Available online at www.sciencedirect.com
SCIENCE DIRECT®
Neurocomputing 63 (2005) 447–463
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Neural network time series forecasting of finite-element mesh adaptation

Larry Manevitz[a,*], Akram Bitar[a], Dan Givoli[b]

[a]*Department of Computer Science, University of Haifa, Haifa, Israel*
[b]*Faculty of Aerospace Engineering, Technion, Israel Institute of Technology, Haifa, Israel*

Accepted 10 June 2004
Communicated by T. Heskes
Available online 9 December 2004

Abstract

Basic learning algorithms and the neural network model are applied to the problem of mesh adaptation for the finite-element method for solving time-dependent partial differential equations. Time series prediction via the neural network methodology is used to *predict* the areas of "interest" in order to obtain an effective mesh refinement at the appropriate times. This allows for increased numerical accuracy with the same computational resources as compared with more "traditional" methods.

Keywords: Neural networks; Mesh adaptation; Time series prediction; Finite-element method; Time-dependent PDEs

1. Introduction

The finite-element method (FEM) [1,11] is the most effective numerical techniques for solving various problems arising from mathematical physics and engineering. Actually, it is the most widely used numerical techniques for solving problems which are described by partial differential equations (PDEs).

*Corresponding author.
E-mail addresses: manevitz@cs.haifa.ac.il (L. Manevitz), akram@il.ibm.com (A. Bitar), givolid@aerodyne.technion.ac.il (D. Givoli).

0925-2312/$ - see front matter
doi:10.1016/j.neucom.2004.06.009

The time-dependent PDEs arise in modeling numerous phenomena in science and engineering, and are tend to be divided into two categories: hyperbolic and parabolic [6]. The hyperbolic PDE is used for transient and harmonic wave propagation in acoustics and electromagnetic, and for transverse motions of membranes; the basic prototype of the hyperbolic PDE is the wave equations. The parabolic PDE is used for unsteady heat transfer in solids, flow in porous media and diffusion problems; the basic prototype parabolic PDE is the heat equations.

The main idea behind the finite-element method is to reduce a continuous physical problem with infinitely many unknown field values to a finite number of unknowns by discretizing the solution region into elements [11]. Then, the values of the field at any point can be approximated by interpolation functions within every element in terms of the field values at specified points called nodes. Nodes are located at the element vertices where adjacent elements are connected. The approximation of the solution on each element should be consistent with neighboring elements.

Several approaches can be used to transform the continuous physical formulation of the problem to its finite-element discrete analogue. For PDEs, the most popular method of their finite element formulation is the *Galerkin* method [1].

In time-dependent problems, e.g. hyperbolic equations, the areas of "interest", i.e. the areas with high gradient, are propagated through the domain. Therefore, the mesh choice should be dynamic and varying with time. For example, when solution of hyperbolic problems involves a shock wave, which propagates through the mesh the location of the shock vicinity keeps changing in time. Thus, one wants to have the mesh more refined around the area of the shock vicinity and less refined elsewhere. Another example is the problem of fluid flow in a cavity, where flow cells are generated and undergo continuous changes in their shapes and size as time proceeds [9].

This means that the mesh adaptation is a crucial part for the efficient computation of the numerical method. In order to achieve an optimal mesh (one which the solution error is low relative to the number of nodes in the mesh), the mesh choice should be dynamic and varying with time.

In current usage, the method is to use indicators (e.g. gradients) from the solution at current time to identify where the mesh should be modified (i.e. where it should be refined and where it can be made coarser) at the next time stage. However, this suffers from the obvious defect that one is always operating one step behind. In other words, if the areas of interest are propagated, then one may be always refining behind the most interesting phenomena.

In this paper, we present a new approach for solving the mesh adaptation problem. Our approach looks at this as a special instance of a control problem and uses the neural network to solve it in a similar way that such networks have been used to predict time series (see [16,21,20]).

The neural network is a universal approximator [4,8,12,10,19] that learns from the past to predict the future values. It receives, in some form, as input the "areas of interest" at recent times and predicts the "areas of interest" at the next time stage. Using this predictor we can forecast the position of the "action" and refine the mesh accordingly.

The methods and experiments developed in this work are for one and two dimensional hyperbolic equations (wave equations), but seem naturally extendible to higher dimensional problems and other time-dependent PDEs.

2. Time series neural networks

Neural networks (NNs) [7,3] are a biologically inspired model, which tries to simulate the network of neurons in the human brain. The artificial neural networks consist of simple calculation elements, called *neurons*, and weighted connections between them called *weights*.

The most common NN model is the supervised-learning, feed-forward net-work. Typically, the feed-forward network contains three types of processing units (neurons), input units, output units and hidden units, organized in a hierarchy of layers: input layer, hidden layers and output layer. The data from input layer or hidden layers are multiplied by the weights associated with a next layer unit and summed together before processing by the unit.

A neural network can be trained to perform complex functions by adjusting the values of the connections (weights) between the elements (neurons) according to one of several training algorithms. *Back-propagation* is the most popular training algorithm in which the training data propagated forward through the network and the output data are calculated. The error between the expected output and the calculated output is computed. Then a minimization procedure is used to adjust the weights between two connection layers starting backwards from the output layer to input layer. There is a number of variations of minimization procedures that are based on different optimization methods, such as *gradient descent* [7,3], *Quasi-Newton* [18,3] and *Levenberg–Marquardt* [3,13,14] methods. The forward and backward propagation are executed iteratively over the training set until a stopping criterion is met. (For example, when the average squared error between the network outputs and the desired outputs reaches an acceptable value. The stopping criteria used in this paper is based on a test set; see the description in Section 3.2.)

Time series is well suited for data where past values in the series may influence future values. In this case, a future value is a nonlinear function of its past m values:

$$x(n) = f(x(n-1), x(n-2), \ldots, x(n-m)). \tag{1}$$

This means, that it is necessary to fit a function x through its past values in order to extrapolate this function to the near future.

Since a three layer feed-forward network can approximate any reasonable function after a suitable amount of training [4,12,10] it can be applied to this problem, by submitting discrete values of this function to the network. The net is then expected to learn the function rule by the training algorithm. The behavior of the network is changed by modifying the values of the weights.

Therefore, we can use the back-propagation network as a nonlinear model that can be trained to map past and future values of a time series. This method is called

time series prediction with neural networks and is used in the forecasting of financial markets [2] (e.g. to predict whether stock market rates will rise or fall).

3. Applying NNs to time-dependent PDEs

For many PDEs critical regions should be subject to local mesh refinement. The critical regions are the regions for which the local gradient shows bigger changes. In order to meet this problem, the FEM adaptation process makes a local refinement in those areas, thus the ensuing mesh may be more gross in the other areas. In time-dependent problems, the mesh refinement should be dynamic and depends on the error estimation in each time stage.

In current usage, most of the error estimate methods take into account the solution gradient; in this work we developed a new approach based on predicting the future gradient value of the solution and we used this as the refinement criteria.

In dynamic systems such as hyperbolic equations, the areas of interest, i.e. the areas with high gradient, are propagated through the domain. Therefore, for each mesh element the future gradient value is influenced by the past gradient values of the element and of its direct neighbors. In other words, the future gradient value can be considered as a nonlinear function of its past values. This is a proper time series problem and the time series neural networks can be used to predict the future gradient values. Fig. 1 illustrates this concept.

The upper domain in Fig. 1 shows the mesh at time t_{n-1} and the lower one shows the mesh at time t_n (at this time stage some elements may be refined). The neural network receives, as input, the gradient of element e and its neighbors $e1$, $e2$ and $e3$ at time t_{n-1} and t_n. (Should one of these elements be refined at time t_{n-1} then at time t_n

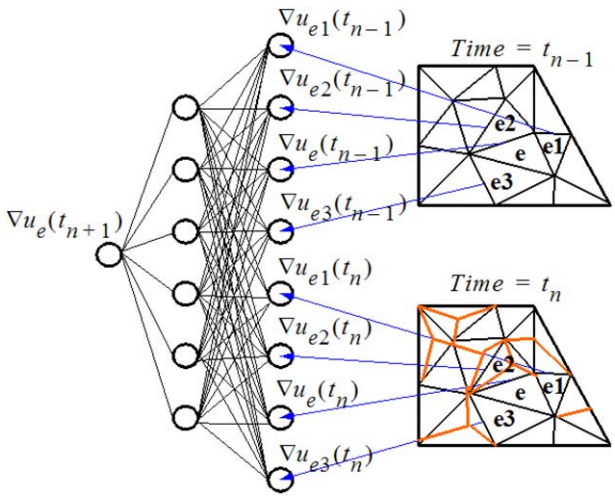


Fig. 1. Using neural networks to forecast future FEM gradient values. Gradient values for the two previous times and all neighboring elements are used as input to the network.

Table 1
One dimension examples

Example 1

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leqslant x \leqslant 10$$
$$u(0,t) = 0 \quad \text{and} \quad u(10,t) = 0$$
$$\frac{\partial u}{\partial t}(x,0) = 0 \quad \text{and} \quad u(x,0) = \begin{cases} 1 - |1-x| & 1 \leqslant x \leqslant 2 \\ 0 & \text{otherwise} \end{cases}$$

Time:12, Time step:0.08, Threshold for refinement = 0.08 (gradient)

Method	Number of elements		L^2 error norm		L^∞ error norm	
	Initial	Final	Max	Average	Max	Average
NN modifier	10	70	0.15756	0.0869	0.1653	0.1056
Standard modifier	10	70	0.1826	0.1022	0.1914	0.1222
No adaptation	10	10	2.5332	1.0053	3.4708	1.0643

Improvement: L^2 error norm = 15%, L^∞ error norm = 13.6%

Example 2

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leqslant x \leqslant 25$$
$$u(0,t) = 0 \quad \text{and} \quad u(25,t) = 0$$
$$\frac{\partial u}{\partial t}(x,0) = 0 \quad \text{and} \quad u(x,0) = \begin{cases} \exp\left(\frac{-(x-5)^2}{2}\right) & 0 \leqslant x \leqslant 10 \\ 0 & \text{otherwise} \end{cases}$$

Time:25, Time step:0.12, Threshold for refinement = 0.2 (gradient)

Method	Number of elements		L^2 error norm		L^∞ error norm	
	Initial	Final	Max	Average	Max	Average
NN modifier	15	98	0.4423	0.1928	0.5190	0.2342
Standard modifier	15	91	0.6671	0.2686	0.8230	0.3142
No adaptation	15	15	1.4622	0.6985	1.6288	0.6456

Improvement: L^2 error norm = 28%, L^∞ error norm = 25%

Comparison between FEMs run with (i) The neural network predictor of the gradient measure. (ii) "Standard" refinements using the gradient measure. (iii) No adaptation.

one uses the average of all the children of the original (t_{n-1}) element.) Its output is the predicted gradient value of element e at time t_{n+1}.

(Should one of these elements be refined at time t_{n-1} then at time t_n one uses the average of all the children of the original (t_{n-1}) element.)

Thus there are two steps to our methodology: (a) training the neural network to predict the indicators (at least) one step in advance and (b) applying the indicators to the refinement in the FEM solution.

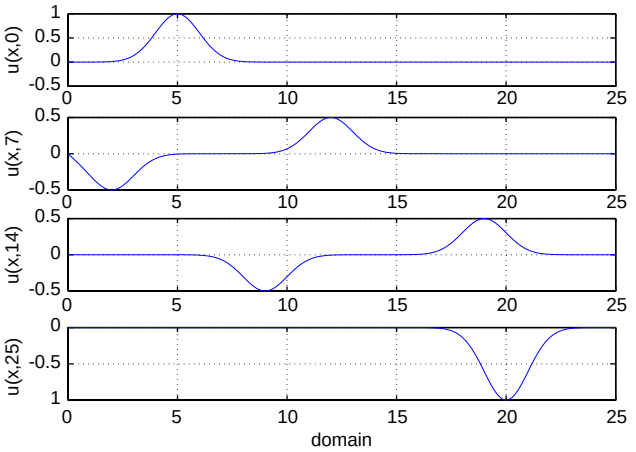


Fig. 2. The analytic solution of Example 2 of Table 1. Here, *domain* denotes the interval $0 \leqslant x \leqslant 25$ where the given PDE is defined, and $u(x,t)$ denotes the PDE solution at time t of the point x.

In our experiments we examine the numerical results of applying this procedure using the FEM on different time-dependent PDE problems using different parameters for the NN algorithm and comparing this with (i) FEM with no adaptation and (ii) FEM using the "standard" adaptation via the current gradient indicator.

3.1. Measures of solution quality

To measure the quality of FEM solution, we calculate both of the L^2 and L^∞ error norm per value in each time stage. (Here u is the analytic solution and u_h is the numerically computed solution.)

$$L^2\text{error/value} = \frac{\sum_{\text{nodes}} |u(\text{node}) - u_h(\text{node})|^2}{\sum_{\text{nodes}} |u(\text{node})|^2}, \tag{2}$$

$$L^\infty\text{error/value} = \frac{\max_{\text{nodes}} |u(\text{node}) - u_h(\text{node})|}{\max_{\text{nodes}} |u(\text{node})|}. \tag{3}$$

The L^2 error norm measures the error in the entire solution space (average error), and the L^∞ measures the maximum error occurring in the solution.

The analytic solution is very important in order to measure the precise error in the solution. When the analytic solution of a PDE is not available, we do one calculation with a very small time step and a very fine constant mesh, and then we use this solution as a reference to the analytic one. In all cases, we report at the end of each experiment the average of L^2 and L^∞ error per value over all the time space.

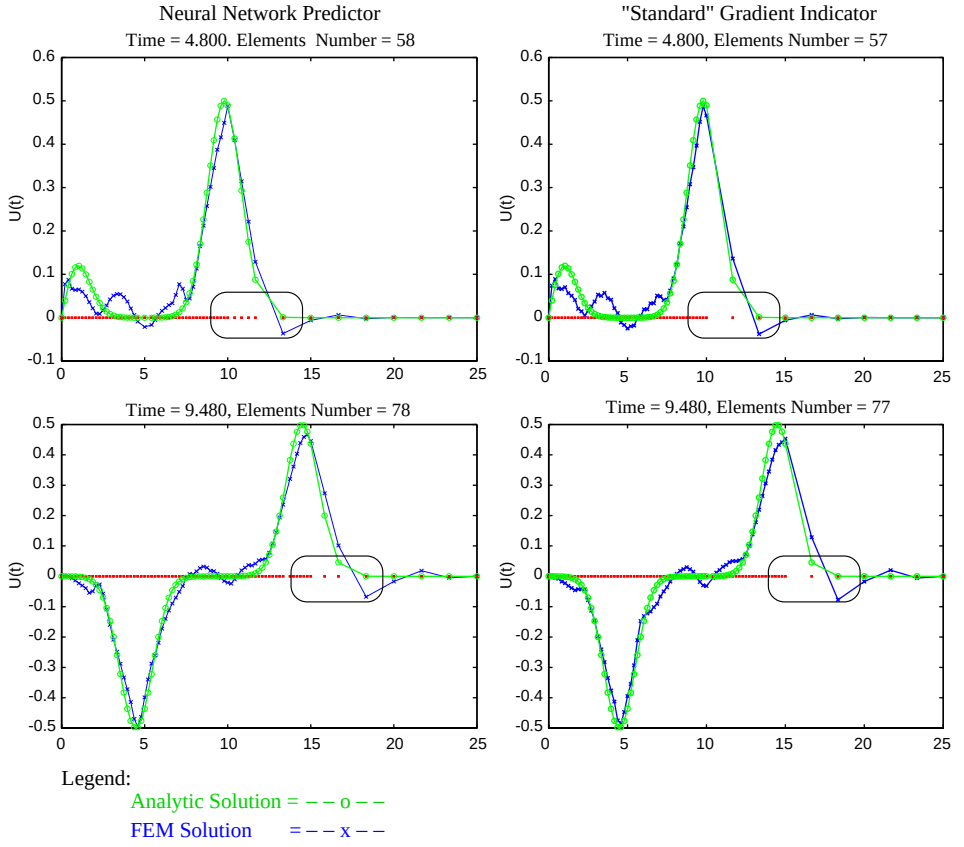


Fig. 3. Results from the FEM on a one dimensional wave equation (see Example 2 in Table 1). The left figures are refined with the NN predictor. The analytic solution is also indicated. The right figures are refined with the "standard" gradient indicator. Compare the segments of the curves on the left (enclosed rounded rectangles) with the corresponding ones on the right to see how the NN predictor focuses the resources in the correct places.

3.2. Neural network architecture and training

In this study, the MATLAB's Neural Network Toolbox [5] was used for designing and training the neural network; and the MATLAB's Partial Differential Equation Toolbox [15] was used for defining, and solving the two dimensional PDEs problems. For one dimensional problems, we used a FEM solver which we developed especially for our research needs. In the examples tested so far the results are fairly dramatic. First, using the Levenberg–Marquardt training algorithm [3,17] the training was both quite swift and exceptionally accurate. Second, the improvement in the FEM numerical results (as compared with the "standard" gradient adaptive method)

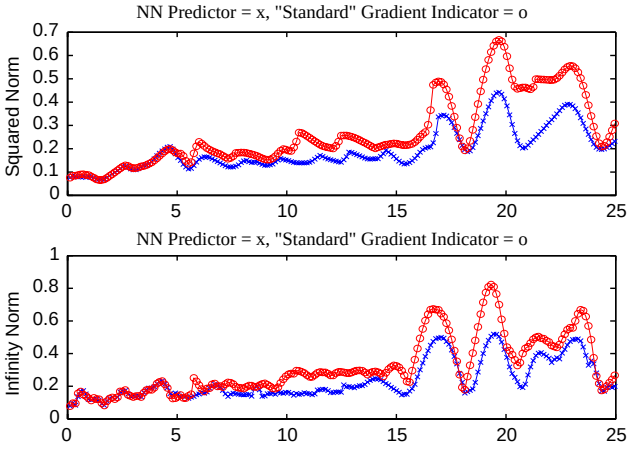


Fig. 4. The L^2 error norm and the L^∞ error norm displayed over time of Example 2 of Table 1. Other one dimensional examples show the same general behavior of the error norms over time; in particular, the NN error results are never higher than the errors generated by a standard modifier.

reached as high as 25% on some examples; and never fell significantly below the standard method. (The variance in the improvement depends on the shape of the wave; and is to be expected. That is, for some waves it is more important to predict the gradient than others.)

We used two different networks, one for boundary elements and one for interior elements. The architecture of networks was six input units for boundary elements network, and eight input units for interior elements networks (corresponding to the value of the gradient of the element and its two neighbors in the current and previous times); and for both networks six hidden units (with hyperbolic tan-sigmoid transfer function), and one output unit (with linear transfer function) that gave the prediction of the output value. See Fig. 1.

In order to make the training more efficient: (a) we normalized the input and output data between the values 0 and 1; and (b) we divided the training data into two disjoint subsets: *training set* and *testing set*. The training set is used for computing the gradient and updating the network weights and biases. The testing on the validation set is monitored during the training process; as long as the error decreases, training continues. When the error begins to increase, the net begins to overfit the data and loses its ability to generalize; at this point the training is stopped.

To generate training data: (a) we calculated the solution on the initial non-dynamic mesh over all the given time space; (b) we chose a random collection of time stages, and used all the elements at each of these time stages together with their appropriate gradients as training examples. The training data consisted of more than 800 examples (about 600 for the training set and 200 for the testing set).

Table 2
FEM mesh refining and coarsening examples

Example 3

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leqslant x \leqslant 12$$
$$u(0,t) = 0 \quad \text{and} \quad u(12,t) = 0$$
$$\frac{\partial u}{\partial t}(x,0) = 0 \quad \text{and} \quad u(x,0) = \begin{cases} \exp(x)\sin(\pi x) & 1 \leqslant x \leqslant 2, \\ 0 & \text{otherwise,} \end{cases}$$

Time:20, Time step:0.09, Threshold for refinement = 2 (gradient)

Method	Number of elements		L^2 error norm		L^∞ error norm	
	Initial	Final	Max	Average	Max	Average
NN modifier	30	90	0.4146	0.2678	0.5097	0.3028
Standard modifier	30	90	0.4468	0.2935	0.5891	0.3201

Improvement: L^2 error norm = 8.7%, L^∞ error norm = 5.4%

Example 4

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leqslant x \leqslant 12$$
$$u(0,t) = 0 \quad \text{and} \quad u(12,t) = 0$$
$$\frac{\partial u}{\partial t}(x,0) = 0 \quad \text{and} \quad u(x,0) = \begin{cases} \exp(x)\sin(\pi x) & 1 \leqslant x \leqslant 2, \\ 0 & \text{otherwise,} \end{cases}$$

Time:20, Time step:0.09
Threshold for refinement = 2 (gradient), Threshold for coarseness = 0.3 (gradient)

Method	Number of elements		L^2 error norm		L^∞ error norm	
	Initial	Final	Max	Average	Max	Average
NN modifier	30	90	0.7155	0.3710	0.5949	0.3478
Standard modifier	30	73	0.7061	0.4085	0.7330	0.3799

Improvement: L^2 error norm = 9.1%, L^∞ error norm = 8.4%

4. One dimensional wave equations

4.1. Mesh refining

We have run the NN modifier over a variety of initial conditions for the one dimensional wave equation. In all cases, the NN predictor was extremely accurate. Training took about 117 epochs to reach the test set stopping criteria for the interior elements prediction network. (At that time both the training error and the testing

error were very small—about 0.00024 for the training set.) Results for the boundary elements were similar.

When applying this modifier to the FEM mesh, the numerical improvement over the "standard" gradient modifier varied from no significant improvement to an improvement of more than 25% (both in the L^2 error norm and in the L^∞ error norm).

For a sample example, where the initial condition of the wave is a Gaussian, see Example 2 in Table 1. The analytic solution is well known for these types of problems and it depends on the initial and boundary conditions. The wave splits into two waves (with the same width but half the height) that travels to the left and to the right with speed $c = 1$. When such a traveling wave reaches the edge it turns over and returns upside down (see Fig. 2). The NN modified solution and the "standard" gradient modifier are displayed in Fig. 3. Observing the areas indicated in the figure, one can see that applying the NN modifier has caused the mesh to be modified so as to increase the computational effectiveness. Looking at the refinement markings (in red or dots on the x-axis); one can see that, as suggested by our theory, the NN is keeping pace with the development of the solution, whereas the "standard" method is always one-step behind, which at critical locations causes increased numerical error.

Since these examples have analytic solutions, we can keep track of the actual numerical errors of each of the methods. In Fig. 4 we track the errors (both in L^2 error norm and in the L^∞ error norm).

The wave of the analytic solution reaches the edge at time $t = 15$ (see Fig. 2) and it starts to turn over and returns upside down. We can see the affect of this clearly in Fig. 4. At time $t = 15$ both error measures (L^2 and L^∞) of both methods start to

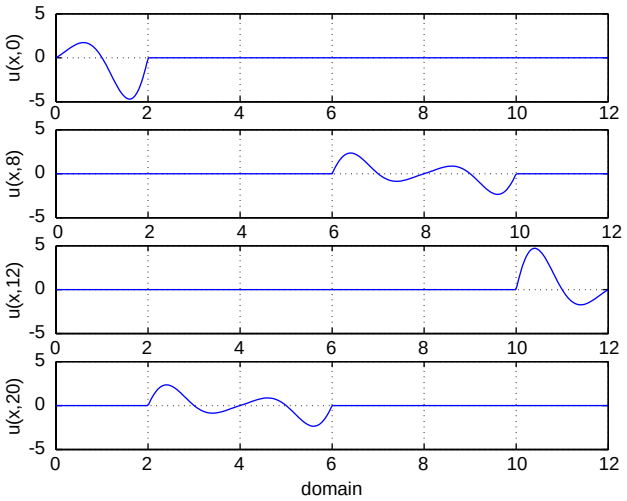


Fig. 5. The analytic solution for the wave equation of Examples 3 and 4 (see Table 2).

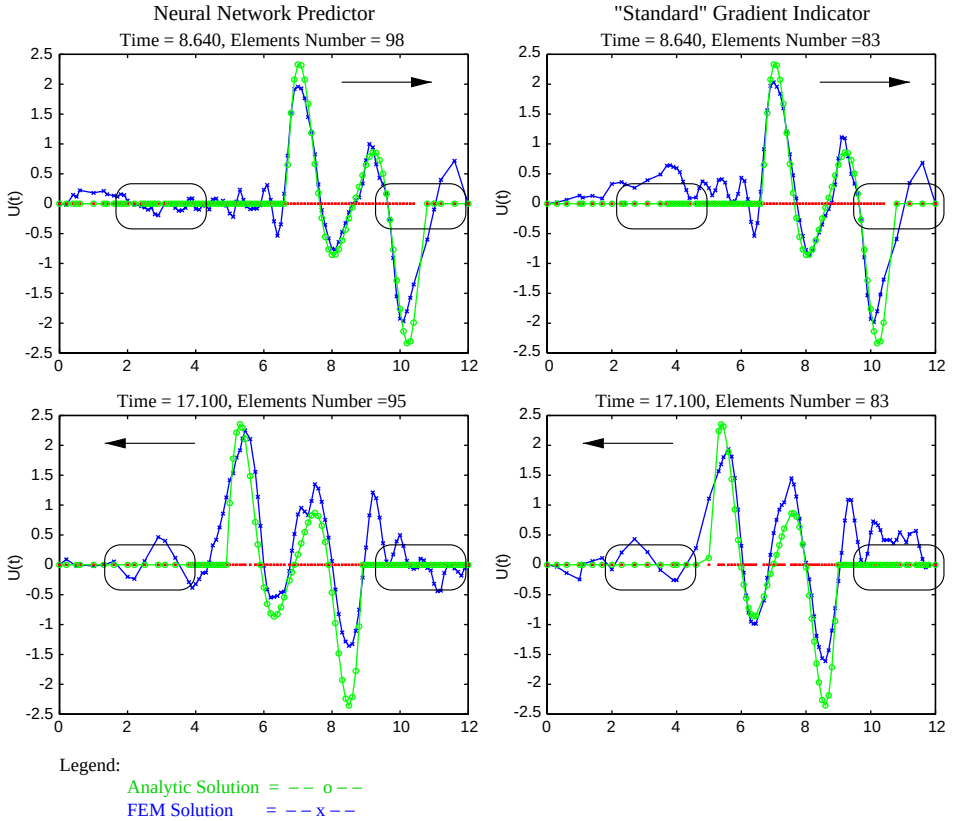


Fig. 6. FEM mesh refining and coarsening results. The left figures are modified (refined and coarsened) with the NN predictor. The analytic solution is also indicated. The right figures are modified with the "standard" gradient indicator. Compare the segments of the curves on the left (enclosed rounded rectangles) with the corresponding ones on the right to see how the NN predictor focuses the resources in the correct places.

increase. Note that while the NN method is better at *all* times than the "standard" method; during the critical period $15 \leqslant t \leqslant 25$ there is a very large improvement.

4.2. *Mesh coarsening*

We can use our NN modifier for refining and coarsening the FEM mesh at the same time (when the gradient of an element is bigger than a given refinement threshold we decide to refine the mesh, and when it is less than a given coarseness threshold we decide to coarsen the mesh). In this section we present two sample examples with the same wave equation in Table 2. In Example 3, we have used the NN modifier for refining the FEM mesh and in Example 4 we have used it for both refining and coarsening the FEM mesh simultaneously.

Table 3
Two dimension examples

Example 5

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad -1 \leqslant x \leqslant 1, \quad -1 \leqslant y \leqslant 1$$

$$u(-1, y, t) = 0 \quad \text{and} \quad u(1, y, t) = 0 \quad \text{for} \quad -1 \leqslant y \leqslant 1$$

$$u(x, -1, t) = 0 \quad \text{and} \quad u(x, -1, t) = 0 \text{ for } -1 \leqslant x \leqslant 1$$

$$\frac{\partial u}{\partial t}(x, y, 0) = 0 \quad \text{and} \quad u(x, y, 0) = \begin{cases} 15x(x+1)y(y+1) & -1 \leqslant x \leqslant 0, \ -1 \leqslant y \leqslant 0 \\ 0 & \text{otherwise} \end{cases}$$

Time:3, Time step:0.05, Threshold for refinement = 1 (gradient)

Method	Number of elements		Average L^2 Error	Average L^∞ Error
	Initial	Final		
NN modifier	28	803	0.4057	0.4846
Standard modifier	28	803	0.4314	0.5029

Improvement: L^2 error norm = 6%, L^∞ error norm = 3.6%

Example 6

$$\tfrac{\partial^2 u}{\partial t^2} = \tfrac{\partial^2 u}{\partial x^2} + \tfrac{\partial^2 u}{\partial y^2}, \quad -1 \leqslant x \leqslant 1, -1 \leqslant y \leqslant 1$$

$$u(-1, y, t) = 0 \quad \text{and} \quad u(1, y, t) = 0 \text{ for } -1 \leqslant y \leqslant 1$$

$$u(x, -1, t) = 0 \quad \text{and} \quad u(x, -1, t) = 0 \text{ for } -1 \leqslant x \leqslant 1$$

$$\tfrac{\partial u}{\partial t}(x, y, 0) = 3 \sin(\pi x) \exp(\sin(\tfrac{\pi}{2y})) \quad \text{and} \quad u(x, y, 0) = \arctan(\cos(\tfrac{\pi}{2x}))$$

Time:3, Time step:0.08, Threshold for refinement = 2.2 (gradient)

Method	Number of elements		Average L^2 Error	Average L^∞ Error
Method	Initial	Final		
NN modifier	28	246	0.2962	0.3359
Standard modifier	28	232	0.3256	0.3807

Improvement: L^2 error norm = 9%, L^∞ error norm = 11%

Comparison between FEMs run with (i) The neural network predictor of the gradient measure. (ii) "Standard" refinements using the gradient measure.

Fig. 5 presents the analytic solution of the given wave equation, Fig. 6 presents the NN modified solution and the "standard" gradient modifier for Example 4.

In both examples we can see that the NN modifier results are an improvement over the "standard" gradient modifier. In Example 3, we reach an improvement of 8.7% in the L^2 error norm and 5.4% in the L^∞ error norm (see Table 2). In Example 4, we can see that the improvement rises from 8.7% to 9.1% (in the L^2 error norm) and from 5.4% to 8.4% (in the L^∞ error norm). (See Table 2.)

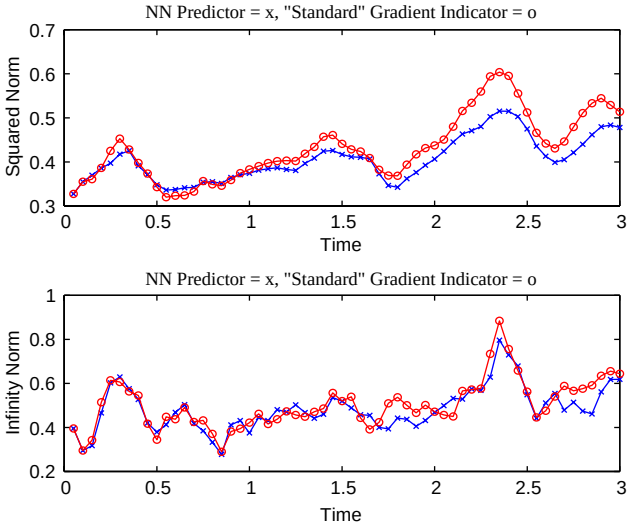


Fig. 7. The L^2 error norm and the L^∞ error norm displayed over time of Example 5 in Table 3 (a two dimensional example).

Comparing the graphs in Fig. 6, we can observe that the NN modifier has placed the resources in the correct places and this explains the enhancement that we have in the results.

5. Two dimensional wave equations

We have run our method over a variety of two dimensional wave equations. In all cases, the NN modifier showed a clear improvement with rates varying from around 2% to 20%. The variance in the improvement depends on the initial conditions (the shape of the wave and its velocity). In all cases the gradient prediction was extremely accurate.

Two of these examples are presented in Table 3 where the improvement in the FEM numerical results were 6% (Example 5) and 9% (Example 6) in the L^2 norm, and 3.6% (Example 5) and 11% (Example 6) in the L^∞ norm. (From Fig. 7 one can also see that the error behavior of the NN modifier is better than the standard method at all times.)

Notice in particular the graphs in Fig. 8. We can see that our method has caused the mesh to be modified so as to increase the computational effectiveness. Looking at the refinement markings (encircled near the area near (0,0)), we can see that the "standard" method trails behind our method. (Compare with the analytic graph as well in the figure.)

Finally, we applied the NN modifier to a PDE which simulates throwing a stone into the middle of a square lake. In this example, we took the boundary conditions

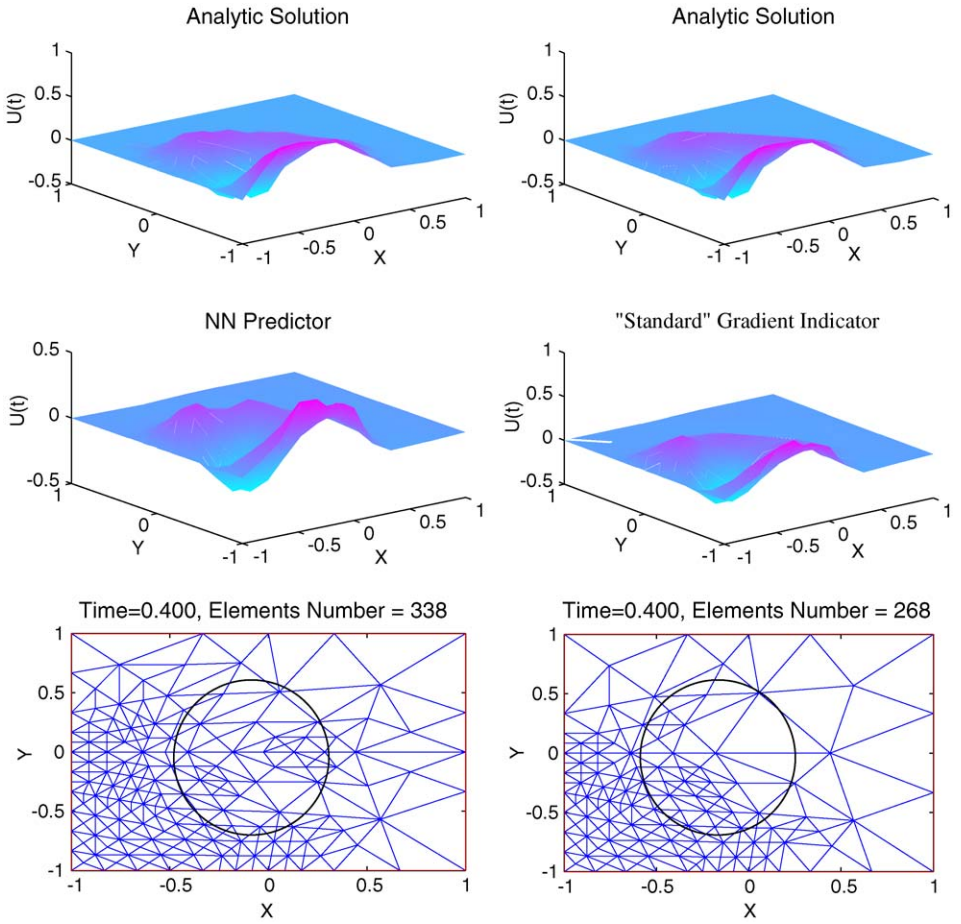


Fig. 8. Results from FEM on two dimensional wave equation (see Example 5 in Table 3). The left figures are refined with the NN predictor. The right figures are refined with the "standard" gradient indicator.

on the square boundary to be $u = 0$ and the initial conditions to be zero everywhere except at one point in the middle where we took $u = 1$ (which models the stone hitting the lake at time $t = 0$).

Throwing a stone into the middle of the lake generates a wave that propagates outwards and reaches the lake boundary, then it reflects and turns over back to the interior of the lake. After the initial wave front reaches the boundary a complex wave pattern is formed due to the difference in geometry between these fronts (circles) and the boundary (square). Fig. 9 shows the FEM numerical solution of the PDE in different times. From Fig. 9 we can clearly see that the NN modifier refined the FEM mesh according to the wave motion.

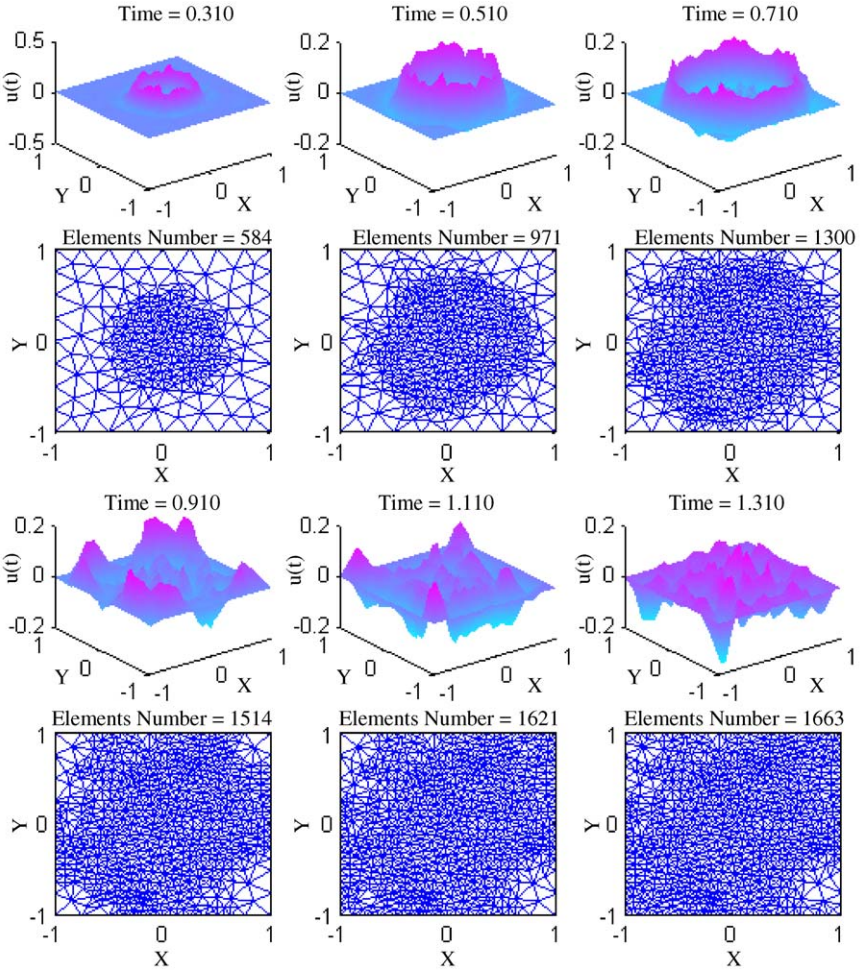


Fig. 9. The FEM solution in different times of the PDE that simulates throwing a stone into the middle of a square lake. The FEM mesh modifications over time show that the NN modifier refined the mesh according to the wave motion.

6. Summary

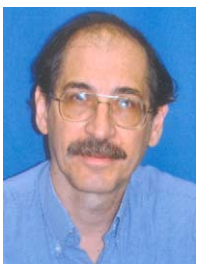
We have implemented a version of a NN modifier for the FEM mesh; designed to adaptively change the mesh based on a *prediction* of the gradient. In experimental work, we have shown that the NN can accurately predict the gradient and applying this mesh results in a substantial numerical improvement.

Acknowledgements

Supported in part by the HIACS Research Center, the University of Haifa.

References

[1] O. Axelsson, V.A. Barker, Finite Element Solution of Boundary Value Problems, Academic Press, Inc., London, 1984.
[2] E.M. Azoff, Neural Network Time Series Forecasting of Financial Markets, Wiley, England, 1994.
[3] M.C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 1995.
[4] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signal System 2 (1989) 303–314.
[5] H. Demuth, M. Beale, Neural Network Toolbox User's Guide, The Math Works Inc., Natick, 2000.
[6] K. Eriksson, D. Estep, P. Hansbo, C. Johnson, Computational Differential Equations, Springer, London, 1996.
[7] L. Fausett, Fundamentals of Neural Networks, Prentice-Hall, Inc., New Jersey, 1994.
[8] A.R. Gallant, H. White, On learning the derivatives on an unknown mapping with multilayer feedforward networks, Neural Networks 5 (1992) 129–138.
[9] D. Givoli, J.E. Flaherty, M.S. Shephard, Simulation of czochralski melt flows using parallel adaptive finite element procedures, Modelling Simul. Mater. Sci Eng. 4 (1996) 623–639.
[10] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward network are universal approximators, Neural Networks 2 (1989) 359–366.
[11] T.J.R. Hughes, The Finite Element Method, Prentice-Hall, New York, 1987.
[12] M. Leshno, V. Lin, A. Pinkus, D. Schocken, Multilayer feedforward networks with non-polynomial activation functions can approximate any continuous function, Neural Networks 6 (1993) 861–867.
[13] K. Levenberg, A method for the solution of certain problems in least squares, Quart. Appl. Math. 2 (1944) 164–168.
[14] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM J. Appl. Math. 11 (1963) 431–441.
[15] Matlab, Partial Differential Equation Toolbox User's Guide, The Math Works Inc., Natick, 1996.
[16] M. Norgaard, O. Ravn, N.K. Poulsen, L.K. Hansen, Neural Networks for Modelling and Control of Dynamic Systems, Cambridge University Press, Cambridge, 2000.
[17] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, 1996.
[18] A.J. Shepherd, Second-Order Methods for Neural Networks, Springer, London, 1997.
[19] M. Shoham, M. Meltser, L. Manevitz, Approximating functions by neural networks: a constructive solution in the uniform norm, Neural Networks 9 (6) (1993) 861–867.
[20] B. Widrow, M. Lehr, 30 years of adaptive neural network: perceptron, madaline, and back-propagation, Proc. IEEE 78 (9) (1990) 1415–1442.
[21] B. Widrow, S. Stearns, Adaptive Signal Processing, Prentice-Hall Inc., Englewood Cliffs, NJ, 1985.



Larry M. Manevitz was born in Rochester, NY, raised in Brooklyn, NY; went to Erasmus Hall High School, Brooklyn College (B.Sc.) and Yale University (M.Phil., Ph.D.). He did his doctoral work under the supervision of Abraham Robinson in the field of applied model theory (a sub-field of mathematical logic).

He has held regular positions in mathematics and computer science departments at CUNY, Hebrew University, Bar Ilan University and University of Haifa. He has held many visiting positions including: Courant Institute, NYU, IBM, NASA Ames Research Center, Polytechnic University (Brooklyn), U. Texas (Austin), U. Maryland (College Park) and most recently Oxford University (Department of Experimental Psychology).

His interest in mathematical logic was fueled by his thesis advisor Abraham Robinson at Yale University. His interest in Artificial Intelligence started with collaborations with Robert Hummel of Courant Institute on discussing ways of combining uncertain information. During a visit to NASA Ames Research Center he met Pentti Kanerva and started his work on neural networks as it related to temporal memory models. Most recently, he has started to work on temporal issues in Brain Science and Modeling.

During a visit to Stanford University, he met Dan Givoli which inaugurated a series of collaborations (of which this paper is the latest) on applying "soft computing" methods to the finite element method. Altogether he has about 50 scientific publications.

He is currently in the Computer Science Department at the University of Haifa, Israel and is the head of both the Neurocomputation Laboratory there and the HIACS (Haifa Interdisciplinary Center for Advanced Computer Science) Research Center.

Akram Bitar completed his B.Sc. and M.Sc. degrees in Mathematics and Computer Science in Haifa University. Since 2000 he is a researcher and a senior programmer in the Storage and Systems department at the IBM Research and Development Labs in Haifa, Israel. His main research areas are: Neural Networks, Finite-Element Methods, Computer Networks, and Distributed Systems.

Prof. Dan Givoli completed his B.Sc. and M.Sc. degrees in Mechanical Engineering in Tel Aviv University, and his Ph.D. degree at Stanford University in 1988. Since 1988 he is at the Technion—Israel Institute of Technology, in the Department of Aerospace Engineering. During 1995–1996 he was a Visiting Associate Professor at Rensselaer Polytechnic Institute in New York. During 2001–2002 he was a Visiting Professor at the Naval Postgraduate School in Monterey, California. Since 2001 he is a full professor at the Department of Aerospace Engineering, and since 2004 he is the Chairman of this department.

The main research area of Prof. Givoli is Computational Mechanics and Aeronautics, and especially computational methods for wave problems and finite element methods. He is the author of the book Numerical Methods for Problems in Infinite Domains that was published in 1992 by Elsevier. In recent years Prof. Givoli edited two books and several special issues in international journals. He serves as an Associate Editor of the journal Wave Motion and is a member of the editorial board of five other international journals.

Prof. Givoli is one of the founders of the Israel Association for Computational Methods in Mechanics (IACMM) and serves today as a Council Member and the editor of the IACMM journal. In addition, Prof. Givoli serves as an elected-member in the Council of International Association of Computational Mechanics (IACM) and a member of the Solid Mechanics Committee of the corresponding European association ECCOMAS. In Israel, Prof. Givoli consults to a number of industrial and academic bodies on subjects of computational aerospace engineering.