Solving Nonlinear Differential Equations by a Neural Network Method

Lucie P. Aarts[1] and Peter Van der Veer[1]

[1] Delft University of Technology, Faculty of Civilengineering and Geosciences,
Section of Civilengineering Informatics,
Stevinweg 1, 2628 CN Delft, The Netherlands
l.aarts@citg.tudelft.nl, p.vdveer@ct.tudelft.nl

Abstract. In this paper we demonstrate a neural network method to solve nonlinear differential equations and its boundary conditions. The idea of our method is to incorporate knowledge about the differential equation and its boundary conditions into neural networks and the training sets. Hereby we obtain specifically structured neural networks. To solve the nonlinear differential equation and its boundary conditions we have to train all obtained neural networks simultaneously. This is realized by applying an evolutionary algorithm.

1 Introduction

In this paper we present a neural network method to solve a nonlinear differential equation and its boundary conditions. In [1] we have already demonstrated how we could solve linear differential and linear partial differential equations by our neural network method. In [2] we showed how to use our neural network method to solve systems of coupled first order linear differential equations. In this paper we demonstrate how we incorporate knowledge about the nonlinear differential equation and its boundary conditions into the structure of the neural networks and the training sets. Training the obtained neural networks simultaneously now solves the nonlinear differential equation and its boundary conditions. Since several of the obtained neural networks are specifically structured, the training of the networks is accomplished by applying an evolutionary algorithm. An evolutionary algorithm tries to find the minimum of a given function. Normally one deals with an evolutionary algorithm working on a single population, i.e. a set of elements of the solution space. We however use an evolutionary algorithm working on multiple subpopulations to obtain results more efficiently. At last we graphically illustrate the obtained results of solving the nonlinear differential equation and its boundary conditions by our neural network method.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 181–189, 2001.

2 Problem Statement

Many of the general laws of nature, like in physics, chemistry, biology and astronomy, find their most natural expression in the language of differential equations. Applications also abound in mathematics itself, especially in geometry, and in engineering, economics, and many other fields of applied science.

In [10] one derives the following nonlinear differential equation and its boundary conditions for the description of the problem of finding the shape assumed by a flexible chain suspended between two points and hanging under its own weight. Further the y -axis pass through the lowest point of the chain.

$$\frac{d^2y}{dx^2} = \sqrt{1+\frac{dy}{dx}\cdot\frac{dy}{dx}}\,, \tag{1}$$

$$y(0) = 1, \tag{2}$$

$$\frac{dy}{dx}(0) = 0\,. \tag{3}$$

Here the linear density of the chain is assumed to be a constant value. In [10] the analytical solution is derived for system (1), (2) and (3) and is given by

$$y(x) = \frac{1}{2}\left(e^{x} + e^{-x}\right). \tag{4}$$

In this paper we consider the system (1), (2) and (3) on the interval $x \in [-1,2]$.

3 Outline of the Method

By knowing the analytical solution of (1), (2) and (3), we may assume that $y(x)$ and its first two derivatives are continuous mappings. Further we define the logsigmoid function f as

$$f(x) = \frac{1}{1+\exp(-x)}\,. \tag{5}$$

By results in [8] we can find such real values of α_i, w_i and b_i that for a certain natural number m the following mappings

$$\varphi(x) = \sum_{i=1}^{m} \alpha_i f(w_i x + b_i), \tag{6}$$

$$\frac{d\varphi}{dx}(x) = \sum_{i=1}^{m} \alpha_i w_i \frac{df}{dx}(w_i x + b_i), \tag{7}$$

$$\frac{d^2\varphi}{dx^2}(x) = \sum_{i=1}^{m} \alpha_i w_i^2 \frac{d^2 f}{dx^2}(w_i x + b_i), \tag{8}$$

respectively approximate $y(x), \frac{dy}{dx}$ and $\frac{d^2 y}{dx^2}$ arbitrarily well. The networks represented by (6), (7) and (8) have one hidden layer containing m neurons and a linear output layer. Further we define the DE-neural network of system (1), (2) and (3) as the not fully connected neural network which is constructed as follows. The output of the network represented by (7) is the input of a layer having the function $g(x) = x^2$ as transfer function. The layer contains one neuron and has no bias. The connection weight between the network represented by (7) and the layer is 1. The output of the layer is the input of a layer with the function $h(x) = \sqrt{x}$ as transfer function. The layer contains one neuron and has a bias with value 1. The connection weight between the two layers is 1.

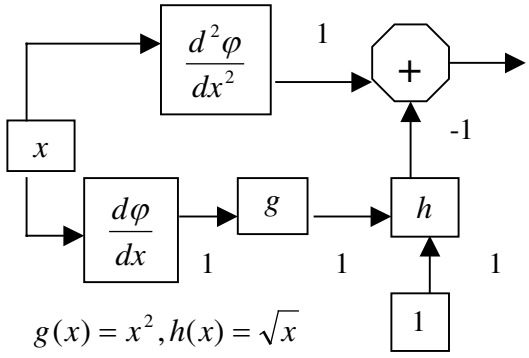

Fig. 1. The DE-neural network for system (1), (2) and (3)

The output of the last layer is subtracted from the output of the network represented by (8). A sketch of the DE-neural network of system (1), (2) and (3) is given in Fig. 1.

Since the learnability of neural networks to simultaneously approximate a given function and its unknown derivatives is made plausible in [5], we observe the following. Assume that we have found such values of the weights that the networks represented by (6), (7) and (8) respectively approximate $y(x)$ and its first two derivatives arbitrarily well on a certain interval. By considering the nonlinear differential equation given by (1) it then follows that the DE-neural network must have a number arbitrarily close to zero as output for any input of the interval. In [6] it is already stated that any network suitably trained to approximate a mapping satisfying some nonlinear partial differential equations will have an output function that itself approximately satisfies the partial differential equations by virtue of its approximation of the mapping's derivatives. Further the network represented by (6) must have for input $x = 0$ an output arbitrarily close to one and the network represented by (7) must give for the same input an output arbitrarily close to zero.

The idea of our neural network method is based on the observation that if we want to fulfil a system like (1), (2) and (3) the DE-neural network should have zero as output for any input of the considered interval $[-1,2]$. Therefore we train the DE-neural network to have zero as output for any input of a training set with inputs $x \in [-1,2]$. Further we have the following restrictions on the values of the weights. The neural network represented by (6) must be trained to have one as output for input $x = 0$ and for the same input the neural network represented by (7) must be trained to have zero as output. If the training of the three networks has well succeeded the mapping φ and its first two derivatives should respectively approximate y and its first two derivatives. Note that we still have to choose the number of neurons in the hidden layers of the networks represented by (6), (7) and (8), i.e. the natural number m by trial and error.

The three neural networks have to be trained simultaneously as a consequence of their inter-relationships. It is a specific point of attention how to adjust the values of the weights of the DE-neural network. The weights of the DE-neural network are highly correlated. In [11] it is stated that an evolutionary algorithm makes it easier to generate neural networks with some special characteristics. Therefore we use an evolutionary algorithm to adjust simultaneously the weights of the three neural networks. Before we describe how we manage this, we give a short outline of what an evolutionary algorithm is.

4 Evolutionary Algorithms with Multiple Subpopulations

Evolutionary algorithms work on a set of elements of the solution space of the function we would like to minimize. The set of elements is called a population and the elements of the set are called individuals. The main idea of evolutionary algorithms is that they explore all regions of the solution space and exploit promising areas through applying recombination, mutation, selection and reinsertion operations to the individu-

als of a population. In this way one hopefully finds the minimum of the given function. Every time all procedures are applied to a population, a new generation is created. Normally one works with a single population. In [9] Pohlheim however states that results are more efficiently obtained when we are working with multiple subpopulations instead of just a single population. Every subpopulation evolves over a few generations isolated (like with a single population evolutionary algorithm) before one or more individuals are exchanged between the subpopulations. To apply an evolutionary algorithm in our case, we define e_1, e_2 and e_3 as the means of the sum-of-squares error on the training sets of respectively the DE-neural network, the network represented by (6) and the network given by (7). Here we mean by the mean of the sum-of-squares error on the training set of a certain network, that the square of the difference between the target and the output of the network is summed for all inputs and that this sum is divided by the number of inputs. To simultaneously train the DE-neural network and the networks represented by φ and $\frac{d\varphi}{dx}$ we minimize the expression

$$e_1 + e_2 + e_3, \tag{9}$$

by using an evolutionary algorithm. Here equation (9) is a function of the variables α_i, w_i and b_i.

5 Results

In this section we show the results of applying our neural network method to the system (1), (2) and (3). Some practical aspects of training neural networks that are well known in literature also hold for our method. In e.g. [3] and [7], it is stated that if we want to approximate an arbitrary mapping with a neural network represented by (6), it is advantageous for the training of the neural networks to scale the inputs and targets so that they fall within a specified range. In this way we can impose fixed limits on the values of the weights. This prevents that we get stuck too far away from a good optimum during the training process. By training the networks with scaled data all weights can remain in small predictable ranges. In [2] more can be found about scaling the variable where the unknown of the differential equation depend on and scaling the function values of the unknown of the differential equation, to improve the training process of the neural networks.

To make sure that in our case the weights of the networks can remain in small predictable ranges, we scale the function values of the unknown of the nonlinear differential equation. Since we normally do not know much about the function values of the unknown we have to guess a good scaling of the function values of the unknown. For solving the system (1), (2) and (3) on the considered interval $[-1,2]$ we decide to scale y in the following way:

$$y_M = \frac{y}{2}. \tag{10}$$

Hereby the system (1), (2) and (3) becomes

$$2\frac{d^2 y_M}{dx^2} = \sqrt{1 + 4\frac{dy_M}{dx} \cdot \frac{dy_M}{dx}}, \tag{11}$$

$$y_M(0) = \frac{1}{2}, \tag{12}$$

$$\frac{dy_M}{dx}(0) = 0. \tag{13}$$

We now solve the system (11), (12) and (13) by applying the neural network method described in Sect. 3. A sketch of the DE-neural network for system (11), (12) and (13) is given in Fig. 2.

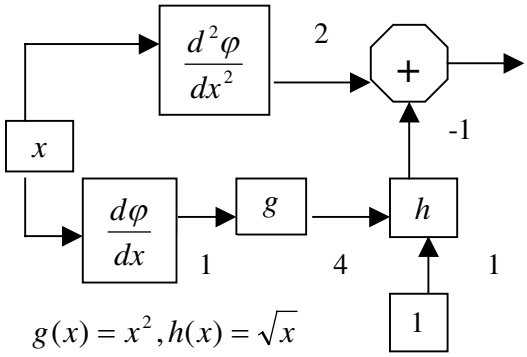

Fig. 2. The DE-neural network for system (11), (12) and (13)

We implemented the neural networks by using the Neural Network Toolbox of Matlab 5.3 ([4]). Further we used the evolutionary algorithm implemented in the GEATbx toolbox ([9]). When we work with the evolutionary algorithms implemented in the GEATbx toolbox, the values of the unknown variables α_i, w_i and b_i have to fall within a specified range. By experiments we noticed that we obtain good results if we restrict the values of the variables α_i, w_i and b_i to the interval $[-5,5]$. The DE-neural network is trained by a training set with inputs $x \in \{-1,-0.9,..,1.9,2\}$ and

the corresponding targets of all inputs are zero. Further we have to train the neural network represented by φ to have one as output for input $x = 0$ and for the same input the neural network represented by $\frac{d\varphi}{dx}$ must have zero as output. The number of neurons in the hidden layer of the neural networks represented by (6), (7) and (8) is taken equal to 6. Therefore the number of variables which have to be adapted is equal to 18. After running the chosen evolutionary algorithm for 1500 generations with 160 individuals divided over 8 subpopulations we take the set $x \in \{-1,-0.95,0.9,..,1.95,2\}$ as input to compute the output of the neural networks represented by $2\varphi, 2\frac{d\varphi}{dx}$ and $2\frac{d^2\varphi}{dx^2}$. We also compute the analytical solution of (1), (2) and (3) and its first two derivatives for $x \in \{-1,-0.95,0.9,..,1.95,2\}$. By comparing the results we conclude that the approximation of y and its first two derivatives by respectively 2φ and its first two derivatives are very good. Both the neural network method solution of (1), (2) and (3) and its first two derivatives as the analytical solution of (1), (2) and (3) and its first two derivatives are graphically illustrated in Fig. 3 and Fig. 4. The errors between the neural network method solution of (1), (2) and (3) and its first two derivatives on the one hand and the analytical solution of (1), (2) and (3) and its first two derivatives on the other hand are graphically illustrated in Fig. 5. The difference between the target of the DE-neural network of the system (1), (2) and (3), i.e. zero for any input x of the set $\{-1,-0.95,0.9,..,1.95,2\}$ and its actual output is also illustrated in Fig. 5. Considering Fig. 5, we can conclude that the approximations of the solution of (1), (2) and (3) and its first derivative are somewhat better than the approximation of the second derivative of the solution of (1), (2) and (3). Since we are however in most numerical solving methods for differential equations interested in the approximation of just the solution itself, our results are really satisfying.

6 Concluding Remarks

In this paper we used our neural network method to solve a system consisting of a nonlinear differential equation and its two boundary conditions. The obtained results are very promising and the concept of the method appears to be feasible. In further research more attention will be paid to practical aspects like the choice of the evolutionary algorithm that is used to train the networks simultaneously. We will also do more extensive experiments on scaling issues in practical situations, especially the scaling of the variable where the unknown of the differential equation depends on.

References

1. Aarts, L.P., Van der Veer, P.: Neural Network Method for Solving Partial Differential Equations. Accepted for publication in Neural Processing Letters (200?)
2. Aarts, L.P., Van der Veer, P.: Solving Systems of First Order Linear Differential Equations by a Neural Network Method. Submitted for publication December 2000
3. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)
4. Demuth, H., Beale, M.: Neural Networks Toolbox For Use with Matlab, User's Guide Version 3. The Math Works, Inc., Natick Ma (1998)
5. Gallant, R.A., White H.: On Learning the Derivatives of an Unknown Mapping With Multilayer Feedforward Networks. Neural Networks 5 (1992) 129-138
6. Hornik, K., Stinchcombe, M., White, H.: Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks. Neural Networks 3 (1990) 551-560
7. Masters, T.: Practical Neural Networks Recipes in C++. Academic Press, Inc. San Diego (1993)
8. Li, X.: Simultaneous approximations of multivariate functions and their derivatives by neural networks with one hidden layer. Neurocomputing 12 (1996), 327-343
9. Pohlheim, H., Documentation for Genetic and Evolutionary Algorithm Toolbox for use with Matlab (GEATbx): version 1.92, more information on http://www.geatbx.com (1999)
10. Simmons, G.F.: Differential equations with applications and historical notes. 2nd ed. McGraw-Hill, Inc., New York (1991)
11. Yao, X.: Evolving Artificial Neural Networks. Proceedings of the IEEE 87(9) (1999) 1423–1447

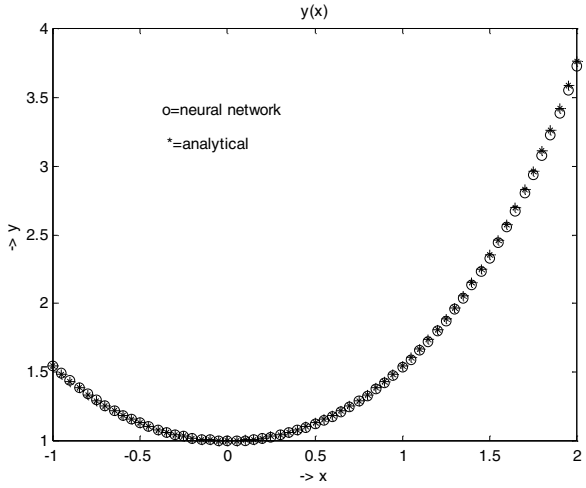

Fig.3. The solution of system (1), (2) and (3)

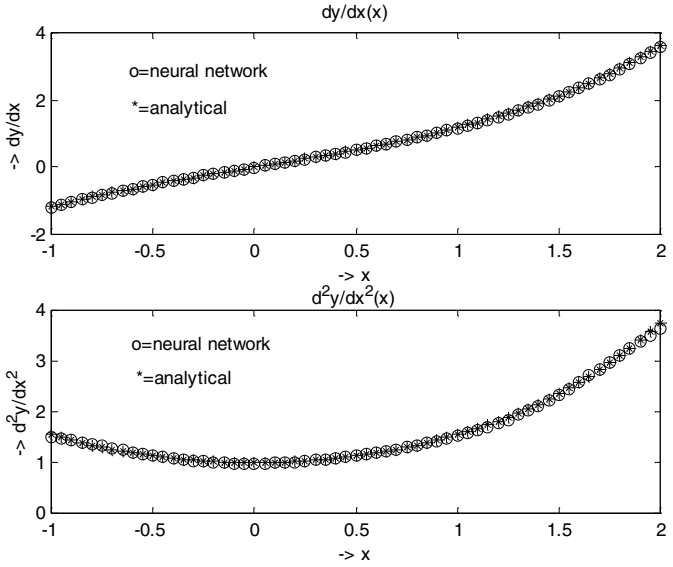

Fig.4. The first two derivatives of the solution of system (1), (2) and (3)

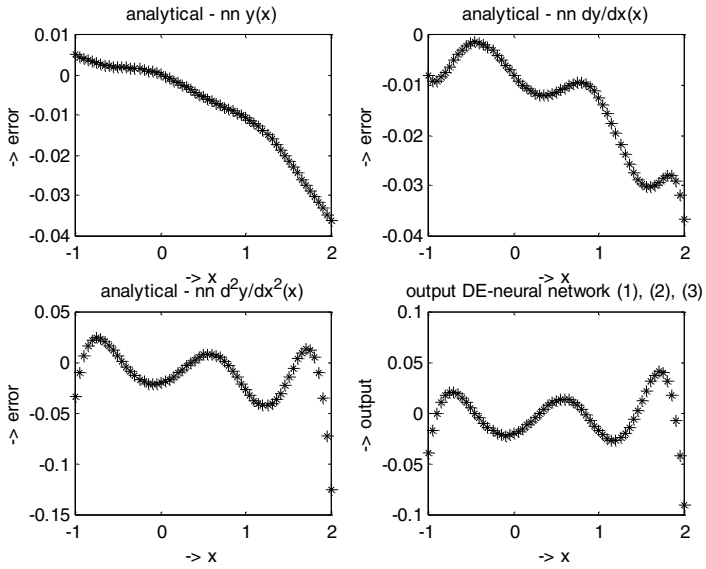

Fig. 5. The errors between the analytical solutions of (1), (2) and (3) and its first two derivatives on the one hand and the neural network method solution of (1), (2) and (3) and its first two derivatives on the other hand. Also the output of the DE-neural network of (1), (2) and (3) is illustrated