of the network is produced by a cascade of multiplicative and/or additive compositions of the activation functions.

Although we are going to present the single output case, it is straightforward to generalize the hybrid neural network to deal with multiple outputs, mainly because of the layered nature of the processing across the network. In this case, each output will require an independent cascade of compositions, but the hidden layer will be the same.

The hybrid model with a single output is depicted in Fig. 3. The output of the network is given by

$$\hat{s} = (\cdots ((z_1 \Theta_1 z_2) \Theta_2 z_3) \cdots)$$  \hspace{1cm} (3)
\[ c_{n-2} \Theta_{n-1} z_n, \]

where each \( \Theta_i, \ i = 1, \ldots, n-1, \) represents the additive or multiplicative operator, and \( c_j, \ j = 1, \ldots, n-2, \) denotes weights connecting the cascade of activations. The weighted output \( z_l, \ l = 1, \ldots, n, \) associated with each hidden neuron is given by

\[ z_l = w_l f_l \left( \sum_{i=1}^{m} v_{il} x_i + v_{0l} \right), \ l = 1, \ldots, n. \]

That is, \( z_l, \ l = 1, \ldots, n, \) is the weighted activation of the \( l \)-th hidden neuron.

Notice that, if the operators \( \Theta_i, \ i = 1, \ldots, n - 1, \) represent only addition and the weights \( c_j, \ j = 1, \ldots, n - 2 \) are all equal to 1, then Eqs (2) and (3) are completely equivalent, although Eq. (3) is restricted to the single output case, which implies that the traditional PPL model is a particular case of this hybrid model.

As the hybrid model depicted in Fig. 3 embodies the neural network models presented in Figs 1 and 2, it is evident that its approximation capability is superior to that of the previous models. However, for each application problem, the definition of:

1. the optimal number of hidden neurons;
2. the best activation functions for each hidden neuron;
3. the best composition of the activation functions; and
4. the weights of the neural network;

is not a trivial task.

The solution proposed in this paper is obtained by means of the application of a genetic algorithm with local search. A genetic algorithm is developed to select the best combination of activation functions for neurons in the hidden layer, out of a finite set of candidate functions. The maximum number of hidden neurons is given, and the evolutionary process is responsible for pruning the excess of neurons, if necessary. The genetic algorithm is also responsible for determining the best sequence of operators for the cascade of hybrid compositions. Each individual of the population represents a hybrid neural network, and the weights of the individuals are adjusted using a conjugate gradient algorithm [3,20]. This iterative optimization process makes the role of the local search, and was adapted to manipulate any cascade of hybrid compositions and any set of activation functions at the hidden layer.

5. Hybrid neural networks: Learning aspects

In this section, we will deduce an expression for the gradient of the squared error at the output of the hybrid neural network. This expression is required to adjust the weights of the hybrid neural network using the conjugate gradient algorithm. Here we will present only the main steps necessary for the derivation of this expression.

Observing Fig. 3, we can realize that all neurons considered (neurons at the hidden layer and neurons at
the cascade of hybrid compositions) can be interpreted as particular cases of a generalized model of a neuron, presented in Fig. 4. The term $u_j$ represents the level of internal activity of the generalized neuron, and is given by

$$u_j = \begin{cases} \sum_{i=1}^{p} w_{ji}x_i, & \text{if } \Theta_j \text{ is the additive operator} \\ \prod_{i=1}^{p} w_{ji}x_i, & \text{if } \Theta_j \text{ is the multiplicative operator} \end{cases}$$

where $p$ will be equal to $m$ for a neuron at the hidden layer, and equal to 2 for a neuron at the cascade of hybrid compositions.

The activation $y_j$ associated with the $j$-th hidden neuron at iteration $l$ (i.e., presentation of the $l$-th training pattern, $l = 1, \ldots, N$) is

$$y_j(l) = f_j(u_j(l)).$$

Neurons at the hidden layer employ additive internal composition and possibly distinct activation functions. On the other hand, neurons responsible for the cascade of hybrid compositions employ possibly distinct composition (additive or multiplicative) and fixed activation functions (the identity function).

The error signal at the output of neuron $j$, at iteration $l$, is defined by

$$e_j(l) = d_j(l) - y_j(l),$$

where $d_j(l)$ is the desired response for the neuron $j$. The instantaneous sum of squared errors of the network is thus written as:

$$\varepsilon(l) = \frac{1}{2} \sum_{j \in C} e_j^2(l),$$

where $C$ is the set of output neurons of the network.

Our goal is to compute the gradient of the instantaneous sum of squared errors relative to the synaptic weights. Following the same steps proposed by Haykin [13], properly adapted to deal with our hybrid architecture, it can be shown that each component of the gradient vector is given by

$$\frac{\partial \varepsilon(l)}{\partial w_{ji}(l)} = \begin{cases} -\delta_j(l)y_j(l), & \text{if } \Theta_j \text{ represents the additive operator} \\ -\delta_j(l)y_j(l) \prod_{k=0}^{p} w_{jk}(l)y_k(l), & \text{if } \Theta_j \text{ represents the multiplicative operator} \end{cases}$$

where:

- $w_{ji}(l)$ is the weight connecting neuron $i$ to neuron $j$, at iteration $l$;
- $y_j(l)$ is the activation of neuron $i$ ($i$-th input of neuron $j$), at iteration $l$;
- $p$ is the number of inputs to neuron $j$;
- $\delta_j(l)$ is the local gradient, at iteration $l$.

As neuron $j$ can be an output neuron or a hidden neuron, the local gradient can be defined as follows:

1. if $j$ is an output neuron, then $\delta_j(l)$ is given by
   $$\delta_j(l) = e_j(l)f'_j(u_j(l));$$
2. if $j$ is a hidden neuron, then $\delta_j(l)$ is given by
   $$\delta_j(l) = \begin{cases} f'_j(u_j(l)) \sum_k \delta_k(l)w_{kj}(l), & \text{if } \Theta_k \text{ is the additive operator} \\ f'_j(u_j(l)) \sum_k \delta_k(l)w_{kj}(l) \prod_{m=0}^{l} w_{km}(l)y_m(l), & \text{if } \Theta_k \text{ is the multiplicative operator} \end{cases}$$
where the index $k$ refers to all the neurons that are connected to the output of neuron $j$, $t$ is the number of inputs of neuron $k$, and the symbol $'$ denotes differentiation with respect to the argument.

With the gradient vector, we can apply any nonlinear optimization method [3,20] that uses first-order information to adjust the weights of the neural network. In this work, we used Møller’s scaled conjugate gradient [22], a powerful second order algorithm that considers consecutive gradient vectors to estimate the second order information.

6. The evolutionary approach

The activation functions $f_j(\cdot)$, $j = 1, \ldots, n$, are selected out of a finite set of candidates, given in Table 1. The reason to choose the first 7 candidates (numbers 0 to 6) is that they represent some of the most frequent types of activation functions considered in the literature. Candidate number 7, denoted null function, creates the possibility of pruning excessive flexibility in the case of less demanding problems. Reducing the effective number of neurons at the hidden layer is not the only way to control flexibility, because the adjustable connections at the cascade of compositions may cancel some terms simply by means of algebraic operations.

A genetic algorithm [21] is implemented to search for the best set of activation functions $f_j(\cdot)$, $j = 1, \ldots, n$, one for each hidden neuron (the numbers in Table 1 will be used in the codification), and for the corresponding best sequence of operators $\Theta_k$, $k = 1, \ldots, n - 1$, at the cascade of compositions.

In the sequence, the main aspects of the specific genetic algorithm used in this work will be described.

6.1. Codification

Each individual of the population represents a hybrid neural network with the same fixed size. As the number of candidates for activation function was chosen to be 8, the activation function of each hidden neuron can be encoded using 3 bits. Equivalently, each operator can be coded using 1 bit, 0 representing addition, and 1 denoting multiplication. For an example, see Fig. 5. For a network with $n$ hidden neurons, the length of the chromosome is given by $3 \times n + (n - 1) = 4 \times n - 1$.

Notice that the weights of the network are not considered in the codification, since they are adjusted by means of a conjugate gradient algorithm (local search).

6.2. Genetic algorithm

Let $P(t) = \{x^1_t, \ldots, x_n^t\}$ be the population of candidate solutions (hybrid neural networks) at generation $t$. Below, we present the main steps of the genetic algorithm, that was developed based on Michalewicz [21].

```
procedure genetic algorithm
begin
  $t \leftarrow 0$
  initialize $P(t)$ randomly
  evaluate $P(t)$
  while ($t < $ maximum-generation) do
    begin
      $t \leftarrow t + 1$
      select $P(t)$ from $P(t - 1)$
      apply crossover
      apply mutation
      evaluate $P(t)$
    end
  end
end
```

In the sequence, we will describe the details of the algorithm: evaluation of the individuals, selection and genetic operators.

6.3. Evaluation of the individuals

Each individual is decoded into the corresponding neural network, and then is trained 5 times, with different initial conditions assigned to the weights, using

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normalized hyperbolic tangent</td>
<td>$s = 1.716 \tanh \left( \frac{4}{5} x \right)$</td>
</tr>
<tr>
<td>1</td>
<td>Logistic</td>
<td>$s = \frac{1}{1 + e^{-x}}$</td>
</tr>
<tr>
<td>2</td>
<td>Linear</td>
<td>$s = x$</td>
</tr>
<tr>
<td>3</td>
<td>Cosine</td>
<td>$s = \cos(x)$</td>
</tr>
<tr>
<td>4</td>
<td>Gaussian</td>
<td>$s = e^{-0.5 \cdot x^2}$</td>
</tr>
<tr>
<td>5</td>
<td>Hyperbolic sine</td>
<td>$s = \sinh(x)$</td>
</tr>
<tr>
<td>6</td>
<td>Hyperbolic cosine</td>
<td>$s = \cosh(x)$</td>
</tr>
<tr>
<td>7</td>
<td>Null function</td>
<td>$s = 0$</td>
</tr>
</tbody>
</table>
the scaled conjugate gradient algorithm [22] for a fixed number of iterations. The fitness of an individual is defined as

$$\text{fitness} = \frac{1}{1 + \text{least}_{\text{MSE}}}$$

where least$_{\text{MSE}}$ is the lowest value among the 5 mean squared errors obtained from different initial conditions. The reason to adopt the fitness as the one produced by the best initial condition is that we are interested in measuring the effective potential of the individual, and the potential is properly determined by the best trial, and not by the average of trials.

6.4. Operators for selection

The selection of individuals to compose the next generation is elitist, that is, the best individual of the present generation is always preserved to compose the next generation. The other individuals of the population are selected by roulette wheel [21].

6.5. Genetic operators

The crossover operator used was the uniform crossover [23], and the mutation operator was the usual one, where a bit selected for mutation is flipped.

7. Qualitative aspects of nonparametric learning devices: A new paradigm of system modeling

Learning algorithms in which both the neural network architecture and its parameters are fitted to the data give rise to a powerful class of nonparametric devices. Every time the complexity of a learning task cannot be determined a priori, the possibility of adjusting the network structure should be explored to estimate the effective flexibility of the approximation model. The question is how to achieve the optimal neural network configuration that correctly maps each input pattern to the corresponding desired output pattern [2].

Computational learning theory is continuously searching for more powerful devices to represent and process information. The ultimate objective is to design better solutions based on the shortest demand for computational resources. The main consequence of this initiative is the replacement of generic purpose devices by strongly dedicated ones. This is a necessary condition to reach parsimonious solutions to the more challenging problems in computational intelligence [17]. As the structure of the final solution is not defined a priori, and depends on the nature and complexity of the problem, the resulting models are called nonparametric devices [12].

The universal approximation capability [5,10,14] presented by neural network models indicates that it is theoretically better for the number of hidden units to be as large as possible. On the other hand, from the standpoint of generalization, this number should not be too large, mainly because of the finiteness of the learning data set and the existence of noise in the sample process that originates the input-output data set [9,11].

The main restriction of the nonparametric approaches already presented in the literature, and involving neural network models, is that the output of all hidden neurons are combined employing a purely additive or purely multiplicative composition [7,18]. However, when the nature of the mapping to be approximated is not purely additive or purely multiplicative, it would certainly be possible to obtain better performance considering a hybrid composition, with some neurons contributing in an additive and others in a multiplicative manner, producing what we call a hybrid neural network.

Here, the essential meaning of better performance deserves a detailed explanation. The performance may be measured considering three distinct aspects:
– the approximation error produced by the solution: theoretically, the three types of neural network models considered in this paper (see Sections 2, 3 and 4) are universal approximators, because the single hidden layer neural network with identical activation functions, presented in Section 2, was proven to be a universal approximator [5,10,14], and happens to be a particular case of the models presented in Sections 3 and 4. So, given an approximation problem, it is not possible to anticipate which one of the three models will present better performance in terms of approximation error [26], no matter the efficiency of the learning algorithm and the adequacy of the initial conditions, when applicable.

– the computational cost to obtain the solution: non-parametric neural network devices invariably require more computational resources to determine all the adjustable attributes of the corresponding architecture. Not only the weights have to be determined, but also the number of hidden neurons, the shape of the activation functions, and the kind of composition among these activations. A projection pursuit neural network is also a nonparametric device, but presents a fixed additive composition at the output layer. So, optimizing all the attributes of a hybrid neural network is certainly the most demanding learning task, but still preserves the computational tractability.

– the computational cost required to implement the obtained solution: as will be shown by the simulation results associated with the most demanding tasks, the hybrid neural network produces the most parsimonious final solution among the three types of models considered. With more attributes of the neural network structure being optimized, the resulting neural network is a dedicated solution. Basically, as already done in the case of projection pursuit learning, the search for the solution is performed in an iterative way, that progressively tries to minimize redundancy and to maximize orthogonality of representation among neurons.

So, nonparametric learning devices represent a new paradigm of system modeling. In this new paradigm, there is a tendency to increase the cost of the design aiming at producing a more parsimonious solution at the end of the process. This is exactly what should be expected in practical applications. For example, if the resulting model represents the solution to an engineering problem, the corresponding computational device will probably be implemented in an industrial scale. So, at the final cost of the industrial product, the computational cost required to implement the obtained solution (smaller for nonparametric models) will prevail over the computational cost to obtain the solution (higher for nonparametric models).

8. Simulation results

Every time that nonparametric models of the type considered here are being proposed, i.e. nonparametric devices produced by a hybrid composition of simpler modules, two types of questions are raised [12]:

1. what would be the performance of the learning process when just one or a small set of these modules has to be present at the final solution? In other words, at which extent is the ability of the learning process to prune flexibility when the solution requires just a fraction of the whole potential presented by the model?

2. what would be the performance of the learning process when the complexity of the problem explores the whole potential presented by the model?

The first question is related to the ability to produce a solution that presents the same level of complexity inherently associated with the problem. The second question guides to an evaluation of the adequacy of the set of modules chosen to take part in the hybrid composition. A positive answer to both questions is given every time the learning process is capable of producing a version of the most parsimonious solution that can be obtained.

So, to evaluate the performance of the proposed hybrid neural network models, we divided the simulations into two parts. In the first part, subdivided into three stages, we tested the ability of the hybrid neural networks to approximate functions that can be formulated in terms of the candidate functions of Table 1. Each candidate function can be directly related to the concept of a module, in the context mentioned above. In the second part, we compare the performance of the hybrid neural network models, we divided the simulations into three stages, we tested the ability of the hybrid neural networks to approximate functions that can be formulated in terms of the candidate functions of Table 1. Each candidate function can be directly related to the concept of a module, in the context mentioned above. In the second part, we compare the performance of the hybrid neural network with other architectures, when applied to more challenging tasks. In all the simulations, in order to evaluate individuals of the population according to Section 6.3, each individual at a given generation of the evolutionary process is trained during 50 iterations, using the scaled conjugate gradient algorithm [22].
Fig. 6. Example of a solution obtained at the first stage of simulation. Here we present a hybrid neural network obtained for \( g(\cdot) \). Notice that only the forth and fifth neurons, with cosine and normalized hyperbolic tangent activation functions, respectively, have a significant contribution to the output, guiding to the normalized hyperbolic tangent mapping.

\[
\hat{x} = (-1.05 + f_s(0.00x + 0.40) + 1.48) + 0.70 + f_s(-1.00x + 0.00)
\]

\[
= (0.10 \cdot \cos(0.00x + 0.40) + 1.48) + 0.70 \cdot \tanh(\frac{2}{3}(-1.00x + 0.00))
\]}

\[
\approx 1.716 \cdot \tanh(\frac{2}{3}x)
\]

Fig. 7. Example of a solution obtained at the first stage of simulation. Here we present a hybrid neural network obtained for \( g(\cdot) \). Notice that only the fifth neuron, with a linear activation function, has a significant contribution to the output, guiding to the identity mapping.

\[
\hat{x} = 0.71 \cdot f_s(1.41x + 0.00) \approx x
\]

8.1. Approximation of functions from the set of candidates

This part of the simulations was subdivided into three stages. Here, the input-output data available for supervised learning is purposely generated from an assumed unknown but very simple hybrid neural network, with the restriction of presenting an architecture that can be exactly described using the model presented in Fig. 3.

In this case, the function to be approximated \( g(\cdot) \) can be precisely reproduced by the approximation model \( \hat{g}(\cdot) \). As a consequence, the results to be presented in this section should not be used to outline general conclusions about the approximation capability of the model or the effectiveness of the learning process.

In all the simulations associated with the three stages, we used a population composed of hybrid neural networks with 5 hidden neurons, and 50 input-output train-
Stage 1. The hybrid neural network is applied to approximate data generated from an assumed unknown function \( g(\cdot) \), completely equivalent to a neural network with just one hidden neuron, whose activation function is chosen among the candidates described in Table 1. We choose the hyperbolic tangent and linear function as two examples:

\[
\delta = 0.68 \times f_1(0.70x + 0.32) \times (-0.90) + 1.00 \times f_1(0.43x - 1.21) \times 0.85 \\
\times (-0.90 \times f_1(1.00x + 0.00) \times (-0.82)) \times (-1.12 \times f_1(1.00x + 0.00)) \\
\equiv (0.00x - 1.41) \times 0.85 \times (0.74 \times \cos(1.00x + 0.00)) \times (-1.12 \times \cos(1.00x + 0.00)) \\
\equiv \cos^2(x)
\]

Fig. 8. Example of solution found at the second stage of simulation. A network obtained for \( g_3(\cdot) \) is depicted. Notice again the presence of a zero valued weight at the cascade of hybrid compositions.

\[
\delta = 1.04 \times f_1(1.00x + 0.00) \times 0.96 + 1.00 \times f_1(0.99x + 0.00) \\
\equiv \cos(x) + \frac{1}{1 + e^{-x^2}}
\]

Fig. 9. Example of solution found at the second stage of simulation. A network obtained for \( g_4(\cdot) \) is depicted. In this case, the sub-network above the dashed line produces an almost constant output \( r \approx -1.20 \) in the approximation interval.

ing samples, uniformly generated in the interval \([-5, 5]\).
\[
\hat{x} = \left[ 0.06 \cdot f_1(-0.48x - 0.29) \cdot 0.47 \cdot f_2(-0.12x + 0.10) \cdot (-0.19) \cdot 1.08 \cdot f_1(-1.28x + 0.00) \right] \\
\times 0.81 \cdot f_3(-x) + 10.00 \\
= \left[ 0.00 \cdot x \cdot \cotanh(-0.48x - 0.29) \cdot 0.00 - 1.38x \right] \left( -0.72 \cdot e^{-0.3\hat{x}} \right) + \frac{1}{1+e^{-x}}
\]

Fig. 10. Example of a solution obtained at the third stage of simulation for \( g_5(\cdot) \).

<table>
<thead>
<tr>
<th>Function</th>
<th>Number of effective hidden neurons</th>
<th>Activation functions (see Table 1)</th>
<th>Sequence of compositions</th>
<th>Mean squared error</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1(\cdot) )</td>
<td>1</td>
<td>( f_0(\cdot) )</td>
<td>+</td>
<td>( 10^{-6} )</td>
</tr>
<tr>
<td>( g_2(\cdot) )</td>
<td>1</td>
<td>( f_2(\cdot) )</td>
<td>+</td>
<td>( 10^{-6} )</td>
</tr>
<tr>
<td>( g_3(\cdot) )</td>
<td>2</td>
<td>( f_3(\cdot), f_4(\cdot) )</td>
<td>+</td>
<td>( 10^{-5} )</td>
</tr>
<tr>
<td>( g_4(\cdot) )</td>
<td>2</td>
<td>( f_3(\cdot), f_3(\cdot) )</td>
<td>( \times )</td>
<td>( 10^{-5} )</td>
</tr>
<tr>
<td>( g_5(\cdot) )</td>
<td>3</td>
<td>( f_2(\cdot), f_3(\cdot), f_4(\cdot) )</td>
<td>( \times + )</td>
<td>( 10^{-6} )</td>
</tr>
</tbody>
</table>

\[
g_1(x) = 1.716 \tanh \left( \frac{2}{3} x \right) \quad \text{and} \quad g_2(x) = x.
\]

The parameters for the genetic algorithm were:
- Population size: 5.
- Maximum number of generations: 10
- Crossover probability: 0.25
- Mutation probability: 0.01

Figures 6 and 7 depict examples of solutions found for the approximation of functions \( g_1(\cdot) \) and \( g_2(\cdot) \), respectively. An interesting aspect to observe is the value \( 0.00 \) attributed to weights belonging to the cascade of hybrid compositions, in both figures. Notice that these weights cancel the effect of the combined activations produced by the previous hidden neurons. As a practical consequence, the resulting neural networks have an effective number of neurons that are in fact less than the total number of neurons available to be composed. As expected, the network outputs of Figs 6 and 7 may be given by \( \hat{x} \equiv 1.716 \tanh(2x/3) \) and \( \hat{s} \equiv x \), respectively.

2) Stage 2. At this stage, we try to approximate an assumed unknown composition of two activation functions, both chosen among the candidates described in Table 1. The following functions were considered:

\[
g_3(x) = \frac{1}{1+e^{-x}} + \cos(x) \quad \text{and} \quad g_4(x) = \cos^2(x).
\]

Function \( g_3(\cdot) \) is an additive composition of functions 1 and 3, both described in Table 1. Function \( g_4(\cdot) \) is a multiplicative composition of function 3 with itself. The parameters for the genetic algorithm were:
- Population size: 10.
- Maximum number of generations: 10.
- Crossover probability: 0.25.
- Mutation probability: 0.01.
Table 3: Simulation results for \( g_6(\cdot) \)

<table>
<thead>
<tr>
<th>Architecture</th>
<th>No. of hidden neurons and sequence of compositions</th>
<th>Mean squared error</th>
<th>Normalized computational cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>12 (+)</td>
<td>( 10^{-4} )</td>
<td>1.00</td>
</tr>
<tr>
<td>PPL</td>
<td>2 (+)</td>
<td>( 10^{-4} )</td>
<td>1.16</td>
</tr>
<tr>
<td>HNN</td>
<td>2 (+)</td>
<td>( 10^{-7} )</td>
<td>2.01</td>
</tr>
</tbody>
</table>

\[
\hat{x} = 1.00 + f_3(1.00x_1 + 1.00x_2) + 1.00 * f_4(1.00x_1 - 1.00x_2) + \cos(x_1 + x_2) + e^{-0.5(x_1^2)}
\]

Fig. 11. Hybrid neural network obtained as an approximation of function \( g_6(\cdot) \).

Figures 8 and 9 show examples of solutions obtained for \( g_3(\cdot) \) and \( g_4(\cdot) \), respectively. In Fig. 8, the zero valued weight in the cascade hybrid composition layer cancels the effect of the first three hidden neurons, so that the output of the network is produced only by an additive composition of the last two hidden neurons. An interesting fact to point out about Fig. 9 is that the sub-network above the dashed line produces an almost constant output \( r \approx -1.20 \), working as a kind of bias. This is a good example to illustrate that, at this stage, solutions with different compositions and with more than 2 effective hidden neurons may be obtained. So, some kind of basic algebraic simplification is necessary to obtain the most parsimonious neural network from the evolved one, as depicted in Fig. 9.

3) Stage 3. At this stage, a more complex composition of activation functions was considered, all chosen among the candidates described in Table 1. The function considered is as follows:

\[
g_5(x) = \frac{1}{1 + e^{-x}} + xe^{-0.5x^2}.
\]

The parameters for the genetic algorithm were:

- Maximum number of generations: 50.
- Crossover probability: 0.25.
- Mutation probability: 0.01.

In Fig. 10, we depict an example of a network obtained for \( g_5(\cdot) \). At this stage, solutions for \( g_5(\cdot) \) with different compositions and with more than 3 effective hidden neurons were more frequent than at previous stages. But the final solution had always less than 5 effective hidden neurons.

Table 2 presents a summary of the results obtained at the first three stages of simulation.

8.2. Comparison with other architectures

In this section, the performance of the hybrid neural network will be compared with other architectures. The architectures to be considered are:

- **MLP:** Multilayer Perceptron with one hidden layer (see Fig. 1). All the hidden neurons used the hyperbolic tangent as activation function and the weights were adjusted by means of a conjugate gradient algorithm. The number of neurons at the hidden layer is given by the minimum number necessary to achieve the specified performance, and is obtained by trial and error.

- **PPL:** Projection pursuit learning architecture (see Fig. 2), with the improvements introduced by Von Zuben and Netto \[24,25\]. The activation functions were determined in a parametric way, using Hermite polynomials. The maximum order of the Hermite polynomials was defined to be 9.

- **HNN:** Hybrid neural network (see Fig. 3). The set of activation functions and the sequence of operators were determined by a genetic algorithm, as described in Section 6. The weights were adjusted by means of a conjugate gradient algorithm.

This section will be divided into two sub-sections, each one corresponding to a distinct problem of function approximation. The stopping criterion adopted in the cases of MLP and PPL was the satisfaction of a specific level for the approximation error. In the case of HNN, a maximum number of generations was adopted.
Table 4
Simulation results for $g_7(\cdot)$

<table>
<thead>
<tr>
<th>Architecture</th>
<th>No. of hidden neurons and sequence of compositions</th>
<th>Testing error (FVU)</th>
<th>Normalized computational cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>$15 (+ + \ldots +)$</td>
<td>$10^{-2}$</td>
<td>2.39</td>
</tr>
<tr>
<td>PPL</td>
<td>$5 (+ + + +)$</td>
<td>$10^{-2}$</td>
<td>1.00</td>
</tr>
<tr>
<td>HNN</td>
<td>$4 (+ \times +)$</td>
<td>$10^{-2}$</td>
<td>48.68</td>
</tr>
</tbody>
</table>

8.2.1. Approximation of a composition of candidate functions

In this problem, the task is to approximate a combination of functions of the candidate set (Table 1):

$$g_6(x_1, x_2) = \cos(x_1 + x_2) + e^{-0.5(x_1 - x_2)^2}.$$  

For this problem, the training set consists of 225 input-output samples uniformly distributed in the compact region $[0, 1] \times [0, 1]$. The parameters for the genetic algorithm were:

- population size: 5;
- number of generations: 20;
- crossover probability: 0.25;
- mutation probability: 0.01.

The results obtained at this stage are depicted in Table 3. In this problem, the hybrid neural network was able to achieve the best result in terms of mean squared error. This result is in accordance with the behavior already produced by the HNN model in the previous section, since the functions that compose $g_6(\cdot)$ belong to the set of candidate functions and, therefore, $g_6(\cdot)$ can be exactly approximated by the hybrid neural network. The computational cost for the hybrid neural network was higher, as expected. See Fig. 11 for an illustration of the hybrid neural network obtained for this problem.

8.2.2. Approximation of a demanding function

Here, the following function was used to test the performance of the neural network models:

$$g_7(x_1, x_2) = 1.9 \times 1.35 + e^{x_1 - x_2} \sin(13(x_1 - 0.6)^2) \sin(7x_2).$$

The task of approximating function $g_7(\cdot)$ is known to be very demanding [15]. See Fig. 12 for a plot of $g_7(\cdot)$, restricted to the compact region of approximation.

The training set consists of $N_{tr} = 225$ input-output samples, uniformly distributed in the compact region $[0, 1] \times [0, 1]$. The test set was composed of $N_{ts} = 10000$ input-output samples (independent from the training set) uniformly distributed in $[0, 1] \times [0, 1]$. To evaluate the performance of the three architectures, we used the fraction of variance unexplained (FVU), defined as
Fig. 13. The fitness of the best individual (solid line) and the mean fitness of the population (dotted line) along the generations for the problem of approximating \( g_7(\cdot) \).

\[
FVU = \frac{\sum_{l=1}^{N_t} (\hat{g}_7(x_{1l}, x_{2l}) - g_7(x_{1l}, x_{2l}))^2}{\sum_{l=1}^{N_t} (\hat{g}_7(x_{1l}, x_{2l}) - \bar{g}_7)^2},
\]

where

\[
\bar{g}_7 = \frac{1}{N_t} \sum_{l=1}^{N_t} g_7(x_{1l}, x_{2l}).
\]

Notice that the FVU is proportional to the mean squared error. The FVU has also been used in Hwang et al. [15], and Kwok and Yeung [19].

The parameters used for the genetic algorithm were:

- population size: 20;
- number of generations: 100;
- crossover probability: 0.4;
- mutation probability: 0.01.

The population was composed of hybrid neural networks with 5 hidden neurons, because this number of neurons is the minimum obtained with the application of projection pursuit learning [15,19,25].

The best results obtained are described in Table 4, with the 3 architectures presenting a similar performance in terms of approximation error. However, although requiring a higher computational cost to obtain the solution (see Table 4), the hybrid neural network was able to provide a solution with only 4 effective neurons, that is, the hybrid neural network could find a more parsimonious solution in terms of implementation cost. In Fig. 13, we show the fitness of the best individual and the mean fitness of the population along generations. In Fig. 14, we depict the hybrid neural network obtained for this problem. Notice the presence of one activation function of the type 7 (the null function) in the final solution with 5 neurons.

As already mentioned, in all the solutions obtained we should consider two kinds of computational cost involved: a cost to obtain the solution, and a cost to implement the solution, once obtained. The cost to obtain the solution using a hybrid neural network is higher, because it involves the cost of the evolutionary process plus the cost of the local search to be applied to all the individuals of the population, at each generation. However, the result presented in Table 4 shows that the hybrid composition of activation functions (additive and multiplicative operators are used) may lead to solutions whose implementation involves less computational resources, expressed in terms of the number of hidden neurons.

9. Conclusions

In this paper, we presented a hybrid neural network architecture that uses different activation functions at
each hidden neuron, and the output of the network is produced by a cascade of compositions of the activation functions. The cascade of compositions may use additive or multiplicative operators. A genetic algorithm was used to find the best set of activation functions and the best sequence of operators, while the weights remain being adjusted by a second-order optimization method, properly adapted to deal with the resulting hybrid architecture. Notice that the effective nature and complexity of the mapping to be approximated have to be automatically determined by the learning process, and not be considered as a previously available information.

Simulation results showed that the proposed learning algorithm to generate hybrid neural networks was able to find higher-quality solutions, when compared with other advanced learning algorithms, in terms of the amount of computational resources necessary to implement the final solution.

The dedicated and parsimonious nature of the solution can be interpreted as a direct consequence of the power to deal with the significant increment in flexibility associated with hybrid neural network architectures.

Acknowledgments

The first author would like to thank CNPq for the financial support (process 133691/97-7). The second author would like to thank FAPESP (process 98/09939-6) and CNPq (process 300910/96-7) for the financial support.

References


