ELSEVIER

Applied Mathematical Modelling 23 (1999) 687–704

APPLIED MATHEMATICAL MODELLING

www.elsevier.nl/locate/apm

Approximation of functions and their derivatives: A neural network implementation with applications

T. Nguyen-Thien [a], T. Tran-Cong [a,b,*]

[a] *Faculty of Engineering and Surveying, University of Southern Queensland, Toowoomba, QLD 4350, Australia*
[b] *Department of Chemical Engineering, Rheology Research Centre, University of Wisconsin, 1415 Engineering Drive, Madison, WI 53706, USA*

Received 21 August 1997; received in revised form 15 December 1998; accepted 12 February 1999

Abstract

This paper reports a neural network (NN) implementation for the numerical approximation of functions of several variables and their first and second order partial derivatives. This approach can result in improved numerical methods for solving partial differential equations by eliminating the need to discretise the volume of the analysis domain. Instead only an unstructured distribution of collocation points throughout the volume is needed. An NN approximation of relevant variables for the whole domain based on these data points is then achieved. Excellent test results are obtained. It is shown how the method of approximation can then be used as part of a boundary element method (BEM) for the analysis of viscoelastic flows. Planar Couette and Poiseuille flows are used as illustrative examples.

Keywords: Neural network; Function approximation; Function derivatives approximation; Boundary element method

1. Introduction

Finite difference method (FDM) [1], finite element method (FEM) [2,3], finite volume method (FVM) [4] and boundary element method (BEM) [5] are among the most popular numerical methods for the solution of partial differential equations governing many problems in engineering sciences. Enormous progress has been achieved in the last three decades or so. However, these methods currently are still based on some discretisation of the domain of analysis into a number of finite elements (in the case of BEM, this is only true for non-linear problems, since for linear problems only the boundary of the analysis domain needs to be discretised). Furthermore, for problems with moving or unknown boundary, remeshing is an added difficulty. For practical analysis, automatic discretisation or meshing is a highly desirable but rarely available in general. Mesh generation could be time consuming and therefore account for a high proportion of analysis cost. The aim of this paper is to report an implementation of a neural-network-based approximation of functions of several variables and their derivatives and illustrate how this method of function approximation can be used in combination with methods such as non-linear BEM to completely eliminate the need of domain discretisation. The remaining of the paper is organised as

* Corresponding author. Fax: +608-262-5434; e-mail: trancong@che.wisc.edu

0307-904X/99/$ – see front matter
PII: S0307-904X(99)00006-2

follows. In Section 2 we will briefly review relevant definition of artificial neurons and feedforward neural networks (FFNNs) and establish our notations. Section 3 gives details of an FFNN approximation of functions and their derivatives. The training of such an FFNN is discussed in Section 4 while Section 5 describes examples of the numerical results obtained. In Section 6 we present applications of FFNN approximation method in a larger numerical scheme (BEM) for the analysis of viscoelastic flows. Section 7 concludes the paper.

2. Artificial neurons and feedforward neural networks

The foundation of neurocomputing research is well summarised in [6], which reprints in chronological order important works between 1890 and 1987, including psychological, biological, phisiological and mathematical aspects of neurocomputing theory. Another collection of important works on neurocomputing analysis can be found in [7]. An excellent perspective review of adaptive neural networks research between 1960 and 1990 is given in [8]. There is tremendous renewed interest in neurocomputing research in the last decade or so and many comprehensive texts are now available (for example, [9,10], just to name a few). In this section we briefly discuss a definition of artificial neurons and neural networks relevant to our present work. In order to be able to present succinct network diagrams we first summarise our notation in Fig. 1. Fig. 2 describes components of a model neuron and its computing ability. Referring to Fig. 2, given an input $\vec{O}_k^{l-1}$, a neuron k can compute an output O_k^l according to its prior training, represented by the weight vector $[w_{k0}^l, \vec{w}_k^{l\mathrm{T}}]^{\mathrm{T}}$ where superscript T denotes the transpose operation. The characteristics of neural computation are represented by the activation function A_k^l. Fig. 3 describes an abstract model of a multi-layer FFNN. In this model, neurons are arranged in layers with the following characteristics:

- The first layer is called the input layer. Neurons in this layer receive input signals to the network.
- Hidden layers: Neurons in each hidden layer may have connection to and hence receive signal from some or all neurons from the immediately preceding layer.
- The last layer is called the output layer. Neurons in this layer provide output signals computed by the network to the environment external to the network. (Note that Fig. 3 shows only one of many possible neurons in the output layer.)
- Neurons are not connected to other neurons in the same layer.

In the rest of this paper we only discuss three layer FFNNs consisting of an input layer, a hidden layer and an output layer. Furthermore, we consider fully connected FFNNs in which a neuron will receive signals from each and every neuron in the immediately preceding layer.

3. Approximation of functions and their derivatives by FFNN

The capability of FFNN with as few as one hidden layer to approximate functions has been theoretically proven by a number of authors (e.g. [11–14], just to name a few). Here, we are also interested in the simultaneous calculation of the derivatives of functions thus approximated. Fortunately, FFNN can be extended for this purpose as shown by Hornik et al. [15,16]. Fig. 4 describes an FFNN with a multi-neuron input layer, a single multi-neuron hidden layer and a single-neuron output layer. The activation functions of the input and output layer neurons are simply the identity function. On the other hand the binary or logistic sigmoidal function is used by the hidden layer neurons. This sigmoidal function and its derivative are given by

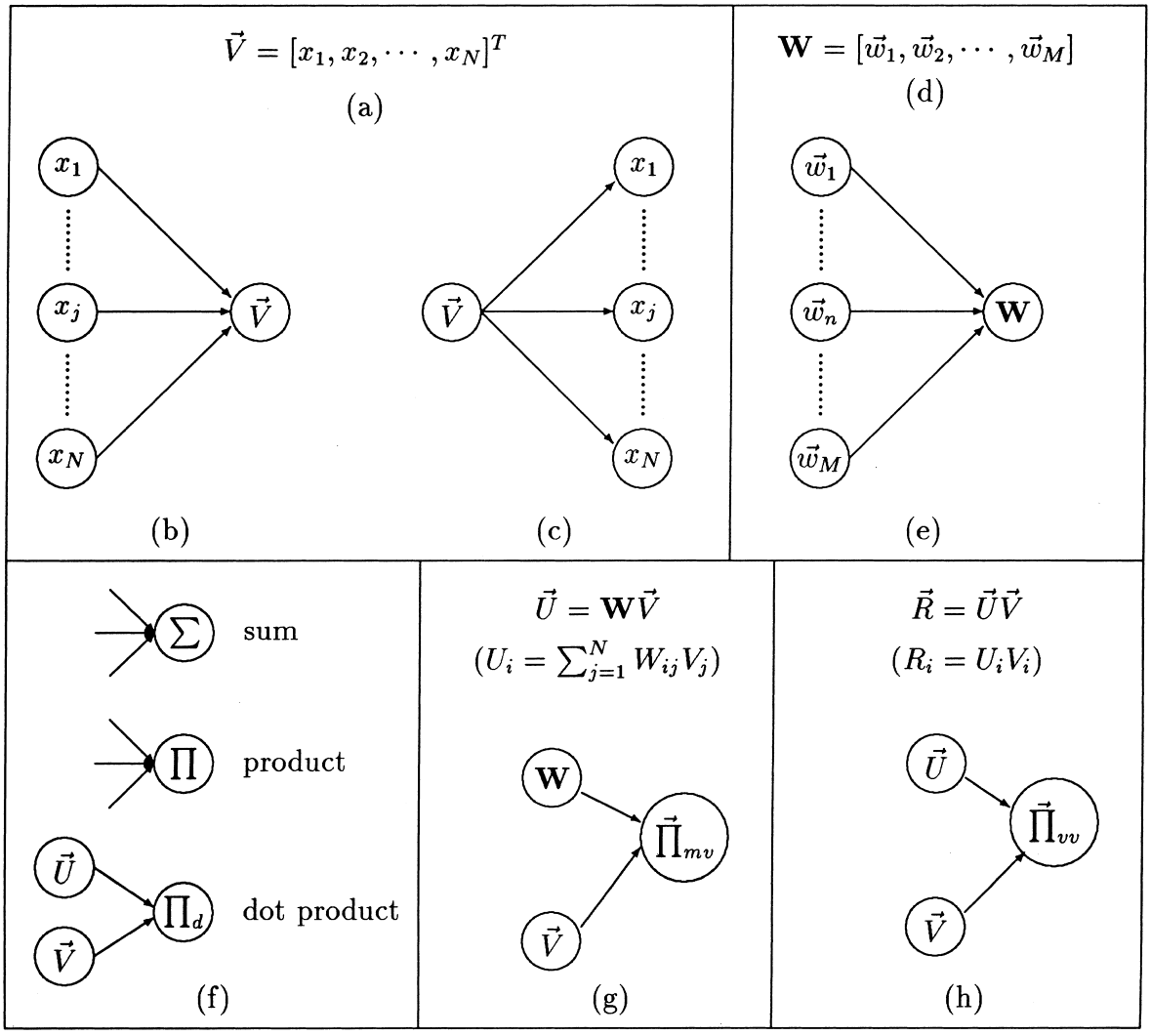

Fig. 1. Definition of the vector and matrix forming units and related operations: (a) vector $\vec{V}$ is formed with N components; (b) network diagram for vector forming operation; (c) network diagram for vector component extraction; (d) matrix $\mathbf{W}$ is formed with M $N \times 1$-vectors; (e) network diagram for matrix forming operation; (f) network diagrams for sum of scalars, product of scalars and dot product of two vectors; (g) network diagram for matrix–vector multiplication and (h) network diagram for vector–vector product.

$$A_n^{\mathrm{h}}(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \tag{1}$$

$$DA_n^{\mathrm{h}}(x) \equiv \frac{\mathrm{d}A_n^{\mathrm{h}}(x)}{\mathrm{d}x} = A_n^{\mathrm{h}}(x)\left[1 - A_n^{\mathrm{h}}(x)\right]. \tag{2}$$

Thus, given an input $\vec{x} \in \mathbb{R}^N$, the function is computed by the network according to

$$f(\vec{x}) = \sum_{j=1}^{M} w_{1j}^{\mathrm{o}} A_j^{\mathrm{h}}(\vec{x}^{\mathrm{T}} \vec{w}_j^{\mathrm{h}}), \tag{3}$$

and the corresponding first order derivatives are given by [15]

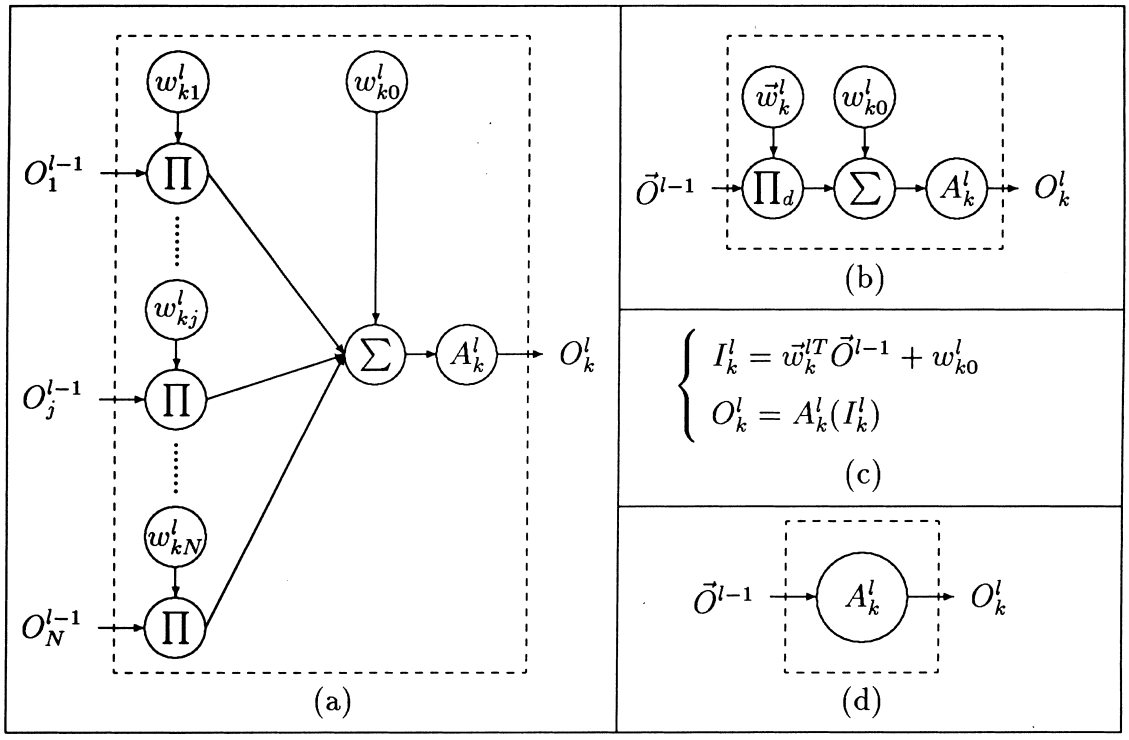

Fig. 2. A neuron model: (a) internal computation mechanism of an artificial neuron k of layer l with input vector $\vec{O}^{l-1} = [O_1^{l-1}, O_2^{l-1}, \ldots, O_N^{l-1}]^{\mathrm{T}}$ from layer $(l-1)$. The weight vector $\vec{w}_k^l = [w_{k1}^l, \ldots, w_{kN}^l]^{\mathrm{T}}$, where superscript T denotes the transpose operation, is an internal property of neuron k of layer l. w_{k0}^l is usually called the bias. A_k^l is the activation function of the model neuron and O_k^l is the output from the model neuron k of layer l; (b) a compact representation of the model; (c) matrix notation of the internal computation of the model neuron and (d) an abstract model neuron showing input vector and a single output.

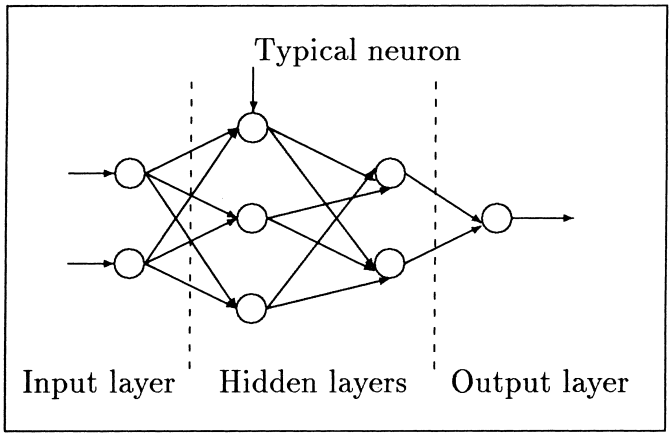

Fig. 3. Abstract model of a multi-layer neural network.

$$\frac{\partial f(\vec{x})}{\partial x_i} = \sum_{j=1}^{M} w_{1j}^{\mathrm{o}} w_{ji}^{\mathrm{h}} DA_j^{\mathrm{h}}(\vec{x}^{\mathrm{T}} \vec{w}_j^{\mathrm{h}}), \quad i = 1, \ldots, N, \tag{4}$$

where M is the number of neurons in the hidden layer. We define the synaptic weight vector as

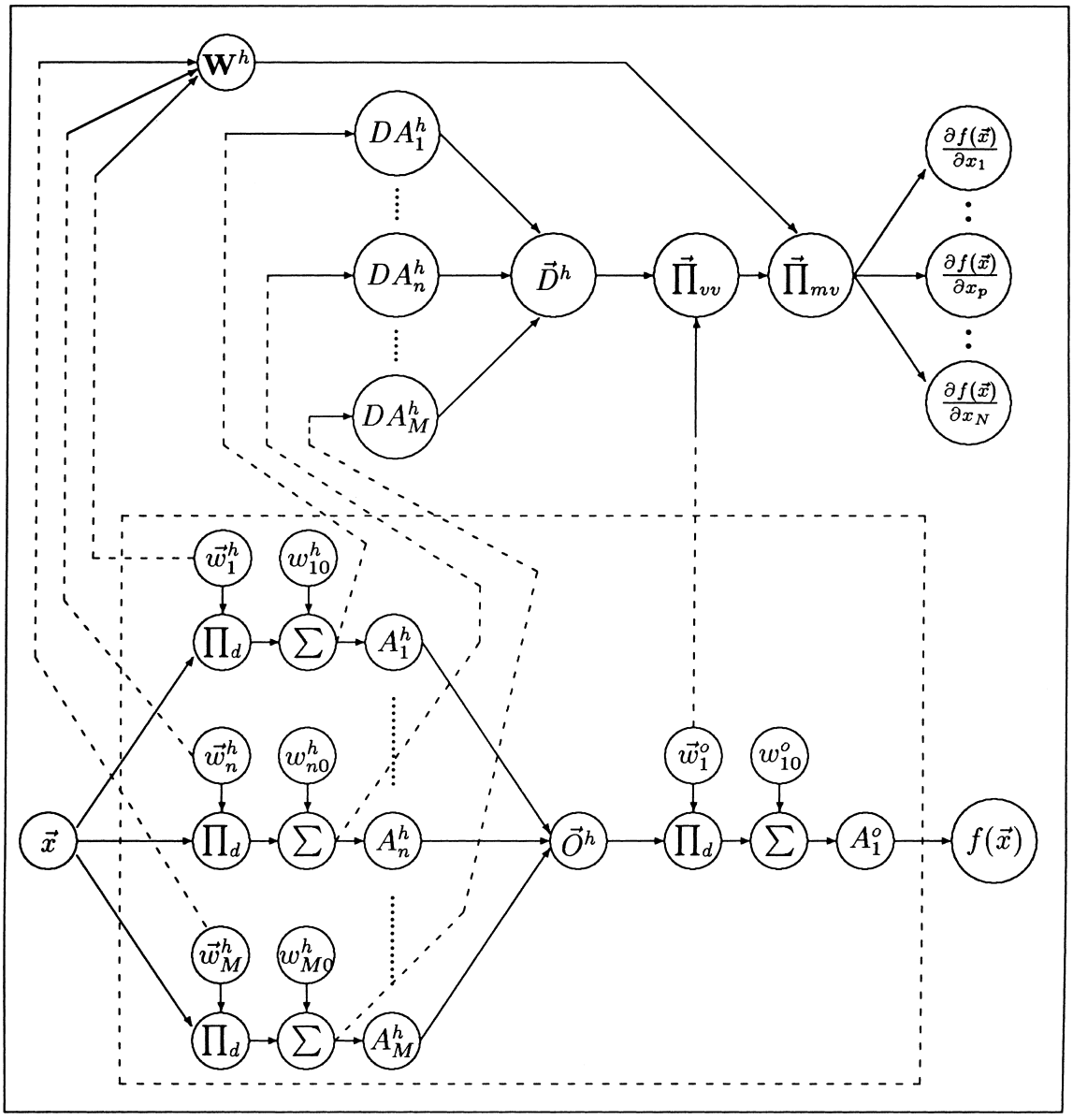

Fig. 4. A three-layer neural network for approximation of a function $f(\vec{x})$ and its first partial derivatives. Superscript h denotes the hidden layer and superscript o the output layer. DA_n^{h} is the derivative of the activation function A_n^{h} of neuron n of layer h. Note that we have only a single neuron in the output layer whose activation function A_1^{o} retains the subscript 1 for consistency.

$$\vec{W} = \left[w_{10}^{\mathrm{h}}, \vec{w}_1^{\mathrm{hT}}, \ldots, w_{M_0}^{\mathrm{h}}, \vec{w}_M^{\mathrm{hT}}, w_{10}^{\mathrm{o}}, \vec{w}_1^{\mathrm{oT}}\right]^{\mathrm{T}}. \tag{5}$$

Note that the biases $w_{k0}^{\mathrm{h}},\ k = 1, \ldots, M$ and w_{10}^{o} are also included. Second derivatives of the function can be derived as

$$\frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_k} = \sum_{j=1}^{M} w_{1j}^{\mathrm{o}} w_{ji}^{\mathrm{h}} w_{jk}^{\mathrm{h}} DA_j^{\mathrm{h}}(\vec{x}^{\mathrm{T}}\vec{w}_j^{\mathrm{h}})\left[1 - 2A_j^{\mathrm{h}}\left(\vec{x}^{\mathrm{T}}\vec{w}_j^{\mathrm{h}}\right)\right], \quad i = 1, \ldots, N, \quad k = 1, \ldots, N. \tag{6}$$

Thus, given an activation function such as Eq. (1), a non-linear function can be approximated by Eq. (3) and its derivatives by Eqs. (4) and (6). It remains to determine the synaptic weight vector (5) from a given set of T data points such as $\{(\vec{x}_1, f_{1t}), (\vec{x}_2, f_{2t}), \ldots, (\vec{x}_T, f_{Tt})\}$. Note that the weight vector is not unique due the symmetry of fully connected FFNNs.

4. FFNN training algorithm

Once a set of discrete data is available (in the context of our applications, this is given by a boundary element method), the FFNN can be trained to approximate or generalise the function over the domain. FFNN training is commonly posed as an optimisation problem in the weight space. The non-linear least squares objective function in this case is defined by

$$E(\vec{W}) = \sum_{p=1}^{T_1} e_p^2, \tag{7}$$

where T_1 is the number of training patterns, p denotes a pattern and

$$e_p^2 = \left[f_{pt}(\vec{x}_p) - f_{pc}(\vec{x}_p) \right]^2 \tag{8}$$

is the squared error associated with the training pattern p; f_{pt} is the target or desired output; and f_{pc} is the computed output corresponding to the input $\vec{x}_p$. Fig. 5 describes the algorithm for the approximation of a function of N variables. Once the network is trained successfully, the calculation of the function's first and second order partial derivatives is given by Eqs. (4) and (6) respectively. (Note that the training is done using only function values as shown in the dash-boxed part of Fig. 4.) The error vector is defined by

$$\vec{e} = [e_1, e_2, \ldots, e_{T_1}]^{\mathrm{T}}. \tag{9}$$

Embedded in the above algorithm are two subprocedures, one dealing with back propagation (BP) and the other with the problem of local minima. We will discuss these two procedures shortly. The general flow of the algorithm is as follows. The T given data points (training patterns) are divided into two sets, one of which contains T_1 patterns for the actual training and the remaining $T_2 = T - T_1$ patterns are used in the local minima avoidance procedure. The two discrete data sets are "interleaved" as much as possible in the following sense. Suppose the coordinates (1D) of the T data points are arranged in ascending order as

$$\{x_1, x_2, \ldots, x_T\}.$$

Then the T_1 data points for the first set are chosen to be

$$\{x_1, x_3, \ldots\},$$

and those (T_2) for the second set to be

$$\{x_2, x_4, \ldots\}.$$

This is essential for the detection of "over-fitting" the data since the resultant curve has already passed through the data points within the chosen tolerance. We start by choosing an initial number of hidden neurons, M_0 (as discussed in Section 4.2), and set a limit on the number of iterations for optimisation, $I_{\max}$. At the beginning of each iteration the weight vector is initialised randomly according to a method proposed by Nguyen and Widrow [17], which was shown to improve the training efficiency. Randomness of the initial guess of the weight vector is essential in this algorithm due to the symmetry of fully connected FFNNs. The corresponding error is

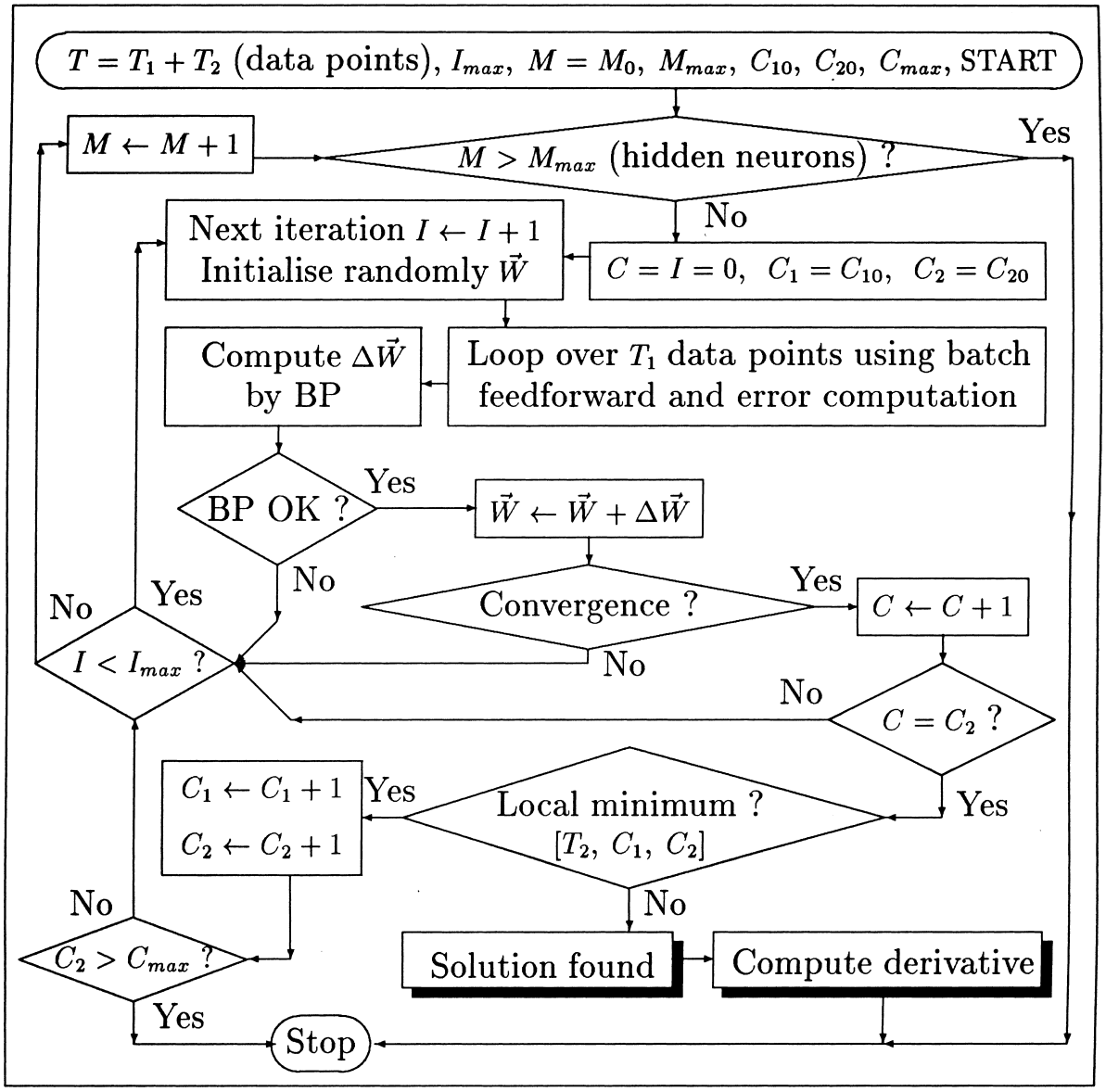

Fig. 5. Computational flowchart: the network is trained with T_1 data points using batch feedforward method and error BP, as explained in Section 4. The remaining data points T_2, together with parameters C_1, C_2 are used in the local minima avoidance algorithm, also as explained in Section 4.

calculated and a corresponding correction to the weight vector is computed. This is repeated until the weight vector is optimised, i.e.

$$E(\vec{W}) = \sum_{p=1}^{T_1} e_p^2 < \text{tol}, \tag{10}$$

where we have set a tolerance of tol $= 10^{-6}$. Such an optimised result is stored and the counter C is incremented by 1. When the counter C reaches a set value, C_2, the local minima avoidance procedure is called to decide whether a global minimum has been achieved. If a global minimum

has not been achieved, the limit C_2 and parameter C_1 are both incremented ready for another attempt. We will now discuss the details of the BP and local minima avoidance procedure.

4.1. Back propagation procedure

Back propagation (BP) is probably the most commonly used method of training for FFNN. The work by Rumelhart et al. [18] is generally recognised as the seminal work on BP but the idea of BP has been used earlier (see [19]). In the batch version of this procedure, the first step is to loop over all training patterns and compute the corresponding errors by feeding the input forward through the network. The error (7) is then computed and used to calculate a correction to the weight vector. In this work we calculate a corrective vector according to the Levenberg–Marquardt algorithm [20–22]:

$$\Delta\vec{W} = (\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J}^{\mathrm{T}}\vec{e}, \tag{11}$$

where

$$\mathbf{J} = \frac{\partial\vec{e}}{\partial\vec{W}}$$

is the Jacobian matrix, λ is a parameter, $\mathbf{I}$ is the identity matrix and $\vec{e}$ is the current error vector. In order to achieve fast convergence, a small iterative process is embedded within the BP procedure to choose an optimum value for λ.

4.2. Local minima avoidance procedure

For FFNNs of the kind considered here it was proved that it is possible to have an architecture such that the error surface is free from local minima [23]. However, this architecture is too expensive in practice because it requires the number of neurons to be equal to the number of training patterns and it is known that the global optimisation problem is intractable (see, e.g. [24]). Our experience in training our FFNNs shows that BP algorithm can get stuck at saddle points. This problem is more severe as the number of neurons increases. Therefore in practice we choose the starting number of neurons (M_0) to be small (we have used $2 \leqslant M_0 \leqslant 7$) and let the algorithm slowly increases M_0 as required by the accuracy criterion. In our present FFNN implementation, it is certain that there exist more than one global minimium due to the symmetry of fully connected networks. This fact is put to effective use in providing a good heuristic for a practical global optimisation of the weight vector by Chen [24] who proposed the global optimization training (GOT) algorithm. GOT is based on the following heurtistic to ensure high probability that the minimum obtained is global. The heuristic is "that different local minima tend to have different values while global minima always have the same value" [24]. Thus we need to find a number of minima using local optimisation tools, which is denoted by the parameter C_2 in our implementation. Then for each solution found, the overall error is computed according to

$$E = \sum_{p=1}^{T} e_p^2,$$

where the error is summed over all $T = T_1 + T_2$ data points. Thus we obtain C_2 error measures as

$$E_i \equiv E(\vec{W}_i), \quad \vec{W}_i \in \{\vec{W}_1, \vec{W}_2, \ldots, \vec{W}_{C_2}\}.$$

Assume without loss of generality that

$$E_1 \leqslant E_2 \leqslant \cdots \leqslant E_{C_1} \leqslant \cdots \leqslant E_{C_2}.$$

If

$$E_1 \approx E_2 \approx \cdots \approx E_{C_1}$$

then according to the above heuristic $\vec{W}_1$ is the best approximation of global minima. For further details, see [24]. We start with $C_1 = 2$ and $C_2 = 5$. Although $C_{\max}$ was used to provide a terminating criterion to guard against infinite loop, in practice our experience shows that for our problems the algorithm always terminates properly, i.e. C_2 never reaches $C_{\max}$.

5. Numerical results

In the following sections we will present examples of FFNN approximation of function of one, two and three variables, which demonstrate that our implementation is successful. Fully connected three layer FFNNs are used for all examples. In the following discussion we refer to "maximum errors" produced by our FFNN approximation. This terminology is understood in the following sense: once the FFNN is successfully trained, we calculate its output at many randomly selected inputs which are different and as distant from the training data as possible. We chose inputs to be from every interval defined by adjacent data points and the "maximum error" is found from this discrete set.

5.1. Example 1: a function of one variable

Fig. 6 shows the plot of function $f(x) = x^3 + x + 0.5$, $-3 \leqslant x \leqslant 2$, its derivatives and their approximation by a 1–5–1 (input layer of 1 neuron, one hidden layer of 5 neurons and one output layer of 1 neuron) FFNN. The weights and biases obtained are shown in Table 1. The numbers of data points used are $T_1 = 26$ and $T_2 = 25$. The accuracy is excellent with the maximum error in function values of 0.0131% and the maximum error in function derivative values of 0.0227% (first derivative) and 0.643% (second derivative). These errors are calculated at points selected roughly at the mid-point of all intervals between the given data points to ensure maximum distance from these data points. This is essential for the detection of "over-fitting" the data since the resultant curve has already passed through the data points within the chosen tolerance.

5.2. Example 2: functions of two variables

5.2.1. $f(x,y) = 2x^3 - xy^2 + 0.5, 1 \leqslant x \leqslant 4, 1 \leqslant y \leqslant 4$

The first function of two variables chosen to test the implementation is the function shown above. The numbers of data points used are $T_1 = 100$ and $T_2 = 81$. The resultant network is a 2–10–1 FFNN that is capable of approximating the function and its partial derivatives with a maximum error of 0.015% (for f), 0.0179% (for $\partial f/\partial x$) and 0.0142% (for $\partial f/\partial y$). The errors in second derivatives are 1.28% (for $\partial^2 f/\partial x^2$), 1.534% (for $\partial^2 f/\partial x \partial y$) and 1.5311% (for $\partial^2 f/\partial y^2$). More accurate results can be obtained by "refining the mesh". This is effectively done in this case by resizing the intervals to $1 \leqslant x \leqslant 2.5$, $1 \leqslant y \leqslant 2.5$ while keeping the number of data points the same. The number of neurons required was found to decrease from 10 to 8 and the accuracies improve significantly with the maximum recorded as follows: 0.012% (for f), 0.015% (for $\partial f/\partial x$) and 0.0121% (for $\partial f/\partial y$). The errors in second derivatives are 0.1600% (for $\partial^2 f/\partial x^2$), 0.7169% (for $\partial^2 f/\partial x \partial y$) and 0.7808% (for $\partial^2 f/\partial y^2$). Table 2 records the weight results for the refined mesh.

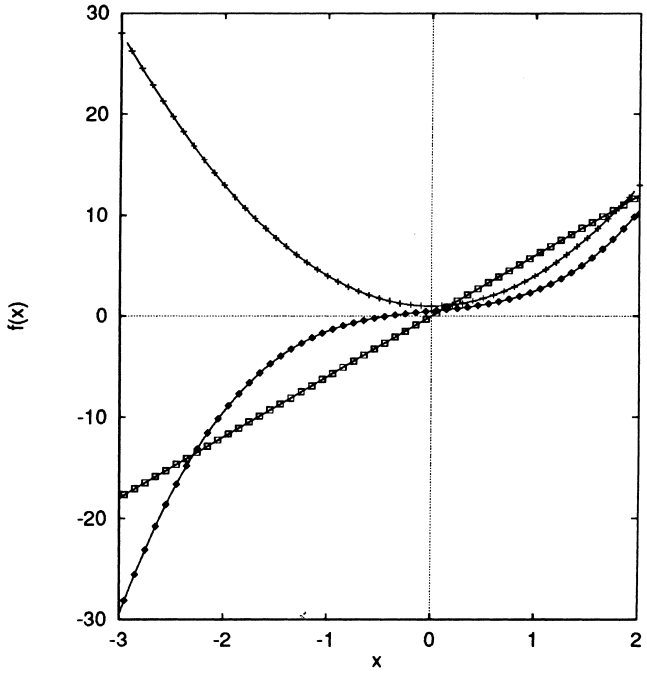

Fig. 6. Plot of function $f(x) = x^3 + x + 0.5,\ -3 \leqslant x \leqslant 2$ (solid line) and its approximation ($\diamondsuit$) by a 1–5–1 FFNN. The first derivative of the function $f'(x) = 3x^2 + 1$ is shown (solid) together with its NN-computed values (+). The second derivative of the function $f''(x) = 6x$ is also shown (solid) together with its NN-computed values ($\square$).

Table 1
1–5–1 NN results for function $f(x) = x^3 + x + 0.5$

Neuron n	1	2	3	4	5
w^{h}_{n0}	6.37265924	19.13443282	3.90595308	0.06480986	−3.08643283
w^{h}_{n1}	1.27551186	−43.52641846	−0.75727779	−0.66606098	−0.84307379
w^{o}_{10}	137.27198260				
w^{o}_{1n}	134.13409825	0.00009480	−293.95177053	42.00490902	−96.32878920

There is only one input and therefore the weights for the hidden layer are w^{h}_{nj} with $j = 0$ (bias) and $j = 1$ (x). There is only one output and therefore the weights for the output layer are w^{o}_{10} (bias) and w^{o}_{1n}.

Table 2
2–8–1 NN results for function $f(x, y) = 2x^3 - xy^2 + 0.5,\ 1 \leqslant x \leqslant 2.5,\ 1 \leqslant y \leqslant 2.5$

Neuron n	w^{h}_{n0}	w^{h}_{n1}	w^{h}_{n2}	w^{o}_{1n}	w^{o}_{10}
1	−4.53289669	1.33653055	−0.14529692	133.74024001	−52.89632886
2	6.00887854	−0.48006699	−1.26292457	20.50263781	
3	−0.64086357	−0.73775831	0.68514044	−48.28927391	
4	0.22221414	−0.99013770	0.10893943	72.12872526	
5	1.13906416	−0.86227267	0.80847187	−26.02429585	
6	−0.19731207	0.22226275	0.60746141	62.10860122	
7	−6.09466574	−16.70836841	1.78144855	−3.30508473	
8	−12.92944275	4.06598457	−0.42967607	9.22629811	

There are two inputs and therefore the weights for the hidden layer are w^{h}_{nj} with $j = 0$ (bias), $j = 1$ (x) and $j = 2$ (y). There is only one output and therefore the weights for the output layer are w^{o}_{10} (bias) and w^{o}_{1n}.

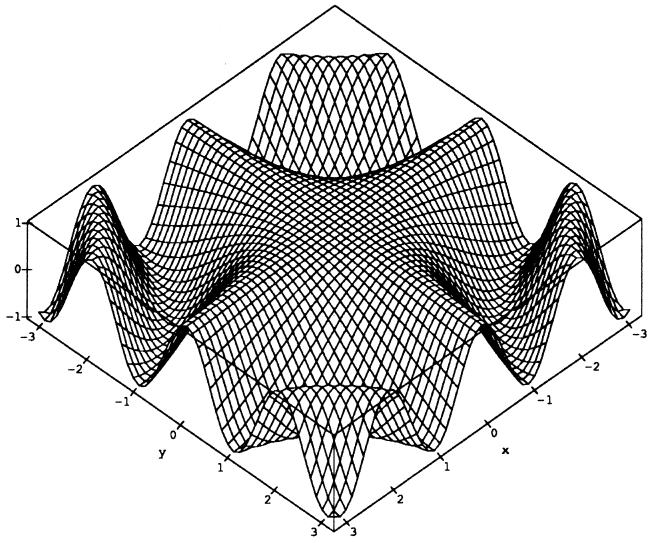

Fig. 7. Plot of function $\cos xy,\ -\pi \leqslant x \leqslant \pi, -\pi \leqslant x \leqslant \pi$.

5.2.2. $f(x,y) = \cos xy, -\pi \leqslant x \leqslant 0, -\pi \leqslant y \leqslant 0$

The above function (see Fig. 7) is also studied and it was found that subregioning of the domain can provide effective means of keeping the number of neurons down and improving accuracy and convergence speed. The domain in this case was divided into nine regions by taking the following subintervals in x and y:

$$-\pi \leqslant x \leqslant -\pi + 1, \quad -\pi + 1 \leqslant x \leqslant -\pi + 1, \quad -\pi + 2 \leqslant x \leqslant 0$$

and

$$-\pi \leqslant y \leqslant -\pi + 1, \quad -\pi + 1 \leqslant y \leqslant -\pi + 1, \quad -\pi + 2 \leqslant y \leqslant 0$$

Each subregion is approximated using $T_1 = 100,\ T_2 = 81$. The resultant number of neurons varies between 10 and 12. For example, in the subregion defined by

$$-\pi \leqslant x \leqslant -\pi + 1, \quad -\pi \leqslant y \leqslant -\pi + 1$$

the resultant network is a 2–11–1 FFNN that is capable of approximating the function and its partial derivatives with a maximum error of 0.1348% (for f), 0.9149% (for $\partial f/\partial x$) and 0.5854% (for $\partial f/\partial y$). The errors in second derivatives are 4.6070% (for $\partial^2 f/\partial x^2$), 4.5386% (for $\partial^2 f/\partial x \partial y$) and 3.9875% (for $\partial^2 f/\partial y^2$). Again, more accurate results can be obtained by "refining the mesh". This is effectively done in this case by resizing the intervals to $-\pi \leqslant x \leqslant -\pi + 0.5,\ -\pi \leqslant y \leqslant -\pi + 0.5$ while keeping the number of data points the same. The number of neurons required was found to decrease from 11 to 9 and overall accuracies improve with the maximum recorded as follows: 0.0366% (for f), 0.8166% (for $\partial f/\partial x$) and 1.2204% (for $\partial f/\partial y$). The errors in second derivatives are 1.1025% (for $\partial^2 f/\partial x^2$), 1.5459% (for $\partial^2 f/\partial x \partial y$) and 1.3671% (for $\partial^2 f/\partial y^2$). Table 3 records the weight results for the refined mesh pertinent to the forementioned subregion.

5.3. Example 3: functions of three variables

5.3.1. $f(x,y,z) = x^2 + yz + 2z^2, 1 \leqslant x \leqslant 2, 2 \leqslant y \leqslant 3, 1 \leqslant z \leqslant 2$

This function is approximated using $T_1 = 218,\ T_2 = 125$. The resultant number of neurons is 9 and the FFNN is capable of approximating the function and its partial derivatives with a maximum error of 0.0019% (for f), 0.2441% (for $\partial f/\partial x$), 0.2063% (for $\partial f/\partial y$) and 0.1187% (for

Table 3
2–9–1 NN results for function $f(x,y) = \cos(xy),\ -\pi \leqslant x \leqslant -\pi + 0.5,\ -\pi \leqslant y \leqslant -\pi + 0.5$

Neuron n	w^{h}_{n0}	w^{h}_{n1}	w^{h}_{n2}	w^{o}_{1n}	w^{o}_{10}
1	−34.31159583	−6.58295313	−6.10968871	−0.36551527	−1.76434483
2	−63.95862404	−10.15498962	−10.82598274	−0.12301715	
3	22.02818475	3.68642412	3.92742902	5.73293136	
4	20.30491734	−5.25705183	10.60392325	−0.02585425	
5	−67.29634996	−10.88610464	−11.78140428	−0.04554958	
6	−15.89373495	−1.66851817	−3.58008664	3.12669118	
7	120.02485036	18.55512466	19.17275993	−0.08361149	
8	31.30910662	4.48125828	7.21523389	0.14988672	
9	16.04365737	3.43204053	1.75277760	−2.95130387	

There are two inputs and therefore the weights for the hidden layer are w^{h}_{nj} with $j = 0$ (bias), $j = 1$ (x) and $j = 2$ (y). There is only one output and therefore the weights for the output layer are w^{o}_{10} (bias) and w^{o}_{1n}.

$\partial f/\partial z$). The errors in second derivatives are 4.4839% (for $\partial^2 f/\partial x^2$), 3.6367% (for $\partial^2 f/\partial y \partial z$) and 3.7722% (for $\partial^2 f/\partial z^2$). Furthermore, the remaining derivatives are expected to be zero and the calculated values are $\partial^2 f/\partial x \partial y = 0.49\text{E}-3$, $\partial^2 f/\partial x \partial z = 0.54\text{E}-3$ and $\partial^2 f/\partial y^2 = 0.27\text{E}-3$. More accurate results can be obtained by "refining the mesh" as discussed in previous examples. This is effectively done in this case by resizing the intervals to $1 \leqslant x \leqslant 1.6,\ 2 \leqslant y \leqslant 2.6,\ 1 \leqslant z \leqslant 1.6$ while keeping the number of data points the same. The number of neurons required was found to decrease from 9 to 8 and overall accuracies improve with the maximum recorded as follows: 0.0008% (for f), 0.01916% (for $\partial f/\partial x$), 0.0378% (for $\partial f/\partial y$), 0.0094% (for $\partial f/\partial z$), 0.9613% (for $\partial^2 f/\partial x^2$), 0.5866% (for $\partial^2 f/\partial y \partial z$), 0.6516% (for $\partial^2 f/\partial z^2$). The remaining derivatives are expected to be zero and the calculated values are $\partial^2 f/\partial x \partial y = 0.39\text{E}-4$, $\partial^2 f/\partial x \partial z = 0.65\text{E}-4$ and $\partial^2 f/\partial y^2 = 0.56\text{E}-4$. Table 4 records the weight results for the refined mesh.

5.3.2. $f(x,y,z) = x^2 + y^2 + z^2, 1 \leqslant x \leqslant 2, 1 \leqslant y \leqslant 3, 1 \leqslant z \leqslant 2$

This function is approximated using $T_1 = 150,\ T_2 = 81$. The resultant number of neurons is 10 and the FFNN is capable of approximating the function and its partial derivatives with a maximum error of 0.6202% (for f), 3.8843% (for $\partial f/\partial x$), 3.1868% (for $\partial f/\partial y$) and 0.1980% (for $\partial f/\partial z$). The errors in second derivatives are 2.8543% (for $\partial^2 f/\partial x^2$), 2.7681% (for $\partial^2 f/\partial y^2$) and 3.0254% (for $\partial^2 f/\partial z^2$). Furthermore, the remaining derivatives are expected to be zero and the calculated values are $\partial^2 f/\partial x \partial y = 0.168\text{E}-3$, $\partial^2 f/\partial x \partial z = 0.271\text{E}-3$ and $\partial^2 f/\partial x \partial z = 0.653\text{E}-3$. As before we increase the density of data points to obtain more accurate results by resizing the

Table 4
3–8–1 NN results for function $f(x,y,z) = x^2 + yz + 2z^2,\ 1 \leqslant x \leqslant 1.6,\ 2 \leqslant y \leqslant 2.6,\ 1 \leqslant z \leqslant 1.6$ (n denotes the nth neuron)

n	w^{h}_{n0}	w^{h}_{n1}	w^{h}_{n2}	w^{h}_{n3}	w^{o}_{1n}	w^{o}_{10}
1	30.2089112	−19.5872877	−2.0983568	0.5284904	−0.0007175	56.6857736
2	3.1151426	−0.0103599	−0.2235200	−0.8984387	−53.1758818	
3	3.0630405	−0.4705316	−0.4898145	0.0212346	12.0587201	
4	31.9941632	−8.0474756	−10.4430264	0.6221967	0.0003500	
5	−8.7476852	0.0598856	1.2831900	5.0796606	0.0520236	
6	−24.8027107	−9.2813775	18.1038390	−3.0273884	0.0000064	
7	3.0953551	−1.1165617	−0.12570990	0.0285581	−19.2548478	
8	14.9489496	−10.9234660	−1.2117528	0.3045551	−0.0054784	

There are three inputs and therefore the weights for the hidden layer are w^{h}_{nj} with $j = 0$ (bias), $j = 1$ (x), $j = 2$ (y) and $j = 3$ (z). There is only one output and therefore the weights for the output layer are w^{o}_{10} (bias) and w^{o}_{1n}.

Table 5
3–8–1 NN results for function $f(x,y,z) = x^2 + y^2 + z^2,\ 1 \leqslant x \leqslant 1.5,\ 2 \leqslant y \leqslant 2.5,\ 1 \leqslant z \leqslant 1.5$ (n denotes the nth neuron)

n	w^{h}_{n0}	w^{h}_{n1}	w^{h}_{n2}	w^{h}_{n3}	w^{o}_{1n}	w^{o}_{10}
1	48.1302262	−0.5900390	−2.1827858	−30.0227116	−0.0019981	7.4574881
2	−3.0396549	−0.0084509	1.6867427	−0.0940673	7.2387652	
3	−3.0583946	0.0162728	0.0764126	1.3694582	11.1062403	
4	44.3952360	−30.6912407	−0.4254922	0.7942585	−0.0025508	
5	3.0578455	−1.4991477	−0.0063356	0.0181081	−9.2937145	
6	−0.6832375	−16.0975773	14.4844952	−2.2641171	−0.0000476	
7	−17.6950544	−0.0708539	11.7395797	−0.6693099	0.0326948	
8	−2.2526392	−16.3358163	11.3609939	8.2521348	0.0000003	

There are three inputs and therefore the weights for the hidden layer are w^{h}_{nj} with $j = 0$ (bias), $j = 1$ (x), $j = 2$ (y) and $j = 3$ (z). There is only one output and therefore the weights for the output layer are w^{o}_{10} (bias) and w^{o}_{1n}.

intervals to $1 \leqslant x \leqslant 1.5,\ 2 \leqslant y \leqslant 2.5,\ 1 \leqslant z \leqslant 1.5$ while keeping the number of data points the same. The number of neurons required was found to decrease from 10 to 8 and overall accuracies improve with the maximum recorded as follows: 0.0018% (for f), 0.0283% (for $\partial f/\partial x$), 0.0170% (for $\partial f/\partial y$), 0.0276% (for $\partial f/\partial z$), 0.8673% (for $\partial^2 f/\partial x^2$), 0.4910% (for $\partial^2 f/\partial y^2$) and 0.6490% (for $\partial^2 f/\partial z^2$). The remaining derivatives are expected to be zero and the calculated values are $\partial^2 f/\partial x\partial y = 0.56\mathrm{E}-4$, $\partial^2 f/\partial x\partial z = 0.46\mathrm{E}-4$ and $\partial^2 f/\partial y\partial z = 0.18\mathrm{E}-4$. Table 5 records the weight results for the refined mesh.

6. Applications

In this section we present examples to illustrate the application of the NN approximation of a function and its derivatives in a solution scheme for viscoelastic fluid flows. We first summarise the context in which the present application is introduced. The problem of interest here is the steady, isothermal flow of polymeric liquid. The problem was solved previously by various techniques such as FDM, FEM and BEM (see, e.g., [25]). Recently, FVM become popular in viscoelastic flow analyses [26–32]. In all of the above methods it is necessary to discretise the whole domain into "finite elements". Volume discretisation is not desirable due to associated inconveniences, especially in three dimensions and/or problems with moving boundaries. Our aim here is to devise a solution scheme in which volume finite element discretisation is eliminated. In the BEM of Tran-Cong and Phan-Thien [33], although a decoupling approach allows the simultaneous system equations to be based only on the boundary discretisation (and hence the name boundary element method), the domain discretisation is still needed for the purpose of computing the non-linear forcing terms for the next iteration. Recently, the requirement for the domain discretisation in the calculation of the volume integrals has been eliminated [34]. It remains to eliminate the need to use volume discretisation to calculate the non-linear extra-stresses, which will be illustrated in the remaining discussion. For the present purpose, it suffices to consider only the part on pseudo-time marching of viscoelastic constitutive equations in the forementioned BEM [33]. For example, using the Phan-Thien–Tanner [35–37] (PTT) model fluid in dimensionless form, we obtain

$$\mathbf{f} \equiv \frac{\partial \boldsymbol{\tau}}{\partial t} = \left[(2\eta\mathbf{D} - g\boldsymbol{\tau}) - \mathrm{Wi}(\mathbf{u}\cdot\nabla\boldsymbol{\tau} - \mathbf{L}\boldsymbol{\tau} - \boldsymbol{\tau}\mathbf{L}^{\mathrm{T}})\right], \tag{12}$$

where $\boldsymbol{\tau}$ is the extra-stress tensor; η is the fluid viscosity; $\mathbf{D}$ is the rate of strain tensor; $\mathbf{u}$ is the velocity; $\mathbf{L}$ is the velocity gradient; g is a parameter; and Wi is the dimensionless Weissenberg number (specific schemes of non-dimensionalisation will be given for following specific examples).

In this approach, the kinematics has been obtained in the previous iteration and held constant while the extra stress field is "marched" from pseudo-time step n to $n+1$ with a suitably chosen pseudo-time step Δt ($\Delta t = 0.01\lambda$ was chosen) in first order Euler scheme according to [33]:

$$\boldsymbol{\tau}^{n+1} \leftarrow \boldsymbol{\tau}^{n} + \mathbf{f}\Delta t, \tag{13}$$

until convergence or otherwise. Results for planar Couette and Poisseuille flows (see, e.g., [25]) are reported in the next two sections, using PTT model fluid with $\epsilon = 0$, where ϵ is a dimensionless parameter embedded in the parameter g mentioned above.

6.1. Couette flow

Referring to Fig. 8, let the typical length be the gap h between the two parallel plates; a typical shear rate be U/h, where U is the velocity of the top plate; $\eta_0 U/h$ is a typical stress where η_0 is the constant zero shear-rate viscosity of the fluid. Variables are non-dimensionalised as follows:

$$\mathbf{x} = h\mathbf{x}', \quad \mathbf{v} = U\mathbf{v}', \quad \boldsymbol{\tau} = \eta_0 \frac{U}{h}\boldsymbol{\tau}', \quad \eta = \eta_0 \eta'.$$

In dimensionless form, dropping the primes, the velocity and extra stress field are given by

$$\mathbf{v} = \begin{bmatrix} z \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} 2\mathrm{Wi}\eta & 0 & \eta \\ 0 & 0 & 0 \\ \eta & 0 & 0 \end{bmatrix}.$$

In these formulas $\mathrm{Wi} = \lambda U/h$ is the Weissenberg number and λ is the relaxation time of the fluid.

To illustrate the use of FFNN in the solution of extra stresses by pseudo-time marching we simply assume the known velocity field for Couette flow and time march the extra stresses at a given Wi number according to Eq. (13). (Note that we do not repeat the total BEM-procedure [33] reported earlier here. We simply replace the stress calculation subprocedure with the present one based on FFNN. Nevertheless, the non-dimensionalised domain of analysis is a unit square in the xz-plane, where x is the flow direction. For this problem each side of the square is adequately represented by a single 2-node linear element.) Once the extra stresses converge for a given Wi number and the kinematics is obtained by BEM, we increment the Wi number and continue time marching with the initial stress field corresponding to the previous Wi number. We use only a single domain for NN approximation with $T_1 = 121$, $T_2 = 81$ and $M = 7$ neurons. The T_1-points are distributed uniformly as follows: $(x,z) = (0.1n, 0.1m)$, $n,m = 0,1,\ldots,10$. The T_2-points are also distributed uniformly and interleaved with T_1-points as follows: $(x,z) = (0.05 + 0.1n, 0.05 + 0.1m)$, $n,m = 1,2,\ldots 9$. Thus the resultant NN is a 2–7–1 architecture for each of the two velocity

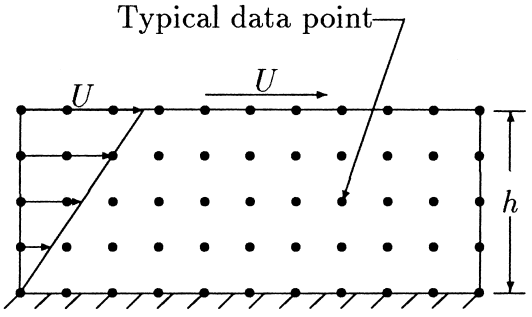

Fig. 8. Couette flow problem: note that the drawing is not to scale and the data point distribution is schematic only.

components and three extra stress components. The whole process is started with the first Wi number equal to 0.1 and increment $\Delta\mathrm{Wi} = 0.1$. We test pseudo-time marching up to $\mathrm{Wi} = 1$. At this Wi number, the maximum errors in the stress components are 3.454% (for τ_{xx}), 1.730% (for τ_{xz}) and the absolute value of τ_{zz} is 1.5E − 4 (analytically, $\tau_{zz} = 0$).

6.2. Poisseuille flow

Poiseuille flow of PTT viscoelastic fluid was given analytically by Sun and Tanner [38]. Referring to Fig. 9, let the typical length, h, be half of the gap between the two parallel plates; a typical shear rate be U/h, where U is the centreline fluid velocity; $\eta_0 U/h$ a typical stress where η_0 the constant zero shear-rate viscosity of the fluid; α the driving pressure gradient. Variables are non-dimensionalised as

$$\mathbf{x} = h\mathbf{x}', \quad \mathbf{v} = U\mathbf{v}', \quad \boldsymbol{\tau} = \eta_0 \frac{U}{h}\boldsymbol{\tau}'; \quad \alpha \equiv \frac{\partial P}{\partial x} = \eta_0 \frac{U}{h^2}\alpha'; \quad \eta = \eta_0 \eta'.$$

In dimensionless form, dropping the primes, the velocity and extra stress field are given by

$$\mathbf{v} = \begin{bmatrix} -\frac{\alpha}{2}(1 - z^2) \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} 2\alpha^2 \eta \mathrm{Wi} z^2 & 0 & \alpha\eta z \\ 0 & 0 & 0 \\ \alpha\eta z & 0 & 0 \end{bmatrix},$$

where $\mathrm{Wi} = \lambda U/h$ is the Weissenberg number and λ is the relaxation time of the fluid.

In this problem we also use a single domain and the same number of data points as in the case of Couette flow. (Again the same comment as in the case of Couette flow regarding the BE procedure applies here. The non-dimensionalised domain of analysis is a unit square in the xz-plane, where x is the flow direction. For this problem each side of the square is adequately represented by a single 3-node quadratic element.) The resultant architecture is 2–8–1 with one more neuron than in the case of Couette flow. This is due to the fact that the velocity and stress fields of Poiseuille flow are varying faster that those of Couette flow. Time marching is continued until $\mathrm{Wi} = 0.8$. At this Wi number, the maximum errors in the stress components are 3.875% (for τ_{xx}), 1.780% (for τ_{xz}) and the absolute value of τ_{zz} is 2.6E − 4 (analytically, $\tau_{zz} = 0$).

7. Concluding remarks

We have successfully implemented an FFNN approximation of functions and their derivatives. The method yields very good accuracies for all functions of one, two and three variables used in this paper, which are non-trivial. We have shown how the method can be incorporated into a non-linear BEM for the analysis of viscoelastic flows, which results in the complete elimination of

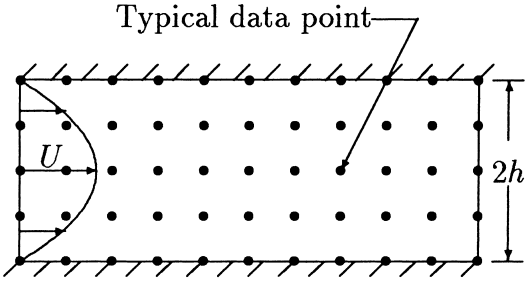

Fig. 9. Poiseuille flow problem: note that the drawing is not to scale and the data point distribution is schematic only.

volume elements. Our initial results here indicate that subregioning is effective in improving accuracy. However the question of adaptive subregioning and solution sensitivity to the number and distribution of the interior points is beyond the scope of this paper. Thus we hope to report in future our on-going works in the following areas:

- *Adaptive subregioning*. Note that subregioning in this context does not make modelling prone to error or more difficult because there is no strict connectivity between regions. In fact, defining data for subregions can even overlap without causing any problem.
- *Parallelisation*. The above subregioning approach is very suitable for parallel implementation on coarse-grained parallel distributed system such as parallel virtual machine (PVM) [39], which is very accessible, affordable and simple to program.
- *Boundary element-neural network* (*BENN*). The performance, including the question of solution sensitivity to the number and distribution of the interior points, of the resultant BENN in solving more complex and realistic problems of viscoelastic flows is being studied.

Finally, we would like to briefly compare the general approach of our method (BENN) with the dual reciprocity method (DRM) [40]. A common feature between the two approaches is that both the present BENN and the DRM do not need volume cells. The other common feature is that both methods are iterative when the problem is non-linear. In the BENN we use the particular solution technique to estimate the non-linear forcing terms as a post-processing operation and thus the coefficients of the discrete system equations depend on geometry only. On the other hand, in the DRM these forcing terms are transformed if it is possible, in such a way that they can be expressed in terms of the primary variables of the formulation (e.g. velocity or displacement). Thus the BENN approaches results in algebraic systems of the form

$$\mathbf{Hu} = \mathbf{Gt} + \mathbf{b}, \tag{14}$$

where $\mathbf{b}$ is the non-linear forcing terms estimated in a post-processing operation; $\mathbf{H}, \mathbf{G}$ are the coefficient matrices; and $\mathbf{u}, \mathbf{t}$ are the primary nodal variables. The DRM, on the other hand, results in algebraic equations of the form

$$(\mathbf{H} + \mathbf{R})\mathbf{u} = \mathbf{Gt}, \tag{15}$$

where the coefficient matrix $\mathbf{R}$ depends on the value of the variable $\mathbf{u}$ obtained in the previous iteration.

For the problems of viscoelastic fluid flow of interest in the present work, the key step of obtaining the relation

$$\mathbf{b} = -\mathbf{Ru} \tag{16}$$

appears to be impossible as far as we can see. This is because $\mathbf{b}$ involves the viscoelastic extra stress which is governed by a constitutive equation in the form of a set of highly non-linear PDEs. For example, the PTT model shown in Eq. (12), which is typical of viscoelastic constitutive equations, involves the non-linear coupling of the velocity field (which is one of the primary variables in the IE formulation), the velocity gradients, the extra stress field and the extra stress gradients and of course other flow dependent material properties. Thus Eq. (12) is solved separately before $\mathbf{b}$ can be estimated in the formulation (14) and the key step (16) is unable to be performed at this stage.

Acknowledgements

This work is supported by a University of Southern Queensland Special Grant. T. Nguyen-Thien is supported by a USQ Scholarship. This support is gratefully acknowledged. We would like to thank the referees for their helpful comments towards the improvement of the paper.

References

[1] G.D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, Claredon Press, Oxford, 1978.
[2] T.J.R. Hughes, The Finite Element Method, Prentice-Hall, Englewood Cliffs, NJ, 1987.
[3] O.C. Zienkiewicz, R.L. Taylor, The Finite Element Method, 4th ed., McGraw-Hill, London, 1991.
[4] S.V. Patankar, Numerical Heat Transfer and Fluid Flow, McGraw-Hill, New York, 1980.
[5] C.A. Brebbia, J.C.F. Telles, L.C. Wrobel, Boundary Element Techniques, Spinger, Berlin, 1984.
[6] J.A. Anderson, E. Rosenfeld (Eds.), Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA, 1988.
[7] C. Lau (Ed.), Neural Networks: Theoretical Foundations and Analysis, IEEE Press, New York, 1992.
[8] B. Widrow, M.A. Lehr, 30 years of adaptive neural networks: perceptron, madaline and backpropagation, Proc. IEEE 78 (9) (1990) 1415–1442.
[9] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice-Hall, NJ, 1994.
[10] A. Cichocki, R. Unbehauen, Neural Networks for Optimization and Signal Processing, Wiley, Chichester, 1993.
[11] K. Hornik, M. Stinchcombe, H. White, Feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[12] K.I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183–192.
[13] Y. Takahashi, Generalization and approximation capabilities of multilayer networks, Neural Comput. 5 (1993) 132–139.
[14] Y. Ito, Approximate capability of layered neural networks with sigmoid units on two layers, Neural Comput. 6 (1994) 1233–1243.
[15] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown function and its derivatives using multilayer feedforward networks, Neural Networks 3 (1990) 551–560.
[16] K. Hornik, M. Stinchcombe, H. White, P. Auer, Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives, Neural Comput. 6 (1994) 1262–1275.
[17] D. Nguyen, B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, Int. Joint Conf. on Neural Network 3 (1990) 21–26.
[18] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing, vol. 1, MIT Press, Cambridge, MA, 1986, pp. 318–362.
[19] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[20] K.A. Levenberg, Method for the solution of certain non-linear problems in least squares, Quart. Appl. Math. 2 (1944) 164–168.
[21] D.W. Marquardt, An algorithm for least squares estimation of non-linear parameters, J. Soc. Indust. Appl. Math. 11 (2) (1963) 431–441.
[22] J.J. Moré, The Levenberg–Marquardt algorithm: implementation and theory, in: G.A. Watson (Ed.), Numerical Analysis, Springer, Berlin, 1977, pp. 105–116.
[23] T. Poston, C.-N., Lee, Y. Choie, Y. Kwon, Local Minima and BP, in: Proc. IJCNN'91, vol. II, Seattle, WA, 1991, pp. 173–176.
[24] L. Chen, A global optimization algorithm for neural network training, in: Proc. Int. Joint Conf. on Neural Networks, vol. 1, IEEE, New York, 1993, pp. 443–446.
[25] R.I. Tanner, Engineering Rheology, Claredon Press, Oxford, 1985.
[26] B. Gervang, P.S. Larsen, Secondary flows in straight ducts of rectangular cross section, J. Non-Newt. Fluid Mech. 39 (1991) 217–237.
[27] Y. Na, J.Y. Yoo, A finite volume technique to simulate the flow of a viscoelastic fluid, Comput. Mech. 8 (1991) 43–55.
[28] H. Jin, N. Phan-Thien, R.I. Tanner, An explicit finite volume method for viscoelastic fluid flows, Comput. Mech. 13 (1994) 443–457.
[29] G.P. Sasmal, A finite volume approach for calculation of viscoelastic flow through an abrupt axisymmetric contraction, J. Non-Newt. Fluid Mech. 56 (1995) 15–47.
[30] S.-C. Xue, N. Phan-Thien, R.I. Tanner, Numerical study of secondary flows of viscoelastic fluid in straight pipes by an implicit finite volume method, J. Non-Newt. Fluid Mech. 59 (1995) 191–213.
[31] H. Huang, N. Phan-Thien, R.I. Tanner, Viscoelastic flow between eccentric rotating cylinders: unstructured control volume method, J. Non-Newt. Fluid Mech. 64 (1996) 71–92.

[32] X.-L. Luo, A control volume approach for integrral viscoelastic models and its application to contraction flow of polymer melts, J. Non-Newt. Fluid Mech. 64 (1996) 173–189.
[33] T. Tran-Cong, N. Phan-Thien, Three dimensional study of extrusion processes by BEM. part 2: extrusion of viscoelastic fluid, Rheologica Acta 27 (1988) 639–648.
[34] T. Nguyen-Thien, T. Tran-Cong, N. Phan-Thien, An improved boundary element method for analysis of polymer profile extrusion, Eng. Anal. Boundary Elements 20 (1997) 81–89.
[35] N. Phan-Thien, R.I. Tanner, A new constitutive equation derived from network theory, J. Non-Newt. Fluid Mech. 2 (1977) 353–365.
[36] N. Phan-Thien, A non-linear network viscoelastic model, J. Rheology 22 (1978) 259–283.
[37] N. Phan-Thien, Squeezing a viscoelastic liquid from a wedge: an exact solution, J. Non-Newt. Fluid Mech. 16 (1984) 329–345.
[38] J. Sun, R.I. Tanner, Computation of steady flow past a sphere in a tube using a Phan-Thien–Tanner integral model, J. Non-Newt. Fluid Mech. 54 (1994) 379–405.
[39] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press, Cambridge, MA, 1994.
[40] P.W. Partridge, C.A. Brebbia, L.C. Wrobel, The Dual Reciprocity Boundary Element Method, Computational Mechanics Publications and Elsevier Applied Science, Southampton and London, 1992.