A framework for the definition and generation of artificial neural networks

Diego Ordóñez, Carlos Dafonte, Bernardino Arcay
University of A Coruña
Department of Information and Communications Technologies
Campus de Elviña, A Coruña 15071
Spain

Minia Manteiga
University of A Coruña
Department Navigation and Earth Sciences
Campus de Riazor, A Coruña 15011
Spain

Abstract: Current platforms for the definition and operation of artificial neural networks are oriented towards a specialized user profile. This article proposes a series of software components that provide abstraction in platform development, both with regard to the implementation of the components and the operation of the network that was chosen to resolve a problem. This proposal has the ambitious purpose of minimizing the learning curve of the tools and providing a framework that is oriented towards a more generic user profile. The framework is composed by three systems: a library, in which we factorize the functionalities related to the networks, and two tools, an interpreter and a Web application, which provide two different points of view for the development of the networks, but both rely on the library to provide the features related to the ANNs.

Key–Words: Artificial neural networks, web application, framework, toolkit, code generation.

1. Introduction

Artificial neural networks (ANNs) are among those computational models that are inspired on the solutions that nature has found in the course of evolution. Our work is focused on developing mechanisms that allow us to manipulate these networks and generalize their use. In order to achieve these objectives, we concentrated on presenting the user with a framework that allows him to think of the networks. We searched for abstraction with regards to the implementation and execution platforms of the ANNs.

Various environments provide the user with high level mechanisms that allow him to accelerate the development of solutions: whereas Matlab's Neural Network Toolbox is a commercial version, SNNS is a free software tool. A common characteristic for this type of tools is that they do not provide a satisfactory support for the export of their functionalities and the use of ANNs in other platforms.

The here proposed alternative tries to solve some of the problems of the existing tools for the development of connexionist models. It is a tool that provides the user with a series of features that are not easily found: generation of multiplatform code, abstraction with respect to the implementation platform, and user-friendliness.

1.1. Objectives and scope

We started by elaborating a library for ANNs that provides support to the features that are proper of networks, such as architecture, training algorithms, activation functions, etc. The design of the library is such that new components can be added as a natural process; a representative subset of networks and algorithms has been implemented.

The library enables the user to export the functionality of the implementation platform ANNs to the platform where they will be used, thanks to a **code generation** mechanism. Our purpose is to enable a network instance that was trained to solve a problem, and does so succesfully, to have the same behaviour regardless of the platform in which it is used. This feature is particularly relevant because of the fact that two different trainings can almost certainly never obtain the same result. The central idea of or work is to train a network once and then use it wherever it is required.

Once defined and implemented, the library provides us with a support for the elaboration of other software that delegates in it the operations related to the ANNs. This minimizes the implementation errors during the generation of other software components that rely on them (since the algorithms are duly tested). We provide two development environments that make use of it. A declarative language and an interpreter, for users who are familiar with programming languages,

and a Web application, oriented towards fast developments. The characteristics of these approaches can be seen in sections 2.3 and 2.2.

Our last objective is to apply this development to real a problem with real data; this will allow us to test its efficiency in searching for solutions using connexionist models. We have tackled a problem that belongs to the field of Astrophysics: the extraction of stellar parameters (such as temperature) from the information held by their spectrum.

2. Components

The kernel of the system is formed by components mentioned in section1.1: a library (NeuralToolkit) and two tools, shown in Figure 1.

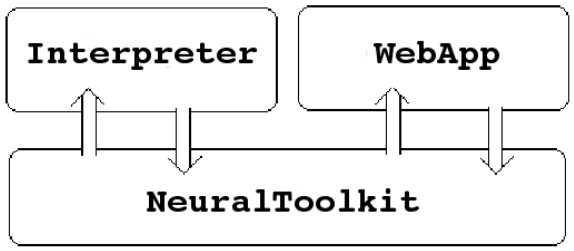

Figura 1: Developed components

2.1. NeuralToolkit

The main component of the system is the library called NeuralToolkit. It is the result of factorizing the training architectures and algorithms and other entities related to the ANNs. Their definition will mark the limit between what can and cannot be done with the rest of the developed components, which only provide the interface for handling the library. The NeuralToolkit was conceived to facilitate the addition of new software components without changes in the system architecture, and therefore plays a fundamental role in incremental developments. This focus provides ample possibilities for the maintenance and the generation of increasingly complete software versions. The current version of the library gives support to various entities. In what follows, we show an inventory of some of these entities according to their category, and a brief explanation of why they were selected:

- **Network Architectures** [1]: We have tried to select a significant subset of architectures: Feed Forward (with proven applicability for the resolution of real problems), SOM (distinctly representative of competitive architectures), Cascade Correlation [2] (architecture network of the incremental type), CPN (typical example of a composed network), and Hopfield (for its behaviour as a self-associative memory).

- **Training algorithms** [1]: Backpropagation (online and batch), Kohonen, Fahlman [2], Hopfield, and Counterpropagation. These algorithms were chosen according to the selected network architectures and, consequently, to train them.

- **Activation Functions**: Linear, Sigmoid, Hyperbolic Tangent, Hardlimit, and their derivatives if they exist. In our opinion, these are the most frequently applied activation functions.

We have incorporated the capability of ANN composition, which is a relevant functionality. The composition propagates the output of a network towards the inputs of another with which it connects. The output of the composed network is the result of propagating inputs throughout all the intermediate ANNs. The output network of the composition is that network whose output is not connected to the inputs of any other network. The idea of the composition can be seen in the example of Figure 2, where we observe a simple composition of two networks: Net 1 and Net 2. Net 1 is the network that behaves as input of the composition and propagates its outputs towards the inputs of Net 2.

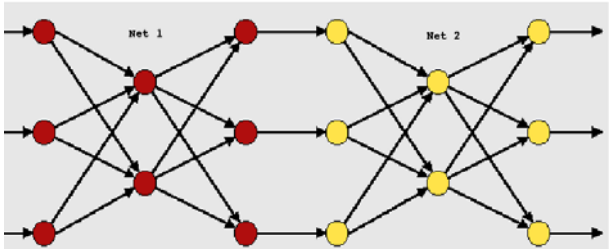

Figura 2: Composition of two feed forward networks

Among the innovating features are the neighbourhood functions, adapted to the problem for the Kohonen training. In NeuralToolkit framework, we can define high level neighbourhood functions for the training (cylindrical, spherical, linear, rectangular etc). This means that instead of indicating connectivities, we need to specify the shape towards which the user wants the forms to tend.

As mentioned in section 1.1, one of the central features of the library is **code generation**. In this version, the network mapping is supported by *ANSI C*. We have opted for this language because the programmes will be fastly executed in almost any platform

for which we compile the code. Also, it is a standard with implementations for multiple platforms, including embedded systems.

NeuralToolkit was implemented in Java. The network architectures form a tree of classes that share the feature of being what is known as serializables, by which an instance of a concrete architecture can be transformed into an array of bytes. This property allows us to store the object into a database or a file and recuperate it afterwards. It is also useful if we need to transmit the object through the network.

2.2. Web Application

The functionality, exposed by the web application, with regard to the handling of entities related to the ANNs, is contained in that of the interpreter, which in turn is contained in the potential functionality of the NeuralToolkit. However, there are differences with regard to the visualization and the information management:

- Focus based on forms
- More detailed information on the training
- Better oriented error messages
- Persistency of the information (patterns management, ANNs, etc)

The focus is similar to what is offered by a standalone application, but with the advantage that it does not require to be installed. The user who possesses an account through which he can access the application has the complete functionality within his reach. The ANNs that belong to a determined user are available in any place. The application provides the user with a way to organize his information: the application provides pattern and ANNs repositories, the possibility to save intermediate networks during the training and retain the one that generalizes best, the possibility to concatenate trainings for a determined instance of a network if we do not agree with the performed training, etc.

We have implemented the necessary mechanisms to guarantee the storage of the information in a database and recuperate it upon request. This feature also provides the user with more capacity to manage the entities he is handling; capacities that manifest themselves in the shape of pattern and ANNs repositories that the user can search and recuperate at any given moment.

The Web application was elaborated according to the architectonic design patterns MVC[1] and Layers.

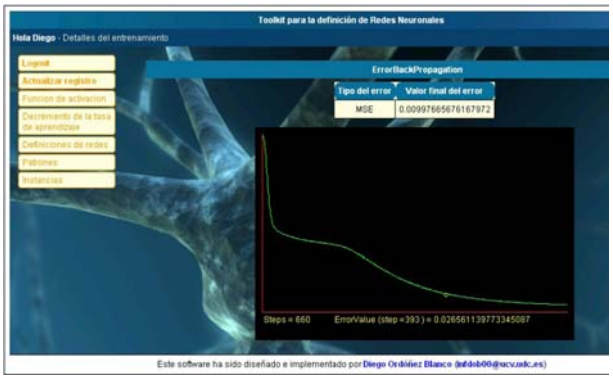

Figura 3: Content of a page of the Web application

The application view was generated with JSP pages, and the controller was developed with Struts [6]. The application is packed in a standard WAR file and deployed in the Web J2EE container Tomcat [7]. The information is saved in a database that can be accessed via JDBC; the used database manager is PostgreSQL [8].

2.3. *Ad Hoc* Language and Interpreter

The grammar was defined for a declarative language that allows us to define the architectures, specify the training parameters, and perform the operations related to the ANNs.

The idea behind this alternative is to equip the user with more expressive capacity: give the user an alternative that allows him to carry out more complex operations according to his knowledge. It is an alternative for applications of the WYSIWYG[2] type, such as the Web application presented in section 2.2.

```
NetDef netId Type FeedForward (
       LayerDef inLayer EEPP 3 Af Lineal (Slope 1.0);
       LayerDef hiddenLayer EEPP 3 Af Sigmoid (Gain 1.0);
       LayerDef outLayer EEPP 2 Af Sigmoid (Gain 1.0);
       InputLayer inLayer;
       OutputLayer outLayer;
       ConnectionModel (
           inLayer[0 .. 2] -> hiddenLayer[0 .. 2];
           hiddenLayer[0 .. 2] -> outLayer[0 .. 1];
       );
);
[...]
Instantiate netId As instanciaA;
[...]
```

Figura 4: Feed Forward network definition example

The code in Figure 4 defines a Feed-Forward ANN of 3 layers (input, output, and hidden). The in-

[1] Model-View-Controller

[2] What You See Is What You Get

put layer has three process elements and a linear activation function, the hidden layer and the output layer are composed by 3 and 2 process elements respectively, with a sigmoid activation function. The connection is full, as specified in the connections model. This network definition acts as if it were the class of an object-oriented language that can be instantiated.

The source programme can be reused in subsequent testing, or it can be used in other developments as a template. The result of the programme will be the code in some or other output language that represents some of the instances declared in the programme. The tool consists in the compiler that interprets this programme and returns the source code for the output platform.

The interpreter provides the functionality of separating the definition of the network from the implementation thanks to the declarative language. This abstraction allows the user to focus on the ANNs instead of on how to implement them.

3. Application to a real problem

3.1. Problem description

The framework was used to solve a real problem in the field of Astrophysics. We are trying to determine characteristic parameters of a star, i.c. temperature and gravity, on the basis of information extracted from its spectrum. Figure 5 shows the spectrum of a star with temperature 4250 K and gravity -0,5.

The patterns were obtained from the repository of synthetic spectra at ESA[3].

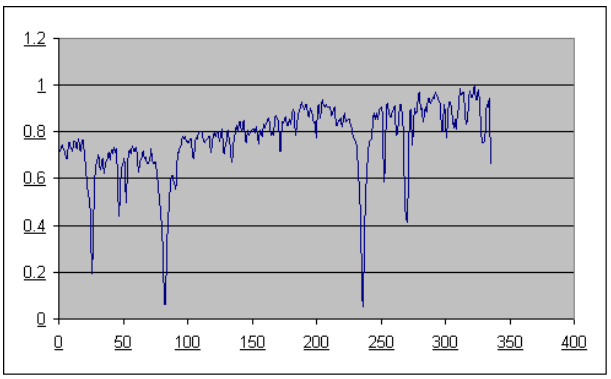

Figura 5: Stellar Spectra Sample

[3]European Spatial Agency

3.2. Description of the training network and algorithm

Training and test inputs are composed by a total of respectively 6135 and 3150 spectra. We have prepared two training sets, one for each parameter[4].

The network architecture that was selected to solve the problem is the three-layered Feed Forward architecture (input, hidden, and output), and the training algorithm is *online* error backpropagation. We opted for this architecture because it has already been tested, and very successfully so, on problems related to stellar spectra [3]. We have tried several configurations of process elements for the hidden layer, but the best results were obtained with a hidden layer of 150 (see section 3.3). Figure 6 shows an example of a Feed Forward ANN, in which vector $X = (X_1, X_2, X_3, X_4, X_5)$ represents the inputs and vector $Z = (Z_1, Z_2)$ the outputs.

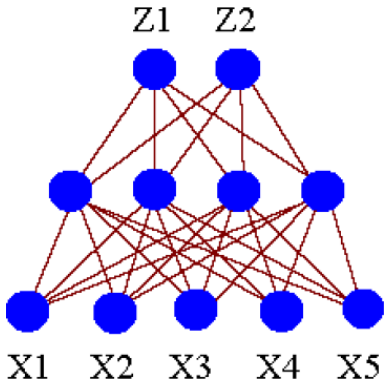

Figura 6: Generic Feed Forward architecture example

3.3. Results

The graphs of Figures 7 and 8 show the average errors. We have carried out the training with a set of patterns that mix noisy and clean inputs, which allow us to train ANNs that better recognize the patterns with noise. The symbols $n10$, $n50$, $n100$ and $n500$ indicate that we have generated various versions of the validation set by applying different noise intensities, i.c. relations signal-to-noise of 10, 50, 100, and 500 respectively. The size of the bars in the diagram represents the measurement of the average error for all the patterns of the validation set and for the corresponding signal-noise relation.

The temperatures of the training set lie in the 4000K to 8000K range[5], and the gravity in the -1.0 to 5.0 range.

[4]Temperature and gravity

[5]This range is the only context that is apt for results analysis

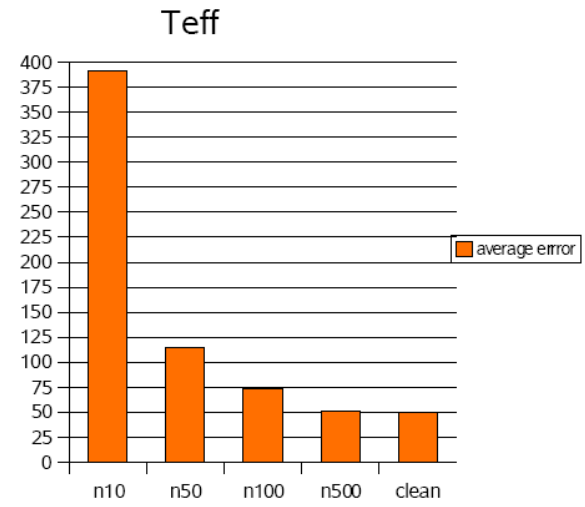

Figura 7: Average errors in temperature

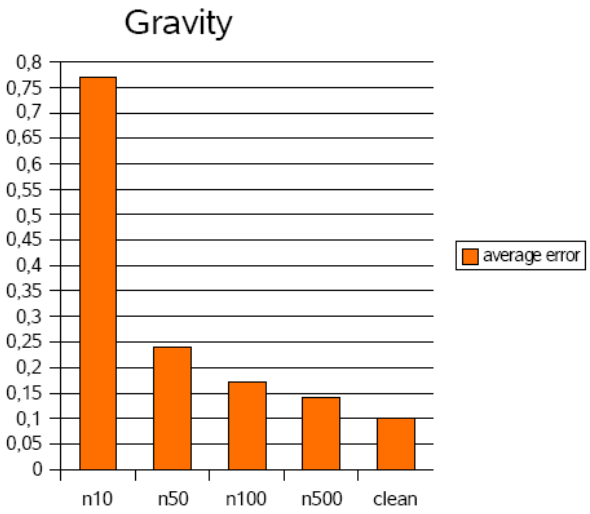

Figura 8: Average errors in gravity

The networks were trained with 10000 steps, during a total of 28 hours on a PC with AMD Athlon 64 processor, 3800 GHz and 2 GB memory. The fact of having trained them with 10000 steps does not mean that the results were obtained with the final network; we save the intermediate ANNs (e.g. each 100 training steps) and use the ANN that presents the best generalization.

The problem is complex, the amount of training and test patterns is high, and the results measured in average errors are satisfactory ([3], [4]).

4. Conclusions

While the library provides what is needed to define the ANNs and their handling, and the tools abstract the implementation platform and the network destination platform, the user focuses exclusively on the development of the networks.

The framework was entirely developed in Java and therefore cannot be used in all the platforms for which a virtual machine exists.

One of the most noticeable features of the developed environment is the possibility to derive from an instance the **output code** of a network. We have observed the addition of new destination languages to the development, as well as the extension of other features such as network architecture, training algorithms, etc. Therefore, and following the guidelines of the design, we can increasingly develop components that subsequently will be added to the rest of the software.

Grouping the functionalities of the ANNs into a library will allow the future development of new tools based on this library without having to re-implement the same features. This implies a smaller error margin and an important increase in development time.

The developed tool was used to solve a real and complex problem such as the extraction of stellar parameters from examples of stellar spectra. As we can observe in Figures 7 and 8, the results were satisfactory.

References:

[1] Freeman, J. A., y Skapura, D. M. *Neural networks: algorithms, applications, and programming techniques*. Addison-Wesley 1991.

[2] Fahlman, S. and Lebiere, C: The cascade-correlation architecture. In Touretzky, D.S,, editor, Advances in Neural Information Processing Structures 2 (1990), pages 524-532. Morgan-Kauffmann.

[3] P.G. Willemsem, C.A.L. Bailer Jones, T.A. Kaempf. *Analysis of Stellar Parameter Uncertainty Estimates from Bootstraping Neural Networks*. 2004 September 08.

[4] A. Recion-Blanco, A. Bijaoui and P. de Laverny. *Automated derivation of stellar atmospheric parameters and chemical abundances: the MATISSE algorithm*, 2006 April 11.

[5] https://javacc.dev.java.net/

[6] http://struts.apache.org

[7] http://tomcat.apache.org

[8] http://postgresql.org