INEB- PSI Technical Report 2006-1

Neural network software tool development: exploring programming language options

Alexandra Oliveira

aao@fe.up.pt

Supervisor:

Professor Joaquim Marques de Sá

June 2006

INEB - Instituto de Engenharia Biomédica

FEUP/DEEC, Rua Dr. Roberto Frias, 4200-645 PORTO

Contents

Introduction

The animal nervous system continually receives information, processes it, and makes appropriate decisions. The brain is a very complex structure with the capability to perform certain computations (e.g., pattern recognition, perception, and motor control) many times faster than the fastest digital computer in existence today. The receptors convert stimuli from the human body or the external environment into electrical impulses that convey information to the neural net [3]. Some independent cellular units - the neurons - are activated. They process the information and activate other neurons with the output of their activity. This dynamics has inspired many computer sciences studies, namely on mimicking some functionalities of the brain with artificial systems: artificial neural networks.

The tasks presented in this report were developed during the first semester of year 2006, learning the concepts related to neural networks and programming skills and software possibilities leading to an adequate selection of language and environment where to build a software tool with a friendly graphical interface that implemented these systems.

Artificial Neural Networks

This chapter describes a short foundation of neural networks which reflects a few months of needed study to implement a software tool.

An artificial neural network is a massively parallel distributed processor made up of simple processing units (neurons), which has the ability to learn functional dependencies from data. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

The procedure used to perform the learning process is called a learning algorithm, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective [3].

Each neuron is a simple processing unit which receives some weighted data, sums them with a bias and calculates an output to be passed on (see figure 1). The function that the neuron uses to calculate the output is called the activation function.

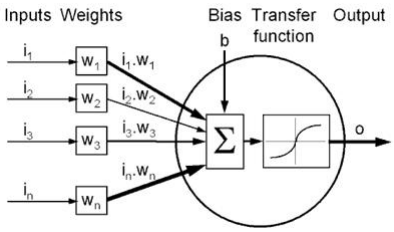

Figure 1: Graphical representation of a neuron where $o = f(i_1.W_1 + i_2.W_2 + i_3.W_3 + \ldots + i_n.W_n + b) = f(\sum_{j=1}^{n} i_j w_j + b)$ where f is the activation function.

Typically, activation functions are generally non-linear having a "squashing" effect. Linear func-

tions are limited because the output is simply proportional to the input. In the figure 2 are shown some common activation functions.

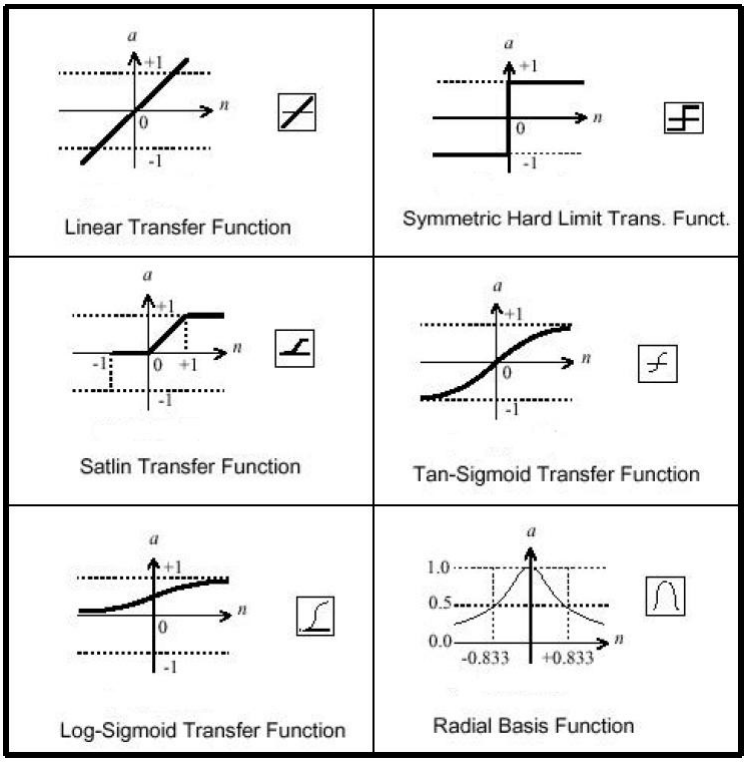

Figure 2: Common activation functions

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network [3]. The most common architecture is the multilayer perceptron (MLP). These networks are a feedforward network where the neurons are structured in one or more hidden layers. Each perceptron in one layer is connected to every perceptron on the next layer, hence information is constantly "fed forward" from one layer to the next.

By varying the number of nodes in the hidden layer, the number of layers, and the number of input and output nodes, one can classify points in arbitrary dimensional space into an arbitrary number of groups. However, the Hornik-Stinchcombe-White theorem, states that a layered artificial neural network with two layers of neurons is sufficient to approximate as closely as desired any piecewise continuous map of a closed bounded subset of a finite-dimensional space into another finite-dimensional space, provided that there are sufficiently many neurons in the single hidden layer [4].

The network learns about the input through an interactive process of adjusting the weights and the bias. This process is called supervised learning and the algorithm used is the learning algorithm.

One of the most common is the error backpropagation algorithm. This algorithm is based on the error-correction learning rule, based on gradient descent in the error surface.

Basically, a set of cases, with the corresponding targets, is given to the network. The input data is entered into the network via the input layer and is processed through the layers - forward pass. Then, the output is compared to the expected output (the targets) for that particular input. This results in an error value. This error value is backpropagated thought the network, against the direction of the weights. The weights and bias are adjusted to make the actual response of the network move closer to the desired response in a statistical sense (see figure 3) .

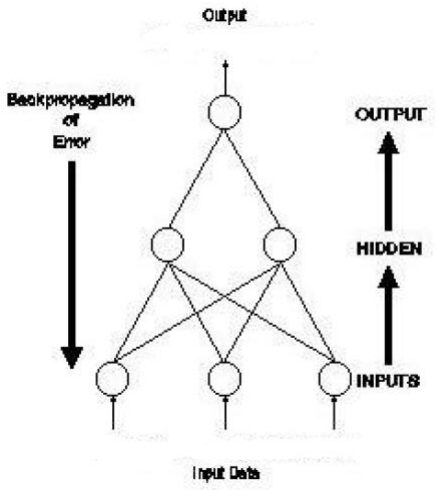

Figure 3: Backpropagation of error

So, the backpropagation algorithm looks for the minimum of the error function in the weight space using the method of gradient descent. The combination of weights which minimizes the error function is considered to be a solution of the learning problem. Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function [8] (see 4).

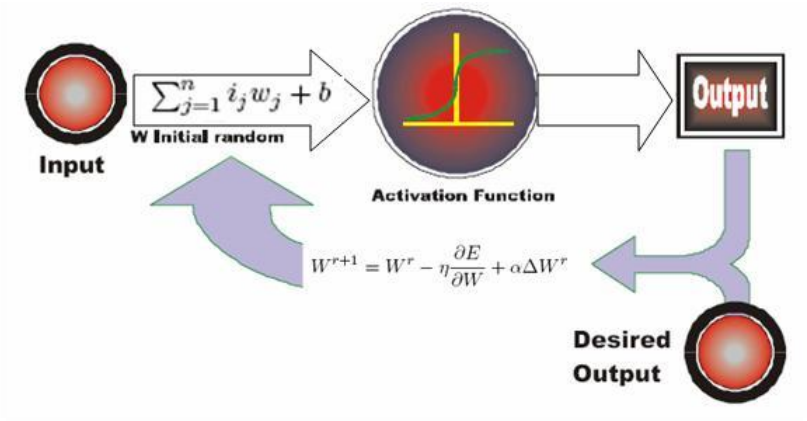

Figure 4: Weights and bias update

Neural Network Software Tool

The main objective of our work is to create a software tool that implements neural networks. This software tool must integrate the algorithm developed by the investigators involved in the ENTNETs project and must have a good and easy graphical interface.

The graphical interface of this new software tool, should be a icon-driven tool that allows the user to specify a block diagram representation of a neural network. The block diagram is composed of icons, representing different components of a neural network (inputs, neural network architecture, learning algorithm, plot outputs,....), chosen from a library and connected to which other through lines.

Matlab

As most of the new algorithms developed in our group were written in MATLAB, and this programming environment has a tool that allows the creation of a graphical interface, this was the first language - environment that was explored during the month of January.

The GUIDE , the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These components have properties, methods, and events. On the layout area may be placed a number of components related to the final graphical effect desired. The tools provide by the GUIDE are shown in the figure 5.

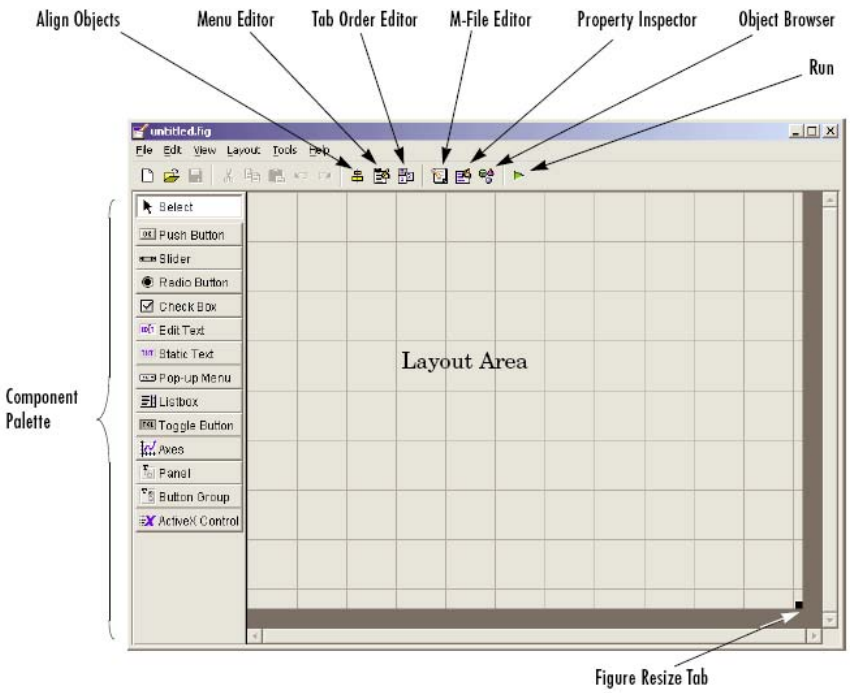

Figure 5: GUIDE

We learned to use and explored these components and finally realized that we needed some extra events, namely related with the desired graphical components properties. These lack of events made the use of MATLAB unsuitable for our purposes. So we started looking for alternatives.

At the same time, object-oriented programming was studied.

C/Borland C++ Builder

With the knowledge of a previous work done by a student member, we started exploring C / C++ and Borland C++ Builder (shown in figure 6) during February and March. In February we traveled to UBI - Universidade da Beira Interior - to meet with this member and to study the potentialities of the development of this project in C++.

Figure 6: Borland C++ Builder

The Borland C++ Builder is a development environment for building applications based on C++ (shown in figure 7).

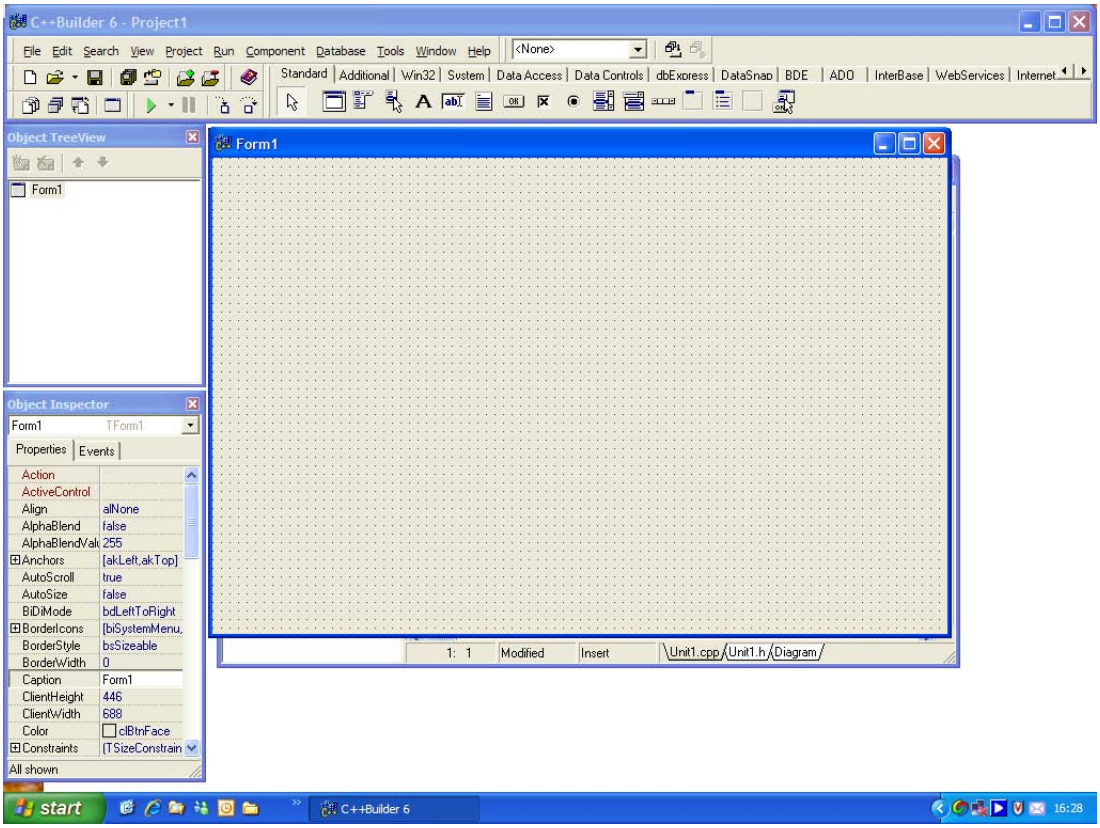

Figure 7: Borland C++ Builder

During the study of this language, we were informed of the existence of a free neural network software - JOONE - whose graphical environment has some desirable characteristics. To avoid repeating software tools, and since JOONE already had some useful functionalities, we studied it until June. JOONE was written in JAVA so learning this computer language became of high priority.

JAVA - JOONE

Joone is a Java framework to build and run artificial intelligence applications based on neural networks. This program consists of a modular architecture based on linkable components that can be extended to build new learning algorithms and neural networks architectures.

Joone applications are built out of components that are pluggable, reusable, and persistent code modules [5]. It aims to build on contributions of many people. The graphical interface of JOONE is shown in figure 8.

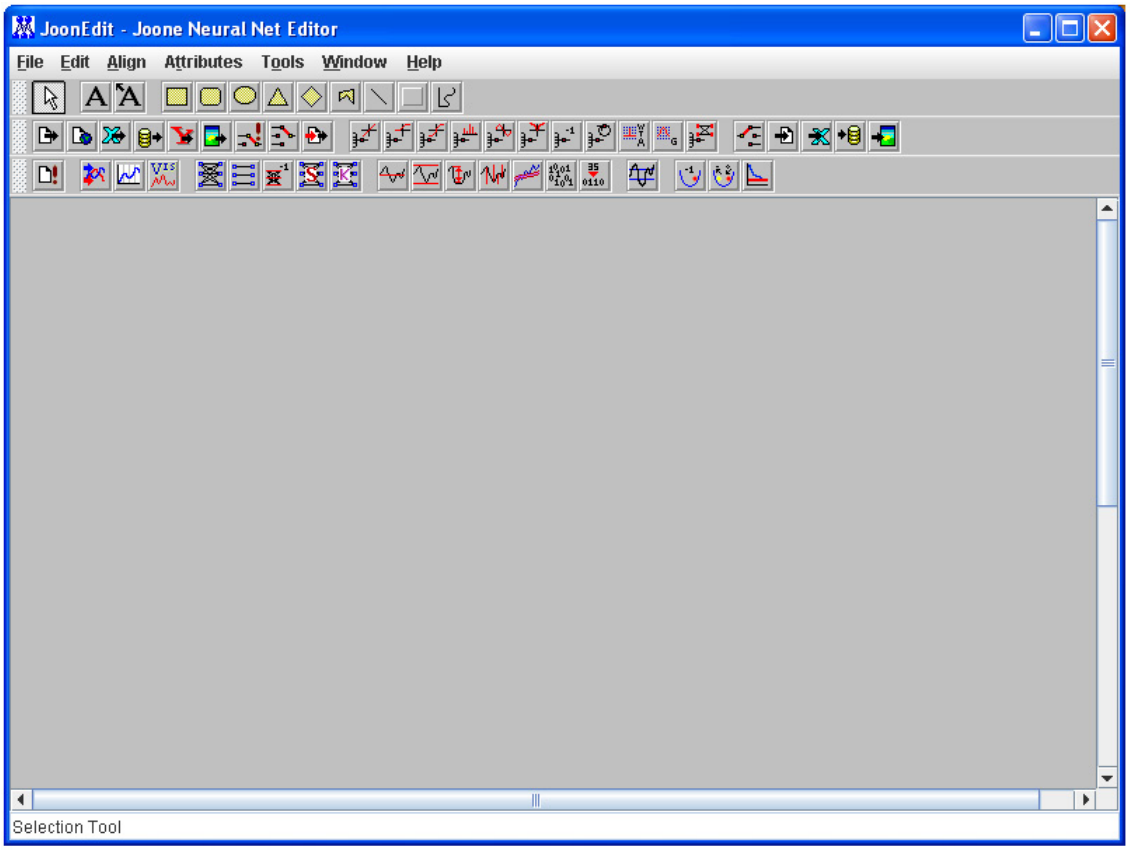

Figure 8: Graphical interface of JOONE

Joone as some features that we explored and are described below.

- Supervised Learning:
  - Feed forward neural networks (FFNN);
  - Recursive neural networks (Elman, Jordan,...);
  - Time delay neural networks (TDNN);

  - Standard backpropagation algorithm (gradient descent, on-line and batch);
  - resilient backpropagation;

- Unsupervised learning:
  - Kohonen SOMs (with WTA or Gaussian output maps);
  - Principal Component Analysis (PCA);

- Modular neural networks (i.e. possibility to mix all the above architectures).

Furthermore JOONE has twelve different activation functions:

- linear;
- biased linear;
- sigmoid;
- hyperbolic tangent;
- logarithmic;
- sine;
- delay;
- context function;
- Gaussian;

and seven built-in data pre-processing mechanisms:

- Normalizer - limiting the input data into a a predefined range (unnormalizer - rescale the output data);
- Center on zero - subtract the average of the input data;
- Delta normalizer - feed a network with the normalized 'delta' values of a time series;
- MinMax - extract the turning points of a time series;
- Moving average - calculate the average values of a time series;

- Shuffler - 'shuffle' the order of the input patterns at each epoch;
- binary - convert the input values to binary format.

Joone was developed with the editor NetBeans (shown in figure 9), so this java IDE (integrated development environment) was also studied carefully.

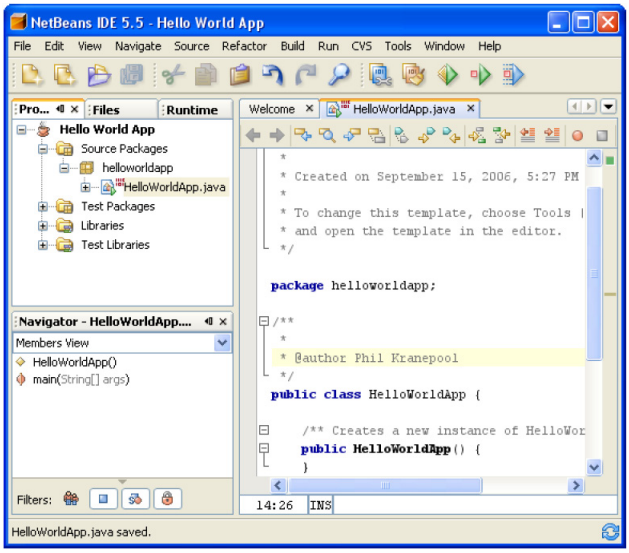

Figure 9: NetBeans IDE

However, during the study of JOONE's structure and experimentation with datasets, we found some problems on compiling it as well as on the clarification of some code. These problems could only be completely solved with the support of the authors of JOONE. We contacted them but the responses where vague and delayed. Also we became aware of several shortcomings of the JOONE structure. So we started looking for other options.

Bibliography

[1] Al Ashi, R. Y.; Al Ameri, Ahmed; *Introduction to Graphical User Interface (GUI) MATLAB 6.5*; UAE University, College of Engineering, Electrical Engineering Department; IEEE UAEU Student Branch

[2] Cortez, P.; *Redes Neuronais Artificiais*, Departamento de Sistemas de Informação, Universidade do Minho

[3] Haykin, S.; *Neural Networks: A comprehensive foundation*; Second Edition; Prentice Hall; 1999

[4] Looney, C. ; *Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists*; Oxford University Press; 1997

[5] Marrone, Paolo; *JOONE - Java Object Oriented Neural Engine - The Complete Guide: All you need to know about Joone*; 3 February 2005

[6] NetBeans; *NetBeans IDE 5.5 Quick Start Guide*

[7] Papadourakis, George; *Introduction To Neural Networks*; Technological Educational Institute of Crete, Department of Applied Informatics and Multimedia, Neural Networks Laboratory

[8] Rojas, R.; Feldman J.; *Neural Networks: A systematic Introduction*; Springer-Verlag, Berlin, New-york; 1996

[9] The Math Works Inc.; *MATLAB: The Language of Technical*

[10] The Math Works Inc.; *Simulink: Dynamic System Simulation for Matlab*