A Planning for Neural Networks Teaching in Control Using a Java-Based Toolkit

E.J. Gonzalez, A. Hamilton, L. Moreno, R.M. Aguilar, R.L. Marichal

Abstract-- Neural networks have been shown as an efficient tool in optimisation problems in general, and in control ones in particular. For this reason, Engineering students should properly learn this tool in subjects as Optimal Control or Intelligent Control. In this way, a planning for neural networks teaching is proposed in this paper, using a Java-Based toolkit (Evenet 2000) that allows to design and train neural networks with arbitrary architecture.

Index Terms-- Neural networks, software tools, education

I. Introduction

Currently, neural networks are widely used as an efficient method in many subjects such as optimisation problems and pattern recognition [1][5]. In this way, students of Engineering courses should properly learn this tool, particularly in subjects such as Optimal Control or Intelligent Control. During theoretical classes students should learn some teorethical aspects of neural networks (definition, structure, topologies or learning methods). Howevwe, after these classes, it would be very pedagogical for these courses to use a tool (or better, several tools) to train different neural networks using several training methods. With this tool, students can visualise the neural networks power in a practical way.

For this aim, there are several tools for neural networks training such as MATLAB Neural Networks Toolbox, NNSYSID, NNCTRL [6] or SNNS. However, most of them are not sufficiently general or they are not user-friendly enough. Frequently, its users cannot visualise the training of a neural network with arbitrary architecture and/or implement new training methods in an easy way. Evenet2000 [3,4], a Java-Based neural network toolkit developed at University of La Laguna, offers these advantages. This tool is based on an approach introduced by Wan and Beaufays to derive gradient algorithms for time-dependent neural networks, by using *Signal Flow Graph theory*. This approach consists of a set of simple block diagram transformation and manipulation rules. However with Evenet-2000 users do not need to know these rules. Moreover, the designed structure makes it not limited to gradient-based algorithms.

Department of Física Fundamental y Experimental, Electrónica y Sistemas. University of La Laguna. Spain.

The aim of this article is to present a planning for neural network teaching by using Evenet2000. Because this tool has interesting aspects for both neural networks teachers and researchers, it is briefly described in section 2. In section 3, teaching planning is detailed and conclusions are shown in section 4.

II. Description of the tool

Regarding the gradient algorithms in neural networks, several researchers [2],[7],[8] have shown that there is a reciprocal nature to the forward propagation of the states and the backward propagation of gradient terms. This reciprocity appears in all network architectures. Based on these properties, and using *Signal Flow Graph theory*, they have deduced a general method for automatic determination of the gradient in an arbitrary neural network. In this paper we will apply the Wan and Beaufays approach.

The first step involves representing the arbitrary network as a block diagram. There are five basic blocks:

- summing junctions
- branching points
- univariate functions
- multivariate functions
- time-delay operators.

For example, a neuron can be seen as a summing junction followed by a univariate function such as sigmoid or tanh.

From this block diagram, an *adjoint* network can be built by reversing the flow direction in the original network, performing a set of transformation rules. This philosophy suggests *object-oriented programming* as implementation method. So, Java, the most popular and powerful object-oriented language has been chosen for development of the toolkit.

Evenet-2000 consists of three basic parts: a calculation library, a user-friendly interface and a graphic neural network editor.

Evenet-2000 calculation library develops theory shown above. Every basic element is assigned an object that implements easily the adjoint method.

The basic elements are not sufficient for a complete library. They must be joined to other types of objects that implement arbitrary neural network trainings. For this,

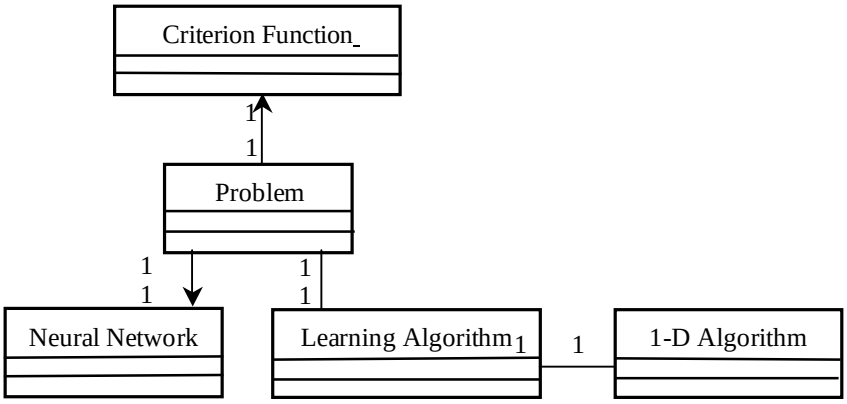

Fig. 1. Evenet-2000 Calculation Library Diagram

Evenet-2000 calculation library follows the UML diagram shown in Figure 1.

With this implementation, training and optimization problems have been uncoupled from the neural network structure specification. Problem object sets the chosen neural network structure, the learning algorithm and the criterion function (problem object is the connection among these objects). In this structure, the problem object sets the network inputs, asks the neural network object for the output vector. Then asks the criterion function to calculate the error vector, and finally the neural network calculates the gradient vector. Once this process has finished, the problem object sends the gradient vector to the algorithm. It calculates a new weight vector, by asking the problem object for cost function value. Unidimensional (1-D) optimisation could be required, so 1-D algorithms have been included. These steps are repeated until the design conditions are reached.

This calculation library can be used independently from the rest of the program. However, although the calculation library is complete and easy to use, people not used to object-oriented programming could not make the most of it. Because of this, the tool has been improved with a user-friendly interface. This way, students in our planning do need to learn objet-oriented programming at all.

This interface allows training directly a multilayer perceptron (MLP). When this case is selected from program main menu, a frame like the shown in Figure 2 appears. From its menu bar, desired learning method, criterion function, optimisation algorithm and other training parameters can be selected.

When training pairs have been loaded, the network is ready to be trained. Initial weight set can be fixed also. When the training finishes, users can study a frame showing the training error evolution and the difference between the desired and obtained outputs (Figure 3). The error and the iteration number after the training are shown in their respective text fields, and results can be saved in a text file that can be analysed later.

Evenet-2000 user-friendly interface allows users to train MLP or recurrent with no code. But this interface has not taken advantage of the possibility of training a neural network with an arbitrary architecture. For getting this purpose, a graphic editor has been included in the tool. This editor, whose frame is shown in Figure 4, can be selected

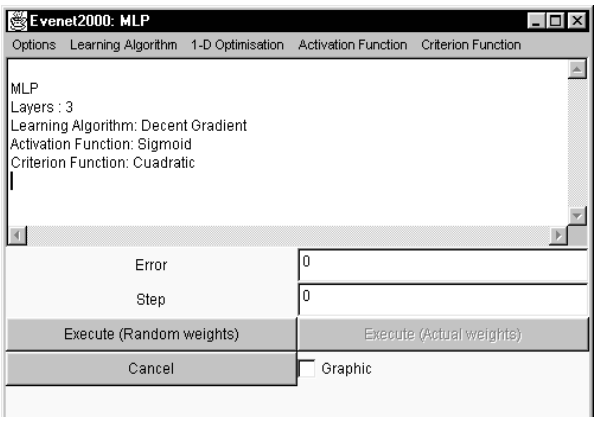

Fig. 2: Evenet-2000 MLP Frame (Fragment)

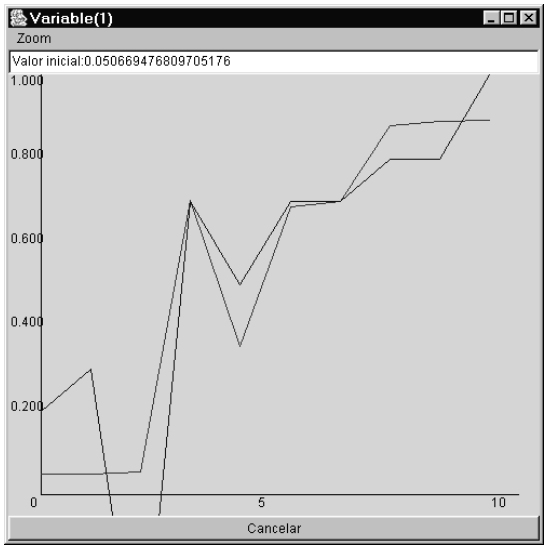

Fig. 3: Desired and obtained output

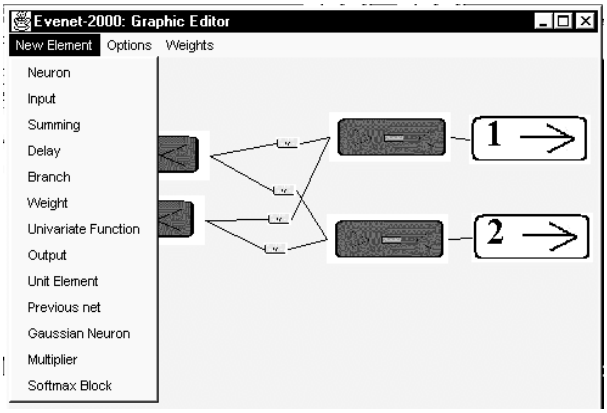

Fig. 4: Graphic Editor Frame (Fragment)

from the program main menu.

This graphic editor allows creating an arbitrary neural network. Users only have to select its elements from the menu and connect them. Its designs can be saved in a text file and loaded as new elements (modularity), developing their own neural network library this way. This text file can be modified without the help of the editor and loaded by other window of the toolkit, similar to the MLP training one described above. This way, neural networks built following any arbitrary architecture can be designed and trained with no code. This is a great advantage for the users and particularly for engineering teachers, since they are not forced to make any complicated calculations.Moreover, students are offered a tool they can visualise neural networks theory with.

III. PLANNING

In this section, the proposed planning of neural network teaching for engineering students is presented. After theoretical classes where students learn some neural networks aspects [5] as its definition, structure, topologies and several learning methods, teachers present them several procedures as practical experiences. They can be divided into two blocks: a basic block where students can visualise the theoretical aspects they have learnt and a more complex block where neural networks are applied to a control problem.

First of all, students are shown the ability of the neural networks of learning patterns. As example, a 4-layers MLP (1, 3,3, 1 and 1 neurons in each layer) is taken. From Evenet2000 menu bar, students selects the training parameters such as initial learning rate (for this example, *0.25*), limit of steps (*1000*), limit of error (*1E-4*) – when one of these limits are reached, the training is stopped – learning method (*Descent Gradient*), 1-D optimisation (*none*), activation function (*Sigmoid*), criterion function (*Cuadratic*) and initial weight set (random). A pattern file which implements the function $y(x) = x$ is loaded. This pattern file consists of 11 patterns ($0<x<1$, step = 0.1). After this

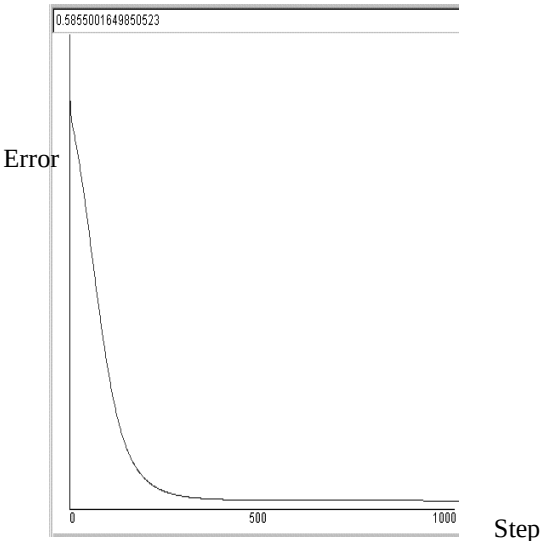

Fig 5: Training error evolution

selection, students train the network. They should obtain a graphic similar to the shown in Figure 5.

This way, students can visualize how the error decreases. Evenet2000 offers the possibility of testing if the network has been well trained. Students check the outputs for several inputs, included or not in the original pattern file (last ones are the most interesting ones), confirming that the network has learnt the proposed function (Fig. 6)

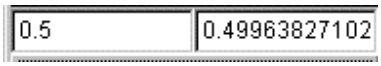

Fig 6: Training test for proposed example

Once the test has finished, students are suggested to vary some training parameters as learning rate, network topology or number of patterns. An illustrative case is obtained when learning rate is excessively big. As can be seen in Figure 7, the error evolution presents some peaks in graphic error.

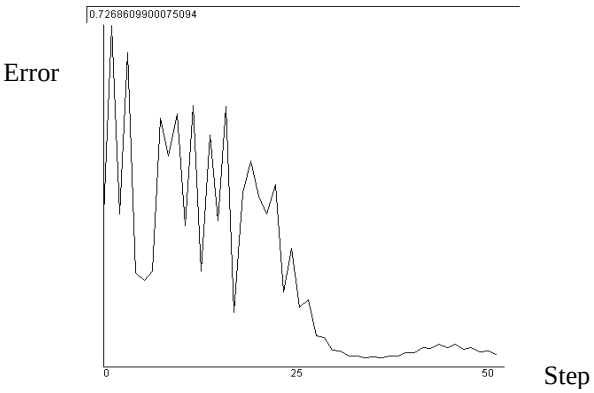

Fig 7: Training with High Learning Rate

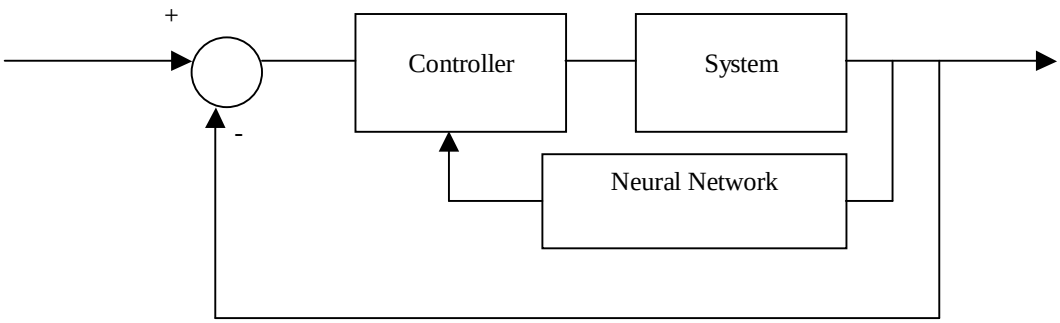

Fig 8: Control structure implemented in a control application.

Students are asked for the reason of these peaks, testing their knowledge.

Next exercises are focused on goodness of a unidimensional optimization of the learning rate. Students select an optimization method for a new training and compare its results with the obtained with no optimization. As it was expected, the number of necessary steps decreases. However more calculations are made in each step, so comparing the training time instead is a better choice. In this step, students are asked again for the goodness of the different algorithms.

These basic experiences may be complemented with studies about recurrent networks and/or other topologies. For this purpose, these topologies can be designed with the graphic editor of the tool.

With this set of experiences, students have visualized the aspects that they had learnt in theoretical classes, getting a deeper knowledge of neural networks. Estimated time for this first block of procedures is about 20 hours divided into 4-5 sessions.

On the other hand, with the second set of procedures, students can learn about applications of neural networks in problems more complex than learning of some patterns. In the planning proposed in this paper, a simple but interesting control problem has been chosen. There are two options for the system to control: a real system (for example the tank system shown in Figure 9) or a software-simulated one. Procedures with real systems offer a great advantage. Engineering students usually find more interesting to work in the "real life" than with simulated systems. However, if the system is not carefully chosen, its control could become too difficult for a laboratory procedure. So it is recommended a first or second order system. On the other hand, working with simulated systems offers more variety and are easier to debug.

In both cases, students should implement a control structure like the shown in Fig 8. Parameters concerning to proportional and integral terms of the controller (students work with discrete equations) are supplied for a neural network, trained through Evenet2000 modules. Students should compare this control structure with a controller with fixed proportional and integral terms. This way, they test the goodness of neural networks for control systems.

Estimated time for this block of procedures is about 20 hours divided into 4-5 sessions.

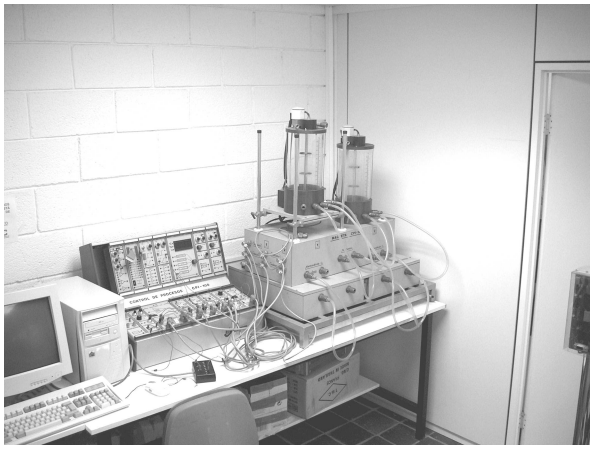

Fig 9: Example of real plant for the second proposed block of procedures

IV. CONCLUSIONS

In this paper, a planning for the teaching of neural networks in Engineering subjects is proposed. For this, Evenet2000, a Java-based neural network toolkit, is used. This toolkit allows students to design and train neural networks with arbitrary architectures. In theoretical classes, students learn aspects as definition, structure, topologies and several learning methods. After these classes, two blocks of procedures are proposed to students. In the first block, students can visualise these theoretical aspects. The second block is more complex and students apply neural networks to a control problem. This block is proposed to students more familiarised with object-oriented programming and control systems.

REFERENCES

[1] Acosta L., Marichal G.N., Moreno L., Rodrigo J.J., Hamilton A., Méndez J.A. (1999). Robotic system based on neural network controllers. *Artificial intelligence in Engineering*, 13, number 4, pp 393-398.

[2] Campolucci P., Marchegiani A., Uncini A., Piazza F. (1997). Signal-Flow-Graph Derivation of On-line Gradient Learning Algorithms. *IEEE International Conference on Neural Networks*, Houston (USA).

[3] González E.J., Moreno L., Hamilton A., Piñeiro J.D., Marichal R., Marichal, G.N. (2000). Evenet2000: A New Java-Based Neural Network Toolkit. *Proceedings of the Second ICSC Symposium on Engineering of Intelligent Systems*, Paisley, June 2000.

[4] Gonzalez E.J., Hamilton A., Moreno L., Sigut J., Marichal R. (2001) Evenet-2000: Designing and Training Arbitrary Neural Networks in Java. *Bio-Inspired Applications of Conectionism*, Springer Verlag, Lectures Notes in Computer Science, 2085.

[5] Haykin, S. S. (1998) Neural Networks: A Comprehensive Foundation. Prentice Hall.

[6] Nørgaard M., Ravn O., Poulsen N.K., Hansen L. K. (2000) Neural Networks for Modelling and Control of Dynamic Systems. Springer-Verlag, London.

[7] Osowski S., Herault J. (1995) Signal Flow Graphs as an Efficient Tool for Gradient and Exact Hessian Determination. *Complex Systems*, 9, 1995.

[8] Wan E. A. and Beaufays F. (1994) Relating real-time backpropagation and backpropagation through time. An application of flow graph interreciprocity. *Neural computation*, 6, number 2, pp. 296-306.