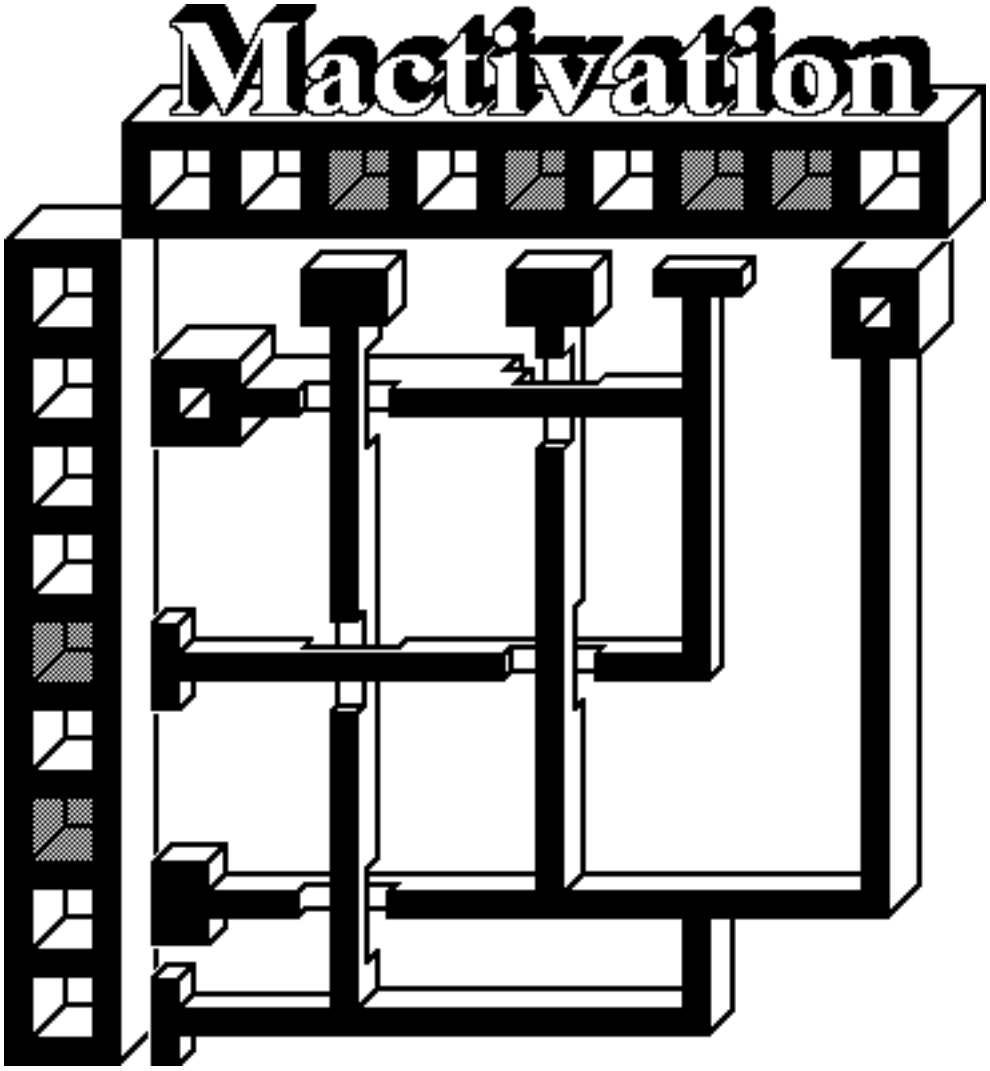
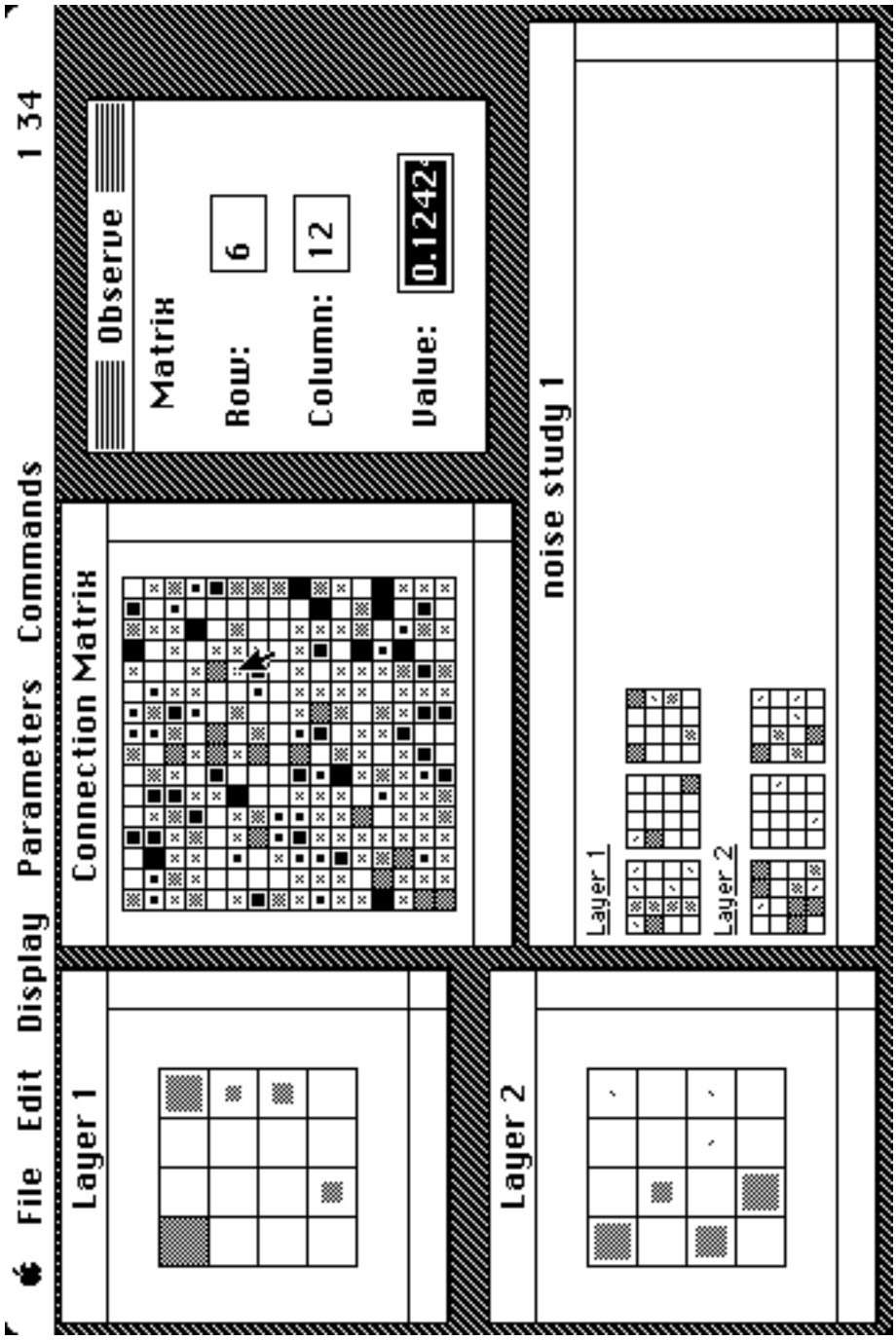


Mactivation™

An Introduction to Connectionist Network Simulation.

User's Manual



Screen 1. After teaching 3 hetero-associations

Mactivation™

by Mike Kranzdorf

Oblio, Inc.
90 East First Street
P.O. Box 1379
Nederland, Colorado 80466-1379
USA
Phone: (303) 258-7994
Fax: (303) 258-7996

mikek@boulder.colorado.edu
{ncar | nbires}!boulder!mikek
AppleLink: oblio

Table of Contents

About Mactivation and its User's Manual…

Mactivation™ is an interactive simulator for investigating the low level concepts of associative memory in a parallel distributed processing (PDP) architecture. A direct interface to units, connections, and network parameters introduces the fundamental principles of all connectionist networks. Functionally, Mactivation performs vector outer products and matrix-vector multiplications. There is access to numerous parameters which modify its behavior during operation. The two basic configurations allow a single layer of units to perform auto-associative memory and two layers to perform hetero-associatively. The delta rule is provided as an example of error correcting operation. Many of the published models of single and double layer networks can be simulated by adjusting the relevant parameters.

This user's manual has several sections. Cell biology is provided for those with an interest in the relationship between connectionist models and real neural networks. This information is not necessary for an understanding of most connectionist systems. It is related to connectionism in a way similar to the relationship of understanding computer architecture and C programming. (Some claim it is essential while others deny any usefulness at all.) Simulated neurons and their networks are described in terms of instantiation and interpretation. Associative memory is Mactivation's forté, so a brief explanation is provided. The role of learning (primarily Hebbian) is included. Vector mathematics and notation common to the literature is outlined. The program reference section describes the windows, menus, and command actions. Mactivation's governing equations are found at the end of this section.

Mactivation 3.3 was written using LightspeedC™ v 2.13 with the 2.15 libraries. It has been recompiled with THINK C™ 4.0. This version appears to run on all Macintosh computers and under all operating system versions. We would appreciate feedback on all aspects of the program. Please send bug reports and suggestions for both the program and the manual to the address on page 4. Everything is subject to change without notice.

MS Word and TEXT:

This manual was written using MS Word, with a copy save as...ed a TEXT document. The MS Word version is much prettier and includes figures. The footnotes are all at the end of the document in the TEXT copy. Sorry about that. MS Word keeps footnotes separate from the document body. Shift-Command-Option-S opens the footnote window.

Neural Network Introduction [1]

Neural networks (also known as connectionist architectures and parallel distributed processing (PDP) systems) are groups of simple, highly interconnected processing units which acts as a whole to perform computable functions. These assemblies exhibit properties of associative memory, recognition, search, learning, and computation. At an abstract level, the units may be simulated biological neurons, and the interconnections their axons and dendrites. The amount and type of biological detail retained in a given model depends on purposes of the model. Examples of biological features which may be preserved by degrees are membrane ionics, bias voltages, interconnection patterns, input/output dynamics, temporal input summation, and long term potentiation. Often the abstracted features are extended beyond what is known in neurobiology, with diverse goals such as ease of simulation, suggestion of principles to seek out in biological preparations, and conceptual computational modeling. Many researchers are not interested in the biological implications, and construct networks purely as problem solving devices. Some representative focuses of simulation work are modeling of brain structures such as the visual cortex, natural language processing and understanding, optimization problems, high density routing tables, and memory systems.

Biological neurons and networks[2]

Neural cells, or *neurons* come in many shapes, sizes, and configurations. A representative neuron consists of a cell body or *soma*, a number of input fibers or *dendrites*, and an output *axon* fiber which can bifurcate many times [Figure 1]. All or any of the constituents may be referred to as a *process*. Cells are connected to each other through *synapses*, which are electro-chemical junctions usually occurring between axons and dendrites.

---

[1]Note: this is a very brief and simple overview of the salient features of neural network modeling. For more complete information, I suggest the PDP series from UCSD, namely:

Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations (1986), D.E Rumelhart, J.L. McClelland, & the PDP Research Group , MIT Press/Bradford Books, Cambridge, Ma.

and

Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models (1986), J.L. McClelland, D.E Rumelhart, & the PDP Research Group , MIT Press/Bradford Books, Cambridge, Ma.

and

Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises(1988), J.L. McClelland, D.E Rumelhart , MIT Press/Bradford Books, Cambridge, Ma. This book comes with simulator software written in C.

[2]For a more thorough explanation of cell function, see for example:

Kuffler, S. W., Nicholls, J. G., and Martin, A. R. (1984) From Neuron To Brain, Sinauer Associates, Sunderland, MA.

Information is carried in electrical signals which flow in only one direction. Electrical activity, say from a dendrite attached to a sensory transducer, flows down the dendrite and into the cell body. This may cause the cell to generate a burst of electrical activity (at the *axon hillock*) which is transmitted outward along its axon. The axon branches and becomes the input to other processes. The details of these events are very complex and interrelated (and are active areas of research).

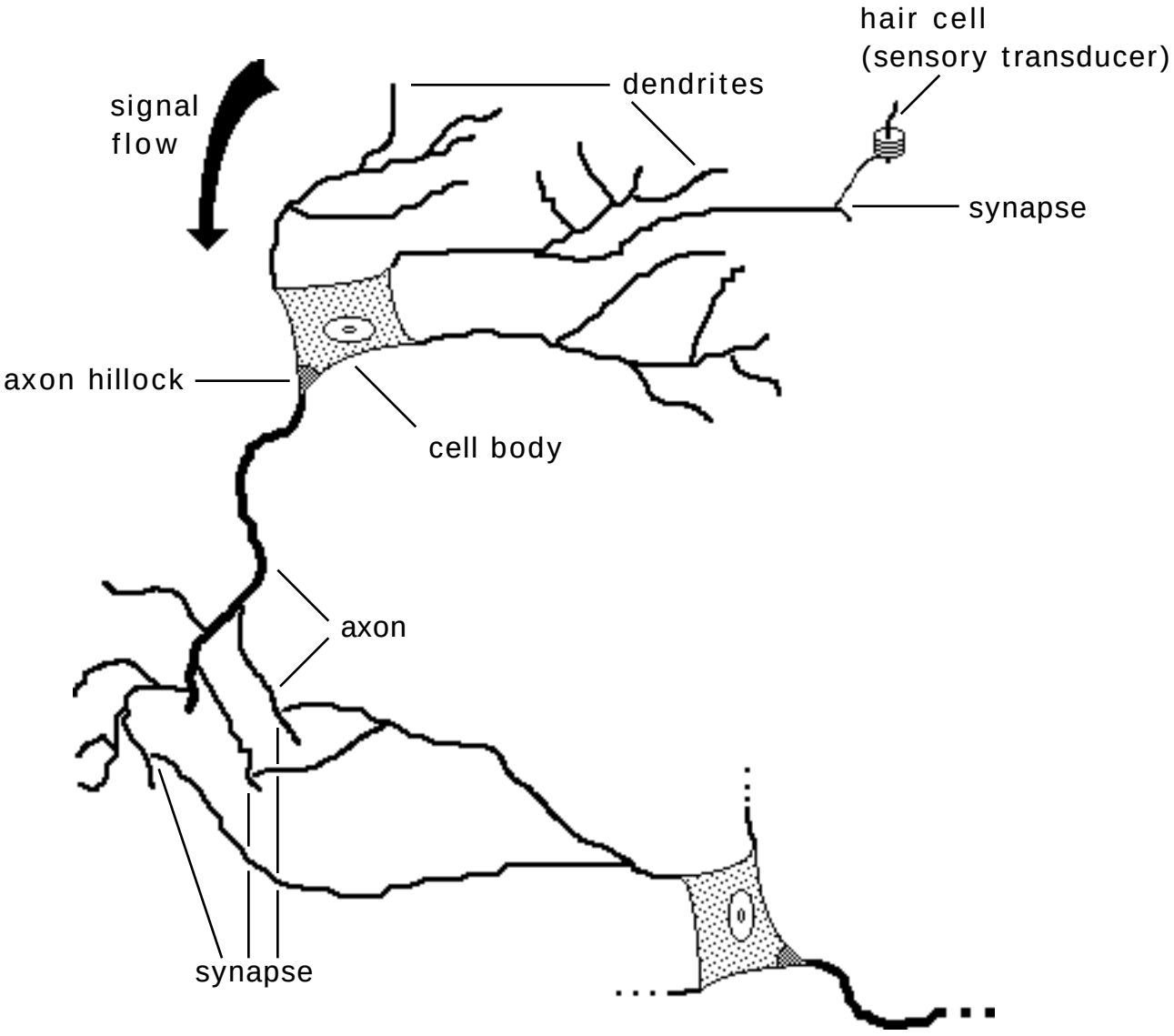


Figure 1. Representative biological neurons

The entire neuron is enclosed in a semipermeable membrane made up of hydrocarbons, phosphates, and proteins. Electrical current passes through these membranes in the form of various ions. (Ions are elements which have excess positive or negative charge.) At rest, the concentrations of these ions inside the cell are different from those outside the cell membrane. There are large negatively charged ions within the cell that can never escape through the membrane which set up this condition. This produces chemical gradients across the membrane down which the chemicals want to flow. The fact that the chemicals are mostly ionic makes electrical gradients as well down which current wants to flow. These may be in the same or opposite directions. For each chemical, the ionic and chemical gradients balance out, influenced by the other chemicals present which are doing the same thing. The result is a *resting potential* of the cell which is lower than that of the surrounding fluid (about -70 millivolts for mammalian neurons). There is a much higher concentration of sodium (Na^+) outside the cell than inside, which contributes to much of this potential.

Neurons are connected to each other through formations called synapses. (The word synapse is used as a noun and a verb.) Most commonly, an axonal branch from one neuron synapses on a dendrite of another neuron. There is usually no physical contact between the two membranes. They are separated by a distance of about 20 nanometers called the synaptic cleft. When the current being carried in an axon arrives at a synapse, certain chemicals known as *neurotransmitters* are released from the *pre-synaptic* membrane into the gap between the two membranes. These transmitters move across the cleft and attach to the opposing cell (or *post-synaptic*) membrane. There, the neurotransmitter acts as a key to change the local permeability of the cell membrane to a given ion. One neurotransmitter mediates the flow of sodium. (The actual mechanisms of membrane transport are far more complicated and incompletely understood.) Both the electrical and chemical gradients drive the positively charged sodium ions into the cell, raising the electrical potential. The effect of the neurotransmitters does not last long, so the result is a small *depolarizing* potential which flows down the dendrite into the soma. If enough of these small currents arrive at the soma from the many dendrites at about the same time, the cell reaches its *threshold* potential. The entire membrane suddenly becomes very permeable to sodium, and the internal potential quickly reaches a level about 50 millivolts higher than the external fluid. A potential*spike* is produced at the *axon hillock*, and the resulting current or *action potential* travels down the axon. Potassium (K^+) then flows outward, bringing the cell back to its resting potential. Active and passive pump mechanisms restore the original gradients. Further dendritic signals are ignored until after a *refractory period*, when the cell is ready to receive inputs and fire again. Information appears to be carried predominantly through the frequency at which a given cell fires. The firing frequency is a nonlinear function of the total amount of input to the soma. Transmission down the axon

occurs through chemical means as well, and can be influenced by local conditions.

The impact that the firing of one cell will have on one to which it is connected is influenced by many factors. The amount of neurotransmitter received by membrane receptors largely determines how much current is sent to the soma. Mediating factors here include but are not limited to the amount and type of transmitter released by the pre-synaptic membrane, the size and shape of the pre- and post-synaptic membranes, the surrounding fluid's exact chemical makeup (which is influenced by metabolic mechanisms), and the synapse's recent history of activity. The amount of current which reaches the soma depends on the transmission properties of the dendrite and the spatial location of the synapse on the cell. In addition, there are *inhibitory* synapses which effectively lower the post-synaptic potential, preventing the cell from firing. There are also synapses among axons and dendrites themselves, on the cell body itself, and synapses can occur on top of other synapses. Some of these properties are evolutionary in nature. That is, the size of a synaptic membrane's active area appears to grow with use. In fact, a mechanism of this nature is currently the most likely candidate for the neurological basis of information storage and learning in the nervous system.

Simulated neurons

A schematic diagram of a representative unit is seen in Figure 2. Each unit contains a number of input lines and one output line, which bifurcates. Signals in biological neural networks are based on trains of equal valued spikes at different frequencies. Most simulations use real numbers to represent these signal frequencies at discrete time steps. Thus the activity level of a unit is a real number, as are the inputs and outputs. Input lines are attached to a unit through connections of (usually) variable strength, which are also represented by real numbers. The contribution of one input to the unit is the product of the activity on that line and the value of the connection strength. The total input to the unit is the sum of all the individual products. Connections may be positive or negative in sign. The former adds activity to the total, while the latter subtracts from it. The output of the unit is some function of the total input. Some commonly used activation functions are shown in Figure 3. Each unit may have associated with it a *bias*, which is the activity value of the unit at rest. This is similar to the resting potential of biological neurons, though it is often viewed as an additional input to the simulated neuron. It is worth repeating that this is a description of a representative simulated neuron. Variations such as temporal summation, delayed outputs, multiplicative (rather then additive) synapses, and internal memory of recent activity often occur in specialized systems.

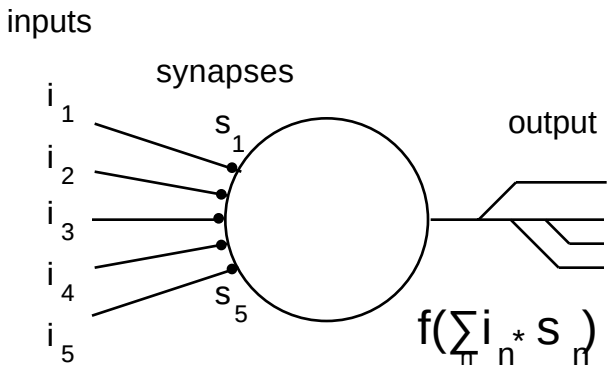


Figure 2. Representative simulated neuron

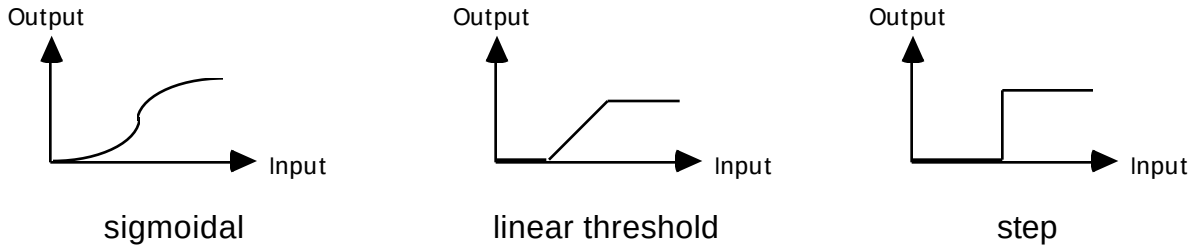


Figure 3. Common activation functions

Network architectures and interpretations

The architecture or *connectivity pattern* of a network describes its physical (virtual, if inside a computer simulation) layout. Spatial relations between units may be specified along with the connections between them. Usually, the spatial layout of a network is just a convenience for the designer, since the topology can often be changed while preserving the connectivity pattern. This is the specification of which unit's outputs connect to which other units as input. To facilitate this, the units in a network are often partitioned in groups. These may be seen as functional subnetworks, layers in a stratified structure, or just as divisions which represent the intentions of the designer. The layers are usually distinguished by restricted connection specifications, such as allowing no connections among the units in a group, but full connectivity between groups. The direction of propagation of the signals among units is usually restricted and is formalized as well. Biological neural networks often appear to be partitionable into such interacting populations, and are modeled as such.

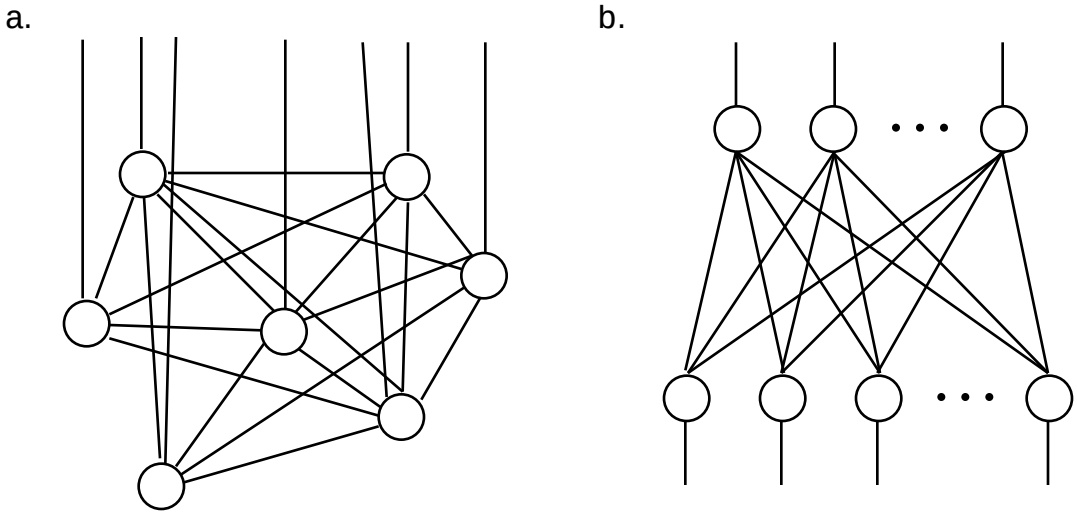


Figure 4. a) single layer network b) two layers

The structure of a network generally reflects the intended interpretations of its layers and units. The individual populations in a multi-layered network might represent the notions of input mechanisms, processing structures, memory systems, and output devices. Within populations, individual units may or may not have simple or discrete interpretations. In *localist* networks, units may correspond to such high level notions as words, concepts, terms in an equation, or categories. For example, a population of input units might represent line segment detectors, with connections to another population of units which represent alphanumeric characters, with connections to another population of units which represent words, with connections to another population of units which represent categories[1]. The term localist is used in reference to the idea that these qualities are localized to specific units. Extracting meaning from a localist network is relatively simple. The activity level of each unit represents the amount to which the indicated concept is involved in the current state of the network. A list of all the units' meanings and their current activity levels can yield a complete description of a network's current "thought". Different *patterns of activity* across the units represent different "thoughts".

On the other end of the spectrum are *distributed* or *coarse coded* representations in which such localization of concepts to units is not possible. Rather, units correspond to so-called *microfeatures* of the environment. A localist unit is replaced by a group of units who's activity levels as a group represent the quality in question. No single unit can be said to represent any describable entity. This makes the interpretation of activity patterns more

[1]For example, McClelland, J. L., and Rumelhart, D. E. (1981) An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings. *Psychological Review* **88**: 375-407

difficult, but makes their interactions more subtle (and interesting). Many concepts can be represented and thus interact in a single group of units. This type of representation also has the advantage of better fault tolerance and noise immunity, in that failure of any single unit should not greatly disrupt the system as a whole. In contrast, failure of a localist unit which represents a word means that the word can no longer be expressed. Similarly, while localist representations define and limit the concepts available to a system, more distributed representations allow flexible interpretations of what concepts are in existence or are available.

Many issues in network and problem representation in general are complex and poorly understood. Local and distributed schemes are regions on a continuum of problem and variable "coarseness". Most traditional artificial intelligence research has focused on localist representation of symbols and meaning, probably influenced by the localist nature of traditional computer architectures. Connectionist systems appear to alleviate this constraint to some degree, but much more work is required before any claims of superiority can be made. A major problem with distributed representations is the difficulty in assigning meaningful activity patterns to concepts and hence to populations. Unless this is done carefully, systems which attempt to be distributed in nature can become nothing more than jumbled and complex versions of local instantiations.

Along with the information contained in the definition of units, there is information in the connections between them. The presence of a unit (or group of units) defined as a concept means that the concept is available to the system. Similarly, a connection between two units (groups) implies that the two concepts may be related and thus influence each other. In the case of a more distributed representation, the connectivity pattern within a population can restrict and shape the activity patterns which are likely to occur. While this is also applicable to localist systems, in the case of distributed representations it deeply affects the way concepts are realized.

A network's connectivity pattern generally defines potential connections among units, but not the specific strengths. It is these strengths which embody a given system's knowledge under its framework. Connection strengths may either be part of a system's definition, or may be learned by the system through experience. The former scheme is usually restricted to localist networks, where the correlations among pre-defined conceptual units are known. The larger and more interesting set of networks learn these correlations.

Learning and memory

A widely held theory concerning the neurological basis of memory is that of long term potentiation (LTP), or Hebbian learning.[1] Though Donald Hebb did

[1]The more accurate term 'long term enhancement' (LTE) has recently been suggested by McNaughton

not create the notion of synaptic modification as the basis of learning, he is considered responsible for its current understanding. In his 1949 book, The Organization of Behavior, he outlined cell assemblies that are the forerunners of today's neural networks[1]. The concept of Hebbian learning specifically refers to the statement *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"* [2]. This was initially offered as an explanation of how information arriving in a brain could be buffered during processing. Connections which quickly adapted to an activity pattern could keep that pattern resonating as long as necessary. Such cell assemblies form the basis for learning in cognitive systems. Hebb suggested, with no physiological evidence, that a change in the area of contact of synaptic knobs was responsible. Sufficient evidence now exists such that this is the predominant paradigm today[3]. It is supposed that this increase in synaptic connection strength is related to the firing frequencies of the two cells involved. In a simulated network, the product of two unit's activity values often determines the change in their connection strength. This is generally what is referred to as Hebbian learning in connectionist systems. Variations commonly arise when the values used in the product are not the cell activity values themselves, but are functions of them. This is often done to include error correcting procedures. Neighboring cells and even more global interactions may also play a role in the formulation of the learning rule.

Associative memory[4]

The notion of memory in traditional computer systems differs greatly from its conception in human cognition. 'Traditional computer systems' refers to serial Von Neuman architectures with separate processing and memory systems. Information is stored in discrete locations within some storage medium. Access to that information can only be made through its 'address', or known physical location. When the content of the information is known but not its location, a search must be performed. In this case, the content must be known exactly, or else some pattern matching function must be used. In contrast, human memories are accessed only by content, with no regard to physical location. Stored information (events, names, faces) are recalled by

---

and Morris (1987). [See below]

[1]Hebb, D.O. (1949) The Organization of Behavior, John Wiley & Sons, New York

[2]Hebb, D.O. (1949), pp. 62

[3]McNaughton, B. L., and Morris, R. G. M. (1987) Hippocampal Synaptic Enhancement and Information Storage Within a Distributed Memory System. *Trends in Neuroscience* **10**:408-415

[4]For a more complete description of associative memory see
Hinton, G. E., and Anderson, J. A. (1981) Parallel Models of Associative Memory, Lawrence Erlbaum Associates, Hillsdale, NJ.
and
Kohonen, T. (1983) Self Organization and Associative Memory, Springer Verlag

thinking about something related, or *associated* with that memory. Additionally, the information used as the key need only be partially complete or correct. Incorrectly spelled names or partial views of objects usually do not impede the acts of recognition or recall. The inability of traditional computer systems to perform such operations is undoubtedly a major obstacle in artificial intelligence research. These most natural and prominent cognitive operations are difficult for these systems to emulate.

Two types of associative memory, auto- and hetero-associative, are often discussed. Auto-associative memory generally refers to the ability of a system to recognize and correctly recall information from partial or corrupted versions used as input. Hetero-associative memory systems 'couple' different information. If two items are stored as a pair, the presentation of one recalls the other. For example, talk of food often makes one hungry. Hetero-associative memory can be seen as a subset of auto-associative memory. Two events that are stored as a pair can be considered as a single event stored auto-associatively. Recall of one item from the other is then the same as recall of the entire memory from one of its parts. The distinction between auto- and hetero-association is most often a notational or architectural convenience.

Connectionist systems are well suited to perform associative memory operations. With a simple learning rule, a group of completely interconnected units (every unit connects to every unit) can perform auto-associative memory (see Figures 4a and 5). A pattern of activity is set up in the population (through dedicated input lines), and the synapses between the active cells are increased in strength. When a subset of that pattern is presented to the group, sufficient activity can spread through the strong connections to enable the missing units to fire, reinstating the original pattern. The information needed to recall a pattern is *distributed* throughout the connections in the system. Missing or innacurate connection strength values do not greatly affect performance, as they represent only a small amount the overall informational content. Many of these distributed representations can be stored simultaneously in the same set of connections. The fidelity of this kind of storage can depend on pattern independence, network size and structure, and learning specifications. Interference between stored patterns can be seen as a hindrance or as a benefit, depending on the aims of a particular model. Bad interference overwrites memories, while good interference allows generalization.

Systems with two layers of units can perform hetero-associative memory. Patterns are presented to each group, and the connections adjust to enable the mapping from one pattern to the other. A previously learned pattern is presented to one of the groups, and activity flows to the other layer, recreating the second pattern there. Flow of activity may or may not be restricted to one direction. When more than two groups are involved, architectures range from hierarchical trees to multiple interacting groups.

The storage and recall of sequences of events is another form of association. It can be viewed as associating the first item with the second, the second with the third, and so on. Presentation of any of the items will evoke the sequence from that point.

The mechanism which distinguishes learning from recall in biological memory, if one exists, is not known. Almost all simulations have distinct operations for them, and their use is under a system level control (usually the programmer).

Vector and matrix notation

A convenient and standard way to represent groups of units is as *vectors*. Each unit in a group is numbered, or indexed, and its activity value is stored in a vector at that index. The number of units in a layer determines the length of the vector, say **n**. A set of activity values in this vector can be viewed as a point in **n** dimensional space, just as the three component vector [x, y, z] is viewed as a point in three dimensional space. The connection strengths between two layers of size **n** can be represented in a *matrix* of size $\mathbf{n}^2$. Each matrix element represents the connection strength between two units. Let all of the units in layer 1 connect through synapses to all of the units in layer 2. View the top row of the matrix as the synapses which connect unit one in layer 2 to each of the units in layer 1. Each row **x** represents the synapses from each unit in layer 1 to the single unit **x** in layer 2. Similarly, a column can be viewed to represent the connections from a single unit in layer 1 to each unit in layer 2. A cell in the matrix, say [(row) 3,(column) 5] holds the value of the connection between unit 3 in layer 1 and unit 5 in layer 2.

To learn patterns of activity, each cell in the matrix can be incremented by the product of the activities of the two units it connects. This formulation is the same as the *outer product* of the two vectors (with layer 1 horizontal and layer 2 vertical) which forms a matrix (figures 5 and 6). Recalling stored patterns is the action of propagating activity from layer 1 to layer 2, through the connections. The layer 1 vector is oriented vertically to the right of the matrix, and a matrix-vector multiplication is performed. The sum of the products of a matrix row **x** and the layer 1 vector is the total input to unit **x** in layer 2 (see figure 7). If there is only one layer of units, the matrix stores the connections between those units. For example, [(row) 3,(column) 5] now holds the value of the connection between unit 3 and unit 5 in a single layer. Some architectures require that there only be one connection between any two units, and thus functions in both directions (so [3,5] = [5,3]). With the simple activity product rule above, this is a natural consequence anyway (since multiplication is commutative). Multiple patterns can be stored in the same matrix simply by superimposing the individual matrices formed by the above method. The associated patterns will interfere with each other if there is too much overlap or just too many patterns. The need to determine these conditions was a major force in the writing of Mactivation.

$$\text{Layer 1 vector} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{array}{c|c} & [\,1\;0\;1\;0\,] \\ \hline \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} & \begin{matrix} 1\;0\;1\;0 \\ 0\;0\;0\;0 \\ 1\;0\;1\;0 \\ 0\;0\;0\;0 \end{matrix} \end{array}$$

Figure 5. Auto-associative outer product connection matrix

$$\text{Layer 2 vector} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{array}{c|c} & [\,1\;0\;1\;0\,] \\ \hline \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} & \begin{matrix} 1\;0\;1\;0 \\ 1\;0\;1\;0 \\ 0\;0\;0\;0 \\ 0\;0\;0\;0 \end{matrix} \end{array}$$

Figure 6. Hetero-associative outer product connection matrix

$$\begin{bmatrix} 1\;0\;1\;0 \\ 1\;0\;1\;0 \\ 0\;0\;0\;0 \\ 0\;0\;0\;0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 0 \end{bmatrix} \text{ (normalizes to Layer 2 Vector)}$$

Figure 7. Hetero-associative recall

Summary

The basic neural ***unit*** receives many inputs which are connected to the unit through ***synapses***. The synapses modulate the amount of signal transmitted. The unit performs some kind of integration of these inputs, forming a total input signal for a given time slice. (This is continuous in biological neurons.) The unit's ***activation function*** determines the output of the unit based on the total input. Units are connected to each other to form ***networks*** with the output of units becoming the input to others. There must be some direct connections from the network to the outside world for input and observation. When learning, the synaptic connection strengths are modified according to ***learning rules*** and the ***pattern of activity*** in the network. These learning rules often have a local component based on what the synapse is connecting, and a more global component based on the ***network***

connectivity pattern. The mathematical ***outer product*** of two ***activity vectors*** is a ***matrix*** which represents the change in connection strengths for learning those patterns. Matrices are superimposed to store multiple sets of patterns. Activity in one layer is propagated to another by a ***matrix-vector multiplication***.

Mactivation

Overview

There are two neural network architectures available in Mactivation. A single layer of fully interconnected units with modifiable synapses performs auto-associative memory. Patterns are presented and learned, and can be recalled with a partial or incorrect presentation of the same pattern. Alternately, two layers of units with full connections from one layer to the other provides hetero-associative memory. A different pattern can be presented to each layer, and the mapping between the patterns is learned. Partial or complete presentation of the source pattern can later recall the target pattern. In addition to learn and recall commands, a cycling option allows output vectors to be transferred to the input vector in a continuous recall and update mechanism.

The Windows

Upon startup, the windows and menus appear as in Screen 1 (though the memory window and all activities would be empty). Mactivation is constantly under development, so details may vary. For example, windows may be in different places (or not there) and the vectors may be of different size and shape∂.

The **Layer 1** and **Layer 2** windows each contain one vector of units. Each is a one dimensional horizontal vector of activity values, but is usually arranged in a two dimensional grid. This allows familiar two dimensional patterns like characters and digits to be entered. Clicking the mouse in a unit in either of the vectors toggles the amount of input to that unit¥. This input value is then put through the activation function for that layer£. If the Caps Lock key is down, the activation function is bypassed. If the resulting activity value is positive, the unit appears grey. If the value is negative, the unit is black. If desired, these color assignments may be reversed¥. A value of zero appears white. The size of the resulting filled region within the unit reflects the ratio of the current value and the maximum possible (absolute) value for that unit. Only the Layer 1 window is visible when in auto-associative mode∂, as there is only one interconnected population of units in this architecture.

The **Connection Matrix** window contains a matrix of the synapses arranged as described in the vector-matrix notation section. Clicking in a matrix cell toggles between the two activity input values. The value is bounded by limits which are set in the Parameters -> Learn dialog box.

∂ Parameters -> Configure (read: this parameter is in the dialog box reached from the Configure item in the Parameters menu)

¥ Display -> Attributes

£ Parameters -> Activations

¥ Display -> Attributes

Positive, negative, and zero connection strengths appear as described above, using the ratio of each connection strength to the limiting values for the overall matrix.

The fields in the **Observe** window reflect the position of the mouse if it is in a unit or connection cell. 'Layer 1', 'Layer 2', 'Matrix', or 'Memory' appears in the top of the window. If the Observe window is active, values may be entered into any of the fields. This can used for getting values other than those set for toggling into the vectors, bypassing the activation function, and changing connection strengths. The best way to use this feature is to make the observe window active. The value field will be selected for keyboard input. Moving the mouse over the desired bit (layer unit or synapse) and typing the new value *and a return* will change that bit's value accordingly. The bit can also be specified by entering values into the row and column fields. (The tab key advances selection to the next field.) A return enters the information.

The **file name window**, with the same name as the current simulation system, displays previously learned vectors, in the order of learning. This window is often referred to as the **Memory** window. In hetero-associative mode, the Layer 2 vectors appear below the Layer 1 vectors. The Layer display threshold characteristics apply.

Modifier Keys

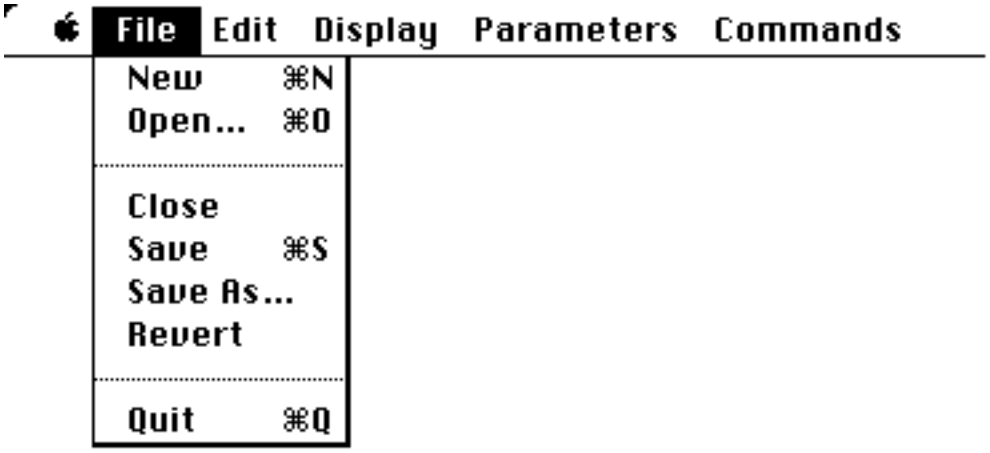
Option-clicking in either Layer vector or in any of the Memory vectors copies the pattern in that set of units into the Layer 1 vector. **Option-shift-clicking** copies the pattern into the Layer 2 vector.

The **Caps Lock** key disables the effect of the activation function when clicking a value into a vector. Similarly, connection strength bounds are ignored when clicking into the matrix.

The Menus

The **Apple** and **Edit** menus are excluded from this section since they are trivial in Mactivation. The **Edit** menu is unsupported, and is provided only for the use of desk accessories. The edit items become active when necessary.

File



The New command does a Reset Everything[1] and names the current simulation “Untitled”. Parameter settings are not changed.

The Open command presents the standard open file dialog box, showing folders and Mactivation documents (file type ‘MtVn’).

The Close command doesn’t do anything; a system must always be open. The menu item is provided for consistency and for desk accessories.

The Save command saves all parameter settings, vector and matrix values, and learned vectors in the file with the current system name. If the current system is Untitled, then Save As is invoked.

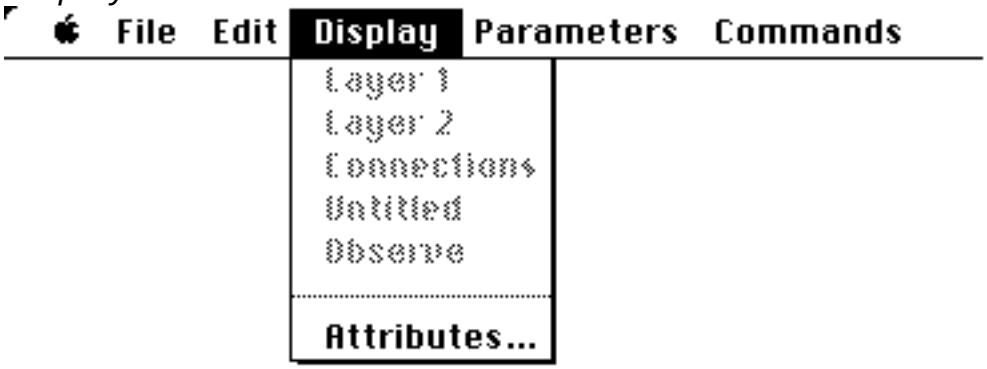
The Save As command presents the standard save as file dialog box.

The Revert command attempts to open the file in the current working directory with the same name as the current system.

Any state setting, learning, or changes to parameter values is considered a system modification and invokes a “Save changes to file xxx” box when quitting and other relevent actions are taken.

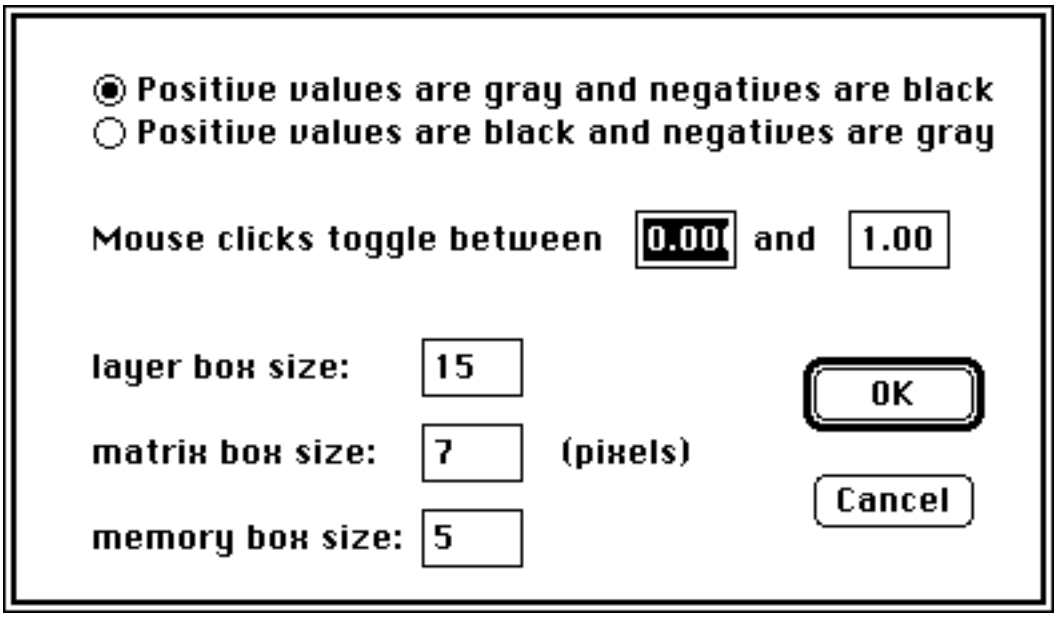
[1] Commands -> Reset Everything

Display



The first five items in the Display menu refer to the simulation windows. Windows currently visible on the Desktop are dimmed in the menu. When a window is completely obscured by other windows, its title becomes active in this menu list. Selecting its title makes the window visible again.

The **Attributes** dialog box:



The top two selections toggle the display 'colors' for positive and negative values in the layers, matrix and memory windows.

The **mouse click values** are the numbers sent to a bit that is clicked on. If in a layer unit, the number is put through the layer's activation function£. If in the matrix, values are bounded by the connection strength limiting valuesΩ. When the Caps Lock key is depressed, neither of these

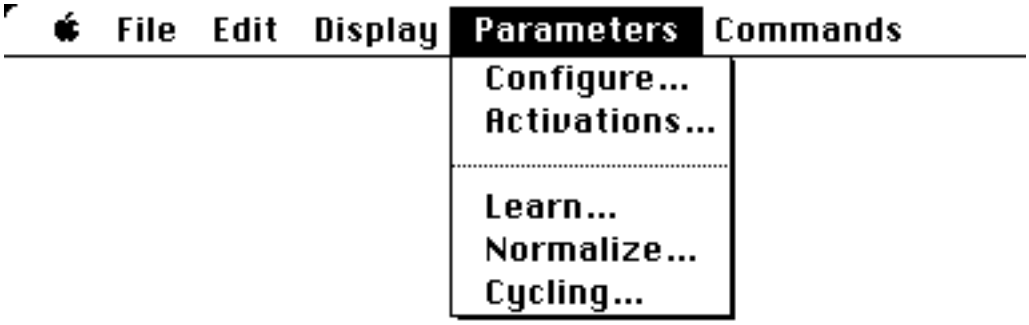
£ Parameters -> Activations

Ω Parameters -> Learn

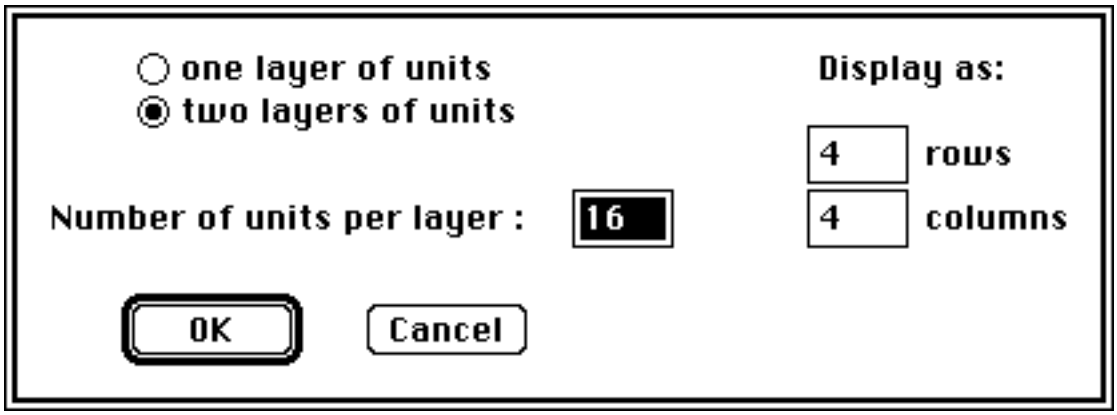
actions occur. Unit states can easily be set to zero and one with this feature, regardless of the activation function. Mactivation does its best to toggle correctly when the bit's pre-click value is neither of these, but you may have to click a couple times to get what you want.

The **box size** specifications let you adjust the simulation size to your liking.

Parameters



The **Configure** dialog box:



One layer or **two layers of units** configures the system for auto-associative or hetero-associative operation, respectively. Auto-associative means that the system is usually used to reconstruct a learned vector from partial or incorrect input. The Layer 1 vector is crossed with itself (outer product) to generate the connection matrix increment values. Hetero-associative allows a Layer 1 vector to map to a different Layer 2 vector. The Layer 2 window is only available when in hetero-associative mode.

[Changing modes after learning is allowed, and makes for interesting results. For example, teaching the hetero-associative system an overlapping sequence of patterns (i.e. a -> b, b -> c, c -> d) and then switching to auto-associative mode enables the single layer to recall the sequence when cycling[ß]. Note that the memory window displays its current understanding of the mode, so only the layer 1 patterns are shown.]

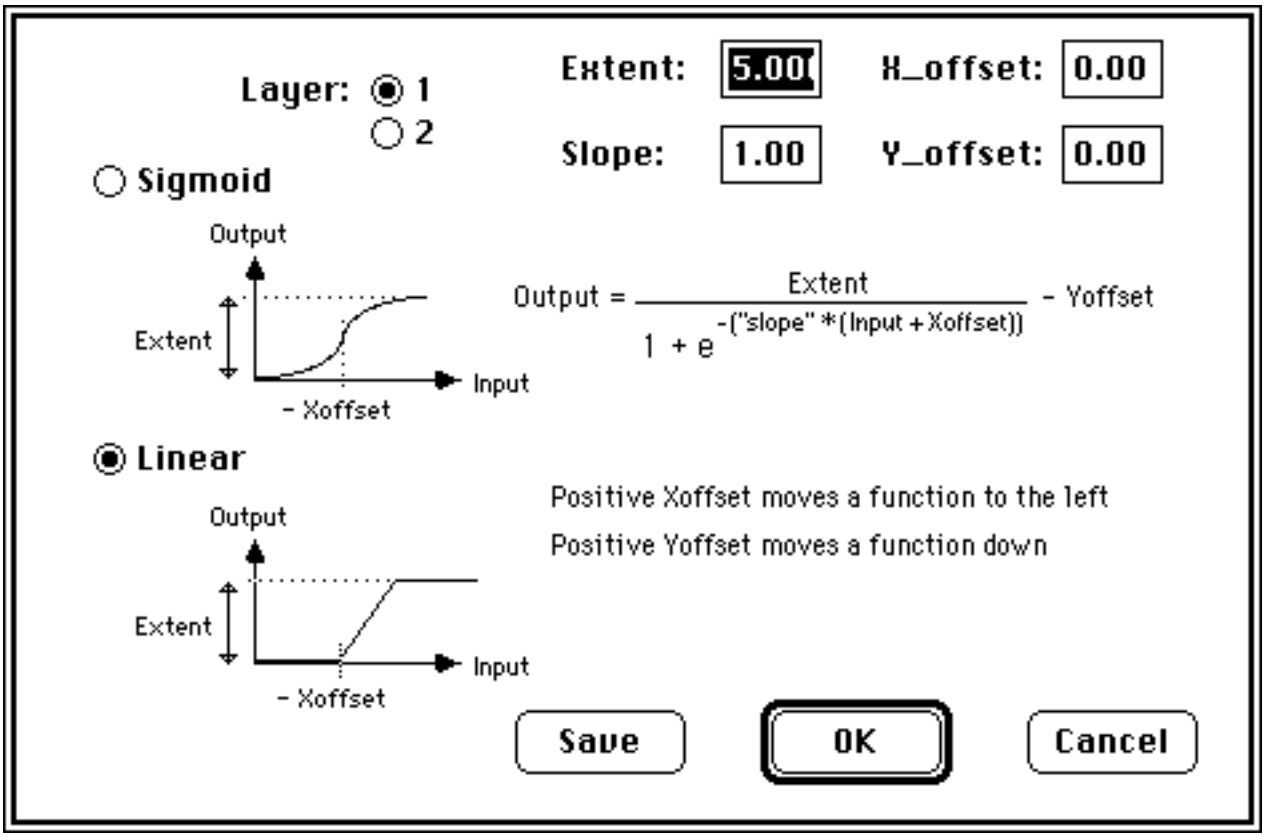
The **Number of units per layer** refers to the state vector(s) for the system, given in “units” or “neurons” or “bits” or whatever term you prefer.

Display as: refers to how the vector appears in the Layer 1, Layer 2,

[ß] Commands -> Cyclic Recall

and Memory windows. Using a rectangle in this manner rather than a one dimensional vector makes things a lot easier to look at and use. However, seeing a one-dimensional vector makes understanding the matrix multiplication easier for a while. Remember that what is displayed is a one-dimensional vector regardless of its shape. The number of rows times the number of columns must equal the vector size (for obvious reasons). Bit positions increase to the right and down, respectively.

The **Activations** dialog box:



Layer 1 and Layer 2 each have a different activation function for their units. Two prototypes are provided, a **sigmoidal** and a bounded **linear** function. Complete descriptions of these formulae are found in the equations section of the manual. "Slope" is in quotes since it is not actually a slope per se, but does change the speed of the function. A good way to get the feel of this function is to vary these parameters and the mouse click toggle amounts¥. Click the mouse to input different integer amounts of activity to units. The integer amount is immediately put through the function and displayed in the Observe window value field.

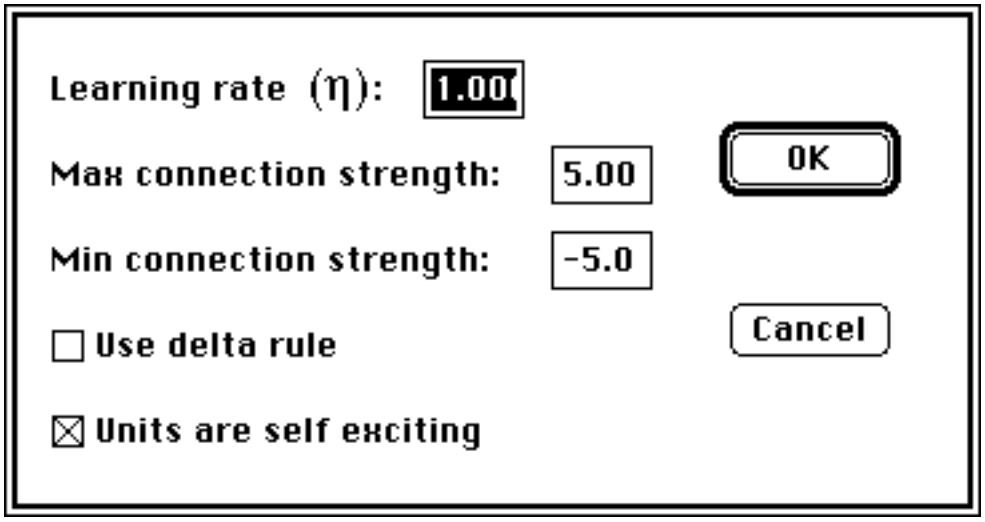
When the linear function is chosen, slope does indeed refer to its slope. A step function can be approximated with a large value for the slope.

Changes in activation functions take effect *immediately*. The current

activity values in the affected vector are put through the new activation function and the display is updated.

The **save** button stores the current settings and leaves the dialog box up. This is for setting one layer's parameters and then the other layer's. If you switch layers without doing a save, any changes made are discarded. **Cancel** does not undo changes which have been saved. **OK** saves the current settings and dismisses the dialog box.

The **Learn** dialog box:



The **Learn** command[f] performs an outer product of the appropriate vectors and adds the result to the connection matrix. (A discussion of learning is found in the Learning and memory section of the Neural Network Introduction.) The **Learning rate (η)**[1] modulates the amount of each synaptic strength change in one learning cycle. The change in strength of a synapse is the product of η and the product of the two units' activity values. The equations are found at the end of the Mactivation reference section.

Max and Min connection strengths are limits for the matrix values. Connection strengths will not be increased (or decreased) beyond these values during learning. Values input through the observe window are not affected by these limits.

The **Delta rule**, also known as the Widrow-Hoff procedure, is an error correcting variation of Hebbian learning. First, the layer 1 vector is

---

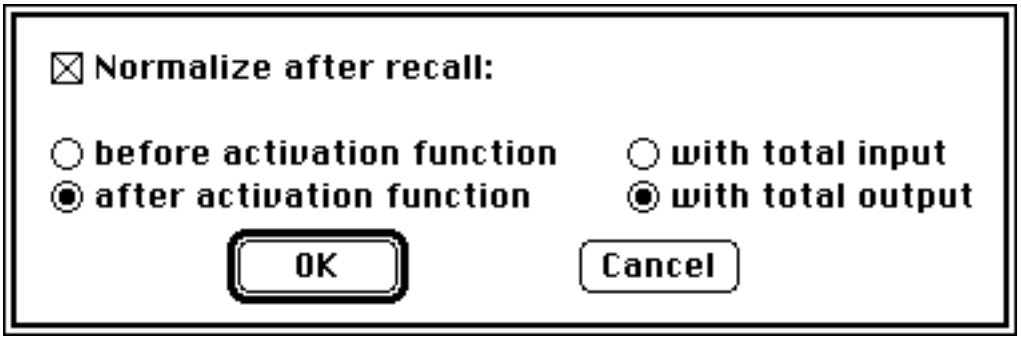
[f] Commands -> Learn Current

[1] This is the symbol "eta" in Symbol 12 font. If you don't have the Symbol font, this may look like a square box. Eta looks like an "n" with a downward extended right edge in case you want to look for it in some other font (it's option-h in Math).

multiplied by the matrix. The resulting vector is what the system 'knows' about the input. (This is the same as the recall procedure.) The result is subtracted from the *target* vector (layer 1 for auto-association, layer 2 for hetero-association), giving an error vector. The outer product of the layer 1 and error vectors is then used to update the connection weights. The equations are found at the end of the Mactivation reference section.

Some auto-associative models, such as the Hopfield net[1], do not allow a unit's output to connect back to the unit itself. Units that do connect to themselves are known as **self-exciting**. If the toggle is off, the diagonal of the matrix will always be zero since there will be no connection from unit *n* to unit *n*. Changing the setting only affects later presentations, not vectors that have already been learned. Hetero-associative operation fully connects Layer 1 to Layer 2, so this item is disabled.

The **Normalize** dialog box:



If a vector is **normalized** after being recalled, it means that each unit's value in a recalled vector will be divided by the sum total activity of a vector. In Mactivation, this division can be performed before or after the output unit's activation function is applied. The divisor can be the sum of either the input vector or the output vector itself. The most common definition of normalization divides the output activity after the activation function by the total of itself. This produces a standard normalized vector, the sum of which is equal to one. The other configurations are provided for experimentation, and to be able to simulate a model specified by McNaughton and Morris[2]. In this model, the division is performed before the activation function, with the total of the input vector.

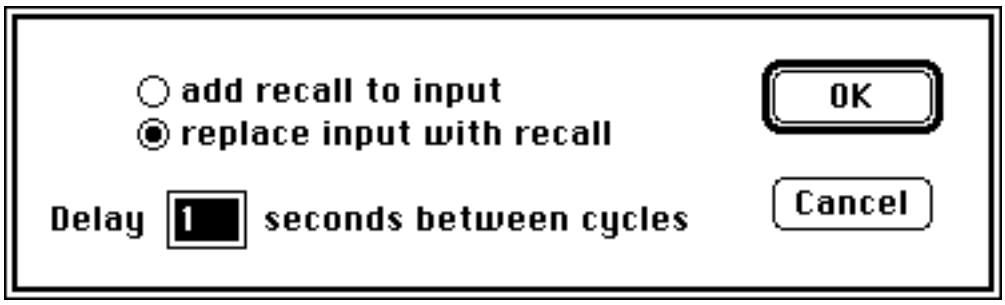
If total input is selected, the total of post-activation activity is taken. If total output is used, the total of pre-activation activity is taken.

---

[1]Hopfield, J.J. (1982) Neural Networks and Physical Systems With Emergent Collective Computational Abilities. *Proc. Natl. Acad. Sci. USA* **79**: 2554-2558

[2]McNaughton and Morris (1987)

The **Cycling** dialog box:

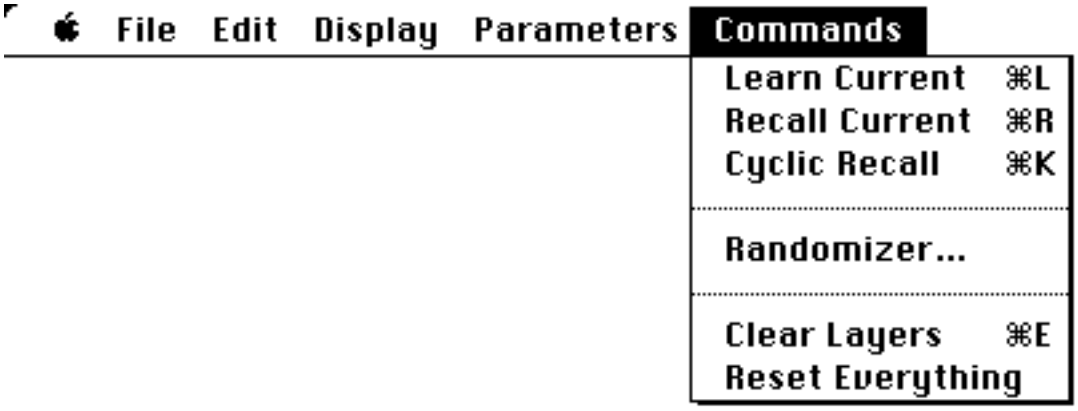


This dialog controls the operation of the **Cyclic Recall** command[ß]. When in hetero-associative mode, and cyclic recall turned on, output from layer 2 is sent back to layer 1 after a recall, and the process repeats. The layer 2 vector can either be **added** to layer 1 or can **replace** it. The activation function for layer 1 is then applied. When in auto-associative mode, the recalls are performed as usual, so the add and replace buttons have no meaning and are disabled.

The **delay** time between cycles, in seconds, is also set here. Zero is an acceptable value, but causes a very tight loop. The cursor changes to signify when a recall operation is being performed. TickCounts are used for this test, so accuracy is to approximately the sixtieth of a second.

[ß] Commands -> Cyclic Recall

Commands



Learn

The **Learn** command performs an outer product of the appropriate vectors and adds the result to the connection matrix. A discussion of learning is found in the Learning and memory section of the Neural Network Introduction. The learning equations follow this reference section. If auto-associative mode is currently being used, an outer product is formed by crossing the Layer 1 vector with itself, and the matrix is updated according to the current system parameters, which are explained in the dialog box descriptions. If in hetero-associative mode, this command crosses the two vectors with Layer 1 horizontal and Layer 2 vertical. The Learn Current command also adds a copy of the learned vector(s) to the Memory window display.

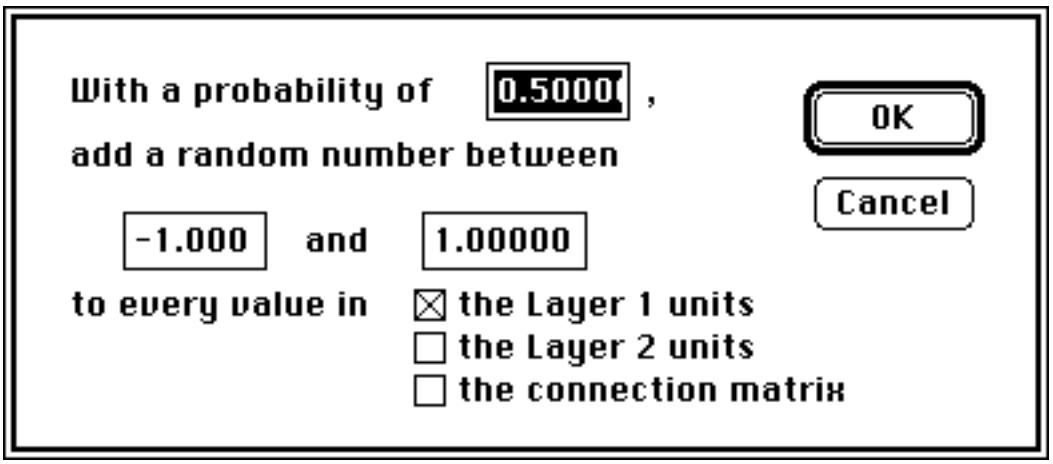
Recall

The **Recall** command multiplies the Layer 1 vector by the connection matrix. In auto-associative mode, the new Layer 1 activities are displayed. In hetero-associative mode, Layer 2 is updated. The appropriate activation function is first applied. One would generally click a partial or distorted input (or option-click a memory pattern) into Layer 1 and then choose this command. Normalization and thresholding occur as specified in the system parameters.

Cyclic Recall

Cyclic Recall is a toggled command and has a check mark to its left when in operation. At the specified interval, recall operations are performed. In hetero-associative mode, the resulting Layer 2 vector can replace or be added to Layer 1. This decision is based on the setting of the corresponding control in the Parameters -> Cycling... dialog box. The Layer 1 activation function is then applied, and the result is displayed.

Randomizer...



Random values, or noise, may be added independently to the layers of units and the connection matrix. The probability value determines how many units in the given structure will be affected. A probability of 0.5 says that about half of the the units will have noise added to them. Higher probabilities allow more units to be influenced.

For each unit selected, a random number which lies between the two specified values is generated and added to the unit's activity (or connection strength). Values are bounded by the maximum value allowed for a unit's activity or matrix connection strenght. The associated algorithm is found in the equation section of the manual.

The Clear commands

Clear Vectors inputs zeros to all of the units in the two layers. The activation function is then applied, leaving the unit at its 'resting state'.

Reset Everything performs a Clear Vectors, fills the matrix with zeros, and empties the memory window.

The equations

[Layer1] = Vector of Layer 1 activation values
[Layer2] = Vector of Layer 2 activation values
[Temp] = A spare vector
[W] = Connection matrix
Wij = Connection weight from unit j to unit i
n = number of units
Δ**x** = Change in value of **x**
= The vector outer product operator
• = Matrix - vector multiplication operator
T = The vector transpose operator
Remanining variables have the same names as in their dialog boxes.

Activation functions:

sigmoidal:

$$\text{Output} = \frac{\text{Extent}}{1 + e^{-(\text{"slope"} * (\text{Input}+\text{Xoffset}))}} - \text{Yoffset}$$

linear:

Output = "slope" * (Input + Xoffset) - Yoffset

if Output < -Yoffset, then Output = -Yoffset
if Output > (-Yoffset + Extent), then Output = (-Yoffset + Extent)

Auto-associative recall:

$$\text{Input to unit i} = \sum_{n} (\text{Wij} * \text{Output(unit j)})$$

Output(unit i) = Activation function (Input to unit i)

In vector notation:

[Temp] = [W] • [Layer 1]
[Layer 1] = Activation function ([Temp])

Hetero-associative recall:

[Layer 2] = Activation function ([W] • [Layer 1])

Learning without the delta rule, auto-associative:

For all Wij:

$\Delta Wij = \eta * Output(unit\ j, layer\ 1) * Output(unit\ i, layer\ 1)$

$Wij = Wij + \Delta Wij$

In vector notation:

$$[\Delta \underline{W}] = \eta * [layer1] \quad [layer1]^T$$

$$[\underline{W}] = [\underline{W}] + [\Delta \underline{W}]$$

Learning without the delta rule, hetero-associative:

$$[\Delta \underline{W}] = \eta * [layer2] \quad [layer1]^T$$

$$[\underline{W}] = [\underline{W}] + [\Delta \underline{W}]$$

Learning with the delta rule, auto-associative:

$[Temp] = [\underline{W}] \bullet [Layer\ 1]$

$[Temp] = Activation\ function\ ([Temp])$

$[Temp] = [Layer\ 1] - [Temp]$

[Temp] now contains an error vector

$$[\Delta \underline{W}] = \eta * [Temp] \quad [Layer\ 1]^T$$

$$[\underline{W}] = [\underline{W}] + [\Delta \underline{W}]$$

Learning with the delta rule, hetero-associative:

$[Temp] = [\underline{W}] \bullet [Layer\ 1]$

$[Temp] = Activation\ function\ ([Temp])$

$[Temp] = [Layer\ 2] - [Temp]$

[Temp] now contains an error vector

$$[\Delta \underline{W}] = \eta * [Temp] \quad [Layer\ 1]^T$$

$$[\underline{W}] = [\underline{W}] + [\Delta \underline{W}]$$

The Randomizer:

The call srand(time()) is made at application startup to seed the rand() function.
The LightspeedC rand() function generates a value between -32767 and +32767

For each unit in a selected layer/matrix:

Generate a random number with rand();
if it's less than the specified probability times 32767, then generate noise for that unit.

To generate the noise value:

Generate and take the absolute value of a rand() number;
Multiply it by the extent of the specified range of acceptable values (max - min);
Divide by 32767;

Add the final value to the unit.
Limit the resulting value by that allowed for the unit.