Exploiting the functional training approach in B-Splines

António E Ruano1,2, Cristiano L. Cabrita2
Pedro M. Ferreira1,2,3, László T. Kóczy4

1Centre for Intelligent Systems, IDMEC, IST
2University of Algarve, 8005-139 Faro, Portugal
3Algarve STP – Algarve Science & Technology Park, 8005-139
4Faculty of Engineering Sciences, Széchenyi István University, Győr, Hungary

(e-mail: aruano@ualg.pt, ccabrita@ualg.pt, pfrazao@ualg.pt, koczy@sze.hu)

Abstract: When used for function approximation purposes, neural networks belong to a class of models whose parameters can be separated into linear and nonlinear, according to their influence in the model output. This concept of parameter separability can also be applied when the training problem is formulated as the minimization of the integral of the (functional) squared error, over the input domain. Using this approach, the computation of the gradient involves terms that are dependent only on the model and the input domain, and terms which are the projection of the target function on the basis functions and on their derivatives with respect to the nonlinear parameters, over the input domain. These later terms can be numerically computed with the data.

The use of the functional approach is introduced here for B-splines. An example shows that, besides great computational complexity savings, this approach obtains better results than the standard, discrete technique, as the performance surface employed is more similar to the one obtained with the function underlying the data. In some cases, as shown in the example, a complete analytical solution can be found.

Keywords: Neural networks training, parameter separability, functional back-propagation

1. INTRODUCTION

Neural networks and neuro-fuzzy systems are recognized tools for nonlinear systems identification. In this role, they act as function approximators and their topology is a linear combination of basis functions, which are, themselves, nonlinear in their parameters. In fact, they belong to a class models whose parameters can be separated into linear and nonlinear, according to their influence in the model output. This concept, for general nonlinear least square problems was introduced in (Golub and Pereyra, 1973). It was proposed to the neural community in (Ruano et al., 1992). Other authors have, more or less explicitly exploited this concept (see, for instance, (Sjoberg and Viberg, 1997)). A good review on the applications of the concept of separability of parameters in general nonlinear least-squares problems can be found in (Golub and Pereyra, 2003).

Having in mind that the ultimate goal of modeling is to obtain a “good” approximation of the function behind the data, and not to the data in itself, the modeling problem can be formulated as the minimization of the integral of the (functional) squared error, over the input domain, and not as the usual sum of the square of the errors. The concept of parameter separability can be also applied to this functional approach, resulting in two types of terms: i) that are dependent only on the model and the input domain, and independent of the target function, and ii) other terms which are dependent on the model, the target function and the input domain. The first class of terms can be analytically computed a priori, according to the model type and topology. The second terms, if the target function was known and an analytical solution could be found, could also be obtained and a complete analytical solution would be available for the training of this class of models. As in practice what is available is the data, the target data can be used to numerically approximate these second terms.

This approach, called functional approach, was introduced in (Ruano et al., 2011), and applied for Radial Basis function networks in (Cabrita et al., 2011). The objective of this paper is to illustrate its application to B-spline networks. A review of the proposed methodology is provided in Section 2. Section 3 illustrates its application to B-splines. A didactical example is given in Section 4. Section 5 concludes the paper and points directions for future work.

2. METHODOLOGY

All models addressed in this paper are assumed to have their parameters separable into linear (denoted by $u$) and nonlinear (denoted by $v$), according to the dependence on the model output on the parameters:

$$y(X, v, u) = \Gamma(X, v)u$$

In (1), it is assumed that the training data is, as usual, discretized. Considering $m$ patterns, $y$ is the output model, a vector with $m$ elements, and $X$ is the input vector, with dimensions $m \times n$ ($n$ is the number of inputs). It is assumed that the model has $n_1$ linear parameters and $n_2$ nonlinear parameters. $\Gamma$ is the matrix of the basis functions (dimensions $m \times n_2$). The training criterion normally employed is:
\[
\Omega_{\nu}(X,v,u,t) = \left\| \Gamma(X,v) \right\|_{\nu}^2 = \left\| \Gamma(X,v,u,t) \right\|_{\nu}^2, 
\]
where \( t \) is the target vector, \( e \) is the error vector and \( \left\| \cdot \right\|_{\nu} \) is the Euclidean norm.

Let us assume that the function generating the data was known, and let it be denoted by \( t(x) \). In this case, (1) and (2) could be recast as:

\[
y(x,v,u) = \Phi_t(x,v) u \\
\Omega_{\nu}(x,v,u,t) = \frac{\int_{x} \left( t(x) - \Phi_t(x,v) u \right)^2 \, dx}{2} \\
\frac{\int_{x} \left( t - u^T \Phi_t(x,v) \right)^2 \, dx}{2} = 2
\]

Notice that in (3) \( X \) is now a multi-dimensional real variable \( x = x_1, x_2, \ldots, x_n \), and that in (4),

\[
f(x,\nu,\nu) = \int_{x} \cdots \int_{x} \int_{x} f(x_1, x_2, \ldots, x_n) dx_1 \cdots dx_n
\]

\( y, t \) and \( e \) are real functions. \( \Phi \) is a vector of basis functions, and not a matrix as in the discrete case. In order to identify the two cases, the subscript \( d \) will denote the discrete case and \( f \) the functional case, whenever needed. The Jacobian matrix (for the discrete case) is given as

\[
J(x,v,u) = \frac{\partial y(X,v,u)}{\partial u} \bigg|_{u^T} = \left[ \Gamma(X,v) \bigg|_{\nu} \right] \frac{\partial \Omega(X,v,u)}{\partial v}
\]

For the functional case

\[
J(x,v,u) = \frac{\partial y(x,v,u)}{\partial u} \bigg|_{u^T} = \left[ J(x,v) \bigg|_{\nu} \right] \frac{\partial \Omega(x,v,u)}{\partial v}
\]

\[\Phi_t(x,v) u \]

For training, another important operator is the gradient, which for the discrete case, is given as

\[
g \Omega_{\nu}(X,v,u,t) = \frac{\partial \Omega_{\nu}}{\partial u} \bigg|_{u^T} = -J^T(X,v,u) e(X,v,u,t)
\]

For the functional case:

\[
g \Omega_{\nu}(x,v,u,t) = \frac{\partial \Omega_{\nu}}{\partial u} \bigg|_{u^T} = -J^T(x,v,u) e(x,v,u,t)
\]

\( \Phi_t(x,v) u \)

For the discrete case:

\[
\hat{u}_j(X,v,t) = (\Gamma^T \Gamma)^{-1} \Gamma^T t
\]

where the symbol \( ^+ \) denotes a pseudo-inverse. For the continuous case, equating (9) to 0:

\[
\int_{x} \phi(x) - \Phi_t(x,v) u \, dx = 0 \Rightarrow \int_{x} \phi(x) \, dx \rightleftarrows \int_{x} \phi(x) \, dx \rightleftarrows \int_{x} \phi(x) \, dx
\]

\[\phi(x) \]

We can incorporate this value in the criterion and introduce a new criterion (independent of the linear parameters). For the discrete case,

\[
\psi_{\nu}(x,v,t) = \frac{1}{2} \left[ \Phi_t(x,v) \right] \left( \int_{x} \phi(x) \, dx \right)^2
\]

\[\Phi_t(x,v) \]

For the functional case:

\[
\psi_{\nu}(x,v,t) = \frac{1}{2} \left[ \Phi_t(x,v) \right] \left( \int_{x} \phi(x) \, dx \right)^2
\]

If we want to compute the gradient of the new criterion, for the discrete case, it can be proved that:

\[
g_{\psi_{\nu}}(x,v,t) \frac{\partial \psi_{\nu}}{\partial v} = -J^T e
\]

For the functional case:

\[
g_{\psi_{\nu}}(x,v,t) \frac{\partial \psi_{\nu}}{\partial v} = -J^T e
\]

2.2 Computation of terms

Let us assume that we want to minimize criterion (14). If the Back-Propagation (BP) method is used, the following update is used:

\[
v[k+1] = v[k] + s[k] \\
s[k] = -\eta g_{\nu}[k]
\]

The gradient is obtained from (17). It requires the availability of \( \hat{u}_j \) (12). Then, Eq. (17) can be expressed as:

\[
g_{\psi_{\nu}}(x,v,t) \frac{\partial \psi_{\nu}}{\partial v} = -J^T e
\]

Therefore, to apply (18), we would need to have available the
mathematical expressions of:

\[
\begin{align*}
\frac{\partial}{\partial x} \int_{x_m}^{x_u} \phi(x) dx &= \int_{x_m}^{x_u} \phi(x) dx \\
\int_{x_m}^{x_u} \phi(x) dx &= \frac{\phi(x) \int_{x_m}^{x_u} \phi(x) dx}{2k^2} + \int_{x_m}^{x_u} \phi(x) dx
\end{align*}
\]  

(20) which are independent of the function to approximate, and can be obtained analytically, off-line, for the model at hand, and evaluated on-line, during training.

The only terms involving the function to approximate are:

\[
\begin{align*}
\int_{x_m}^{x_u} \phi(x) dx \\
\int_{x_m}^{x_u} \phi(x) dx
\end{align*}
\]  

(21) In a practical application the underlying function is not known (otherwise it should be used). The integrals (22) and (23) can be numerically approximated using the training data and a suitable numerical integration technique.

Besides being an elegant solution – the gradient is computed with a set of terms that only depend on the model and the input domain, and another set of terms which are the projection of the function to approximate on the basis functions and on their partial derivatives, over the domain – the use of this approach reduces also the computational complexity. If we observe the computation of the discrete gradient (eqs (15) to (17)) they involve the computation of a pseudo-inverse (which has, at least a complexity of \(O(mn^3)\)) and matrix multiplications, whose complexity is \(O(mn^2)\).

If (19) is used, 4 matrix-vector multiplications are needed, but the quantities involved have only size \(n_i\), and are independent of \(m\). The numerical computation of the projections has a complexity of \(O(mn)\).

3. B-SPLINES

A description of the B-spline architecture can be found, for instance, in (Brown and Harris, 1994). B-splines can be functionally equivalent to Mamdani fuzzy models, under certain assumptions. By extending this equivalence, all derivative-based training algorithms (and also the functional approach proposed here) are also applicable to Sugeno and Takagi-Sugeno fuzzy models. For more details, please see (Ruan et al., 2002).

B-Spline neural networks belong to the class of networks denoted as lattice-based associative memories networks. In order to define a lattice in the input space, vectors of knots must be defined, one for each input axis.

The interior knots, for the \(i^{th}\) input, are \(\lambda_{i,j}, j = 1, \cdots, r_i\). They are arranged in such a way that:

\[
x_{i}^{\min} < \lambda_{i,1} \leq \lambda_{i,2} \leq \cdots \leq \lambda_{i,r_i} < x_{i}^{\max}
\]  

(24) In the context of the previous discussion, the interior knots are the nonlinear parameters. At each extreme of each axis, a set of \(k_i\) exterior knots must be given which satisfy:

\[
\lambda_{i,1} = x_{i}^{\min}, \lambda_{i,r_i} = x_{i}^{\max}
\]  

These exterior knots are required to generate the basis functions that are close to the boundaries. These knots are usually coincident with the extreme of the input axes, or are equidistant. The network input space is \([x_{i}^{\min}, x_{i}^{\max}] \times \cdots \times [x_{i}^{\min}, x_{i}^{\max}]\), and so the exterior knots are only used for defining these basis functions at the extreme of the lattice.

The \(j^{th}\) interval of the \(i^{th}\) input is denoted as \(I_{i,j}\) and is defined as:

\[
I_{i,j} = [\lambda_{i,j-1}, \lambda_{i,j}, \lambda_{i,j+1}], \quad j = r_i + 1
\]  

(26) This way, within the range of the \(i^{th}\) input, there are \(r_i+1\) intervals, which means that there are \(p' = \prod_{i=1}^{n} (r_i + 1)\) cells in an \(n\)-dimensional lattice.

The output of the hidden layer is determined by a set of \(p\) basis functions defined on the \(n\)-dimensional lattice. The shape, size and distribution of the basis functions are characteristics of the particular AMN employed, and the support of each basis function is bounded.

In B-Splines neural networks, the order of the spline implicitly sets the size of the basis functions support and its shape. The univariate B-Spline basis function of order \(k\) has a support, which is \(k\) intervals wide. Hence, each input is assigned to \(k\) basis functions.

The \(j^{th}\) univariate basis function of order \(k\) is denoted \(N_{i,j}^k(x)\), and it is defined by the following relationships:

\[
N_{i,j}^k(x) = \begin{cases} 
\frac{x - \lambda_{i,j}}{\lambda_{i,j+1} - \lambda_{i,j}}, & 0 < x < \lambda_{i,j} \\
\frac{\lambda_{i,j+1} - x}{\lambda_{i,j+1} - \lambda_{i,j}}, & \lambda_{i,j} < x < \lambda_{i,j+1} \\
0, & \text{otherwise}
\end{cases}
\]  

(27) Multivariate basis functions are formed by taking the tensor product of the univariate basis functions. Therefore, each multivariable basis function is formed from the product of \(n\) univariate basis functions, one from each input, and every possible combination of univariate basis function is taken.

\[
N_{i}^k(x) = \prod_{i=1}^{n} N_{i,j}^k(x)
\]  

(28) The number of basis functions of order \(k\) defined on an axis with \(r_i\) interior knots is \(r_i+k\). Therefore, the total number of basis functions for a multivariate B-Spline is:

\[
p' = \prod_{i=1}^{n} (r_i + k)
\]  

(29) The output of an AMN is a linear combination of the outputs of the basis functions:

\[
y = \sum_{i=1}^{p} \varphi_i u_i
\]  

(30) where \(\varphi_i = N_i^k(x), \quad i = 1, \ldots, p\).

In order to apply the functional approach to B-splines, we
must derive a way to efficiently compute (20), (21), and the derivatives of the basis functions, (23).

Before listing these calculations, it should be stressed that, as splines have a compact support, the integral over the domain is computed as a sum of \( p \) integrals, each one computed over each cell.

3.1 Computation of (20)

In order to reduce space, we will consider the inverse of (20). Let us define:

\[
\Phi = \int_{x_{\text{min}}}^{x_{\text{max}}} \phi \phi^T \, dx
\]  

(31)

**Univariate case**

**Constant splines \((k=1)\)**

\[
\Phi_{ij} = \begin{cases} 
0, & i \neq j, i = j \\
\lambda - \lambda_{i,j}, & i = j 
\end{cases}
\]  

(32)

**Triangular splines \((k=2)\)**

\[
\Phi_{ij} = \begin{cases} 
\frac{\lambda - \lambda_{i,j}}{6}, & i = j \\
\frac{\lambda - \lambda_{i,j}}{3}, & i = j \\
0, & \text{otherwise}
\end{cases}
\]  

(33)

**Quadratic splines \((k=3)\)**

\[
\Phi_{ij} = \begin{cases} 
\frac{(\lambda_{i}-\lambda_{j})(\lambda_{i}^{2}+\lambda_{i} \lambda_{j}+\lambda_{j}^{2}-3\lambda_{i} \lambda_{j}+3\lambda_{j}^{2}-\lambda_{i} \lambda_{j}^{2})}{15(\lambda_{i}-\lambda_{j})(\lambda_{i}^{2}-\lambda_{j}^{2})}, & i = j \\
\frac{30(\lambda_{i}-\lambda_{j})^{3}}{(\lambda_{i}-\lambda_{j})^{2}}, & i = j \\
\frac{1}{\lambda_{i}-\lambda_{j}}[\lambda_{i}^{3}-\lambda_{j}^{3}+6\lambda_{i}^{2} \lambda_{j}-9\lambda_{i} \lambda_{j}^{2}], & j = i + 1, i = j + 1
\end{cases}
\]  

(34)

**Multivariate case**

Consider that \( \Phi^{(i)} = \int_{x_{\text{min}}}^{x_{\text{max}}} \phi^{(i)} \phi^{(i)\top} dx \), i.e., is the integral for the \( i \)th dimension, computed with the univariate basis functions for this dimension. If \( \otimes \) denotes the tensor product operation, then

\[
\Phi = \int_{x_{\text{min}}}^{x_{\text{max}}} \phi \phi^T \, dx = \Phi^{(1)} \otimes \cdots \otimes \Phi^{(i)} \otimes \cdots \otimes \Phi^{(n)}
\]  

(35)

3.2 Computation of \( \frac{\partial \Phi}{\partial \nu} \)

**Univariate case**

Using the recurrence formulae (27), we have:

\[
\frac{\partial N_{i}^{1}(\nu)}{\partial \lambda_{i}} = \frac{\partial}{\partial \lambda_{i}} \left( \sum_{j=1}^{k} N_{i,j}^{1}(\nu) \right) = \sum_{j=1}^{k} \frac{\partial N_{i,j}^{1}(\nu)}{\partial \lambda_{i}}
\]  

(36)

**Multivariate case**

As every knot is associated with only one dimension, we have:

\[
\frac{\partial}{\partial \lambda_{i,j}} \Phi = \Phi^{(i)} \otimes \cdots \otimes \Phi^{(k-1)} \otimes \frac{\partial \Phi^{(k)}}{\partial \lambda_{i,j}} \otimes \Phi^{(k+1)} \otimes \cdots \otimes \Phi^{(n)}
\]  

(37)

4. EXAMPLE

For this example triangular splines (order 2), with 2 internal knots will be considered. The nonlinear parameters are \( \nu = [\lambda_{1}, \lambda_{2}] \). There are 4 linear weights, associated with the 4 basis functions For this model type, the function to be approximated will be \( f(x) = x^2 \), i.e, we shall be approximating a 3rd order spline, with no internal knots, by a 2nd order B-spline, with 2 interior knots. The domain considered will be \( x \in [-1,1] \).

This example is chosen as it can be completely solved analytically using the equations in Section 2. Fig. 1 shows the optimal weights (12), as function of \( \nu \).

The gradient (19) is given by:

\[
\mathbf{g}_{\nu} = \frac{1}{6(3+5\lambda_{1}-\lambda_{2})}\left[(\lambda_{1}-2\lambda_{2})(3+5\lambda_{1}-3(\lambda_{1}-1)\lambda_{2})\right]
\]  

(41)
Eq (41) is null for the point \( \hat{\lambda}_{12} = [-1/3, 1/3]^T \). Actually, if \( x_{\text{min}} \) and \( x_{\text{max}} \) are also parameters, the global optimum is located at
\[
\hat{\lambda} = \left[ \begin{array}{c} 2x_{\text{min}} - x_{\text{MAX}} \\ x_{\text{MIN}} + 2x_{\text{MAX}} \end{array} \right] / 3.
\]
For the case at hand, the optimal value of the integral of the squared error (14), has the value \( \hat{\Psi} = 4/3645 \approx 1.097 e^{-3} \).

For this particular case, we have a total analytical solution. When this is not possible (i.e., an analytically solution for \( g_{\Psi} = 0 \) is not known), the BP algorithm can be used.

Fig. 2 shows the performance surface of \( \Psi_f(\lambda, \lambda_f) \), when the true function is used, together with four different evolutions of BP (18), starting from different initial points. In order to differ this from the case where (22) and (23) are approximated using sampled data, we shall denote the performance surface using the true function as \( \Psi_f(\lambda, \lambda_f) \), and the performance surface using a numerical integration method as \( \Psi_f(\lambda, \lambda_f) \). In all simulations, the learning rate \( \eta \) is set to 1. Training stops when the maximum number of 200 iterations is reached, or if the following set of termination criteria is satisfied:
\[
\begin{align*}
\Psi[k] &- \Psi[k-1] < \beta[k] \\
\|v[k] - v[k-1]\| &< \sqrt{\tau(1 + \|v[k]\|)} \\
\|v[k]\| &< \sqrt{\tau(1 + \|v[k]\|)}
\end{align*}
\]

Where \( \beta[k] = \tau (1 + \|v[k]\|) \) and \( \tau \) is a measure of the desired number of correct digits in the objective function. This is a criterion typically used in unconstrained optimization (Gill et al., 1981).

As it can be seen, the performance surface is a nice, smooth function with just 1 (global) minimum, as pointed out before. Some statistics related with the four different trainings can be found in Table 1.

As it can be seen the first three trainings reach the optimum (with \( \tau = 1 e^{-8} \)) with a number of iterations smaller than the designated maximum (200). The fourth run will also converge to the global minimum, but will need a few more iterations.

Let us now assume that the training function was unknown and only data was available. Using 8 sample points employed by Gaussian quadrature within the range \([-1,1]\), BP was applied to minimize (2), where the gradient is obtained using (15). This is the standard approach. Its performance surface, together with the evolution of the 4 different trainings, is presented in Fig. 3.

As it can be seen, each one of the 4 different trainings arrives at a different local minimum. These are presented in the last line of Table 3. The third line in this table shows the value obtained for \( \Psi_f(14) \), i.e., evaluated with the true function,
for the local minimum achieved in each training. Clearly, this is the most important objective in terms of the training, as we want to obtain the best approximation to the function underlying the data, and not to the data itself.

<table>
<thead>
<tr>
<th>$\Psi_{r}$ $\Psi_{d}$</th>
<th>$\Psi_{r}$ $\Psi_{d}$</th>
<th>$\Psi_{r}$ $\Psi_{d}$</th>
<th>$\Psi_{r}$ $\Psi_{d}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.394 e-3</td>
<td>2.120 e-3</td>
<td>2.838 e-3</td>
<td>6.297 e-3</td>
</tr>
<tr>
<td>2.282 e-3</td>
<td>2.282 e-3</td>
<td>1.986 e-3</td>
<td>3.013 e-3</td>
</tr>
<tr>
<td>8.953 e-3</td>
<td>8.953 e-3</td>
<td>5.891 e-3</td>
<td>1.112 e-2</td>
</tr>
</tbody>
</table>

With the data only, and being unfamiliar with the true function, the best we can do is to minimize (14), using in the computation of the gradient (17), a numerical integration method for approximating (22) and (23). In this example a Gaussian quadrature method will be employed. We shall denote the performance surface obtained with this approach as $\Psi_{r}$. The latter is shown in Fig. 4, together with the evolution of the trainings, for the same initial points.

The latter is shown in Fig. 4, together with the evolution of the trainings, for the same initial points.

| $V[1]$ $V[n]$ $\Psi_{r}$ $\Psi_{d}$ $d$ $a$ | $V[1]$ $V[n]$ $\Psi_{r}$ $\Psi_{d}$ $d$ $a$ |
|----------------|----------------|----------------|----------------|
| [0.04] [0.04] | [-0.116 0.447] | 2.108 e-3      | 1.018 e-3      |
| [0.20] [0.159] | [-0.405 0.159] | 1.108 e-3      | 1.108 e-3      |
| [-0.6 0.4]     | [-0.578 0.311] | 3.694 e-4      | 3.694 e-4      |
| [-0.6 -0.5]    | [-0.028 0.943] | 2.538 e-3      | 2.538 e-3      |

Analyzing the 3 Tables, it should first be underlined that the line in bold denotes the objective function that is actually minimized. For this reason, the values in the $\Psi_{r}$ line of Table 3 are smaller than the corresponding line in Table 1, where $\Psi_{r}$ is minimized. For 3 out of the 4 initial points, the trainings using $\Psi_{r}$ arrive to the global minimum of $\Psi_{d}$. The same does not happen in any case for $\Psi_{d}$. In fact, comparing the two last lines in Tables 2 and 3, the functional approach achieves, in 3 out of the 4 cases, better values for $\Psi_{d}$, although minimizing $\Psi_{r}$. As pointed out earlier, this is due to the fact that the functional approach achieves a better approximation to the function underlying the data, than the discrete approach.

6. CONCLUSIONS

In this paper we have applied the functional approach, to the training of B-splines. It has the following advantages:

i) If the training criterion exploits the separability of the parameters, it reduces the computational complexity in the calculation of the gradient, as some terms are independent of the data (and therefore its size), and avoids the inverse operation (or pseudo-inverse) in the computation of the optimal linear parameters;

ii) As some of the terms involved in the calculation of the gradient are only dependent on the limits of the input domain and are independent of the training data, the gradient uses a performance surface which is closer to the one obtained if the true function was actually used. The degree of similarity depends on the error obtained in approximating the projection of the function on the basis functions and on their partial derivatives, over the domain, with the training data.

iii) If the function generating the data is known, the local minima and the performance of a specified model can be determined. In the case where integrals (22) and (23) and the gradient zeros can be analytically computed, an analytical solution is obtained, as shown in the example. In other cases, a numerical solution can be found.

For lack of space, it was not shown how to apply the Levenberg-Marquardt algorithm using the proposed approach, for B-splines. The proposed approach can be applied to all kinds of models where parameter separability can be applied. Future publications will address this issue.

REFERENCES


