NTUNE – AN EDUCATIONAL NEURAL NETWORK SIMULATOR

Michal Czardybon[1], Lars Nolle[2] and Gerald Schaefer[2]

[1]Silesian University of Technology
Faculty of Automatic Control, Electronics and Computer Science
Wydzial Automatyki, Elektroniki i Informatyki
ul. Akademicka 16, 44-100 Gliwice, Poland

[2]School of Computing and Informatics
The Nottingham Trent University
Burton Street, Nottingham, NG1 4BU, UK
lars.nolle@ntu.ac.uk

KEYWORDS

Artificial Neural Networks, Simulator, Education.

ABSTRACT

This paper describes NTUNE (Nottingham Trent University Neural Environment), a multi-platform neural network simulation tool for feed-forward network architectures, which was especially designed as an educational tool for teaching purposes in higher education. NTUNE can easily be extended and trained networks can be exported to be used in other C++ applications.

INTRODUCTION

When looking for a tool to be used for teaching undergraduates the principles of standard feed-forward neural networks (Picton, 2000), it became eminent that freely available tools such as SNNS (Zell, 1994) were designed for experts, who would apply them to real problems, rather than for students to learn the principles of neural networks. Hence, these tools tend to be very complex, which means that undergraduates would have to spend a long time to learn how to use these tools rather than to learn about neural networks. The aim of this research was to develop an easy to use simulation tool for standard artificial neural networks for teaching purposes in higher education.

Artificial Neuronal Networks

First models of artificial neural networks were introduced in 1943 by Warren McCulloch and Walter Pitts (1943) but it took another 20 years to overcome the problems of their first approach and to find appropriate learning methods. Since then, applications of artificial neural networks have grown rapidly in a wide area of science and industry.

Artificial neural networks simulate the biological structure of neural networks in brains in a simplified way to endow computers with the very considerable abilities of biological neural networks: the ability of learning from examples, pattern classification, generalization and prediction.

Both artificial and biological neural networks consist of a large number of simple processing elements called units or nodes which are connected to other processing elements. The great performance of neural networks stems from this massive parallelism (Rummelhart and McClelland, 1986).

It was was proven by Hornik, et al. (1989) that "*... standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximation any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy ...*"

Hornik also provided evidence that such networks "*... are capable of arbitrarily accurate approximation to a function and its derivatives*" (Hornik, 1991). Therefore, feed-forward networks can be used to approximate a given process by presenting suitable examples to it, provided a deterministic relationship exists between the input parameter and the output parameter. Because of this, the feed-forward network architecture is the most commonly used type of artificial neural networks. Therefore, the network simulator was implemented for this network type.

Feed-forward networks

A feed-forward network can be seen as a black box with defined and directed inputs and outputs (Figure 1).

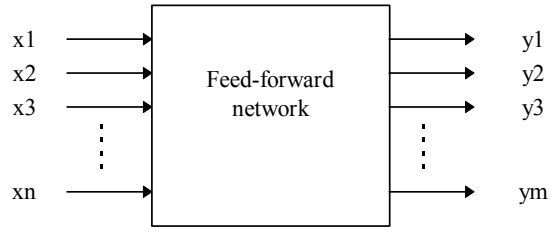


Figure 1 – Feed-forward network as black box.

That means the information is fed through the net in only one direction without any feedback loops. Figure 2 shows a single processing element of a feed-forward network.

Proceedings 19th European Conference on Modelling and Simulation
Yuri Merkuryev, Richard Zobel, Eugène Kerckhoffs © ECMS, 2005
ISBN 1-84233-112-4 (Set) / ISBN 1-84233-113-2 (CD)

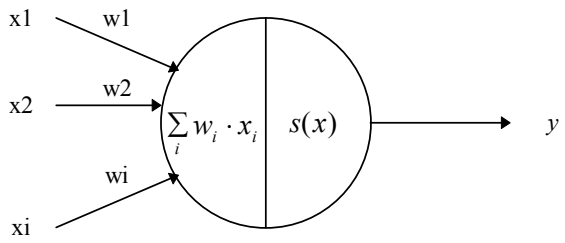


Figure 2 – Processing element.

The output of the processing element is calculated by summing up all inputs multiplied with their associated weight (Equation 1) and passing the sum through a non-linear transfer function, usually a sigmoid or tangent hyperbolic function (Equation 2).

$$x = \sum_i x_i \cdot w_i \qquad (1)$$

$$y = s(x) = \frac{1}{1 + e^{-x}} \qquad (2)$$

The processing elements are arranged in layers, one input layer, one or more hidden layers - so called because they are 'invisible' from the outside - and one output layer. Each output of each unit in one layer is connected usually with every unit in the next layer. Units within the same layer are not connected to each other (Figure 3).

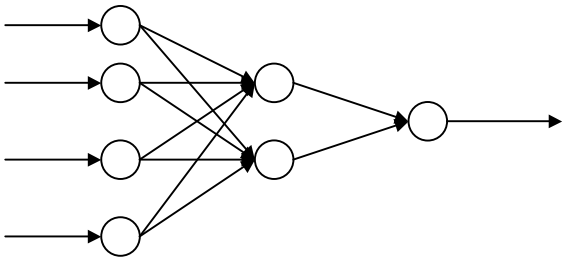


Figure 3 – Feed-forward network with one hidden layer.

The networks work in two modes: the training mode and the testing mode. In training mode, a set of training input patterns together with the demanded output pattern is used to adjust the weights in order to minimize the error of the produced network output and the demanded network output. In test mode, new input patterns can be presented to the network and the network output is used in the application.

Requirements

The following requirements were identified before the development began. The system should have an easy to use graphical user interface. The model error and the correlation between the model output and the expected output should be plotted in real-time. The system should be easily extendible. It should be possible to save trained networks in order to use them in other applications, and it should also be possible to export the plot data so that it could be processed with other software products. Another requirement was that the simulator should be available for different computer platforms, e.g. MS Windows and Linux. Finally, in order to allow the software to be released as an Open Source package under the GNU Public License, no commercial tools or libraries could be used.

NTUNE

NTUNE is a software package consisting of a C++ library and a multi-platform graphical tool for multi-layer feed forward neural networks. The graphical user interface is developed using the wxWidgets library (Smart, 2003), which is available for many different operating systems under the GPL. NTUNE can be used to create and train neural networks in a graphical environment using data provided by the user. The accompanying C++ class library can then be employed to use the trained networks in other programs. Alternatively, the user can use the library to create, train and use feed-forward networks without the graphical tool from within their own programs.

The design of the library is fully object-oriented with much emphasis put on high reuse and extension capabilities. In general, this objective was met by using the highest level of abstraction that was possible using the C++ language. An important goal was to fully separate the learning algorithm from the structure of the network. This was achieved by using the Decorator design pattern (Freeman et. al, 2004) to have a basic computing class, which is dynamically extended with the ability to learn with a particular learning algorithm. The standard error back-propagation algorithm (Rummelhart et al, 1986) is provided with the library, but others can easily be added.

There are several groups of potential users of the package, for example students or researchers who want to experiment with multi-layer feed-forward neural networks, but do not have the time to program it from a scratch. Other potential users are the ones who want to apply neural networks to solve a particular problem. They can make use of the graphical tool to train a network and integrate it into their own programs using the library.

Additionally, NTUNE was designed for learning and studying the error back-propagation learning process, as there is a special visualization feature: a function learned by a network with two inputs and one output can be considered as a gray-scale image. This image can be displayed during the training. Usually it shows emerging and moving regions, which finally form the target image.

Features

NTUNE simulates multi-layer feed-forward neural networks and provides a standard error back-propagation supervised learning algorithm. In the learning phase, the learning rate and the momentum

constant can be adjusted. Also, the error and the correlation for both training and testing sets are plotted in real-time. Once a network is created and trained, it can be exported to be used in other C++ applications. The plot data can also be exported to be further processed. A special feature of NTUNE is a visualization tool for networks with two inputs and one output

Main window

Figure 4 shows the main window after the program is started.

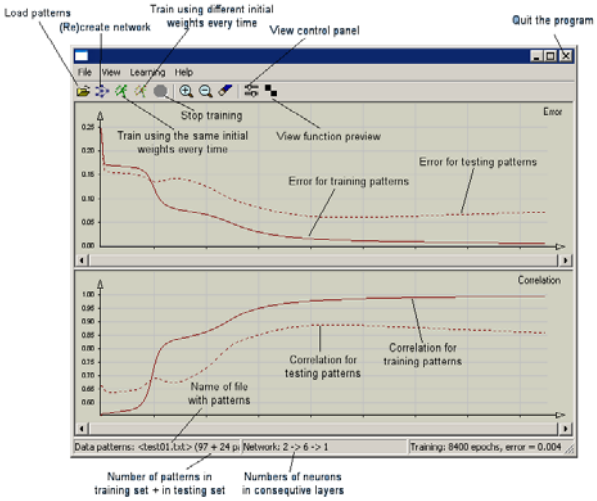


Figure 4 – NTUNE main windows.

There are four main menus, File, View, Learning, and Help. The File menu can be used to load pattern files, to create a new network, and to export the network data or the plot data.

The View menu can be used to zoom in and out, and to clear the plot windows. The update rate for the plot windows can be adjusted, and the Control Panel and the Function Preview Window can be enabled or disabled.

The Learning menu is used to start training the network, initialising it with either the same random values every time a training is started, or with different random values in different training sessions.
The training can also be stopped and a single pattern can be presented to the network.

Finally, the Help menu can be used to access the on-line documentation.

There is a menu bar and icons for easy access to the main functions of the program. The main window is then split in two plot sections. The upper one displays the network error over training cycles in real-time, whereas the bottom one displays the correlation of the network over training cycles in real-time.

Creating and training a network

Before a network can be created, a pattern file needs to be loaded. After loading a patter file, the user needs to specify the percentage of patterns that have to be used as training set, while the remaining ones will form the test set (Figure 5).

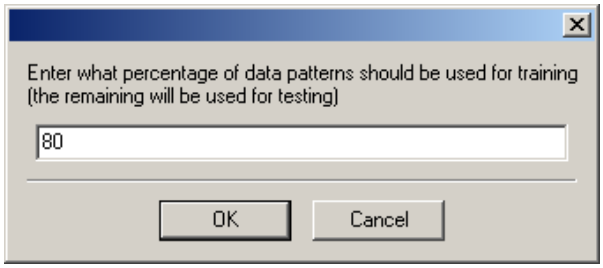


Figure 5 – Dividing the pattern data.

The pattern file defines the number of units in the input layer and in the output layer, but it is also necessary to specify the number of hidden layers and the number of units in each of the hidden layers using the dialog box shown in Figure 6:

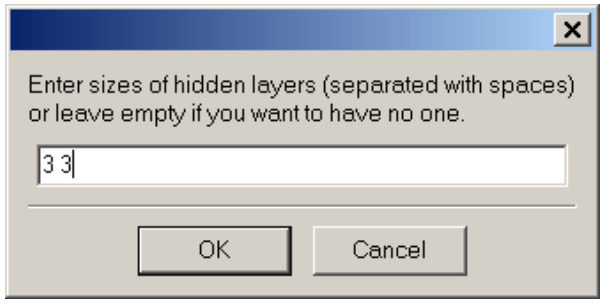


Figure 6 – Entering the number of hidden layers and their numbers of units.

The next step is to select the Learning Rate and the Momentum for the Back-propagation algorithm (Figure 7). The order in which the patterns are used within one cycle can also be defined as fixed sequence, i.e. as they appear in the file (Batch mode), or in random order.

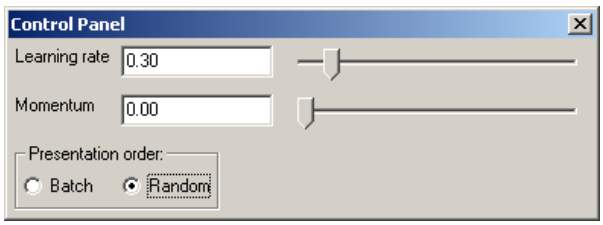


Figure 7 – Control Panel for learning algorithm.

The next figure shows a typical training run for a network (Figure 8). The errors over time for the training set and the test set are displayed on-line in the top graph, while the correlation over time for the training set and the test set are displayed in the bottom graph.

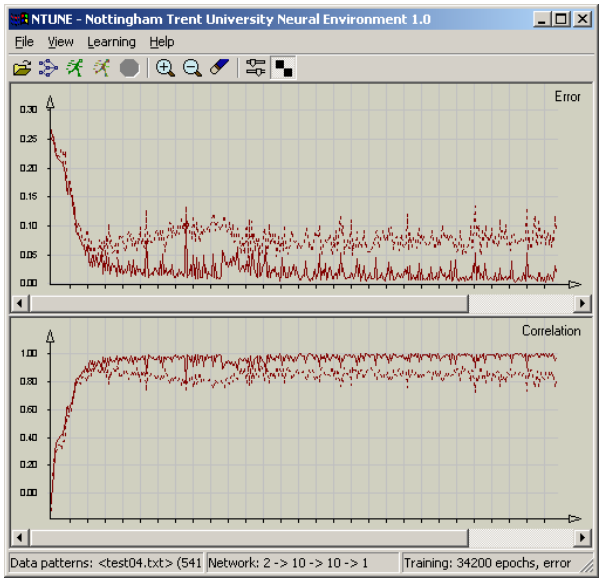


Figure 8 – Typical training run.

A novel feature of NTUNE is the Function Preview Window, which shows in real-time the development of the trained function for networks with a two-dimensional input space and one output neuron. This feature will be described in the next section.

The Function Preview Window

In order to visualize the learning process and to demonstrate the concept of linearly separable functions (Picton, 2000) a Function Preview Window has been developed, which shows in real-time the development of the trained function for networks with a two-dimensional input space and one output neuron.

Once the pattern file has been loaded, the patterns of the training set or the test set are displayed in a two-dimensional grid, and the patterns are displayed either red or green, indicating the associated output values below 0.5 or above (Figure 9).

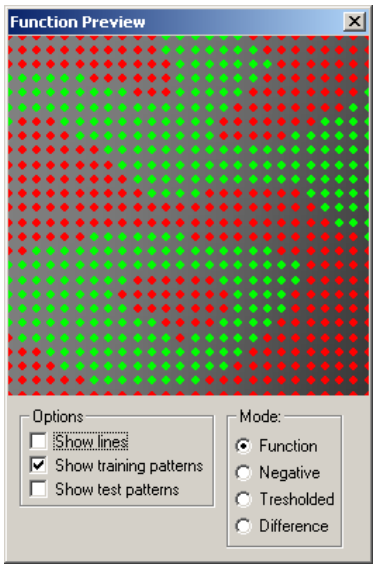


Figure 9 – Function Preview Window with patterns.

As the search progresses, the underlying function that the network has learned is displayed online (Figure 10) where a zero value is displayed as a white pixel and a one value is displayed as a black pixel. Intermediate values are displayed in shades of grey.

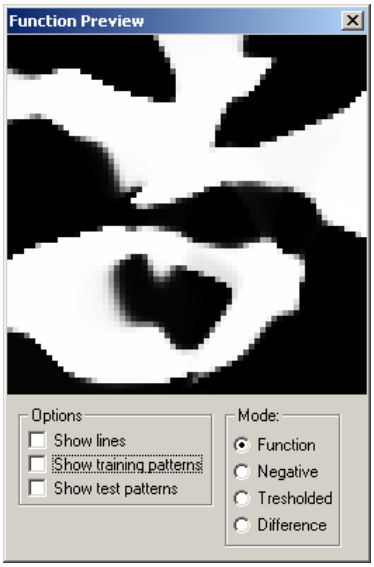


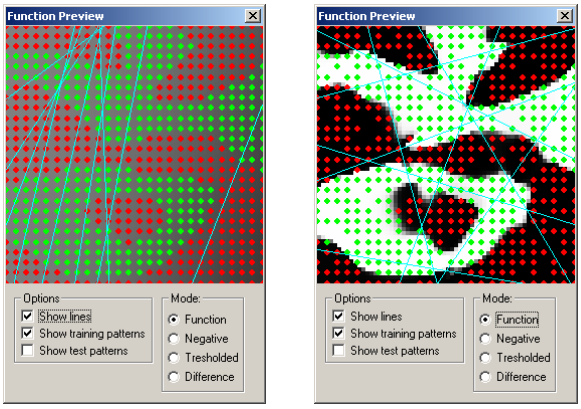
Figure 10 – Underlying function learned by the network.

It is also possible to display the lines corresponding to decision boundaries of neurons in the first layer on-line during training. Decision lines are determined by:

$$w_1 \cdot x + w_2 \cdot y + b = 0 \qquad (3)$$

where w_1 and w_2 are the weights of a neuron and b is the bias of a neuron.

Figure 11a shows the decision lines before training takes place and Figure 11b shows the decision lines after training:



a) Decision lines before training

b) Decision lines after training

Figure 11 – Function Preview Window showing decision lines.

Testing single patterns

Once a network is trained, it can be tested manually with new patters using the option "test by hand" in the learning menu (Figure 12).

Figure 12 – Dialog box for testing a single pattern.

In this dialog box, a single pattern can be entered and the resulting response from the network will be displayed in a message box (Figure 13).

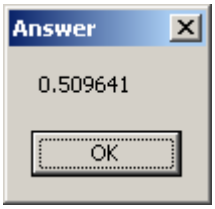


Figure 13 – Resulting answer from the network.

This feature can be used, for example, by students to validate a trained network.

FILE FORMATS

The following section describes the file formats used with NTUNE, the format of the patterns files, the network files, and the plot data files.

Format of patterns files

Patterns files are ASCII text files. At the beginning, there are three numbers: number of inputs for the network, number of outputs of the network, and the number of patterns in the file. Then follow the patterns. Each pattern is described as a sequence of real numbers for consecutive inputs and then the outputs. These numbers need to be from the range [0,1]. Figure 14 shows an example of a pattern file:

```
2 1
10
0.560000 1.000000 0.000000
0.800000 0.120000 0.000000
0.080000 1.000000 0.000000
0.960000 0.200000 1.000000
0.720000 0.000000 0.000000
0.200000 0.600000 1.000000
0.680000 0.400000 1.000000
0.440000 0.640000 1.000000
0.720000 0.920000 0.000000
0.120000 0.600000 1.000000
```

Figure 14 – Example of a NTUNE pattern file.

Note: Some problems may occur under Linux - the end-of-line character probably has to be the Unix-type one and on some distributions the decimal separator character for floating-point numbers has to be comma instead of dot.

Format of network files

Network files are also ASCII text files. At the beginning, there are tree numbers: the number of inputs nodes, the number of outputs and the number of network layers. Then follow descriptions of each layer: each description starts with the number of neurons in that layer. Then follows a list of weights, real numbers corresponding to consecutive weights. Figure 15 shows an example of a network description file.

```
2 1
3
2
-0.264863300744 0.469694440597 0.386450929964
-0.109636118507 -0.804090330799 -
0.147520877600
2
-1.844448806314 0.075209802857 -1.228686078689
-1.647298861078 -1.276881329589 -
0.109270145153
1
0.076063794241 -0.707801538823 0.426283502664
```

Figure 15 – Example of a network description file.

Format of Plot files

Finally, the data representing the network errors and the correlation for the datasets over training cycles can be exported and processed with other applications. Figure 16 shows an example of an NTUNE plot file.

```
---------- Error plots ----------
Plot #0:
   0.252300
   0.257100
   0.251700
   0.252200
   0.250000
   0.250300
Plot #1:
   0.247300
   0.247000
   0.245400
   0.245000
   0.263000
   0.255200
---------- Correlation plots ----------
Plot #0:
   -0.018000
   -0.021800
   -0.009100
   -0.004300
   0.005700
   0.006200
   0.007600
Plot #1:
   0.000000
   0.000900
   0.003600
   0.001800
   0.016700
   0.017000
```

Figure 16 – Example of an NTUNE plot file.

A plot file starts with the data for the error plots, followed by the data for the correlation plots.

CONCUSIONS

NTUNE is an educational neural network tool, which can not only be used for teaching purposes but also for real applications. It is released under the GNU General Public Licence and hence is freely available. Installation files for Linux and Windows are available at http://www.mczard.republika.pl/ntune.html.

AUTHOR BIOGRAPHY

Michal Czardybon studies Computer Science at The Silesian University of Technology and he is expected to finish his Master degree in 2005. He was visiting the Nottingham Trend University one semester as a Socrates/Erasmus student in 2004. His research interests include artificial intelligence and programming languages design.

Lars Nolle graduated from the University of Applied Science and Arts in Hanover in 1995 with a degree in Computer Science and Electronics. After receiving his PhD in Applied Computational Intelligence from The Open University, he worked as a System Engineer for EDS. He returned to The Open University as a Research Fellow in 2000. He joined The Nottingham Trent University as a Senior Lecturer in Computing in February 2002. His research interests include: applied computational intelligence, distributed systems, expert systems, optimisation and control of technical processes

Gerald Schaefer gained his PhD in Computer Vision from the University of East Anglia. He joined the School of Computing and Informaticcs at Nottingham Trent University as a Senior Lecturer in 2001. His research interests include colour image analysis, physics-based vision, image retrieval, medical imaging and artificial intelligence.

REFERENCES

Freeman, E., Freeman, E., Sierra, K., Bates, B. (2004) "Head First Design Patterns", O'Reilly

Hornik, K., Stinchcombe, M., White, H. (1989) "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, Vol. 2, pp 359-366

Hornik, K. (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, Vol. 4, pp 251-257

McCulloch, W.S., Pitts, W. (1949) "A logical calculus of the ideas imminent in nervous activity", Bulletin of Mathematical Biophysics, Vol.5, pp115-133

Picton, P. D. (2000) "Neural Networks", 2nd Edition, Palgrave

Rumelhart, D.E., McClelland, J.L. (1986) "Parallel Distributed Processing", MIT Press, Cambridge, Massachusetts

Rummelhart, D. E., Hinton, G. E., Williams, R. J. (1986) "Learning representations by back-propagating errors", Nature, vol. 323, pp 533-536

Smart, J. (2003) "Presenting xwWindows: a native UI cross-platform application framework", FOSDEM 2003, Brussels

Zell, A. (1994) "Simulation Neuronaler Netze", Addison-Wesley