NeuroSolutions
An Object Oriented Evolution
Jose Principe

What is NeuroSolutions?
NeuroSolutions is an object oriented environment for designing, prototyping, simulating, and deploying artificial neural network (ANN) solutions. This environment is inherently modular, as a neural network breaks down into a fundamental set of neural components. Individually simple, these components combine to form networks that can solve complex problems.

NeuroSolutions supports a practically infinite number of neural models. It provides the user with an icon-based interface from which the neural components are arbitrarily interconnectable.

The concept that ANNs lend themselves naturally to a modular, object oriented implementation is not new, but NeuroSolutions is the first simulation environment to extend the benefits of this structure to the developer. Think of the Neurosolutions GUI as "Neural-CAD." With NeuroSolutions, you design ANNs by connecting neural components on graphical breadboards. Over 90 predefined components are available, and developers can add their own components through C-based and C++-based DLLs.

Some History
The most difficult challenge in designing a modular simulator was to decide how to dissect neural algorithms into appropriate modules. These modules had to be flexible enough to support a wide range of well-known neural models, as well as models that no one had ever tried. In addition, the modules had to be computationally efficient.

A Master’s Thesis supplied the foundation. Published in late 1991, the thesis came from the University of Florida’s Computational Neural Engineering Labor-
atory (CNEL). NeuroDimension Vice President Curt LaFebvre wrote this thesis, and is the principal developer of NeuroSolutions. The thesis proposed a general procedure to implement and train any neural topology belonging to the additive neural model.

The idea was to divide the well-known McCulloch and Pitts neuron into two component-classes, axons and synapses. Simulations fell into three functional planes: activation, sensitivity, and gradient. A controller that implemented a data flow machine orchestrated data-firing through these planes.

This way of organizing the simulations is important. Cutting across the entrenched equation-based simulation of specialized models, this scheme is responsible for our ability to mix neural models and provide design flexibility.

Object Oriented Core
Surprisingly, this modular algorithmic design directly mapped into an Object Oriented Programming (OOP) paradigm. OOP designs have an enormous advantage with respect to simulation speed. Traditionally, interpreted environments are flexible but slow. On the other hand, compiled code is very fast, but it normally restricts the user interface. Object oriented designs combine the advantages of both methods. Object instantiation provides the user with a great deal of flexibility, while the objects themselves still run as compiled code.

Temporal Processing
Like most neural network libraries, our library initially only supported networks trained with a static learning rule (i.e., they could only use information from the current instant in time). This severe shortcoming yielded suboptimal performance on a huge number of applications such as process control, time series analysis and prediction, and classification of non-stationary signals.

In order to extract information over time, we enhanced the library with recurrent topologies and dynamic neural networks (i.e., networks enhanced with memory structures). To extract error information over time, we implemented advanced learning rules (like backpropagation through time). The result was a complete coverage of temporal processing with ANNs. Because these enhancements fit precisely with the rest of the algorithmic structure we developed, we believe that our approach captures the essence of neural computation.

Graphical User Interface
Although extremely powerful, the class library required ANN researchers to be competent C++ programmers. Implementing a neural network was often more a programming task than a design and experimentation task. An easier and more efficient way was possible.

The transformation from the C++ libraries into the visual programming environment of NeuroSolutions came with the concept of associating each instance of a C++ class with a graphical representation.

The process of designing an electronic circuit inspired the “feel” of NeuroSolutions’ user interface. Electronic components such as resistors, capacitors, and transistors, are laid out on a breadboard and wired together to form a circuit. To test the circuit, a designer injects signals and probes responses.

NeuroSolutions provides all the tools required to build a neural “circuit”. A developer lays out neural components (e.g., axons, synapses, and gradient search engines) on a graphical breadboard and connects them together to form a neural network (see Figure 1). Input components inject signals, and probe components visualize the network’s response.
The developer interacts with a component via mouse and keyboard operations. A unique icon represents each component. Drag-and-drop operations move components around the breadboard. The properties of a component are modifiable through graphical controls.

**Probing**

Some critics label neural networks a "black box" technology. The root of this label is a simulation environment that does not provide access to processes inside the network. In addition, many do not know how to interpret internal network variables. The natural progression of NeuroSolutions was to develop an extensive set of probes to take care of the first shortcoming and then to help mitigate the second through human ingenuity.

As with the neural components, NeuroSolutions’ probes are inherently modular. The display format depends on the probe you grab, and the data to be displayed depend on where you drop it. This provides access to all important variables (inputs/outputs, weights, errors, hidden states, gradients, and sensitivities). Static probes display only the current data. Temporal probes buffer the data, visualizing them across time, across space, across frequency, or in state space.

**Windows**

Originally prototyped on — and developed for — the NEXTSTEP environment of the NEXT Computer, the first version of NeuroDimension experienced several months of slow sales. It became apparent that the product had to follow the customers, which meant a port to MS Windows.

To the surprise of NeuroDimension’s engineers, the software development tools under 32-bit Windows (Windows NT and Win95) had in most respects caught up to, and in some respects surpassed, the tools bundled with NEXTSTEP. NEXTSTEP provided a good environment for prototyping an object oriented application. Without it, we could not have developed the Windows version within 9 months, and the structure would not have been nearly as solid nor as flexible as it is now.

Although the object oriented user interface of NeuroSolutions provides great design flexibility for constructing neural networks, this level of access requires a substantial amount of neural network knowledge. We needed a utility to circumvent this requirement and open up NeuroSolutions to a wider audience. This concept quickly evolved into what we call the NeuralWizard.

The NeuralWizard utility is a high-level interface to the network design process. It hides the complexities of the network, streamlining the design process down to an easy, step-by-step procedure. The developer chooses from a list of available models:

- Multilayer Perceptron
- Generalized Feedforward
- Modular
- Jordan/Elman
- Self-Organizing Feature Map
- Principal Component Analysis
- Radial Basis Function
- Time-Lag Recurrent networks

and specifies the input data. From there, the NeuralWizard automatically builds a fully functional neural network, using default parameters if necessary. Most importantly, the developer can probe all internal network data and modify the network’s topology, just as if he or she had built the network from scratch.

**Meeting the Developer’s Needs**

NeuroDimension’s goal is to cater to advanced developers as well as to novices. Researchers must be able to integrate their own algorithms into NeuroSolutions. Application developers have to be able to integrate NeuroSolutions algorithms into their own algorithms. Developers who prototype large networks within NeuroSolutions may want to run them on faster computers. The latest two versions of the product accommodate all of these requirements.

The Professional version generates ANSI-compatible C++ source code for any network designed within the GUI. This allows a simulation prototyped within the GUI to run on a variety of hardware platforms. In addition, NeuroSolutions’ networks are integratable into other applications.

The Developer’s version allows the integration of developer algorithms into NeuroSolutions through dynamic link libraries.
The Paradigms of NeuroSolutions

Multilayer perceptrons (MLPs) are layered feedforward networks trained using gradient backpropagation. These networks are found in many software applications for solving classification, regression, and prediction problems. MLPs are good at solving problems with many input/output relationships. However, they can train slowly, require large amounts of data, and cannot extract information from unstructured inputs.

Generalized feedforward networks extend the MLP concept by creating a cycle network, a connection through which neurons can influence themselves. This forms a recurrent network that has the ability to learn and remember patterns.

Modular feedforward networks are a special class of MLP networks suitable for parallel MLPs to process their input and then learn their parameters. This tends to create some interactivity within the network, which fosters specialization of function in each feature. In contrast to the modular networks, recurrent networks require connections among their layers. These networks require fewer weights than the same-size MLP network, which in turn reduces the amount of training data.

Jordan and Elman networks extend the MLP with context units, processing sequential data that remember past activity. Context units provide the network with the ability to extract temporal information from the data. These networks are typically trained using backpropagation, however, which does not allow data-based memory depth adaptation. Elman networks copy the activity of the first hidden layer to the context units, while the Jordan network copies the output of the network.

Self-organizing feature maps (SOMs) transform input of arbitrary dimensionality into output on two-dimensional surfaces with topological (neighborhood-preserving) constraint. Kohonen unsupervised learning computes the feature maps. The output of the SOM can serve as input to a supervised classification network, such as an MLP. This network's key advantage is the clustering it produces, which reduces the input space to a small number of representative features.

Principal component analysis (PCA) networks also combine unsupervised and supervised training in the same topology. Principal component analysis is an unsupervised feature extraction technique that finds a set of uncorrelated features that best represent the data from the input. Some form of supervised learning is then used to classify the reduced features.

Recurrent backpropagation networks are a hybrid attempt to extend the capabilities of feedforward networks for processing sequential data. In contrast to recurrent networks, these networks use a modified backpropagation algorithm to optimally train the network.

(DLLs). Every GUI component implements a function from the NeuroSolutions Simulation Protocol (NSP). Developers can add components by writing ANSI-compatible C and C++ functions that conform to this protocol.

Looking Ahead

As the developer version becomes more widely used, we expect that more DLL components will appear. NeuroDimension will continually add to its library of components and make these components freely accessible. The real growth should come from developers sharing their components with others by posting them to our archive site on the Internet. We also expect a commercial market of third-party DLL components to emerge. In particular, the availability of signal processing components is a natural fit to help take neural network solutions out of the laboratory and into the real world.

Although the implementation of NeuroSolutions is extremely efficient and PC speed continually increases at a breathtaking pace, developers will always need faster simulations. The code generation feature enables the user to cut the training times down considerably by running the simulations on high-end workstations. In order to get speed increases of two orders of magnitude or more, however, one needs the level of parallel processing found only in dedicated neural network hardware. Accordingly, NeuroDimension's next major milestone is the integration of its software with the fastest parallel processing hardware available for the PC bus.

Jose Principe is a Professor of Electrical Engineering at the University of Florida, Director of the University's Computational Neural Engineering Lab, and Vice President of NeuroDimension, Inc. He is writing a neural network textbook during his one-year sabbatical from the University.