NSL - NEURAL SIMULATION LANGUAGE

Alfredo Weitzenfeld

Departamento Académico de Computación
Instituto Tecnológico Autónomo de México
Rio Hondo #1, San Angel 01000
México D.F., México

NSL, Neural Simulation Language, is a general purpose simulation system providing a high-level language with many constructs and libraries developed to ease the specification of large neural networks. NSL integrates *object-oriented* programming methodologies in its design and implementation, providing a simulation environment for users with little programming background, as well as those with more extensive programming expertise, who can use C++ as an extension to NSL's modeling language. NSL is widely used in research and teaching, having lead to many different neural network models, both in the artificial and biological domains. NSL enables the simulation of models with different levels of neural details, with special support for the *leaky integrator*.

1. INTRODUCTION

In the area of neural network simulation many tools have been built to facilitate the scientist in the task of modeling and simulating neurons at different levels of detail. The more detailed neuronal models, such as the *Hodgkin-Huxley* model (Hodgkin and Huxley, 1952), and the *compartmental* model (Rall, 1959), permit only the modeling of a few neurons at a time, and are supported by simulation systems such as GENESIS or NEURON (De Schutter, 1992). The coarser neural models, such as the *leaky integrator* model (Arbib, 1989), permit the modeling of thousands of neurons, and are supported by simulation systems such as NSL (Weitzenfeld, 1991; Weitzenfeld and Arbib, 1994).

2. NSL SYSTEM

NSL, Neural Simulation Language, is a general purpose simulation system providing a high-level language with many constructs and libraries developed to ease the specification of large neural networks. NSL integrates *object-oriented* programming methodologies (Wegner, 1990) in its design and implementation, providing a simulation environment for users with little programming background, as well as those with more extensive programming expertise, who can use C++ (Stroustrup, 1991) as an extension to NSL's modeling language. NSL is offered as public domain software (*anonymous ftp* from **usc.edu**).

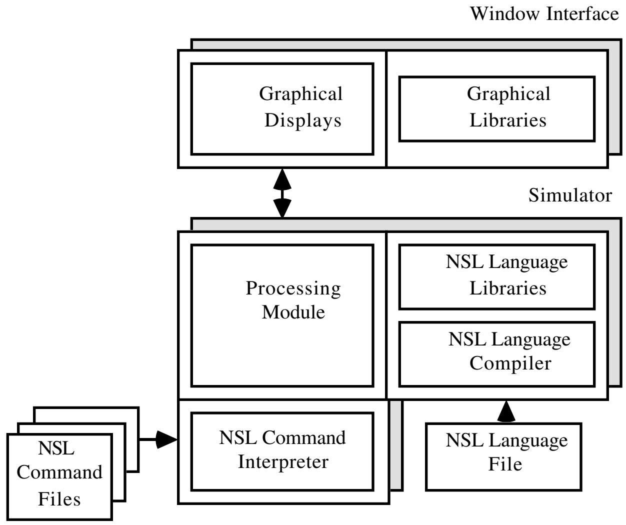

Figure 1. The NSL simulation system is composed of two units: (1) the Simulator, containing the Processing Module, NSL Language Compiler, NSL Language Libraries, and NSL Command Interpreter; (2) the Window Interface, containing the Graphical Displays, and the Graphics Libraries.

The system, whose architecture is shown in Figure 1, includes a command interpreter for interactive and batch processing, an X windows graphical interface, and temporal and spatial displays, including 2D and 3D graphics.

3. NSL LANGUAGE

In order to model neural networks with NSL (NSL Language Compiler shown in Figure 1) it is necessary to describe (1) the neurons making up the network, (2) the neurons' interconnections, and (3) the dynamics of the neurons and their interconnections.

3.1. Neurons

The basic neural model in NSL is the single-compartment neuron, having one output and many inputs, as shown in Figure 2. The internal state of the neuron is described by a single scalar quantity, its membrane potential m, which depends on the neuron's inputs and its past history. The output is described by another single scalar quantity, its firing rate M, and may serve as input to many other neurons, including itself. As the input to a neuron varies, the membrane potential and firing rate also vary.

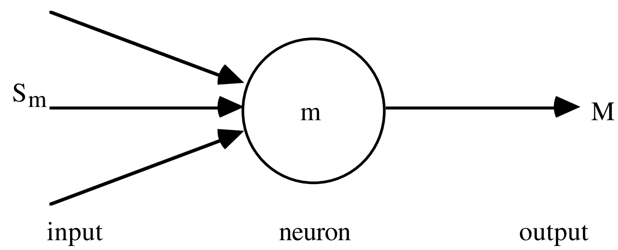

Figure 2. The single-compartment neuron model is represented by one value m corresponding to its membrane potential, and one value M corresponding to its firing rate. S_m represents the set of inputs to the neuron. There is a single output.

The *membrane potential* for m is described by the differential equation

$$\tau_m \frac{dm(t)}{dt} = f(S_m, m, t)$$

which depends on the neuron's input S_m, previous values of m, and time parameter t. τ_m is the time constant. The choice of f defines the particular neural model utilized. In particular the *leaky integrator* model is described by $f(S_m,m,t) = -m(t) + S_m(t)$, or

$$\tau_m \frac{dm(t)}{dt} = -m(t) + S_m(t)$$

The *firing rate* M, the output of the neuron, is obtained by applying a *threshold function* to the neuron's membrane potential,

$$M(t) = \sigma(m(t))$$

where σ is usually a non-linear function. Some of the most common threshold functions, such as *ramp*, *step*, *saturation* and *sigmoidal*, are shown in Figure 3.

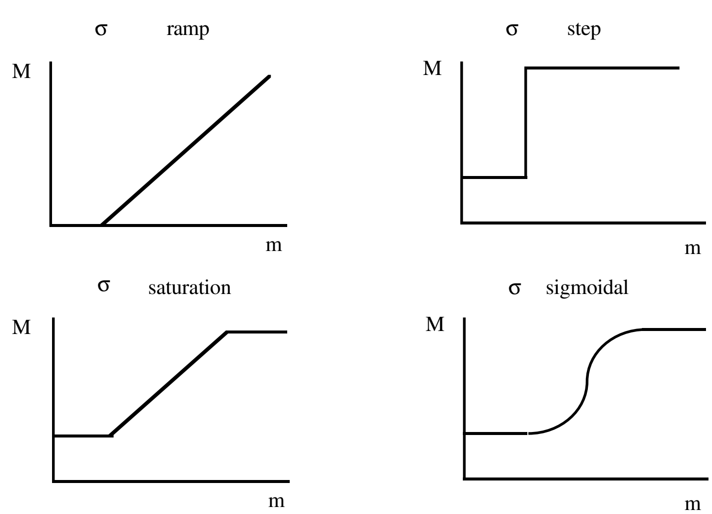

Figure 3. The figures shows some typical threshold functions.

In NSL two **DATA** structures are required to represent a single neuron, one structure corresponds to the membrane potential and the other one to the firing rate. The notation is as follows (with a semicolon at the end of each statement):

```
DATA m;
DATA M;
```

The membrane potential *m* is represented by a differential equation

```
DIFF(m, τm) = f(Sm, m);
```

where **DIFF** defines a first order differential equation for *m* with time decay t_m (time parameter *t* is implicit in the equation). The leaky integrator model corresponds to

```
DIFF(m, τm) = -m + Sm;
```

The firing rate *M* is represented simply by

```
M = σ(m);
```

where σ represents the choice of threshold function.

3.2. Interconnections

When building neural networks, the output of a neuron serves as input to other neurons. Links among neurons carry a connection weight which describes how neurons affect each other. Links are excitatory or inhibitory depending on whether the weight is positive or negative. The most common formula for the input to a neuron *v* is

$$S_v = \sum_{i=1}^{n} w_i M_i(t)$$

where $M_i(t)$ is the firing rate of neuron *m* whose output is connected to the i^{th} input of neuron v, and w_i is the weight on that link.

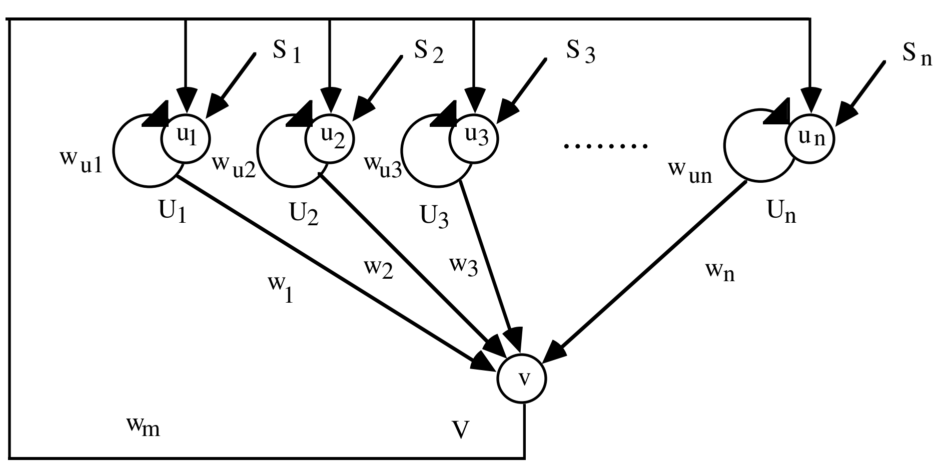

Figure 4. The neural network shown corresponds to the architecture of the Maximum Selector model (see Arbib, 1989), where ui and v represent membrane potentials, Ui and V represent firing rates, Si represent inputs to the network, and wi represent connection weights.

For example, a neural network architecture corresponding to the *Maximum Selector* model (see Arbib 1989) is shown in Figure 4. *u* and *v* represent membrane potential (analogous to *m*), and *U* and *V* represent firing rate (analogous to *M*).

The input to neuron *v* is given by

$S_v = w_1 U_1 + w_2 U_2 + w_2 U_2 + ... + w_n U_n$

while the input to the u_i neuron is (there is *n* such equations)

$S_{ui} = w_m V + w_{ui} U_i + S_i$

In NSL, these expressions describing interconnections among neurons in the neural network are described in a similar way. For example, the input to neuron *v*, represented by S_v, would be the summation of the outputs of all the neurons *u* multiplied by the corresponding connection weights *w*:

```
Sv = w1*U1 + w2*U2 + w3*U3 + .... + wn*Un
```

The input to neuron u_i, is represented by S_{ui} (there is *n* such equations)

```
Sui = wm*V + wui*Ui + Si
```

3.3. Layers and Masks

When modeling thousands of neurons and their interconnections it becomes extremely difficult to name every single one of them. Since in the brain we often find neural networks structured into two-dimensional homogeneous neural layers, with regular connection patterns between various layers, we extend the basic neuron abstraction into neural layers and connection masks.

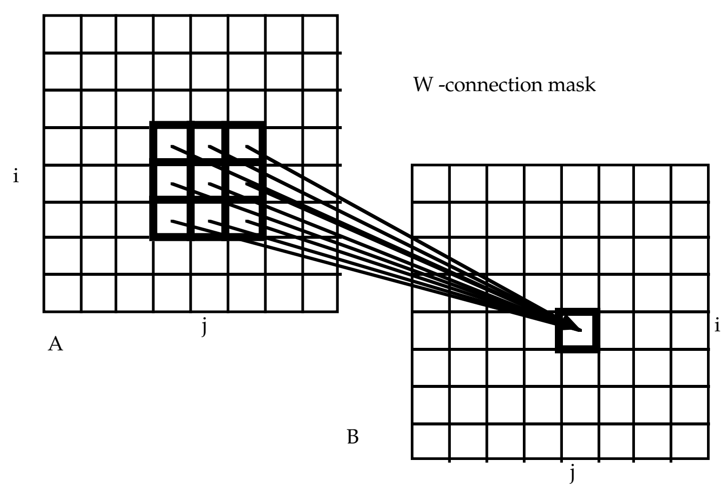

Figure 5. *W* represents the connection or convolution mask between layers *A* and *B* corresponding to the equation *B=W*A*. In this example *W* is a 3x3 mask which is overlapped over a window of *A* to obtain a single value in *B*.

The computational advantage of introducing such concepts when describing a neural network is that neural layers and interconnection masks can then be concisely described as higher level data structures. Instead of describing neurons on a one by one basis, a layer can be described as an array and, similarly, the connections between layers can be described by a mask storing synaptic weights. An interconnection among neurons would then be processed by computing a spatial convolution of a mask and a layer. For example, as shown in Figure 5, if A represents an array of outputs from one layer of neurons, and B represents the array of inputs to another layer, and if the mask $W(k,l)$ (for $-d \leq k,l \leq d$) represents the synaptic weight from the $A(i+k,j+l)$ (for $-d \leq k,l \leq d$) elements to $B(i,j)$ element for each i and j, we then have

$$B = \sum_{k=-d}^{d} \sum_{l=-d}^{d} W(k,l) A(i+k, j+l)$$

which can be described by a simple expression

$$B = W * A$$

In order to support layers and masks, the basic **DATA** structure in NSL is extended with two layers types, **VECTOR** and **MATRIX**, differing according to the number of dimensions they have. To simplify matters, masks, which may have any rectangular shape, are also defined as layers whose values are interpreted in a different way.

For example, the layers of neurons shown in Figure 4 would be described by

```
VECTOR(S,n);
VECTOR(u,n);
VECTOR(U,n);
DATA(v);
DATA(V);
```

3.4. Sample Model

The complete set of equations describing the *Maximum Selector* model, shown in Figure 4, are:

$$\tau_u \frac{du_i(t)}{dt} = -u_i + w_u f(u_i) - w_m g(v) - h_1 + s_i, \; 1 \leq i \leq n$$

$$\tau_v \frac{dv}{dt} = -v + w_n \sum_{i=1}^{n} f(u_i) - h_2$$

where w_{ui} is the connection weight for the self connection of u_i, and $w_{u1} = w_{u2} = \ldots. = w_{un} = w_u$ and $w_1 = w_2 = \ldots. = w_n$ (in this particular model these connection weights are the same), h_1 and h_2 are constants, and the threshold functions are

$$f(u_i) = \begin{cases} 1 & u_i > 0 \\ 0 & u_i \leq 0 \end{cases}, \quad g(s_i) = \begin{cases} s_i & s_i > 0 \\ 0 & s_i \leq 0 \end{cases}$$

In NSL the description of the complete model is arranged into modules, the **INIT_MODULE** containing re-initialization statements, and the **RUN_MODULE** containing statements which are continuously executed as part of the simulation.

The above equations correspond to the following code arranged in two **RUN_MODULE**s:

```
RUN_MODULE(U)
{
   DIFF(u,tu) = - u + wu*U - wm*V - h1 + S;
   U = step(u);
}
```

```
RUN_MODULE(V)
{
   DIFF(v,tv) = - v + SUM(wn*U) - h2;
   V = ramp(v);
}
```

Note that S,u, and U are vector layers and all operations are applied to the layer as a whole. SUM(w_n*U) first multiplies the connection weight w_n by the firing rate U, and the returns a single value corresponding to the vector summation of the expression. This is necessary since v is a single element layer.

4. DISCUSSION

NSL has been successfully utilized as a simulation tool for both biological and artificial neural networks, where various types of applications have been developed, such as the *visuomotor coordination* model (Arbib and Lee, 1993), and the *generation of saccades* model (Dominey and Arbib, 1992). The main challenge in the development of NSL, as well as with other simulation tools, is on one hand to provide a general purpose user-friendly simulation environment, while at the same time being as efficient as possible in the time consuming process of neural network simulation.

As NSL keeps on evolving, it will offer a distributed and parallel framework for the simulation of neural networks (Weitzenfeld and Arbib, 1991) integrating with *schema* models, as described in ASL, Abstract Schema Language (Weitzenfeld, 1993), to enable the development of hierarchical and distributed neural networks, such as needed in robotics applications (Fagg et al., 1992).

5. REFERENCES

Arbib, M.A., 1989, The Metaphorical Brain 2: Neural Networks and Beyond, Wiley.

Arbib, M.A., and Lee, H.B., 1993, Anuran Visuomotor Coordination for Detour Behavior: From Retina to Motor Schemas, in From Animals to Animats 2: Proc. of 2nd International Conference on Simulation of Adaptive Behavior (J.-A. Meyer, H.L. Roitblat, and S. Wilson, Eds), A Bradford Book/MIT Press :42-51.

De Schutter, E., 1992, A Consumer Guide to Neuronal Modeling Software, in Trends in Neuroscience, 15(11):462-464.

Dominey, P.F., and Arbib, M.A., 1992, A Cortico-Subcortical Model for Generation of Spatially Accurate Sequential Saccades, Cerebral Cortex, 2:153-175.

Fagg, A.H., King, I.K., Lewis, M.A., Liaw, J.S., Weitzenfeld, A., 1992, A Neural Network Based Testbed for Modeling Sensorimotor Integration in Robotics Applications, Proc. of IJCNN '92, Baltimore, MD.

Hodgkin, A.L., and Huxley, A.F., 1952, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiology, London, 117:500-544.

Rall, W., 1959, Branching dendritic trees and motoneuron membrane resistivity, Exp. Neurol., 2: 503-532.

Stroustrup, B., 1991, The C++ Programming Language, 2nd. Ed., Addison-Wesley.

Wegner, P., 1990, Concepts and Paradigms of Object-Oriented Programming, SIGPLAN, OOPS Messenger, 1(1):7-87, Aug.

Weitzenfeld, A., 1991, NSL: Neural Simulation Language, Version 2.1, CNE-TR 91-05, University of Southern California, Center for Neural Engineering, Los Angeles, CA.

Weitzenfeld, A., 1993, A Hierarchical Computational Model for Distributed Heterogeneous Systems, TR 93-02, Center for Neural Engineering, University of Southern California, Los Angeles, California, May.

Weitzenfeld, A., 1995, NSL - Neural Simulation Language, in The Handbook of Brain Theory and Neural Networks, Editor M.A. Arbib, Bradford/MIT Press (in press).

Weitzenfeld, A., and Arbib, M., 1991, A Concurrent Object-Oriented Framework for the Simulation of Neural Networks, Proceedings of ECOOP/OOPSLA '90 Workshop on Object-Based Concurrent Programming, SIGPLAN, OOPS Messenger, 2(2):120-124, April.

Weitzenfeld, A., and Arbib, M.A., 1994, NSL Neural Simulation Language, in Neural Network Simulation Environments, Ed. J. Skrzypek, Kluwer.

Weitzenfeld, A., Arbib, M.A., 1996, NSL - Neural Simulation Language: System and Applications, MIT Press (in preparation).