The Finite Element Neural Network And Its Applications To Forward And Inverse Problems

L.N.M .Tawfiq
College of Education, Ibn Al- Haitham , University of Baghdad

Abstract

In this paper, first we reformulated the finite element model (FEM) into a neural network structure using a simple two - dimensional problem. The structure of this neural network is described , followed by its application to solving the forward and inverse problems. This model is then extended to the general case and the advantages and disadvantages of this approach are described along with an analysis of the sensitivity of the algorithm to errors in the measurements. Consider a typical boundary value problem with the governing differential equation: $L\varphi = f$, where L is a differential operator, f is the forcing function and φ is the unknown quantity. This differential equation can be solved in conjunction with boundary conditions on the boundary Γ enclosing the domain. A commonly used approach to solve this problem is to use the finite element approach.

Introduction

Neural networks are connectionist models proposed in an attempt to mimic the function of the human brain. A neural network consists of a large number of simple processing elements called neurons(or nodes)(1)(2). Neurons implement simple functions and are massively interconnected by means of weighted interconnections. These weights, determined by means of a training process, determine the functionality of the neural network .(3). The training process uses a training database to determine the network parameters (weights). The functionality of the neural network is also determined by its topology. In addition, to input and output layers there are usually layers of neurons that are not directly connected to either the input or the

output, called hidden layers. The corresponding nodes are referred to as hidden nodes. Hidden layers give the network the ability to approximate complex, nonlinear functions. (4)

The advantages of using neural networks are numerous: neural networks are learning machines that can learn any arbitrary functional mapping between input and output, they are fast machines and can be implemented in parallel, either in software or in hardware. In fact, the computational complexity of neural networks is polynomial in the number of neurons used in the network. Parallelism also brings with it the advantages of robustness and fault tolerance. Efficient learning algorithms ensure that the network can learn mappings to any arbitrary precision in a short amount of time. Furthermore, the input-output mapping is explicitly known in a neural network and conjugate gradient procedures (5) can be used advantageously to perform the inversion process.

- The Finite Element Method

Consider a typical boundary value problem with the governing differential equation $L\phi = f$ where L is a differential operator, f is function and ϕ is the unknown quantity.This differential equation can be solved in conjunction with boundary conditions on the boundary Γ enclosing the domain. A commonly used approach to solve this problem is to use the finite element approach.

The variational formulation of this approach determines the unknown ϕ by minimizing the functional (6): $F(\tilde{\phi}) = \frac{1}{2}\langle L\tilde{\phi},\tilde{\phi}\rangle - \frac{1}{2}\langle \tilde{\phi}, f\rangle - \frac{1}{2}\langle f,\tilde{\phi}\rangle$

with respect to the trial function $\tilde{\phi}$. The minimization procedure starts by dividing the domain of interest into small subdomains called elements and representing $\tilde{\phi}$ in each element by means of basis functions defined over the element :

$(\tilde{\phi})^e = \sum_{j=1}^{n} (N_j)^e (\phi_j)^e$ ..... [*], where$(\tilde{\phi})^e$ is the unknown solution

in element e, $(N_j)^e$ is the basis function associated with node j in element e, $(\phi_j)^e$ is the value of the unknown quantity at node j and n is the total number of nodes associated with element e. In general, the basis functions(also referred to as interpolation functions)can be linear, quadratic or higher order basis functions. Higher order polynomials, though more accurate, generally result in higher computational complexity, and hence, linear basis functions are

commonly used. Once the domain is divided into smaller elements, the functional can be expressed as: $F(\bar{\phi}) = \sum_{e=1}^{M} F((\bar{\phi})^e)$ .....[**] ,

where M is the total number of elements and $F((\bar{\phi})^e)$ represents the value of the functional within element e :

$F((\bar{\phi})^e) = \frac{1}{2}\int_{\Omega^e} (\bar{\phi})^e L(\bar{\phi})^e d\Omega - \int_{\Omega^e} f(\bar{\phi})^e d\Omega$ .....[1]

By substituting [*]in [1], we obtain the discrete version of the functional within each element :

$F((\bar{\phi})^e) = \frac{1}{2}(\phi^e)^T \int_{\Omega^e} N^e L N^{eT} d\Omega\, \phi^e - \phi^{eT} \int_{\Omega^e} f N^e d\Omega$

where $(\ldots)^T$is the transpose of a matrix or a vector. This can be written in matrix-vector form as: $F((\bar{\phi})^e) = \frac{1}{2}\phi^{eT} K^e \phi^e - \phi^{eT} b^e$.....[2]

where K^e is thenxn elemental matrix with elemcnts

$(K_{ij})^e = \int_{\Omega^e} (N_i)^e L(N_j)^e d\Omega$ and b is an nx1 vector with elements

$(b_i)^e = \int_{\Omega^e} f(N_i)^e d\Omega$. From [**] and [2], we obtain: $F = (1/2)\phi^T K \phi - \phi^T b$ .....[3]

where K is the NxN global matrix derived from the terms of the elemental matrices for different elements, and N is the total number of nodes. Equation [3] is the discrete version of the functional and can be minimized with respect to the nodal parameters ϕ by taking the derivative of F with respect to ϕ and setting it equal to zero. The resulting equation

$\partial F / \partial \phi = K\phi - b = 0$ .....[4]

can be solved for the nodal parameters ϕ :

$\phi = K^{-1} b$ .....[5]

Boundary conditions for these problems are usually of two types: natural boundary conditions and essential boundary conditions. Essential boundary conditions (also referred to as Dirichlet boundary conditions) impose constraints on the value of the unknown ϕ at several nodes. Natural boundary conditions (of which Neumann boundary conditions are a special case) impose constraints on the change in ϕ across a boundary. Imposition of these conditions into the finite element formulation is straightforward. Natural boundary conditions are incorporated into the functional and are satisfied automatically during the solution procedure.

These conditions are imposed on the functional minimization equation [4], by deleting the rows and columns of the K matrix corresponding to the nodes that are part of the boundary.

Suppose the given boundary condition is $\phi = p$ on the boundary Γ and let the i^{th} node represent the boundary Γ. Then, equations [4] are modified as follows:

$b_i \leftarrow p$, $b_j \leftarrow b_j - K_{ji}\, p$, $j \neq i$ & $K_{ii} \leftarrow 1$, $K_{ij} \leftarrow 0$, $K_{ji} \leftarrow 0$, $j \neq i$ .This process can be repeated for each node on the boundary Γ and the resulting matrix can be solved as indicated in [5] to obtain the solution subject to the Dirichlet conditions.

- The Finite Element Neural Network

The finite element model can be easily converted into a neural network form. To see this, consider the simple two-dimensional example: $-\nabla\cdot(\alpha\ \nabla\varphi) + \beta\varphi = f$ ..... [***], with boundary conditions $\varphi = p$ on Γ_1 and $\alpha\ \nabla\varphi \cdot f_i + \gamma\ \varphi = q$ on Γ_2 , where α and β are constants depending on the material, f is the function, $\Gamma = \Gamma_1 + \Gamma_2$ is the boundary enclosing the domain, f_i is its outward normal unit vector and γ, p and q are known parameters associated with the boundary. Assume that the domain is divided into two elements with 4 nodes. The elemental matrices K^e and b^e can be derived as (6):

$(K^e)_{ij} = \int_\Omega^e (\alpha\ \nabla N^e_i\ .\ \nabla N^e_j + \beta\ N^e_i\ N^e_j\ )\ d\Omega$ ..... [6]

and $b^e_i = \int_\Omega^e f N^e_i\ d\Omega$ ..... [7]

The global matrix equation can be assembled by combining the two elemental matrices. To do this, we need the node-element connectivity information given in table (1) This table contains information about the various nodes that make up each element, as well as their position in the element (often called the local node number).Each node also has a global node number, indicating its position in the entire finite element model system. The columns in the table marked n(i.e.) refer to the i^{th} node in element e and the value of n(i. e.) is the global node number.For instance,node2 appears as the second node in element 1 and the third node in element 2

The connectivity array shown in Table(1) can be used to obtain the global matrix K, by Combi- ning the appropriate members of the elemental matrices. Consider node 2 as an example. Since node 2 appears as the second node in element 1 and the third node in element 2, we

combine the corresponding members $(K_{22})^1$ and $(K_{33})^2$ of the elemental matrices to obtain $K_{22} = (K_{22})^1 + (K_{33})^2$ .This process is repeated for each of the four nodes, giving :

$$\mathbf{K} = \begin{bmatrix} K_{11}^{1} & K_{12}^{1} & K_{13}^{1} & 0 \\ K_{21}^{1} & K_{22}^{1} + K_{33}^{2} & K_{23}^{1} + K_{32}^{2} & K_{31}^{2} \\ K_{31}^{1} & K_{32}^{1} + K_{23}^{2} & K_{33}^{1} + K_{22}^{2} & K_{21}^{2} \\ 0 & K_{13}^{2} & K_{12}^{2} & K_{11}^{2} \end{bmatrix}$$

Similarly, the vector b is given by:

$$b = \begin{bmatrix} b_1 \\ b_2 + b_3 \\ b_3 + b_2 \\ b_1 \end{bmatrix}$$

In order to convert the FEM into ANN, we start by first separating $(K^e)_{ij}$ into two components: one component dependent on the material properties α and β and the second component independent of these material properties. This can be achieved by rewriting [6] as:

$(K^e)_{ij} = \alpha \int_{\Omega^e} \nabla(N^e)_i \cdot \nabla(N^e)_j \, d\Omega + \beta \int_{\Omega^e} (N^e)_i (N^e)_j \, d\Omega$ ..... [8]

$= \alpha (S^e)_{ij} + \beta(T^e)_{ij}$, where $(S^e)_{ij} = \int_{\Omega^e} \nabla(N^e)_i \cdot \nabla(N^e)_j \, d\Omega$ and $(T^e)_{ij} = \int_{\Omega^e} (N^e)_i \cdot (N^e)_j \, d\Omega$

Rewriting the original equation as shown in [8] assumes that the material properties are constant in an element. In general, this is not an unreasonable assumption, since the elements are usually small enough for this assumption to be true. Equations [8] can be converted into ANN. The structure of this network is shown in fig. (1) The neural network is a three layer neural network, with input, out - put and hidden layers. The input layer has two groups of 2 neurons, with one group taking the α value in each element as input and the second group taking in the values of βin each element as input. The hidden layer has16 neurons that are arranged in groups of 4 neurons. The output of each group of hidden layer neuron is the corresponding row vector of K. In the general case with M elements and N neurons in the FEM mesh, the input layer has 2M neurons, with the inputs being the material properties α and β in each of the M elements. The hidden layer has N^2 neurons arranged in N groups of N neurons, corresponding to the N^2 elements in the global matrix K. The weights

from the input to the hidden layer are set to the appropriate values of (S_{ij}) and (T_{ij}) examples of these weights are shown in fig.(1) Each neuron in the hidden layer acts as a summation unit, and the outputs of the hidden layer neurons are the elements K_{ij} of the global matrix K :

$$K_{ij} = \sum_{e=1}^{2} (\alpha_e (w_{ij})^e + \beta_e (g_{ij})^e) \quad \ldots\ldots\ldots\ldots\ldots\ldots [9]$$

Where $(w_{ij})^e = (S_{ij})^e$ if nodes i and j are part of element e, and $(w_{ij})^e$ otherwise. Similarly, $(g_{ij})^e = (T_{ij})^e$ if nodes i and j are part of element e, and $(g_{ij})^e = 0$ else.

Each group of hidden neurons is connected to one output neuron(giving a total of 4 out put neurons) by a set of weights φ, with each element of φ representing the nodal values φ_j . Each output neuron is also a summation unit, and the output of each neuron is equal to b_i :

$$b_i = \sum_{j=1}^{4} K_{ij} \varphi_j = \sum_{j=1}^{4} \varphi_j (\sum_{e=1}^{2} (\alpha_e (w_{ij})^e + \beta_e (g_{ij})^e) \ldots\ldots\ldots [10]$$

where the second part of [10] is obtained by using [9]. The number of output neurons in the general case increases to N.

-Forward and Inverse Problem Formulation Using FENN

Applying the FENN to solve the forward problem is straightforward. The forward problem involves determining the weights φ given the material parameters. Given these values, the procedure for solving the forward problem is as follows:

- Apply the material parameters α and β to the input neurons and compute the output of each of the hidden neurons. Any natural boundary conditions are applied as bias inputs to the hidden layer neurons. Initialize the value of φ randomly for all the free weights. Fix the values of the clamped weights according to the Dirichlet boundary conditions. Let the value of φ at iteration t be denoted by $\varphi(t)$.
- At iteration t, compute the output of the neural network:

$$\hat{(b_i)}(t) = \sum_{j=1}^{N} K_{ij} \varphi_j(t) \ , \ i = 1, 2, \ldots, N \ldots\ldots\ldots\ldots\ldots [11]$$

- Compute the error at the output of the neural network:

$$E(t) = \tfrac{1}{2} \| b - \hat{b}(t) \| = \tfrac{1}{2} \sum_{i=1}^{N} (b_i - \hat{(b_i)}) = \tfrac{1}{2} \sum_{i=1}^{N} (E_i (t))^2 \quad \ldots\ldots [12]$$

- Compute the gradient of the error with respect to the free hidden layer weights :

$g_j(t) = \partial E(t) / \partial \varphi_j = -\sum_{i=1}^{N} E_i(t) K_{ij}$, $\rho_j(0) = -g_j(0)$

-Update the free weights using the conjugate gradient (CG) equation :

$\varphi_j(t+1) = \varphi_j(t) + \eta \rho_j(t)$, $\rho_j(t) = -g_j(t) + \beta_j \rho_j(t-1)$ .....[13]

where $\beta_j = \Delta g(t-1)^T g(t) / g(t-1)^T g(t-1)$ ..... [14]

Repeat steps 2-5 till convergence is achieved. The convergence criterion considered

here is that the output error must fall below a threshold.

The same ANN can be applied easily to solve the inverse problem. The inverse problem involves determining the material properties α and β, given the measurement φ and b. We apply the following algorithm to solve the inverse problem :

- Initialize the values of α and β. Let $\alpha(t)$and $\beta(t)$ denote the values of the material properties at iteration t. Fix the weights between the hidden layer and the output layer neurons to the measurement φ.
- At iteration t, apply $\alpha(t)$ and $\beta(t)$ at the input layer of the FENN. Compute the output of the network using equations [9] and [10]. Call this output $\hat{b}(t)$.
- Compute the error at the output :

$$E(t) = \tfrac{1}{2} \| b - \hat{b}(t) \| = \tfrac{1}{2} \sum_{i=1}^{N} (b_i - (\hat{b}_i)(t))^2 = \tfrac{1}{2} \sum_{i=1}^{N} (E_i)^2$$

- Compute the gradient of the error due to α (t) and $\beta(t)$:

$g_0 = \partial E(0)/\partial \alpha_e$, $\rho_0 = -g_0$, $g(t) = \partial E(t)/\partial \alpha_e = -\sum_{i=1}^{N} E_i (\sum_{j=1}^{N} \varphi_j (w_{ij})^e$, $\rho(t) = -g(t) + \beta_t \rho(t-1)$

$\hat{g}_0 = \partial E(0)/\partial \beta_e$, $\rho_0 = -\hat{g}_0$, $\partial E(t)/\partial \beta_e = -\sum_{i=1}^{N} E_i (\sum_{j=1}^{N} \varphi_j (\hat{g}_{ij})^e)$, $\rho(t) = -\hat{g}(t) + \beta_t \rho(t-1)$ ...[15]

- Update the values of $\alpha(t)$ and $\beta(t)$ using the CG equation :

$\alpha_e(t+1) = \alpha_e(t) + \eta \rho(t)$ & $\beta_e(t+1) = \beta_e(t) + \eta \rho(t)$

- Repeat steps 2-5 till convergence. Again, the algorithm converges when the error falls below a threshold. Note β_t computed as [14]

- Advantages and Modifications

The major advantage of this formulation of the FENN is that it represents the finite element model in a parallel form , enabling parallel implementation in either hardware or software. Further, computing gradients in the FENN is very simple. This is an advantage in solving both forward and inverse problems using CG methods (7) (8). The expressions for the gradients(shown in the previous subsection) also indicate that the gradients can be computed in parallel, enabling even faster solution of both the forward and inverse problems. Secondly, the network has been derived to make the solution of inverse problems tractable. A major advantage of this approach for solving inverse problems is that it avoids inverting the global matrix in each iteration. This results in considerable computational savings. In addition, the approach lends itself easily to solution of the forward problem. This is in contrast to other approaches described in the literature, where the networks are derived to simplify the solution procedure for the forward problem, and need considerable modification to solve the inverse problem. The FENN also does not require any training , since most of its weights can be computed in advance and stored. The weights depend on the governing differential equation and its associated boundary conditions, and as long as these two factors do not change, the weights do not change. This approach also reduces the computational effort associated with the network.

The major drawback of the FENN is the number of neurons and weights necessary. However, the memory requirements can be reduced considerably, since most of the weights between the input and hidden layer are zero. These weights, and the corresponding connections, can be discarded. Similarly, most of the elements of the K matrix are also zero. The corresponding neurons in the hidden layer and the associated connections can also be discarded, reducing memory and computation requirements considerably. Furthermore, the weights between each group of hidden layer neurons and the output layer are the same (φ).

-Sensitivity Analysis of the Inverse Problem

Intuitively, an error in the measurement φ will result in an error (which can be large for ill-posed problems) in the estimate of the material parameters α andβ.In order to quantify the error in the material parameters, we assume that $\varphi_j = \varphi_j + \Delta\varphi_j$ is the measured

value of the potentials where φ_j is the true value of the potential and $\Delta\varphi_j$ is the error in the measurement at node j. Let $\acute{\alpha}_e(n)$ and $\beta_e(n)$ be the corresponding estimated values of the material parameters in element e at iteration n. We assume that : $\acute{\alpha}_e(n) = \alpha_e(n) + \delta_e(n)$ where $\alpha_e(n)$ is the corresponding estimate of α when the measurement error in φ_j is zero. Similarly, let $\tilde{\beta}_e(n) = \beta_e(n) + \varepsilon_e(n)$ .Then, define the output of the FENN as :

$$\tilde{b}i(n) = \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \acute{\alpha}_e(n) (w_{ij})^e + \beta_e(n) (g_{ij})^e \right) \tilde{\varphi}_j$$

$$= \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \alpha_e(n) (w_{ij})^e + \beta_e(n) (g_{ij})^e + \delta_e(n) (w_{ij})^e + \varepsilon_e(n) (g_{ij})^e \right) (\varphi_j + \Delta\varphi_j)$$

$$= \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \alpha_e(n)(w_{ij})^e + \beta_e(n)(g_{ij})^e \right)\varphi_j + \sum_{j=1}^{N} \left(\sum^{M} \alpha_e (n)(w_{ij})^e + \beta_e (n)(g_{ij})^e \right)\Delta\varphi_j + \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \delta_e(n)(w_{ij})^e + \varepsilon_e(n)(g_{ij})^e \right) (\varphi_j + \Delta\varphi_j)$$

$$= \hat{b}_i(n) + \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \alpha_e(n)(w_{ij})^e + \beta_e(n)(g_{ij})^e \right)\Delta\varphi_j + \sum_{j=1}^{N} \left(\sum^{M} \delta_e(n)(w_{ij})^e + \varepsilon_e(n)(g_{ij})^e \right)(\varphi_j+\Delta\varphi_j)$$

where $\hat{b}_i(n) = \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \alpha_e(n)(w_{ij})^e + \beta_e(n)(g_{ij})^e \right)\varphi_j$ is the output of the FENN when the measurement noise is zero. The corresponding sum-squared error at the output of the FENN is

$$\tilde{E}(n) = \tfrac{1}{2} \sum_{i=1}^{N} (\tilde{E}_i(n))^2 \quad \text{where} \quad \tilde{E}_i(n) = b_i - \tilde{b}_i(n)$$

$$= b_i - \tilde{b}_i(n) - \sum_{j=1}^{N} \left(\sum_{e=1}^{M} \alpha_e(n)(w_{ij})^e + \beta_e(n)(g_{ij})^e \right)\Delta\varphi_j - \sum_{j=1}^{N} \left(\sum^{M} \delta_e(n)(w_{ij})^e + \varepsilon_e(n)(g_{ij})^e \right)(\varphi_j + \Delta\varphi_j)$$

$$= E_i(n) - \sum_{j=1}^{M} \left(\sum_{e=1}^{N} \alpha_e(n)(w_{ij})^e + \beta_e(n)(g_{ij})^e \right)\Delta\varphi_j - \sum_{j=1}^{N} \left(\sum^{N} \delta_e(n)(w_{ij})^e + \varepsilon_e (n) (g_{ij})^e \right) (\varphi_j + \Delta\varphi_j)$$

and $E_i(n)$ is the error at the FENN output when the measurement noise is zero. Then, the CG update equations for $\acute{\alpha}$ and β are given by:

$$\acute{\alpha}_e(n) = \acute{\alpha}_e(n-1) + \eta\, \rho(n-1) \quad \ldots\ldots\ldots\ldots \quad [16a]$$

$$\tilde{\beta}_e (n) = \tilde{\beta}_e (n-1) + \acute{\eta}\, \bar{\rho}(n-1) \quad \ldots\ldots\ldots\ldots \quad [16b]$$

The search direction in [16] can be obtained from [15] .

-Applications

- One Dimensional Problems - Forward Model Results

The finite element model neural network(FENN) was tested using a one-dimensional version of Poisson's equation, which is a special case of [***] : $-\nabla \bullet (\varepsilon \nabla \varphi) = \rho$ .....[17]

For a one-dimensional problem, this reduces to:

$-(\partial/\partial x) (\varepsilon \partial\varphi/\partial x) = \rho$ ..... [18]

Several different examples based on [18] were used to test the performance of the FENN on the forward problem. The first problem that was tested was :

$\partial^2\varphi / \partial x^2 = 0$,$x \in [0,1]$,with the boundary conditions:$\varphi(0)= 0$and $\varphi(1)= K$,$K \in I-\{0\}$

The analytical solution to this problem is :

$$\Phi = \begin{cases} Kx , & x \in [0,1] \\ 0 , & \text{otherwise} \end{cases}$$

The FENN was tested by setting $\varepsilon = 1, \rho = 0$ and $K = 1$.The domain of interest [0,1] was divided into 10 elements (with 11 nodes) and the weights for the first layer of neurons were pre-computed. The results for this problem are shown in fig. (2) The solid line shows the analytical solution while the stars show the result determined by the FENN. This results also obtained using 10 elements and 11 nodes.

The second example that was tested was the one-dimensional Poisson's equation with $\varepsilon = 1$ and $\rho = -10$ with the same boundary conditions as above :

$\varphi(0) = 0$ and $\varphi(1) = K$, $K \in I-\{0\}$. Again, the domain of interest was divided up into ten elements. Similarly, for $\rho = 10$ and $K = 5$, the analytical solution is :

$$\varphi = \begin{cases} -5x^2 + 10x , & x \in [0,1] \\ 0 , & \text{otherwise} \end{cases}$$

and a comparison of the analytical solution (squares), the FEM solution and FENN solution for this problem is shown in Fig. (3). Again, the initial solution is indicated by the dashed line with the triangles.

These results indicate that the FENN is capable of accurately solving for the potential φ. The algorithm also converged in relatively few iterations (approximately 500 iterations on average for all problems) and the sum-squared error over all the out- put nodes was less than 0.0001 for all sets of results.

As mentioned above, one advantage of the FENN approach is the input - first hidden layer nodes can be computed once. The only changes necessary to solve the different

problems are changes in the input (ε) and the desired output (ρ).

- One Dimensional Problems - Inverse Model Results

The FENN was also used to solve several simple inverse problems based on Poisson's equation and Laplace equation. In all cases, the objective was to determine the value of ε for given values of ρ and φ .The results of this exercise are summarized below. The first problem involves determining ε given ρ =1 and φ = x , x ∈ [0,1] .The analytical solution to this inverse problem is :

$\varepsilon = K - x$, $x \in [0,1]$ and $K \in R$ ..... [19]

As seen from [19], this inverse problem, and all the others that follow, have an infinite

number of solutions and we expect the solution procedure to converge to one of these solutions depending on the initialization. Fig. 4(a) and (b) show the two solutions to this inverse problem for two different initializations(shown using triangles): ε = x and ε = 1+ x, respectively. The solution converges to ε =1- x and ε = 2- x respectively and the FENN solution is seen to match the analytical solution exactly. In order to further test the algorithm, the same problem was solved using four more initializations. The first two initialized ε to a constant value and the results are shown in Fig. (5) Similarly, Fig. (6) shows the results for a random initialization. In this case, the analytical result was obtained by drawing a straight line between the first and last values of ε . Similar results are presented in Fig.(7) for the inverse problem when ρ = -1 and φ = x, x ∈ [0,1]. The analytical solution is: ε = K+ x , x ∈ [0,1] and K ∈ R. The FENN results indicate that the algorithm converges to ε = x and ε =1+ x for the initialization solutions ε =1-x and ε = 2- x, respectively . The results presented for the inverse problem indicate that the solution is not unique and depends on the initialization. In order to obtain a unique solution, we need to impose constraints. For the second order differential equation [18] we need to constrain the value of ε at a known node on the sample in order to obtain a unique solution.

This is usually possible to assign in practice. This approach was applied to determine ε every- where given that φ and f are specified as follows in [18] : $\varphi = x^2$, $x \in [0,1]$ and

$f = -2K - 2\sin(x) - 2x\cos(x)$, $K \in R$. The analytical solution for this equation is $\varepsilon = \sin(x) + K$.

To solve this problem, we set K =1 and clamp the value of ε at x = 0 : $\varepsilon(x = 0) = K$.

The results of the inversion are shown in fig. (8 – 9). fig. 8(a) shows the comparison between the analytical solution and the FENN result . The initial value of ε was selected randomly and is shown in the figure as a dashed line. This result was obtained using 11 nodes and 10 elements in the corresponding finite element mesh. Fig. (8b) shows the error in the forcing function f at the FENN output. The squares indicate the desired value of f while the circles show the actual network output. This result indicates that, though the error in the function is small, the error in the inversion result is fairly large. Similar results for 21 nodes
(20 elements) in the mesh are shown in Fig.(9a).
It is seen that increasing the discretization significantly improves the solution. It should also be noted that the FENN inversion algorithm for 21 nodes has not converged to the desired error goal (as seen from Figure9(b))and a larger number of iterations are necessary to further improve the solution.

References

1. Baras, J. S. and Lavigna, A. (2002) Convergence of a neural network classifier, Systems Research Center, University of Maryland, College Park .

2. Turmon, J. (1995). Assessing generalization of feed forward neural networks, Ph.D. Thesis, August Cornell University.

3. He,S. ; Reif, K. and Unbehauen, R. (2000). Neural Networks , Vol 13, P.385-396 .

4. Cheney, E. W. and Light, W.(2000). Course in approximation theory, 2000, Pub. Books, Colepub. Company.

5. Yegnanarayana, B. (2000). Artificial neural networks, Newdelhi .

6. Mitchell, A. R. and Wait, R. (1978). The finite eement mthod in prtial dfferential cuations, 1978.

7. Wang,Q. and Aoyama, T. (2001). Neuron Computing, Vol. 1,NO.1.

8. Masuoka, R. (2000).IEICE TRANS.INF.&SYST.,Vol.E83-D,NO.6.

Table (1) Node-element connectivity array for the 2-element mesh

e	n(1,e)	n(2,e)	n(3,e)
1	1	2	3
2	4	3	2

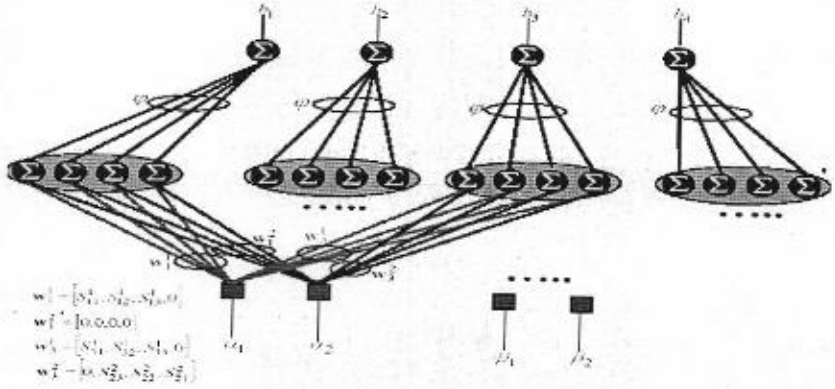


Fig. (1) The finite element neural network

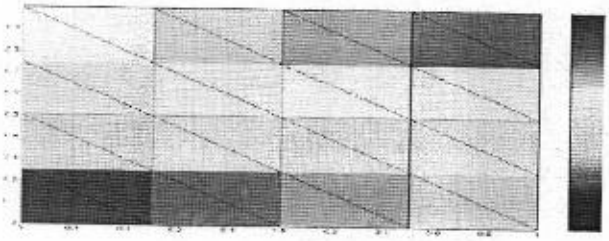


Fig. (2) Comparison of analytical solution and FENN solution for Laplace's equation with K=1

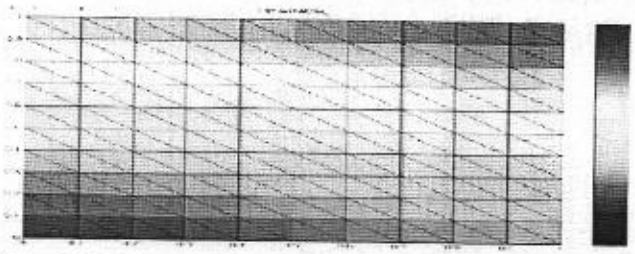
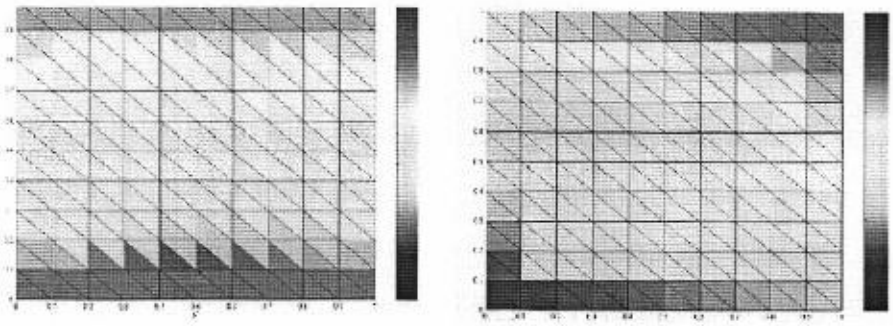
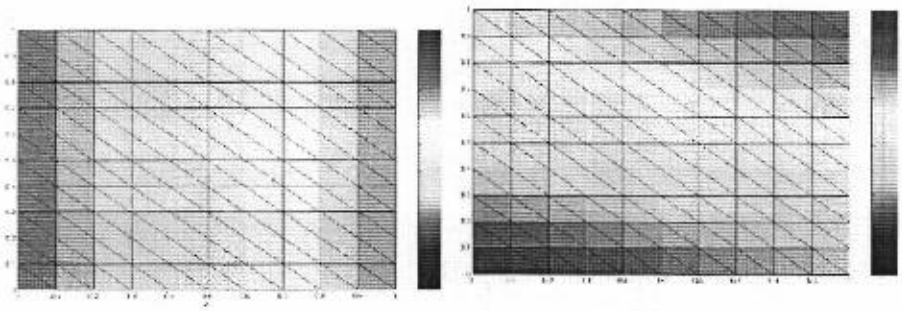


Fig. (3) Comparison of analytical, FEM and FENN solutions for Poisson's equation (ρ=10)



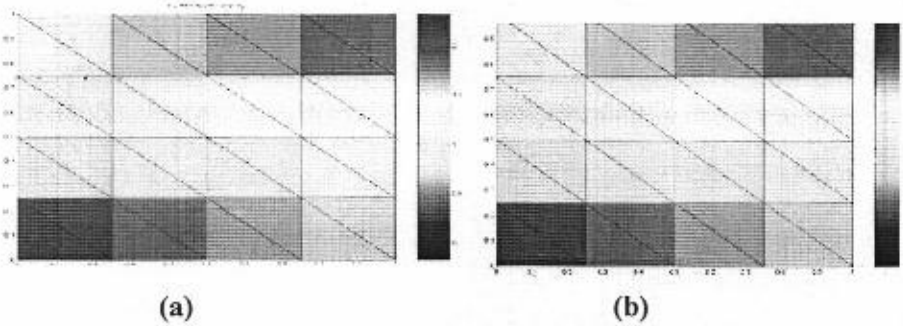
(a) (b)

Fig. (4) FENN inversion results for Poisson's equation with (a) initial solution $\varepsilon = x$ and (b) initial solution $\varepsilon = 1 + x$.



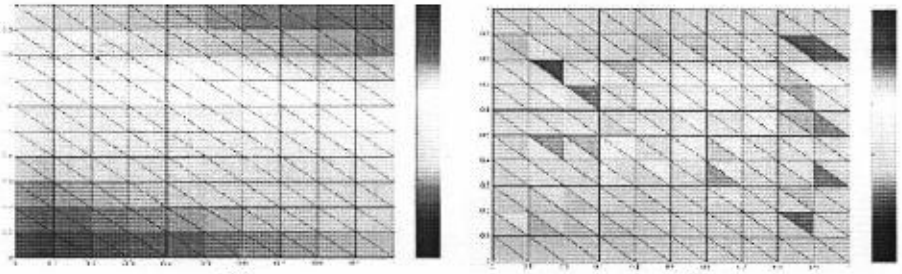
(a) (b)

Fig.(5) Inversion result for Poisson's equation with initial solution(a)$\varepsilon = 0.5$(b)$\varepsilon = 1$



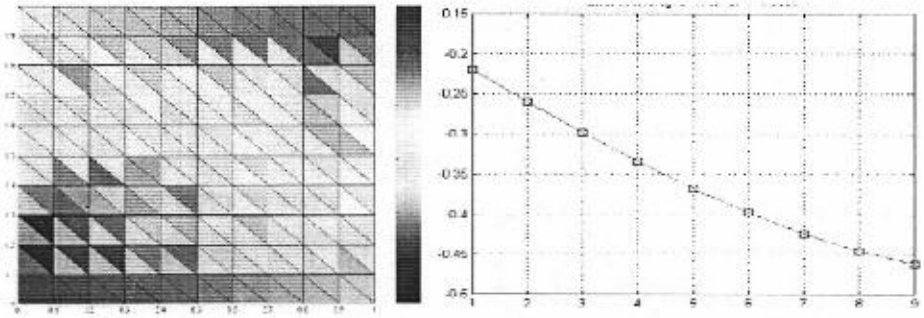
(a) (b)

Fig. (6) Inversion result for Poisson's equation with (a) random initialization 1 (b) random initialization 2



(a) (b)

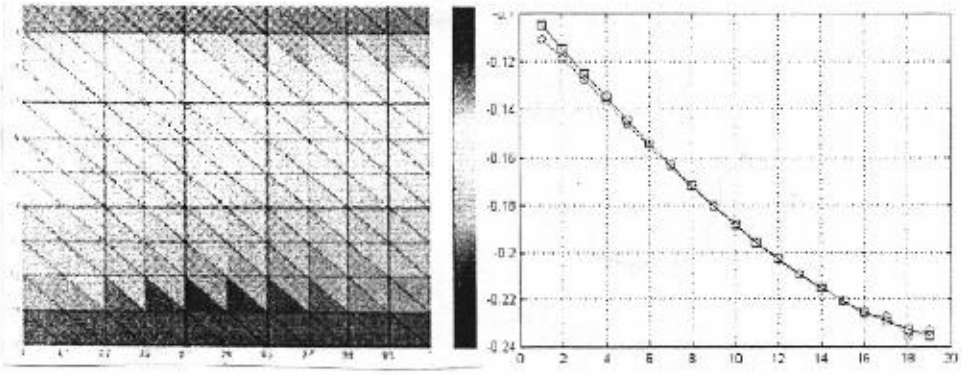
Fig. (7) FENN inversion results for Poisson's equation with initial solution(a) ε= 1- x (b) ε = 2- x.



(a)Constrained inversion result with eleven nodes

(b)Error in the forcing function for an eleven node discretization

Fig. (8)



(a) Constrained inversion results for 21 node discretizatization

(b) Error in the forcing

Fig(9)

مجلة ابن الهيثم للعلوم الصرفة والتطبيقية المجلد 19 (4) 2006

الشبكـــة العصبيـــة للعناصر المنتهيـــة و تطبيقاتها في مسائل المسار التقدمي و العكسي

لمى ناجـــي محمـــد توفيـــق
قسم الرياضيات، كلية التربية-أبن الهيثم ،جامعة بغداد.

الخلاصة

يتضمن البحـــث أعاده صياغـــة نمـــوذج العناصر المنتهيـــة فـــي هيكـــل شبكـــــة عصبيـــة، وذلك لأهمية النموذج والتي تكمن في إعطـــاء الحلول بدقـــة عاليـــة ولكثـــرة حسابات هذه الطريقـــة يجعلها قليلـــة الاستخدام لذلك استخدمنـــا الشبكـــات العصبيـــــة ذات المعالجـــة المتوازيـــة في إعادة صياغـــة النموذج لتذليـــــل تلـــك الصعوبـــــات وتطبيقها على مسائل البعد الثاني و في حل مسائـــل ذات المسار التقدمي و مسائـــل ذات المسار العكسي و من ثم ناقشـــنا أفضليـــة و عدم أفضليـــة تقارب هذا النمـــــوذج مـــع تحـــليل سرعـــة تأثـــر خوارزميـــة المسار العكسي في قياس الخطـــأ .