Artificial Neural Networks for Solving Ordinary and Partial Differential Equations

Isaac Elias Lagaris, Aristidis Likas, *Member, IEEE*, and Dimitrios I. Fotiadis

***Abstract*—We present a method to solve initial and boundary value problems using artificial neural networks. A trial solution of the differential equation is written as a sum of two parts. The first part satisfies the initial/boundary conditions and contains no adjustable parameters. The second part is constructed so as not to affect the initial/boundary conditions. This part involves a feedforward neural network containing adjustable parameters (the weights). Hence by construction the initial/boundary conditions are satisfied and the network is trained to satisfy the differential equation. The applicability of this approach ranges from single ordinary differential equations (ODE's), to systems of coupled ODE's and also to partial differential equations (PDE's). In this article, we illustrate the method by solving a variety of model problems and present comparisons with solutions obtained using the Galekrkin finite element method for several cases of partial differential equations. With the advent of neuroprocessors and digital signal processors the method becomes particularly interesting due to the expected essential gains in the execution speed.**

***Index Terms*—Collocation method, finite elements, neural networks, neuroprocessors, ordinary differential equations, partial differential equations.**

I. Introduction

MANY methods have been developed so far for solving differential equations. Some of them produce a solution in the form of an array that contains the value of the solution at a selected group of points. Others use basis-functions to represent the solution in analytic form and transform the original problem usually to a system of algebraic equations. Most of the previous work in solving differential equations using neural networks is restricted to the case of solving the systems of algebraic equations which result from the discretization of the domain. The solution of a linear system of equations is mapped onto the architecture of a Hopfield neural network. The minimization of the network's energy function provides the solution to the system of equations [2], [6], [5].

Another approach to the solution of ordinary differential equations is based on the fact that certain types of splines, for instance B_1-splines, can be derived by the superposition of piecewise linear activation functions [3], [4]. The solution of a differential equation using B_1-splines as basis functions, can be obtained by solving a system of linear or nonlinear equations in order to determine the coefficients of splines. Such a solution form is mapped directly on the architecture of a feedforward neural network by replacing each spline with the sum of piecewise linear activation functions that correspond to the hidden units. This method considers local basis-functions and in general requires many splines (and consequently network parameters) in order to yield accurate solutions. Furthermore, it is not easy to extend these techniques to multidimensional domains.

In this article we view the problem from a different angle. We present a method for solving both ordinary differential equations (ODE's) and partial differential equations (PDE's) (defined on *orthogonal box* domains) that relies on the function approximation capabilities of feedforward neural networks and results in the construction of a solution written in a differentiable, closed analytic form. This form employs a feedforward neural network as the basic approximation element, whose parameters (weights and biases) are adjusted to minimize an appropriate error function. To train the network we employ optimization techniques, which in turn require the computation of the gradient of the error with respect to the network parameters. In the proposed approach the model function is expressed as the sum of two terms: the first term satisfies the initial/boundary conditions and contains no adjustable parameters. The second term involves a feedforward neural network to be trained so as to satisfy the differential equation. Since it is known that a multilayer perceptron with one hidden layer can approximate any function to arbitrary accuracy, it is reasonable to consider this type of network architecture as a candidate model for treating differential equations. 9 November 2022

The employment of a neural architecture adds many attractive features to the method:

- The solution via ANN's is a *differentiable, closed analytic form* easily used in any subsequent calculation. Most other techniques offer a discrete solution (for example predictor-corrector, or Runge–Kutta methods) or a solution of limited differentiability (for example finite elements).
- The employment of neural networks provides a solution with very good generalization properties. Comparative results with the finite element method presented in this work illustrate this point clearly.
- The required number of model parameters is far less than any other solution technique and, therefore, compact solution models are obtained with very low demand on memory space.
- The method is general and can be applied to ODE's, systems of ODE's and to PDE's defined on orthogonal

Manuscript received December 1, 1996; revised May 21, 1998.

The authors are with the Department of Computer Science, University of Ioannina, GR 45110 Ioannina, Greece.

Publisher Item Identifier S 1045-9227(98)05911-6.

box boundaries. Moreover, work is in progress to treat the case of irregular (arbitrarily shaped) boundaries.

- The method can be realized in hardware, using neuroprocessors, and hence offer the opportunity to tackle in real-time difficult differential equation problems arising in many engineering applications.
- The method can also be efficiently implemented on parallel architectures.

In the next section we describe the general formulation of the proposed approach and derive formulas for computing the gradient of the error function. Section III illustrates some classes of problems where the proposed method can be applied, and describes the appropriate form of the trial solution. Section IV presents numerical examples from the application of the technique to several test problems, and provides details concerning the implementation of the method and the accuracy of the obtained solution. We also make a comparison of our results with those obtained by the finite element method for the examined PDE problems. Finally, Section VI contains conclusions and directions for future research.

II. Description of the Method

The proposed approach will be illustrated in terms of the following general differential equation definition:

$$\boxed{G(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x})) = 0,} \quad \vec{x} \in D \tag{1}$$

subject to certain boundary conditions (BC's) (for instance Dirichlet and/or Neumann), where $\vec{x} = (x_1, \cdots, x_n) \in R^n, D \subset R^n$ denotes the definition domain and $\Psi(\vec{x})$ is the solution to be computed.

To obtain a solution to the above differential equation, the collocation method is adopted [1] which assumes a discretization of the domain D and its boundary S into a set points $\hat{D}$ and $\hat{S}$, respectively. The problem is then transformed into the following system of equations:

$$\boxed{G(\vec{x}_i, \Psi(\vec{x}_i), \nabla\Psi(\vec{x}_i), \nabla^2\Psi(\vec{x}_i)) = 0,} \quad \forall \vec{x}_i \in \hat{D} \tag{2}$$

subject to the constraints imposed by the BC's. F.E method

If $\Psi_t(\vec{x}, \vec{p})$ denotes a trial solution with adjustable parameters $\vec{p}$, the problem is transformed to

$$\boxed{\min_{\vec{p}} \sum_{\vec{x}_i \in \hat{D}} (G(\vec{x}_i, \Psi_t(\vec{x}_i, \vec{p}), \nabla\Psi_t(\vec{x}_i, \vec{p}), \nabla^2\Psi_t(\vec{x}_i, \vec{p})))^2} \tag{3}$$

subject to the constraints imposed by the BC's.

In the proposed approach, the trial solution Ψ_t employs a feedforward neural network and the parameters $\vec{p}$ correspond to the weights and biases of the neural architecture. We choose a form for the trial function $\Psi_t(\vec{x})$ such that by construction satisfies the BC's. This is achieved by writing it as a sum of two terms

$$\boxed{\Psi_t(\vec{x}) = A(\vec{x}) + F(\vec{x}, N(\vec{x}, \vec{p}))} \tag{4}$$

where $N(\vec{x}, \vec{p})$ is a single-output feedforward neural network with parameters $\vec{p}$ and n input units fed with the input vector $\vec{x}$.

$\vec{x} = (x_1, \cdots, x_n)$

The term $A(\vec{x})$ contains no adjustable parameters and satisfies the boundary conditions. The second term F is constructed so as not to contribute to the BC's, since $\Psi_t(\vec{x})$ must also satisfy them. This term employs a neural network whose weights and biases are to be adjusted in order to deal with the minimization problem. Note at this point that the problem has been reduced from the original constrained optimization problem to an unconstrained one (which is much easier to handle) due to the choice of the form of the trial solution that satisfies by construction the BC's.

In the next section we present a systematic way to construct the trial solution, i.e., the functional forms of both A and F. We treat several common cases that one frequently encounters in various scientific fields. As indicated by our experiments, the approach based on the above formulation is very effective and provides in reasonable computing time accurate solutions with impressive generalization (interpolation) properties.

A. *Gradient Computation* $\rightarrow$ TODO: Compare with Baydin's AD

The efficient minimization of (3) can be considered as a procedure of training the neural network, where the error corresponding to each input vector $\vec{x}_i$ is the value $G(\vec{x}_i)$ which has to become zero. Computation of this error value involves not only the network output (as is the case in conventional training) but also the derivatives of the output with respect to any of its inputs. Therefore, in computing the gradient of the error with respect to the network weights, we need to compute not only the gradient of the network but also the gradient of the network derivatives with respect to its inputs.

Consider a multilayer perceptron with n input units, one hidden layer with H sigmoid units and a linear output unit. The extension to the case of more than one hidden layers can be obtained accordingly. For a given input vector $\vec{x} = (x_1, \cdots, x_n)$ the output of the network is $N = \Sigma_{i=1}^{H} v_i \sigma(z_i)$ where $z_i = \Sigma_{j=1}^{n} w_{ij} x_j + u_i, w_{ij}$ denotes the weight from the input unit j to the hidden unit i, v_i denotes the weight from the hidden unit i to the output, u_i denotes the bias of hidden unit i, and $\sigma(z)$ is the sigmoid transfer function. It is straightforward to show that

$$\frac{\partial^k N}{\partial x_j^k} = \sum_{i=1}^{H} v_i w_{ij}^k \sigma_i^{(k)} \tag{5}$$

where $\sigma_i = \sigma(z_i)$ and $\sigma^{(k)}$ denotes the kth-order derivative of the sigmoid. Moreover, it is readily verifiable that

$$\boxed{\frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}}} \boxed{\frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}}} \cdots \boxed{\frac{\partial^{\lambda_n}}{\partial x_2^{\lambda_n}}} N = \sum_{i=1}^{n} v_i P_i \sigma_i^{(\Lambda)} \tag{6}$$

where

$$P_i = \prod_{k=1}^{n} w_{ik}^{\lambda_k} \tag{7}$$

and $\Lambda = \Sigma_{i=1}^{n} \lambda_i$.

Equation (6) indicates that the derivative of the network with respect to any of its inputs is equivalent to a feedforward neural network $N_g(\vec{x})$ with one hidden layer, having the same values for the weights w_{ij} and thresholds u_i and with each weight v_i being replaced with $v_i P_i$. Moreover, the transfer function of each hidden unit is replaced with the Λth-order derivative of the sigmoid.

Therefore, the gradient of N_g with respect to the parameters of the original network can be easily obtained as

$$\frac{\partial N_g}{\partial v_i} = P_i \sigma_i^{(\Lambda)} \tag{8}$$

$$\frac{\partial N_g}{\partial u_i} = v_i P_i \sigma_i^{(\Lambda+1)} \tag{9}$$

$$\boxed{\frac{\partial N_g}{\partial w_{ij}}} = x_j v_i P_i \sigma_i^{(\Lambda+1)} + v_i \lambda_j w_{ij}^{\lambda_j - 1} \left(\prod_{k=1, k\neq j} w_{ik}^{\lambda_k} \right) \sigma_i^{(\Lambda)}. \tag{10}$$

Once the derivative of the error with respect to the network parameters has been defined it is then straightforward to employ almost any minimization technique. For example it is possible to use either the steepest descent (i.e., the backpropagation algorithm or any of its variants), or the conjugate gradient method or other techniques proposed in the literature. In our experiments we have employed the quasi-Newton BFGS method [9] (independently proposed at 1970 by Broyden *et al.*) that is quadratically convergent and has demonstrated excellent performance. It must also be noted, that the derivatives of each network (or gradient network) with respect to the parameters for a given grid point may be obtained simultaneously in the case where parallel hardware is available. Moreover, in the case of backpropagation, the on-line or batch mode of weight updates may be employed.

III. Illustration of the Method

A. Solution of Single ODE's and Systems of Coupled ODE's

To illustrate the method, we consider the *first-order ODE*

$$\boxed{\frac{d\Psi(x)}{dx}} = f(x, \Psi) \tag{11}$$

with $x \in [0, 1]$ and the IC $\Psi(0) = A$.

A trial solution is written as

$$\Psi_t(x) = A + xN(x, \vec{p}) \tag{12}$$

where $N(x, \vec{p})$ is the output of a feedforward neural network with one input unit for x and weights $\vec{p}$. Note that $\Psi_t(x)$ satisfies the IC by construction. The error quantity to be minimized is given by

$$E[\vec{p}] = \sum_i \left\{ \boxed{\frac{d\Psi_t(x_i)}{dx}} - f(x_i, \Psi_t(x_i)) \right\}^2 \tag{13}$$

where the x_i's are points in [0, 1]. Since $d\Psi_t(x)/dx = N(x, \vec{p}) + x dN(x, \vec{p})/dx$, it is straightforward to compute the gradient of the error with respect to the parameters $\vec{p}$ using (5)–(10). The same holds for all subsequent model problems.

The same procedure can be applied to the *second-order ODE*

$$\boxed{\frac{d^2\Psi(x)}{dx^2}} = f\left(x, \Psi, \frac{d\Psi}{dx}\right). \tag{14}$$

For the *initial value* problem: $\Psi(0) = A$ and $(d/dx)\Psi(0) = A'$, the trial solution can be cast as

$$\Psi_t(x) = A + A'x + x^2 N(x, \vec{p}). \tag{15}$$

For the *two point Dirichlet* BC: $\Psi(0) = A$ and $\Psi(1) = B$, the trial solution is written as

$$\Psi_t(x) = A(1-x) + Bx + x(1-x)N(x, \vec{p}). \tag{16}$$

In the above two cases of second-order ODE's the error function to be minimized is given by the following equation:

$$E[\vec{p}] = \sum_i \left\{ \boxed{\frac{d^2\Psi_t(x_i)}{dx^2}} - f\left(x_i, \Psi_t(x_i), \boxed{\frac{d\Psi_t(x_i)}{dx}}\right) \right\}^2. \tag{17}$$

For *systems of K first-order ODE's*

$$\frac{d\Psi_i}{dx} = f_i(x, \Psi_1, \Psi_2, \cdots \Psi_K) \tag{18}$$

with $\Psi_i(0) = A_i, (i = 1, \cdots, K)$ we consider one neural network for each trial solution Ψ_{t_i} $(i = 1, \cdots, K)$ which is written as

$$\Psi_{t_i}(x) = A_i + xN_i(x, \vec{p}_i) \tag{19}$$

and we minimize the following error quantity:

$$E[\vec{p}] = \sum_{k=1}^{K} \sum_i \left\{ \boxed{\frac{d\Psi_{t_k}(x_i)}{dx}} - f_k(x_i, \Psi_{t_1}, \Psi_{t_2}, \cdots, \Psi_{t_K}) \right\}^2. \tag{20}$$

B. Solution of Single PDE's

We treat here two–dimensional problems only. However, it is straightforward to extend the method to more dimensions. For example, consider the *Poisson equation*

$$\boxed{\frac{\partial^2\Psi(x,y)}{\partial x^2}} + \boxed{\frac{\partial^2\Psi(x,y)}{\partial y^2}} = f(x, y). \tag{21}$$

$x \in [0, 1], y \in [0, 1]$ with *Dirichlet* BC: $\Psi(0, y) = f_0(y), \Psi(1, y) = f_1(y)$, $\Psi(x, 0) = g_0(x)$ and $\Psi(x, 1) = g_1(x)$. The trial solution is written as

$$\Psi_t(x, y) = A(x, y) + x(1-x)y(1-y)N(x, y, \vec{p}) \tag{22}$$

where $A(x, y)$ is chosen so as to satisfy the BC, namely

$$\begin{aligned} A(x, y) = {} & (1-x)f_0(y) + xf_1(y) \\ & + (1-y)\{g_0(x) - [(1-x)g_0(0) + xg_0(1)]\} \\ & + y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\}. \end{aligned} \tag{23}$$

For *mixed boundary conditions* of the form $\Psi(0, y) = f_0(y)$, $\Psi(1, y) = f_1(y)$, $\Psi(x, 0) = g_0(x)$, and $(\partial\Psi(x, 1)/\partial y) = g_1(x)$ (i.e., *Dirichlet* on part of the boundary and *Neumann* elsewhere), the trial solution is written as

$$\begin{aligned} \Psi_t(x, y) = {} & B(x, y) + x(1-x)y\Big[N(x, y, \vec{p}) \\ & - N(x, 1, \vec{p}) - \boxed{\frac{\partial N(x, 1, \vec{p})}{\partial y}}\Big] \end{aligned} \tag{24}$$

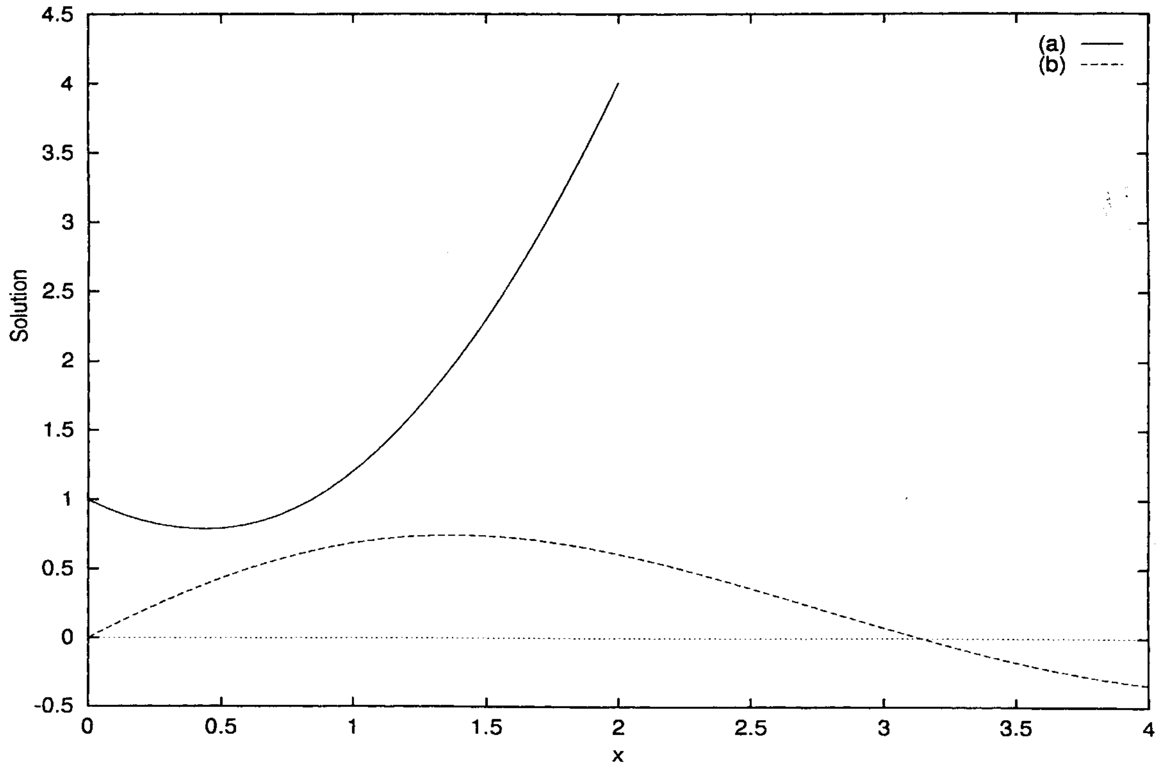

Fig. 1. Exact solutions of ODE problems 1 and 2.

and $B(x,y)$ is again chosen so as to satisfy the BC's:

$$\begin{aligned}B(x,y) &= (1-x)f_0(y) + xf_1(y) + g_0(x)\\ &\quad - [(1-x)g_0(0) + xg_0(1)]\\ &\quad + y\{g_1(x) - [(1-x)g_1(0) + xg_1(1)]\}. \qquad (25)\end{aligned}$$

Note that the second term of the trial solution does not affect the boundary conditions since it vanishes at the part of the boundary where Dirichlet BC's are imposed and its gradient component normal to the boundary vanishes at the part of the boundary where Neumann BC's are imposed.

In all the above PDE problems the error to be minimized is given by

$$E[\vec{p}] = \sum_i \left\{ \frac{\partial^2 \Psi(x_i,y_i)}{\partial x^2} + \frac{\partial^2 \Psi(x_i,y_i)}{\partial y^2} - f(x_i,y_i) \right\}^2 \qquad (26)$$

where (x_i, y_i) are points in [0, 1] $\times$ [0, 1].

IV. Examples

In this section we report on the solution of a number of model problems. In all cases we used a multilayer perceptron having one hidden layer with ten hidden units and one linear output unit. The sigmoid activation of each hidden unit is $\sigma(x) = 1/(1+e^{-x})$. For each test problem the exact analytic solution $\Psi_a(\vec{x})$ was known in advance. Therefore we test the accuracy of the obtained solutions by computing the deviation $\Delta\Psi(\vec{x}) = \Psi_t(\vec{x}) - \Psi_a(\vec{x})$. To perform the error minimization we employed the Merlin/MCL 3.0 [7], [8] optimization package. From the several algorithms that are implemented therein, the quasi-Newton *BFGS* [9] method seemed to perform better in these problems and hence we used it in all of our experiments. A simple criterion for the gradient norm was used for termination. In order to illustrate the characteristics of the solutions provided by the neural method, we provide figures displaying the corresponding deviation $\Delta\Psi(\vec{x})$ both at the few points (training points) that were used for training and at many other points (test points) of the domain of each equation. The latter kind of figures are of major importance since they show the interpolation capabilities of the neural solutions which seem to be superior compared to other solutions. Moreover, in the case of ODE's we also consider points outside the training interval in order to obtain an estimate of the extrapolation performance of the obtained solution.

A. ODE's and Systems of ODE's

1) Problem 1:

$$\frac{d\Psi}{dx} + \left(x + \frac{1+3x^2}{1+x+x^3}\right)\Psi = x^3 + 2x + x^2\,\frac{1+3x^2}{1+x+x^3} \qquad (27)$$

with $\Psi(0) = 1$ and $x \in [0,1]$. The analytic solution is $\Psi_a(x) = e^{-x^2/2}/(1+x+x^3) + x^2$ and is displayed in Fig. 1(a). According to (12) the trial neural form of the solution is taken to be $\Psi_t(x) = 1 + xN(x,\vec{p})$. The network was trained using a grid of ten equidistant points in [0, 1]. Fig. 2 displays the deviation $\Delta\Psi(x)$ from the exact solution corresponding at the grid points (diamonds) and the deviation at many other points

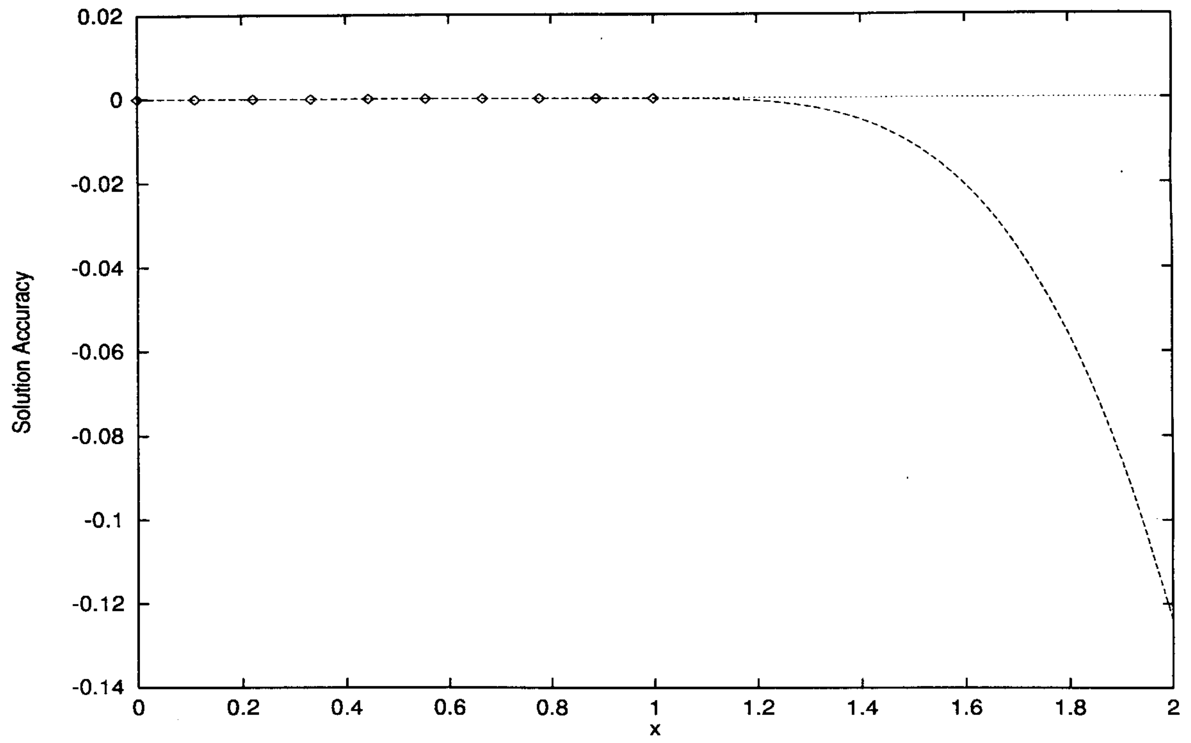

Fig. 2. Problem 1: Accuracy of the computed solution.

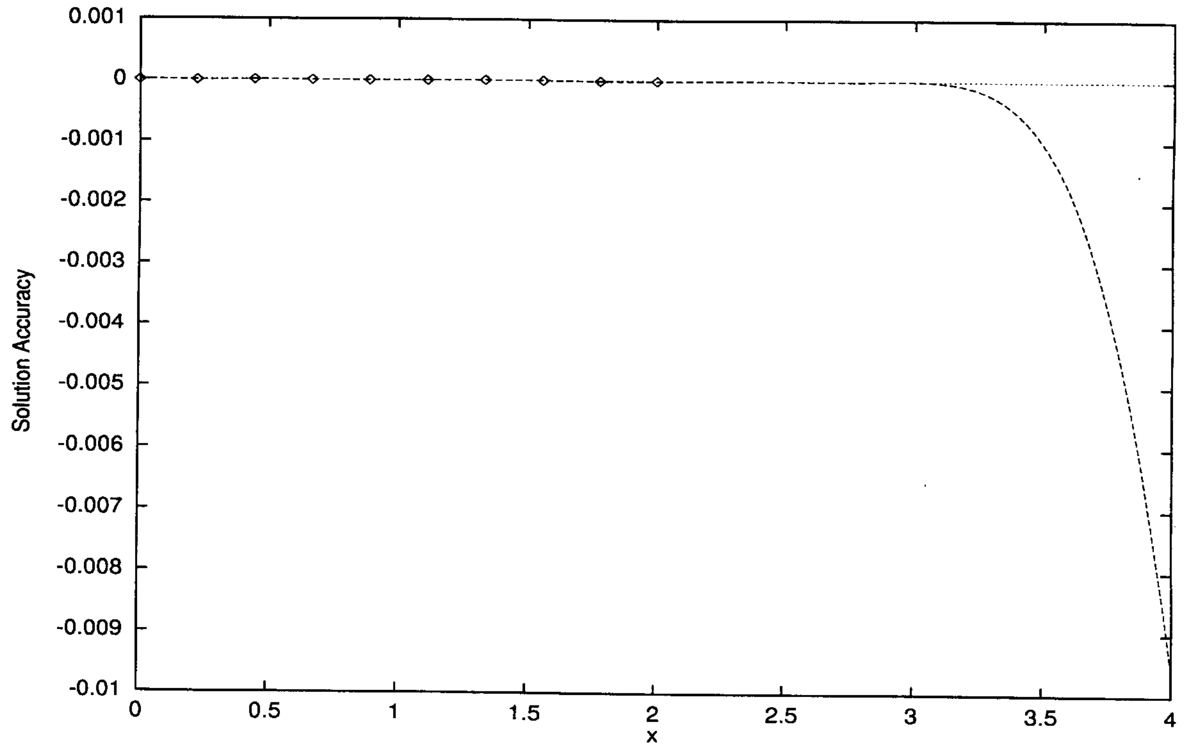

Fig. 3. Problem 2: Accuracy of the computed solution.

in [0, 1] as well as outside that interval (dashed line). It is clear that the solution is of high accuracy, although training was performed using a small number of points. Moreover, the extrapolation error remains low for points near the equation domain.

2) Problem 2:

$$\frac{d\Psi}{dx} + \frac{1}{5}\Psi = e^{-(x/5)}\cos(x) \tag{28}$$

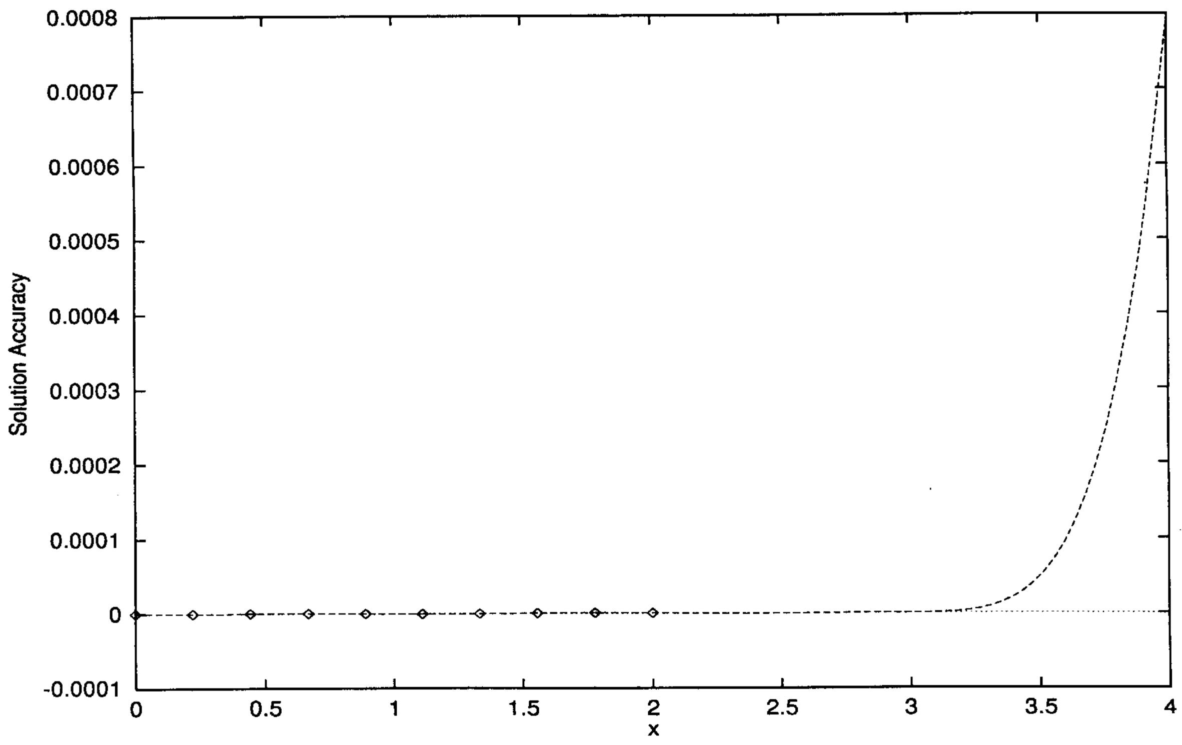

Fig. 4. Problem 3 with initial conditions: Accuracy of the computed solution.

with $\Psi(0) = 0$ and $x \in [0, 2]$. The analytic solution is $\Psi_a(x) = e^{-(x/5)} \sin(x)$ and is presented in Fig. 1(b). The trial neural form is $\Psi_t(x) = xN(x, \vec{p})$ according to (12). As before we used a grid of ten equidistant points in [0, 2] to perform the training. In analogy with the previous case, Fig. 3 displays the deviation $\Delta\Psi(x)$ at the grid points (diamonds) and at many other points inside and outside the training interval (dashed line).

3) Problem 3: Given the differential equation

$$\frac{d^2\Psi}{dx^2} + \frac{1}{5}\,\frac{d\Psi}{dx} + \Psi = -\frac{1}{5}\,e^{-(x/5)} \cos x \tag{29}$$

consider the *initial value* problem: $\Psi(0) = 0$ and $(d/dx)\,\Psi(0) = 1$ with $x \in [0, 2]$. The exact solution is $\Psi(x) = e^{-(x/5)} \sin(x)$ and the trial neural form is $\Psi_t(x) = x + x^2 N(x, \vec{p})$ [from (15)].

Consider also the *boundary value* problem: $\Psi(0) = 0$ and $\Psi(1) = \sin(1)e^{-(1/5)}, x \in [0, 1]$. The exact solution is the same as above, but the appropriate trial neural form is $\Psi_t(x) = x \sin(1)e^{-(1/5)} + x(1 - x)N(x, \vec{p})$ [from (16)].

Again, as before, we used a grid of ten equidistant points and the plots of the deviation from the exact solution are displayed at Figs. 4 and 5 for the initial value and boundary value problem, respectively. The interpretation of the figures is the same as in the previous cases.

From all the above cases it is clear that the method can handle effectively all kinds of ODE's and provide analytic solutions that retain the accuracy throughout the whole domain and not only at the training points.

4) Problem 4: Consider the system of two coupled first-order ODE's

$$\frac{d\Psi_1}{dx} = \cos(x) + \Psi_1^2 + \Psi_2 - (1 + x^2 + \sin^2(x)) \tag{30}$$

$$\frac{d\Psi_2}{dx} = 2x - (1 + x^2)\sin(x) + \Psi_1\Psi_2 \tag{31}$$

with $x \in [0, 3]$ and $\Psi_1(0) = 0$ and $\Psi_2(0) = 1$. The analytic solutions are $\Psi_{a1}(x) = \sin(x)$ and $\Psi_{a2}(x) = 1 + x^2$ and are displayed at Fig. 6(a) and (b), respectively. Following (19) the trial neural solutions are $\Psi_{t_1}(x) = xN_1(x, \vec{p}_1)$ and $\Psi_{t_2}(x) = 1 + xN_2(x, \vec{p}_2)$ where the networks N_1 and N_2 have the same architecture as in the previous cases. Results concerning the accuracy of the obtained solutions at the grid points (diamonds and crosses) and at many other points (dashed line) are presented in Fig. 7(a) and (b) for the functions Ψ_{t_1} and Ψ_{t_2}, respectively.

B. PDE's

We consider boundary value problems with Dirichlet and Neumann BC's. All subsequent problems were defined on the domain [0, 1] × [0, 1] and in order to perform training we consider a mesh of 100 points obtained by considering ten equidistant points of the domain [0, 1] of each variable. In analogy with the previous cases the neural architecture was considered to be a MLP with two inputs (accepting the coordinates x and y of each point), ten sigmoid hidden units, and one linear output unit.

1) Problem 5:

$$\nabla^2\Psi(x, y) = e^{-x}(x - 2 + y^3 + 6y) \tag{32}$$

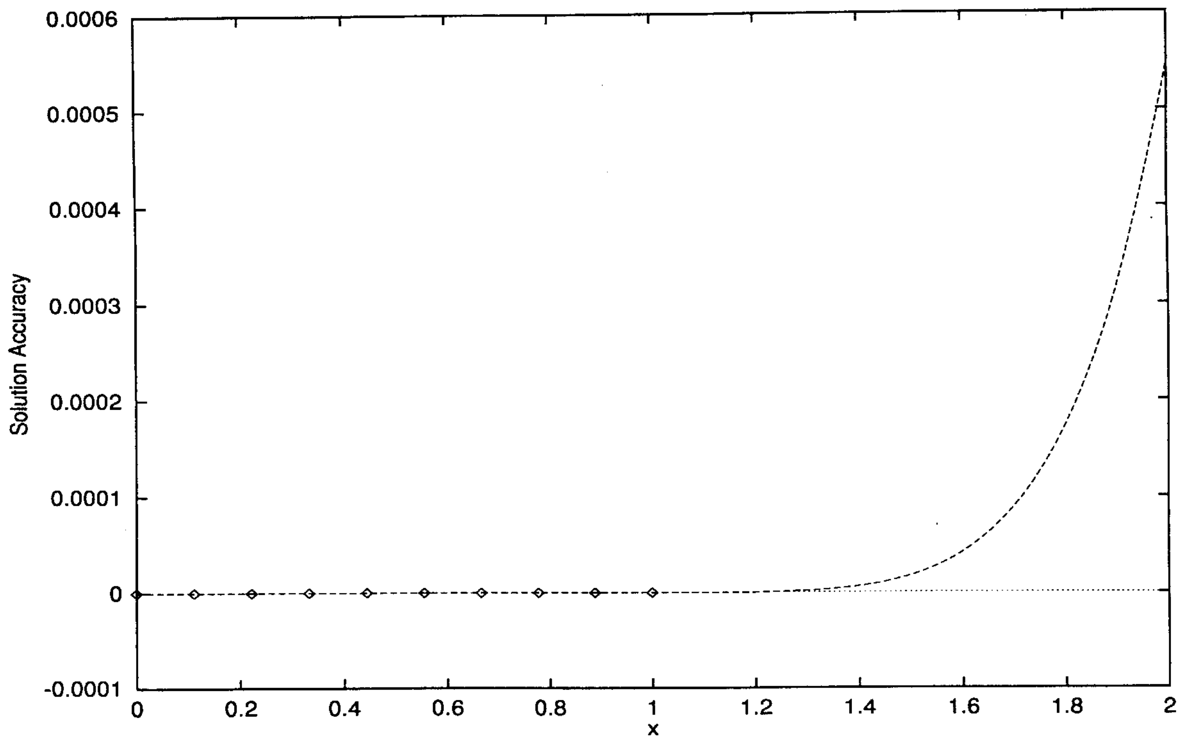

Fig. 5. Problem 3 with boundary conditions: Accuracy of the computed solution.

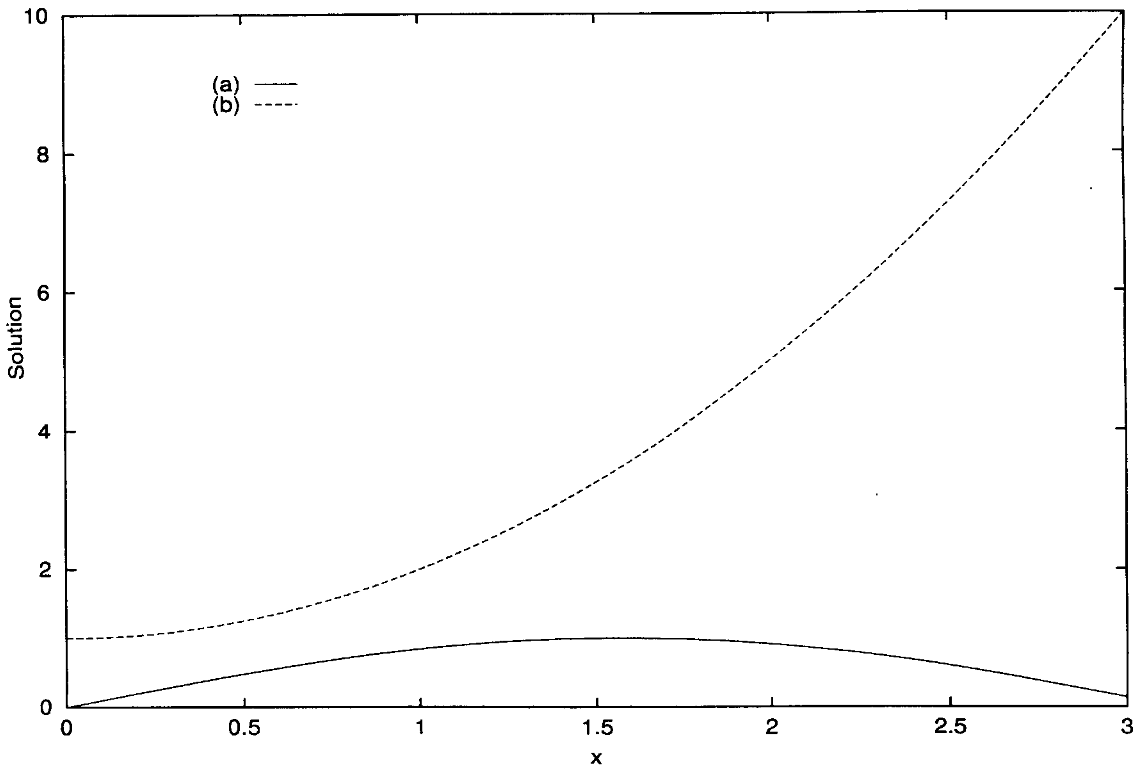

Fig. 6. Exact solutions of the system of coupled ODE's (problem 4).

with $x, y \in [0,1]$ and the Dirichlet BC's: $\Psi(0,y) = y^3$, $\Psi(1,y) = (1+y^3)e^{-1}$ and $\Psi(x,0) = xe^{-x}, \Psi(x,1) = e^{-x}(x+1)$. The analytic solution is $\Psi_a(x,y) = e^{-x}(x+y^3)$ and is displayed in Fig. 8. Using (22) the trial neural form must be written $\Psi_t(x,y) = A(x,y) + x(1-x)y(1-y)N(x,y,\vec{p})$ and $A(x,y)$ is obtained by direct substitution in the general

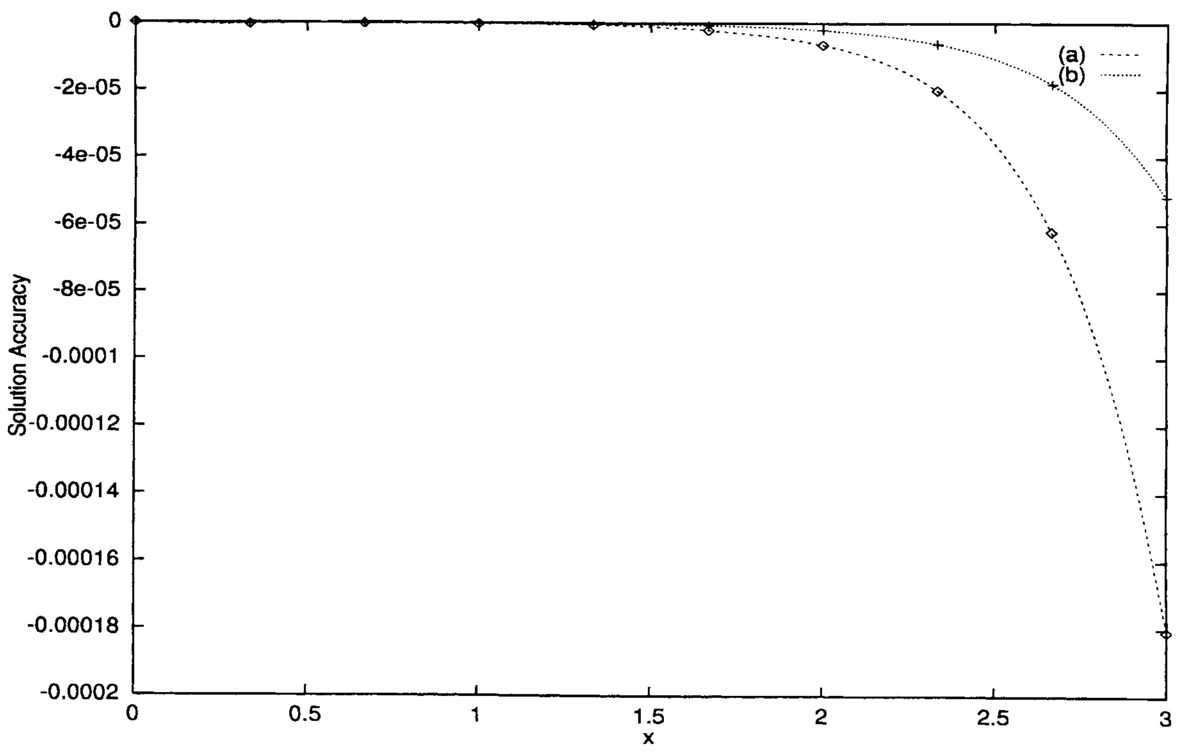

Fig. 7. Problem 4: Accuracy of the computed solutions.

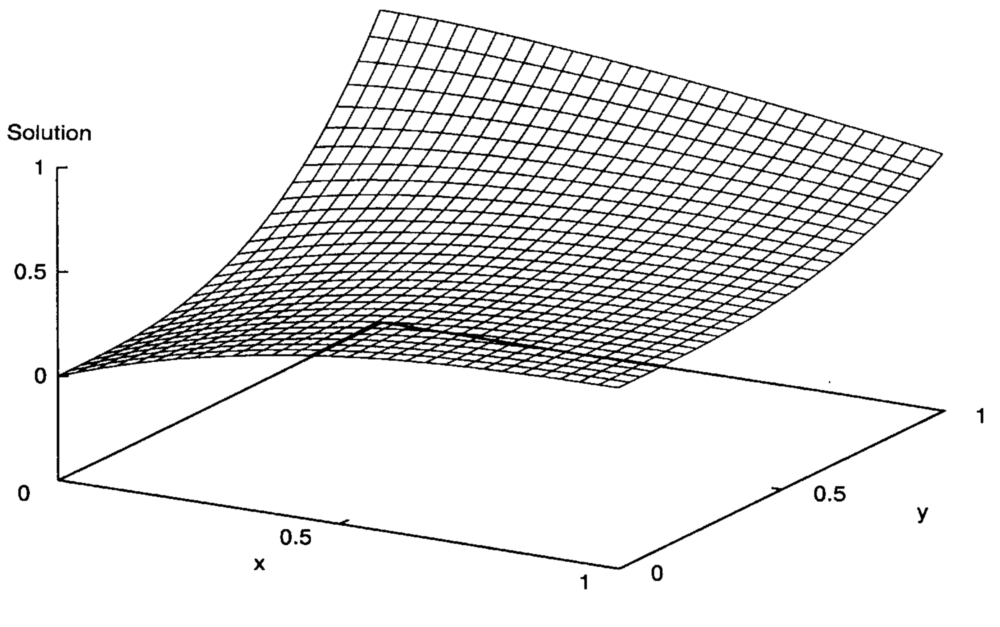

Fig. 8. Exact solution of PDE problem 5.

form given by (23)

$$
\begin{aligned}
A(x,y) = &(1-x)y^3 + x(1+y^3)e^{-1} \\
&+ (1-y)x(e^{-x} - e^{-1}) \\
&+ y[(1+x)e^{-x} - (1-x-2xe^{-1})].
\end{aligned}
$$

Fig. 9 presents the deviation $\Delta\Psi(x,y)$ of the obtained solution at the 100 grid points that were selected for training while Fig. 10 displays the deviation at 900 other points of the equation domain. It clear that the solution is very accurate and the accuracy remains high at all points of the domain.

2) Problem 6:

$$\nabla^2 \Psi(x,y) = (2-\pi^2 y^2)\sin(\pi x) \tag{33}$$

with $x,y \in [0,1]$ and with mixed BC's: $\Psi(0,y) = 0, \Psi(1,y) = 0$ and $\Psi(x,0) = 0, (\partial/\partial y)\Psi(x,1) = 2\sin(\pi x)$. The analytic solution is $\Psi_a(x,y) = y^2\sin(\pi x)$ and is presented in Fig. 11. The trial neural form is specified

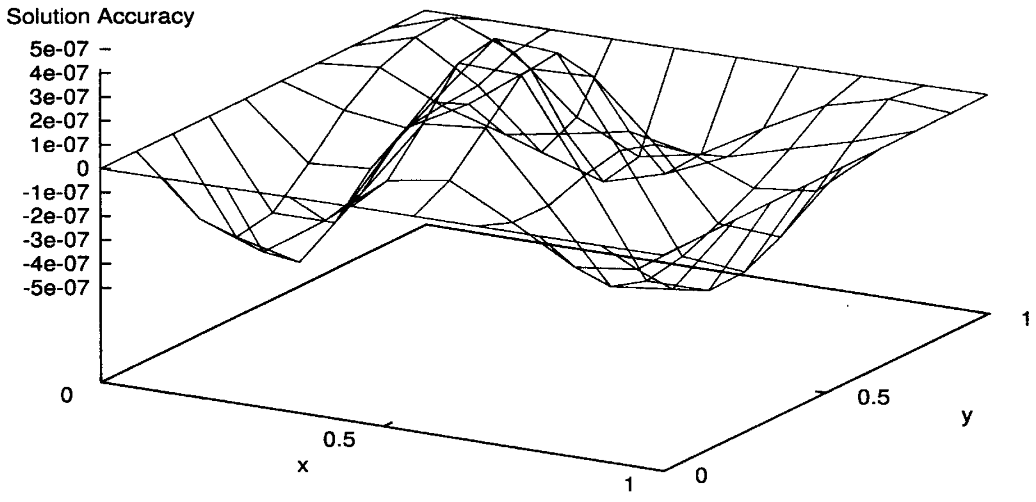

Fig. 9. Problem 5: Accuracy of the computed solution at the training points.

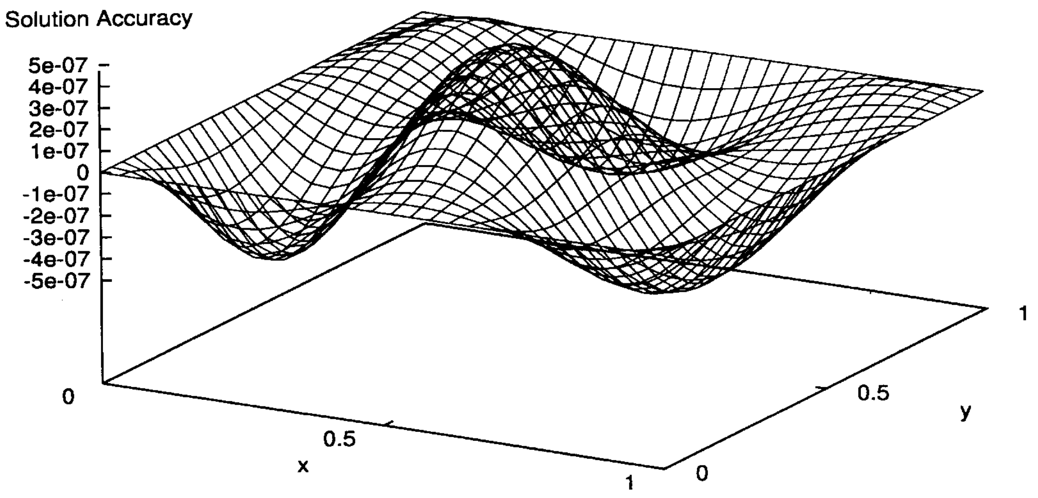

Fig. 10. Problem 5: Accuracy of the computed solution at the test points.

according to (24)

$$\Psi_t(x,y) = B(x,y) + x(1-x)y\left[N(x,y,\vec{p}) - N(x,1,\vec{p}) - \frac{\partial N(x,1,\vec{p})}{\partial y}\right] \tag{34}$$

where $B(x,y)$ is obtained by direct substitution in (25). The accuracy of the neural solution is depicted in Figs. 12 and 13 for training and test points, respectively.

3) Problem 7: This is an example of a *nonlinear* PDE

$$\nabla^2\Psi(x,y) + \Psi(x,y)\,\frac{\partial}{\partial y}\,\Psi(x,y) = \sin(\pi x)(2 - \pi^2 y^2 + 2y^3\sin(\pi x)) \tag{35}$$

with the same mixed BC's as in the previous problem. The exact solution is again $\Psi_a(x,y) = y^2\sin(\pi x)$ and the parameterization of the trial neural form is the same as in problem 6. No plots of the accuracy are presented since they are almost the same with those of problem 6.

C. Comparison with Finite Elements

The above PDE problems were also solved with the finite element method which has been widely acknowledged as one of the most effective approaches to the solution of differential equations [10]. The used Galerkin finite element method (GFEM) calls for the weighted residuals R_i to vanish at each nodal position i

$$R_i = \int_D G(x,y)\det(J)\,d\xi\,dn = 0 \tag{36}$$

where G is given by (1) and J is the Jacobian of the isoparametric mapping, ξ, η the coordinates of the computational domain and x, y the coordinates of the physical domain. This requirement along with the imposed boundary conditions constitute a set of nonlinear algebraic equations $(R_i = 0)$. The inner products involved in the finite element formulation

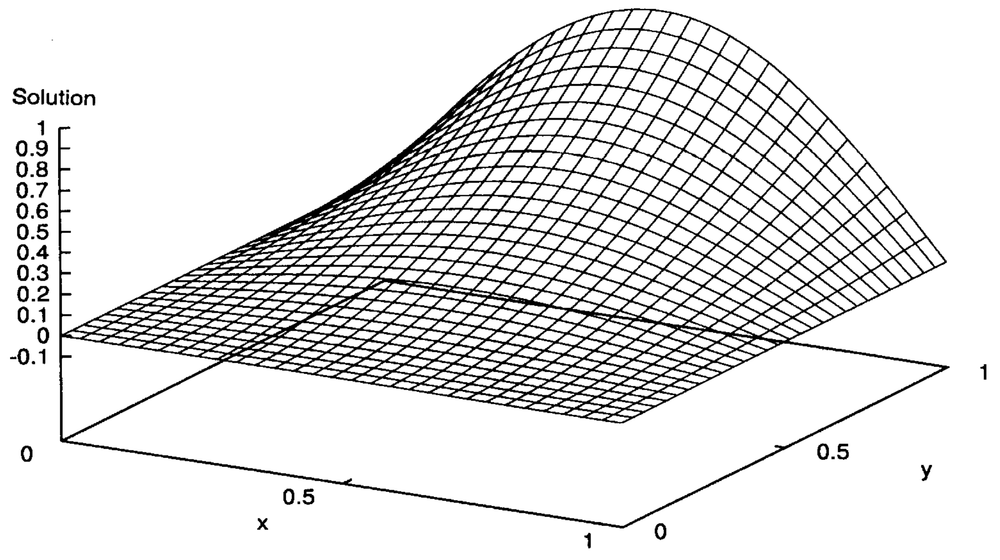

Fig. 11. Exact solution of PDE problems 6 and 7.

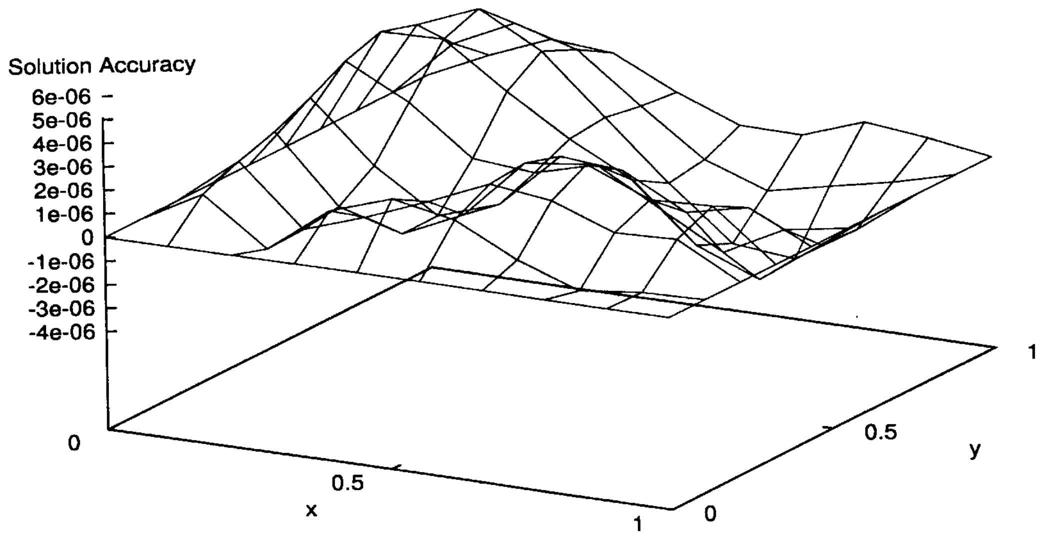

Fig. 12. Problem 6: Accuracy of the computed solution at the training points.

are computed using the nine-node Gaussian quadrature. The system of equations is solved for the nodal coefficients of the basis function expansion using the Newton's method forming the Jacobian of the system explicitly (for both linear and nonlinear differential operators). The initial guess $\vec{\Psi}^{(0)}$ is chosen at random. For linear problems convergence is achieved in one iteration and for nonlinear problems in one to five iterations.

All PDE problems 5–7 are solved on a rectangular domain of 18×18 elements resulting in a linear system with 1369 unknowns. This is in contrast with the neural approach which assumes a small number of parameters (30 for ODE's and 40 for PDE's), but requires more sophisticated minimization algorithms. As the number of employed elements increases the finite element approach requires an excessive number of parameters. This fact may lead to higher memory requirements particularly in the case of three or higher dimensional problems.

In the finite element case, interpolation is performed using a rectangular grid of 23×23 equidistant points (test points). It must be stressed that in the finite element case the solution is not expressed in closed analytical form as in the neural case, but additional interpolation computations are required in order to find the value of the solution at an arbitrary point in the domain. Figs. 14 and 15 display the deviation $|\Psi(x,y) - \Psi_a(x,y)|$ for PDE problem 6 at the training set and the interpolation set of points, respectively. Table I reports the maximum deviation corresponding to the neural and to the finite element method at the training and at the interpolation set of points for PDE problems 5–7. It is obvious that at the training points the solution of the finite element method is very satisfactory and in some cases it is better than that obtained using the neural method. It is also clear that the accuracy at the interpolation points is orders of magnitude lower as compared to that at the training points.

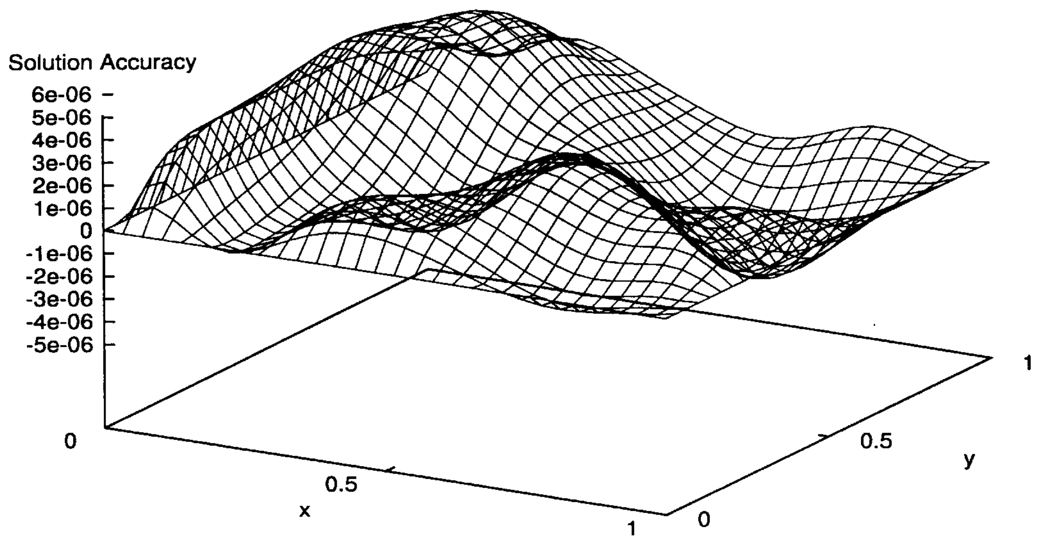

Fig. 13. Problem 6.: Accuracy of the computed solution at the test points.

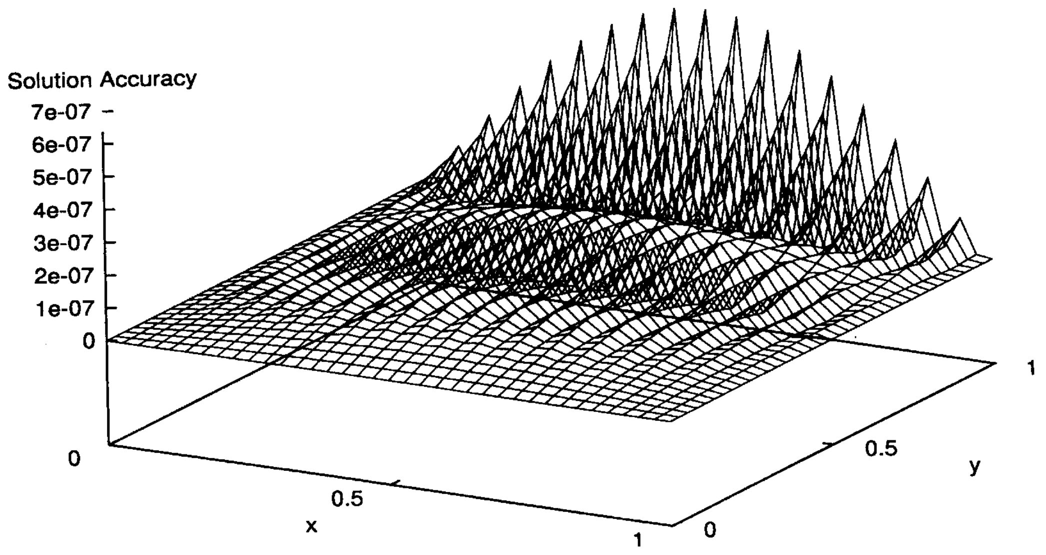

Fig. 14. Problem 7: Accuracy of the FEM solution at the training points.

On the contrary, the neural method provides solutions of excellent interpolation accuracy, since, as Table I indicates, the deviations at the training and at the interpolation points are comparable. It must also be stressed that the accuracy of the finite element method decreases as the grid becomes coarser, and that the neural approach considers a mesh of 10 $\times$ 10 points while in the finite element case a 18 $\times$ 18 mesh was employed.

Fig. 16 provides a plot of the logarithm of the interpolation error with respect to the number of parameters for the neural and the FEM case, respectively for the nonlinear problem 7. The number of parameters in the x-axis is normalized, and in the neural case the actual number of parameters is $20x$, while in the FEM case is $225x$. It is clear that the neural method is superior and it is also obvious that the accuracy of the neural method can be controlled by increasing the number of hidden units. In what concerns execution time, the plots of Fig. 17 suggest that in the neural approach time increases linearly with the (normalized) number of parameters, while in the FEM case, time scales almost quadratically. It must be noted that our experiments have been carried out on a Sun Ultra Sparc workstation with 512Mb of main memory.

For a linear differential equation, accuracy control can be obtained using the following iterative improvement procedure [11]. Consider the differential equation $G\Psi = f$ (with Dirichlet, Neumann, or mixed boundary conditions), where G is a linear differential operator. If Ψ_1 is a first aproximation to the solution of the differential equation it satisfies exactly

$$G\Psi_1 - f = R \tag{37}$$

where R is the residual function. Then if one writes that

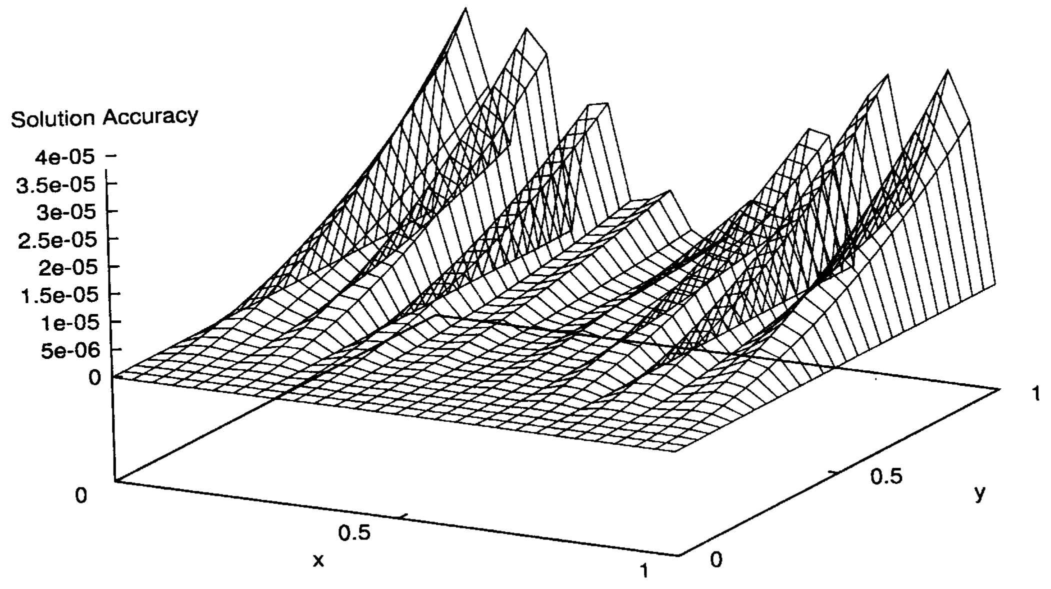

Fig. 15. Problem 7: Accuracy of the FEM solution at the test points.

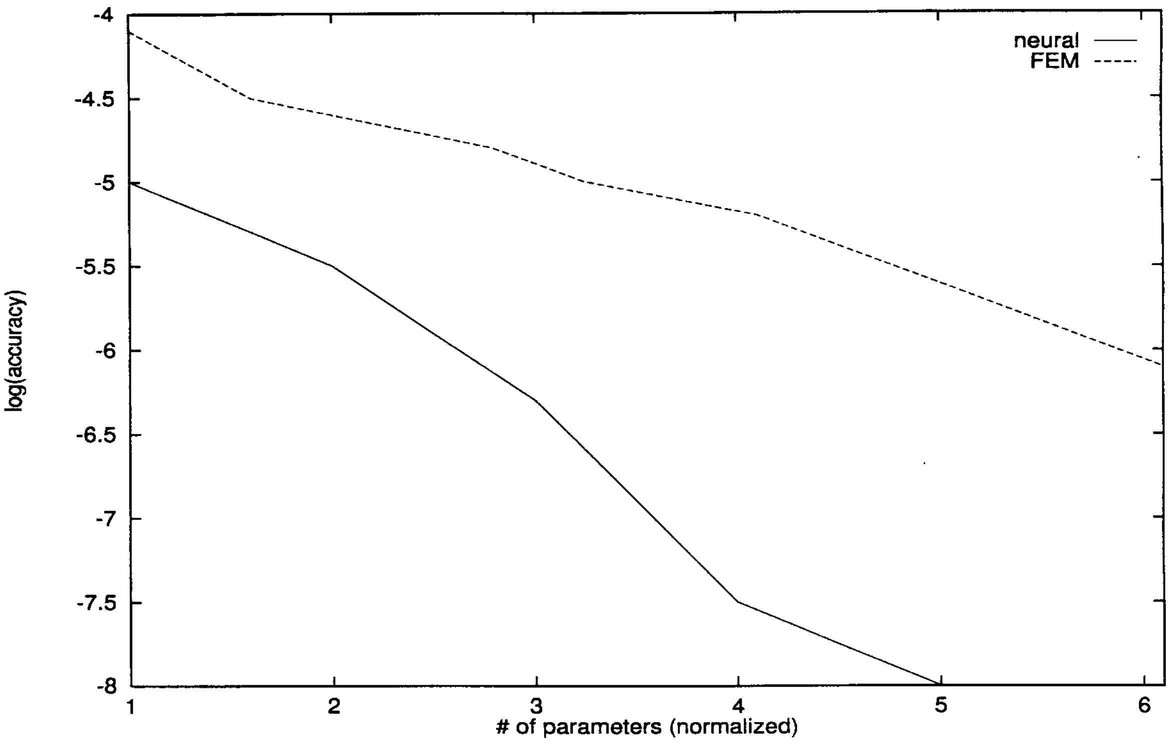

Fig. 16. Plot of logarithm of the maximum convergence error at the interpolation points as a function of the normalized number of parameters for the neural and the FEM approach.

TABLE I
MAXIMUM DEVIATION FROM THE EXACT SOLUTION FOR THE NEURAL AND THE FINITE-ELEMENT METHODS

	Neural Method		*Finite Element*	
Problem No.	Training set	Interpolation set	Training set	Interpolation set
5	5×10^{-7}	5×10^{-7}	2×10^{-8}	1.5×10^{-5}
6	6×10^{-6}	6×10^{-6}	7×10^{-7}	4×10^{-5}
7	1.5×10^{-5}	1.5×10^{-5}	6×10^{-7}	4×10^{-5}

$\Psi = \Psi_1 + \Psi_2$ where Ψ_2 is a correction to Ψ_1, the original problem $G\Psi = f$ can be rewritten (in the case where G is linear) as

$$G\Psi_2 = -R \tag{38}$$

with Ψ_2 obeying null boundary conditions, since Ψ_1 satisfies the boundary conditions exactly. The above process may be repeated more than once to obtain more accurate solutions to the problem.

According to our modeling technique for the case of Dirichlet BC's Ψ_1 is modeled as: $\Psi_1(x) = B(x) + Z(x)N_1(x)$,

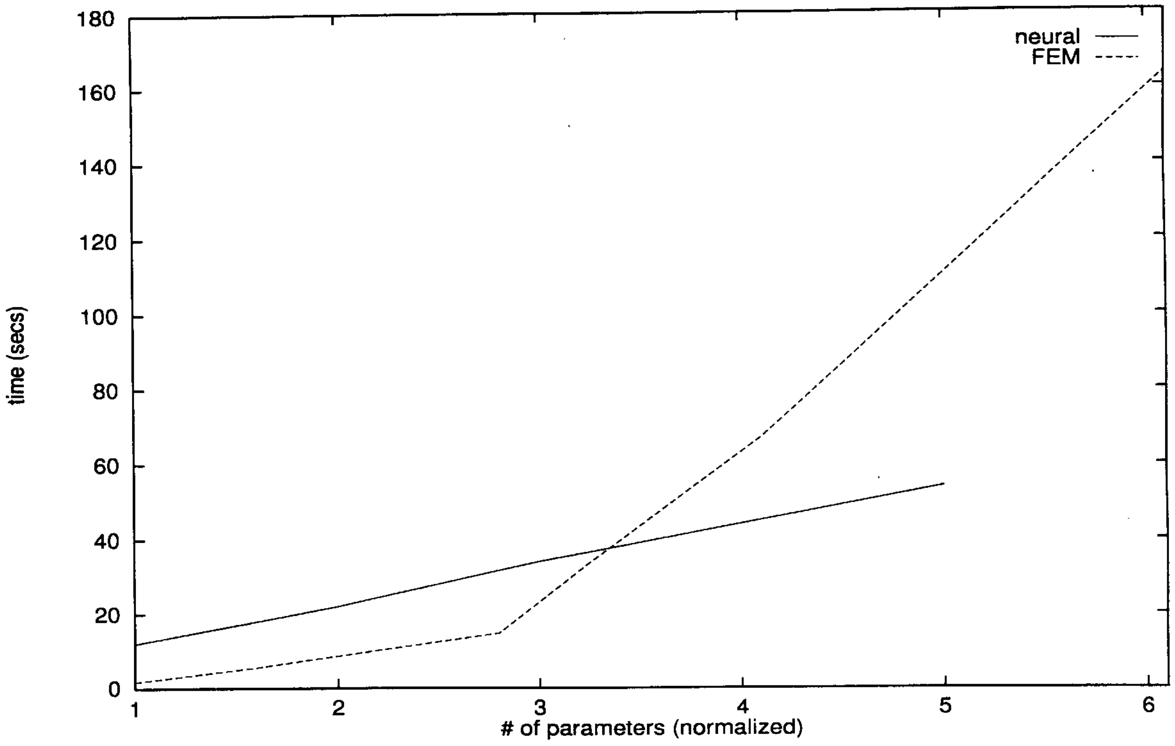

Fig. 17. Plot of the time to converge as a function of the normalized number of parameters for the neural and the FEM approach.

while Ψ_2 and all successive corrections as $\Psi_2 = Z(x)N_2(x)$, The improved approximate solution is now $\Psi(x) = \Psi_1(x) + \Psi_2(x) = B(x) + Z(x)[N_1(x) + N_2(x)]$. For MLP's with one hidden layer and a linear output, the sum of two (or more) of them can be written as a single network of the same type and having a number of hidden nodes equal to the sum of the hidden nodes of the parent networks.

As stated in the introduction, one of the attractive features of our approach is the possibility of effective parallel implementation. In the proposed approach the employement of neural networks makes the method attractive to parallelization. It is well-known that in neural network training the following types of parallelism have been identified: 1) Data parallelism, where the data set (grid points) is split into subsets each one assigned to a different processor and therefore the error values corresponding to different grid points can be computed simultaneously. 2) Spatial parallelism, i.e., the outputs of the sigmoid units in the hidden layer are computed in parallel. This kind of parallelism is better exploited in the case where hardware implementations are used (neuroprocessors) and the speedup obtained is proportional to the number of hidden units. It is also possible to implement a combination of the above kinds of parallelism.

In the case of finite elements parallelism arises mainly in the solution of the linear system of equations. There are also approaches that exploit parallelism in the tasks of mesh generation and finite element construction. Parallelism in finite elements can be exploited mainly at a coarse grain level using general purpose multiprocessor architectures [12]. In general it is much easier to exploit parallelism when using the neural method, since neural networks constitute models with intrinsic parallelization capabilities and various kinds of specialized harware have been developed to exploit this property.

V. Conclusions and Future Research

A method has been presented for solving differential equations defined on orthogonal box boundaries that relies upon the function approximation capabilities of feedforward neural networks and provides accurate and differentiable solutions in a closed analytic form. The success of the method can be attributed to two factors. The first is the employment of neural networks that are excellent function approximators and the second is the form of the trial solution that satisfies by construction the BC's and therefore the constrained optimization problem becomes a substantially simpler unconstrained one.

Unlike most previous approaches, the method is general and can be applied to both ODE's and PDE's by constructing the appropriate form of the trial solution. As indicated by our experiments the method exhibits excellent generalization performance since the deviation at the test points was in no case greater than the maximum deviation at the training points. This is in contrast with the finite element method where the deviation at the testing points was significantly greater than the deviation at the training points.

We note that the neural architecture employed was fixed in all the experiments and we did not attempt to find optimal configurations or to consider architectures containing more than one hidden layers. A study of the effect of the neural architecture on the quality of the solution constitutes one of our research objectives.

Another issue that needs to be examined is related with the sampling of the grid points that are used for training. In the above experiments the grid was constructed in a simple way by considering equidistant points. It is expected that better results will be obtained in the case where the grid density will vary during training according to the corresponding error values. This means that it is possible to consider more training points at regions where the error values are higher.

It must also be stressed that the proposed method can easily be used for dealing with domains of higher dimensions (three or more). As the dimensionality increases, the number of training points becomes large. This fact becomes a serious problem for methods that consider local functions around each grid point since the required number of parameters becomes excessively large and, therefore, both memory and computation time requirements become extremely high. In the case of the neural method, the number of training parameters remains almost fixed as the problem dimensionality increases. The only effect on the computation time stems from the fact that each training pass requires the presentation of more points, i.e., the training set becomes larger. This problem can be tackled by considering either parallel implementations, or implementations on a neuroprocessor that can be embedded in a conventional machine and provide considerably better execution times. Such an implementation on neural hardware is one of our near future objectives, since it will permit the treatment of many difficult real-world problems.

Another important direction of research concerns differential equation problems defined on irregular boundaries. Such problems are very interesting and arise in many real engineering applications. Work is in progress to treat this kind of problems using a trial solution form employing a multilayer perceptron and a radial basis function network, where the latter is responsible for the satisfaction of the boundary conditions, while the former is used for minimizing the training error at the grid ponts. Initial experimental results are very promising.

Moreover, we have already applied our approach to other types of problems of similar nature, as for example eigenvalue problems for differential operators. More specifically, we have considered eigenvalue problems arising in the field of quantum mechanics (solution of the Schrondinger equation) and obtained very accurate results [13].

References

[1] D. Kincaid and W. Cheney, *Numerical Analysis*. Monterey, CA: Brooks/Cole, 1991.

[2] H. Lee and I. Kang, "Neural algorithms for solving differential equations," *J. Comput. Phys.*, vol. 91, pp. 110–117, 1990.

[3] A. J. Meade, Jr., and A. A. Fernandez, "The numerical solution of linear ordinary differential equations by feedforward neural networks," *Math. Comput. Modeling*, vol. 19, no. 12, pp. 1–25, 1994.

[4] ——, "Solution of nonlinear ordinary differential equations by feedforward neural networks," *Math. Comput. Modeling*, vol. 20, no. 9, pp. 19–44, 1994.

[5] L. Wang and J. M. Mendel, "Structured trainable networks for matrix algebra," *IEEE Int. Joint Conf. Neural Networks*, vol. 2, pp. 125–128, 1990.

[6] R. Yentis and M. E. Zaghoul, "VLSI implementation of locally connected neural network for solving partial differential equations," *IEEE Trans. Circuits Syst. I*, vol. 43, no. 8, pp. 687–690, 1996.

[7] D. G. Papageorgiou, I. N. Demetropoulos, and I. E. Lagaris, "Merlin 3.0, A multidimensional optimization environment," *Comput. Phys. Commun.*, vol. 109, pp. 250–275, 1998.

[8] ——, "The Merlin control language for strategic optimization," *Comput. Phys. Commun.*, vol. 109, pp. 250–275, 1998.

[9] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. New York: Wiley, 1987.

[10] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, 4th ed., vol. 1. New York: McGraw-Hill, 1989.

[11] W. L. Briggs, *A Multigrid Tutorial*. Philadelphia, PA: SIAM, 1987.

[12] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, and S. Mittal, "Parallel finite element computation of 3-D flows," *IEEE Comput.*, vol. 26, no. 10, pp. 27–36, 1993.

[13] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural network methods in quantum mechanics," *Comput. Phys. Commun.*, vol. 104, pp. 1–14, 1997.

Isaac Elias Lagaris received the B.Sc. degree in physics from the University of Ioannina, Greece, in 1975. He received the M.Sc. degree in 1977 and the Ph.D. degree in 1981, both from the Physics Department of the University of Illinois, Urbana-Champaign.

He was a Lecturer in the Physics Department of the University of Ioannina for a number of years and since 1994 he has been an Associate Professor in the Department of Computer Science. His research interests include modeling and simulation of classical and quantum systems, high-performance computing, optimization, and neural networks.

Aristidis Likas (S'91–M'96) was born in Athens, Greece, in 1968. He received the Diploma degree in electrical engineering and the Ph.D. degree in electrical and computer engineering, both from the National Technical University of Athens.

Since 1996, he has been with the Department of Computer Science, University of Ioannina, Greece, where he is currently a Lecturer. His research interests include neural networks, optimization, pattern recognition, and parallel processing.

Dimitrios I. Fotiadis received the Diploma degree in chemical engineering from the National Technical University of Athens, Greece, in 1985 and the Ph.D. degree in chemical engineering from the University of Minnesota, Duluth, in 1990.

He is currently a Lecturer at the Department of Computer Science at the University of Ioannina, Greece. He worked As Visiting Scientist in the Department of Chemical Engineering at the Massachusetts Institute of Technology, Cambridge, and as Visiting Researcher in RWTH, Aachen, Germany. He also worked as Research Assistant at the Minnesota Supercomputer Institute and as Managing Director of Egnatia Epirus Foundation, Greece. He has many years of industrial and research experience in high-performance computing, computational medicine, and biomedical engineering. His current research interests include computational medicine and biomedical technology with emphasis on methods and instruments used for diagnosis and therapeutic reasons of various diseases.