Finite-Element Mesh Adaptation for Time-Dependent Partial Differential Equations via Time Series Prediction Using Neural Networks Extended Abstract*

Larry Manevitz[1], Akram Bitar[1] and Dan Givoli[2]

[1]Department of Computer Science
University of Haifa
manevitz@cs.haifa.ac.il
akram@il.ibm.com

[2]Faculty of Aerospace Engineering
Technion
Israel Institute of Technology
givolid@aerodyne.technion.ac.il

Haifa, Israel

June 21, 2001

Abstract

In this paper, 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: finite-element method, neural networks, time-dependent PDEs, time series prediction, mesh adaptation.

Topics: Anticipatory Systems, Industrial Applications of Neural Networks.

*Partially supported by *HIACS* Research Center.

1 Introduction and Background

The finite-element method (FEM) [5, 9] is a computationally intensive method for the numerical solution of partial differential equations (PDEs). It is a widely used tool and in many cases is the method of choice. Basically, the FEM works by deciding, *a priori* on a certain kind of simple approximation to the solution, by dividing up the domain of solution into a finite number of nonoverlapping elements (i.e. subdomains), and by allowing the parameters of the simple approximation to vary from element to element. This collection of elements and the connections between them constitutes the finite-element mesh. The requirement that the individual local solutions remain consistent with each other and with the boundary conditions results in linear constraints on the parameters. These are then solved by standard linear algebra techniques.

The quality of the numerical results thus depends on the geometry; i.e. the domain of solution and its division to small elements as well as the kind of approximation taken (see [7]).

In addition, time is typically treated differently than spatial dimensions in the solution phase of time-dependent partial differential equations. That is typically time is not treated as simply another dimension, but instead time is *simulated* i.e. the equation is repeatedly solved for different constant times; using the previous solution as the starting condition for the next one.

However, in time-dependent PDEs [5, 6], this implies that one should not use the same mesh at different times since the areas of "interest" (i.e. areas where the simple approximations are inherently less accurate) are of course changing 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.

This means that the mesh adaptation is a crucial part for the efficient computation of the numerical method. Thus, 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 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 [8]).

The neural network is a universal approximator 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.

1.1 Background of Time Series Neural Networks

Neural networks (NNs) [4] is a biologically inspired model, which tries to simulate the network of neurons, or the nervous systems, in the human brain. The artificial neural networks consist of simple calculation elements, called neurons, and weighted connections between them. 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 algorithms. In general, neural network tasks may be divided into four main types of distinct applications: classifications, associations, codification and simulations.

For our purposes, the neural network may be considered simply as a data processing technique that maps, or relates, some type of stream information to an output stream of data. For example, to classify input vectors by dividing the input space into two regions: one for input vectors which are above and to the left of a decision boundary line, and the other for vectors which are below and to the right of the decision boundary line. The most common NN model is the supervised-learning, feed-forward network. Typical the feed-forward network contains three types of processing units, input units, output units and hidden units, organized in a hierarchy of layers, as demonstrated in Figure 1.

The learning algorithms such as back-propagation or Newton-Gauss consist of two phases: feed-forward propagation and backward propagation. In forward propagation, the input units send the input signals forward through the network to produce an output. Then, the difference between the actual and desired outputs produces error signals which are sent backwards through the network to modify the weights between neurons. The forward and backward propagation are executed iteratively over the training set until convergence occurs (when the average squared error between the network outputs and the desired outputs reaches an acceptable value).

Time series are 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:

$$x(n) = f(x(n-1), x(n-2), ..., x(n-m))$$

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 feed-forward network can approximate any function after a suitable amount of training [2] 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 nonlinear model that can be trained to map past and future values of a time series. This method is

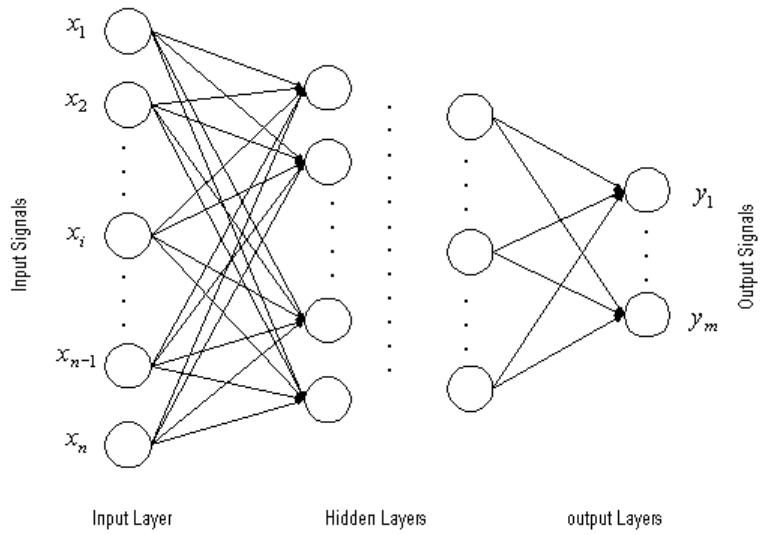

Figure 1: Feed-forward network architecture

called time series prediction with neural networks and is used in the forecasting of financial markets [1]. (e.g. to predict whether stock market rates will rise or fill). The following section describes how we use this method to predict the mesh refinement placement for time-dependent PDEs.

1.2 Time-dependent PDEs

PDEs arise in modeling numerous phenomena in science and engineering. The time-dependent PDEs tend to be divided into two categories: hyperbolic and parabolic. 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 PDEs is the heat equations.

2 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 refinements should be dynamic and depends on the error estimation in each time stage.

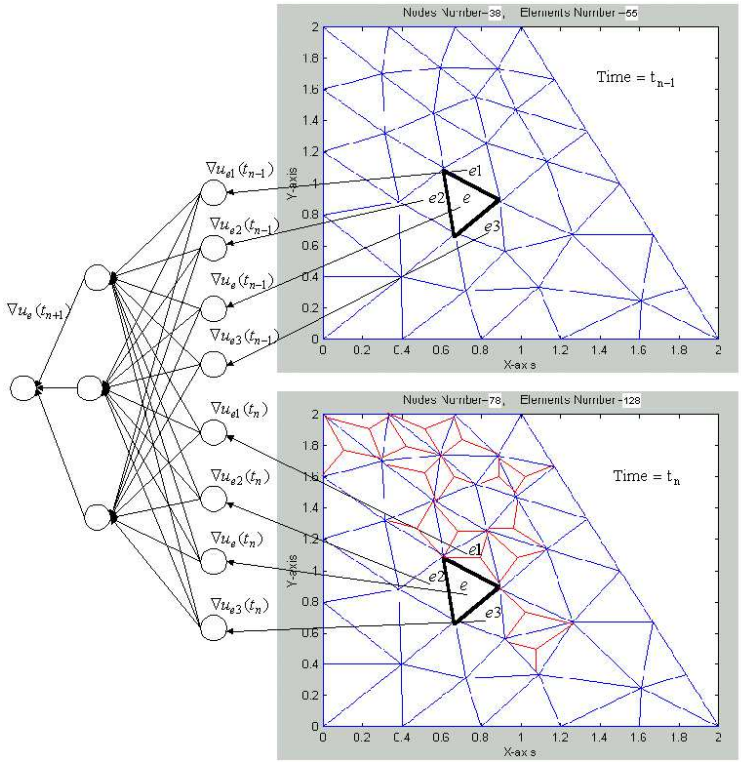

Figure 2: Using Neural Networks to forecast future FEM gradient values. Gradient values for two previous times and all neighboring elements are used as input to the network.

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, e.g. 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. Figure 2 illustrates this concept.

Thus there are two steps to our methodology: (1) training the neural network to predict the indicators (at least) one step in advance (2) applying the indicators to the refinement in the FEM solution.

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.

2.1 Neural Network Architecture and Training

In this study, the MATLAB's Neural Network Toolbox [3] was used for designing and training the neural network. In the examples tested so far the results are fairly dramatic. First, using the Levenberg-Marquardt training algorithm [10] (a variant of Gauss-Newton) the training was both quite swift and exceptionally accurate. (See figure 3.) Second, the improvement in the FEM numerical results (as compared with the "standard" gradient adaptive method) reached as high as 28% 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 both networks was six input units (corresponding to the value of the gradient of the element and its two neighbors in the current and previous times), six hidden units (with tan-sigmoid transfer function), and one output unit (with linear transfer function) that gave the prediction of the output value. See Figure 2.

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 it's ability to generalize; at this point the training is stopped. In the following section we present the results of our experiments on the one and two dimension wave equations.

2.1.1 One Dimension Wave Equations

We have run the NN modifier over a variety of initial conditions for the wave equation. In all cases, the NN predictor was extremely accurate. Figure 3 shows the results of a typical prediction test for interior and boundary elements. Training took about 117 epochs to converge to extremely small error (about 0.00024) in the interior elements prediction. 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 norm and in the L^∞ norm).

In this abstract we present one sample example, where the initial condition of the wave is a Gaussian. See Table 1.

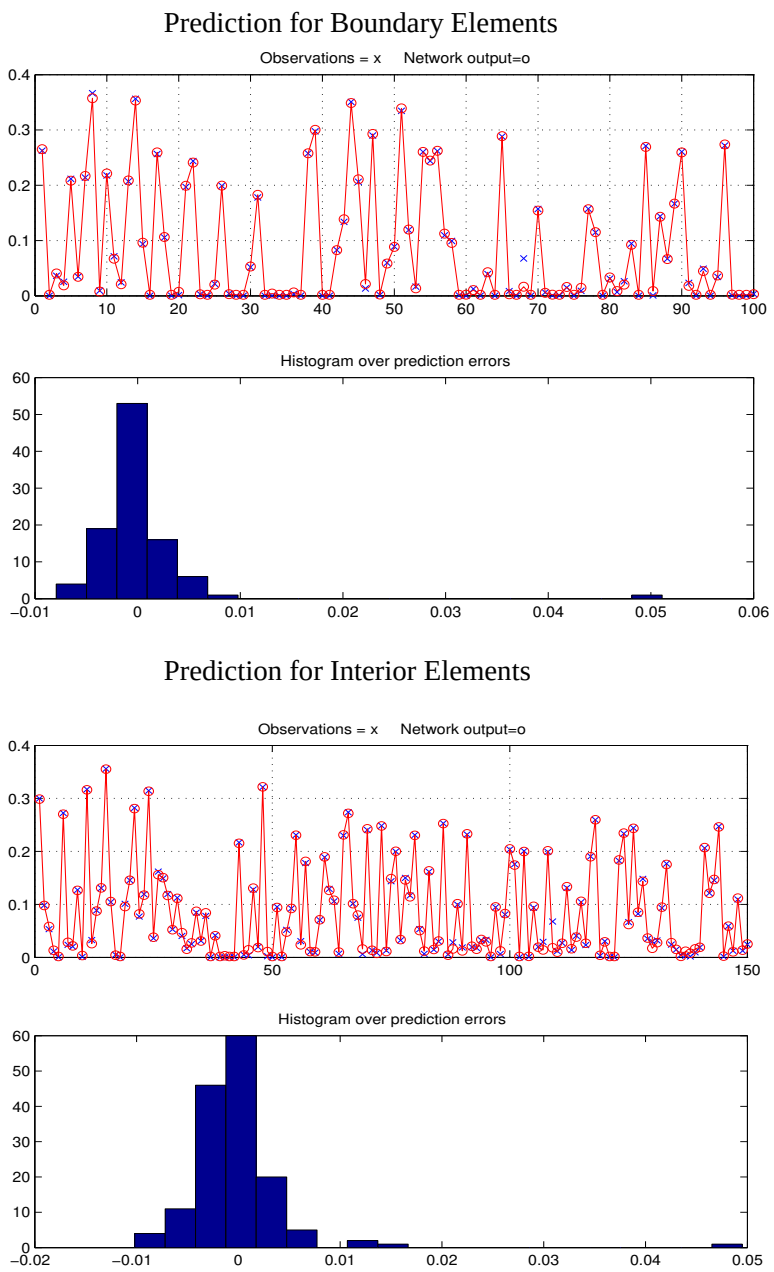

Figure 3: Time Series Prediction Test for the 1D Wave Equation. (See Text.) Blue (x) indicates test values; red (o) the network response.

Example 1

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

$$0 \leq x \leq 10$$

$$u(0,t) = 0 \text{ and } u(10,t) = 0$$

$$u(x,0) = \begin{cases} 1 - |1-x| & 1 \leq x \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

Number of Initial Elements:10, Time:12, Time Step:0.08
Threshold for refinement = 0.08 (gradient)

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

Method	Number of Refined Elements	L^2 Norm		L^∞ Norm	
		Max	Average	Max	Average
NN Modifier	70	0.15756	0.0869	0.1653	0.1056
Standard Modifier	70	0.1826	0.1022	0.1914	0.1222
No Adaptation	0	2.5332	1.0053	3.4708	1.0643

Example 2

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

$$0 \leq x \leq 25$$

$$u(0,t) = 0 \text{ and } u(25,t) = 0$$

$$u(x,0) = \begin{cases} \exp(\frac{-(x-5)^2}{2}) & 0 \leq x \leq 25 \\ 0 & otherwise \end{cases}$$

Number of Initial Elements:15, Time:25, Time Step:0.12
Threshold for refinement = 0.2 (gradient)

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

Method	Number of Refined Elements	L^2 Norm		L^∞ Norm	
		Max	Average	Max	Average
NN Modifier	98	0.4423	0.1928	0.5190	0.2342
Standard Modifier	91	0.6671	0.2686	0.8230	0.3142
No Adaptation	0	1.4622	0.6985	1.6288	0.6456

Table 1: One dimension examples. Comparison between FEMs run with (i) The neural network predictor of the gradient measure. (ii)"Standard" refinements using the gradient measure. (iii) No adaptation.

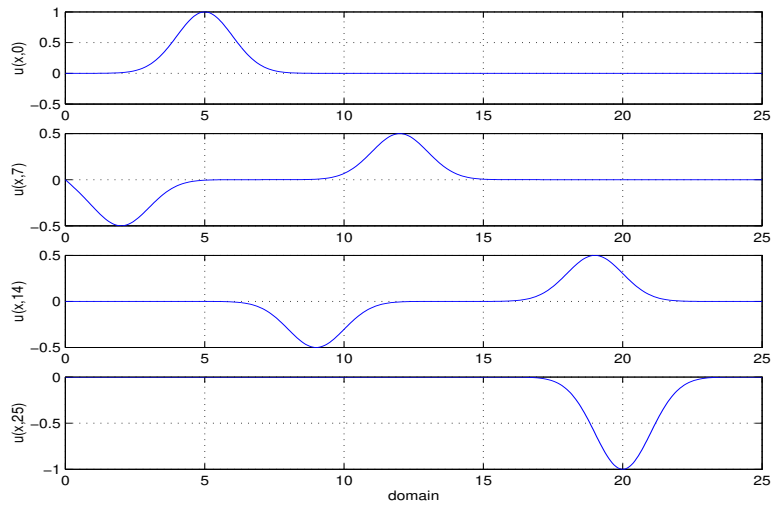

Figure 4: Analytic Solution

The analytical 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 travelling triangle reaches the edge it turns over and returns upside down (see Figure 4). The NN modified solution and the "standard" gradient modifier are displayed in figure 5. Compare the graphs in Figure 5. Observing the areas indicated in the figure, one can see that the NN has chosen to place its resources in the correct places. 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 figure 6 we track the errors (both in L^2 norm and in the L^∞ norm).

2.2 Two Dimensional Waves

So far we have done only a couple of initial experiments in two dimensions. In the examples done, see Table 2, (i) the prediction of the gradient was very accurate (see figure 7 and figure 8) and (ii) the improvement in the FEM numerical results were around 6.5% over the standard gradient methods.

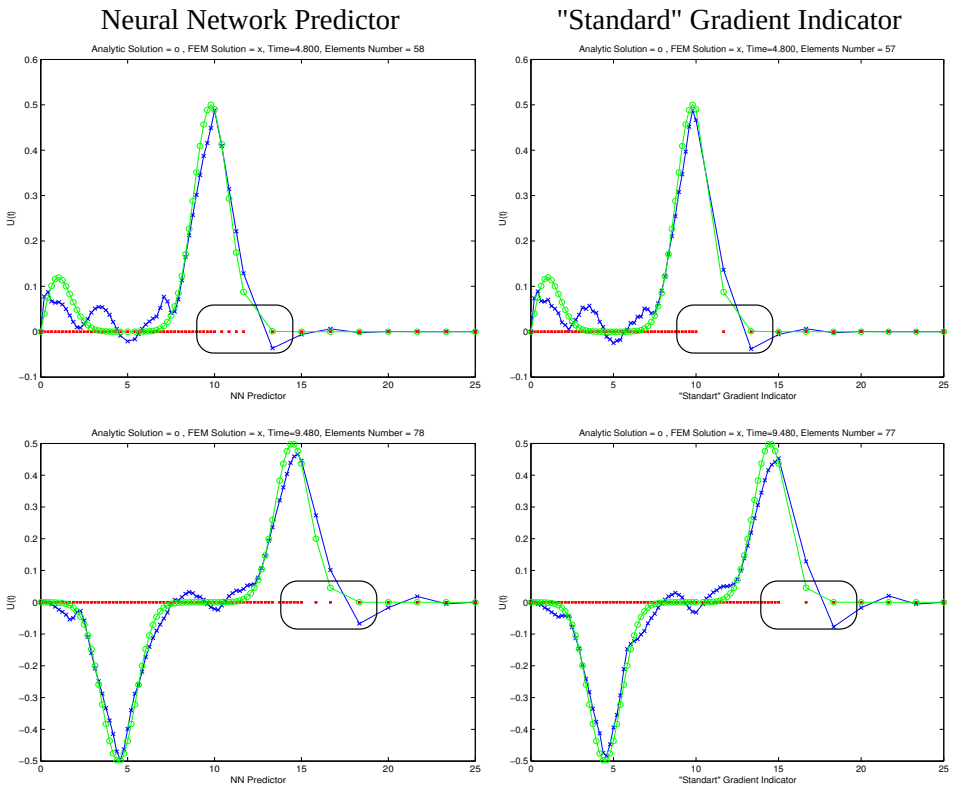

Figure 5: Results from FEM on 1D Wave Equation. The left figures are refined with the NN predictor. Also indicated is the analytic solution. The right figures are refined with the "standard" gradient indicator. Compare the segments of the curves on the left (enclosed rectangles) with the corresponding ones on the right to see how the NN predictory focuses the resources in the correct places.

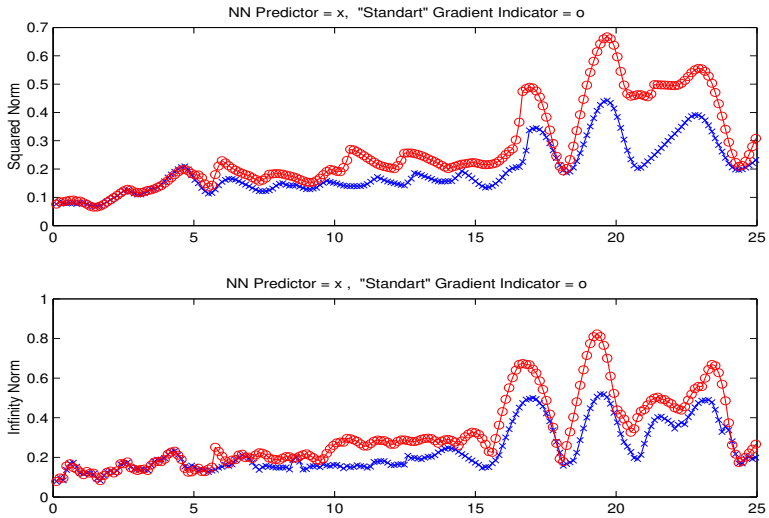

Figure 6: L^2 error norm and L^∞ error norm over time

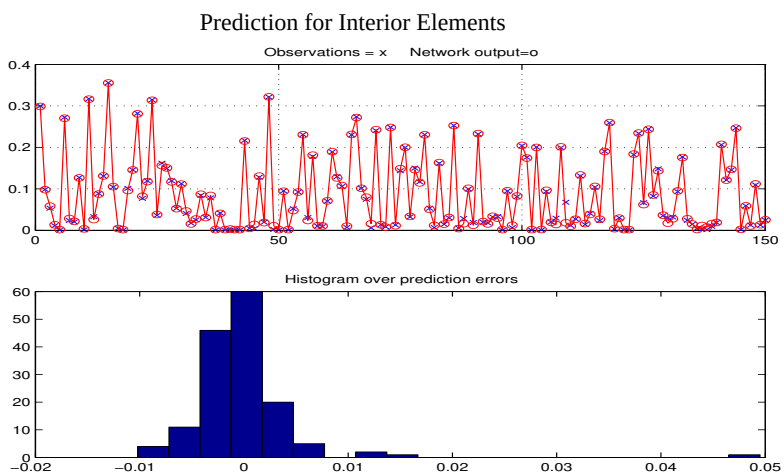

Figure 7: Time Series Prediction Test for the 2D Wave Equation. Blue (x) indicates test values; red (o) the network response.

Example 1

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

$$-1 \leq x \leq 1 \ , -1 \leq y \leq 1$$

$$u(-1, y, t) = 0 \ \ and \ \ u(1, y, t) = 0 \ \ for -1 \leq y \leq 1$$
$$u(x, -1, t) = 0 \ \ and \ \ u(x, -1, t) = 0 \ \ for -1 \leq x \leq 1$$

$$u(x, y, 0) = \begin{cases} 15\text{x(x+1)y(y+1)} & -1 \leq x \leq 0 \ \ , \text{-1} \leq y \leq 0 \\ 0 & otherwise \end{cases}$$

Number of Initial Elements:28, Time:3, Time Step:0.05

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

Method	Number of Refined Elements	Average L^2 Norm	Average L^∞ Norm
NN Modifier	803	0.4057	0.4846
Standard Modifier	803	0.4314	0.5029

Example 2

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

$$-1 \leq x \leq 1 \ , -1 \leq y \leq 1$$

$$u(-1, y, t) = 0 \ \ and \ \ u(1, y, t) = 0 \ \ for -1 \leq y \leq 1$$
$$u(x, -1, t) = 0 \ \ and \ \ u(x, -1, t) = 0 \ \ for -1 \leq x \leq 1$$

$$u(x, y, 0) = \cot(\cos(\tfrac{\Pi}{2x}))$$

$$\tfrac{\partial^2 u}{\partial t^2}(x, y, 0) = 3\sin(\Pi x)\exp(\sin(\tfrac{\Pi}{2y}))$$

Number of Initial Elements:28, Time:3, Time Step:0.08

Improvement: L^2 error norm = 6.5%, L^∞ error norm = 6.2%

Method	Number of Refined Elements	Average L^2 Norm	Average L^∞ Norm
NN Modifier	381	0.3938	0.4145
Standard Modifier	341	0.4213	0.4422

Table 2: Two dimension examples. Comparison between FEMs run with (i) The neural network predictor of the gradient measure. (ii)"Standard" refinements using the gradient measure.

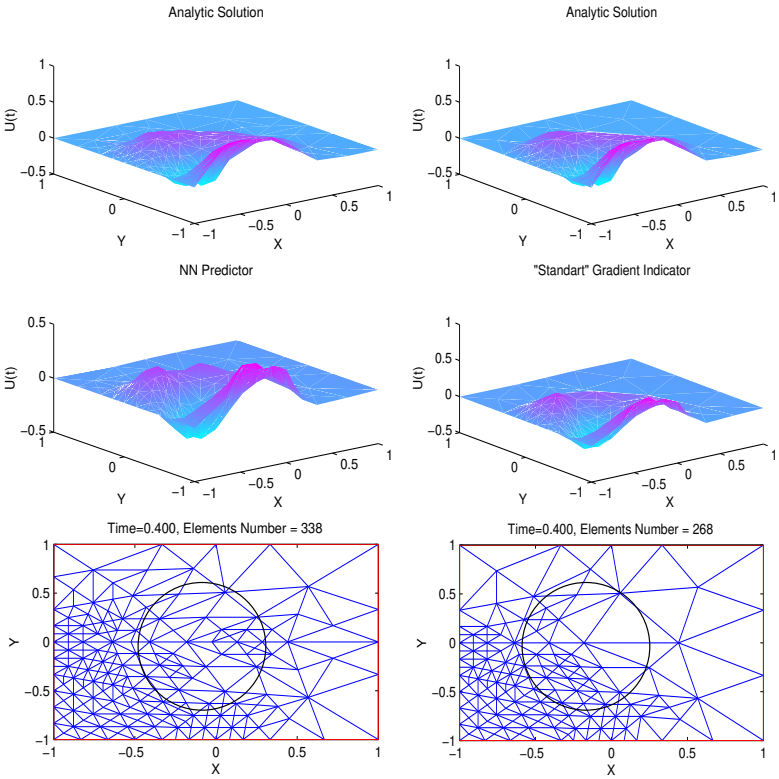

Figure 8: FEM result; 2D Wave Equation Example 1 (see Table 2. The left figures are refined with the NN predictor. The right figuers are refined with the "standard" gradient indicator.

3 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.

Acknowledgment

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

References

[1] E. Michael Azoff. *Neural Network Time Series Forecasting Of Financial Markets.* John Wiley & Sons Ltd, England, 1994.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signal System*, 2:303–314, 1989.

[3] Howard Demuth and Mark Beale. *Neural Network Toolbox User's Guide.* The Math Works Inc., Natick, 2000.

[4] Laurene Fausett. *Fundamentals Of Neural Networks.* Prentice Hall, Inc., New Jersey, 1994.

[5] T.J.R. Hughes. *The Finite Element Method.* Prentice-Hall, New York, 1987.

[6] K.Eriksson, D.Estep, P.Hansbo, and C.Johnson. *Computational Differential Equations.* Springer-Verlag,, London, 1996.

[7] L. Manevitz, D. Givoli, and M. Yousef. Finite-element mesh generation using self-organizing neural networks. *Microcomputers in Civil Engineering*, 12:233–250, 1997.

[8] M. Norgaard, O. Ravn, N.K. Poulsen, and L.K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems.* Cambridge University Press, Cambridge, 2000.

[9] O.Axelsson and V.A. Barker. *Finite Element Solution Of Boundary Value Problems.* Academic Press, Inc., London, 1984.

[10] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, 1996.