SLONN: A Simulation Language for modeling Of Neural Networks*

DeLiang Wang
Brain Simulation Laboratory
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782, USA

Chochun Hsu
Department of Computer Science and Technology
Peking University, Beijing, P.R.China

This paper presents a general purpose Simulation Language for modeling Of Neural Networks (SLONN) which has been implemented in our laboratory. Based on a new neuron model, SLONN can represent both spatial and temporal summation of a single neuron and synaptic plasticity. By introducing fork *to describe a connection pattern between neurons and by using* repetition connection, module type *and* module array *to specify large networks, SLONN can be used to specify both small and large neural networks effectively. This language is distinguished by its hierarchical organization, which makes it possible to catch very general aspects at higher levels as well as very specific properties at lower levels. As an example to demonstrate some features of SLONN, we have modeled the habituation and sensitization behaviors in* Aplysia.

Keywords: artificial neural networks, connection pattern, empirical neuron model, habituation, leaky integrator model, learning rule, module array, module type, neural modeling, neural networks, simulation language, SLONN.

I. Introduction

During recent years there has been a resurgence of interest in the study of neural networks. This interest has been driven by the desire to reach a better understanding of how the brain works and to develop more intelligent systems. Computer simulation of neural networks has twofold significance. One is on the side of computational neuroscience: neural networks which are carefully constructed to be consistent with neurobiological data are used to form theories of neural structures responsible for certain animal behaviors. For example, much work has been done on visuomotor mechanisms of toad and frog by neural modeling (Arbib and Lara, 1982; Cervantes, *et al.*, 1985). Another is on the side of neural computing: a single neuron is viewed as a computing unit which can be connected to and work in parallel with other units so as to solve certain problems. In this paradigm neural networks do not have to be biologically plausible but must perform effective computations (Rumelhart and McClelland, 1986; McClelland and Rumelhart, 1986; Lippmann, 1987). Although behaviors of neural networks are diverse, three things are crucial: *a neuron model* which serves as a computing unit; *a connection pattern* which connects the computing units to form a specific network; and a *learning rule* which is used for modifying connection weights in order to update network behaviors in a certain way.

In order to facilitate the work of constructing neural networks and carrying out simulation experiments on a computer, *i.e.*, to provide an easy environment for modelers of neural networks, we designed and implemented a high-level, general purpose simulation language, SLONN, for

* *Preparation of this paper was supported in part by grant no. 1R01 NS24926 from the National Institute of Health (Michael A. Arbib, Principal Investigator).*

neural network modeling. Users of SLONN can define specific neuron models and connect neurons together in a very convenient way. With the help of presynaptic connection to a synapse, SLONN provides four basic learning forms: *habituation, sensitization, conditioning* and *facilitation*. Also short-term memory and long-term memory are modeled with a single S-shaped curve.

The remaining part of this paper is organized in this way. Section II provides a brief description of the two neuron models implemented in SLONN. Section III describes the language features of SLONN. Section IV demonstrates some of these features through an example. Finally in Section V, we summarize this project and compare it with some related work and propose the direction of future research.

II. Underlying Neuron Model

In order to meet the needs of both neurobiological modelers and researchers in artificial neural networks, two different types of neuron model are provided in SLONN:

(1) Empirical model. This model is based on typical biological data (Brazier, 1977; Schmidt, 1979; Stein, 1982), and the idea here is to provide a neuron model very close to biological neurons.

(2) Leaky-integrator model. This model takes the assumption of a linear relation between the change of membrane potential and the whole input to the neuron. This model has been widely used in neural network modeling.

Empirical neuron model

Generally, a single neuron model receives N inputs $x_1(t)$, ..., $x_N(t)$ through N corresponding synapses S_1,..., S_N, and each S_i can be connected upon by a presynaptic synapse S_i^* which receives input $x_i^*(t)$, as shown in Fig.1A. The function of a presynaptic synapse depends on the type of the synapse it projects on. For an *fixed synapse* (defined later), presynaptic synapse provides presynaptic inhibition; and for a *memory synapse* (defined later), it modulates the weight of the synapse it projects on. Weights of S_i and S_i^* are denoted as W_i and W_i^* respectively, which range from –1.0 to 1.0. The output and all inputs of the model are binary, but the effect of each input is a graded function of time. In SLONN, states of all neurons are changed synchronously at discrete time. We assume that there is a uniform clock interval, Δ, which is the interval between two consecutive system states.

Unlike other neuron models, an incoming impulse $x_i(t)$ (or $x_i^*(t)$) generates an EPSP (excitatory post-synaptic potential) or IPSP (inhibitory post-synaptic potential) as shown in Fig.1B, depending on the sign of W_i (or W_i^*). The temporal courses for the EPSP and IPSP are defined by functions $f_e(t)$ and $f_i(t)$ which usually last for several Δ's. EPSP and IPSP form the basis for temporal summation over a synapse. The membrane potential $m(t)$ and final output of the model are given by

$$m(t) = \sum_{i=1}^{N} \sum_{r=1}^{c} U_{i,r}(t) + E \qquad (1)$$

$$output(t) = H_t(m(t)) \qquad (2)$$

where E is the resting potential of the neuron model and $U_{i,r}(t)$ is the contribution of input $x_i(t-r\Delta)$ at synapse S_i to the neuron membrane potential, formulated differently in terms of an ordinary synapse and memory synapse. *output(t)* which is in form of firing spikes is generated from a membrane potential by a threshold function H_t which equals 1 when m(t) is larger than or equal to θ (threshold value) and 0 otherwise (a step function). Value *c* is chosen so that the neuron sums up all the EPSP's or IPSP's, whose time courses depend partly on *c*, induced by incoming impulses in the most recent $c\Delta$ time interval.

For any fixed synapse (meaning that synaptic weight does not change), the effect is as might be expected when there is no presynaptic activity:

$$U_{i,r}(t) = \begin{cases} W_i \bullet x_i(t-r\Delta) \bullet f_e(r\Delta) & \text{if } W_i \geq 0,\ W_i^* = 0 \\ -W_i \bullet x_i(t-r\Delta) \bullet f_i(r\Delta) & \text{if } W_i < 0,\ W_i^* = 0 \end{cases} \qquad (3)$$

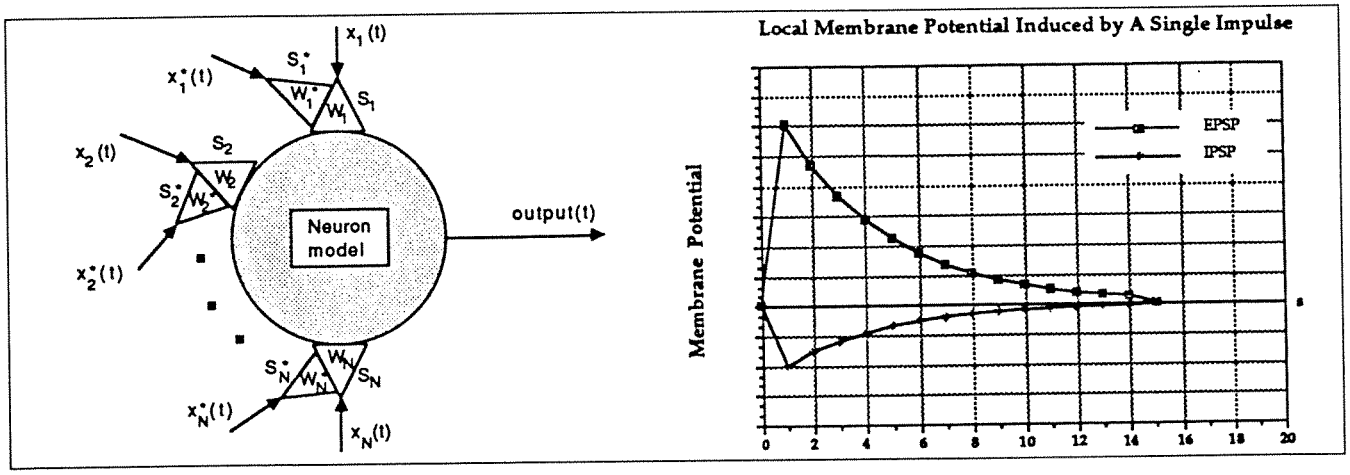

Figure 1. A. Diagram of the empirical neuron model. **B.**Typical curves of an EPSP and an IPSP created by a single incoming impulse.

In the situation that there is a presynaptic projection, we only consider presynaptic *inhibition* since no presynaptic excitation has been observed yet:

$$U_i r(t) = \begin{cases} \text{Max}(0, W_i \bullet x_i(t-r\Delta) \bullet f_e(r\Delta) - W_i^* \bullet x_i^*(t-r\Delta) \bullet f_i(r\Delta)) & \text{if } W_i>0,\ W_i^*<0 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (4)$$

For a memory synapse (meaning that synaptic weight varies over time, recorded as $W_i(t)$):

$$U_i r(t) = \begin{cases} W_i(t-r\Delta) \bullet x_i(t-r\Delta) \bullet f_e(r\Delta) & \text{if } W > 0 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (5)$$

where $W_i(0) = W$, the initial synaptic weight defined by the user when specifying a network. In this case, the role of a presynaptic connection, if any, is to modulate $W_i(t)$ (defined later) and it has no direct contribution to the postsynaptic neuron. Therefore in the above formula there is nothing to say about presynaptic connections.

In the model three curves are introduced for each memory synapse in order to describe the learning rule which is reflected in the varying course of $W_i(t)$. First, learning (acquisition) is described by an S-shaped curve (recorded as $f_S(t)$) which satisfies the following equation,

$$\frac{dy}{dt} = \frac{4k}{W} y - \frac{4k}{W^2} y^2 \quad (6)$$

and $y = y_0$ when $t = t_0$. Parameter k is the slope at the inflection point (where the second-order derivative of the curve equals 0), and W is an initial synaptic weight. Fig. 2A shows a typical curve. Since the S-shaped curve has two varying stages depending on the sign of the second-order derivative and a single turning point: inflection point, this property of the S-shaped curve matches the two forms of memory so that both short-term memory (STM) and long-term memory (LTM) are represented by a single curve and the transfer of short-term memory to long-term memory corresponds to the memory value going beyond the inflection point. This conforms with the view that short-term and long-term memory can be thought externally as the two forms of a common mechanism (Castellucci *et al.*, 1978; Wingfield and Byrnes, 1981). The Ebbinghaus curve (1913), shown in Fig. 2B, for long-term retention of nonsense syllables memory is used in this model with replacement of some constants by parameters (recorded as $f_E(t)$):

$$y = \frac{W \bullet G}{(\log t)^d + G} \quad (7)$$

where G and d are parameters. For the short-term memory, the curve discovered by Peterson and Peterson (1959) for short-term retention of nonsense syllables with numbers, which is shown in Fig. 2C, is used to represent short-term retention in this model. The curve, recorded as $f_P(t)$, is formulated as follows,

$$y = \begin{cases} \frac{W}{2}(1-\sqrt{t/P}) & \text{if } t < P \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where parameter P represents the length of short-term memory. It should be mentioned that $f_E(t)$ and $f_P(t)$ are the experimental memory curves at the behavioral level. We use these two curves as memory models at the synaptic level since no data for synaptic memory are ready to be used.

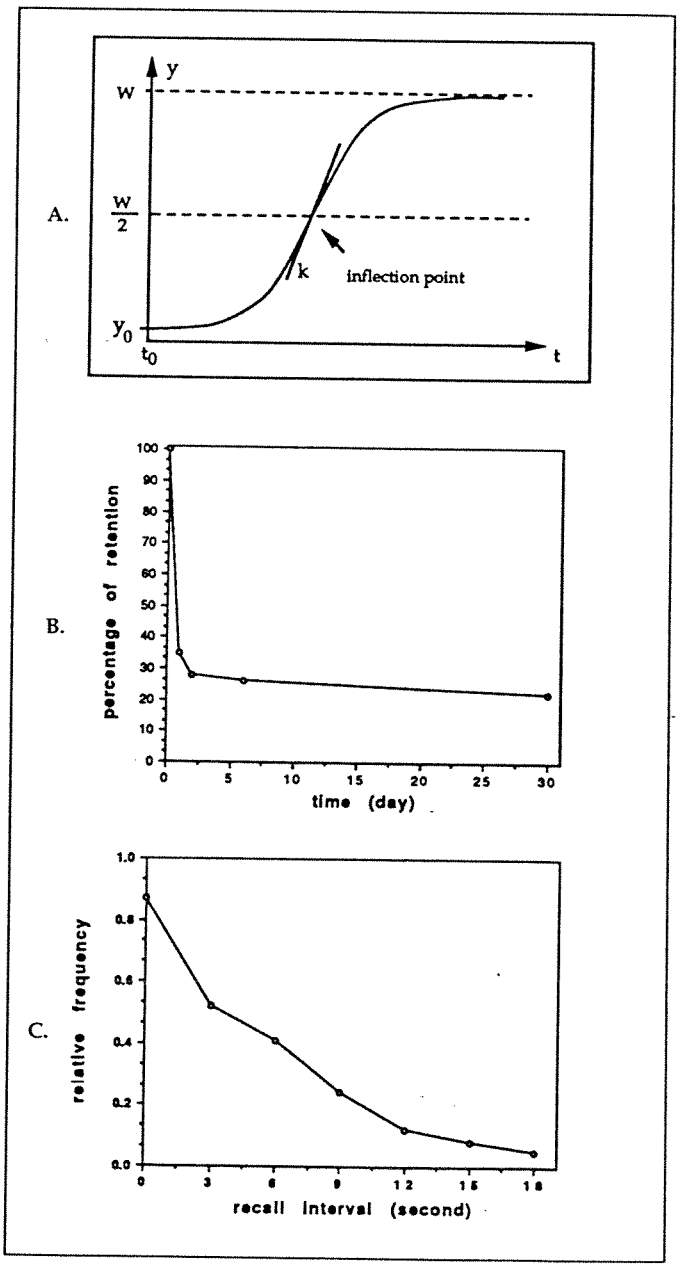

Figure 2. A. Diagram of an S-shaped curve $f_S(t)$. B. The Ebbinghaus curve of long-term memory retention for nonsense syllables (from Ebbinghaus, 1913). C. The curve of short-term memory retention (from Peterson and Peterson, 1959).

In order to describe $W_i(t)$ based on the above three curves, we use a memory function $M_i(t)$ which increases as a learning stimulus $z_i(t)$ is acting upon the synapse S_i (*i.e.*, $z_i(t) = 1$) and decreases a- $z_i(t) = 0$. The decrease of $M_i(t)$ follows different processes in terms of STM or LTM, which are represented by a state variable $L_i(t)$ such that $L_i(t) = 0$ when the synapse is in "training state" (STM) and $L_i(t) = 1$ when the synapse is in "learned state" (LTM). The dynamics of $M_i(t)$ and $L_i(t)$ is given in Appendix. SLONN provides four learning types for each memory synapse which use the same memory curves defined above:

1° Habituation: $W_i(t)$ decreases if $x_i(t) = 1$:

$$W_i(t) = \text{Max}(0, W - M_i(t)) \qquad z_i(t) = x_i(t) \tag{9}$$

2° Sensitization: $W_i(t)$ increases if $x_i^*(t) = 1$.

$$W_i(t) = \text{Min}(1, W + M_i(t)) \qquad z_i(t) = x_i^*(t) \tag{10}$$

3° Conditioning: $W_i(t)$ increases if $x_i(t) = 1 \wedge x_i^*(t) = 1$.

$$W_i(t) = \text{Min}(1, W + M_i(t)) \qquad z_i(t) = x_i(t) \bullet x_i^*(t) \tag{11}$$

4° Facilitation (Hebbian rule, see Hebb 1949): $W_i(t)$ increases if $x_i(t) = 1 \wedge \text{output}(t) = 1$.

$$W_i(t) = \text{Min}(1, W + M_i(t)) \qquad z_i(t) = x_i(t) \bullet \text{output}(t) \tag{12}$$

During learning, the memory value of a synapse raises according to acquisition curve $f_S(t)$. Otherwise the memory value goes down (forgetting) in terms of short-term or long-term memory: once the value of $f_S(t)$ goes above its inflection point, later forgetting will follow the long-term retention curve $f_E(t)$; otherwise it will follow the short-term retention curve $f_P(t)$ (see Appendix for details).

Compared to other neuron models (for example, see McCulloch and Pitts, 1943; Caianiello, 1961), there are two new features in this model. First it takes temporal summation into consideration. Temporal summation is important in temporal information processing. Another feature is the learning mechanism, which will be reflected later in the example. Some other neuronal characteristics, such as synaptic delay, are also modeled. The complete description of this model is given in Wang and Hsu (1988).

Leaky Integrator model

The fundamental equation describing the dynamics of the membrane potential m(t) for a neuron in the leaky integrator model is of the form:

$$mc\frac{dm(t)}{dt} = -Km(t) + I(t) \tag{13}$$

where *mc* and *K* are the membrane constant and I(t) the weighted sum of excitatory and inhibitory inputs. The output of the neuron is formed by (same as formula (2))

$$\text{output}(t) = H_t(m(t))$$

Here the threshold function H_t can take several forms such as a step function or sigmoid function, as chosen by the user. The analytical solution to equation (13) for a constant input *I* is:

$$m(t) = \frac{I}{K} + (m(0) - \frac{I}{K})e^{-Kt/mc} \tag{14}$$

where m(0) is the initial value of m(t) and *I* represents the constant input.

However, the overall input I(t) is generally not a constant and could contain m(t) itself as in feedback situation, so we cannot guarantee an analytical solution. SLONN adopts the following method for an approximate solution (Smith, 1987):

$$m(t+\Delta t) = (1-j)\frac{I(t)}{K} + j \bullet m(t) \qquad \text{with } j = e^{-K\Delta t/mc} \tag{15}$$

where Δt is a parameter with the constraint of being multiple of the system clock interval Δ. That this difference equation does match the analytical solution can be shown by removing the recursion and comparing it to the analytical solution given in (14). A more extensive discussion of this method is given by Smith (1987).

III. Language Description

SLONN is a very high-level non-procedural language implemented in C under the UNIX operating system, and its syntax definition is supported by YACC and LEX software utilities in UNIX (Bell Laboratories, 1983). The use of these standard software utilities, which use a revised Backus-Naur form as format, makes SLONN specification concise and easy to extend.

A SLONN program is composed of four serial parts: *constant part* (optional), *neuron definition part* (optional), *network definition part*, and *execution part*. A user can define in the *constant part* a set of identifiers as constants and assign them values as in conventional languages. The *neuron definition part* is for users to define specific neuron models other than the standard models provided in the system. An entire network is described in the *network definition part*. Finally, the user controls actual simulation runs in the *execution part*.

In terms of network description, any neural network can be specified in SLONN in three steps:

1. Definition of single neurons. Using neuron definition statements, a user can describe specific neuron types with particular properties, such as neuron parameters, and specific shapes of the curves described in Section II.

2. Small network. By introducing the fork statement to describe connection patterns, SLONN can naturally represent both the divergent connection (one neuron makes synapses on many neurons) and the convergent connection (many neurons make synapses on one

neuron), which are the two typical ways of connection in the nervous system.

3. Large network. SLONN provides repetition connection (for), module type (module) and module array for describing large networks.

In what follows, we are not going to provide a full description of the language, but instead attempt to give the reader the flavor of the language and its capabilities through incomplete program examples. Let us note that all system reserved words appear *underlined* in the text.

Neuron Specification

There are two ways to specify a single neuron: (1) by providing a standard neuron type neur. The parameters of the standard neuron are assigned with system defaults. (2) by providing both neuron and leaky statements, which represent *empirical* and *leaky* integrator models respectively, for users to define their desired neuron types. As we have seen before, many aspects of the two neuron models can be adjusted by users. Below is an example (in the remainder of this paper, example programs will appear in a smaller font and indented, and anything enclosed by '/*' and '*/' is ignored by the SLONN parser),

```
neural
        neuron ntype1 {
                theta = 45.0;          /* threshold value */
                tc = 78;
        }
        leaky ntype2 {
                delta_t = 3;           /* Δt in formula (15) */
                Tsigma (1.0, 3.0, 1.7, 0.0);
        }
... ...
neur n1, n2, n3;
ntype1 pear1[8, 8];
ntype2 pear2[8, 8];
```

The above segment defines two neuron types: *ntype1* and *ntype2*, where *ntype1* is the empirical neuron model which has the parameter *theta* equal to 45.0. The time constant *tc* (*c* in Eq.1) of *ntype1*, which determines the length of the EPSP or IPSP, is equal to 78. *ntype2* is a neuron type of the leaky-integrator model with Δt equal to 3Δ and threshold function being *Tsigma* (standing for a sigmoid threshold function) with parameters k1 = 1.0, k2 = 3.0, k3 = 1.7, and k4 = 0.0. The sigmoid function is defined as follows:

$$\text{Tsigma}(x) = \begin{cases} k4 & \text{if } x < k1 \\ k4 + \left[\dfrac{(x-k1)\ (k3-k4)}{k2-k1}\right]^2 \left(3 - \dfrac{2(x-k1)\ (k3-k4)}{k2-k1}\right) & \text{if } x \geq k1 \text{ and } x < k2 \\ k3 & \text{if } x \geq k2 \end{cases}$$

Neurons appearing in the network specification must be declared with a type (either neur or user defined type names) beforehand. Neurons *n1*, *n2* and *n3* are of the standard empirical type; *pear1[8,8]* is an 8x8 matrix of neurons with the type *ntype2*; and *pear2[8,8]* with type *ntype2*. In SLONN a multi-dimensional neuron array (layer) can be specified just like a single neuron.

In the following sections we will leave out detailed properties of single neurons, and view them as abstract nodes in a neural network. We will concentrate on how nodes are connected together to form a network.

Network Connection: fork and for Statements

Two steps are taken to define a neural network. The first step is to define a connection type (*link*) by means of the fork statement, which specifies branches and their weights as well as connection direction. The second step is to associate neuron instances with a connection type to form a concrete neural network. For example,

```
fork 3 (to 0.5, 0.6, 0.7): triple;
triple(n1; n2, n3, n4);
```

The first statement forms a connection pattern *triple* where the number 3 indicates three branches and the reserved word to indicates that the connection direction is from one efferent neuron to many afferent neurons (an opposite direction is indicated by another reserved word from). The second statement indicates that *n1* is an efferent neuron and *n2*, *n3*, *n4* are afferent neurons; the weights of the synapses that *n1* makes on *n2*, *n3*, and *n4* (*0.5, 0.6, 0.7* respectively) are given by *triple*. The network described is shown in Fig. 3A. Type *triple* can be used many times, e.g., *triple(g1; g2, g3, g4)*; will form another network shown in Fig. 3B. The following example shows how to specify memory synapses,

```
neur sen, mov, intn;
fork 1(to <0.45, habit>): tree;
fork 1(to <0.5, sensa>): ftom;
tree (sen, mov);
itom (intn; <sen, mov>);
```

A *value-symbol* pair within a fork statement specifies a memory synapse whose initial weight is designated by the value and learning type by the reserved symbol, while a *symbol-symbol* pair indicates a synapse from the neuron named by the first symbol to the neuron named by the second symbol. In the above example, habit indicates habituation and sensa indicates sensitization (defined in Eqs. 9 and 10). Fig. 3C shows the constructed network.

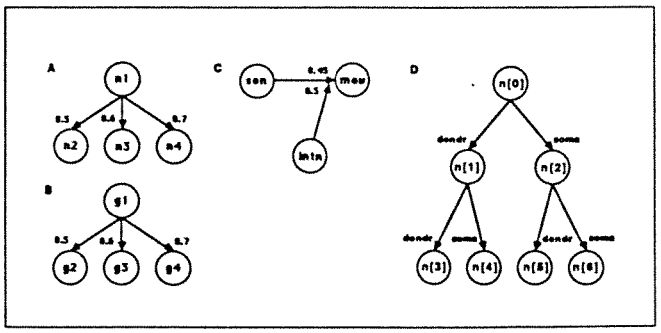

Figure 3. Examples of networks. Here a circle represents a neuron; symbols in a circle indicate neuron names; numbers or constant names (soma, dendr) are synaptic weights.

In order to support iterative connections (particularly useful for layered networks) using the same fork, the

language provides a repetition connection statement for. The for statement can be used in a similar way as in conventional languages. For instance,

```
integer i;
neur n[7];
fork 2(to soma dendr): beta;
i = (0 for 2)
        beta(n[i]; n[2i+1], n[2i+2]);
```

The for statement indicates that the pattern *beta* repeats for 3 times and *i* takes value of 0, 1, 2 in sequel. Identifiers *soma* and *dendr* are the system constants standing for standard *axon-soma* and *axon-dendrite* connections repectively. The resulting network is depicted in Fig. 3D.

Module Type

A module type is used to define a sub-network which appears more than once in the network to be specified. The idea for introducing module type is to establish levels in specifying networks. A module is itself a neural network, but it can also be viewed as a single node at a higher level. A module can be constructed by other modules too. The following example shows the format of module definition.

```
MODULE spade
{
        ... ...
}
spade snet1, snet2, snet3;
```

defines a module type *spade* which is described in braces in the same way as any network. The last statement declares three subnetworks: *snet1*, *snet2* and *snet3* with the type *spade*. Module type can naturally reflect network concepts such as *layers*, *clusters* etc., and elementary regular structures in the animal nervous system such as *cortical columns*. In a certain sense, the module concept is similar to the subroutine structure in conventional languages. A concrete module example is given in the next subsection.

Module Array

A module array is an array (up to 3 dimensions) of modules. Two particular treatments are taken for this module array: (1) The connections between neighboring modules of an array, called *outer connections*, need not be explicitly described. (2) The connections which do not project to any neurons will be cut off from a module array automatically. At most six directions are provided for connecting neighboring modules. The semantics of these directions is self-defined. For a 1-dimensional array, the corresponding outer connected directions are: *right*, *left*; for a 2-dimensional array, the corresponding directions: *right*, *left*, *front*, *hind*; and for a 3-dimensional array, the corresponding directions: *right*, *left*, *front*, *hind*, *above*, *below*. If there are connections beyond neighboring modules, the for statement can always be used to specify connections between any modules in a module array. To demonstate the capability of the module array declaration, an example is given below, and the resulting network is shown in Fig. 4.

```
module star
{
  neur n[9];
  fork 4(to dendr): alpha;  /* This equals fork 4 (to dendr,
                               dendr, dendr, dendr): alpha; */
  fork 4(to soma): beta;
  fork 4(from soma): gamma;
  right
        alpha(n[4]; n[0], n[2], n[5], n[7]);   /* This connects n[3] in one module to n[0], n[2], n[5], and n[7] in its right neighboring module */
  left
        alpha(n[4]; n[1], n[3], n[6], n[8]);
  front
        alpha(n[4]; n[0], n[1], n[5], n[6]);
  hind
        alpha(n[4]; n[2], n[3], n[7], n[8]);
  above
        alpha(n[4]; n[5], n[6], n[7], n[8]);
  below
        alpha(n[4]; n[0], n[1], n[2], n[3]);
  inner                 /* what is defined beneath inner
                           are the connections within a
                           module */
        gamma(n[4]; n[0], n[1], n[2], n[3]);
        beta(n[4]; n[5], n[6], n[7], n[8]);
}
  ..... .....
star network[8, 8, 8];          /* This statement builds a neural network which has 512 modules connected together as shown in Fig.4 */
  ..... .....
```

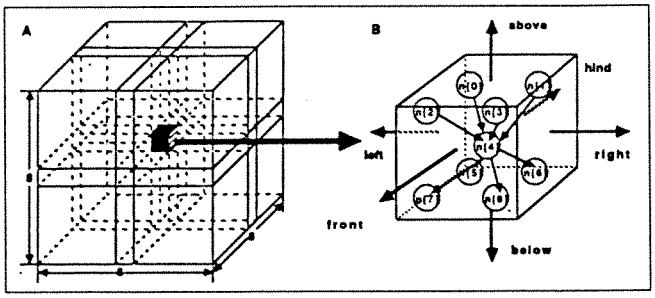

Figure 4. A 3-dimensional module array. **A.** The whole neural network formed by the declaration of the above SLONN program segment. This network consists of 8x8x8 modules connected together. **B.** One typical module which is a subnet composed of 9 neurons, connected with its six neighbors. The connections to this module are omitted for clarity.

Input and Output Description

Inputs to a neural network are represented by impulse trains of sensory neurons, which consist of 0 or 1, indicating impulse presence. Two methods are provided for defining any impulse train:

1. Users can describe input patterns directly as binary trains. For example, *optic = {011}:3* defines a stimulus composed of 3 repetitions of {011}, i.e. {011011011}.

2. Users can define random patterns by means of system functions which make intervals between consecutive impulses satisfying certain stochastic distributions. Random generators, such as *uniform, exponential* (Poisson process), and *normal* distributions, etc., are provided as built-in functions in SLONN.

The defined impulse trains are exerted on sensory nodes through the statement of stimulate, for example:

```
neur skin, head;
stimulate (skin, head <- {001}:3);
```

defines two sensory neurons: *skin, head* which are stimulated by stimulus {011011011}.

Output is reflected by graphic display of activities of neurons, which includes membrane potential and firing pattern. The graphic function of SLONN is implemented in the Suncore utility. Four different display forms are provided: so-called *1-dimensional,* which shows firing states of a set of single cells; *2-dimensional,* which shows firing states of a matrix of cells; *3-dimensional,* which gives a snapshot of activities of a matrix of neurons; and *4-dimensional,* which shows activities of a matrix of cells over a period of time. All these kinds of display are specified in statement display. As an example, the 2-D display in Fig. 5 comes from this statement "display (2; model[8,8])", where 2 indicates 2-D. This item is optional, and 1-D is the default. Figure 5 provides a typical 2-D and 3-D display.

In addition to the display of neuronal activities, a user can also show or change any part of the neural network described, such as synaptic weights, values of memory synapses, and neuronal parameters.

Execution Control

Once a neural network and its input and display have been specified, the user can start simulation process by the statement:

simulate (expr)

where *expr* gives the length of simulation (in steps). The specified network will run for *expr* cycles, within each of which all neurons in the network will be updated once. The output is displayed while the simulation is running.

To meet various modeling requirements, some other control statements have been implemented. For example, a *temporal jump statement* "last (expr)" is designed to facilitate the simulation with long-term memory. By using analytic solutions of memory curves instead of step by step simulation, this statement brings the network into the state *expr* cycles (1 cycle = 1000Δ) later when there is no stimulus. Another example is *serial execution* which allows a series of simulations on a single network. This is done with statement "reset", which sets the network to its initial state (the state before simulation is carried out).

As a summary of the description of the language, Fig. 6 shows the SLONN system organization which illustrates internal processes for executing a user program. One limitation with the current version which can be easily seen from the diagram is the compiled nature of SLONN program. We will discuss this later in Section V.

IV. An Example

The following simple example provides a small system of neurons which has been implemented and run using SLONN. This simulation aims at neural modeling of learning behaviors of habituation and sensitization in *Aplysia*. While our purpose is to illustrate how to use SLONN to develop neural modeling and not to go into detailed physiology of *Aplysia*, it helps to know some experimental results. Through the work by Kandel and his colleagues, we know that when a stimulus is applied to the siphon (sensory skin), *Aplysia's* gill contracts and withdraws

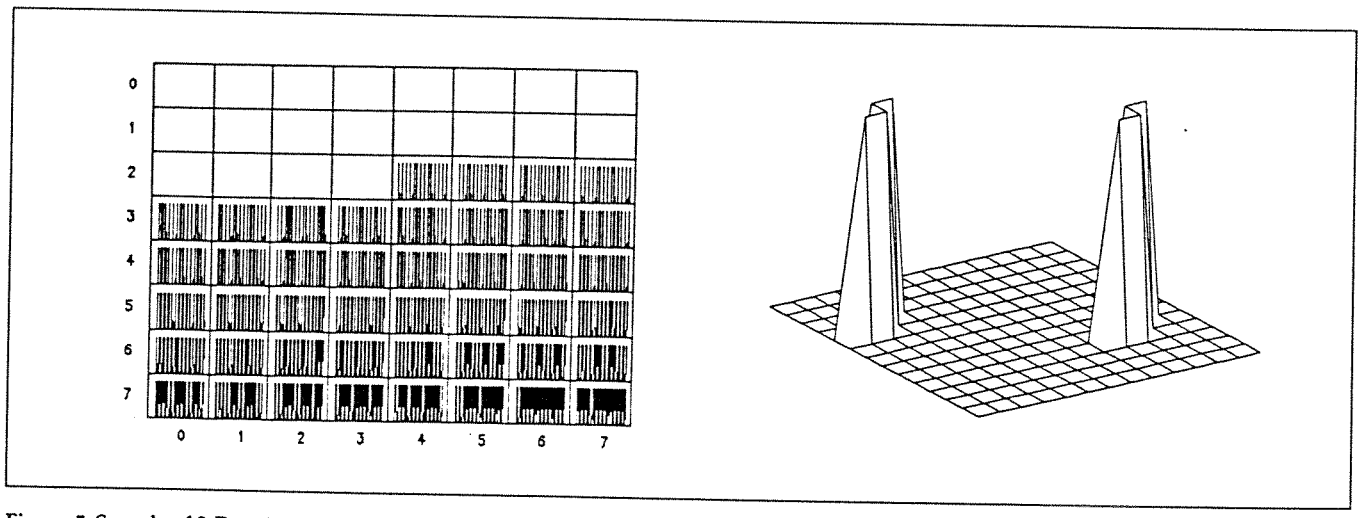

Figure 5. Sample of 2-D and 3-D display from the SLONN system. The display in the left side is a sample output of a 2-D frequency ordering model. Each small box shows the temporal response of a neuron. The display in the right side is a 3-D snapshot of the response of a matrix of neurons to a two-spot stimulus.

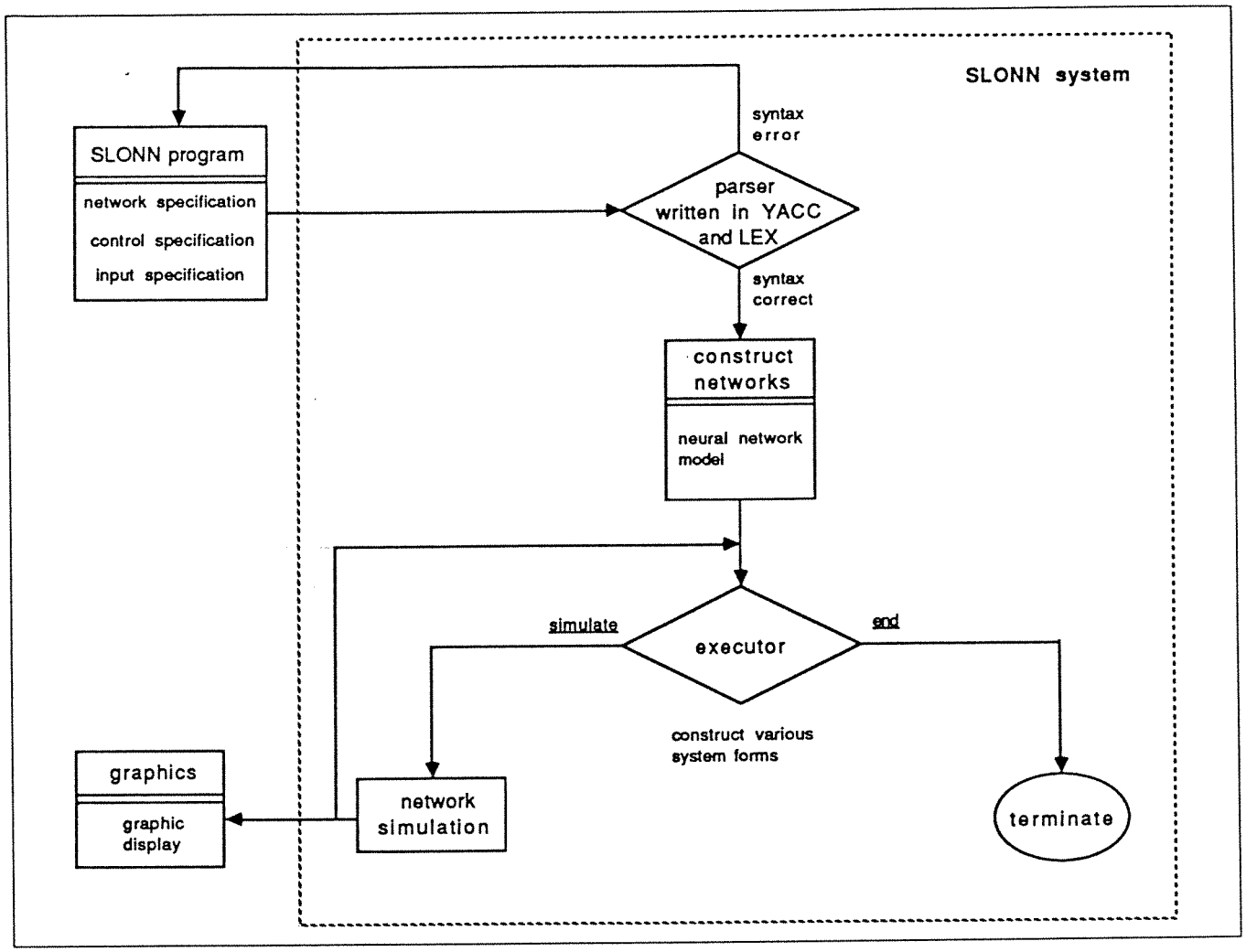

Figure 6. System organization of the SLONN language

into the mantle cavity (Kandel, 1976). This behavior is called the defensive gill-withdrawal reflex. Two simple learning forms can be demonstrated with this reflex: *habituation* and *sensitization*. Habituation is a decrease in the strength of a behavioral response that occurs when an initially novel stimulus is presented repeatedly. Sensitization, however, is the prolonged enhancement of an animal's preexisting response to a stimulus as a result of the presentation of a second noxious stimulus. They found that habituation and sensitization in *Aplysia* give rise to both short- and long-term memory. In the following, we are going to model these learning behaviors. Through a group of simulations we will see that the SLONN model is capable of reproducing learning behaviors in *Aplysia*.

Behavior and Neural Model

According to their findings, after a single training session of from 10 to 15 tactile stimuli to the siphon the withdrawal reflex habituates, but this habituation is short-lived. However four repeated training sessions of only 10 stimuli each produce long-term habituation. They find that short-term habituation involves a transient decrease in synaptic efficacy and long-term habituation produces a more prolonged and profound change (Castellucci, *et al.*, 1978). The data demonstrate that short-term and long-term habituation can share a common locus, namely the synapses the sensory neurons make on the motor neurons.

When an *Aplysia* is presented with a noxious stimulus to the head, the gill-withdrawal reflex response to a repeated stimulus to the siphon is greatly enhanced. According to Castellucci and Kandel (1976), sensitization entails an alteration of the synapses made by the sensory neurons on the motor neurons. The neurons mediating sensitization end near the synaptic terminals of sensory neurons (presynaptic synapse). So the same locus—the presynaptic terminals of sensory neurons—can therefore be regulated in opposite ways by the opposite forms of learning.

We simplify and remould the suggested *Aplysia* neural network underlying the learning behaviors (Kandel, 1979) into the model shown in Figure 7 followed by a segment of SLONN program which specifies the model. The detailed synaptic weights have to be determined during the simulation.

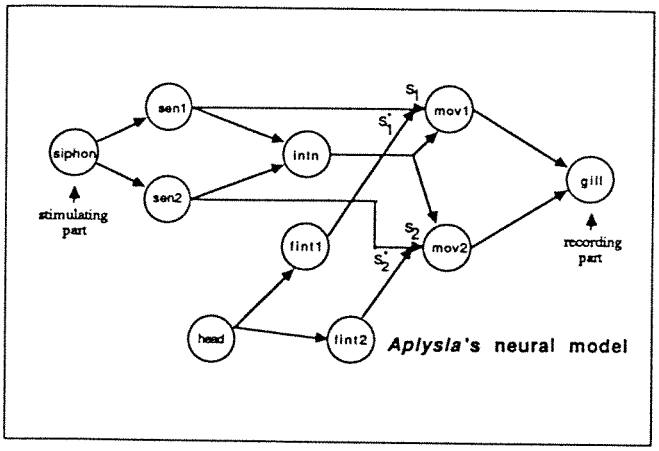

Figure 7. Simplified model of the unilateral neural network of *Aplysia*. Synapses are represented by arrowheads, neurons by circles. The names within a circle indicate the neuron names. Symbols S_1, S_2, S_1* and S_2* are synapse names, and S_1* and S_2* indicate presynaptic synapses.

```
neural                          /* neuron definition part */
  neuron type1 {
        tc = 5;
  }
net                             /* network definition part */
{
  type1 siphon, sen1, sen2, intn;
  type1 fint1, fint2, head, mov1, mov2, gill;

  fork 2(to 0.55): branch;
  fork 1(to <0.45, habit>): tree;
  fork 2(to 0.05): itom;
  fork 2(from 0.38): mtos;
  fork 1(to <0.5, sensa>): ftom;

  branch(siphon; sen1, sen2);
  branch(head; fint1, fint2);
  tree(sen1; mov1);
  tree(sen2; mov2);
  itom(intn; mov1, mov2);
  mtos(gill; mov1, mov2);
  ftom(fint1; <sen1, mov1>);
  ftom(fint2; <sen2, mov2>);
}
```

Simulation Results

The simulation is performed by stimulating sensory cells: *siphon* and *head*, and the response of neuron *gill* represents the gill-withdrawal reflex. We tested the above model by 12 simulations. In the following, we give an exposition of certain experiments, then provide a relevant segment of SLONN code for simulating the experimental result, and finally present the corresponding simulation result. In the simulation, the stimulus for testing the gill-withdrawal reflex, called *touch*, and the training stimulus, called *train*, are described below.

```
begin                           /* beginning of execution part */
        string touch, train;
        touch = {0011100};
        train = {0010111};
```

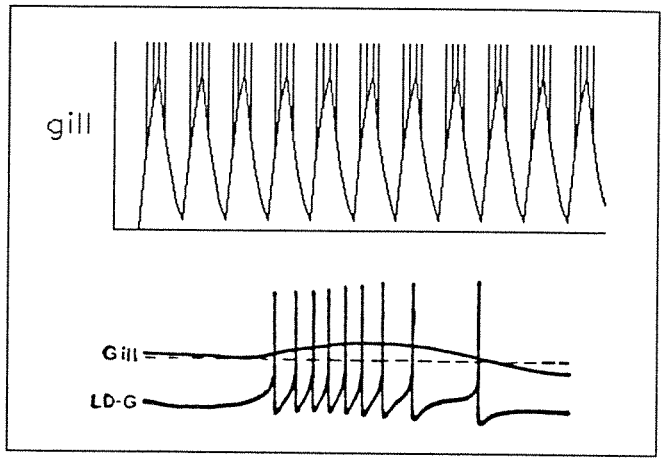

Figure 8. A. Gill-withdrawal reflex. The response of neuron gill when cell *siphon* is stimulated with a period of tactile stimulus. The curve indicates the varying course of membrane potential. When a potential is greater than the threshold the neuron membrane generates an impulse which is represented by a vertical bar. B. Spontaneous gill contractions and simultaneous excitation of the LD-G motor cell (from Kupfermann and Kandel 1969).

The only reason for choosing these stimulus patterns is to make the firing of neuron *gill* easy to be viewed. Therefore we can basically reproduce the following results using a different group of stimulus patterns. The gill-withdrawal reflex was tested first for later comparison.

```
                /* simulation 1: a gill-withdrawal reflex */
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);
```

Fig. 8A shows the simulated gill-withdrawal reflex before any training. At this time the firing frequency of *gill* is 4/7. In Fig. 8B, a typical response of a motor neuron which innervates the gill of *Aplysia*, is provided for giving a sense of what the real response looks like.

As mentioned before, a single training session of from 10 to 15 tactile stimuli habituates the withdrawal reflex. But this is a short-term habituation. The shortest time in which full recovery occurred was 10 minutes, whereas the longest time in which the response was not fully recovered was 122 minutes (Pinsker *et al.*, 1969). The test of this short-term habituation was done with the following two simulations, and the result is shown in Fig. 9. Note that for each execution session initiated by a statement simulate the input and output for the current session is specified by a program segment between the current simulate statement and the previous simulate statement.

```
        /* simulation 2: training for short-term memory */
stimulate(skin <- train: 9; head <- {0});
simulate(65);
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);
        /* simulation 3: test for short-term forgetting */
last(2000);             /* for temporal jump, cf. Section III */
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);
```

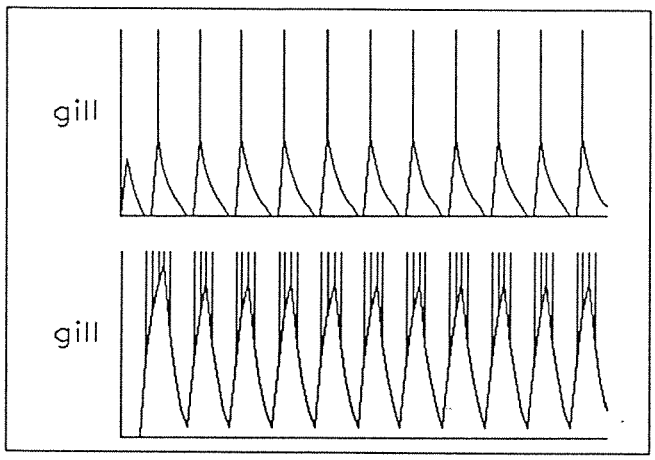

Figure 9. A. The gill-withdrawal reflex after the model is exposed to a period of tactile stimulus (9 trainings). B. Following A, the gill-withdrawal reflex after a silence (no stimulus) for 2,000,000 Δ's (one Δ corresponds to about 2ms).

The response in Fig. 9A, which is the reflex after 9 trainings to *siphon*, decreases from 4/7 to 1/7. Here one training corresponds to one tactile stimulus in the experiment. The habituation is exhibited in this model due to tactile training. In Fig. 9B, we can see that the learned habituation has been "forgotten" after about 60 minutes. This, comparable with real data, shows that the previous habituation is short-lived.

If several (four or more) training sessions are given, habituation is retained for days or weeks (Carew *et al.*, 1972). Fig. 10A provides the average retention curve. The following code is for training a long-term habituation and testing its retention. The corresponding output is shown in Fig. 10B, C, D.

```
/* simulation 4: training for long-term memory */
        stimulate(skin <- train: 36; head <- {0});
/*simulation4:training for long-term memory*/
  simulate(36*7);
  stimulate(skin <- touch: 12; head <- {0});
  display(gill);
  simulate(80);
        /* simulation 5: test for short-term retention */
  last(2000);
  stimulate(skin <- touch: 12; head <- {0});
  display(gill);
  simulate(80);

        /* simulation 6: test for long-term forgetting */
  last(1000000);
  stimulate(skin <- touch: 12; head <- {0});
  display(gill);
  simulate(80);
```

In the simulation on the left, we raised the training amount to 36 trainings, four times as many as before. We see in Fig. 10B, *gill* cannot generate any impulses and the model falls into a profound habituation. In Fig. 10C, *gill* is still inactive after 60 minutes forgetting. Compared to Fig. 9B, this result demonstrates that a long-term habituation occurs. In Fig. 10D, a considerable forgetting occurs after 3 weeks silence, The firing frequency of *gill* (3/7) is close to the pretraining one. This is the result of long-term forgetting, and comparable in general with the experimental data

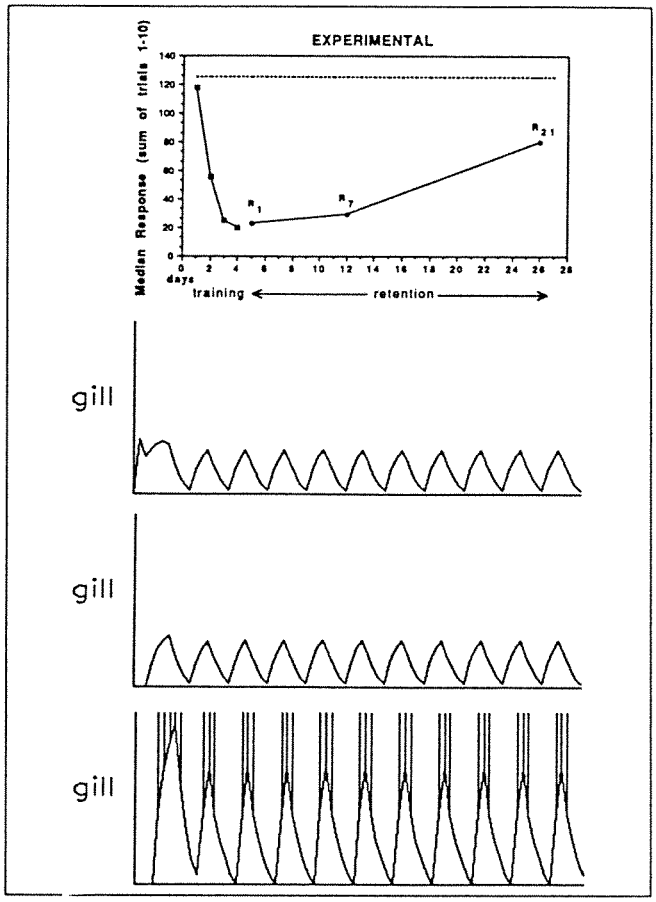

Figure 10. A. Time course of habituation. Habituation within each daily session is expressed as a single score, the sum of ten trials. The following retention is compared with control habituation, the upper dashed line (from Carew *et al.*, 1972). B. The gill-withdrawal reflex after the model is trained with a period of tactile stimulus (40 trainings). C. The gill-withdrawal reflex after a silence for 2,000,000 Δ's. D. Following the previous simulation, the gill-withdrawal reflex after a silence for 10^9 Δ's (about 3 weeks).

given in Fig. 10A. In terms of the above neuron model, the training amount with 36 trainings is sufficient to bring the $f_s(t)$ curves corresponding to synapses S_1 and S_2 in Fig. 7 above their inflection points, i.e. $L(t) = 1$ after this training session.

The reflex response can be abruptly enhanced for many minutes if a single strong sensitizing stimulus is applied to the head, and a long-term sensitization due to repeated stimuli can last for several weeks, similar to long-term habituation (Pinsker *et al.*, 1969). In the following simulation, we only tested for long-term sensitization, and the short-term one can be obtained with less training in the similar way as in the above habituation training. The corresponding code is provided below, and the result is shown in Fig. 11.

```
        /* simulation 7: reset, then test for gill-withdrawal reflex
*/

  reset;
  stimulate(skin <- touch: 12; head <- {0});
  display(gill);
  simulate(80);
      /* simulation 8: training for sensitization */
  stimulate(skin <- touch: {0}; head <- train: 25);
```

```
simulate(25*7);
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);

        /* simulation 9: test for long-term forgetting */
last(1000000);
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);
```

In Fig. 11A, the gill-withdrawal reflex is brought to the pretraining state. Now we can see the effect of sensitization in Fig. 11B due to 25 trainings to neuron *head*, where the firing frequency of *gill* raises from 4/7 to 6/7, a remarkable enhancement. Fig. 11C shows the forgetting of sensitization after 3 week silence, which exhibits a complete restoration of the reflex.

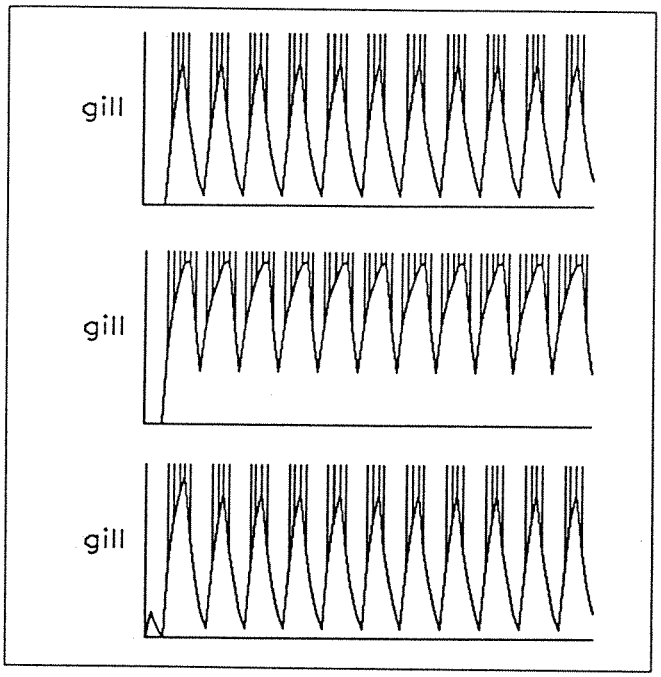

Figure 11. A. The gill-withdrawal reflex after the model is initiated. **B.** The reflex after neuron *head* is stimulated with 25 trainings. **C.** Following the above simulation, the gill-withdrawal reflex after a silence for 3 weeks.

Can sensitization counteract the profound depression in the reflex produced by long-term habituation? This question was examined and the answer is yes. The synapses that were functionally inactivated were restored within an hour by a sensitizing stimulus to the head (Carew *et al.*, 1971). In our final simulations, this phenomenon was tested by the following code, and the result is given in Fig. 12.

```
        /* simulation 10: form habituation */
reset;
stimulate(skin <- train: 36; head <- {0});
simulate(36*7);
```

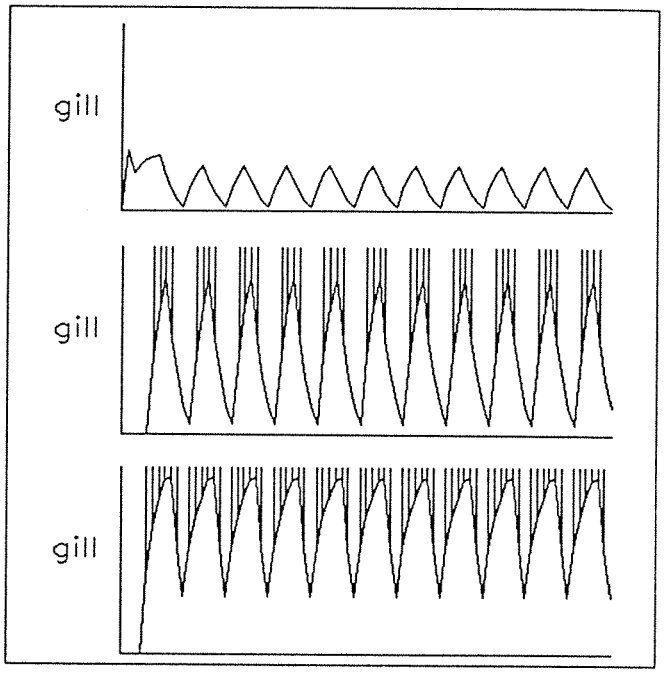

Figure 12. A. The response of neuron gill after the model is initiated and then given the same training as in simulation 4. **B.** Following A, the gill-withdrawal reflex after neuron head is presented with a period of stimulus (16 trainings). **C.** Following B, the gill-withdrawal reflex after head is further stimulated with 23 trainings.

```
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);

        /* simulation 11: training for sensitization */
stimulate(skin <- {0}; head <- train: 16);
simulate(16*7);
stimulate(skin <- touch: 12; head <- {0});
display(gill);
simulate(80);

        /* simulation 12: sensitization, again */
stimulate(skin <- touch: {0}; head <-train: 23);
simulate(23*7);
stimulate(skin <- touch: 12; head <- {0});
display(gill);
        simulate(80);
end                          /* end of execution part */
```

In Fig. 12A, the reflex is trained to be deeply habituated as in the preparation of simulation 4. In Fig. 12B, the response of *gill* returns to the pretraining state due to 16 trainings to the *head* neuron. This simulation shows that sensitization awakes the behavior which has been depressed by habituation. Note that two memory curves ($M_i(t)$), a *major curve* and a *minor curve*, are introduced at one memory synapse to model two learning forms interacting at the same synapse. The result in Fig. 12C demonstrates that sensitization further enhances the reflex after it reverses the depressed behavior.

Table 1. Parameter values of the learning curves.

Name and corresponding formula number	t_0(6)	y_0(6)	G(7)	d(7)	P(8)
Value	10	W/2	$W^2/8$	2.0	100Δ

Table 1 shows the parameter values of various learning curves that we used in this example. In summary, this simulation series demonstrates that the proposed model of *Aplysia* allows us to replicate the following learning properties:

- habituation of gill-withdrawal reflex to a repetitive presentation of a stimulus;
- short-term and long-term habituation are due to different amount of training, and these two stages of habituation share a common locus;
- sensitization (facilitation) of gill-withdrawal reflex due to a noxious stimulus to the head;
- sensitization shares the same locus as with habituation (acting at a presynaptic synapse), and counteracts the profound depression in the reflex produced by long-term habituation.

The reason that we are interested in these simple forms of learning is that they may form a basis for more sophisticated learning processes. Recent studies indicate that the cellular mechanism underlying classical conditioning of the *Aplysia* siphon withdrawal reflex is an extension of the mechanism underlying sensitization (Hawkins *et al.*, 1983). Hawkins and Kandel (1984) made a theoretical effort to construct some more complex learning processes from *habituation* and *sensitization*, and their suggestion for constructing associative learning was later tested by a computational model (Gluck and Thompson, 1987). Without changing any overall language structures, SLONN can be used to reproduce their simulation results by adding some equations they used for their modeling.

V. Discussion

SLONN is developed in a UNIX environment and written in C as well as YACC and LEX. It was first implemented in a Fortune/32 at Peking University, Beijing, and now runs on a Sun workstation in the Brain Simulation Laboratory of the University of Southern California. SLONN has been designed with specification capability as the top criterion. Any kind of connection patterns among neurons (or more generally *units*) can be specified in SLONN effectively.

Through many years of effort in the computer simulation of neural models, quite a few simulation systems for neural networks, and connectionist models, have been proposed to assist developing computer simulations. Earlier representative examples include PABLO (Perkel 1976a; Perkel and Smith 1976; Perkel 1976b), and BOSS (Wittie 1978a, b). PABLO is a simulation program for networks of synaptically interacting neurons which have a variety of physiological properties, such as absolute and relative refractoriness, pacemaker activity, post-inhibitory rebound, and so on. This system is only useful for small neuronal networks, since it is based on a very detailed neuron model and has no abstract specification for constructing a network. BOSS, on the other hand, is a system of routines designed to set up and simulate large models of regular neural structures such as cerebellar cortex, mainly by introducing decomposition schemes and parameterization schemes. These early systems are not general enough to meet requirements of various modelers, for example, no general specification language is provided. Also they lack powerful system supports like graphics.

More recently, some general purpose simulation systems have been developed for connectionist models (*parallel distributed processing systems*). These include P3 (Zipser and Rabin, 1986), Rochester Connectionist Simulator (RCS) (Fanty, 1986; Goddard *et al.*, 1987), and MIRRORS/II (D'Autrechy *et al.*, 1988). Compared to earlier simulation systems, these recent simulators are more general-purpose and environment-like. They are not mainly oriented to neural modeling. For example, they have *units*, *sites* and *links* or *connections* instead of *neurons*, *synapses* and *axons*. The user has to program functions or methods that are used by units to form their outputs. In summary, these simulation systems are more oriented to artificial neural networks. Table 2 summarizes certain features of these recent simulation languages.

SLONN can be characterized as a sort of combination of earlier systems for neural networks and recent systems for connectionist models. It has a high-level, non-procedural network specification language, and therefore is more similar to recent systems in this sense. Although in SLONN we still use terminologies like neurons and synapses, it can be readily used in simulation of connectionist models because of the structural characteristic of SLONN. In fact, in the first implementation of the language only the empirical neuron model was available. Later we easily added the leaky integrator model into the system without destroying any of the original system features. While SLONN is a general purpose language, it also reflects many detailed properties of neurophysiology. In this aspect, it is more comparable to earlier simulation systems designed mainly for neurobiologists, like PABLO. During the system design period, we absorbed many features from other network simulation languages like the SPICE simulation language for electrical circuits (Vladimirescu, 1980) and OCCAM for describing communications between concurrent processes (INMOS, 1983). SLONN is more powerful in describing large networks than those languages because it views a network module as a unit and provides the module array specification.

SLONN achieves both generality and specificity by a key feature of the language: *hierarchy*. We divide the task of developing neural networks into three rather independent levels: *single neurons*, *network modules*, and *networks*. We can change single neuron models while specifications at module level and network level remain unchanged. The same is true for module level and network level. The inclusion of the leaky integrator model demonstrates these characteristics. In implemention, each neuron is represented in the system by an item in a neuron array. Originally each item had only one pointer which points to a structure of parameters that are required for defining the empirical neuron model. Later for adding the leaky integrator model, we added another pointer into each item of the neuron array. This new pointer will point to a structure of parameters necessary for defining a leaky integrator neuron model if the corresponding neuron type is leaky integrator. Then we wrote a C routine to create the output of a neuron according to formula (13). That is basically all we need to do in order to add a new neuron model, and based on the type of a neuron the system will choose a specific routine when updating the neuron. In addition, in SLONN, we separate neuron types vs. concrete neurons and connection patterns vs. concrete networks. This hierarchical structure of the system can provide a very general purpose

specification at the highest level, while it can also capture very detailed properties at the lowest level. Another important advantage of the hierarchical structure is in extensibility. Because each level is relatively independent of the others, new features and functions can be included into the system without updating the whole system. The use of the YACC and LEX utilities also contributes considerably to the extensibility of SLONN, since it is very easy to add new system functions through the interface provided by YACC and LEX.

As mentioned before, the current SLONN system only provides compilation for the sake of speed of execution. The unfortunate result is that the user cannot control network execution interactively. Also, the user cannot add new functions, for instance a new threshold function, into the SLONN system directly. These and some other undesirable aspects of the language will be overcome in the next version of SLONN.

We have long argued about whether it is possible and beneficial to build a single very general purpose and powerful simulation language for neural networks (including connectionist models). Our opinion on this issue is that with hierarchical structure it is possible to develop a very general simulation language without sacrificing much specificity, and we think that SLONN lays a good ground for this effort*.

Acknowledgement

We are indebted to Dr. Michael Arbib for his many insightful comments on this work and earlier drafts of this paper.

To order copies of the software described in this paper, please contact: Alfredo Weitzenfeld, Brain Simulation Laboratory, USC-Computer Science Dept, SAL 200, Los Angeles, CA 90089-0782. E-mail: alfredo@rana.usc.edu. The lab will charge a nominal fee of $50 for tape, user's manual and delivery.

References

Arbib M.A and Lara R. (1982): "A neural model of the interaction of tectal columns in prey-catching behavior," *Biol.Cybern.* 44, 185-196.

Bell Laboratories (1983): *UNIX Programmer's Manual — Volume* 2. Holt, Rinehart and Winston.

Brazier M.A.B. (1977): *Electrical Activity of the Nervous System.* The Williams and Wilkins Company, Baltimore, MD.

Caianiello E.R. (1961): "Outline of a theory of thought-processes and thinking machines," *J.Theoret.Biol.* 2, 204-235.

Carew T.J., Castellucci V.F., and Kandel E.R. (1971): "An analysis of dishabituation and sensitization of the gill-withdrawal reflex in Aplysia," *Int.J.Neurosci.* 2, 79-98.

Carew T.J., Pinsker H.M., and Kandel E.R. (1972): "Long-term habituation of a defensive withdrawal reflex in Aplysia," *Science* 175, 451-454.

Castellucci V.F. and Kandel E.R. (1976): "Presynaptic facilitation as a mechanism for behavioral sensitization in *Aplysia*," *Science* 194, 1176-1178.

Castellucci V.F., Carew T.J., and Kandel E.R.(1978): "Cellular analysis of long-term habituation of the gill-withdrawal reflex of Aplysia California," *Science* 202, 1306-1308.

Cervantes F., Lara R. and Arbib M.A. (1985): "A neural model of interactions sybserving prey-predator descrimination and size preference in Anuran Amphibia," *J.Theoret.Biol. 113*, 117-152.

D'Autrechy C.L. *et al.* (1988): "A general-purpose simulation environment for developing connectionist models," *SIMULATION* 51, 5-19.

Ebbinghaus H.M. (1913): *Memory: A Contribution to Experimental Psychology.* Teachers College Press, New York.

Fanty M. (1986): A connectionist simulator for the BBN Butterfly multiprocessor. Technical Report TR-164. Department of Computer Science, University of Rochester, Rochester, N.Y.

Table 2. A summary of some recent simulation systems

Property	RCS	P3	MIRRORS/II	SLONN
Design purpose	Artificial neural networks	artificial neural networks	artificial neural networks	biological models as well as artificial neural networks
Network specification language	C code	high-level procedural	high-level, non-procedural	high-level, non-procedural
Network specification capabilities	some primitive functions: making units, making links, etc.	"connect" statement; iteration and module functions	node descriptor set descriptor "connects" statement	"fork" statement, iteration, module and module array functions
system hierarchy	none	unit type; subnetwork	set description	neuron type; connection type; subnetwork; subnetwork array
Learning Abilities	user-written functions	user-written functions	some system-defined methods; user-defined functions	four system-defined learning types with parameters, both STM and LTM
Extensibility	programmed user methods	programmed user methods	programmed user methods	hierarchical nature; use of YACC and LEX
Graphics interface	network viewing; graphs of values over time; display of specific units	window oriented; graphs of values over time	window oriented; animates spread of activation over time	display of neurons and matrix of neurons in 1-D, 2-D, 3-D, and 4-D modes
Implementation modes	compilation	interpretation	interpretation	compilation
Implementation language	C	LISP	simulator-LISP; graphics-C	C

Gluck M.A. and Thompson R.F. (1987): "Modeling the neural substrates of associative learning and memory: a computational approach," *Psychol.Rev. 94,* 1-16.

Goddard N., Lynne K.J., and Mintz T. (1987): Rochester connectionist simulator. Technical Report TR-233. Department of Computer Science, University of Rochester, Rochester, N.Y.

Hawkins R.D. *et al.* (1983): "A cellular mechanism of classical conditioning in *Aplysia*: activity-dependent amplification of presynaptic facilitation," *Science* 219, 400-405.

Hebb D.O. (1949): *The Organization of Behavior.* Wiley, New York.

INMOS (1983): Programming Manual in Occam. INMOS Limited, Bristol.

Kandel E.R. (1976): *Cellular Basis of Behavior: an Introduction to Behavioral Neurobiology.* W.H.Freeman and Company, New York.

Kandel E.R.(1979): "Small system of neurons," *Sci. Am.* 241, 67-76.

Kupfermann I. and Kandel E.R. (1969): "Neuronal controls of a behavioral response mediated by the abdominal ganglion of Aplysia," *Science* 164, 847-850.

Lippmann R.P. (1987): "An introduction to computing with neural nets," *IEEE ASSP Magazine,* April 1987.

McClelland J.L. and Rumelhart D.E (eds, 1986): *Parallel Distributed Processing — Volume 2: Psychological and Biological Models.* MIT Press, Cambridge, MA.

McCulloch W.S. and Pitts W. (1943): "A logical calculus of the ideas immanent in nervous activity," *Bullet.Math.Biophysics* 5, 115-133.

Pavlov I.P.(1928): *Lectures on Conditioned Reflexes.* International Publishers, New York.

Perkel D. (1976a): "A computer program for simulating a network of interacting neurons, I. organization and physiological assumptions," *Computers and Biomedical Research* 9, 31-43.

Perkel D. and Smith M. (1976): "A computer program for simulating a network of interacting neurons, II. programming aspects." *Computers and Biomedical Research* 9, 45-66.

Perkel D. (1976b): "A computer program for simulating a network of interacting neurons, III. applications," *Computers and Biomedical Research* 9, 67-74.

Peterson L.R. and Peterson M.J. (1959): "Short-term retention of individual verbal items," *J. Exp. Psychol.* 58, 193-198.

Pinsker H. *et al.* (1969): "Habituation and dishabituation of the gill-withdrawal reflex in *Aplysia*," *Science* 167, 1740-1742.

Rumelhart D.E and McClelland J.L. (eds, 1986): *Parallel Distributed Processing — Volume 1: Foundations.* MIT Press, Cambridge, MA.

Schmidt R.F. *et al.* (1979): *Grundriss der Neurophysiologie.* Springer-Verlag, Berlin.

Smith J.M. (1987): *Mathematical Modeling and Digital Simulation for Engineers and Scientists.* Wiley & Sons, New York.

Stein J.F. (1982): *An Introduction to Neurophysiology.* Blackwell Scientific Publication, London.

Wang D.L. and Hsu C.C. (1988): "A neuron model for computer simulation of neural networks," *Acta Automatica Sinica* 14, 424-430.

Wingfield A. and Byrnes D.L. (1981): *The Psychology of Human Memory.* Academic Press, New York.

Wittie L.D. (1978a): "Large-scale simulation of brain cortices," *SIMULATION* 31, 73-78.

Wittie L.D. (1978b): "Large network models using the brain organization simulation system (BOSS)," *SIMULATION* 31, 117-122.

Vladimirescu A. and Liu S. (1980): "The simulation of MOS integrated circuits using SPICE2." Memo UCS/ERL M80/7, College of Engineering, University of California, Berkeley, CA.

Zipser D. and Rabin D. (1986): "P3: a parallel network simulating system," In *Parallel Distributed Processing — Volume 1: Foundations,* D.E.Rumelhart and J.L.McClelland (eds), MIT Press, Cambridge, MA.

Appendix. Formulation of $M_i(t)$ and $L_i(t)$

Equation (6) has the following analytical solution:

$$y = \frac{W y_0}{y_0 + (W - y_0)\, e^{-(4k/W)(t-t_0)}} \qquad (A.1)$$

Before formulating $M_i(t)$ and $L_i(t)$, the following three transient functions are defined:

$$h_S(y) = f_S(f_S^{-1}(y) + \Delta) \qquad (A.2)$$

$$h_E(y) = f_E(f_E^{-1}(y) + \Delta) \qquad (A.3)$$

$$h_P(y) = f_P(f_P^{-1}(y) + \Delta) \qquad (A.4)$$

where argument y represents the memory value of synapse S_i. The meaning of these three transient functions is depicted in Fig.A1 a,b,c respectively.

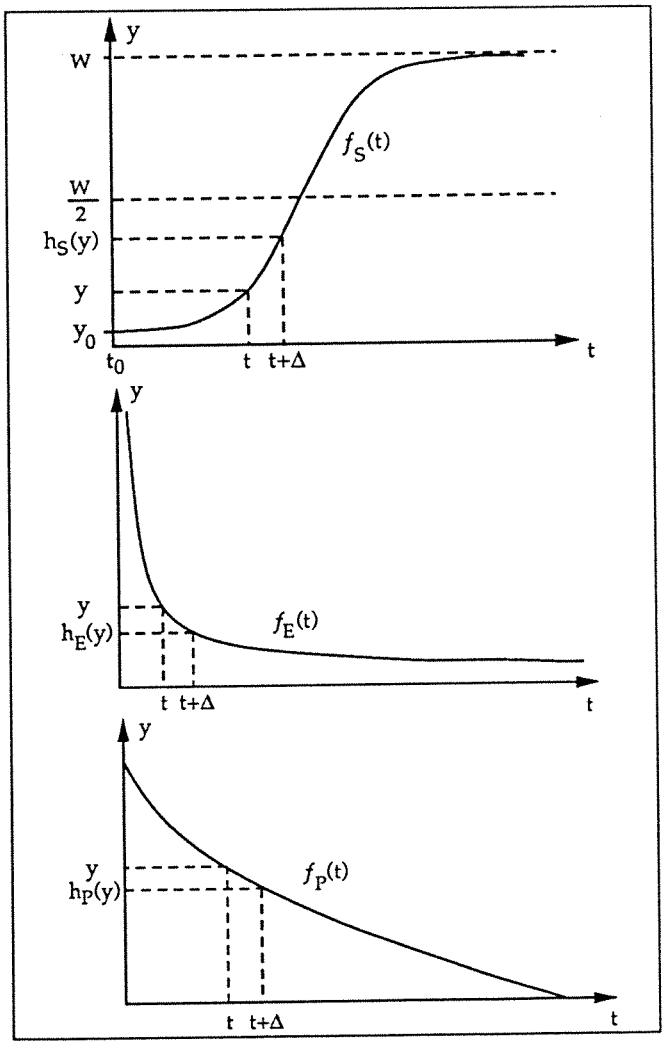

Figure A1 a. The meaning of $h_S(y)$; b. The meaning of $h_E(y)$; c. The meaning of $h_P(y)$.

According to (7), (8) and (A.1), it is straightforward to obtain:

$$h_S(y) = \frac{W}{1+\left(\frac{W}{y}-1\right) e^{\frac{-4\Delta k W^3}{2W^2-1}}} \qquad (A.5)$$

$$h_E(y) = \frac{WG}{\left(\log\left(10^{(G(W-y)/y)^{1/d}}+\Delta\right)\right)^d + G} \qquad (A.6)$$

and

$$hp(y) = \frac{W}{2}\left(1-\sqrt{\left(1-\frac{2y}{W}\right)^2+\frac{\Delta}{P}}\right) \qquad (A.7)$$

Finally we define:

$$M_i(t+\Delta) = \begin{cases} h_S(M_i(t)) & \text{if } z_i(t) = 1 \\ h_E(M_i(t)) & \text{if } z_i(t) = 0,\ L_i(t) = 1 \\ h_P(M_i(t)) & \text{if } z_i(t) = 0,\ L_i(t) = 0 \end{cases} \qquad (A.8)$$

$$L_i(t+\Delta) = \begin{cases} 0 & \text{if } L_i(t) = 0,\ M_i(t+\Delta) \le W/2 \\ 1 & \text{if } L_i(t) = 1 \text{ or } M_i(t+\Delta) > W/2 \end{cases} \qquad (A.9)$$

Value $L_i(t)$ is determined by whether value $M_i(t)$ grows above the inflection point (where $M_i(t) = W/2$) or not, and $L_i(t)$ will not return to the "training state" once it leaves it.

DELIANG WANG was born in Anhui, the People's Republic of China on January 27, 1963. He received the B.S. degree and the M.S. degree in Computer Science from Peking University, Beijing, China, in 1983 and 1986 respectively. From July 1986 to December 1987 he was with the Institute of Computing Technology, Academia Sinica, Beijing, China. He is currently completing a Ph.D. degree in Computer Science at the University of Southern California, Los Angeles, California.

His present research interests include neural mechanisms of visuomotor coordinations, temporal sequence learning using neural networks, visual pattern perception, and neural simulation systems.

CHOCHUN HSU received his B.S. degree in Computational Mathematics from Peking University, Beijing, China in 1957. He is currently a chairman and professor of the Department of Computer Science and Technology, Peking University.

Professor Hsu was a member of the faculty of the Mathematics Department and Electronics Department of the same university, before joining the Department of Computer Science and Technology. He is the coauthor of *Principles of Computer Organization* and *Data Structure* published by the Academic Press in Beijing, and the author or coauthor of numerous technical papers. His current interests are Knowledge Engineering and Information Systems.