7th Seminar on Neural Network Applications in Electrical Engineering, *NEUREL-2004*
Faculty of Electrical Engineering, University of Belgrade, Serbia and Montenegro, September 23-25, 2004

A GUI-BASED ARTIFICIAL NEURAL NETWORK SIMULATOR[1]

George E. Nasr, *Member, IEEE*, Carla Joun and Wadih Zaatar, *Member, IEEE*

Abstract— **In this paper, a java-based artificial neural network simulator is presented. Requirements for an educational model are also introduced and implementation procedure is then depicted. Feature comparison between this model and some well know similar tools such as Xerion and SNNS are also performed. Finally, extension possibilities, including a client-server approach suitable for education and classroom approach are also discussed.**

Index terms— **Artificial Neural Network, Education, Graphical User Interface, and Java.**

INTRODUCTION

Artificial neural networks (ANNs) have been applied to a wide range of real-life problems, situations where generic linear and non-linear methods have failed. If trained carefully, ANNs may exhibit generalized solutions that go beyond the initial model used for the training itself. Widrow [1] presented a comprehensive survey of ANN applications in many different fields of business, industry and science. Currently, neural networks algorithms, applications and programming techniques are being introduced to undergraduate students in most computer science and engineering curricula. Simulation tools to enhance the comprehension of the various aspects of ANNs by students are becoming essential to the educational process. This paper presents a user friendly java-base ANN simulator for this purpose. The main objective of this work is to provide a new neural network modeler that allows the user to experiment with the various available ANN algorithms and to build his/her own model followed by a comprehensive testing and assessment of the model.

This paper presents the proposed model for the Graphical Artificial Neural Network simulator, alongside the most important functions that allow inserting a user-customized learning or activation function and other related parameters. A short comparison with other simulation tools will also be discussed. Finally, directions of future versions and features of the application are discussed.
This tool is currently being used in AI classes where students build their own ANN models for a wide range of applications through extensive training and testing. The simulator is also used by students to perform a sensitivity analysis of the ANNs with respect to the activation and error functions and to other pertinent network parameters. In addition, the resulting models can be evaluated using any specified performance or error measure.

ARTIFICIAL NEURAL NETWORK OVERVIEW

Artificial neural networks have been developed as generalizations of mathematical models of human cognition and neural biology; the basic unit of an ANN being an artificial neuron, similarly to its human counterpart. Every input to a neuron has an associated parameter, called weight attached to it. It receives this input from other neurons or external sources [2]. Each neuron computes a function, called the activation function from the weighted sum of its input: It is this activation function that drastically affects the ANN's performance, therefore varying this function as well as others and observing the generated output is one of the most important topics in ANN research fields. [3]

PROBLEM FORMULATION

Neural networks have always had their fair share of implementation in the computing world; many applications have been developed to ease the process

[1] This work was supported in part by the Lebanese American University under grant # URC-t2004-28.
George Nasr, Carla Joun and Wadih Zaatar are with the Lebanese American University, Department of Electrical and Computer Engineering, Byblos, Lebanon (George E. Nasr, phone: +961 (9) 547254 ext. 2367, fax: +961 (9) 547256, email: genasr@lau.edu.lb)

of understanding and solving problems using artificial neural networks. However, due its complexity, this has not been an easy task. To be operational in an educational context, a neural network simulation application should have the following characteristics:

- A graphical user interface, allowing the user to formulate his problem in a friendly manner, with all input parameters clearly defined. A good implementation would also provide a way for displaying output properly.
- Code portability, either source or executable. The solution should be transposable from one operating system to the other, preferably with minimum to no code modifications.
- Modularity, as neural network problems vary, the application should be modular enough to allow inclusion or update of several functionalities, typically the learning algorithm or the activation function.
- Responsiveness, as the application should not only be user friendly, but also fast. Neural network simulations can become quite tedious, especially when it comes to complex activation functions.
- Versatility, as this tool would be used in the educational arena, it might be interesting to include server/client or instructor-led capabilities.

PROPOSED MODEL

To be able to satisfy the model requirements, several design issues had to be answered:

- Using a platform-independent programming language, the choice was set on Java, as being cross-platform, and thus fully portable. The cost of using Java would have a significant impact on responsiveness; due to the fact Java bytecode although platform compatible with most operating systems gets interpreted and not compiled at runtime.
- Java's generic class files support allows a high modularity level. This will allow decomposing the Neural Network engine into smaller, user-customizable blocks. [4]

A. Objects Description

Object oriented programming offers a new and powerful model for writing ANN computer software. OOP is, at its highest level, the process of building applications or systems with objects. This approach speeds the development of new programs, and improves the maintenance, reusability, and modifiability of software. Our package is fully developed using Java programming language and the platform used is the sun Java 2 SDK v.1.3 product. Table 1 describes some of the available objects within the Java GANN simulator.

B. Functionality

The GANN simulator proposes two execution models:

The first model is a stand-alone, shell based command-line where the user feeds in the system with his input parameters using a GANN nfo file, this file's format is strictly preset and allows the user to enter various information, such as the learning rate, momentum, slope, number of hidden layers and

Table 1: GANN Java Classes

Class Name	Class Description
ActivationFunction	This class implements the activation function class.
ArtificialNeuron	This class provides the basic data model and function for an artificial neuron.
BackpropagationAlgorithm	This class implements the back-propagation learning algorithm.
Gaussian	This class implements the Gaussian activation function.
LearningAlgorithm	This abstract class implements the data and behavior common to all learning algorithms.
MLPFullyConnectedNet	This class implements the Fully Connected multi-layer perceptron network.
MLPNonFullyConnectedNet	This class implements the non-fully connected multi-layer perceptron network.
NetworkLayer	This class implements the basic network layer class.
NeuralNetwork	This class implements a basic network of neurons; and provides a base object independent of any learning algorithm or architecture .
PiecewiseLinear	This class implements the piecewise-linear activation function.
Sigmoid	This abstract class implements the Sigmoid activation function
CustomError	This is a user-definable class that feeds in a custom error function.
CustomActivation	This is a user-definable class that feeds in a custom Activation function.

the corresponding number of hidden units, Table 2 shows the nfo file format description.

The second model is a graphical user interface, where the user will be given the opportunity to enter his parameters using the GUI which will save these parameters as a GANN nfo file and pipe them to the command-line interface. Parameter file modification being extremely fast, this will allow the user to experiment with several input parameters and check his results accordingly. The GUI's main screen is presented in figure 1.

C. *Modularity*

Due to the usage of Java as the programming language, we are able to segment the computational model into several blocks. GANN.class and GANN-shell.class respectively being the GUI and command-line interface for the application; this will also allow us integrating external, user customized class implementations. The following three classes are currently customizable: ActivationFunction (CustomActivation.class), ErrorFunction (CustomError.class) and LearningFunction (CustomLearning.class).
By default these classes are not called by the GANN simulator; a user planning to test his own function would simply replace the proper class file with his own and set the configuration file's proper branch trigger value to True. This will replace the standard function with the user defined one. This feature is currently available from the shell and not supported by the GUI.

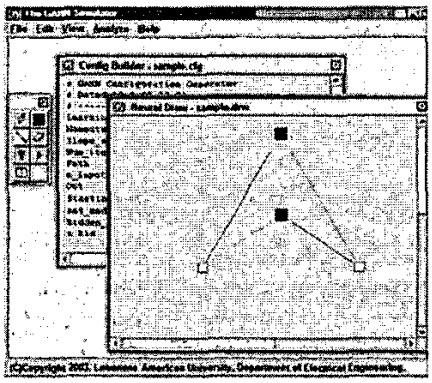

Fig. 1: The GANN GUI

D. *Execution Sequence*

Following is a typical execution:
- Depending on the user command line, the program would enter load the configuration file that will direct it to all other files (in_training, in_evaluation and in_testing) or start with the Graphical User Interface that simply builds up is neural network from scratch or loads an existing model. In either case, the GUI simply writes the design constraints in a 'nfo' files and returns control to the core engine. Once all parameters are defined, the program moves to normalizing the input matrix to values between 0 and 1 (exclusive) as networks have been observed to be per formant when data is normalized.
- Once the normalization process is completed, control is turned over to the Network Builder that builds the global predictive network based on the user input in the nfo file. At this stage, all essential functions are selected and the presence of a user defined CustomActivation function or CustomLearner function would override the default functions (Gaussian, Piecewise Linear or Sigmoid for the Activation and Backpropagation for the Learning Algorithm)
- Control is then handed over to the training block that applies the selected propagation algorithm and initializes the training process.

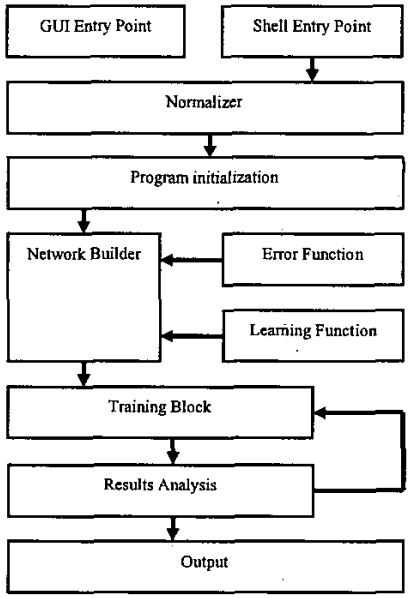

Fig. 2: GANN Execution Sequence

COMPARATIVE ANALYSIS

Many educational ANN simulation packages have been present for quite some time. A fine example of these packages is the Stuttgart Neural Network Simulator (SNNS), with its latest Java-based

implementation. SNNS presents many features that supersede GANN in several ways, to mention a few:

- An increasingly big library of activation, learning and error functions.
- An advanced Graphical User Interface capable of modeling the neural networks in both 2D and 3D.
- Caching support to accelerate the training operations. [5]
- Many more features. [6]

Table 2: GANN NFO File Descriptors

Parameter	Description
DataFolder	Folder for the stored training
Parameters	Number of Parameters to load
Param.n	Parameter n, number of inputs
LRate	Learning Rate
Iterations	The number of iterations
Hidden	Number of hidden layers
CActivation	Custom Activation branch Trigger
CError	Custom Error branch Trigger
CLearning	Custom Learning branch Trigger

On the other hand, one can run complete ANN simulations using a professional package such as MatLAB®. This computational software would allow modeling any ANN system with some programming.

It is, however important to mention that GANN's "raison d'etre" is not to compete with an advanced simulator like SNNS, nor is it to offer the wide range of functions that a powerful but general computing platform can offer, it should be considered as a hybrid solution, specifically targeting classroom-based experimental ANN analysis and implementation. GANN has been designed with two objectives in mind:

- Providing the user with a fair set of ANN models, sufficiently large to solve simple real-life problems and targeted toward analyzing how the ANN model solves the problem and not the solution itself.
- A GUI allowing a user to create, edit and manipulate 2D neural networks.
- The possibility to modify the most important parameters in the ANN models by providing the user with a way to include and test his own ANN functions.

Most importantly, GANN is open source, the code being freely downloadable online. It has been used by several students to solve ANN problems in Electrical forecasting, speech and pattern recognition. [7]

CONCLUSION AND FUTURE WORK

In this paper, a generic, multi-purpose Artificial Neural Network simulator has been presented. This tool is currently being used in IA classes where students experiment and test various functions and their impact on outcome of the ANN simulation. A brief description of objects being used by the simulation, as well as a model flowchart of the execution sequence has been portrayed. Many future enhancements and add-ons are now being considered for the application enhancement. An online, web-based GUI which allows building the blocks and feeding the functions to be run on the server is being considered; another methodology, involving server/client architecture for instructor led sessions is also being assessed. This would increase the usability of the application in the educational field.

REFERENCES

[1] B. Widrow, D.E. Rumehart, M.A Lehr: Newvol Networks. Applications in industry, business and science, *Communications of the ACM*, 37(3), 93-105, 1994.

[2] L. Fausett eds, "*Fundamentals of Neural Networks*", New Jersey, Prentice-Hall, 1994.

[3] P. Peretto eds, "*An Introduction to the Modeling of Neural Networks*. Cambridge University Press, 1992.

[4] J. Heaton: An Introduction to Neural Networks in Java, Informit.Com, 1992.

[5] J. Johnson, P. Picton: How to train a neural network. Complexity; 1:13-28, 1996.

[6] A. Zell, et al: SNNS: Stuttgart Neural Network Simulator. Version 4.1. Computer software and reference manual. University of Stuttgart: Institute for Parallel and Distributed High Performance Systems, 1995.

[7] G. E. Nasr, E. A. Badr, M. R. Younes: Neural Networks in Forecasting Electrical Energy Consumption. *FLAIRS Conference 2001*: 489-492, 2001.