Artificial Neural Networks for the Solution of Inverse Problems

P. Dadvand, R. Lopez and E. Oñate

International Center for Numerical Methods in Engineering (CIMNE)
Edificio C1, Campus Norte UPC. Gran Capitan s/n. 08034 Barcelona, Spain.
E-mail: pooyan@cimne.upc.edu, rlopez@cimne.upc.edu, onate@cimne.upc.edu
Web page: www.cimne.upc.edu

Kew Words: Inverse Problems, Functional Analysis, Variational Calculus, Artificial Neural Networks, Multilayer Perceptron, Partial Differential Equations, Finite Element Method.

Abstract

In this work we present a variational formulation for neural networks. Within this formulation, the learning problem for the multilayer perceptron lies in terms of finding a function which is an extremal for some functional. As we will see, a variational formulation for neural networks provides a direct method for the solution of general variational problems, in any dimension and up to any degree of accuracy. In order to validate this technique for the solution of inverse problems we train multilayer perceptron networks to solve an input estimation problem and a properties estimation problem.

1 Introduction

Inverse problems can be described as being opposed to direct problems. In a direct problem the cause is given, and the effect is determined. In an inverse problem the effect is given, and the cause is estimated [1]. There are two main types of inverse problems: input estimation problems, in which the system properties and output are known and the input is to be estimated; and properties estimation problems, in which the the system input and output are known and the properties are to be estimated [1]. Inverse problems are found in many areas of science and engineering.

Mathematically, inverse problems fall into the more general class of variational problems [2]. The aim of a variational problem is to find a function which is the minimal or the maximal value of a specified functional. By a functional, we mean a correspondence which assigns a number to each function belonging to some class. Also, inverse problems might be ill-posed [2]. In an ill-posed problem the solution might not meet existence, uniqueness or stability.

While some simple inverse problems can be solved analytically, the only practical technique for general problems is to approximate the solution using direct methods [2]. The fundamental idea underlying the so called direct methods is to consider the variational problem as a limit problem for some function optimization problem in many dimensions. Unfortunately, due to both their variational and ill-posed nature, inverse problems are difficult to solve. Hence, new numerical methods need to be developed in order to overcome that troubles.

Neural networks is one of the main fields of artificial intelligence. The multilayer perceptron is an important model of neural network, and much of the literature in the field is referred to that model. Traditionally, the learning problem for the multilayer perceptron has been formulated in terms of the minimization of an error function of the free parameters in the network, in order to fit the neural network to an input-target data set [3]. In that way, the only learning tasks allowed for the multilayer perceptron are data modelling type problems, such as function regression or pattern recognition.

In this work we present a variational formulation for neural networks. Within this formulation, the learning problem for the multilayer perceptron lies in terms of solving a variational problem by minimizing a performance functional of the function space spanned by the network. The choice of a suitable performance functional depends on the variational problem at hand. In order to evaluate the performance functional we might need to integrate a function, an ordinary differential equation or a partial differential equation.

As we will see, neural networks are not only able to solve data modelling problems, but also a wide range of mathematical and physical problems. More specifically, a variational formulation for neural networks provides a direct method for solving general variational problems, and therefore inverse problems. Also, this numerical method allows the use of any regularization technique in order to obtain well-posed solutions. By way of illustration, we train a multilayer perceptron to solve an input estimation problem and a properties estimation problem.

2 A Variational Formulation for the Multilayer Perceptron

The multilayer perceptron can be seen as a mathematical model of the performance in a biological neural network. Here we present a theory of the multilayer perceptron from the perspective of functional analysis and variational calculus. Within this theory, a multilayer perceptron is described by four concepts: a neuron model, the perceptron, a network architecture, the feed-forward, and associated performance functionals and training algorithms.

2.1 The Perceptron Neuron Model

A neuron model is the basic information processing unit in a neural network. The perceptron is the characteristic neuron model in the multilayer perceptron [4]. It computes a net input signal u as a function h of the input signals $\mathbf{x}$ and the free parameters $(b, \mathbf{w})$. The net input signal is then subjected to an activation function g to produce an output signal y [5]. Figure 1 shows a perceptron with n inputs.

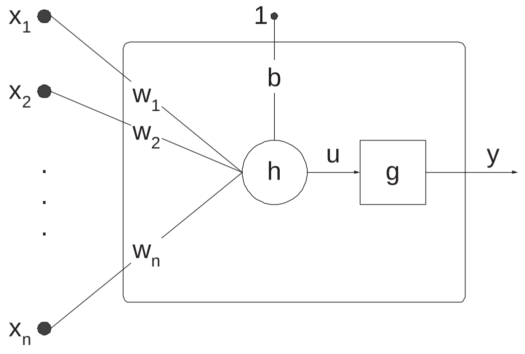

Figure 1: A perceptron with n inputs.

Mathematically, a perceptron neuron model spans a parameterized function space V from an input $X \subseteq \mathbf{R}^n$ to an output $Y \subseteq \mathbf{R}$ [6]. Elements of V are parameterized by the free parameters of the neuron $(b, \mathbf{w})$. The dimension of the function space V is therefore $n+1$. The elements of the function space spanned by a perceptron are of the form

$$\begin{aligned} y : \mathbf{R}^n &\to \mathbf{R} \\ \mathbf{x} &\mapsto y(\mathbf{x}; b, \mathbf{w}), \end{aligned}$$

where

$$y(\mathbf{x}; b, \mathbf{w}) = g\left(b + \sum_{i=1}^{n} w_i x_i\right). \qquad (1)$$

Two of the most used activation functions are the sigmoid function, $g(u) = tanh(u)$, and the linear function, $g(u) = u$ [7].

2.2 The Feed-forward Network Architecture

Neurons can be combined to form a neural network. The architecture of a neural network refers to the number of neurons, their arrangement and connectivity [4]. The characteristic network architecture in the multilayer perceptron is the so called feed-forward architecture. A feed-forward architecture typically consists on an input layer of sensorial nodes, one or more hidden layers of neurons, and an output layer of neurons. Communication proceeds layer by layer from the input layer via the hidden layers up to the output layer [4]. In this way, a multilayer perceptron is a feed-forward network architecture of perceptron neuron models. Figure 2 shows a multilayer perceptron with n inputs, one hidden layer with h_1 neurons and m neurons in the output layer.

In a similar way as it happens with a single perceptron, a multilayer perceptron spans a parameterized function space V from an input $X \subseteq \mathbf{R}^n$ to an output $Y \subseteq \mathbf{R}^m$ [6]. Elements of V are parameterized by the free parameters in the network, which can be grouped together in a s-dimensional free parameter vector $\underline{\alpha} = (\alpha_1, ..., \alpha_s)$. The dimension of the function space V is therefore s. The elements of the function space spanned by the multilayer perceptron in Figure 2 are of the form

$$\begin{aligned} \mathbf{y} : \mathbf{R}^n &\to \mathbf{R}^m \\ \mathbf{x} &\mapsto \mathbf{y}(\mathbf{x}; \underline{\alpha}), \end{aligned}$$

where

$$y_k(\mathbf{x}; \underline{\alpha}) = g^{(2)}\left(b_k^{(2)} + \sum_{j=1}^{h_1} w_{kj}^{(2)} \cdot g^{(1)}\left(b_j^{(1)} + \sum_{i=1}^{n} w_{ji}^{(1)} x_i\right)\right), \quad k = 1, ..., m. \qquad (2)$$

A multilayer perceptron with as few as one hidden layer of sigmoid neurons and an output layer of linear neurons provides a general framework for approximating any function from one finite dimensional space to another up to any desired degree of accuracy, provided sufficiently many hidden neurons are available. In this sense, multilayer perceptron networks are a class of universal approximators [8].

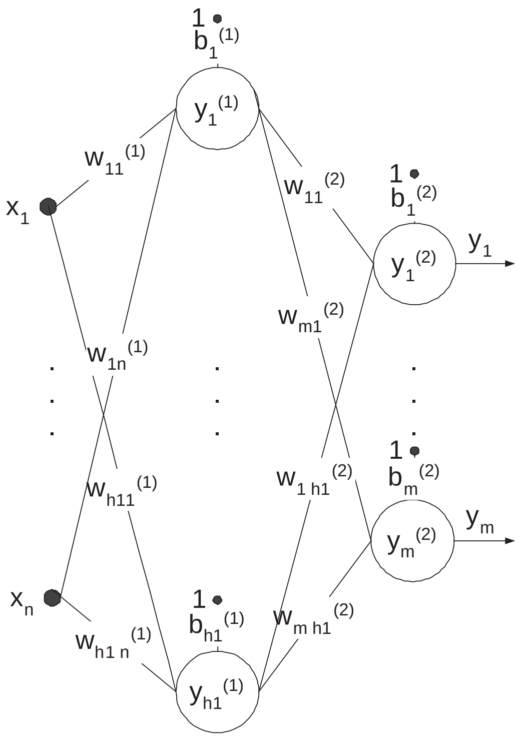

Figure 2: A two-layer perceptron with n inputs, one hidden layer with h_1 neurons and m neurons in the output layer.

2.3 The Performance Functional

Traditionally, the learning problem for the multilayer perceptron has been formulated in terms of the minimization of an error function of the free parameters in the network, in order to fit the neural network to an input-target data set [3]. In that way, the only learning tasks allowed for the multilayer perceptron are data modelling type problems.

In a variational formulation for the multilayer perceptron, the concept of error function, $e(\underline{\alpha})$, is changed by the concept or performance functional, $F[\mathbf{y}(\mathbf{x};\underline{\alpha})]$ [6]. A performance functional for the multilayer perceptron is of the form

$$\begin{array}{rccl} F: & V & \to & \mathbf{R} \\ & \mathbf{y}(\mathbf{x};\underline{\alpha}) & \mapsto & F[\mathbf{y}(\mathbf{x};\underline{\alpha})]. \end{array}$$

The performance functional defines the task that the network is required to accomplish and provides a measure of the quality of the representation that the network is required to learn. In this way, the choice of a suitable performance functional depends on the particular application. As we will see, changing the concept of error function by the concept of performance functional allows us to extend the number of learning tasks for the multilayer perceptron to any variational problem. Some examples are optimal control problems [6], inverse problems or optimal shape design problems. The learning problem for the multilayer perceptron can then be formulated in terms of the minimization of a performance functional of the function space spanned by the neural network [6]:

Problem 1 (Variational problem for the multilayer perceptron) *Let V be the space of all functions $\mathbf{y}(\mathbf{x};\underline{\alpha})$ spanned by a multilayer perceptron, and let s be the dimension of V. Find a function $\mathbf{y}^*(\mathbf{x};\underline{\alpha}^*) \in V$ for which the functional $F[\mathbf{y}(\mathbf{x};\underline{\alpha})]$, defined on V, takes on a minimum or a maximum value.*

The performance functional, $F[\mathbf{y}(\mathbf{x};\underline{\alpha})]$, has a performance function associated, $f(\underline{\alpha})$, which is defined as a function of the free parameters in the network [6]. A performance function for the multilayer perceptron is of the form

$$\begin{array}{rcl} f : \mathbf{R}^s & \rightarrow & \mathbf{R} \\ \underline{\alpha} & \mapsto & f(\underline{\alpha}). \end{array}$$

The minimum or maximum value of the performance functional is achieved for a vector of free parameters at which the performance function takes on a minimum or maximum value, respectively. Therefore, the learning problem for the multilayer perceptron, formulated as a variational problem, can be reduced to a function optimization problem [6]:

Problem 2 (Reduced function optimization problem for the multilayer perceptron) *Let $\mathbf{R}^s$ be the space of all vectors $\underline{\alpha}$ spanned by the free parameters of a multilayer perceptron. Find a vector $\underline{\alpha}^* \in \mathbf{R}^s$ for which the function $f(\underline{\alpha})$, defined on $\mathbf{R}^s$, takes on a minimum or a maximum value.*

In this sense, a variational formulation for the multilayer perceptron provides a direct method to approximate the solution of general variational problems, in any dimension and up to any desired degree of accuracy [6].

2.4 The Training Algorithm

The training algorithm is entrusted to solve the reduced function optimization problem. There are many different training algorithms for the multilayer perceptron, which have different requirements and characteristics. Moreover, there is no one best suited to all locations. According to the information they require, training algorithms can be classified as:

1. Zero order algorithms, which make use of the performance function only (genetic algorithm, ...).

2. First order algorithms, which make use of the performance function and its gradient vector (conjugate gradient, ...).

3. Second order algorithms, which make use of the performance function, its gradient vector and its Hessian matrix (quasi-Newton, ...).

According to their character, training algorithms can be classified as:

1. Local algorithms (conjugate gradient, quasi-Newton, ...).

2. Global algorithms (genetic algorithm, ...).

3 Application to Inverse Problems

A variational formulation for the multilayer perceptron provides a direct method for the solution of general variational problems, and consequently inverse problems. Here we train a multilayer perceptron to solve an input estimation problem and a properties estimation problem. Both problems are solved with the Flood library [9]. The partial differential equations are integrated in the Kratos library [10].

3.1 The Boundary Temperature Estimation Problem

For the boundary temperature estimation problem, consider the heat equation in the square $\Omega = \{(x, y) : |x| \leq 0.5, |y| \leq 0.5\}$ with boundary $\Gamma = \{(x, y) : |x| = 0.5, |y| = 0.5\}$,

$$\nabla^2 u(x, y; t) = \frac{\partial u(x, y; t)}{\partial t} \quad in \quad \Omega, \tag{3}$$

for $t \in [0, 1]$, with initial condition $u(x, y; 0) = 0$ in Ω. The problem is to estimate the boundary temperature $y(t)$ on Γ and for $t \in [0, 1]$, from measurements of the temperature at the center of the square $u(0, 0; t)$ for $t \in [0, 1]$,

$$\begin{array}{cc} t_1 & u_1(0,0;t_1) \\ t_2 & u_2(0,0;t_2) \\ \vdots & \vdots \\ t_P & u_P(0,0;t_P) \end{array}$$

where P is the number of time steps considered. For this problem we use 101 time steps.

The first stage in solving this problem is to choose a network architecture to represent the boundary temperature $y(t)$ for $t \in [0, 1]$. Here a multilayer perceptron with a sigmoid hidden layer and a linear output layer is used. This neural network is a class of universal approximator [8]. The network must have one input and one output neuron. We guess a good number of neurons in the hidden layer to be six. This neural network spans a family V of parameterized functions $y(t; \underline{\alpha})$ of dimension $s = 19$, which is the number of free parameters in the network.

The second stage is to derive a performance functional for the boundary temperature estimation problem. This is to be the mean squared error between the computed temperature at the center of the square for a given boundary temperature and the measured temperature at the center of the square,

$$E[y(t; \underline{\alpha})] \quad = \quad \frac{1}{P} \sum_{i=1}^{P} \left(\hat{u}_{y(t;\underline{\alpha})}(0, 0; t_i) - u_i(0, 0; t_i) \right)^2 . \tag{4}$$

Please note that evaluation of the performance functional (4) requires a numerical method for integration of partial differential equations. Here we choose the Finite Element Method [11]. For this problem we use a triangular mesh with 888 elements and 485 nodes.

The boundary temperature estimation problem for the multilayer perceptron can then be formulated as follows:

Problem 3 (Boundary temperature estimation problem for the multilayer perceptron) *Let V be the space consisting of all functions $y(t; \underline{\alpha})$ spanned by a multilayer perceptron with* 1 *input,* 6 *sigmoid neurons in the hidden layer and* 1 *linear output neuron. The dimension of V is* 19*. Find a vector of free parameters $\underline{\alpha}^* \in \mathbf{R}^{19}$ that addresses a function $y^*(t; \underline{\alpha}^*) \in V$ for which the functional (4), defined on V, takes on a minimum value.*

The third stage in solving this problem is to choose a suitable training algorithm. Here we use a conjugate gradient with Polak-Ribiere train direction and Brent optimal train rate [3]. The tolerance in the Brent's method is set to 10^{-6}. Training of the neural network with the conjugate gradient algorithm requires the evaluation of the performance function gradient vector $\nabla f(\underline{\alpha})$. This is carried out by means of numerical differentiation [3]. In particular, we use the symmetrical central differences method [3] with an ϵ value of 10^{-6}.

In this example, we set the training algorithm to stop when the norm of the performance function gradient $\nabla f(\underline{\alpha})$ falls below 10^{-6}. That means that the necessary condition for a local minimum has about been satisfied. The neural network is initialized with a vector of free parameters chosen at random in the interval $[-1, 1]$. During the training process the performance function decreases until the stopping criterium is satisfied. Table 1 shows the training results for this problem. Here N denotes the number of epochs, M the number of performance evaluations, $F[y^*(t; \underline{\alpha}^*)]$ the final performance, and $\nabla f(\underline{\alpha}^*)$ the final performance function gradient norm.

N	M	$F[y^*(t; \underline{\alpha}^*)]$	$\nabla f(\underline{\alpha}^*)$
421	25352	$2.456 \cdot 10^{-4}$	$8.251 \cdot 10^{-7}$

Table 1: Training results for the boundary temperature estimation problem.

Figure 3 shows the actual boundary temperature, the boundary temperature estimated by the neural network, and the measured temperature at the center of the square for this problem. The solution here is good, since the estimated boundary temperature matches the actual boundary temperature.

3.2 The Diffusion Coefficient Estimation Problem

For the diffusion coefficient estimation problem, consider the equation of diffusion in an inhomogeneous medium in the square $\Omega = \{(x, y) : |x| \leq 0.5, |y| \leq 0.5\}$ with boundary $\Gamma = \{(x, y) : |x| = 0.5, |y| = 0.5\}$,

$$\nabla \left(\kappa(x, y) \nabla u(x, y; t) \right) = \frac{\partial u(x, y; t)}{\partial t} \quad in \quad \Omega, \tag{5}$$

for $t \in [0, 1]$ and where $\kappa(x, y)$ is called the diffusion coefficient, with boundary condition $u(x, y; t) = 0$ on Γ and for $t \in [0, 1]$, and initial condition $u(x, y; 0) = 1$ in Ω. The problem is to estimate the diffusion coefficient $\kappa(x, y)$ in Ω, from measurements of the temperature at different points on the square $u(x, y; t)$ in Ω and for $t \in [0, 1]$,

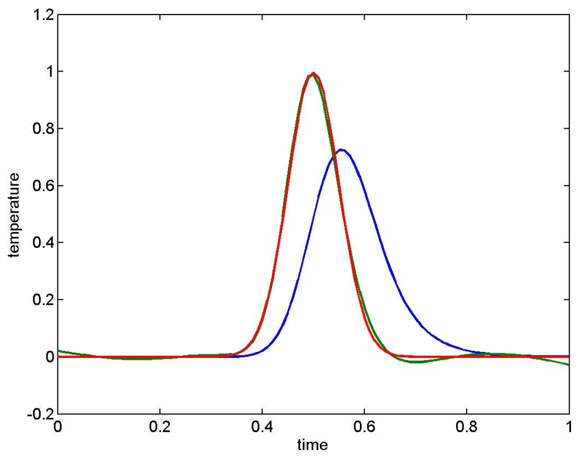

Figure 3: Actual boundary temperature (red), estimated boundary temperature (green) and measured temperature at the center of the square (blue) for the boundary temperature estimation problem.

$$\begin{array}{ccccc}
t_1 & u_{11}(x_1,y_1;t_1) & u_{12}(x_2,y_2;t_1) & \ldots & u_{1Q}(x_Q,y_Q;t_1) \\
t_2 & u_{21}(x_1,y_1;t_2) & u_{22}(x_2,y_2;t_2) & \ldots & u_{2Q}(x_Q,y_Q;t_2) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
t_P & u_{P1}(x_1,y_1;t_P) & u_{P2}(x_2,y_2;t_P) & \ldots & u_{PQ}(x_Q,y_Q;t_P)
\end{array}$$

where P and Q are the number of points and time steps considered, respectively. For this problem we use 485 points and 11 time steps.

The first stage in solving this problem is to choose a network architecture to represent the diffusion coefficient $\kappa(x,y)$ in Ω. Here a multilayer perceptron with a sigmoid hidden layer and a linear output layer is used. This neural network is a class of universal approximator [8]. The neural network must have two inputs and one output neuron. We guess a good number of neurons in the hidden layer to be six. This neural network is denoted as a $2:6:1$ multilayer perceptron. It spans a family V of parameterized functions $\kappa(x,y;\underline{\alpha})$ of dimension $s=25$, which is the number of free parameters in the network.

The second stage is to derive a performance functional for the diffusion coefficient estimation problem. The performance functional for this problem is to be the mean squared error between the computed temperature for a given diffusion coefficient and the measured temperature,

$$E[\kappa(x,y;\underline{\alpha})] = \frac{1}{PQ}\sum_{i=1}^{P}\left(\sum_{j=1}^{Q}\left(\hat{u}_{\kappa(x,y;\underline{\alpha})}(x_j,y_j;t_i)-u_{ij}(x_j,y_j;t_i)\right)^2\right), \tag{6}$$

The diffusion coefficient estimation problem for the multilayer perceptron can then be formulated as follows:

Problem 4 (Diffusion coefficient estimation problem for the multilayer perceptron) *Let V be the space consisting of all functions $\kappa(x, y; \underline{\alpha})$ spanned by a multilayer perceptron with 2 inputs, 6 sigmoid neurons in the hidden layer and 1 linear output neuron. The dimension of V is 25. Find a vector of free parameters $\underline{\alpha}^* \in \mathbf{R}^{25}$ that addresses a function $\kappa^*(x, y; \underline{\alpha}^*) \in V$ for which the functional (6), defined on V, takes on a minimum value.*

Evaluation of the performance functional (6) requires a numerical method for integration of partial differential equations. Here we choose the Finite Element Method [11]. For this problem we use a triangular mesh with 888 elements and 485 nodes.

The third stage is to choose a suitable algorithm for training. Here we use a conjugate gradient with Polak-Ribiere train direction and Brent optimal train rate methods for training [3]. The tolerance in the Brent's method is set to 10^{-6}. Training of the neural network with the conjugate gradient algorithm requires the evaluation of the performance function gradient vector $\nabla f(\underline{\alpha})$ [3]. This is carried out by means of numerical differentiation. In particular, we use the symmetrical central differences method [3] with $\epsilon = 10^{-6}$.

In this example, we set the training algorithm to stop when the norm of the performance function gradient falls below 10^{-6}. That means that the necessary condition for a local minimum has about been satisfied. The neural network is initialized with a vector of free parameters chosen at random in the interval $[-1, 1]$. Table 1 shows the training results for this problem. Here N denotes the number of epochs, M the number of performance evaluations, $F[\kappa^*(x, y; \underline{\alpha}^*)]$ the final performance, and $\nabla f(\underline{\alpha}^*)$ the final performance function gradient norm.

N	M	$F[\kappa^*(x, y; \underline{\alpha}^*)]$	$\nabla f(\underline{\alpha}^*)$
312	22652	$1.562 \cdot 10^{-5}$	$7.156 \cdot 10^{-7}$

Table 2: Training results for the diffusion coefficient estimation problem.

Figure 3 shows the actual diffusion coefficient and the diffusion coefficient estimated by the neural network for this problem. The solution here is good, since the estimated diffusion coefficient matches the actual diffusion coefficient.

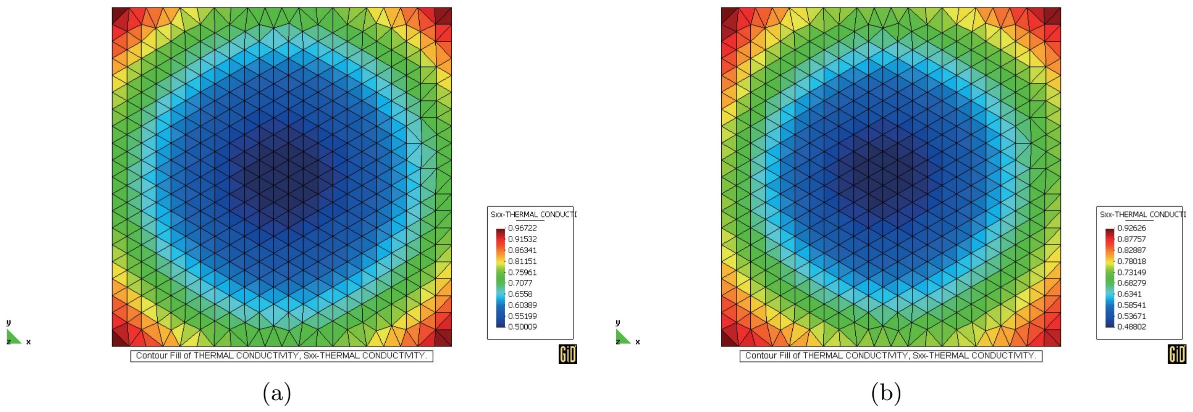

Figure 4: Actual diffusion coefficient (a) and estimated diffusion coefficient (b) for the diffusion coefficient estimation problem.

4 Conclusions and Future Work

A variational formulation for neural networks provides a direct method for the solution of general variational problems. The solution of the boundary temperature estimation problem and the diffusion coefficient estimation problem demonstrates the capability of the method in dealing with different types of inverse problems and in many dimensions. Ongoing work focuses on the solution of inverse problems of real engineering interest.

References

[1] A. Kirsch. *An Introduction to the Mathematical Theory of Inverse Problems*, Springer, 1996.

[2] M. Tanaka (Editor). *Inverse Problems in Engineering Mechanics IV*, Elsevier, 2003.

[3] C. Bishop. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[4] J. Šíma and P. Orponen. *General-Purpose Computation with Neural Networks: A Survey of Complexity Theoretic Results*, Neural Computation 15 (2727-2778), 2003.

[5] S. Haykin. *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1994.

[6] R. Lopez, E. Balsa-Canto and E. Oñate, *Artificial Neural Networks for the Solution of Optimal Control Problems*, Proceedings of the Sixth Conference on Evolutionary and Deterministic Methods for Design, Optimisation and Control with Applications to Industrial and Societal Problems (65-66), 2005.

[7] H. Demuth and M. Beale. *Neural Network Toolbox for Use with MATLAB. User's Gide*, The MathWorks, 2002.

[8] K. Hornik, M. Stinchcombe and H. White. *Multilayer feedforward networks are universal approximators*, Neural Networks 2-5 (359-366), 1989.

[9] R. Lopez. *Flood: An Open Source Neural Networks C++ Library*, www.cimne.com/flood, 2005.

[10] *Kratos: An Object-Oriented Environment for Development of Multi-Physics Analysis Software*, www.cimne.upc.edu/kratos, 2005.

[11] O. Zienkiewicz and R. Taylor *The Finite Element Method, Volumes 1 and 2*, Mc Graw Hill, 1988.