Piecewise Neural Networks for Function Approximation, Cast in a Form Suitable for Parallel Computation

Ioannis G. Tsoulos, Isaac E. Lagaris, and Aristidis C. Likas

Dept. of Computer Science, University of Ioannina,
Ioannina - GREECE 45110

Abstract. We present a technique for function approximation in a partitioned domain. In each of the partitions a form containing a Neural Network is utilized with parameterized boundary conditions. This parameterization renders feasible the parallelization of the computation. Conditions of continuity across the partitions are studied for the function itself and for a number of its derivatives. A comparison is made with traditional methods and the results are reported.

1 Introduction

1.1 Rationale and Motivation

Piecewise continuous polynomials are well established tools for approximation and interpolation. As examples we refer to the Natural splines, to B-splines and to Hermite splines[1]. In this article we present a partitioning technique, where instead of polynomials we introduce Neural Networks as the basic approximation element, obtaining so a scheme that may be referred to as "Neural Splines". 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]. 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, ranging from 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 problems. However there are more points to consider. It is not clear if partitioning is always worthwhile, since in most cases is being accompanied by computational overhead, matching discontinuities and increased complexity. However a serious problem with extended domains is that since non-linear optimization is often the only method of choice, the resulting objective function possesses a large number of useless local minima, a fact that corresponds to excessive computational load that diminishes the efficiency of any method, hence in that respect partitioning has an edge. Note also that partitioning schemes may profit dramatically from parallel processing if formulated properly. Taking all the above into account, we developed a method that uses partitions and manages to cope with the mentioned difficulties and in addition is cast in a suitable form so as to benefit when executed on parallel multiprocessor machines or on a distributed system.

1.2 General Description of the Method

Let us first consider the classical fitting problem:

Given M points and associated values (x_i, y_i), $i = 1, 2, ..., M$, *where the points* $x_i \in R^{(N)}$, *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 consequently 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 x-points, is an N-dimensional rectangular hyperbox and we proceed by first partitioning it in several non-overlapping rectangular subdomains D_i. In each of these subdomains, the solution is represented by a proper model $\psi_i(x, p^i, q^i)$ that is constructed in such a way so as to meet certain conditions on the subdomain-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.

If we define the least squares "local error", i.e. the error 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 is given by:

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

The parameters p^i are determined by minimizing $E_L[p^i, q^i]$ for a given set of values for q^i. The additional parameters q^i, are then adjusted so that the complete solution written as:

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

minimizes the "total error" given by equation 2. The above steps are repeated until a convergence criterion prevails. A detailed algorithmic description is deferred to section 3.

2 Definitions and Terms

2.1 Obreshkov Polynomials and Related Operators

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

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

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

Obreshkov [9], obtained the following result for the unique polynomial of the minimal degree $k+m+1$.

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

We may then define an operator $L_{x\in[a,b]}^{m,k}$ via the following relation:

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

We define the quantities:

$$S_{a,b}^{k,m}(f,x) \equiv f(x) - P_{a,b}^{k,m}(f,x) = (1 - L_{x\in[a,b]}^{k,m}) f(x) \tag{7}$$

with the understanding that outside the domain, i.e. for $x \notin [a,b]$, $S_{a,b}^{k,m}(f,x)$ vanishes, and

$$\begin{aligned} B_{a,b}^{k,m}(f,x) &\equiv f(x) - S_{a,b}^{k,m}(f,x) = (1 - (1 - L_{x\in[a,b]}^{k,m})) f(x) \\ &= L_{x\in[a,b]}^{k,m} f(x) = P_{a,b}^{k,m}(f,x) \end{aligned} \tag{8}$$

$S_{a,b}^{k,m}(f,x)$ has the property that at $x = a$, $(x = b)$ vanishes along with all its derivatives up to k^{th}, (m^{th}) order. We call it an f-spline (since it is based on the function f) and the quantity $B_{a,b}^{k,m}(f,x)$ a boundary match (since it resembles f on the boundary).

2.2 Neural Splines and Model Description

When $f(.)$ is chosen to be a Neural Network, then we may call $S(f,.)$ a Neural Spline. In each of the rectangular subdomains D_i we represent our model as:

$$\psi_i(x, p^i, q^i) = B_{a,b}^{k,m}(f,x) + S_{a,b}^{l,n}(N,x) \tag{9}$$

where $N(x,p^i)$ is a Neural Network with weights denoted by p^i. The parameters q^i represent the values of $f(x)$ and possibly of its derivatives on the boundary ∂D_i. In one dimension the model $\psi_i(x,p^i,q^i)$ so defined, satisfies by construction the following boundary conditions:

$$\frac{d^j}{dx^j}\psi_i(x,p^i,q^i)|_{x=a} = f^{(j)}(a),\ j=0,...,\min(k,l) \tag{10}$$

$$\frac{d^j}{dx^j}\psi_i(x,p^i,q^i)|_{x=b} = f^{(j)}(b),\ j=0,...,\min(m,n) \tag{11}$$

As an example in the case $k=l=m=n=0$, the one-dimensional model is written as:

$$\begin{aligned}\psi(x,p,q) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a} + \\ N(x,p) - [N(a,p)\frac{x-b}{a-b} + N(b,p)\frac{x-a}{b-a}]\end{aligned} \tag{12}$$

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

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

where the following notation is used:

$$\begin{aligned}\pi_{1,0}(x,a,b) &= \frac{x-b}{a-b} \\ \pi_{3,0}(x,a,b) &= \frac{(x-b)^2}{(a-b)^2}\left(1+2\frac{x-a}{b-a}\right) \\ \pi_{3,1}(x,t_{i-1},t_i) &= (x-a)\frac{(x-b)^2}{(a-b)^2} \\ \tau_{2k+1,j}(x,a,b) &\equiv \pi_{2k+1,j}(x,b,a)\end{aligned} \tag{13}$$

3 Partitioning and Procedures

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 $D_i=[t_i,t_{i+1}]$.

1. Introduce a set of external parameters $f_i^{(0)}, f_i^{(1)}, \cdots, f_i^{(k)}$ (collectively denoted by q^i) that specify 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 conditions specified at the two bracketing knots t_i and t_{i+1} and minimize the local least squares "error" $E_i[p^i, q^i]$ with respect to p^i, keeping the external q^i parameters fixed.
3. Adjust the external parameters q^i (i.e. the prescribed values at the knots) in such a way so as to minimize the total "error" $E_T\left[p, q\right] = \sum_i E_L\left[p^i, q^i\right]$ keeping p^i fixed.
4. Repeat from step 2, until some termination criterion is satisfied.

Note that the procedure in step 2, can be implemented in parallel, since the local models are being determined independently, given that the external parameters remain constant, as it will become evident shortly. 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 domains. However this part is not time consuming and hence it is not critical. There are some important points that must be stressed. The initial values for the external parameters are extremely important. Far off values, may decelerate the convergence dramatically. Hence we deviced a preprocessing scheme to ensure that the initial values are close to their actual values. This is achieved by fitting a single neural network in every interval and then use this model to generate the initial values for the external parameters. The network parameters resulting from the preprocessing are subsequently used to initialize the weights p^i of the final model $\psi_i(x, p^i, q^i) = B(f, x) + S(N, x)$. In this article the Neural Network used is the sigmoidal perceptron with one hidden layer, given by:

$$N(x, p) = \sum_{i=1}^{H} p_{3i-2}\sigma(p_{3i-1}x + p_{3i}), \quad \sigma(z) = \frac{1}{1 + e^{-z}} \tag{14}$$

Global optimization is used in each subdomain in the phase of preprocessing. In practice, in order to accelerate the process, we proceed by first applying a local search procedure, and only if this proves to be inadequate (i.e. if it produces a local error above a set threshold) we employ global optimization techniques.

4 Numerical Experiments

Experiments were conducted with several data sets. 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 Figure 1

Several cases were examined by varying the number of partitions and the number of hidden units of the Neural Networks in each partition. Two sets of points were used: a rather sparse point set for the training and a dense point set for testing. Since the local models are mutually independent, there are no propagating errors across the subdomains and so a rather modest optimization stopping criterion may be used, that accelerates the process without substantially sacrificing the model's approximation capability.

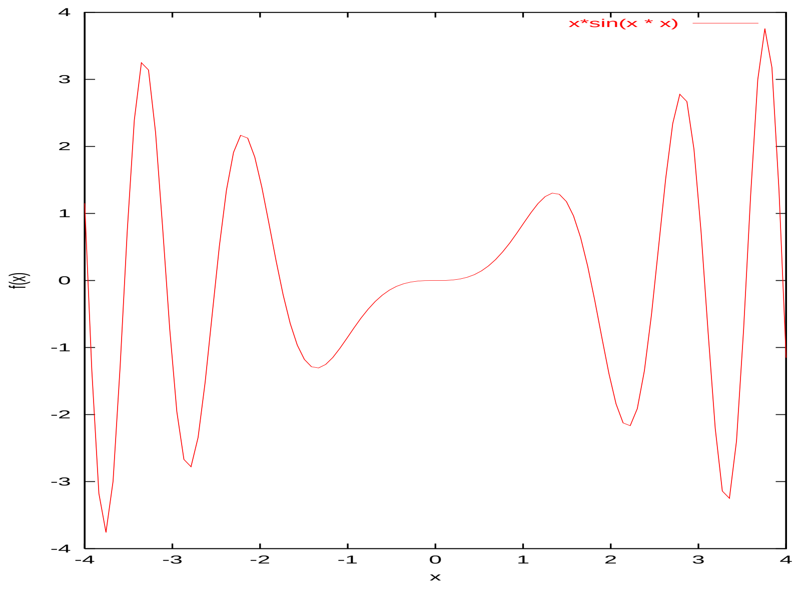

Fig. 1. Plotting of $x \sin x^2$

To test the efficiency we compared solution times for several combinations of the partition number and the number of the hidden nodes of each network keeping their product at comparable values to avoid overblown model complexity. A solution is taken to be one that produces a prescribed value for the max absolute pointwise error for the training set. The solution time is taken to be the cpu time spent by a uniprocessor system. In order to test the gain coming from parallel processing, our implementation that is based on message passing programming, was run on a multiprocessor system and observed how the solution time scaled down.

5 Experiments

5.1 Resources

The following results were obtained by using 25 Pentium III - 450MHZ machines running on Linux with kernel 2.4.0 The Lam v6.5.3 of MPI was employed for the distributed processing.

5.2 Results

In table 1 we list the square approximation error (columns ERR) and the number of knots (columns N) for cubic spline interpolation. 1000 points were used for testing. Diagrammatically tis is represented in figure 2.

Table 1. Approximation error for the cubic spline interpolation

N	ERR	N	ERR
40	168.05	75	7.86
45	90.95	80	6.08
50	52.37	85	4.82
55	32.18	90	3.90
60	21.00	95	3.22
65	14.54	100	2.69
70	10.47		

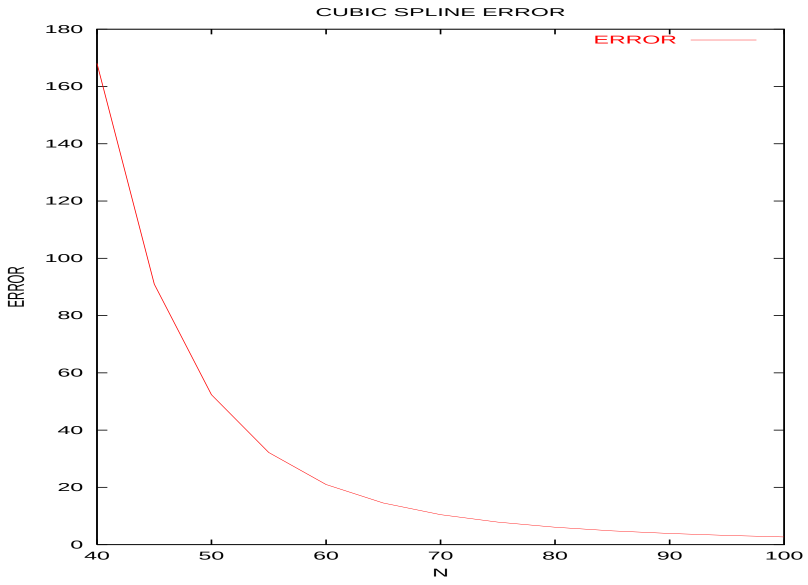

Fig. 2. Approximation error for the cubic spline interpolation

In all of our experiments we used 200 randomly selected points from the interval [-4,4] for training and 1000 points for testing. The reported approximation error refers to the test error. In table 2 we list the square approximation error (column ERR) and the training time (column TIME) for a single neural network. For the train of the neural network we used the single linkage clustering global optimization method due to Kan[10]. The column NODES represents the number of hidden nodes in the neural network.

In table 3 we list the square approximation error (column ERR) and the training time (column TIME) for the suggested method. For the training of the neural networks we used the single linkage clustering global optimization method due to Kan[10] The column INTERVALS represents the number of the intervals,

Table 2. Approximation error and execution time for a single neural network

NODES	ERR	TIME
8	4.52	46.27
10	$1.9 * 10^{-3}$	110.86
12	$2.0 * 10^{-4}$	179.33
14	$1.3 * 10^{-5}$	349.75
16	$2.0 * 10^{-6}$	418.334
18	$1.4 * 10^{-7}$	488.219
24	$9 * 10^{-8}$	598.124
30	$6 * 10^{-8}$	634.896
36	$4.3 * 10^{-8}$	697.150

in which we partitioned the problem. In this experiments we used 4 hidden nodes in each of the neural networks.

Table 3. Approximation error and execution time for the proposed method

INTERVALS	ERR	TIME
2	4.733	56.25
4	0.1497	90.72
8	$2.7 * 10^{-5}$	182.86
10	$1.1 * 10^{-5}$	193.55
15	$2.3 * 10^{-7}$	87.30

In table 4 we list the square approximation error (column ERR) and the training time (column TIME) for the suggested method. For the training of the neural networks we used the single linkage clustering global optimization method due to Kan[10]. The column INTERVALS represents the number of the intervals, in which we partitioned the problem. In this experiments we used 8 hidden nodes in each of the neural networks.

In figure 5.2 we show the absolute difference between $x \sin(x^2)$ and our approximation for 10 intervals and 8 nodes at each interval.

In table 5 we compare the training time for the multiple interval method on one processor (column T_1) in comparison with the training time for the same method run on multiple processors (column T_I, where I is the number of processors). We use column I for the number of intervals and column N for the number of hidden nodes in each of the neural networks. We use column E for the maximum absolute approximation error. In the column DIFF we have the relative difference between the multiple processor case and the single processor case.

Table 4. Approximation error and execution time for the proposed method

INTERVALS	ERR	TIME
2	$1.1 * 10^{-5}$	296.53
4	$3 * 10^{-7}$	126.53
8	$2 * 10^{-7}$	103.076
10	$2 * 10^{-8}$	86.40
15	10^{-8}	276.67

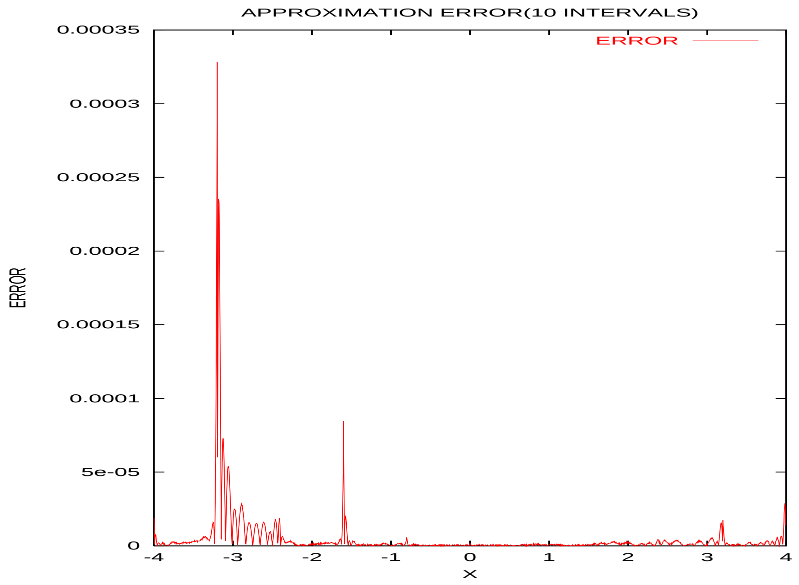

Fig. 3. Approximation error for the proposed method

Table 5. Multiple processors vs one processor

I	N	E	T_1	T_I	$DIFF$
2	8	0.0021	603.19	296.53	-50.84%
4	8	0.00012	476.90	126.53	-73.47%
8	4	0.0038	857.46	182.86	-78.67%
10	2	0.072	620.96	135.71	-78.15%
10	4	0.0054	866.38	193.55	-77.66%
15	1	0.073	1065.74	216.87	-79.65%
15	2	0.015	1187.66	282.51	-76.21%

6 Conclusions

Although our results are only preliminary, we can however
draw some conclusions.

– The Neural Splines seem to be quite convenient and offer a flexible basis for functional approximation.
– Parallel processing plays an important role to the efficiency of the method, as can be realised by comparing the times spent on uniprocessor and multiprocessor systems. For large problems this will be the key advantage of our method.
– The scaling behaviour seems to be described by:

$$\frac{T_1}{T_I} = \frac{1 - e^{-\gamma I}}{1 - e^{-\gamma}}, \quad \gamma > 0$$

which for small values of I, scales linearly.(I, denotes the number of processors that is equal to the number of the partitions). The value of γ reflects the overhead of the calculation as well as the non-parallelized parts of it. Note that the optimization with respect to the external parameters can be accelerated by applying even-odd knot parallelization that will further reduce the value of γ. This will be important for problems in higher dimensions, since there the number of the external parameters is expected to grow significantly.

Future research will focus on higher dimensional problems and to the solution of differential equations.

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. Stochastic Global Optimization Methods: Clustering Methods. A.H.G Rinnooy Kan, G.T. Timmer, Mathematical Prograaming 39(1987) pp:27-56.

11. F. Theos, Master Thesis, June 2001, Department of Computer Science, University of Ioannina, Greece.
12. Papageorgiou D. G., Demetropoulos I. N. and Lagaris I. E., *The Merlin Control Language for strategic optimization,* Comput. Phys. Commun. 109(1998)250-275
13. Papageorgiou D. G., Demetropoulos I. N. and Lagaris I. E., *MERLIN-3.0 A multidimensional optimization environment,* Comput. Phys. Commun. 109(1998)227-249