Learning-Logic: Casting the Cortex of the Human Brain in Silicon

by David B. Parker

Technical Report TR-47 February 1985

Center for Computational Research
in Economics and Management Science
Building E40-140
Massachusetts Institute of Technology
Cambridge, MA 02139

Phone: (617) 253-8419
BitNet: DBPDEV@MITVMA
Arpanet: DBPDEV%MITVMA@MIT-MULTICS.ARPA

ABSTRACT

Learning-Logic is two things: 1) a model of the neurons in the human cortex and 2) a practical way to create electronic circuits that can learn and, I believe, think.

As a model of our brains, Learning-Logic can explain why there are two types of neurons in our cortex -- the pyramidal and stellate neurons -- and can describe mathematically how each type works. As part of this, Learning-Logic can explain how long term memory works. These explanations can be verified experimentally because Learning-Logic makes several specific predictions about the behavior of the pyramidal and stellate neurons. By being able to identify the significance of various neural features to learning and thinking, Learning-Logic may be of help in the search for cures to certain types of learning disorders.

As a practical invention, I believe Learning-Logic will enable us to bypass the 5th generation of computers and move directly to the 6th -- to computers which require no programming at all and which have the same capabilities as the human mind. Learning-Logic cells, which correspond to the neurons of our brains, can be simulated in software that runs on standard computers or can be created directly on integrated circuit chips. Stanford University is patenting Learning-Logic.

Table of Contents

INTRODUCTION

Off and on for the past 4½ years I have been thinking about how the neurons in our brains work: how they operate individually and in concert. I now feel ready to present the results of my investigations, which are:

- Mathematical learning algorithms for individual cells. Each cell is basically doing a least-squares fit. The algorithms presented are $O(p^2)$ approximate updating algorithms (where p is the number of parameters) that converge to exact least-squares solutions.

- Mathematical algorithms for connecting arbitrary numbers of such cells into networks. The interconnections are arranged so that each cell changes as little as possible when the network as a whole learns something new. This ensures that the network remembers old information as long as possible. The derivation of the interconnection algorithm is based on the method of Lagrange multipliers. A consequence of this is that in order to connect cells together in a manner similar to the neurons in our brains, two different types of cells are required.

- An association between the mathematical elements of the above algorithms and the physical elements of real neurons. Some observed phenomena that can be explained if this correspondence can be experimentally verified are:

  1. The two types of neurons in our cortex, the stellate and pyramidal neurons, correspond to the two types of cells mentioned above.

  2. Long term memory consists of certain physical changes in the neuron. All the necessary changes could be carried out by the neuron simply changing its shape -- all it would need to vary would be the distances between the synapses and the axon hillock, and the inter-synaptic distances. However, some of these changes are probably carried out by different, though functionally equivalent means, such as synapses changing their electrical properties. In "Questions and Answers About Our Neurons" I discuss these variations, but for the rest of the paper I'll assume the neuron is varying just distances, to keep things uniform. Figure 1 is a crude picture of a generic "distance-varying" neuron.

  3. Short term memory most likely consists of variations in the patterns of signals sent from neuron to neuron. Our neurons can learn the digital functions used in electrical circuits which are often used as analogies for short term memory.

There are also some as yet unobserved phenomena that Learning-Logic predicts. The most startling prediction, at least to me, is that

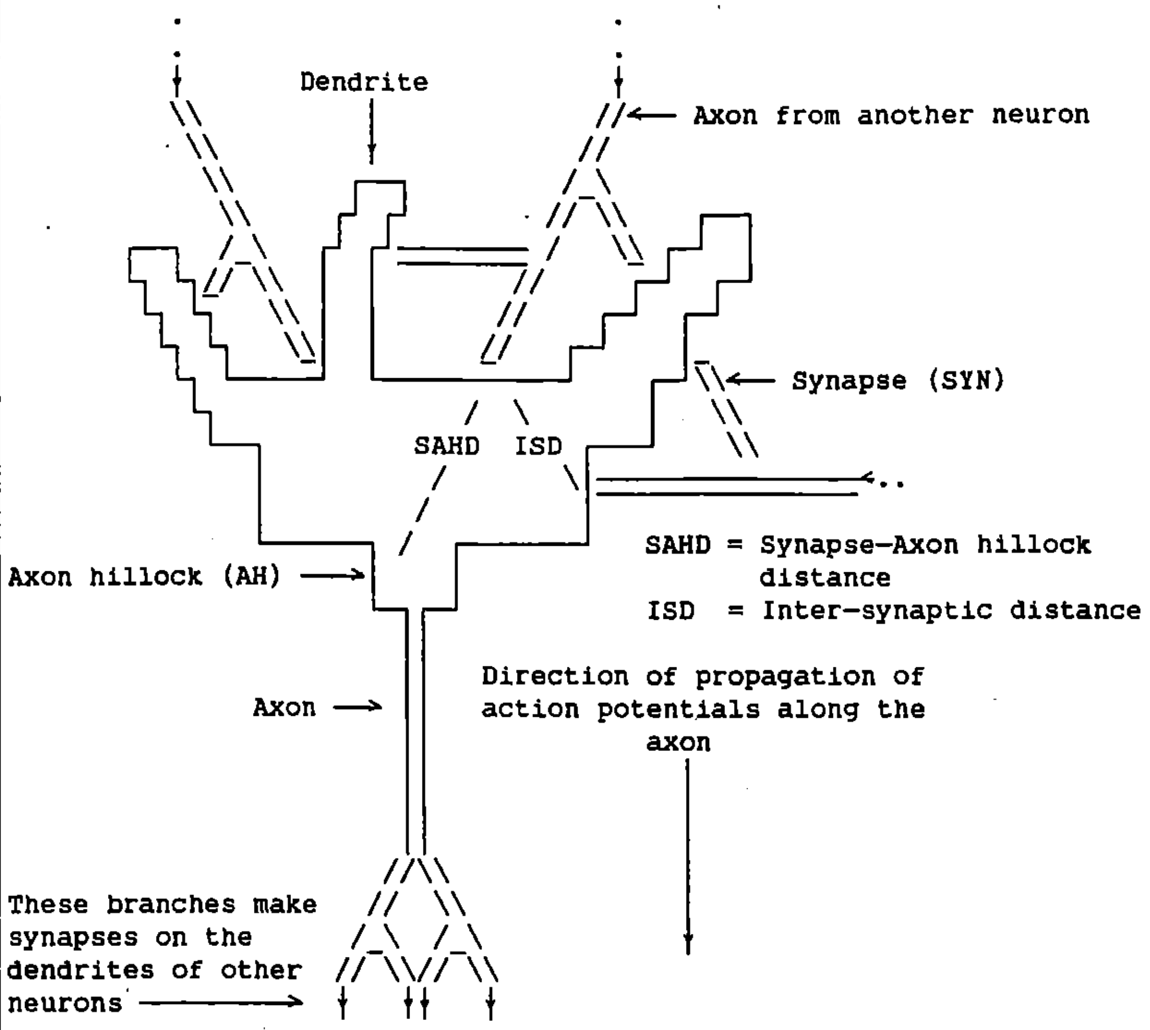

Dendrite
Axon from another neuron
Synapse (SYN)
SAHD
ISD
SAHD = Synapse-Axon hillock distance
ISD = Inter-synaptic distance
Axon hillock (AH)
Axon
Direction of propagation of action potentials along the axon
These branches make synapses on the dendrites of other neurons

Figure 1: A typical neuron. The parts of the neuron are labeled as though it is maintaining long term memory using distances only, to correspond with Figure 7 and Figure 15.

stellate neurons shouldn't be able to cause pyramidal neurons to fire, and vice versa (or at least the effect should be much less than the effect of stellate neurons on stellate neurons and pyramidal neurons on pyramidal neurons).

In addition, the association between the algorithms and real neurons points toward several possible causes for some kinds of learning disabilities. Perhaps Learning-Logic may be of use some day in helping to cure them.

- Practical ways to create artificial neurons that can be connected into trainable learning networks. Independently of whether these mathematical algorithms correspond to our brains or not, they still work and so can be implemented in hardware or software to form what I call "Learning-Logic" (the name comes from the fact that current logic cirucits, such as NAND or NOR gates, can't learn and so might be called "fixed-logic"). I have drawn the circuit diagrams in this paper in an implementation independent fashion, using op-amp like circuit elements whose equivalent elements in any particular implementation can readily be worked out. Some examples of Learning-Logic implementations are:

  1. A whole network can be simulated in software that runs on an ordinary computer.

  2. Several microprocessors, each simulating one or more cells, can be connected together to form a larger network, thus allowing the cells to operate in parallel.

  3. Learning-Logic can be directly implemented on integrated circuit chips, with one or more cells per chip, all operating in parallel. Capacitors on the chips could be used to store various parameters as electrical charges.

  Stanford University is patenting Learning-Logic. If you wish to obtain a license for it, you can write to:

  Office of Technology Licensing
  Stanford University
  Stanford, California 94305

Learning-Logic cells can be compared to hypothetical smart multiple input transistors. Just as transistors can be used in either analog or digital circuits, Learning-Logic cells can learn to perform either analog or digital functions, or various combinations of both. What makes Learning-Logic smart is that it can learn what the optimal amplification should be for each input, whereas transistors have to be told how much to amplify.

There have been several other efforts to create networks of cells that can learn. The most recent I know of is the Boltzmann Machine of Hinton, Sejnowski and Ackley (1984). One of the more famous types of cells is the Perceptron of Rosenblatt (1962). The cells that I am most familiar with, and which Learning-Logic is closest to in spirit, are the ADALINE's of Widrow, et. al., (1967).

All of these other cells require some form of randomness in their signals in order to guarantee convergence -- and the reason for that, I believe, is because they are all basically $O(p)$ devices, where p is the number of parameters in each cell. Most types of Learning-Logic cells are $O(p^2)$ devices, as I believe the pyramidal neurons of our brains are, and so can be shown to converge without requiring any sort of randomness. Interestingly enough, if our cortical neurons are at all like Learning-Logic then our stellate neurons have to be $O(p)$ devices. In fact, I believe our stellate neurons are basically ADALINE's.

If you'd like to try out Learning-Logic, I am distributing a Learning-Logic program beginning in June, 1985. It is written for IBM PC's, XT's or AT's that have at least 128K of memory, a floppy disk drive, an 8087 or 80287 chip and that are running DOS 2.0 or higher. If you send me $20 to cover the cost of materials, copying, shipping and handling, I'll send you a diskette containing:

1. The Learning-Logic program (written in assembler language for speed). It enables you to create and run networks consisting of all the types of cells discussed in this paper.

2. Instructions on how to use the program.

3. Some sample networks and data to get you started.

I would like to thank everyone in the Computation Research Group, directed by Prof. Jerry Friedman, at the Stanford Linear Accelerator Center for all their help and encouragement on this project. Much of my work on Learning-Logic was carried out at SLAC. Similarly, I would like to thank everyone at the Center for Computational Research in Economics and Management Science, directed by Prof. Ed Kuh, at the Massachusetts Institute of Technology, where this paper was prepared.

Two people I would like to give a special thanks to are JoAnn Malina, of SLAC, for all our midnight discussions, and Prof. Bernard Widrow, of Stanford, for getting me pointed in the right direction.

UPDATING ALGORITHMS

The algorithms used by Learning-Logic cells are known as <u>updating algorithms</u>. The variables and parameters inside a Learning-Logic cell are being continuously modified as they are simultaneously being used.

For example, an updating algorithm for a vector $\bar{a}$ might be expressed as follows:

$$\frac{\partial \bar{a}}{\partial t} = \bar{e} + d\,\frac{\partial \bar{a}}{\partial t}$$

This should be interpreted as:

$$\frac{\partial \bar{a}(t+\epsilon)}{\partial t} = \bar{e}(t) + d(t)\,\frac{\partial \bar{a}(t)}{\partial t}$$

where ϵ is a measure of the delay time through the circuitry that is calculating $\partial\bar{a}/\partial t$. If ϵ is sufficiently small compared to the rate at which $\bar{e}$ and d are changing and if certain other conditions are met -- in this case, if $-1 < d < 1$ -- then we can solve for $\partial\bar{a}/\partial t$ to get:

$$\frac{\partial \bar{a}}{\partial t} = \frac{1}{1-d}\,\bar{e}$$

AVERAGES

Throughout this discussion, great use will be made of averages. There are several ways that these averages can be calculated (see Figure 2). However, there is one particular form of average -- called an approximate running average -- which is not only the most practical to use for Learning-Logic cells, but which is, I believe, used by real neurons.

Let us start by examining an ordinary average (see Part 1 of Figure 2):

$$\mathrm{avg}(x(t)) = \frac{1}{t}\int_0^t x(\tau)\,d\tau$$

From now on where it is clear that a function depends on time, the t's and τ's will be dropped, so we can write:

$$\mathrm{avg}(x) = \frac{1}{t}\int_0^t x\,d\tau$$

An ordinary average remembers all history from $t = 0$. This is useful in many circumstances, but for Learning-Logic it will be more helpful to have an average that can forget part of its history (as we will see in the next section on signals).

$$\mathrm{avg}(x(t)) = \frac{1}{t}\int_0^t x(\tau)\,d\tau$$

$$\frac{\partial \mathrm{avg}(x(t))}{\partial t} = \frac{1}{t}\left(x(t)-\mathrm{avg}(x(t)) \right)$$

1) An <u>ordinary average</u>.

$$\mathrm{avg}(x(t)) = \frac{1}{h}\int_{t-h}^t x(\tau)\,d\tau$$

$$\frac{\partial \mathrm{avg}(x(t))}{\partial t} = \frac{1}{h}\left(x(t)-x(t-h) \right)$$

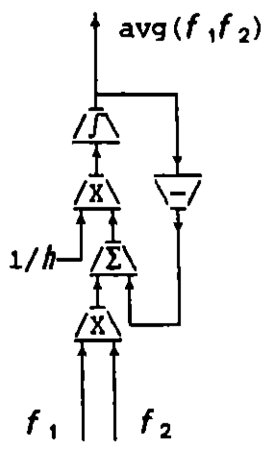
2) A <u>running average</u>, where h is a positive constant.

$$\mathrm{avg}(x(t)) = \frac{1}{h}\int_0^t \exp(-(t-\tau)/h)\; x(\tau)\,d\tau$$

$$\frac{\partial \mathrm{avg}(x(t))}{\partial t} = \frac{1}{h}\left(x(t)-\mathrm{avg}(x(t)) \right)$$

3) An <u>approximate running average</u>, where h is a positive constant.

<u>Figure 2:</u> Three types of averages. The positive constant h is a measure of the <u>history</u> retained by the running average and approximate running average. The only reason h is constant in the above equations is to suppress extraneous mathematical details from the expressions. In many cases it will be of advantage to allow h to vary with time. For example, if we set $h = t$ then both the running average and approximate running average become identical to the ordinary average.



$$\frac{\partial \mathrm{avg}(f_1 f_2)}{\partial t} = \frac{1}{h}\left(f_1 f_2 - \mathrm{avg}(f_1 f_2) \right)$$

/∫\ Integrator /X\ Multiplier /-\ Inverter /Σ\ Summer

Figure 3: Circuit diagram for an averaging multiplier.

One kind of average that can forget part of its history is the running average (see Part 2 of Figure 2):

$$\mathrm{avg}(x) = \frac{1}{h}\int_{t-h}^{t} x \, d\tau$$

Here h is be a postive constant, or a function of time, that explicitly states how much history we wish to retain. This would be perfect for Learning-Logic, except for a problem with its derivative:

$$\frac{\partial \mathrm{avg}(x)}{\partial t} = \frac{1}{h}\left(x(t) - x(t-h) \right)$$

We will want to use the derivative to update the average, but to do so for a running average means we would have to retain all values of x from $t-h$ to t. This is too much information to save.

We can get almost the same effect as a running average, without having to save past values of x, if we use an approximate running average (see Part 3 of Figure 2). What we do is this: instead of using the value $x(t-h)$ when updating the running average, we will instead use our best guess as to what $x(t-h)$ was, which is of course just avg(x):

$$\frac{\partial \mathrm{avg}(x)}{\partial t} = \frac{1}{h}\left(x - \mathrm{avg}(x) \right)$$

This gives us an average that allows us to control the amount of history retained and that is easy to update.

A component we will use later that is based on an approximate running average is the averaging multiplier (see Figure 3). It performs the function:

$$\frac{\partial \mathrm{avg}(f_1 f_2)}{\partial t} = \frac{1}{h}\left(f_1 f_2 - \mathrm{avg}(f_1 f_2) \right)$$

An averaging multiplier is used in Figure 8.

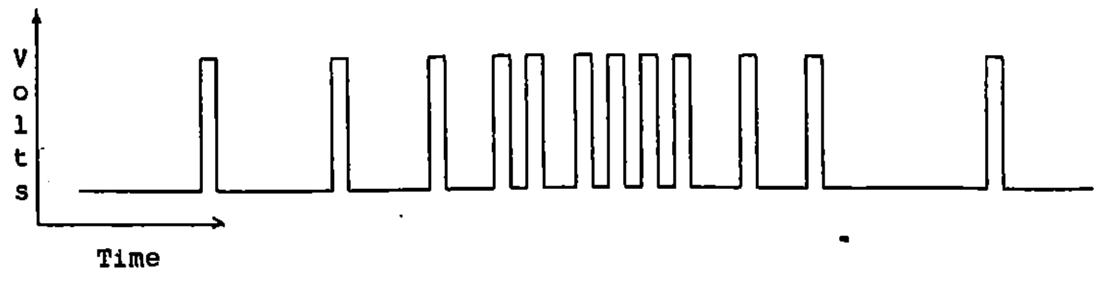
Variations and extensions of approximate running averages can be found in the section "Practical Considerations" under "Use of Exponentials and Logarithms".

SIGNALS

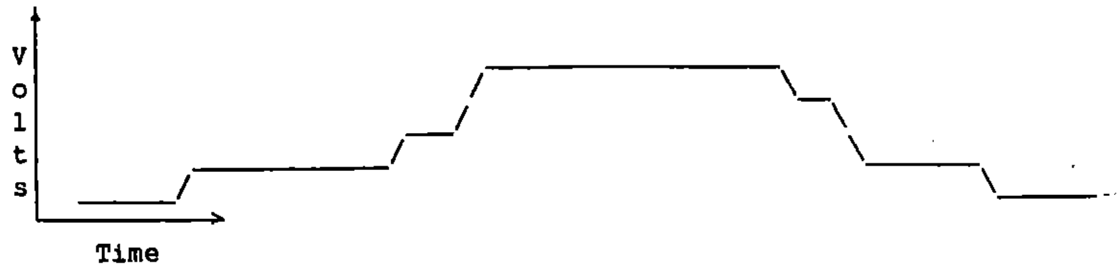
Learning-Logic cells, like the neurons of our brains, communicate by means of signals (see Figure 4). There are various kinds of signals that different types of cells might use.

The signals used by our neurons are pulses called action potentials (see Part 1 of Figure 4). These pulses strongly resemble the binary 0 or 1 pulses used in digital electronics. However, if Learning-Logic is an accurate model of our cortical neurons then there is a distinct difference between digital electronic pulses and action potentials. Digital electronic pulses are themselves the carrier of information, but I believe it is the frequency of the action potentials, and not the individual pulses themselves, which carries useful information.

Thus I believe that our neurons are basically analog devices, not digital, with the analog signal being the frequency of the action potentials. One way in which our neurons could measure the frequency of a series of action potentials is by means of an approximate running average:



1) Pulses. The frequency at a given instant is proportional to the average number of pulses in a small region around that instant.



2) Analog signals. Instead of using the frequency of a series of pulses, an analog signal can be transmitted directly.

```
.00  .09  .14  .27  .45  1.00  1.00  1.00  .86  .33  .25  .19  .02
──────────→
Time
```

3) Numbers. Integers or floating point numbers can be used to approximate an analog signal or series of pulses.

Figure 4: Some examples of signals. Many other types of signals are possible, depending on how the cells are constructed.

$$\frac{\partial f}{\partial t} = \frac{\partial \text{avg}(x)}{\partial t} = \frac{1}{h}\left(x-\text{avg}(x) \right) = \frac{1}{h}\left(x-f \right)$$

where x is a function of time representing the action potentials (like Part 1 of Figure 4) and f is a measure of their frequency. This is an example of why we wanted an average that could forget part of its history: to make an accurate judgement of the instantaneous frequency of a series of pulses, we should only count the pulses in some pertinent region of the immediate past.

Actually f isn't a frequency. It is scaled so that the minimum frequency (no pulses) corresponds to the minimum value of X and the maximum frequency (continuous pulses) corresponds to the maximum value of X. For example, suppose the minimum value of X is 0 and the maximum is 1. Then if there were no pulses f would be 0 and if X was a series of continuous pulses (i.e. at no time does it go to 0), then f would be 1. If X were alternately 0 and 1, then f would be ½.

A physical manifestation of this hypothesis would be that a neuron would appear to remain sensitive to succeeding action potentials of a series. The time of sensitivity is a measure of h. Neurons that remain sensitive for longer periods of time would be better at dealing with low frequency phenomena; neurons with shorter sensitive periods would be better at dealing with high frequency phenomena (and cells that have 0 sensitivity would in fact be dealing with pulses on a pulse by pulse basis).

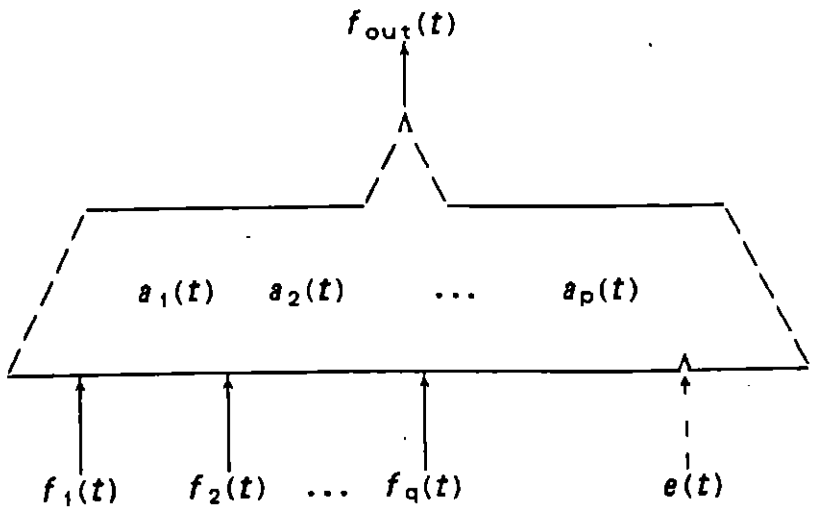
Another example of how to convert pulses to analog signals is in the section "Practical Considerations" under "Thresholding".

Learning-Logic cells need not be built, however, to reproduce the action potentials of our brains, as action potentials probably arose as an evolutionary expedient. We can transmit <u>analog signals</u> directly from cell to cell (see Part 2 of Figure 4). If Learning-Logic was being constructed directly on integrated circuit chips, voltage levels would probably be the appropriate analog signals to use. We could even combine the pulse and analog signal approaches and send an analog signal which is averaged like the pulses.

If one is simulating Learning-Logic on a computer, integers or floating point <u>numbers</u> can be used in place of pulses or analog signals. (see Part 3 of Figure 4).

HOW FUNCTION CELLS WORK

<u>Function cells</u>, either by themselves or as subcomponents of other types of cells, are basic components of any Learning-Logic network. I believe that the pyramidal cells of our cortex are the biological equivalent of function cells.


$f_{out}(t)$
$a_1(t)$ $a_2(t)$ ... $a_p(t)$
$f_1(t)$ $f_2(t)$ ... $f_q(t)$
$e(t)$

$$f_{out}(t) = f_{out}(\, a_1(t),\ldots,a_p(t), f_1(t),\ldots,f_q(t)\,)$$

$$e(t) = d(t) - f_{out}(t)$$

Figure 5: "Black box" diagram of a function cell, where $f_1,\ldots,f_q$ are input signals, $a_1,\ldots,a_p$ are internal parameters, f_{out} is the output signal, e is the error signal, and d is the desired function. The cell continuously updates its internal parameters to minimize the error signal.

Figure 5 is a picture of a general function cell. It is given $f_1,\ldots,f_q$ as input signals from which it calculates

$$f_{out} = f_{out}(\, a_1,\ldots,a_p,f_1,\ldots,f_q\,)$$

where $a_1,\ldots,a_p$ are the cell's internal parameters, which the cell is continuously updating as part of its learning and remembering process. To help it update its internal parameters, the cell uses an error signal, e, to guide it.

The function f_{out} can be either linear or non-linear, but to keep the notation as simple as possible in the derivations and proofs, I will assume that the function is linear:

$$f_{out} = a_1f_1 + a_2f_2 + \ldots + a_pf_p$$

The results of these derivations and proofs will, of course, be stated for both the linear and non-linear cases. For the linear case, if we let $\vec{f}$ be the vector of input signals, and $\bar{a}$ be the vector of internal parameters, we can write:

$$f_{\mathrm{out}} = a_1 f_1 + a_2 f_2 + \ \ldots \ + a_p f_p = \bar{a}^{\mathrm{T}}\vec{f} = \vec{f}^{\mathrm{T}}\bar{a}$$

We will assume that the error signal e is being generated as the difference between a desired function d and the actual output of the cell f_{out}:

$$e = d - f_{\mathrm{out}} = d - \bar{a}^{\mathrm{T}}\vec{f} = d - \vec{f}^{\mathrm{T}}\bar{a}$$

This makes the derivations and proofs much easier. In real life, the error signals needn't be related to the function outputs at all. For instance: it's painful when you lose an hour's worth of editing because the computer happens to die, but the computer didn't die just because you had been typing for an hour (although there does exist some evidence to the contrary).

There are many methods a function cell could use to update its internal parameters $\bar{a}$ to try and minimize e, and most of them will probably work with the interconnection method discussed in the next section. However, we will concentrate on variations of the method of <u>least-squares</u>.

The exact least-squares method works as follows: a reasonable way to choose $\bar{a}$ is to find the $\bar{a}$ that minimizes the total squared error from $t = 0$ to the present:

$$e^2_{\mathrm{total}} = \int_0^t e^2 \, d\tau = \int_0^t (d - f_{\mathrm{out}})^2 \, d\tau = \int_0^t (d - \vec{f}^{\mathrm{T}}\bar{a})^2 \, d\tau$$

$$= \int_0^t (d - a_1 f_1 - a_2 f_2 - \ldots - a_p f_p)^2 \, d\tau$$

The well-known solution is that:

$$\bar{a} = \left[\int_0^t \vec{f}\vec{f}^{\mathrm{T}} \, d\tau \right]^{-1} \int_0^t d\vec{f} \, d\tau = \left[\int_0^t \vec{f}\vec{f}^{\mathrm{T}} \, d\tau \right]^{-1} \int_0^t (f_{\mathrm{out}} + e)\vec{f} \, d\tau$$

We can scale everything by $1/t$ to arrive at the equivalent average formula:

$$\bar{a} = \mathrm{avg}(\vec{f}\vec{f}^{\mathrm{T}})^{-1}\mathrm{avg}(d\vec{f}) = \mathrm{avg}(\vec{f}\vec{f}^{\mathrm{T}})^{-1}\mathrm{avg}((f_{\mathrm{out}} + e)\vec{f})$$

To convert this to an updating formula, we take $\partial/\partial t$ to get:

$$\frac{\partial \bar{a}}{\partial t} = \mathrm{avg}(\bar{f}\bar{f}^{T})^{-1}H^{-1}e\bar{f}$$

H is a diagonal matrix, the elements along the diagonal being the amount of history that the averages are retaining (we are assuming, for convenience, that all the averages involved are retaining the same amount of history). For instance: if the averages are all ordinary averages, retaining history from $t = 0$ to the present, then:

$$H = \begin{bmatrix} t & 0 & . & . & 0 \\ 0 & t & . & . & . \\ . & . & . & . & . \\ . & . & . & t & 0 \\ 0 & . & . & 0 & t \end{bmatrix}, \quad H^{-1} = \begin{bmatrix} 1/t & 0 & . & . & 0 \\ 0 & 1/t & . & . & . \\ . & . & . & . & . \\ . & . & . & 1/t & 0 \\ 0 & . & . & 0 & 1/t \end{bmatrix}$$

This exact updating formula is inconvenient to use for Learning-Logic cells, however. To invert the matrix $\mathrm{avg}(\bar{f}\bar{f}^{T})$ in real time -- order(1) time, or O(1) time for short -- would require at least $O(p^3)$ circuit elements.

An ideal function cell would require only $O(p)$ circuit elements -- one for each internal parameter. However, I don't think that such an ideal cell can be constructed. For a while I tried to find such a cell and only succeeded in deriving exact $O(p)$ updating formulas for such restricted cases as series of orthogonal functions. The ADALINE's of Widrow, et. al., (1967) come as close to being $O(p)$ function cells as I think possible, for they were shown to converge to the exact $O(p^3)$ least-squares solution under certain circumstances, but not in general.

So, I decided to look for an $O(p^2)$ algorithm that converges to the $O(p^3)$ exact least-squares solution. There is one algorithm in particular that I was able to devise that is my favorite because it requires the fewest circuit elements of any I have seen and because of its suggestive correspondence with the neurons of our brains. Figure 6 gives the algorithm and Figure 7 is a circuit diagram for it. Later I found that there is another $O(p^2)$ algorithm described in the literature -- the Recursive Least-Squares algorithm (Ljung and Soderstrom, 1983) -- but it is not nearly as efficient in terms of circuit elements required, nor does it correspond in any particular fashion with our neurons.

To derive my favorite algorithm, we start with the $O(p^3)$ exact updating formula in its average form:

$$\frac{\partial \bar{a}}{\partial t} = U^{-1} \left[H^{-1} e \frac{\partial f_{out}}{\partial \bar{a}} - \left[\text{avg}\left(\frac{\partial f_{out}}{\partial \bar{a}} \frac{\partial f_{out}}{\partial \bar{a}}^{T}\right) - \text{diag}\left(\text{avg}\left(\frac{\partial f_{out}}{\partial \bar{a}} \frac{\partial f_{out}}{\partial \bar{a}}^{T}\right)\right) \right] \frac{\partial \bar{a}}{\partial t} \right]$$

1) Updating formula if f_{out} is non-linear.

$$\frac{\partial \bar{a}}{\partial t} = U^{-1} \left[H^{-1} e \bar{f} - \left[\text{avg}(\bar{f}\bar{f}^{T}) - \text{diag}(\text{avg}(\bar{f}\bar{f}^{T})) \right] \frac{\partial \bar{a}}{\partial t} \right]$$

2) Updating formula if f_{out} is linear.

Figure 6: My favorite updating alogithm, in its non-linear and linear forms.

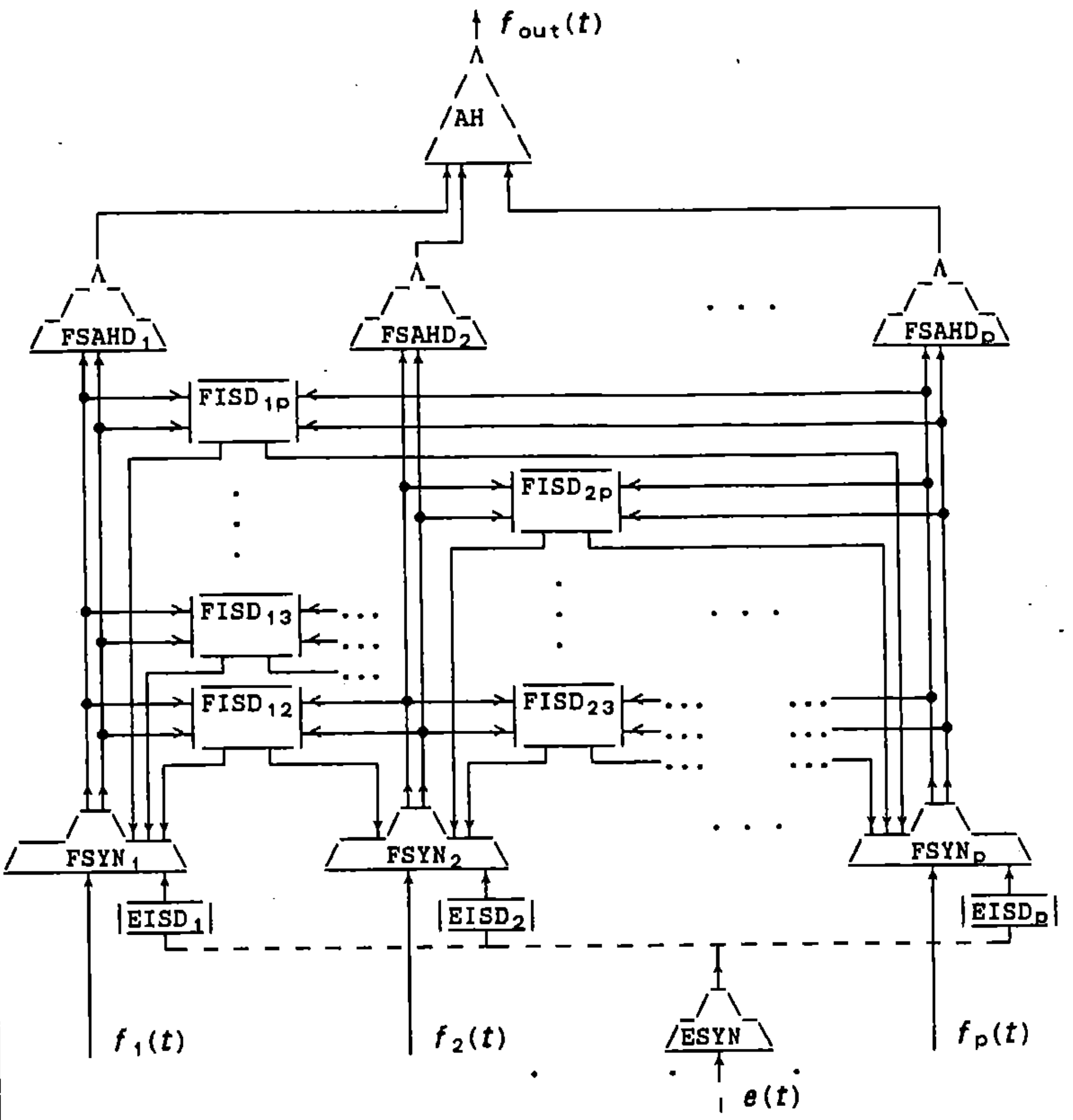
$$\frac{\partial \bar{a}}{\partial t} = \text{avg}(\bar{f}\bar{f}^{T})^{-1} H^{-1} e \bar{f}$$

If we multiply both sides by $\text{avg}(\bar{f}\bar{f}^{T})$, we get:

$$\text{avg}(\bar{f}\bar{f}^{T}) \frac{\partial \bar{a}}{\partial t} = H^{-1} e \bar{f}$$

At this point we can write out in full the components of the matrix $\text{avg}(\bar{f}\bar{f}^{T})$, the matrix H (calling the non-zero elements, which all lie on the diagonal, h_1 to h_p), the vector $\bar{f}$ and the vector $\partial\bar{a}/\partial t$:

$$\begin{array}{ccccccc}
\text{avg}(f_1 f_1) \frac{\partial a_1}{\partial t} & + & \text{avg}(f_1 f_2) \frac{\partial a_2}{\partial t} & + \ldots + & \text{avg}(f_1 f_p) \frac{\partial a_p}{\partial t} & = & h_1^{-1} e f_1 \\
\text{avg}(f_2 f_1) \frac{\partial a_1}{\partial t} & + & \text{avg}(f_2 f_2) \frac{\partial a_2}{\partial t} & + \ldots + & \text{avg}(f_2 f_p) \frac{\partial a_p}{\partial t} & = & h_2^{-1} e f_2 \\
\vdots & & \vdots & \ldots & \vdots & & \vdots \\
\text{avg}(f_p f_1) \frac{\partial a_1}{\partial t} & + & \text{avg}(f_p f_2) \frac{\partial a_2}{\partial t} & + \ldots + & \text{avg}(f_p f_p) \frac{\partial a_p}{\partial t} & = & h_p^{-1} e f_p
\end{array}$$



ESYN	Error synapse
$FSYN_i$	Function synapse i
$EISD_i$	Error to function inter-synaptic distance i
$FISD_{ij}$	Function to function inter-synaptic distance i,j
$FSAHD_i$	Function synapse to axon hillock distance i
AH	Axon hillock

Figure 7: Circuit diagram for a linear function cell based on the algorithm of Figure 6. I believe this is approximately how the pyramidal neurons of our cortex work. The subcomponents have been suggestively labeled to correspond with the diagram of a "distance-varying" neuron in Figure 1. Circuit diagrams for these subcomponents are given in Figure 8.

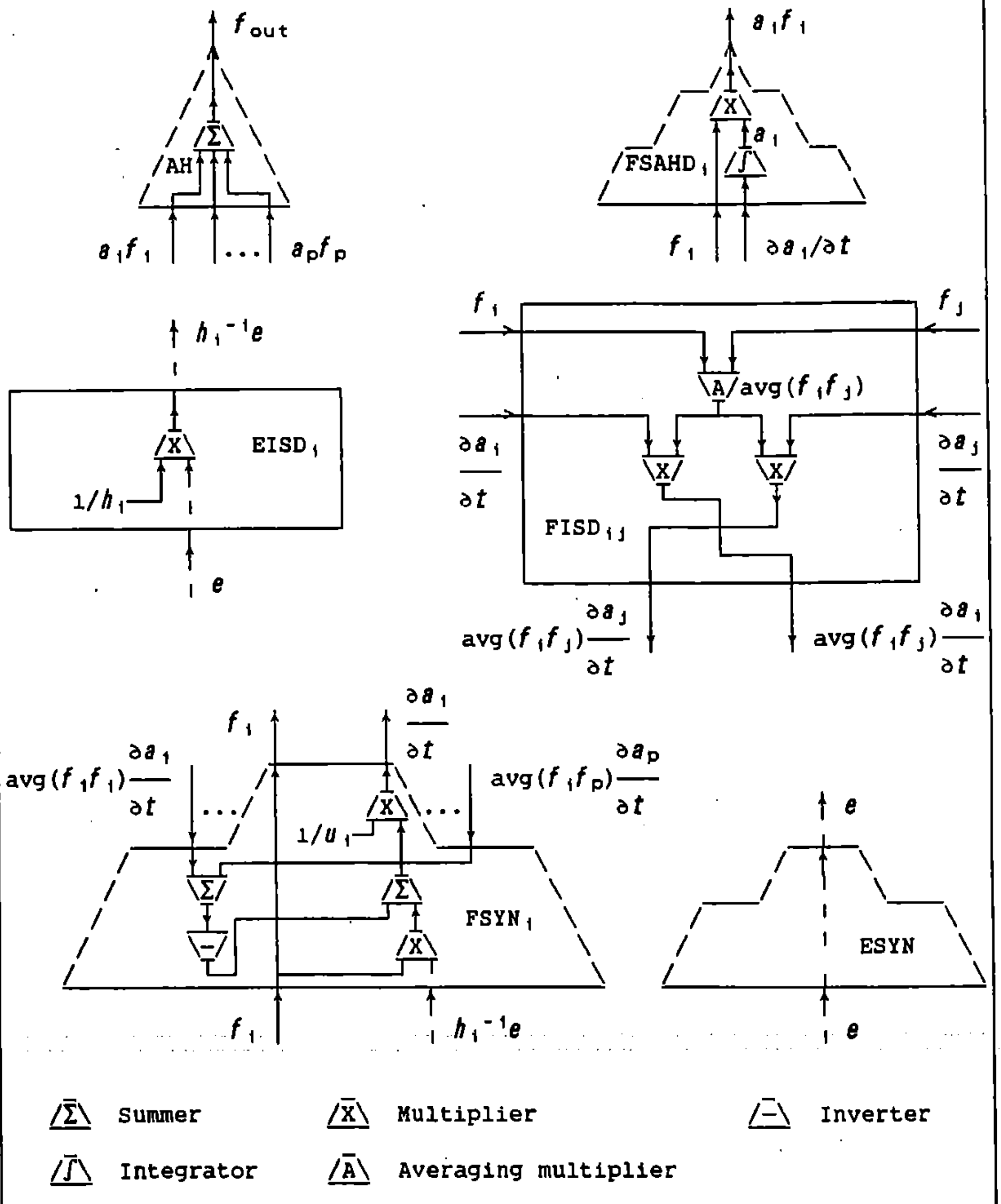


Figure 8: Circuit diagrams for the subcomponents of the linear function cell in Figure 7. Neurons implement a variation on these subcomponents (because they use pulses instead of analog voltages, may use methods other than distance to maintain some variables, etc.), but the functionality is the same (see "Questions and Answers About Neurons" for details).

Then we can solve the first equation for $\partial a_1/\partial t$, the second for $\partial a_2/\partial t$, and so on:

$$\begin{aligned}
\text{avg}(f_1f_1)\frac{\partial a_1}{\partial t} &= h_1^{-1}ef_1 \qquad\qquad -\text{avg}(f_1f_2)\frac{\partial a_2}{\partial t} - \ldots -\text{avg}(f_1f_p)\frac{\partial a_p}{\partial t} \\
\text{avg}(f_2f_2)\frac{\partial a_2}{\partial t} &= h_2^{-1}ef_2 - \text{avg}(f_2f_1)\frac{\partial a_1}{\partial at} \qquad - \ldots -\text{avg}(f_2f_p)\frac{\partial a_p}{\partial t} \\
&\ \ \vdots \\
\text{avg}(f_pf_p)\frac{\partial a_p}{\partial t} &= h_p^{-1}ef_p - \text{avg}(f_pf_1)\frac{\partial a_1}{\partial t} - \text{avg}(f_pf_2)\frac{\partial a_2}{\partial t} - \ldots
\end{aligned}$$

We can recast this in matrix-and-vector form as:

$$\text{diag}(\text{avg}(\bar{f}\bar{f}^T))\ \frac{\partial \bar{a}}{\partial t} = H^{-1}e\bar{f} - [\ \text{avg}(\bar{f}\bar{f}^T) - \text{diag}(\text{avg}(\bar{f}\bar{f}^T))\ ]\ \frac{\partial \bar{a}}{\partial t}$$

where $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))$ is a diagonal matrix consisting of the diagonal elements of $\text{avg}(\bar{f}\bar{f}^T)$. Multiplying both sides by $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))^{-1}$ gives:

$$\frac{\partial \bar{a}}{\partial t} = \text{diag}(\text{avg}(\bar{f}\bar{f}^T))^{-1} \left[H^{-1}e\bar{f} - [\ \text{avg}(\bar{f}\bar{f}^T) - \text{diag}(\text{avg}(\bar{f}\bar{f}^T))\ ]\ \frac{\partial \bar{a}}{\partial t} \right]$$

This formula for $\partial \bar{a}/\partial t$ makes for an excellent updating algorithm. If $\bar{f}$ and e change slowly compared to the speed at which $\bar{a}$ can be updated, then this method is basically doing an exact least-squares fit.

Finally, we can reach my favorite algorithm by making a slight change that reduces the number of circuit elements required, yet the algorithm will still converge to the exact least-squares solution. We will replace the matrix $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))^{-1}$ by another diagonal matrix U^{-1}:

$$\frac{\partial \bar{a}}{\partial t} = U^{-1} \left[H^{-1}e\bar{f} - [\ \text{avg}(\bar{f}\bar{f}^T) - \text{diag}(\text{avg}(\bar{f}\bar{f}^T))\ ]\ \frac{\partial \bar{a}}{\partial t} \right]$$

As we shall see later, this algorithm will converge to the exact least-squares solution as long as the non-zero elements of U (which all lie along the diagonal) are greater than the corresponding elements of $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))$. Thus, for instance, if we have an upper bound for the elements of $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))$ then we can replace it by a constant matrix.

If U^{-1} is sufficiently small and if our updating mechanism is sufficiently quick, then we can solve for $\partial \bar{a}/\partial t$ to get:

$$\frac{\partial \bar{a}}{\partial t} = [\ \mathrm{avg}(\vec{f}\vec{f}^{T}) - \mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T})) + U\ ]^{-1}\ H^{-1} e \vec{f}$$

Comparing this with the exact updating formula:

$$\frac{\partial \bar{a}}{\partial t} = \mathrm{avg}(\vec{f}\vec{f}^{T})^{-1} H^{-1} e \vec{f}$$

we see that essentially all we are doing is replacing the diagonal of $\mathrm{avg}(\vec{f}\vec{f}^{T})$.

Not only does this slight change reduce the number of circuit elements required, but it makes it possible for the algorithm to then correspond to real neurons. Assuming that our neurons are maintaining all variables as distances, then each element of $\bar{a}$ corresponds to the distance between a function synapse (i.e. a synapse made by another function cell) and the axon hillock. Similarly, each off-diagonal term of $\mathrm{avg}(\vec{f}\vec{f}^{T})$ corresponds to the distance between two function synapses (remembering that there are really only $\frac{1}{2}(p^2-p)$ different values in $\mathrm{avg}(\vec{f}\vec{f}^{T})-\mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T}))$ because it is a symmetrix matrix with zeros along the diagonal, and that this number is exactly the number of function inter-synaptic distances between p function synapses). Each term of H^{-1} corresponds to the distance between each function synapse and the error synapse, and each element of U^{-1} governs the rate at which a function synapse can change its distance from the axon hillock. See "Questions and Answers About Our Neurons" for more details.

To show that this algorithm converges to the exact least-squares solution, we can start from the expressions for the derivatives (primed variables represent the exact values in the following):

$$\text{exact:} \quad \frac{\partial \bar{a}'}{\partial t} = \mathrm{avg}(\vec{f}\vec{f}^{T})^{-1}\ H^{-1} e' \vec{f}$$

$$\text{approximate:} \quad \frac{\partial \bar{a}}{\partial t} = [\ \mathrm{avg}(\vec{f}\vec{f}^{T}) - \mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T})) + U\ ]^{-1}\ H^{-1} e \vec{f}$$

Rewriting e' and e as $d-\vec{f}^{T}\bar{a}'$ and $d-\vec{f}^{T}\bar{a}$, respectively, and rearranging things a bit gives:

$$\mathrm{avg}(\vec{f}\vec{f}^{T})\ \frac{\partial \bar{a}'}{\partial t} = H^{-1} d\vec{f} - H^{-1}\vec{f}\vec{f}^{T}\bar{a}'$$

$$[\ \mathrm{avg}(\vec{f}\vec{f}^{T}) - \mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T})) + U\ ]\ \frac{\partial \bar{a}}{\partial t} = H^{-1} d\vec{f} - H^{-1}\vec{f}\vec{f}^{T}\bar{a}$$

Finally, we can take the difference between these two equations and solve for $\partial\bar{a}/\partial t$:

$$\frac{\partial \bar{a}}{\partial t} = [\ \mathrm{avg}(\vec{f}\vec{f}^{T})-\mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T}))+U\ ]^{-1}\ \mathrm{avg}(\vec{f}\vec{f}^{T})\ \frac{\partial \bar{a}'}{\partial t}$$

$$+\ [\ \mathrm{avg}(\vec{f}\vec{f}^{T})-\mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T}))+U\ ]^{-1}\ H^{-1}\vec{f}\vec{f}^{T}\ (\bar{a}'-\bar{a})$$

We are left with an expression for $\partial\bar{a}/\partial t$ with two terms: one involving $\partial\bar{a}'/\partial t$, and the other involving $\bar{a}'-\bar{a}$.

Of the two terms, the first one is the least important. If the exact algorithm converges then beyond some point the expected value of $\partial\bar{a}'/\partial t$ will be 0. From then on the first term will, on average, contribute nothing to $\partial\bar{a}/\partial t$ and so can be ignored. If the exact alogithm doesn't converge (i.e. d isn't stationary within the amount of history we are retaining) then the contribution to $\partial\bar{a}/\partial t$ from the first term will at least be within 90° of $\partial\bar{a}'/\partial t$ if U is chosen as discussed below.

The second term, involving $\bar{a}'-\bar{a}$, is the more important of the two. Since H and $\vec{f}\vec{f}^{T}$ are already positive semi-definite, the contribution to $\partial\bar{a}/\partial t$ from the second term will cause convergence (i.e. will decrease $\bar{a}'-\bar{a}$) if we can ensure that

$$\mathrm{avg}(\vec{f}\vec{f}^{T})\ -\ \mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^{T}))\ +\ U$$

is also positive semi-definite. This leads to our requirement that the elements of U must be greater than or equal to the elements along the diagonal of $\mathrm{avg}(\vec{f}\vec{f}^{T})$.

Although the $O(p^2)$ approximate updating algorithm I have described here is my favorite, because of its efficiency and the suggestive way in which the mathematical elements of the algorithm can be associated with the physical elements of real neurons, it is not the only $O(p^2)$ approximate updating algorithm. Trivially, various multiplicative factors can be inserted in the above algorithm. Non-trivially, another algorithm I have used with success is, in its non-linear form:

$$\frac{\partial \bar{a}}{\partial t} = U^{-1}\left[\ \mathrm{avg}\left(f_{out}\frac{\partial f_{out}}{\partial \bar{a}}\right) + \mathrm{avg}\left(e\frac{\partial f_{out}}{\partial \bar{a}}\right) - \mathrm{avg}\left(\frac{\partial f_{out}}{\partial \bar{a}}\frac{\partial f_{out}}{\partial \bar{a}}^{T}\right)\bar{a}\ \right]$$

and in its linear form:

$$\frac{\partial \bar{a}}{\partial t} = U^{-1}\ [\ \mathrm{avg}(f_{out}\vec{f}) + \mathrm{avg}(e\vec{f}) - \mathrm{avg}(\vec{f}\vec{f}^{T})\bar{a}\ ]$$

This formula doesn't work as well in the non-linear case, however, because it depends on the values of the internal parameters and not just on their derivatives.

HOW BASIC LEARNING-LOGIC CELLS WORK

To connect function cells into a network, we need to add a little bit of extra circuitry to each cell. This added circuitry upgrades function cells to <u>basic Learning-Logic cells</u>. Figure 9 is an example of a network of basic Learning-Logic cells, and Figure 10 shows their internal structure.

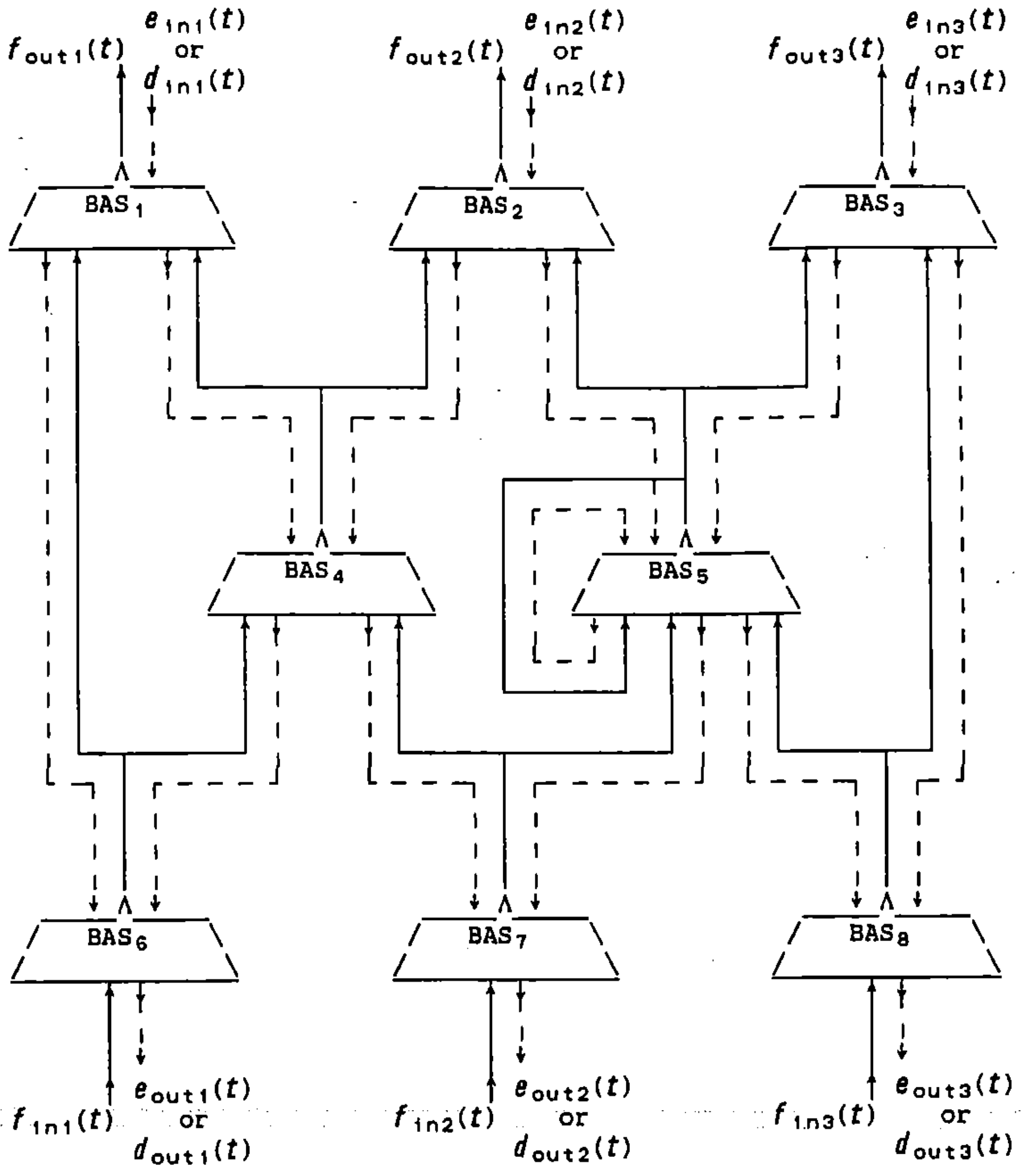
For basic Learning-Logic cells, the training signals propagated from cell to cell can be either error signals, e, or desired function signals, d. These correspond to training by correction or training by example. Slightly different cells need to be constructed for each case, the difference being addition and subtraction units that need to be added to each cell for the d-type cells. Since the e-type cells are the more basic of the two, and since the d-type are easily derived from them, we will work exclusively with the e-type cells. Besides, I believe the neurons in our brains are mostly e-type cells. Not to imply that networks of e-type cells can't learn by example: they can if the error signals depend on the function outputs, f, from the network (e.g. if $e = d-f$).

The method we will use to connect the function cells together is based upon the following observation: when learning the new, it is good to remember the old. Speaking mathematically this means that when converging to a solution, we should try to minimize the changes to the internal parameters.

We will now derive an algorithm for connecting function cells together that approximately minimizes the sum of the squares of the changes to the internal parameters at any instant.

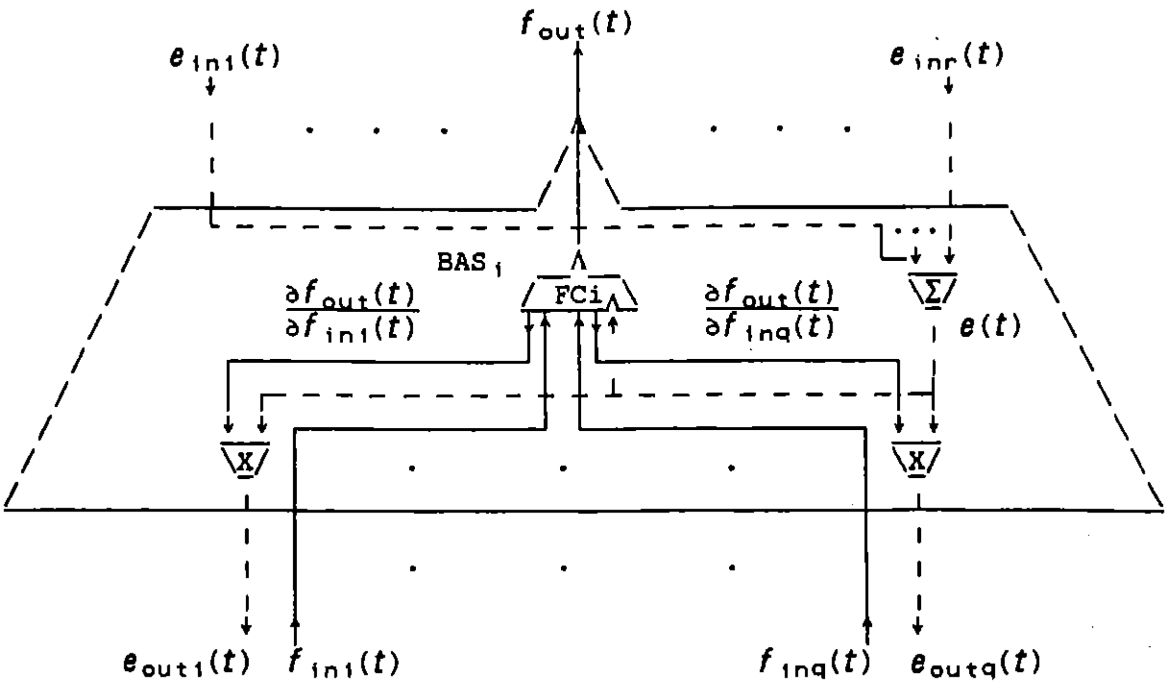
Consider the sample networks of Figure 11. For simplicity, we'll make a few assumptions that won't change the conclusion we reach but that will make the derivation less cluttered. We'll assume:

1. that the input signal f_4 remains constant while we adjust the internal parameters a_1, a_2 and a_3.

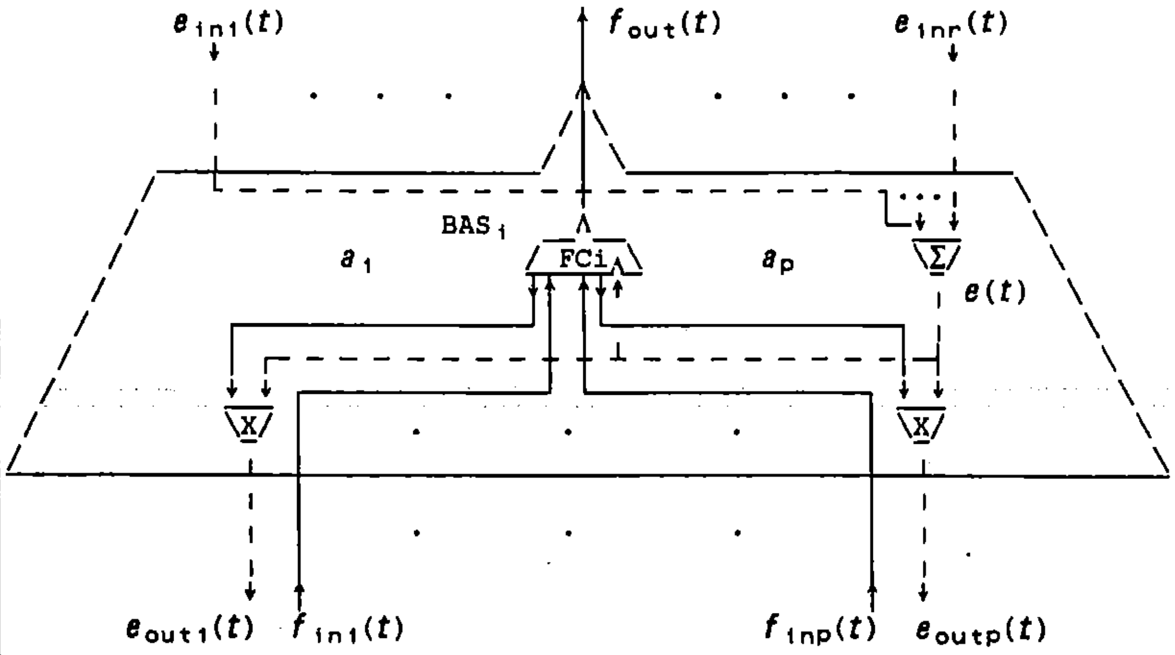

$f_{out1}(t)$
$e_{in1}(t)$ or $d_{in1}(t)$
$f_{out2}(t)$
$e_{in2}(t)$ or $d_{in2}(t)$
$f_{out3}(t)$
$e_{in3}(t)$ or $d_{in3}(t)$
BAS_1
BAS_2
BAS_3
BAS_4
BAS_5
BAS_6
BAS_7
BAS_8
$f_{in1}(t)$
$e_{out1}(t)$ or $d_{out1}(t)$
$f_{in2}(t)$
$e_{out2}(t)$ or $d_{out2}(t)$
$f_{in3}(t)$
$e_{out3}(t)$ or $d_{out3}(t)$

——— Function signals $\quad$ BAS_i Basic Learning-Logic cell i
– – – – Training signals

Figure 9: An example of a network of basic Learning-Logic cells. The training signals for a network of basic Learning-Logic cells can either be error signals, *e*, or desired function signals, *d*, depending on how the cells are constructed.

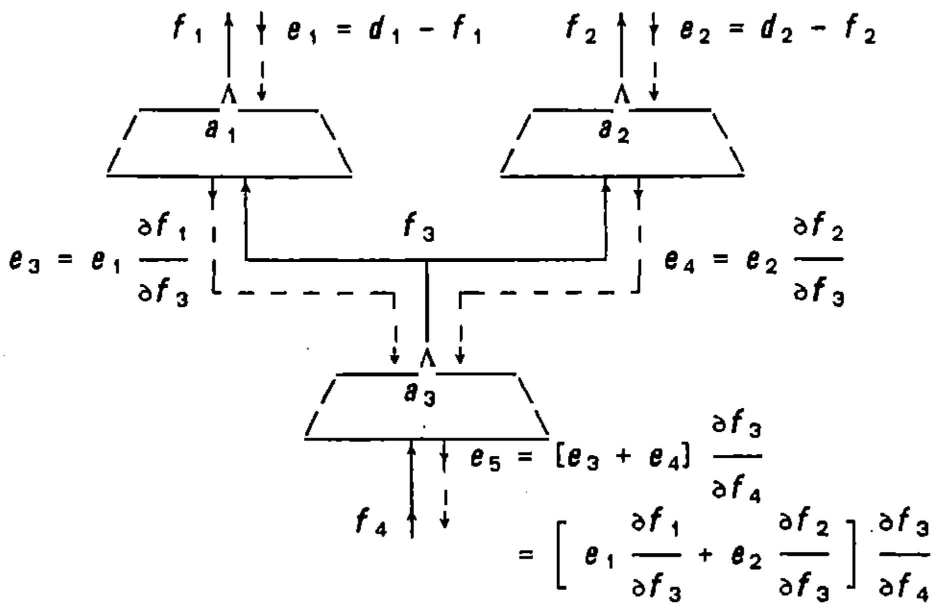


1) Non-linear basic Learning-Logic cell.

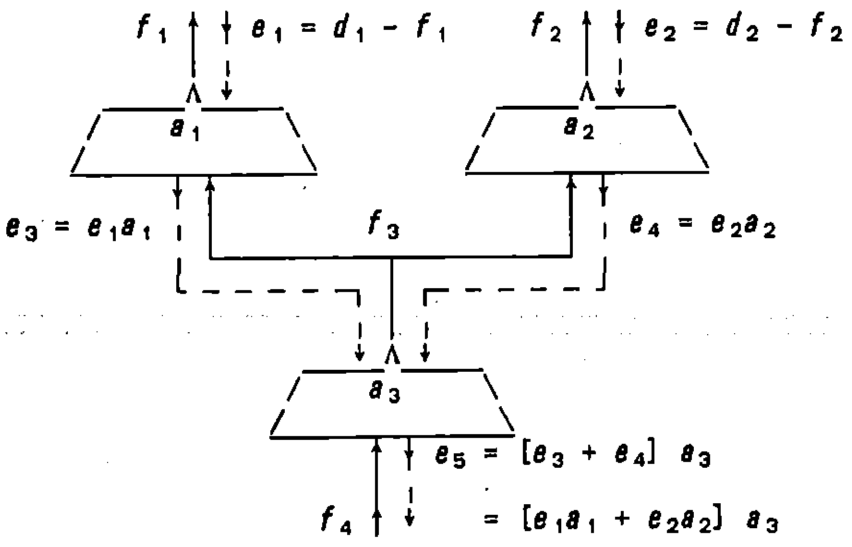


2) Linear basic Learning-Logic cell.

Figure 10: Diagram of a basic Learning-Logic cell. Note that we have pulled out some signals that we need from inside the function cell.


f_1
$e_1 = d_1 - f_1$
f_2
$e_2 = d_2 - f_2$
a_1
a_2
$e_3 = e_1 \frac{\partial f_1}{\partial f_3}$
f_3
$e_4 = e_2 \frac{\partial f_2}{\partial f_3}$
a_3
$e_5 = [e_3 + e_4] \frac{\partial f_3}{\partial f_4}$
f_4
$= \left[e_1 \frac{\partial f_1}{\partial f_3} + e_2 \frac{\partial f_2}{\partial f_3} \right] \frac{\partial f_3}{\partial f_4}$

1) Non-linear sample network: $f_1 = f_1(a_1, f_3)$, etc.



2) Linear sample network: $f_1 = a_1 f_3$, etc.

Figure 11: Small sample networks used to illustrate the calculation of the error signals.

2. that the desired functions d_1 and d_2 remain constant while we adjust the internal parameters.

3. that U^{-1} is small enough so that we can make the following approximation to the updating algorithm in Figure 6:

   <u>Non-linear case</u> <u>Linear case</u>

$$\frac{\partial \bar{a}}{\partial t} \simeq U^{-1}H^{-1}e\,\frac{\partial \bar{f}}{\partial \bar{a}} \qquad = U^{-1}H^{-1}e\bar{f}$$

   This essentially converts the cells into the ADALINE's that were studied by Widrow, et. al., (1967) and which were shown to converge under certain circumstances (and, incidentally, the following algorithm can be used to connect ADALINE's into networks, although the networks aren't as powerful as Learning-Logic networks).

Our strategy will be to reduce the errors e_1 and e_2 in Figure 11 to zero in a step-like fashion, at each step making small changes Δa_1, Δa_2 and Δa_3 in the internal parameters a_1, a_2 and a_3. The amounts Δe_1 and Δe_2 that we thus reduce e_1 and e_2 by are approximately:

<u>Non-linear case</u> <u>Linear case</u>

$$-\Delta e_1 = \Delta f_1 = \frac{\partial f_1}{\partial a_1}\Delta a_1 + \frac{\partial f_1}{\partial f_3}\Delta f_3 = f_3\Delta a_1 + a_1\Delta f_3$$

$$= \frac{\partial f_1}{\partial a_1}\Delta a_1 + \frac{\partial f_1}{\partial f_3}\frac{\partial f_3}{\partial a_3}\Delta a_3 = f_3\Delta a_1 + a_1 f_4\Delta a_3$$

$$-\Delta e_2 = \Delta f_2 = \frac{\partial f_2}{\partial a_2}\Delta a_2 + \frac{\partial f_2}{\partial f_3}\Delta f_3 = f_3\Delta a_2 + a_2\Delta f_3$$

$$= \frac{\partial f_2}{\partial a_2}\Delta a_2 + \frac{\partial f_2}{\partial f_3}\frac{\partial f_3}{\partial a_3}\Delta a_3 = f_3\Delta a_2 + a_2 f_4\Delta a_3$$

Even though Δe_1 and Δe_2 depend on Δa_1, Δa_2 and Δa_3, we can pretend that we were given Δe_1 and Δe_2 first. Then the above two equations can be viewed as constraints, where our object at each step is to find the Δa_1, Δa_2 and Δa_3 which minimize:

$$\Delta a_1^2 + \Delta a_2^2 + \Delta a_3^2$$

We can solve this constrained minimization problem using the method of Lagrange multipliers. If we let λ_1 and λ_2 be the Lagrange multipliers for the 2 constraint equations, we will end up with 5 equations in 7 unknowns:

Non-linear case | Linear case

$$\Delta a_1 = \tfrac{1}{2}\lambda_1 \frac{\partial f_1}{\partial a_1} = \tfrac{1}{2}\lambda_1 f_3$$

$$\Delta a_2 = \tfrac{1}{2}\lambda_2 \frac{\partial f_2}{\partial a_2} = \tfrac{1}{2}\lambda_2 f_3$$

$$\Delta a_3 = \tfrac{1}{2}\left[\lambda_1 \frac{\partial f_1}{\partial f_3} + \lambda_2 \frac{\partial f_2}{\partial f_3}\right]\frac{\partial f_3}{\partial a_3} = \tfrac{1}{2}\left[\lambda_1 a_1 + \lambda_2 a_2\right] f_4$$

$$-\Delta e_1 = \frac{\partial f_1}{\partial a_1}\Delta a_1 + \frac{\partial f_1}{\partial f_3}\frac{\partial f_3}{\partial a_3}\Delta a_3 = f_3\Delta a_1 + a_1 f_4 \Delta a_3$$

$$-\Delta e_2 = \frac{\partial f_2}{\partial a_2}\Delta a_2 + \frac{\partial f_2}{\partial f_3}\frac{\partial f_3}{\partial a_3}\Delta a_3 = f_3\Delta a_2 + a_2 f_4 \Delta a_3$$

Switching perspectives again, we will assume that Δe_1, Δe_2, Δa_1, Δa_2 and Δa_3 are all variables which depend on λ_1 and λ_2. We see then that we can choose λ_1 and λ_2 at will with the assurance that no matter what Δe_1 and Δe_2 turn out to be, the corresponding $\Delta a_1^2 + \Delta a_2^2 + \Delta a_3^2$ will be the minimum that could account for them. Thus, if we choose λ_1's and λ_2's wisely so that successive Δe_1's and Δe_2's reduce e_1 and e_2 to zero, then we have converged the network while at the same time minimizing the changes to the internal parameters at each step.

In particular: under assumptions 1 and 2 the approximate algorithm of assumption 3 will cause the cells to converge (except in a few degenerate situations), so we can set $\tfrac{1}{2}\lambda_1$ to $U^{-1}H^{-1}e_1$ and $\tfrac{1}{2}\lambda_2$ to $U^{-1}H^{-1}e_2$ to get:

Non-linear case | Linear case

$$\Delta a_1 = U^{-1}H^{-1}e_1 \frac{\partial f_1}{\partial a_1} = U^{-1}H^{-1}e_1 f_3$$

$$\Delta a_2 = U^{-1}H^{-1}e_2 \frac{\partial f_2}{\partial a_2} = U^{-1}H^{-1}e_2 f_3$$

$$\Delta a_3 = U^{-1}H^{-1}\left[e_1 \frac{\partial f_1}{\partial f_3} + e_2 \frac{\partial f_2}{\partial f_3}\right]\frac{\partial f_3}{\partial a_3} = U^{-1}H^{-1}\left[\, e_1 a_1 + e_2 a_2 \,\right] f_4$$

From these equations we see, as in Figure 11, that e_3 should be set to $e_1 * \partial f_1/\partial f_3$, e_4 should be set to $e_2 * \partial f_2/\partial f_3$, and that the cell containing a_3 should treat the sum of these two as the total error that it should correct for.

The argument above can be extended to networks of any number of cells. The only requirement is that each cell must send the appropriate error signal to each of its input cells. That is, the connection between two cells must be bi-directional, with function signals going one direction and error signals the opposite.

HOW ERROR CELLS WORK

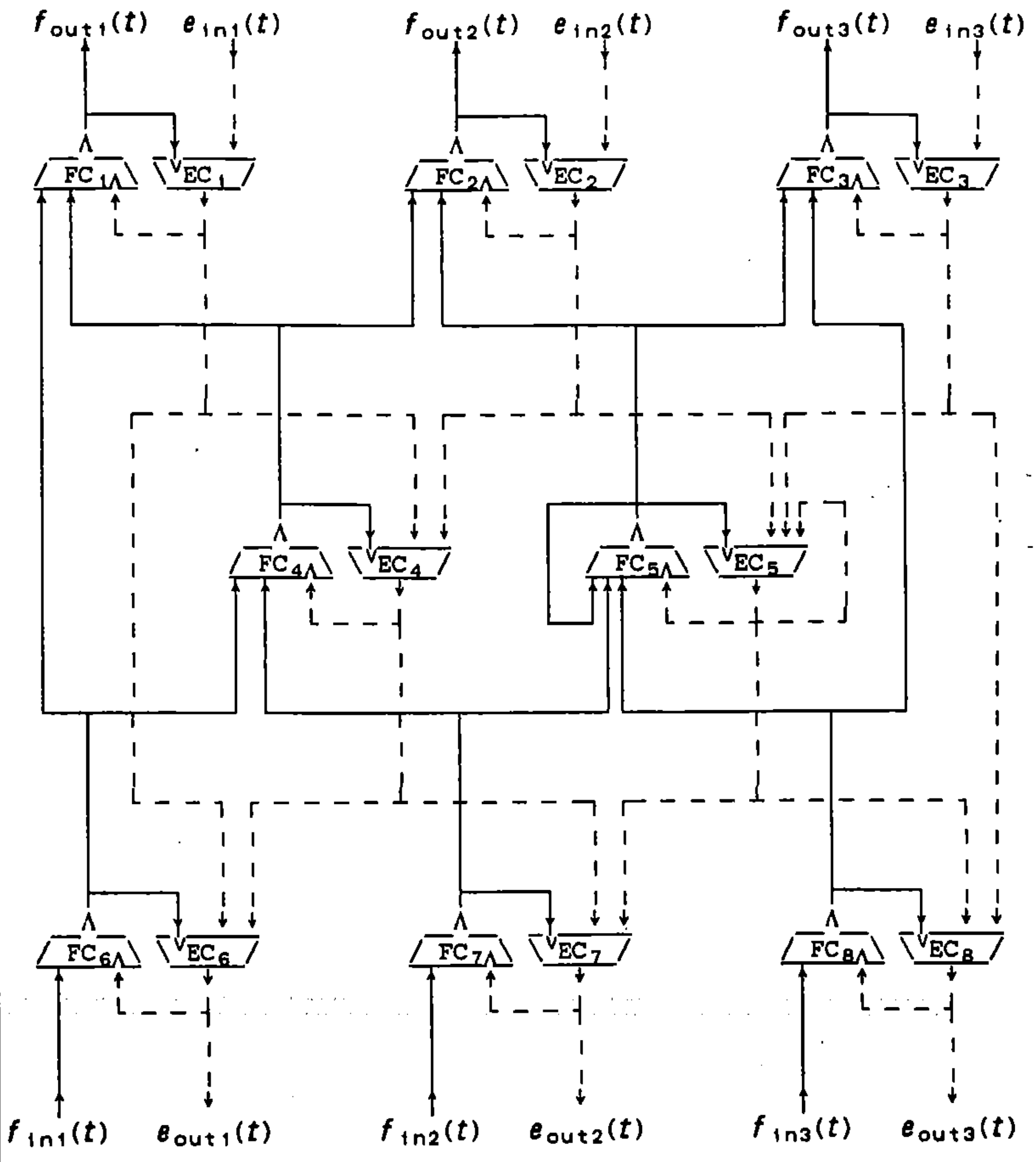
Basic Learning-Logic cells aren't accurate models of the neurons of our cortex. They require bi-directional conduction of the function and error signals, whereas real neurons conduct signals in only one direction.

We can, however, pair each function cell with an error cell to form networks that very much resemble our cortical neurons (Figure 12 is an example of such a network). I believe that error cells correspond to the stellate neurons.

Each pair of function and error cells approximates the behavior of a basic Learning-Logic cell -- in fact, one can enclose each pair in a box to form approximate Learning-Logic cells (see Figure 17). The network of approximate Learning-Logic cells in Figure 18 is identical to the network of function cell-error cell pairs in Figure 12. Both in turn are approximately equivalent to the network of basic Learning-Logic cells in Figure 9.

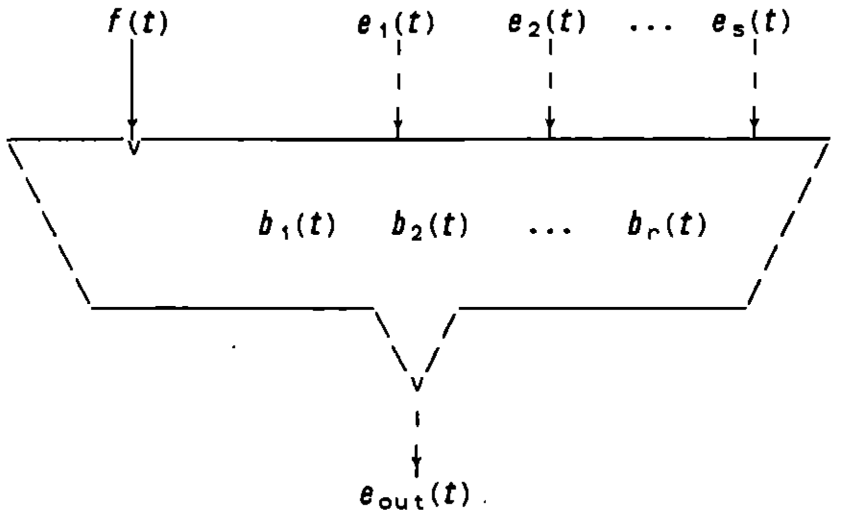
The purpose of an error cell is to decide what the total error e_{out} should be for its associated function cell. To make its decision, it is given the output f of its associated function cell (see Figure 13) and, for all the function cells which receive the output of its associated function cell, it receives their total errors $e_1,\ldots,e_s$.

To derive the algorithm used by the error cells, we will work with a specific example -- the three pairs of cells in the top left-hand quarter of Figure 12. We will derive the algorithm used by EC_4. EC_4 is given the output of FC_4 and, because FC_1 and FC_2 receive the output of FC_4, EC_4 also receives the total errors from EC_1 and EC_2. In what



———	Function signals	FC_i	Function cell i
– – – –	Error signals	EC_i	Error cell i

Figure 12: An example of a network of function and error cells, corresponding, I believe, to the pyramidal and stellate neurons.



$$e_{out}(t) = e_{out}(\, b_1(t),\ldots,b_r(t),e_1(t),\ldots,e_s(t)\,)$$

Figure 13: "Black box" diagram of an error cell.

follows we will use FC_4 to mean either the cell labeled FC_4 or the output of cell FC_4, etc.

From "How Basic Learning-Logic Cells Work" we know that optimally the output of EC_4 should be:

$$EC_4 = \frac{\partial FC_1}{\partial FC_4} EC_1 + \frac{\partial FC_2}{\partial FC_4} EC_2$$

Unfortunately EC_4 has no knowledge of what $\partial FC_1/\partial FC_4$ and $\partial FC_2/\partial FC_4$ are because these are maintained in FC_1 and FC_2, respectively (these are the signals that basic Learning-Logic cells pull out of the function cells, as in Figure 10).

So, EC_4 must try its best to estimate what these derivatives are. Letting b_1 be the estimate for $\partial FC_1/\partial FC_4$ and b_2 be the estimate for $\partial FC_2/\partial FC_4$, then the output of EC_4 will be:

$$EC_4 = b_1 EC_1 + b_2 EC_2$$

Let's start with b_1. The desired function d_1 for cell FC_1 can be written as:

$$\frac{\partial b}{\partial t} = U^{-1}H^{-1}f\,\frac{\partial e_{out}}{\partial b}$$

1) Updating formula if e_{out} is non-linear.

$$\frac{\partial b}{\partial t} = U^{-1}H^{-1}f\bar{e}$$

2) Updating formula if e_{out} is linear.

Figure 14: Updating algorithm for an error cell. The diagonal matrices U^{-1} and H^{-1} can be combined into a single diagonal matrix, but I kept both of them in the formula because they arose in separate ways.

$$d_1 = EC_1 + FC_1$$

or equivalently:

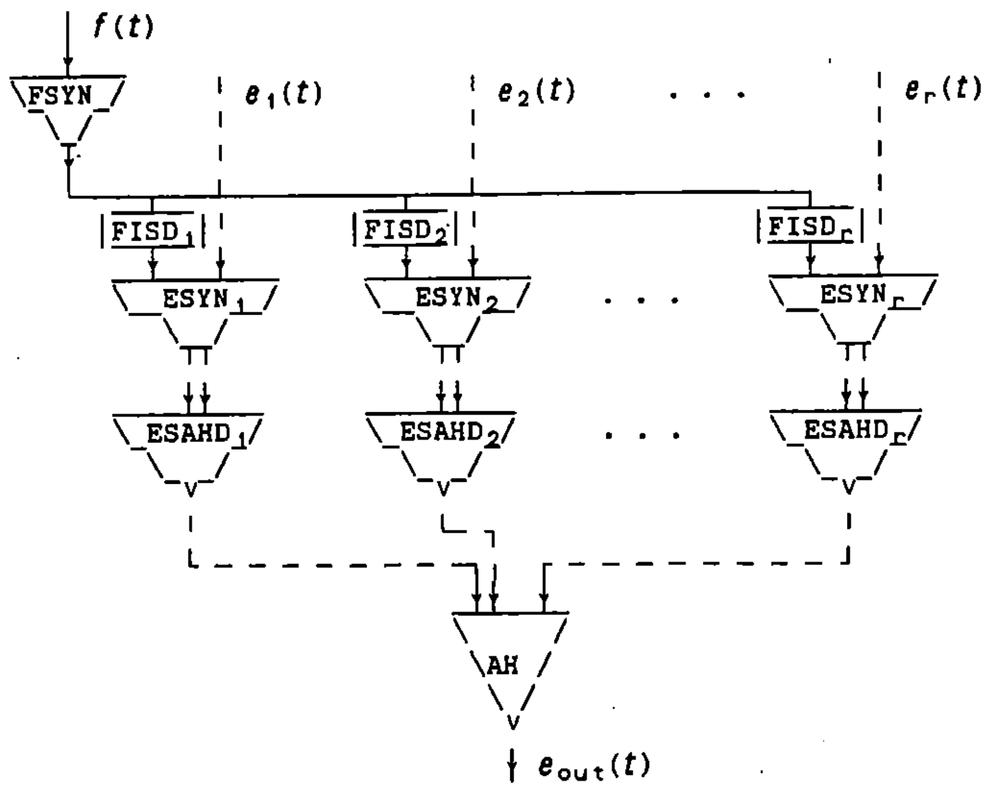
$$EC_1 = d_1 - FC_1$$

FC_1 is a function of many variables, but we know that at least one of them is FC_4. So, we can linearize FC_1 around the current value of FC_4 to get:

$$\begin{aligned} EC_1 &= d_1 - FC_1(FC_4(t_0),\ldots) - \frac{\partial FC_1(FC_4(t_0),\ldots)}{\partial FC_4(t_0)}(FC_4(t)-FC_4(t_0)) \\ &= d_1 - FC_1(FC_4(t_0),\ldots) + \frac{\partial FC_1(FC_4(t_0),\ldots)}{\partial FC_4(t_0)}FC_4(t_0) \\ &\quad - \frac{\partial FC_1(FC_4(t_0),\ldots)}{\partial FC_4(t_0)}FC_4(t) \end{aligned}$$

Further, we'll assume that $\partial FC_1(t)/\partial FC_4(t)$ stays fairly close to $\partial FC_1(FC_4(t_0),\ldots)/\partial FC_4(t_0)$ for a sufficiently useful interval, so that we can write:

$$EC_1 = d_1 - FC_{1junk} - \frac{\partial FC_1}{\partial FC_4}FC_4$$



$ESYN_i$	Error synapse i
FSYN	Function synapse
$FISD_i$	Function to error inter-synaptic distance i
$ESAHD_i$	Error synapse to axon hillock distance i
AH	Axon hillock

Figure 15: Circuit diagram for a linear error cell based on the algorithm of Figure 14. I believe this is approximately how the stellate neurons of our cortex work. The subcomponents have been suggestively labeled to correspond with the diagram of a "distance-varying" neuron in Figure 1. Circuit diagrams for these subcomponents are given in Figure 16.

where:

$$FC_{1junk} = FC_1(FC_4(t_0),\ldots) - \frac{\partial FC_1(FC_4(t_0),\ldots)}{\partial FC_4(t_0)} FC_4(t_0)$$

Now, we know that FC_1 is adjusting itself so as to minimize:

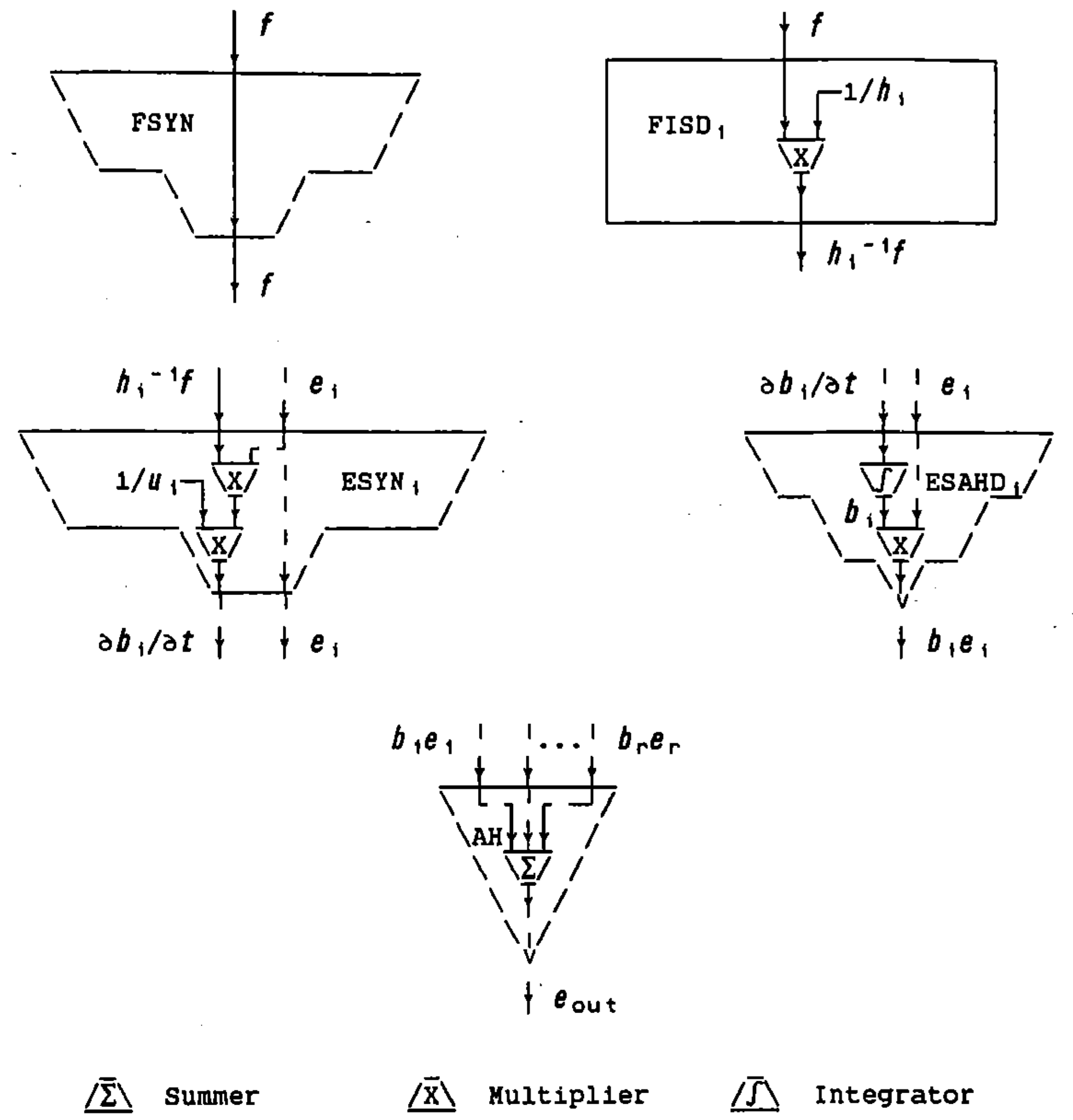


Figure 16: Circuit diagrams for the subcomponents of the linear error cell in Figure 15.

$$\int_0^t EC_1{}^2 \, d\tau = \int_0^t \left(d_1 - FC_{1junk} - \frac{\partial FC_1}{\partial FC_4} FC_4 \right)^2 d\tau$$

So, we can get a good estimate of $\partial FC_1/\partial FC_4$ by adjusting b_1 to minimize the same thing:

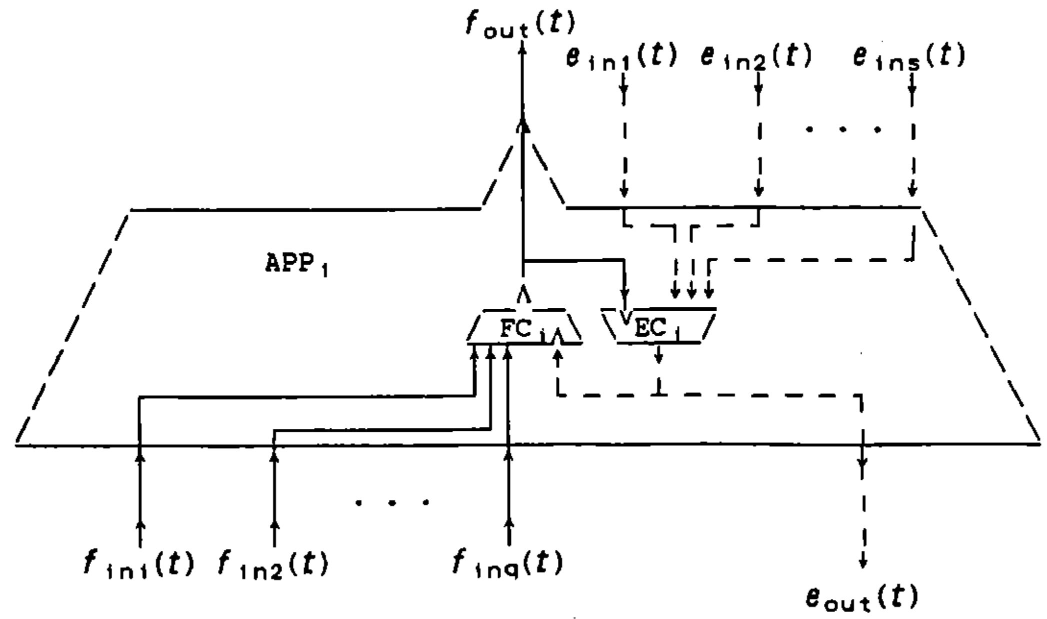


Figure 17: An approximate Learning-Logic cell, which is nothing more than a pair of function and error cells drawn in a box.

$$\int_0^t EC_1^2\, d\tau = \int_0^t (\, d_1 - FC_{1junk} - b_1 FC_4 \,)^2\, d\tau$$

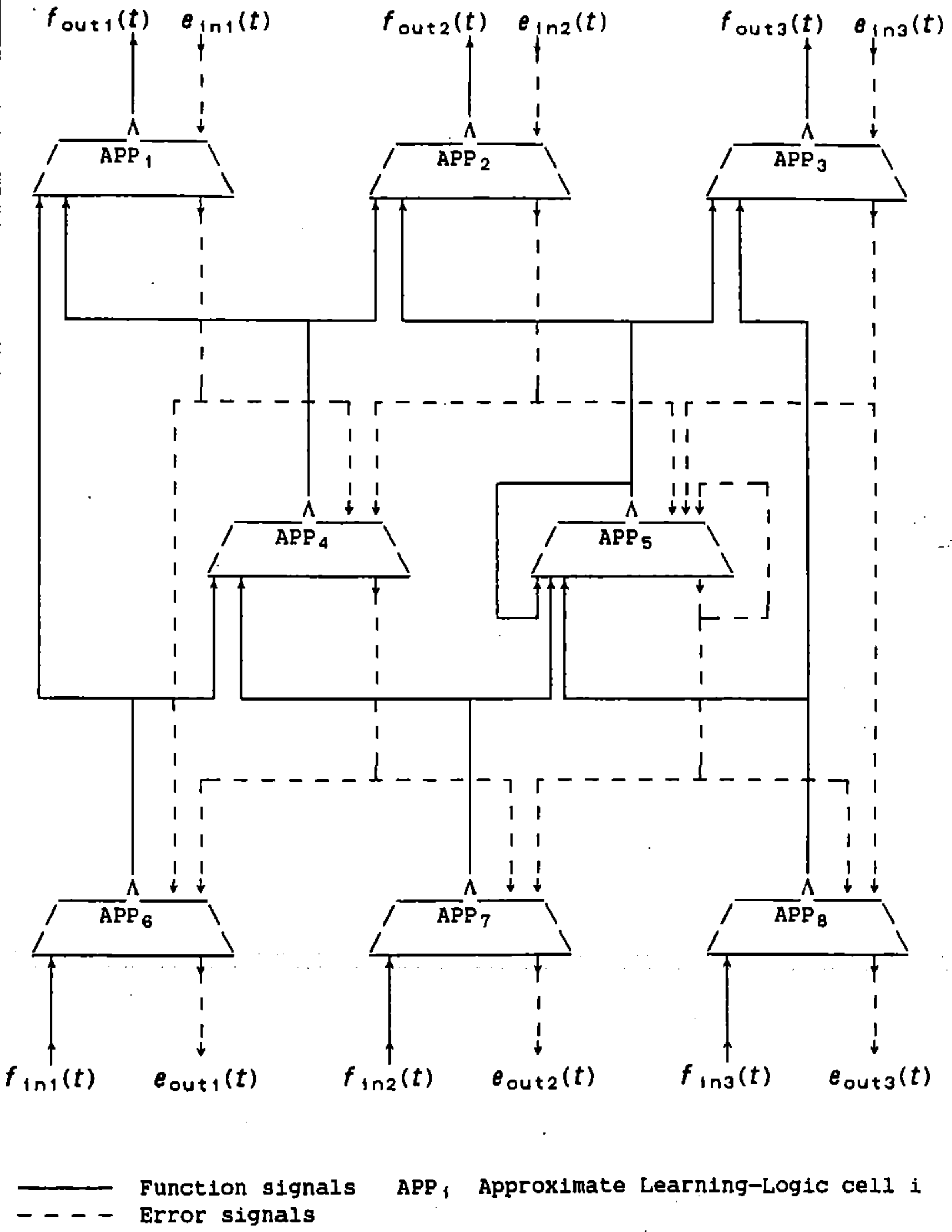
This is a simple least-squares problem in only one variable, and the well-known solution for b_1 is:

$$b_1 = \left[\int_0^t FC_4^2\, d\tau \right]^{-1} \int_0^t (\, d_1 - FC_{1junk} \,) FC_4\, d\tau$$

Scaling everything by $1/t$, we arrive at the equivalent average formula:

$$b_1 = \mathrm{avg}(FC_4^2)^{-1} \mathrm{avg}((d_1 - FC_{1junk}) FC_4)$$

However, the error cell can't use this formula to calculate b_1 because it isn't given d_1 or FC_{1junk}. From earlier in the derivation, though, we know that:



<u>Figure 18:</u> An example of a network of approximate Learning-Logic cells.

$$d_1 - FC_{1junk} = EC_1 + \frac{\partial FC_1}{\partial FC_4} FC_4$$

Making this substitution gives us:

$$b_1 = \text{avg}(FC_4{}^2)^{-1}\text{avg}((EC_1 + \frac{\partial FC_1}{\partial FC_4} FC_4)FC_4)$$

We're closer, but this formula still can't be used because the error cell wasn't given the past values of $\partial FC_1/\partial FC_4$ which are needed to calculate the average. Instead, though, we can use the past values of b_1 -- which is our estimator for $\partial FC_1/\partial FC_4$ -- to get:

$$b_1 = \text{avg}(FC_4{}^2)^{-1}\text{avg}((EC_1 + b_1 FC_4)FC_4)$$

The error cell could now use this formula because either it is given or already contains all of the necessary information. We will find it simplifies things, though, if we take the $\partial/\partial t$ to arrive at the following updating formula:

$$\frac{\partial b_1}{\partial t} = \text{avg}(FC_4{}^2)^{-1} h_1{}^{-1} FC_4 EC_1$$

where h_1 is the amount of history that was involved in the average. This is also a good formula to use, but we are going to simplify it even further. In "How Function Cells Work" we saw we could replace the diagonal matrix $\text{diag}(\text{avg}(\bar{f}\bar{f}^T))^{-1}$ by another diagonal matrix U^{-1}. Similarly, we can replace $\text{avg}(FC_4{}^2)^{-1}$ by any $u_1{}^{-1}$ where $u_1 > \text{avg}(FC_4{}^2)$:

$$\frac{\partial b_1}{\partial t} = u_1{}^{-1} h_1{}^{-1} FC_4 EC_1$$

If we went through the same process for b_2 we would find:

$$\frac{\partial b_2}{\partial t} = u_2{}^{-1} h_2{}^{-1} FC_4 EC_2$$

These formulas can be generalized because the same reasoning applies to any error cell. We can write b_1 and b_2 as the vector $\bar{b}$, EC_1 and EC_2 as the vector $\bar{e}$, FC_4 as f, h_1 and h_2 as the diagonal matrix H, and u_1 and u_2 as the diagonal matrix U (see Figure 13 and Figure 14):

$$\frac{\partial \bar{b}}{\partial t} = U^{-1} H^{-1} f \bar{e}$$

Of course, we can combine the two diagonal matrices U^{-1} and H^{-1} into a single diagonal matrix, but since they arose for different reasons we'll leave them as they are. In fact, real Learning-Logic networks -- including our neurons -- would probably have at least these two multiplicative factors and possibly more.

This updating formula is mathematically identical to the algorithm used by Widrow, et. al., (1967) for their ADALINE's, although they derived it in a different context. Many others have probably derived mathematically identical algorithms, too.

It is interesting to note the similarities between the algorithms used by the function and error cells. The error cell algorithm (Figure 14) is the limiting case of the function cell algorithm (Figure 6) in at least three cases:

1. When U^{-1} in the function cell algorithm becomes very small -- as noted in "How Basic Learning-Logic Cells Work".

2. When $\mathrm{avg}(\vec{f}\vec{f}^T)-\mathrm{diag}(\mathrm{avg}(\vec{f}\vec{f}^T))$ in the function cell algorithm equals zero.

3. When there is only one input to the function cell.

In fact, really the only difference between a function cell and an error cell is that the function cell can accumulate correlations between its inputs. Since the pyramidal cells presumably evolved from more primitive cells (maybe even the stellate cells), it is comforting to know that not much of a change was necessary.

HOW GENERAL LEARNING-LOGIC CELLS WORK

More general Learning-Logic cells can be constructed. However, the principles and algorithms used are all simple extensions of those used in making function cells, error cells, basic Learning-Logic cells and approximate Learning-Logic cells, so I won't bother discussing more general Learning-Logic cells except to show you a picture of one (Figure 19).

I don't imagine that more general Learning-Logic cells will be used much, since they can be replaced by networks of simpler cells. However, one advantage a more general cell has is that it may require fewer components because it needn't duplicate the components that might be common to several simpler cells -- such as the $\mathrm{avg}(\vec{f}\vec{f}^T)$ correlation matrix.

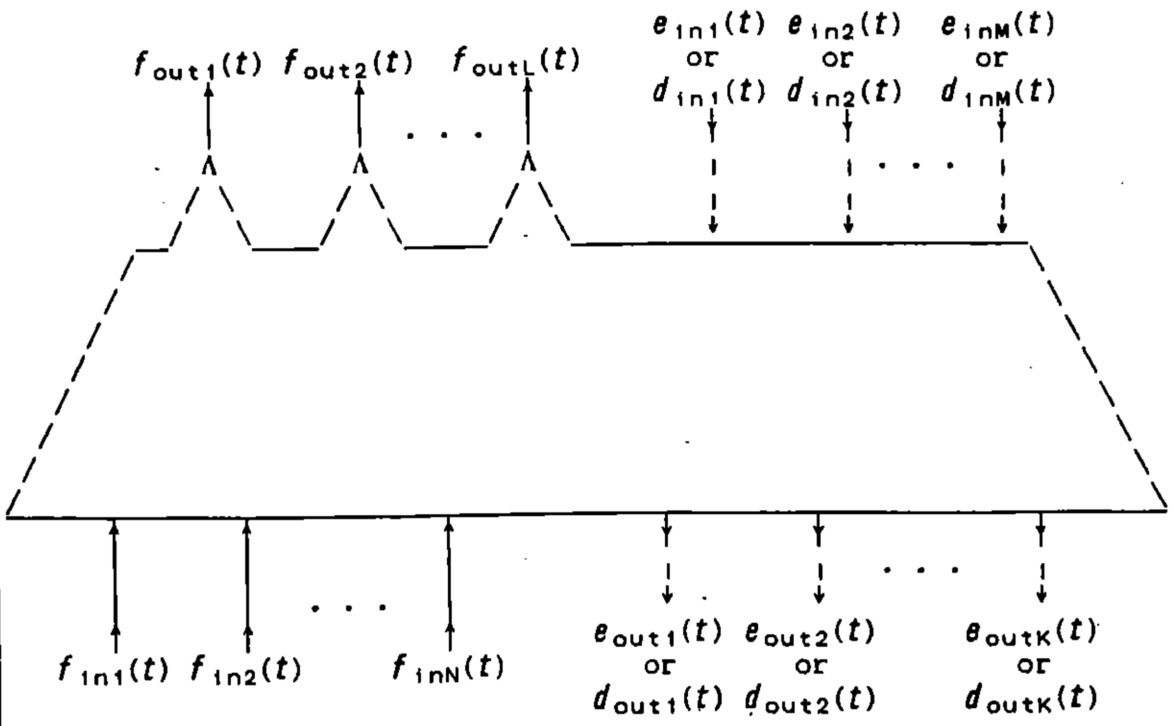

$f_{out1}(t)$
$f_{out2}(t)$
$f_{outL}(t)$
$e_{in1}(t)$ or $d_{in1}(t)$
$e_{in2}(t)$ or $d_{in2}(t)$
$e_{inM}(t)$ or $d_{inM}(t)$
$f_{in1}(t)$
$f_{in2}(t)$
$f_{inN}(t)$
$e_{out1}(t)$ or $d_{out1}(t)$
$e_{out2}(t)$ or $d_{out2}(t)$
$e_{outK}(t)$ or $d_{outK}(t)$

Figure 19: The most general Learning-Logic cell.

There are other generalizations possible -- such as cells that treat function signals as error signals or vice-versa, or that convert signals from one form to another -- but again they don't involve anything beyond what has already been discussed.

PRACTICAL CONSIDERATIONS

When connecting function cells, error cells, basic Learning-Logic cells or approximate Learning-Logic cells into networks, there are some practical matters to take into consideration. Many of these practical modifications increase the class of functions that can be learned by a network of linear Learning-Logic cells (without increasing the amount of circuitry involved), make the network converge faster, or otherwise enhance its behavior. Of course, the practical matters that need to be considered depend on the particular implementation of Learning-Logic. For instance, there are modifications that can be made to Learning-Logic software that are impractical for hardware implementations.

However, there are three modifications that I believe are applicable to a wide variety of Learning-Logic implementations. I believe that all three of them are used, for example, by our cortical neurons. In decreasing order of importance, they are:

- Clipping.
- Use of exponentials and logarithms.
- Thresholding.

Because of their wide applicability, I will examine each of them in some detail.

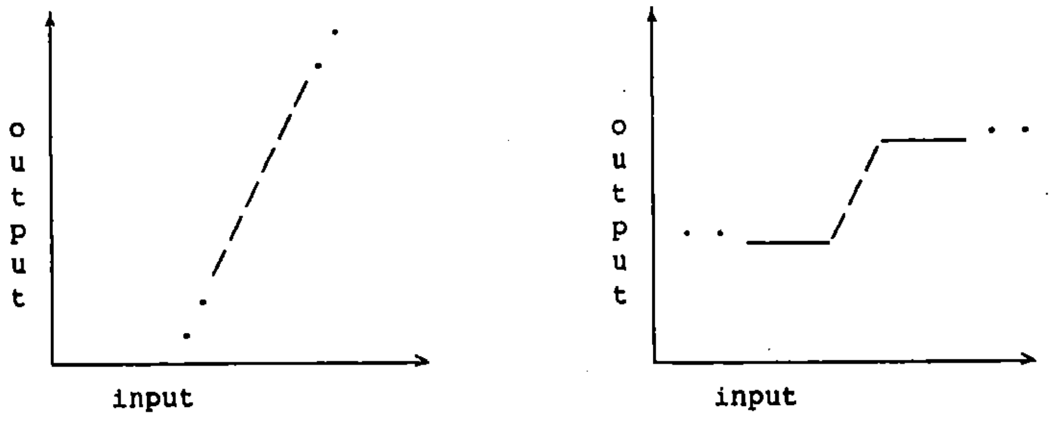
Clipping

In an ideal Learning-Logic cell or network, any parameter or signal could theoretically range from $-\infty$ to $+\infty$. Clipping refers to limiting these ranges to within some practical bounds (see Figure 20). For example, theoretically the output of a linear Learning-Logic cell may range from $-\infty$ to $+\infty$. However, any physical implementation of Learning-Logic will place lower and upper bounds on what the output actually can be. Thus, in a practical implementation of linear Learning-Logic, the output will be approximately linear in some region and then will be clipped (or will saturate) at the upper and lower bounds of the linear region.

This is the same behavior as is observed in transistors, for instance. They amplify approximately linearly over some region, but clip the signal to some minimum or maximum value outside that region.

Clipping is a very desirable property for linear Learning-Logic. Just as clipping in transistors is the basis for digital electronics (the lower and upper bounds can represent 0 and 1, or false and true), it enables linear Learning-Logic cells to learn both digital and analog functions.

Besides allowing linear Linear-Logic cells to learn digital functions, clipping allows networks of linear Learning-Logic cells to represent other non-linear functions as well. A theoretical network of perfectly linear Learning-Logic cells isn't very useful, because linear functions of linear functions are still linear functions, so clipping is a way of extending the power of a network of linear Learning-Logic cells without increasing the amount of circuitry involved.



1) <u>Unclipped</u> linear signal. 2) <u>Clipped</u> linear signal.

<u>Figure 20:</u> Clipping example.

Clipping can be observed in our neurons. The most obvious occurence is in the frequency of the action potentials, which can range from 0 -- the lower bound -- up to the maximum firing rate of the neuron, several hundred pulses per second, say -- which is the upper bound.

<u>Use of Exponentials and Logarithms</u>

Exponential functions occur so frequently in nature, and are of such practical importance, that it seems appropriate to discuss their use in Learning-Logic.

The two most important uses for exponentials and logarithms are to multiply signals together and to update parameters.

A common way to multiply signals together is to take their logarithms, then add the logarithms together, and then take the exponential of the result:

$$a_i f_i = \exp(\ \log(a_i)+\log(f_i)\ )$$

This is how many electronic analog multipliers actually work. Note that all the quantities involved must be positive: this only multiplies quantities in the first quadrant.

The other major use for exponentials and logarithms is in the accumulation of parameters. Neurobiologists have found that due to the electrical properties of a neuron, the influences of input signals fall off exponentially as the distances from their synapses to the axon hillock. Assuming those distances contained the internal parameters, then a neuron instead of performing the linear function

$$f_{out} = a_1 f_1 + a_2 f_2 + \dots + a_p f_p = \bar{a}^T \bar{f} = \bar{f}^T \bar{a}$$

is basically performing the function

$$f_{out} = \exp(a'_1) f_1 + \exp(a'_2) f_2 + \dots + \exp(a'_p) f_p = \exp(\bar{a}'^T) \bar{f} = \bar{f}^T \exp(\bar{a}')$$

The a''s are the logarithms of the a's, and they correspond roughly to the negatives of the distances from the corresponding synapses to the axon hillock, multiplied by some constant.

It would seem that if the neurons are actually updating the logarithm of the internal parameters instead of the internal parameters themselves, that they could no longer use the linear form of my favorite updating algorithm. It turns out, however, that they can. The updating formula

$$\frac{\partial \bar{a}}{\partial t} = U^{-1} \left[H^{-1} e \bar{f} - [\ \mathrm{avg}(\bar{f}\bar{f}^T) - \mathrm{diag}(\mathrm{avg}(\bar{f}\bar{f}^T))\ ] \frac{\partial \bar{a}}{\partial t} \right]$$

converges to the exact least-squares solution for either the internal parameters or the logarithms of the internal parameters depending on whether $\bar{a}$ represents the internal parameters or the logarithms of the internal parameters. In fact, one can substitute any suitably well-behaved functions $g_i(a'_i)$ for the internal parameters a_i and the linear algorithm will still converage to the exact least-squares solution for a'_i as long as $\partial g_i/\partial a'_i$ is always non-negative (and probably even in lots of cases where it's not). The proof of this fortunate occurence will be left to the interested reader. It exactly parallels the proof of convergence in the completely linear case given in the section "How Function Cells Work".

Another example of how logarithms can be used in the accumulation of parameters is in the averaging multiplier of the section "Averages". If the average product of two quantities is going to be fairly small, it may be of advantage to accumulate the logarithm of the average product. The following updating algorithm can then be used:

$$\frac{\partial \log(\mathrm{avg}(f_1 f_2))}{\partial t} = \frac{1}{h} \left(f_1 f_2 - \exp(\log(\mathrm{avg}(f_1 f_2))) \right)$$

As above, other functions besides logarithms and exponentials will work with this algorithm.

Thresholding is a method used to generate pulses whose frequency transmits information (see Figure 21). It would only be of use in Learning-Logic cells that were using pulses as signals. A thresholding mechanism can be viewed as an analog-to-pulse converter (the inverse of the pulse-to-analog converter discussed in "Signals"). In fact, the thresholding mechanism can be used to eliminate pulse-to-analog conversion of a cell's inputs, because accumulating the threshold variable V automatically converts whatever is being accumulated to analog. I believe that a neuron's axon hillock acts as a thresholding mechanism (Kuffler and Nicholls, page 15).

A taxi stop can be used as a simple analogy for a thresholding mechanism. The number of people per minute who want to take a taxi can be viewed as an analog signal f_{out} -- at 4:30 a.m. it is close to 0, while at rush hour up to 30 people per minute, say, might want to take a taxi. The accumulation variable V is the number of people who are currently waiting at the taxi stop. For instance, if no taxis came for a while then the value of V might build up to over 100. However, we'll make sure to supply enough taxis to handle the load. We'll suppose that each taxi waits until there are two people ready to go, so that the threshold value V_0 is 2, and that it then takes both of them, so that V_{out} is also 2. We can then view each departing taxi as a pulse. At 4:30 a.m. the number of taxis leaving would be close to 0, while at rush hour there would be approximately 15 taxis per minute departing. Thus, the analog signal has been converted to a series of pulses whose frequency transmits the relevant information.

In this example, the factor of 1/2 between the analog signal f_{out} and the frequency of the pulses illustrates the inverse amplification possible depending on our choice of V_{out}. If we used taxis that held four people instead of two, then the number of pulses per second would have been cut down to about 7. Similarly, if we used taxis that held only one person, the number of pulses would have gone up to 30. Fortunately, the updating algorithm for the internal parameters $\bar{a}$ is transparent to this amplification -- it will automatically correct $\bar{a}$ for it.

Another item to note about this example is that the frequency of the output pulses is relatively insensitive to the threshold value V_0. If the taxi drivers were really rude and waited till there were four people waiting before driving off with only two of them, there would still be close to 0 taxis per minute leaving at 4:30 a.m. and close to 15 taxis per minute leaving at rush hour.

$$\frac{\partial v}{\partial t} = f_{out} - v_{out}$$

$$x_{out} = \begin{cases} \text{pulse,} & \text{if } v > v_0 \\ \text{no pulse,} & \text{if } v < v_0 \end{cases}$$

v_0 = threshold value of the threshold variable v.
v_{out} = change in v if a pulse occurs, 0 if no pulse occurs.
f_{out} = signal to convert to pulses -- for a linear function cell:
  $f_{out} = \bar{a}^T\bar{f}$ if the inputs have already been converted to analog.
  $f_{out} = \bar{a}^T\bar{x}$ if the inputs are still in pulse form.

<u>Figure 21:</u> Description of <u>thresholding</u>.

A third thing to note is that since the frequency of the pulses now carries the relevant information, the magnitude of the pulses is unimportant. For instance, if the taxi drivers were all homocidal and killed one of their two passengers, so that only one passenger was delivered to their destination, the frequency of the taxis would still be the same.

Finally, before dropping this example, we can note that a leaky integrator can replace the ideal integrator we've been using to accumulate v. The equation for v would then become:

$$\frac{\partial v}{\partial t} = f_{out} - v_{out} - h^{-1}v$$

where h is a measure of the history involved, i.e. how long people are willing to wait for a cab. If h is close to zero then if there isn't a taxi immediately available everybody gets disgusted and leaves, and if h is close to infinity everyone will wait forever for a taxi. I imagine that this same sort of leakage occurs at the axon hillock of our neurons, and probably everywhere else an integrator is used in our neurons. Many implementations of Learning-Logic will also involve leaky integrators.

This section explores the correspondences between Learning-Logic and our cortical neurons, in question-and-answer format.

<u>What evidence is there that Learning-Logic works in any way like our cortical neurons?</u>

There are many pieces of evidence:

1. In order to get Learning-Logic to work well with cells that conduct signals in only one direction (as all neurons do), mathematically there have to be two types of cells in the network: function and error cells. I don't think it is a coincidence that there just happen to be two types of neurons in our cortex: the pyramidal and stellate neurons.

2. Since error cells are simpler than function cells, one of these two types of neurons <u>has</u> to be simpler than the other. And, indeed, it seems that stellate neurons are simpler than pyramidal neurons (see the answer to the next question). The reason a network needs both is that Learning-Logic doesn't work very well if all of the cells are either function or error cells. There has to be a mixture of the two. (This isn't true for bi-directional Learning-Logic: networks composed of only basic Learning-Logic cells work fine.)

3. Function and error cells need to conduct their signals in roughly opposite directions, and in the picture I use (Kuffler and Nicholls, page 39) the axons of the stellate and pyramidal neurons seem to be going in opposite directions.

4. Learning-Logic contradicts nothing else that I know about our cortical neurons (if it contradicts something you know, please tell me). I know of no other theory of our cortex that 1) can be shown to work on its own and 2) that doesn't contradict some observed fact about our cortical neurons. Beyond that, Learning-Logic provides several hypotheses about neural behavior that can be tested experimentally (see the question about the predictions Learning-Logic makes).

5. Learning-Logic networks can learn both digital and analog functions, just as humans can -- and even blend the two, just as humans can. This gives me confidence that suitably constructed networks consisting of large numbers of Learning-Logic cells (aided, perhaps, by some "fixed-logic" or hardwired cells, just as our cortex is probably aided by more primitive areas of our brains) will soon be exhibiting "intelligent" behavior. All of the building blocks seem to be there.

What is the evidence that function and error cells correspond to pyramidal and stellate neurons, and not vice-versa?

There are two main pieces of evidence:

1. Function cells are more advanced than error cells, and in the picture I use (Kuffler and Nicholls, page 39) pyramidal neurons look more complex than stellate neurons.

2. Error cells are likely to have evolved first, because error cells are simpler, and stellate neurons seem to be more common in primitive areas of the brain (Kuffler and Nicholls, page 11).

If it turns out that Learning-Logic does correspond to our cortical neurons, but that I have the correspondences reversed, then I'll have made a biological faux pas equivalent to hanging a piece of modern art upside down.

What can Learning-Logic say about long term memory in pyramidal neurons?

If pyramidal neurons correspond to function cells, then there are two components to its long term memory. If a pyramidal neuron has p synapses from other pyramidal neurons, it needs to remember the p internal parameters (the vector $\bar{a}$) for those inputs. It also needs to remember the $(p^2-p)/2$ correlations between those inputs (the off-diagonal terms of the symmetric correlation matrix $\text{avg}(\bar{f}\bar{f}^T)$).

1. It seems to me, and I think to many others (Kuffler and Nicholls, Chapter 16), that there are two plausible ways a neuron could remember the p internal parameters:

   - As the distance between each synapse and the axon hillock. Since the electrical signal that arrives at the axon hillock from a synapse is thought to decay roughly exponentially with the distance between the synapse and the axon hillock (Kuffler and Nicholls, page 135) that distance could contain the internal parameter for that synapse. The neuron could adjust this internal parameter by growing or shrinking appropriately.

   - As the strength of the electrical signal transmitted across the synapse. If the distance between a synapse and the axon hillock remains roughly constant, then the only other (plausible) way for the neuron to adjust the internal parameter for that synapse is to adjust the electrical signal itself. This might be done on the dendrite's side of the synapse, on the axon's side, or in the synapse itself (Kuffler and Nicholls, Part Two).

An implausible way that the neuron could remember the internal parameters is as the number of synapses made by a given axon. The neuron would have to both selectively stimulate growth of new synapses, and selectively shed them. This seems to me to be unlikely, given the simplicity of the other two methods.

2. It seems to me that there is only one plausible way a pyramidal neuron could remember the $(p^2-p)/2$ correlations between the inputs, and that is as the distances between the synapses. Whether those distances are measured through the body of the neuron and the surrounding matter, or along the neuron's cell membrane, I don't know. Whichever, it implies that the neuron can grow or shrink between each pair of synapses, stimulated by the signals at each synapse.

   The other two obvious possibilites, that the neuron can adjust the electrical or chemical properties between each pair of synapses, or that synapses can be selectively grown or shed, seem too complicated.

   There is at least one other outside shot that deserves mentioning, and that is that the pyramidal neuron could maintain the correlations as tensions along its cell membrane.

How about long term memory in stellate neurons?

If stellate neurons correspond to error cells, then they only have to remember the r internal parameters (the vector $\bar{b}$) for the r synapses from other stellate neurons. They could do this in a manner similar to pyramidal neurons, as in 1) above. Stellate neurons wouldn't have to remember the correlations between the inputs, as in 2) above: that is why they are more primitive.

How about short term memory?

There is nothing in neurons that I know of that can change fast enough to account for short term memory -- except for the signals sent between the neurons. So, I believe that short term memory is maintained in the pattern of signals being sent. This view is bolstered by the fact that Learning-Logic cells can learn to perform functions similar to those that digital electronic circuits perform, and the registers and memories of digital electronics are often used as analogies for short term memory.

What about inhibitory and excitatory neurons?

I believe inhibitory and excitatory neurons are a practical modification made necessary by the fact that almost all the ways that neurons might accumulate their internal parameters only accumulate positive values. Thus, in order for an input to have a negative influence, a negative (inhibitory) copy of the input (or something negatively correlated with it) is needed. In fact, in many man-made implementations of Learning-Logic it may be better to have two versions of each signal -- one considered positive and the other negative -- than to have to one version that can be either positive or negative.

What are some of the differences between our neurons and perfect Learning-Logic cells, and do they matter?

I doubt that our neurons are mathematically perfect Learning-Logic cells. Forturnately, Learning-Logic seems to be very tolerant of imperfections in its components and connections. In fact, I believe there are at least two ways in which our neurons aren't ideal Learning-Logic cells, but they don't matter.

1. I don't believe that pyramidal neurons update their avg($\vec{f}\vec{f}^T$) correlation matrices exactly. To do so, each synapse would have to interact with every other synapse. Instead, I believe that each synapse interacts with only a few nearby synapses. The neuron might not converge, except for a saving stroke: each axon usually makes many synapses on the neuron. That means that a synapse from one axon is likely to be near enough to a synapse from every other axon so that a sufficient number of elements of avg($\vec{f}\vec{f}^T$) are accumulated.

2. I don't believe that the pyramidal and stellate neurons are connected as orderly as in Figure 12. However, that just means that the error signals are distributed only approximately correctly. The network will still converge, except the neurons may change their parameters a little more than they would in an ideal network. The only observable consequence of this would be a tendency to forget some things a little faster, though the difference would probably be insignificant.

   There are probably even situations where it is better not to be connected as orderly as in Figure 12. For some purposes, for instance, it might be better to break the feedback loop from EC_5 to itself while keeping the feedback loop from FC_5 to itself.

Can Learning-Logic say anything about learning disorders?

If Learning-Logic corresponds to our cortical neurons, then knowing how they work obviously narrows down the possibilities that need to be investigated in determining the root cause of a specific learning disorder. Each of the parameters in a Learning-Logic cell is a potential cause of a learning disorder. In fact, it may become possible to manufacture a network of Learning-Logic cells that incorporates the suspected defect to see if its behavior corresponds to the observed human behavior.

On a more specific level, Learning-Logic can predict a cause for a type of learning disorder. Unfortunately, I'm not sure which type of learning disorder -- the cause is all I'm fairly sure of.

The most likely cause for a learning disorder seems to me to be some defect in the pyramidal neurons' ability to update the correlations between their inputs. This makes sense to me for several reasons:

- A defect in the pyramidal neurons makes more sense than a defect in the stellate neurons, because the stellate neurons are more primitive. I would think that defects in the more primitive neurons would tend to be fatal instead of crippling because the more primitive neurons control our basic bodily functions.

- Because the ability to update the correlations between their inputs is the chief innovation of the pyramidal neurons, it seems likely to be the thing most easily subject to defects. In evolutionary terms, it is probably the "newest" thing in our brains so presumably it is also the thing we can most easily live without -- although living without it probably means living without intelligence, which just happens to itself be a correspondingly recent evolutionary innovation.

- A network of stellate and pyramidal neurons in which the pyramidal neurons couldn't update the correlations between their inputs could still learn fairly simple functions. It couldn't be guaranteed to converge, though, so that learning new things would tend to make it more quickly forget old ones. Also, some functions are provably impossible for it to learn. This seems to me to correspond with the behavior associated with some kinds of learning disabilities.

If this bears out, then hopefully positively identifying the exact mechanism the pyramidal neurons use to update the correlations between their inputs will lead to an understanding of how to fix them when they don't.

<u>What predictions does Learning-Logic make about our cortical neurons that can be tested experimentally?</u>

Many predictions can be made based on the matters discussed in this paper, but the two things I feel will be the easiest to test are:

- Stellate neurons shouldn't be able to make pyramidal neurons fire, and vice-versa. The reason is that function cells and error cells only use each other's signals for training purposes and not to determine each other's firing rates. This prediction should hold even if I have goofed on which type of neuron corresponds to function cells and which to error cells.

- The change in the pyramidal neurons that corresponds to updating the correlations (probably a change in distance between synapses) could be measured, and then the lack of the same feature in stellate neurons could be verified.

REFERENCES

Hinton, Geoffrey E., Sejnowski, Terrence J. and Ackley, David H., Boltzmann Machines: Constraint Satisfaction Networks that Learn, May 1984, Carnegie-Mellon University Technical Report CMU-CS-84-119

Kuffler, Stephen W. and Nicholls, John G., From Neuron to Brain, 1976, Sinauer Associates

Ljung, Lennart and Soderstrom, Torsten, Theory and Practice of Recursive Identification, 1983, MIT Press

Rosenblatt, Frank, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, 1962, Spartan Books

Widrow, B., Mantey, P.E., Griffiths, L.J. and Goode, B.B., "Adaptive Antenna Systems", Proceedings of the IEEE, Dec. 1967, Vol. 55, No. 12, p. 2143