Fitting 3D data points by extending the neural networks paradigm

A. Iglesias, A. Gálvez
Dept. of Applied Mathematics and Computational Sciences,
University of Cantabria, Spain

Abstract

This paper describes a new method to fit 3D data points from engineering environments (for example, from a numerical-controlled machine or by digitizing a real model). Since these points are subjected to measurement errors, approximation techniques are required. Among them, several techniques based on neural networks have been extensively applied in the last recent years. However, for data points from a B-spline surface the approximation problem could not be well described in terms of neural networks. To overcome this limitation, this paper introduces a new extension of the neural networks paradigm, the functional networks, in which the weights are replaced by neural functions with a multivariate character. In addition, these neural functions can be different for different neurons. The performance of this new approach has been illustrated by its application to fit a given set of data from a tensor product B-spline surface, by large, the most common family of surfaces in industrial design. The analysis of the errors allows us to determine the degree and the coefficients of the polynomial surface that fits the data points better. Finally, we use two sets of data (the training and the testing data) to check for overfitting. The obtained results show that this does not occur here and that all these new features of functional networks allow us to solve outstanding industrial problems.

1 Introduction

In many engineering environments (car bodies, aircraft parts, ship building, shoes, medical imaging, etc.) free-form and sculptured surfaces are frequently built from a set of measured 3D data points. These points (often obtained by digitizing a real model or from a numerical-controlled machine)
are subjected, in general, to measurement errors, thus requiring approximation techniques to be fitted.

A number of different methods for approximation have been described in the literature (see [1, 2] and references therein). Some of them are based on Artificial Intelligence techniques, such as the Artificial Neural Networks. However, not any approximation problem is adequately described in terms of a neural network. The following example (with vectors denoted in bold) illustrates this situation:

Suppose that we look for the most general family of parametric surfaces \( P(s,t) \), such that their isoparametric curves \( s = s_0 \) and \( t = t_0 \) (see Farin [3] for a description) are linear combinations of the sets of functions: \( f(s) = \{ f_0(s), f_1(s), \ldots, f_m(s) \} \) and \( f^*(t) = \{ f_0^*(t), f_1^*(t), \ldots, f_n^*(t) \} \) respectively. To be more precise, we look for surfaces \( P(s,t) \) such that they satisfy the system of functional equations

\[
P(s,t) = \sum_{j=0}^{n} \alpha_j(s)f_j^*(t) = \sum_{i=0}^{m} \beta_i(t)f_i(s),
\]

where the sets \( \{ \alpha_j(s); j = 0,1,\ldots,n \} \) and \( \{ \beta_i(t); i = 0,1,\ldots,m \} \) can be assumed, without loss of generality, as sets of linearly independent functions. Note that if they are not, we can rewrite equations in (1) in the same form but with linearly independent sets.

![Figure 1](image-url)

Figure 1: (left) Graphical representation of a functional network for the parametric surface of eqn (1); (right) Functional network associated with eqn (5). It is equivalent to the functional network on the left.
This problem admits the graphical representation given in Figure 1 (left) which, at first sight, looks like a neural network. However, the previous description in terms of neural networks presents some problems. For instance, the neural functions in neural networks are identical, whereas neural functions in our example are different. In addition, the neuron outputs of neural networks are different; on the contrary, some neuron outputs in the example are coincident (this is the case of the outputs associated with the last layer of neurons leading to the value of \( P(s, t) \)).

These and other disadvantages suggest that the neural networks scheme should be extended. In this paper we propose a new Artificial Intelligence paradigm which generalizes the standard neural networks: the functional networks. They will be used here to fit a given set of data from a B-spline surface by using a polynomial surface whose degree and coefficients are to be determined from the data points.

The structure of this paper is the following: firstly, Section 2 introduces some mathematical concepts and definitions. Section 3 describes the functional networks paradigm. Differences between neural and functional networks will also be discussed in this section. Section 4 gives a general methodology to work with these networks. Finally, the paper closes with the main conclusions of this work.

2 Some mathematical definitions

In this section we give a brief review of the mathematics required in this paper. A more detailed discussion about B-spline curves and surfaces can be found in Piegl and Tiller [4].

Let \( \mathcal{S} = \{s_0, s_1, s_2, \ldots, s_{r-1}, s_r \} \) be a nondecreasing sequence of real numbers called knots. \( \mathcal{S} \) is called the knot vector. The \( i \)th B-spline basis function \( N_{ik}(s) \) of order \( k \) (or degree \( k - 1 \)) is defined by the recurrence relations

\[
N_{i1}(s) = \begin{cases} 
1 & \text{if } s_i \leq s \leq s_{i+1} \\
0 & \text{otherwise} 
\end{cases}
\]  

(2)

and

\[
N_{ik}(s) = \frac{s - s_i}{s_{i+k-1} - s_i} N_{i,k-1}(s) + \frac{s_{i+k} - s}{s_{i+k} - s_{i+1}} N_{i+1,k-1}(s)
\]  

(3)

for \( k > 1 \). With the same notation, given a set of control points \( \{P_{ij}; i = 0, \ldots, m; j = 0, \ldots, n \} \) in a bidirectional net and two knot vectors \( \mathcal{S} = \{s_0, s_1, \ldots, s_r \} \) and \( \mathcal{T} = \{t_0, t_1, \ldots, t_h \} \) with \( r = m + k \) and \( h = n + l \), a B-spline surface \( S(s, t) \) of order \( (k, l) \) is defined by

\[
S(s, t) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} N_{ik}(s) N_{jl}(t),
\]  

(4)

where the \( \{N_{ik}(s)\}_i \) and \( \{N_{jl}(t)\}_j \) are the B-spline basis functions of order \( k \) and \( l \) respectively, defined following eqns (2) and (3).
3 Functional networks

3.1 Components of a functional network

From Figure 1(left) the main components of a functional network become clear:

1. Several layers of storage units. They include a first layer of input units, that contains the input information, the last layer of output units, that contains the output information and a set of intermediate layers of storage units, that are not neurons but units storing intermediate information. This last set is optional and allows connecting more than one neuron output to the same unit. In Figure 1(left) there are two intermediate layers of storage units, which are represented by small circles in black.

2. One or more layers of neurons or computing units. A neuron is a computing unit which evaluates a set of input values, coming from the previous layer, of input or intermediate units, and gives a set of output values to the next layer, of intermediate or output units. Neurons are represented by circles with the name of the corresponding neural function inside. For example, in Figure 1(left), we have three layers of neurons.

3. A set of directed links. They connect the input or intermediate layers to its adjacent layer of neurons, and neurons of one layer to its adjacent intermediate layers or to the output layer. Connections are represented by arrows, indicating the information flow direction.

3.2 Differences between functional and neural networks

Some of the differences between functional and neural networks were already introduced in Section 1. In these paragraphs, we discuss these differences (see Figure 2) and the advantages of using functional networks instead of standard neural networks. These advantages imply that some problems (e.g., the one introduced in Section 1) require functional networks instead of neural networks for their solution.

In neural networks each artificial neuron receives an input value from the input layer or the neurons in the previous layer. Then it computes a scalar output \( y = f(\sum w_{ik} x_k) \) from a linear combination of the received inputs \( x_1, x_2, \ldots, x_n \) using a set of weights \( w_{ik} \) associated with each of the links and a given scalar function \( f \) (the activation function), which is assumed the same for all neurons (see [5, 6]). Therefore, their neural functions have only one argument. On the contrary, neural functions in functional networks can have several arguments. In a given functional network the weights are replaced by neural functions with a multivariate character. In addition, neural functions can be different for different neurons. Finally, in neural networks the neuron outputs are different, while in functional networks neuron outputs can be coincident. As we shall see, this fact leads to a set of functional equations (see [7] for a survey), which have to be solved.
Figure 2: (a) Neural network, and (b) its equivalent functional network.

4 Working with functional networks

In this section, we describe the functional networks methodology, which is organized, for clarity, into eight different steps. These steps are described by their application to the example introduced in Section 1.

Step 1 (Statement of the problem): Understanding the problem to be solved. This is a crucial step, which has been done in Section 1.

Step 2 (Initial topology): In this step, the topology of the initial functional network is selected. Thus, the system of functional equations (1) leads to the functional network in Figure 1(left).

Step 3 (Simplification): Two functional networks are said to be equivalent if they have the same input and output units and they give the same output for any given input. The practical importance of this concept is that we can define equivalent classes of functional networks and then choose the simplest in each class to be used in applications. This is done in this step by using functional equations [7]. In fact, the most general family of solutions for eqn (1) is of the form

\[ P(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{P}_{ij} f_i(s) f_j^*(t) = f(s).\mathbf{P}.(f^*(t))^{T}, \]  

where \((.)^{T}\) indicates the transpose of a matrix and \(\mathbf{P}_{ij}\) are elements of an arbitrary matrix \(\mathbf{P}\); therefore, \(\mathbf{P}(s,t)\) is a tensor product surface. This equation also shows that the functional network in Figure 1(right) is equivalent to the functional network in Figure 1(left).

Step 4 (Uniqueness of representation): Here, conditions for the neural func-
Computational Methods and Experimental Measures

Figures of the simplified functional network must be obtained. For eqn (5), two cases must be considered:

**Case 1**: The $f_i(s)$ and $f_j(t)$ functions are given: Assume that there are two matrices $P = \{P_{ij}\}$ and $P^* = \{P^*_{ij}\}$ such that

$$P(s, t) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} f_i(s)f_j^*(t) = \sum_{i=0}^{m} \sum_{j=0}^{n} P^*_{ij} f_i(s)f_j^*(t). \quad (6)$$

To solve this equation, we write eqn (6) in the form

$$\sum_{i=0}^{m} \sum_{j=0}^{n} (P_{ij} - P^*_{ij}) f_i(s)f_j^*(t) = 0. \quad (7)$$

Since the functions in the set $\{f_i(s), f_j^*(t) \mid i = 0, 1, \ldots, m; j = 0, 1, \ldots, n\}$ are linearly independent, from eqn (7) we have $P_{ij} = P^*_{ij}; i = 0, 1, \ldots, m; j = 0, 1, \ldots, n$, that is, the coefficients $P_{ij}$ in eqn (5) are unique.

**Case 2**: The $f_i(s)$ and $f_j^*(t)$ functions are to be learned: In this case, assume that there are two sets of functions $\{f_i(s), f_j^*(t)\}$ and $\{\tilde{f}_i(s), \tilde{f}_j^*(t)\}$, and two matrices $P$ and $\tilde{P}$ such that

$$P(s, t) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} f_i(s)f_j^*(t) = \sum_{i=0}^{m} \sum_{j=0}^{n} \tilde{P}_{ij} \tilde{f}_i(s)\tilde{f}_j^*(t). \quad (8)$$

Then we have

$$\sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} f_i(s)f_j^*(t) - \sum_{i=0}^{m} \sum_{j=0}^{n} \tilde{P}_{ij} \tilde{f}_i(s)\tilde{f}_j^*(t) = 0. \quad (9)$$

The solution satisfies (see [8])

$$\begin{pmatrix}
\sum_{i=0}^{m} P_{i0} f_i(s) \\
\sum_{i=0}^{m} P_{i1} f_i(s) \\
\vdots \\
\sum_{i=0}^{m} P_{im} f_i(s) \\
\sum_{i=0}^{m} \tilde{P}_{i0} \tilde{f}_i(s) \\
\sum_{i=0}^{m} \tilde{P}_{i1} \tilde{f}_i(s) \\
\vdots \\
\sum_{i=0}^{m} \tilde{P}_{im} \tilde{f}_i(s)
\end{pmatrix} = \begin{pmatrix}
P^T \\
B
\end{pmatrix} f^T(s) \quad ; \quad \begin{pmatrix}
f_0^*(t) \\
f_1^*(t) \\
\vdots \\
f_n^*(t) \\
-f_0^*(t) \\
-f_1^*(t) \\
\vdots \\
-f_n^*(t)
\end{pmatrix} = \begin{pmatrix}
I \\
C
\end{pmatrix} (f^*(t))^T,$n)}

(10)
From eqn (10) we get

\[ \bar{P}^T \bar{f}^T(s) = Bf^T(s) \quad ; \quad (\bar{f}^*(t))^T = -C (f^*(t))^T, \]

Expression (12) gives the relations between both equivalent solutions and the degrees of freedom we have. However, if we have to learn \( f(s) \) and \( f^*(t) \) we can approximate them as:

\[ f(s) = \phi(s) B \quad ; \quad f^*(t) = \psi(t) C, \]

and we get

\[ P(s, t) = f(s).P.(f^*(t))^T = \phi(s).B.P.C^T.\psi(t)^T = \phi(s).\bar{P}.\psi(t)^T, \]

which is equivalent to eqn (5) but with functions \( \{ \phi(s), \psi(t) \} \) instead of \( \{ f(s), f^*(t) \} \). Thus, this case reduces to the first one.

**Step 5 (Data collection):** To describe how functional networks work we need some data. To this aim, we have selected a set of 256 data points \( \{ T_{pq} ; p, q = 1, \ldots, 16 \} \) (in the following, the training points) in a regular 16 × 16 grid from a B-spline surface with \( m = n = 5, k = l = 3 \) (see eqn (4)) and periodic and nonperiodic knot vectors for \( s \) and \( t \) respectively (see ref. [4]). In order to check the robustness of the proposed method, the third coordinate of the 256 three-dimensional points \( (x_k, y_k, z_k) \) was slightly modified by adding a real uniform random variable \( \epsilon_k \) of mean 0 and variance 0.05, that plays the role of a measure error to be used in the estimation step.

**Step 6 (Learning):** At this point, the neural functions are estimated (learned), by using some minimization method. Our learning process consists of approximating each neural function \( f_i \) by a linear combination of functions in a given family \( \{ \phi_{i1}, \ldots, \phi_{im_i} \} \). Thus, the approximated neural function \( \hat{f}_i(x) \) becomes

\[ \hat{f}_i(x) = \sum_{j=1}^{m_i} a_{ij} \phi_{ij}(x), \]

where \( x \) are the inputs associated with the \( i \)-th neuron. In the case of our problem, the problem of learning the above functional network reduces to estimate the neuron functions \( x(s, t), y(s, t) \) and \( z(s, t) \) from a given sequence of triplets \( \{ (x_k, y_k, z_k), k = 1, \ldots, 256 \} \) which depend on \( s \) and \( t \) so that \( x(s_k, t_k) = x_k \) and so on. To this aim we build the sum of squared errors function:

\[ Q_\alpha = \sum_{k=1}^{256} \left( \alpha_k - \sum_{i=1}^{I} \sum_{j=1}^{J} a_{ij} \phi_i(s_k) \psi_j(t_k) \right)^2, \]
where, in the present example, we should consider an error function for each variable $x$, $y$ and $z$. However, since we have just introduced a measure error into the $z$ coordinate, eqn (16) must be interpreted as an equation for $\alpha = z$ only.

The optimum value is obtained when

$$\frac{\partial Q_\alpha}{\partial \alpha_{rs}} = \sum_{k=1}^{256} \left( \alpha_k - \sum_i \sum_j a_{ij} \phi_i(s_k) \psi_j(t_k) \right) \phi^*_r(s_k) \psi^*_s(t_k) = 0$$

(17)

To fit the 256 data points of our example, we have used monomials in $s$ and $t$ variables for the functions \{\phi_i(s) = s^i | i = 0,1,...,I\} and \{\psi_j(t) = t^j | j = 0,1,...,J\} in eqn (16). Of course, every different choice for $I$ and $J$ yields to the corresponding system (17), which must be solved. In particular, we have taken values for $I$ and $J$ from 2 to 6. A simple visual inspection from the data reveals that unit values for $I$ and/or $J$ are not adequate; on the other hand, degrees larger than 6 may lead to numerical round-off errors and are not widely used in industry.

Table 1. Mean, maximum and root mean squared errors of the 256 training points for different values of $I$ and $J$.

<table>
<thead>
<tr>
<th></th>
<th>$J = 2$</th>
<th>$J = 3$</th>
<th>$J = 4$</th>
<th>$J = 5$</th>
<th>$J = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I = 2$</td>
<td>1.510</td>
<td>1.078</td>
<td>1.083</td>
<td>0.948</td>
<td>0.932</td>
</tr>
<tr>
<td></td>
<td>0.351</td>
<td>0.252</td>
<td>0.242</td>
<td>0.235</td>
<td>0.232</td>
</tr>
<tr>
<td></td>
<td>0.0610</td>
<td>0.0450</td>
<td>0.0441</td>
<td>0.0426</td>
<td>0.0425</td>
</tr>
<tr>
<td>$I = 3$</td>
<td>1.207</td>
<td>0.734</td>
<td>0.741</td>
<td>0.595</td>
<td>0.608</td>
</tr>
<tr>
<td></td>
<td>0.315</td>
<td>0.200</td>
<td>0.187</td>
<td>0.172</td>
<td>0.167</td>
</tr>
<tr>
<td></td>
<td>0.0535</td>
<td>0.0336</td>
<td>0.0324</td>
<td>0.0304</td>
<td>0.0302</td>
</tr>
<tr>
<td>$I = 4$</td>
<td>1.216</td>
<td>0.745</td>
<td>0.756</td>
<td>0.615</td>
<td>0.596</td>
</tr>
<tr>
<td></td>
<td>0.307</td>
<td>0.187</td>
<td>0.179</td>
<td>0.161</td>
<td>0.158</td>
</tr>
<tr>
<td></td>
<td>0.0524</td>
<td>0.0320</td>
<td>0.0307</td>
<td>0.0280</td>
<td>0.0280</td>
</tr>
<tr>
<td>$I = 5$</td>
<td>0.948</td>
<td>0.401</td>
<td>0.404</td>
<td>0.273</td>
<td>0.299</td>
</tr>
<tr>
<td></td>
<td>0.276</td>
<td>0.120</td>
<td>0.106</td>
<td>0.071</td>
<td>0.065</td>
</tr>
<tr>
<td></td>
<td>0.0461</td>
<td>0.0192</td>
<td>0.0176</td>
<td>0.0123</td>
<td>0.0116</td>
</tr>
<tr>
<td>$I = 6$</td>
<td>0.938</td>
<td>0.387</td>
<td>0.392</td>
<td>0.246</td>
<td>0.248</td>
</tr>
<tr>
<td></td>
<td>0.275</td>
<td>0.110</td>
<td>0.095</td>
<td>0.062</td>
<td>0.0563</td>
</tr>
<tr>
<td></td>
<td>0.0462</td>
<td>0.0187</td>
<td>0.0163</td>
<td>0.0102</td>
<td>0.0100</td>
</tr>
</tbody>
</table>

**Step 7 (Model validation):** At this step, a test for quality and/or the cross validation of the model is performed. Checking the obtained error is important to see whether or not the selected family of approximating functions is adequate. A cross validation of the model is also convenient. To cross validate the model:
1. We have calculated the mean, the maximum and the root mean squared (RMS) errors, for the 256 training data points. The obtained results for the different values of $I$ and $J$ are reported in Table 1. In this case, complexity of the shape is reflected in the fact that the best approximation was obtained for the highest value we allow: $I = J = 6$. For this value, the mean and the RMS errors are 0.0563 and 0.0100, respectively. Since errors are small, the selected approximating bivariate polynomial was considered adequate.

2. We have also used the fitted model to predict a new set of 1681 testing data points, and calculated the mean, the maximum and the root mean squared (RMS) errors, obtaining the results shown in Table 2. Once more, the smallest error is obtained for $I = J = 6$. A comparison between mean and RMS error values for the training and testing data shows that, for this choice, they are comparable. Thus, we can conclude that no overfitting occurs. Note that a variance for the training data significantly smaller than the variance for the testing data is a clear indication of overfitting. This does not occur here.

Table 2. Mean, maximum and root mean squared errors of the 1681 testing points for different values of $I$ and $J$.

<table>
<thead>
<tr>
<th></th>
<th>$J = 2$</th>
<th>$J = 3$</th>
<th>$J = 4$</th>
<th>$J = 5$</th>
<th>$J = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I = 2$</td>
<td>1.532</td>
<td>1.078</td>
<td>1.084</td>
<td>0.959</td>
<td>0.933</td>
</tr>
<tr>
<td></td>
<td>0.354</td>
<td>0.256</td>
<td>0.249</td>
<td>0.236</td>
<td>0.235</td>
</tr>
<tr>
<td></td>
<td>0.0244</td>
<td>0.0206</td>
<td>0.0205</td>
<td>0.0200</td>
<td>0.0200</td>
</tr>
<tr>
<td>$I = 3$</td>
<td>1.238</td>
<td>0.746</td>
<td>0.742</td>
<td>0.673</td>
<td>0.654</td>
</tr>
<tr>
<td></td>
<td>0.317</td>
<td>0.203</td>
<td>0.196</td>
<td>0.174</td>
<td>0.172</td>
</tr>
<tr>
<td></td>
<td>0.0227</td>
<td>0.0181</td>
<td>0.0179</td>
<td>0.0172</td>
<td>0.0172</td>
</tr>
<tr>
<td>$I = 4$</td>
<td>1.261</td>
<td>1.025</td>
<td>1.022</td>
<td>1.071</td>
<td>1.074</td>
</tr>
<tr>
<td></td>
<td>0.324</td>
<td>0.196</td>
<td>0.187</td>
<td>0.169</td>
<td>0.164</td>
</tr>
<tr>
<td></td>
<td>0.0228</td>
<td>0.0182</td>
<td>0.0179</td>
<td>0.0173</td>
<td>0.0093</td>
</tr>
<tr>
<td>$I = 5$</td>
<td>1.122</td>
<td>0.632</td>
<td>0.526</td>
<td>0.306</td>
<td>0.353</td>
</tr>
<tr>
<td></td>
<td>0.280</td>
<td>0.127</td>
<td>0.113</td>
<td>0.074</td>
<td>0.072</td>
</tr>
<tr>
<td></td>
<td>0.0212</td>
<td>0.0149</td>
<td>0.0145</td>
<td>0.0130</td>
<td>0.0061</td>
</tr>
<tr>
<td>$I = 6$</td>
<td>1.134</td>
<td>0.581</td>
<td>0.724</td>
<td>0.772</td>
<td>0.534</td>
</tr>
<tr>
<td></td>
<td>0.286</td>
<td>0.124</td>
<td>0.111</td>
<td>0.070</td>
<td>0.0691</td>
</tr>
<tr>
<td></td>
<td>0.0213</td>
<td>0.0150</td>
<td>0.0148</td>
<td>0.0085</td>
<td>0.0083</td>
</tr>
</tbody>
</table>

Step 8 (Use of the model): Once the model has been satisfactorily validated, it is ready to be used in predicting new points on the surface.

5 Conclusions

This paper introduces an example of a problem which cannot be solved through the standard neural networks. This is the motivation to introduce
the functional networks, a new extension of the neural networks paradigm in which the weights are replaced by neural functions with a multivariate character. In addition, these neural functions can be different for different neurons. The performance of this new approach has been illustrated by its application to fit a given set of data from a B-spline surface. The analysis of the errors allows us to determine the degree and the coefficients of the polynomial surface that fits the data points better. Results (including the analysis for overfitting) show that this new proposal is more powerful than the one based on neural networks and that can be successfully applied to several other interesting problems.

Acknowledgements

Authors would like to acknowledge the CICYT of the Spanish Ministry of Education (project PB98-0640) and the European Funds FEDER (Contract 1FD97-0409) for partial support of this work.

References


