Neural Splines: Exploiting Parallelism for Function Approximation Using Modular Neural Networks

I.G.Tsoulos[1], I.E.Lagaris[1] and A.Likas[1]

[1] Department of Computer Science, University of Ioannina
Ioannina - GREECE

Abstract
We introduce the *Neural Spline*, that is a mathematical model built by combining a neural network and an associated Obreshkov polynomial. The neural spline has finite support and can be used as the basic element in constructing continuous modular neural-based models. These models are suitable for function approximation in partitioned domains and are also amenable to efficient parallel or distributed implementation. Experimental results are presented for test problems in one and two dimensions which illustrate the effectiveness of the proposed function approximation scheme.
Keywords: Modular neural networks, global optimization, neural splines, parallel computation, function approximation.

1. INTRODUCTION

Piecewise continuous polynomials are well established tools for approximation and interpolation in partitioned domains. As examples we refer to the Natural splines, to B-splines and to Hermite splines [1]. In this article we present a new approximation technique based on domain partitioning, where instead of polynomials we introduce Neural Networks as the basic approximation element, obtaining so a "Neural Spline" approximation scheme. Other non-polynomial splines have been developed in the past, for instance we mention the "Tension Splines" that are based on the exponential function [2]. Feedforward neural networks are well known for their universal approximation capabilities [3, 4] and have been employed for interpolation, approximation and modeling tasks in many cases, such as pattern recognition [5], signal processing, control and the solution of ordinary and partial differential equations [6, 7, 8].
Partitioning a large domain into smaller ones, has the obvious advantage of the reduced problem size and the disadvantage of the increased number of subproblems. However, there are more points to consider. It is not clear if partitioning is always worthwhile, since in many cases is being accompanied by computational overhead, *border discontinuities* and increased complexity. On the other hand, a serious problem with non-partitioned domains is that, usually larger models are required and,

since non-linear optimization is often the only method of choice for training, the resulting objective function usually possesses a large number of local minima. This fact corresponds to excessive computational load that diminishes the efficiency of any method. Note also that partitioning schemes may profit dramatically from parallel processing if formulated properly. Taking all the above into account, we developed a spline-like method using neural networks that is based on domain partitioning and manages to cope with the above mentioned difficulties. In addition, the proposed method is cast in a suitable form so as to benefit when executed on parallel multiprocessor machines or on distributed systems.

In section 2 we state the problem and we provide the general idea of our approach. In section 3 the Obreshkov polynomials are presented along with the neural-splines and the detailed model description. In section 4 we dwell on the one-dimensional case, and describe both the partitioning scheme and the followed algorithm. We also present results of numerical experiments and draw some conclusions. Similarly, in section 5 we present the relevant procedures for the two-dimensional case and report the obtained results.

2. GENERAL DESCRIPTION OF THE APPROXIMATION MODEL

We consider the classical data fitting problem:

Given M points and associated values $(x_i, y_i),\ i = 1, 2, ..., M,$ *with* $x_i \in R^{(N)}$, $y_i \in R$, *draw a smooth hypersurface, that is optimal in the least squares sense.*

The traditional way of solving the above is to assume a parametric model $\Psi(x, p)$ for the solution, and adjust the parameters p, so as to minimize the least squares "Total Error": $E_T[p] = \sum_{i=1}^{M} [\Psi(x_i, p) - y_i]^2$.

In this article we assume that the domain D containing the points x_i, is an N-dimensional rectangular hyperbox and we proceed by first partitioning D into a number of non-overlapping rectangular subdomains D_i. Let ∂D_i denote the boundary of subdomain D_i. In each subdomain D_i the solution is represented by a proper model $\psi_i(x, p^i, q^i)$ (with parameters p^i) that is constructed in such a way so as to meet certain conditions on the boundary ∂D_i, imposed by continuity requirements. These boundary conditions depend on the additional parameters denoted by q^i but are independent of p^i. In other words, the parameters q^i are additional parameters used to ensure that the global solution model $\Psi(x)$, obtained from the combination of the local models ψ_i, will be continuous on every subdomain boundary ∂D_i. If the boundary continuity requirements are not taken into account, then we have the straightforward modular approach, where the individual models are independent and the resulting global solution exhibits discontinuities on every ∂D_i. Such discontinuities may not be

acceptable in several problems and, in addition, they badly influence the accuracy of the obtained solutions, especially in the case of noisy data, as will be demonstrated by numerical experiments.

The global solution model $\Psi(x)$ is defined as

$$\Psi(x,p,q) = \psi_i(x,p^i,q^i),\ \forall\, x \in D_i,\ i = 1,2,\cdots$$

If we define the least squares "local error", i.e. the error for the training points in the subdomain D_i as:

$$E_L[p^i,q^i] = \sum_{x_k \in D_i} [\psi_i(x_k,p^i,q^i) - y_k]^2,\ \forall\, i = 1,2.\ldots \tag{1}$$

then, the total error (for all training points) is given by:

$$E_T[p,q] = \sum_i E_L[p^i,q^i] \tag{2}$$

For each domain D_i the parameters p^i are adjusted by minimizing $E_L[p^i,q^i]$ for a given set of values for q^i. It is obvious that this minimization can be performed independently for each D_i. The additional parameters q^i, are then adjusted so that the complete solution model $\Psi(x,p,q)$ minimizes the "total error" given by equation (2). The above two-stage minimization is repeated until a convergence criterion prevails. A detailed algorithmic description is deferred to section 3. We first develop the necessary terms and notions for one-dimensional problems. Generalization to higher dimensions follows.

3. DEFINITIONS AND TERMS

3.1 Obreshkov polynomials and related operators

Consider an one-dimensional continuously differentiable function $f(x)$, with $x \in [a,b]$, and a polynomial $P_{a,b}^{k,m}(f,x)$ with the following properties:

$$\frac{d^j}{dx^j} P_{a,b}^{k,m}(f,a) = \frac{d^j}{dx^j} f(x)|_{x=a} \equiv f^{(j)}(a), \forall\, j = 0,1,\ldots,k \tag{3}$$

$$\frac{d^j}{dx^j} P_{a,b}^{k,m}(f,b) = \frac{d^j}{dx^j} f(x)|_{x=b} \equiv f^{(j)}(b), \forall\, j = 0,1,\ldots,m \tag{4}$$

Obreshkov [9], found that the unique polynomial of minimal degree $k+m+1$ satisfying the above properties is the following:

$$P_{a,b}^{k,m}(f,x) = \sum_{j=0}^{k} f^{(j)}(a) \frac{(x-b)^{m+1}(x-a)^j}{j!(a-b)^{m+1}} \sum_{i=0}^{k-j} \binom{m+i}{i} \frac{(x-a)^i}{(b-a)^i} +$$
$$\sum_{j=0}^{m} f^{(j)}(b) \frac{(x-a)^{k+1}(x-b)^j}{j!(b-a)^{k+1}} \sum_{i=0}^{m-j} \binom{k+i}{i} \frac{(x-b)^i}{(a-b)^i} \tag{5}$$

We may then define the "Obreshkov operator" $L^{k,m}_{x\in[a,b]}$ applied to function $f(x)$ as follows:

$$L^{k,m}_{x\in[a,b]}f(x) = P^{k,m}_{a,b}(f,x) \tag{6}$$

For $x \in [a,b]$ the *spline-like* function is defined as:

$$S^{k,m}_{a,b}(f,x) \equiv f(x) - P^{k,m}_{a,b}(f,x) = (1 - L^{k,m}_{x\in[a,b]})f(x) \tag{7}$$

The function $S^{k,m}_{a,b}(f,x)$ has the property that at $x = a$, $(x = b)$ vanishes along with all its derivatives up to the k^{th} (m^{th}) order, and hence it behaves like a spline. We also define for $x \in [a,b]$:

$$\begin{aligned} B^{k,m}_{a,b}(f,x) &\equiv f(x) - S^{k,m}_{a,b}(f,x) = (1 - (1 - L^{k,m}_{x\in[a,b]}))f(x) \\ &= L^{k,m}_{x\in[a,b]}f(x) = P^{k,m}_{a,b}(f,x) \end{aligned} \tag{8}$$

Note that $B^{k,m}_{a,b}(f,x)$ resembles f on the boundary, and hence it may be called a "Boundary Match".
In two dimensions where the subdomain becomes $[a_1,b_1]\otimes[a_2,b_2]$ the above definitions may generalize as:

$$S(f,x_1,x_2) \equiv (1 - L^{k_1,m_1}_{x_1\in[a_1,b_1]})(1 - L^{k_2,m_2}_{x_2\in[a_2,b_2]})f(x_1,x_2) \tag{9}$$

and

$$\begin{aligned} B(f,x_1,x_2) &= f(x_1,x_2) - S(f,x_1,x_2) = \\ &= (L^{k_1,m_1}_{x_1\in[a_1,b_1]} + L^{k_2,m_2}_{x_2\in[a_2,b_2]} - L^{k_1,m_1}_{x_1\in[a_1,b_1]}L^{k_2,m_2}_{x_2\in[a_2,b_2]})f(x_1,x_2) \end{aligned} \tag{10}$$

and similarly for higher dimensions. A corresponding generalization for polynomial splines in two dimensions, known as "*Spline–Blended Functions*" has been developed for use in designing surfaces for technical products by [10].

3.2 Neural splines and model description

Let $N(x,p)$ be a neural network model with one input x and weights denoted by p. Then the spline $S(N,x)$ defined using equation (7) is referred to as a Neural Spline in one dimension. In each of the subdomains $D_i = [a_i,b_i]$ we represent our model as:

$$\psi_i(x,p^i,q^i) = B^{k,m}_{D_i}(f,x) + S^{l,n}_{D_i}(N,x) \tag{11}$$

where f is the unknown function to be approximated. The parameters q^i represent the unknown values of $f(x)$ and possibly of its derivatives on the boundary ∂D_i.

The model $\psi_i(x, p^i, q^i)$ so defined, satisfies by construction the following boundary conditions:

$$\frac{d^j}{dx^j}\psi_i(x, p^i, q^i)|_{x=a} = f^{(j)}(a),\ j = 0, ..., \min(k, l) \tag{12}$$

$$\frac{d^j}{dx^j}\psi_i(x, p^i, q^i)|_{x=b} = f^{(j)}(b),\ j = 0, ..., \min(m, n) \tag{13}$$

where $f^{(j)}(a), f^{(j)}(b)$ are parameters constituting the elements of the parameter vector q^i. As an example, in the case $k = l = m = n = 0$, the one-dimensional model is written as:

$$\begin{aligned}\psi(x, p, q) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a} + \\ \{N(x, p) - [N(a, p)\frac{x-b}{a-b} + N(b, p)\frac{x-a}{b-a}]\}\end{aligned} \tag{14}$$

with q referring collectively to $f(a)$ and $f(b)$. As it can readily be verified, this model satisfies $\psi(a, p, q) = f(a)$ and $\psi(b, p, q) = f(b)$. For the case $k = l = m = n = 1$, we have the following one-dimensional model:

$$\begin{aligned}\psi(x, p, q) = & f^{(0)}(a)\pi_{3,0}(x, a, b) + f^{(1)}(a)\pi_{3,1}(x, a, b) \\ & + f^{(0)}(b)\tau_{3,0}(x, a, b) + f^{(1)}(b)\tau_{3,1}(x, a, b) \\ & + \{N(x, p) - [N(a, p)\pi_{3,0}(x, a, b) + N^{(1)}(a, p)\pi_{3,1}(x, a, b) \\ & \quad + N(b, p)\tau_{3,0}(x, a, b) + N^{(1)}(b, p)\tau_{3,1}(x, a, b)]\}\end{aligned}$$

where the following notation is used:

$$\begin{aligned}\pi_{1,0}(x, a, b) &= \frac{x-b}{a-b} \\ \pi_{3,0}(x, a, b) &= \frac{(x-b)^2}{(a-b)^2}\left(1 + 2\frac{x-a}{b-a}\right) \\ \pi_{3,1}(x, a, b) &= (x-a)\frac{(x-b)^2}{(a-b)^2} \\ \tau_{2k+1,j}(x, a, b) &\equiv \pi_{2k+1,j}(x, b, a)\end{aligned} \tag{15}$$

In two dimensions, setting $k_1 = k_2 = m_1 = m_2 = 0$ in equation (10) we obtain:

$$\begin{aligned}B(f, x_1, x_2) = & f(a_1, x_2)\frac{x_1-b_1}{a_1-b_1} + f(b_1, x_2)\frac{x_1-a_1}{b_1-a_1} \\ & + f(x_1, a_2)\frac{x_2-b_2}{a_2-b_2} + f(x_1, b_2)\frac{x_2-a_2}{b_2-a_2} \\ & - \{\,[f(a_1, a_2)\frac{x_1-b_1}{a_1-b_1} + f(b_1, a_2)\frac{x_1-a_1}{b_1-a_1}]\frac{x_2-b_2}{a_2-b_2} \\ & \quad + [f(a_1, b_2)\frac{x_1-b_1}{a_1-b_1} + f(x_1, b_2)\frac{x_1-a_1}{b_1-a_1}]\frac{x_2-a_2}{b_2-a_2}\,\}\end{aligned} \tag{16}$$

and similarly we can define the two-dimensional Neural spline. The definition of the external parameters q_i in two dimensions, which determine the solution on the boundary, is given in section 5.

4. ONE DIMENSIONAL CASE

4.1 Partitioning and procedures

In the one-dimensional case, we proceed by first defining a number of knots t_i, i.e. points that partition the domain of interest D in several non-overlapping subdomains $D_i = [t_i, t_{i+1}]$. Then the following procedure is applied:

1. Introduce the vector of external parameters q^i with elements $f_i^{(0)}, f_i^{(1)}, \cdots, f_i^{(k)}$ that represent values for the solution and for a number of its derivatives at each knot t_i.

2. For $i = 1, 2, \ldots$ use a model $\psi_i(x, p^i, q^i)$ for $x \in D_i$ that satisfies the continuity conditions specified at the two bracketing knots t_i and t_{i+1} and minimize the local least squares error $E_L[p^i, q^i]$ with respect to p^i, keeping the external parameters q^i fixed. Terminate if all errors are below a preset value.

3. Adjust the external parameters q^i so as to minimize the total error $E_T[p, q] = \sum_i E_L[p^i, q^i]$ keeping p^i fixed.

4. Repeat from step 2.

Note that the training of each local model in step 2, can be implemented in parallel, since the local models are being determined independently, given that the external parameters remain fixed. This is not the case for the procedure in step 3, where a change in the external parameters at the knot t_i affects the representation in both the D_{i-1} and the D_i subdomains.
The initialization of the external parameters q^i is extremely important. Far off values, may decelerate the convergence dramatically. Hence, we employ a preprocessing scheme to ensure that the initial values are close to the actual ones. This is achieved by first fitting a single neural network around each knot (using nearby points for training) and then use this network to generate the initial values for the external parameters. It has been observed that only rarely these parameters need to be readjusted if the estimation is sufficiently accurate, which is the case when the data points around the knot are dense.
The model for $x \in D_i$ is taken as: $\psi_i(x, p^i, q^i) = B(f, x) + S(N, x)$. The neural

network model we have used is the multilayer perceptron with one hidden layer of sigmoid hidden units:

$$N(x,p) = \sum_{i=1}^{H} v_i \sigma(w_i x + b_i), \quad \sigma(z) = \frac{1}{1+e^{-z}} \tag{17}$$

Since the local models are mutually independent, there are no propagating errors across the subdomains and hence there is no error accumulation. Global optimization is used for neural network training in each subdomain. This is affordable because the partitioning, renders possible an economic representation, i.e. the employment of a small neural network. Therefore, the adjustable model parameters in each interval are few, hence the existence of different local minima is rather limited. In practice, and in order to accelerate the process, we stop the global search as soon as the local error becomes smaller than a preset threshold. The global optimization procedure used was "Multistart".
We also experimented with more sophisticated methods such as the Multi-level algorithm due to [11], as modified by [12] taking in account the functional topography, without noticing substantial difference in performance. As a local search method, the quasi-Newton BFGS method [13] was employed, as modified in [14] (supports simple bounds and linear constraints).

4.2 Numerical experiments

Experiments were conducted with two types of data sets "Clean" data sets, that are created by directly evaluating some test function at several points on a grid, and "noisy" data sets that are produced from clean data sets through the addition of noise. We present in what follows experiments with the function $f(x) = x\sin(x^2)$ whose plot in the interval $[-4, 4]$ is shown in Fig. 1. We first fit this function via a single neural network, to obtain an indication of the effort required and of the difficulty of the problem. Then we experimented with several ways of partitioning. In each subdomain a neural network was trained independently without paying attention to continuity. We compare this approach with the continuous solutions obtained using our method.
Several cases were examined by varying the number of partitions and the number of hidden units of the neural networks used in each partition. In every experiment we used a rather sparse dataset for training and a dense dataset for testing. We conducted experiments with both randomly chosen and with equidistant training points. When clean data sets are used one can hardly observe any difference. However, with 'noisy' data sets the quality of the fit with our model is clearly superior. Note that these experiments were performed on a distributed multiprocessor system. The number of

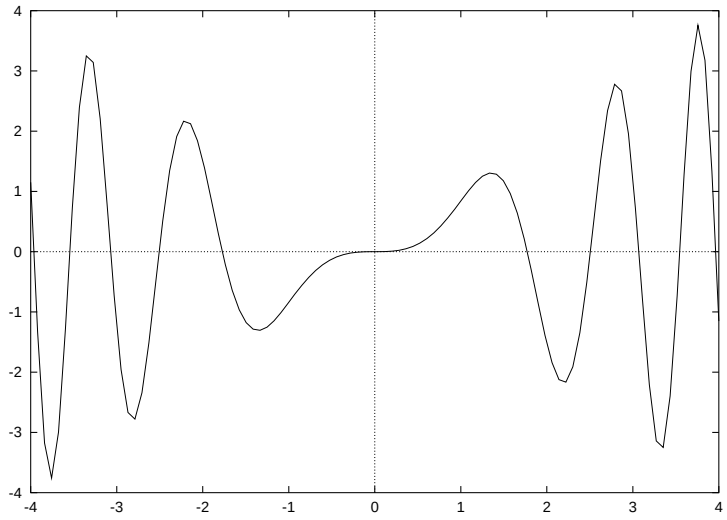

Figure 1: Plot of $xsin(x^2)$

CPUs employed was the same with the number of partitions, i.e.., each partition was treated by a different CPU and our implementation was based on message passing programming (MPI). We compared solution times for several combinations of the number of partitions and the number of the hidden nodes keeping their product at comparable values to avoid overblown model complexity. A solution is taken to be one that reaches a prescribed low value for the mean point-wise error for the training set. For the 'clean' data set with randomly chosen training points (RP), we present in *Table I*, the outcome of the approximation with a single neural network covering the whole domain $x \in [-4, 4]$. The first entry is the number of the hidden units of each network, the second and third entries correspond to the mean square training and test errors, and the last entry is the CPU time in seconds. In *Table II* we present results when independent neural networks are used to approximate the function in each interval. The first entry (NP) is the number of partitions in which the domain $[-4, 4]$ is split and the last entry is the real time to complete the training process. In *Table III* we present results using the neural-spline approach, for the case where only the function value is continuous at the knots, and in *Table IV* for the case where in addition to the function, the first derivative is continuous as well. Similarly we present the corresponding results using the four methods for a noisy data set created by adding 15% white noise to the clean data. For the noisy data set, the reported training error is the mean squared difference of the approximation model from the corrupt data, while the test–error is the mean squared difference of the approximation model from the corresponding function values (without corruption). The corresponding tables are: V, VI, VII and VIII. For the case of equidistant training points (EP), tables IX - XII and XIII - XVI are the corresponding tables for the clean and for the noisy data set respectively.

Table I - Clean Data - RP

Nodes	Train	Test	Time
10	2.9E-5	6.5E-5	13560
15	3.2E-8	4.9E-6	20066
20	5.3E-9	5.5E-6	17200

Single Neural Network

Table II - Clean Data - RP

NP	Nodes	Train	Test	Time
2	10	2.3E-9	4.4E-7	5152
2	15	2.0E-10	9.7E-8	4679
5	5	4.5E-9	8.6E-7	1215
5	10	5.0E-11	1.2E-9	182
10	3	5.1E-8	1.5E-5	831
10	5	5.0E-11	1.9E-7	47

Piecewise Neural Networks

Table III - Clean Data - RP

NP	Nodes	Train	Test	Time
2	10	1.3E-9	1.9E-7	17176
2	15	2.0E-10	1.6E-7	20976
5	5	3.6E-9	9.3E-6	2035
5	10	5.0E-11	6.0E-6	976
10	3	1.5E-11	5.2E-8	525
10	5	5.0E-12	3.1E-8	134

Neural Splines Model, $\Psi \subset C^{(0)}$

Table IV - Clean Data - RP

NP	Nodes	Train	Test	Time
2	10	2.4E-9	1.7E-7	33330
2	15	1.1E-9	2.1E-7	9326
5	5	3.0E-10	5.1E-7	3795
5	10	5.0E-11	4.3E-7	2330
10	3	1.0E-10	5.3E-8	756
10	5	5.0E-11	4.3E-8	202

Neural Splines Model, $\Psi \subset C^{(1)}$

Table V - Noisy Data - RP

Nodes	Train	Test	Time
10	6.8E-3	7.8E-3	13307
15	5.0E-3	6.1E-3	13345
20	5.6E-3	9.4E-3	15406

Single Neural Network

Table VI - Noisy Data - RP

NP	Nodes	Train	Test	Time
2	10	4.5E-3	7.8E-3	6873
2	15	4.1E-3	7.9E-2	10401
5	5	4.8E-3	74.E0	2438
5	10	3.5E-3	1.4E-1	6559
10	3	4.3E-3	9.1E-1	1254
10	5	2.6E-3	5.7E0	2060

Piecewise Neural Networks.

Table VII - Noisy Data - RP

NP	Nodes	Train	Test	Time
2	10	4.6E-3	8.2E-3	18700
2	15	4.3E-3	2.5E-2	20655
5	5	5.2E-3	2.2E-2	4951
5	10	4.0E-3	5.0E-2	8302
10	3	4.5E-3	2.7E-1	2448
10	5	2.5E-3	3.7E-1	3012

Neural Splines Model, $\Psi \subset C^{(0)}$

Table VIII - Noisy Data - RP

NP	Nodes	Train	Test	Time
2	10	1.0E-2	1.3E-2	47572
2	15	8.3E-3	1.3E-2	131208
5	5	5.3E-3	3.0E-1	16252
5	10	4.2E-3	1.1E-1	45861
10	3	4.7E-3	3.5E-2	7167
10	5	2.7E-3	2.6E-1	10357

Neural Splines Model, $\Psi \subset C^{(1)}$

It can be observed that the time $T(n_p)$ needed to perform the calculation when there are n_p partitions and each partition is handled by a differnet CPU, descents faster than n_p^{-1}. This is expected since apart from the gain due to parallelism (a n_p^{-1} factor), there is an additional gain coming from the fact that the optimization problem is simpler when the number of partitions grows. One way of looking at it is by noting that in each partition, the time spent is proportional to the number of the training points belonging to the partition as well as to the number of the function evaluations $N_f(n_p)$ required by the optimization algorithm to train one out of n_p networks.
Namely $T(n_p) \propto \frac{M}{n_p} N_f(n_p)$ where M is the total number of training points and M/n_p is the number of points in each of the n_p partitions. Comparing $T(n_p)$ with $T(1)$ i.e. the time needed when no partitioning is performed, we deduce:

$$\frac{T(n_p)}{T(1)} = \frac{1}{n_p} \frac{N_f(n_p)}{N_f(1)} < \frac{1}{n_p}$$

A rough rule of thumb extracted from our experiments, when on the average the product $n_p \times h(n_p)$ is kept essentially constant, ($h(n_p)$ being the number of hidden nodes when we have n_p partitions), is $T(n_p) \propto n_p^{-4/3}$.

Most data sets in practice are contaminated with noise. Hence this case is important as far as applications are concerned. When noise is present the advantage of the proposed model is evident. Observe in Fig. (2) the discontinuities in the curve corresponding to the piecewise neural networks (PNN) case. Fig. (3) is a blow-up of Fig. (2) around a knot where a discontinuity appears.

Note there the smoother curves produced using the proposed model corresponding to the $C^{(0)}$ and to the $C^{(1)}$ case.

5. TWO-DIMENSIONAL CASE

5.1 Partitioning and preprocessing

In the two-dimensional case, we partition the domain in $K_x \times K_y$ rectangular boxes by dividing the x-range in K_x parts and the y-range in K_y segments accordingly. The model in each box becomes

$$\psi(x, y, p, q) = B(f, x, y) + S(N, x, y) \tag{18}$$

with $S(N, x, y)$ and $B(f, x, y)$ given by equations (9) and (10), with $k_1 = k_2 = m_1 = m_2 = 0$. The neural network $N(x, y, p)$ is a two-input multilayer perceptron with sigmoid hidden nodes and parameter vector p.

Table IX - Clean Data - EP

Nodes	Train	Test	Time
10	2.1E-5	2.1E-5	14844
15	1.1E-7	1.0E-7	21953
20	5.0E-9	5.1E-9	16724

Single Neural Network

Table X - Clean Data - EP

NP	Nodes	Train	Test	Time
2	10	1.6E-9	2.9E-8	7750
2	15	1.5E-10	5.6E-8	5537
5	5	5.7E-9	1.6E-7	3119
5	10	5.0E-11	2.7E-9	550
10	3	5.4E-8	1.6E-7	760
10	5	5.0E-11	6.0E-8	76

Piecewise Neural Networks

Table XI - Clean Data - EP

NP	Nodes	Train	Test	Time
2	10	9.5E-10	1.4E-9	13945
2	15	2.0E-10	2.6E-10	20215
5	5	4.0E-10	4.5E-10	1974
5	10	5.0E-11	7.0E-11	270
10	3	3.0E-10	4.1E-10	445
10	5	5.0E-11	1.0E-10	83

Neural Splines Model, $\Psi \subset C^{(0)}$

Table XII - Clean Data - EP

NP	Nodes	Train	Test	Time
2	10	2.9E-9	4.3E-9	14106
2	15	1.1E-9	2.9E-9	13254
5	5	4.9E-9	1.8E-8	2590
5	10	5.0E-11	1.1E-8	3313
10	3	7.5E-10	2.1E-9	670
10	5	1.5E-10	1.3E-9	144

Neural Splines Model, $\Psi \subset C^{(1)}$

Table XIII - Noisy Data - EP

Nodes	Train	Test	Time
10	1.8E-1	8.3E-2	16319
15	1.6E-1	1.0E-1	17746
20	1.4E-1	1.3E-1	16425

Single Neural Network

Table XIV - Noisy Data - EP

NP	Nodes	Train	Test	Time
2	10	1.3E-1	2.1E-1	6874
2	15	1.1E-1	1.8E-1	10401
5	5	1.4E-1	7.5E+2	2438
5	10	9.2E-2	1.7E+0	6559
10	3	1.3E-1	4.5E+1	1254
10	5	6.7E-2	2.1E+0	2060

Piecewise Neural Networks.

Table XV - Noisy Data - EP

NP	Nodes	Train	Test	Time
2	10	1.3E-1	1.5E-1	18700
2	15	1.2E-1	1.7E-1	20656
5	5	1.5E-1	1.8E-1	4951
5	10	9.7E-2	2.1E-1	8302
10	3	1.4E-1	1.6E-1	2448
10	5	6.7E-2	3.0E-1	3013

Neural Splines Model, $\Psi \subset C^{(0)}$

Table XVI - Noisy Data - EP

NP	Nodes	Train	Test	Time
2	10	1.3E-1	1.9E-1	47572
2	15	1.3E-1	1.8E-1	131208
5	5	1.5E-1	1.8E-1	16252
5	10	1.0E-1	2.1E-1	45861
10	3	1.3E-1	1.8E-1	7167
10	5	7.2E-2	2.4E-1	10358

Neural Splines Model, $\Psi \subset C^{(1)}$

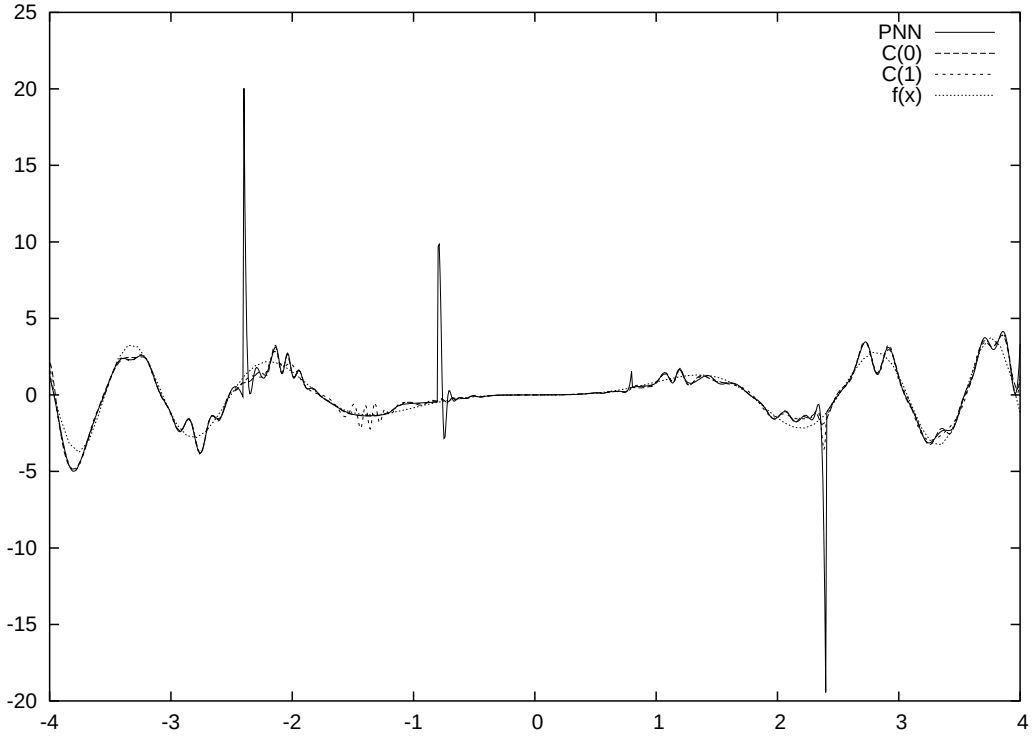

Figure 2: Plots of the approximation obtained using several methods

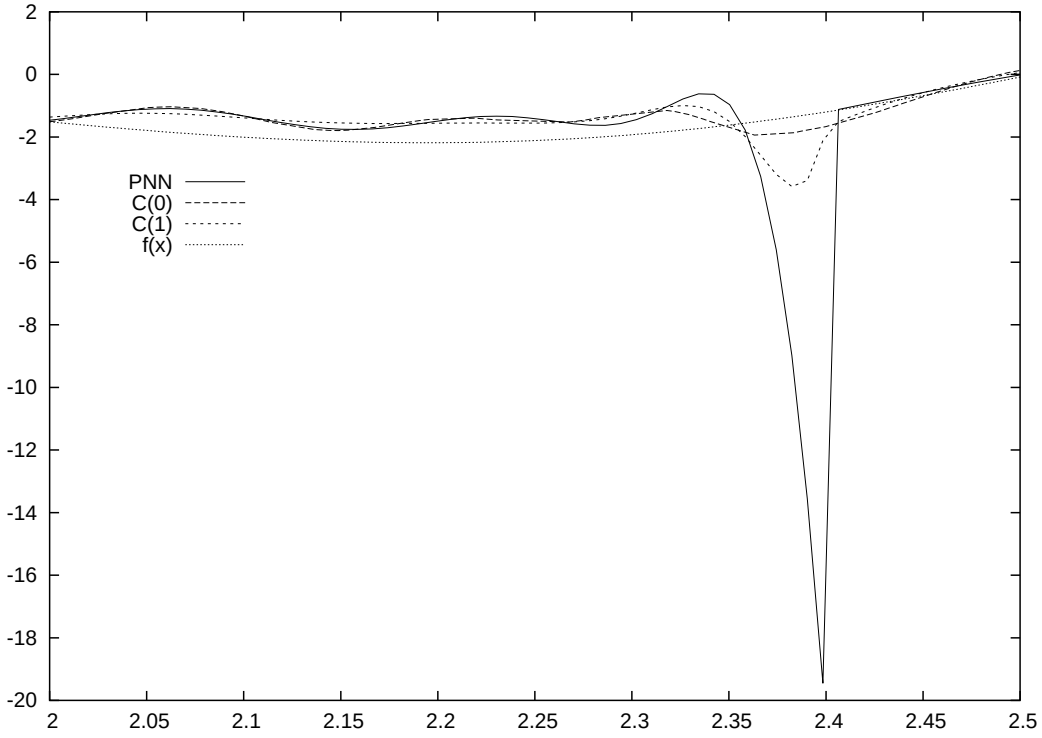

Figure 3: Details of the approximation around a knot

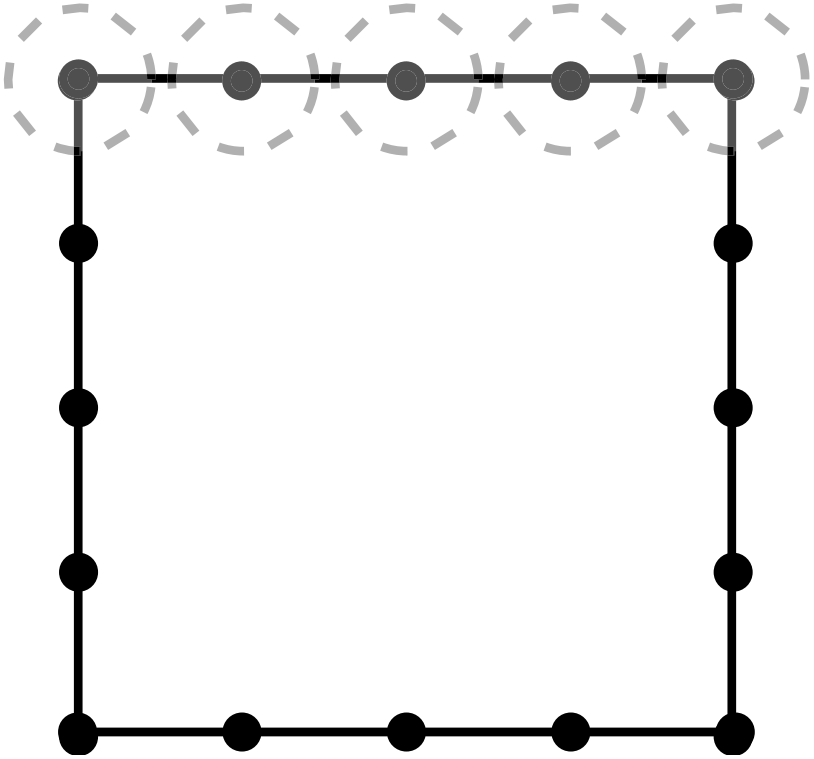

Figure 4: The boundary points of a partition and the top side estimation area

From eq. (16) it is clear that in order to construct the boundary match function B, we need to specify an estimate of the solution along each side of the box. Namely we need four one-dimensional functions to describe the solution at the top, bottom, left and right side of the box. We next describe the procedure that determines the function at the top side of the box. For the other sides one can proceed accordingly. Consider the five boundary points on the top side of the box as displayed in Fig. (4).

- Estimate the values of the function at these points and consider them as external parameters. The estimation is performed by fitting a two–dimensional neural network with training set containing points around each boundary point. The area covered by such training set is indicated by the dotted circles.
- Use a cubic fitting spline [15], to represent the solution on that segment.

Once the boundary match functions are constructed, the model in equation (18) is trained in every box. This is the major step of the method, i.e. the most time–consuming part. Since we keep the boundary matches unchanged, the training in each box is independent and therefore the load can be spread to multiple CPUs, ideally the training in each box is assigned to a different CPU.
One may vary the external parameters (i.e. the estimated function values on the box

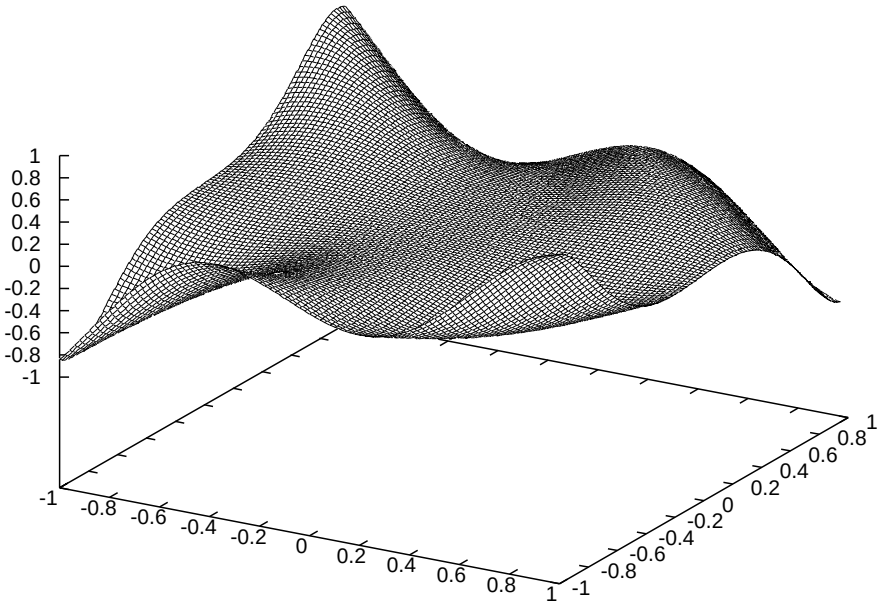

Figure 5: Plot of: $xysin(2y^2 + 3x)$

sides) while keeping the neural spline parameters fixed, and then repeat this step, but we find that in almost all cases this was unnecessary, since the proposed estimation scheme at each side of a box is quite accurate. For $k_1 = k_2 = m_1 = m_2 = 1$, i.e. when the resulting approximation guarantees the continuity of the normal derivative as well, one needs also to define as external parameters the values of the partial derivatives at the boundary points and the model becomes more complicated.

5.2 Numerical Experiments

We conducted experiments for a two-dimensional problem considering the case where only the solution is guaranteed to be continuous across the boundaries, not its partial derivatives. We considered fitting the function

$$f(x, y) = xysin(2y^2 + 3x)$$

a plot of which is displayed in Fig. (5). We made a comparison between piecewise neural networks and the proposed model using 'clean' and 'noisy' data. Again as in the one-dimensional case, when there is no noise in the data the results are almost identical. When we add noise to the data then the proposed model is to be preferred as far as the quality of the approximation is concerned. We present experiments with 10% noise. We use 1600 equidistant training points and 10000 equidistant points for

the test set.

In Tables XVII and XVIII the results for the case of clean uniformly distributed data are displayed for the piecewise neural network model and for the proposed model correspondingly. Similarly in Tables XIX and XX we present the results for randomly sampled noisy data. Column "NP" denotes the number of partitions of the domain, column "Nodes" denotes the number of hidden nodes of each neural network, column "Train" denotes the model's average training error, column "Test" denotes the model's average test error and column "Time" denotes the needed execution time in seconds. Similarly the results for the single neural network case are laid out in tables XXI and XXII. A feature that can be inferred is that the neural-spline model provides generalization performance comparable to a well-trained single neural network, despite the fact that its training is significantly faster (gaining both from the parallel implementation and from the complexity reduction of the optimization subproblem). On the other hand, the piecewise model can be trained in even less time, however its generalization performance is clearly inferior, due to the discontinuities at the boundaries of the partitions.

6. CONCLUSIONS

Modular neural networks constitute an efficient parallel approach for data fitting in large domains. However, until now little or no attention was paid to the model discontinuities appearing at the boundaries of the partitions. Such discontinuities could significantly deteriorate the quality of the solution at regions around a boundary, especially in the case of noisy data. In this work we have elaborated on this issue and presented a methodology that effectively treats this problem and leads to the development of modular but everywhere continuous neural network solutions. The proposed methodology exploits the approximation capabilities of the Multilayer Perceptron and appropriately combines an MLP with Obreshkov polynomials for the definition of a "Neural Spline". Neural Splines have convenient properties that enable the construction of continuous approximation models in a partitioned domain and, in addition, allow for efficient processing in parallel or distributed computational systems. We applied the method for data fitting in one and two dimensions with very encouraging results, especially in the case of noisy data. It is clear that the imposition of continuity constraints acts as a regularization mechanism that prevents overtraining and leads to smooth modular approximation models. In the present work we used non-overlapping Neural Splines. Future work could include an attempt to understand how overlapping Neural Splines can be used for function approximation, or extend the proposed methodology to other problems such as the solution of differential equations.

Table XVII - Clean data - EP

NP	Nodes	Train	Test	Time
4x4	6	2.6E-9	4.5E-9	262
4x4	8	4.6E-11	1.8E-10	413
5x5	6	5.0E-10	3.7E-9	159
5x5	8	2.3E-11	1.5E-9	78

2D Piecewise Neural Networks

Table XVIII - Clean data - EP

NP	Nodes	Train	Test	Time
4x4	6	2.6E-9	1.2E-8	781
4x4	8	1.3E-9	1.7E-7	1321
5x5	6	1.2E-9	1.3E-8	365
5x5	8	8.6E-10	4.5E-8	771

2D Neural Splines Model

Table XXI - Clean data - EP

NODES	Train	Test	TIme
5	4.5E-3	4.3E-3	889
10	1.9E-4	1.7E-4	5455
15	6.6E-6	5.8E-6	13327
20	2.3E-7	2.1E-7	146514

2D Single Neural Network

Table XIX - Noisy data - EP

NP	Nodes	Train	Test	Time
4x4	6	4.6E-4	8.5E-2	79
4x4	8	3.3E-4	9.5E-2	157
5x5	6	3.7E-4	1.4E-2	92
5x5	8	2.6E-4	3.3E-1	144

2D Piecewise Neural Networks

Table XX - Noisy data - EP

NP	Nodes	Train	Test	Time
4x4	6	4.9E-4	5.6E-4	295
4x4	8	4.3E-4	6.9E-4	551
5x5	6	4.8E-4	6.1E-4	234
5x5	8	4.1E-4	3.5E-3	490

2D Neural Splines Model.

Table XXII - Noisy data - EP

NODES	Train	Test	TIme
5	5.1E-3	4.1E-3	972
10	8.4E-4	1.1E-4	3423
15	7.3E-4	7.9E-5	8098
20	6.4E-4	1.3E-4	32880

2D Single Neural Network

ACKNOWLEDGMENTS

Most of the computations were performed on a Beowulf cluster of the Department of Materials Science and we would like to thank Prof. E. Kaxiras and Prof. D. Papageorgiou for granting access to the system and technical assistance. This research has been supported in part via a grant from the University of Ioannina Research Committee.

References

[1] De Boor C., *A practical guide to Splines*, Springer-Verlag, New York 1978.

[2] Kincaid D., and Cheney W., *Numerical Analysis*, Brooks/Cole Publishing Company 1991.

[3] Hornik K., Stinchcombe M., and White H., Neural Networks 2 (1989) 359.

[4] Cybenko G., *Approximation by superpositions of a sigmoidal function*, Mathematics of Control Signals and Systems 2 (1989) 303-314.

[5] Bishop C., *Neural Networks for Pattern recognition*, Oxford University Press, 1995.

[6] Lagaris I. E., Likas A., Fotiadis D. I., *Artificial Neural Networks for solving ordinary and partial differential equations*, IEEE Trans. on Neural Networks, 9 (1998) 987-1000.

[7] Lagaris I. E., Likas A., Fotiadis D. I., *Artificial Neural Network methods in Quantum Mechanics*, Computer Physics Communications, 104 (1997) 1-14.

[8] Lagaris I. E., Likas A., Papageorgiou D. G., *Neural Network methods for boundary value problems with irregular boundaries*, IEEE Trans. on Neural Networks, 11 (2000) 1041-1049.

[9] Obreshkov N., *On the Mechanical Quadratures*, J. Bulgar. Acad. Sci. and Arts LXV-8, (1942) 191-289.

[10] Gordon W. J., *Spline-Blended Surface Interpolation Through Curve Networks*, Journal of Mathematics and Mechanics 18 (1969) 931-952.

[11] Rinnooy Kan A. H. G. and Timmer G. T., *Stochastic Global Optimization Methods. Part II: Multi-level Methods*, Math. Programming 39 (1987) 57-78.

[12] Ali M. M. and Storey C., *Topographical Multilevel Single Linkage*, Journal of Global Optimization 5 (1994) 349-458.

[13] Fletcher R., *A new approach to variable metric algorithms*, Computer Journal 13 (1970) 317-322

[14] M. J. D. Powell, *A Tolerant Algorithm for Linearly Constrained Optimization Calculations*, Mathematical Programming, **45** (1989) 547.

[15] Gisela Engeln-Müllges, Frank Uhlig *Numerical Algorithms with Fortran*, Springer-Verlag, Berlin Heidelberg 1996.