A CLASS OF SELF-GENERATING NEURAL NETWORKS

Roderick Edwards, Reinhard Illner,
and
Graeme Leeming

DMS-710-IR July 1995

A Class of Self-Generating Neural Networks

by

Roderick Edwards, Reinhard Illner, and Graeme Leeming

Department of Mathematics and Statistics,
University of Victoria, P.O. Box 3045,
Victoria, B.C. V8W 3P4, Canada

ABSTRACT

We discuss the design and behavior of families of neural networks which grow out of a single "mother" neuron in response to external stimuli and to the activities present in various parts of the net at a given time. The growth process is subject to a few fundamental rules, like

-the ability of neurons to grow new neurons or connections is gradually exhausted with the number of generations
-neurons are either of excitatory or inhibitory type
-inhibitive neurons have a tendency to form long-range connections, whereas excitatory neurons "prefer" short-range connections.

In addition, there are a number of free parameters in the equations driving the time evolution of the neural activities as well as the changes in the connection strengths. Our design is implemented using Matlab, such that the growth process of the network and its activity can be observed and controlled interactively on the computer screen.

1. Introduction

Standard neural networks are either feed-forward nets of perceptron type, or they allow feedback between the various layers of neurons. The latter type is usually described as a dynamical system, and the equations for the membrane potential (or firing rate, or activity) of the individual neurons form a coupled system of nonlinear differential equations. For surveys on the theory of these nets, we refer the reader to the books by Hertz, Krogh, and Palmer [HKP], Amit [A], or the (older) papers by Grossberg [Gr] or Lippmann [Li].

What all these nets have in common is that their graph structure is fixed once and for all. Connections exist or not, and training or learning of the net is done by adjusting the connection weights. This process follows the biological process of changing synaptic efficacies; various learning rules are used, the most common being variants of the Hebb Rule [He]. In essence, a neural net consists of the following three elements:

1. A set of neurons at certain locations; each neuron can send output or receive input via connections. A connection is a one-way street; from the point of view of the sending neuron, we call it an axonal connection; from the point of view of the receiving neuron, we call it a synaptic connection. The synaptic inputs are multiplied with a "gain factor" (which is a parameter μ), fed through (modulated by) a gain function, multiplied with "synaptic strengths" (connectivity factors) and added up. The result then determines the rate of change or the output of the neuron.
2. A set of existing connections between the various neurons, which gives the whole object the structure of a directed graph.
3. A matrix of connectivity (synaptic) strengths, which can change in the learning process.

In a standard net, the first two of these elements are fixed; only the third is subject to change.

This fact leads to conceptual and biological quandaries for large nets. For, say, 10^6 neurons, the number of possible connections is 10^{12}, which is

an astronomically large number. If all these connections are allowed, training or learning becomes an onerous task. In principle, every conceivable network with 10^6 neurons could emerge from a net with a full connection matrix (the unnecessary connections getting eventually strength 0), but the training process towards a particular net may be awkward to the point of being impossible. Besides, there may not be a clear correspondence between classes of information processing tasks and classes of network structures.

Also, biological evidence points in a different direction. Complicated biological neural networks, i.e., brains, evolved from simpler ones in an evolutionary process, a process which is repeated, following a genetic blueprint, with some accuracy in each individual as the brain develops. The end results are large brains, with billions of neurons but a relatively sparse connection matrix.

Neurons differ from other body cells in the sense that they do not split. As a consequence, a mature brain is a more stable biological structure than, say, a muscle or any other organ, where the individual cells are not so important. Neurons, once they die, are not replaced; the only way that the functionality of the brain is maintained is by sharing of tasks by many neurons. This redundancy is indeed a central feature of biological neural nets.

It is easy to think of reasons why neurons in the mature brain were not designed to split. Brain functions are infinitely more complex than the functions of other organs, and the way that neurons interact is crucial for this function. Random splitting of neurons would presumably alter the brain function in ways which would be incompatible with maintaining the multiple specific tasks assigned to the brain by nature.

In the process of evolution, however, such splittings must have occurred in one way or the other, because otherwise there would be no sophisticated brains at all (when we use the word "brain" here, we mean any biological neural structure with at least a few dozen interconnected neurons). Splitting of some kind must also occur in the growing brain, albeit controlled by the genetic code.

The objective of the project described in this article is to attempt a

simulation of internal growth processes for neural networks. We start with a network of the most primitive conceivable type, namely a single neuron (in the course of our research, we half-jokingly referred to it as the "mother neuron"; this seems indeed a very appropriate name for it). The mother neuron has the ability to split, or, more accurately, "sprout" new neurons, which can sprout again, or form connections to already existing other neurons, etc. The "child" neurons, "grandchild" neurons etc. are connected to their parent by a synaptic connection, and they can, with some randomness in the process, grow axonal connections to other neurons and "share" their activity, as measured by their membrane potentials. The whole process is driven by external inputs and by the activity level of the existing neurons at any given time. The sprouting and connecting routines are subject to random fluctuations. Synaptic connection strengths change over time and can be excitatory or inhibitive, but a given neuron can grow either excitatory or inhibitive connections, not both. We shall therefore call neurons excitatory or inhibitive, depending on what type of connections they grow. Inhibitive neurons are never allowed to sprout, because a "child" of such a neuron would show little activity and therefore contribute little to the information processing ability of the net. Instead, when an inhibitive neuron grows a connection, this connection must lead to an already existing neuron.

We implemented limits to the size of the net by setting low upper limits to how many connections to new or existing neurons a given neuron can sprout and how many synaptic connections it can receive. If a connection is inactive for too long, it can be severed, thus freeing the two connected neurons for new connections. For all the details of the process, see Sections 2 and 3.

While working on the project, the neural nets which we grew as well as the rules we used went through an evolutionary change themselves. A first net was designed such that any neuron could sprout to any location, and it could connect (essentially randomly) to other neurons. We assigned location to the neurons and displayed the growing graph in three dimensions. The activity of the neural net was displayed as a function of the label of the neuron on

another graph. The problem which we ran into was that we could not discern any correlations between the structure of the net and the activity graph. This is likely not a problem of the net, but just a consequence of the limitations of the human visual ability. We saw chaotic dynamical behavior in this net for suitable choices of certain parameters; we saw characteristic responses to inputs applied to groups of neurons; but we had no way of understanding what kind of processing mechanisms had developed in the net.

Therefore, we designed a second type of network, presented in this paper, whose growth process is dependent on the current spatial configuration. As a courtesy to the human visual system, the system evolves on a two-dimensional grid, and the direction in which a new neuron sprouts from an existing one depends on a "potential" created by the neurons already in place. In the sense of this potential, the neurons can "feel" each others presence even without being connected, and they can even feel the number of available synaptic connections left in other neurons (we simply made the potential dependent on this number).

For obvious reasons, we tried to keep the growth process as simple as possible. We do not know whether real biological evolution followed a similar pattern (probably not), but we borrowed some biological observations (notably the "Hayflick phenomenon" [Ha] which implies that cells other than neurons will split only finitely many times, the fact that neurons are either excitatory or inhibitive, but never both at the same time, and the rule "long-range inhibition, short-range excitation") and implemented them as simple rules in our program. These rules limit the size of the growing nets automatically and guarantee that the connection matrix will remain sparse. It may well be that there are conceptually many ways to grow interesting neural networks; nature found such ways after millions of years of evolution, but it was confined to certain chemical and physical processes in growing neurons and connections with the desired properties. The mathematician (or computer scientist, or electrical engineer, or neurobiologist) of the present day has the advantage that all these elements can be created in an abstract sense in the computer, and the behavior of the resulting brain can be studied

immediately.

This is what we did and what we report on in the present work. We caution the reader that while we are mathematicians, there is little mathematics in this paper. There is hardly any "intelligent" behavior of our grown networks to report, hence we have not created any artificial intelligence that deserves this name. In fact, our "brains" appear more "stupid" than neural networks of the perceptron or Hopfield type which are designed and trained for specific purposes and which are already widely used in engineering applications. The point of our research is not so much the structure of the resulting networks (which is, in any case, too complicated for detailed understanding), but rather an understanding of the ground rules which must govern, or have governed, the evolution of neural nets. We present, discuss and test a set of such ground rules in this paper, but we emphasize that these ground rules can be, and must be, tinkered with.

We dwell some more on the problem why the mathematical analysis of complex neural networks seems so hard if attempted directly. A network of, say, 100 neurons, in which each neuron is connected to 10 others, will be described by 100 coupled nonlinear ordinary differential equations, with 10 coupling terms on the right hand side of each equation. If equations for the connectivities (on a slower time scale) are added, the number of equations will grow by another thousand. Even if we disregard these latter equations for the modeling of the neural dynamics, the state space of the system has 100 dimensions, and it seems nearly hopeless to predict the dynamical behavior of such a system, or its response to external inputs (modeled as force terms on the right-hand sides of the equations). Even for much simpler systems of differential equations, with only 4 or 5 dependent variables, prediction of the dynamic response to inputs is a real challenge (for a case study on the Lorenz system of equations, see [EIK]). Attempts to simplify such systems by simulating subregions of the neural net by nonlinear diffusion equations were made by Cottet [Co] and, in greater generality, by Edwards [Ed], but the analysis done in these papers shows that such mean field approximations are only feasible under very special assumptions on the connection matrix (in

Ref. [Ed], connections are local, predominantly excitatory, and nearby connections cannot have very different strengths; other approaches have similar limitations).

Yet nature has found ways to design much larger networks, described by systems of equations of unimaginable complexity, which display highly complex yet highly organized behavior. Our paper is a very first step to repeat such a design process in a computer.

2. The model

We now describe, in a series of steps, how our network starts from the mother neuron and grows in response to external stimuli and, eventually, internal activity. The activity of a given neuron is identified with the membrane potential of that neuron. External inputs are identified with outer forces on the right-hand sides of the equations. For a given size of the net, the network equations have the form

$$\dot{u}_i = -u_i + \sum_{j \neq i} T_{ij} \mathrm{sgm}(\mu(u_j - w_j)) + I_i. \tag{2.1}$$

Here, u_i is the membrane potential of the ith neuron. There is leakage of this potential, expressed by the first term on the right. The T_{ij} are the connection strengths between the neurons with label j and i, where j is the label of the sender, i the label of the receiver. An outer force term, interpreted as a temporary input upon neuron i, is given by the term I_i (I_i can be switched on or off, and can take any real value inside an interval $[-I_{max}, I_{max}]$, thereby enhancing or inhibiting the activity of the i-th neuron). The symbol sgm denotes a sigmoid function. In this paper, we take

$$\mathrm{sgm}x = \frac{1}{2}(1 + \tanh x) = \frac{1}{1 + e^{-2x}},$$

which assumes values in (0,1). Other sigmoids are certainly possible. The constants w_j are inserted as possible shifts which further differentiate the activity of inhibitive from that of excitatory neurons. In our experiments, we

chose $w_j = 0$ for all excitatory neurons, $w_j = .5$ for all inhibitive neurons. The rationale for this is explained in Refs. [Ed2] and [PBK]. In essence, choosing the constant μ sufficiently large and combining inhibitive neurons with a shift with excitatory neurons without a shift is likely to create conditions which will lead to chaotic behavior of the net in the absence of external input. The experiments discussed in Section 3 confirm this. The constant μ is known as the gain, and $\mathrm{sgm}(\mu(u_i - w_i))$ is the firing rate of the i-th neuron. By construction, the membrane potential takes values in $\Re$, whereas the firing rate takes only values in (0,1).

We will also need a rule to initialize, and change in the learning process, the connection strengths T_{ij}. As this requires two separate steps, namely initialization when the connection is first created and updating later on, we introduce this rule as part of the growth process.

THE GROWTH PROCESS.

2.1. The mother neuron. The mother neuron is really just an ordinary neuron, but it has the ability to sprout new neurons, to which it will be connected via axonal connections, in response to its own activity. Specifically, the mother neuron can receive an input, i.e., be stimulated externally, and if its activity rises above a predetermined threshhold τ, it will sprout an axon with a child neuron. The mother neuron is by definition an excitatory neuron, meaning that the axonal connection which connects it with its child is excitatory. The child neuron can be an inhibitive or excitatory neuron; the choice is made randomly, and the probability that a sprouted neuron is inhibitive is a free parameter in the program, denoted by "*xratio* ".

If a child neuron is excitatory, is has again the ability to sprout new neurons in response to sufficient activity (Note that the sprouting takes place when the *activity* is above the threshold; for a sufficiently large network, this can happen even when the neuron receives no direct input). If the child is inhibitive, we do not allow it to sprout "grandchildren"; its only way to build connections is to connect back to the mother neuron or, later, to other neurons already in existence.

We require that the mother neuron can sprout child neurons, or connect via axonal connections to other neurons (e.g., grandchildren), only a finite number of times; the mechanism which we use to enforce this is to define an integer G ("*max_gen*" in the computer program), read as "maximal possible number of generations". We require that $G > 1$ and set the number of admissible axons sprouting from the mother neuron as G. The number of admissible axons sprouting from the next generation is then set as $G - 1$, etc. This rule implies that every subsequent generation can have one fewer connection than the previous generation and is a convenient (and natural) way to limit the size of the net. This rule is modelled after the famous Hayflick-phenomenon, which states that cells of mortal organisms will split only finitely many times (and the number of splittings is related to the characteristic life span of the organism under consideration).

Generations are therefore counted downward, beginning with G. If g_i is the generation label of the $i-$th neuron, then g_i is an integer between 0 and G. By $a_i(t)$ we denote the number of remaining potential axonal connections of the $i-$th neuron at time t. If t_* is the moment of creation of this neuron, then $a_i(t_*) = g_i$.

The third characteristic number associated with the mother neuron is again an integer, which we denote by S ("*max_synapse*" in the program). S is the maximal number of synaptic connections which the mother neuron (or any other neuron) can potentially accommodate.

We summarize: In the beginning, there are three parameters, τ (threshold for sprouting), G, the number of possible generations, and S, the number of possible synaptic connections. A schematic graphic representation of the situation is given in Fig. 1.

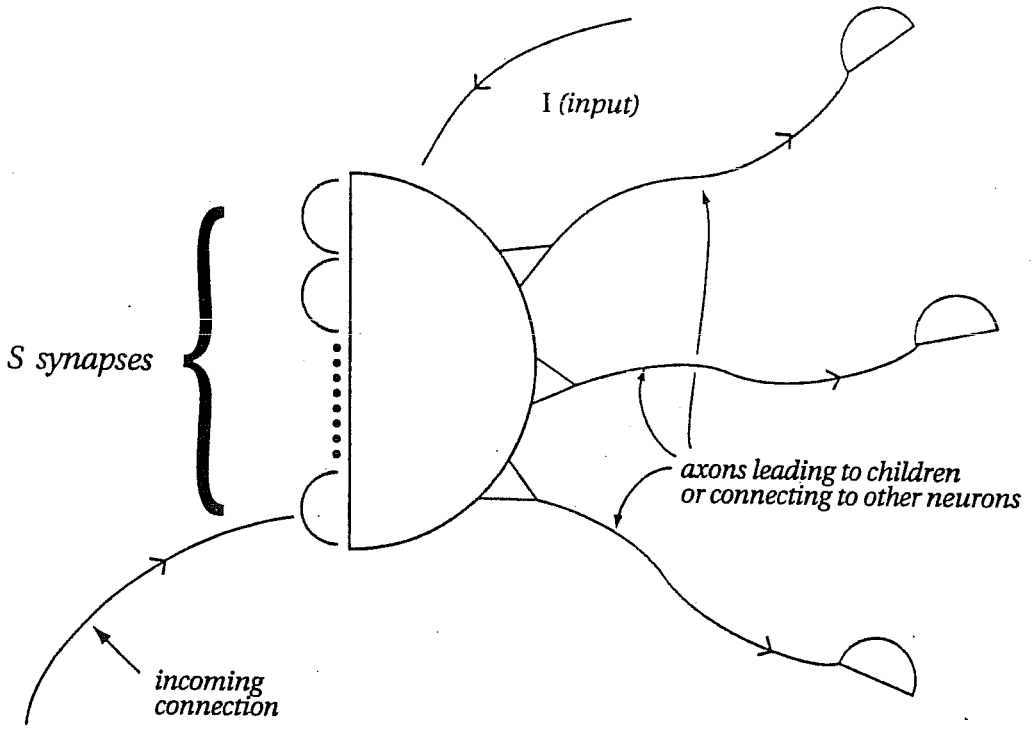


Figure 1. *Configuration of the mother neuron*

2.2. The next generations. The neurons of the next generation (the children) are set to have essentially the same properties as the mother neuron: They have again room for S synaptic connections, and their threshold for sprouting or connecting is again τ. However, we allow them to sprout or connect only $G-1$ times, and the next generation will be allowed to sprout or connect only $G-2$ times, etc. This is a built-in mechanism to limit the size of the net, and it follows that the net can never grow to more than $\sum_{k=0}^{G} \frac{G!}{(G-k)!} = \sum_{k=0}^{G} k! \binom{G}{k}$ neurons.

Once a few neurons are in place, neurons which receive an input above the chosen threshold do not necessarily have to sprout a new neuron. We also give them the option to build a connection to an already existing neuron, e.g., to the mother neuron, a sister neuron, or any other neuron with room for synaptic connections. In fact, as already mentioned above for the first child of the mother neuron, inhibitive neurons have only this option.

We have so far not mentioned the geometric structure of the emerging

net. In fact, it is conceivable to ignore this aspect completely and simply not assign the new neurons a spatial location at all; in such a model, every neuron will consider every other one as a candidate for a possible connection, and may make a random choice among these candidates. As mentioned in the introduction, we did in fact try this; in order to get an idea what type of net would emerge, we placed new neurons randomly in a three-dimensional box (on the computer screen) and outlined connections by straight lines. The result was (predictably) a mess whenever more than 10 neurons had formed, and we found it all but impossible to explore interesting correlations between the given input, the structure of the net and the output.

Therefore, we eventually postulated that our neurons actually had to have a spatial location. As a courtesy to the human visual system, that location must be on a grid point in a two-dimensional finite grid, typically a square of grid size 30×30 (such that there is room for 900 neurons; the grid size is another free parameter which can be changed from experiment to experiment). As a simple mechanism to record whether a site in the grid is occupied by a neuron or not, we introduce an "occupation matrix" $O(i,j)$ whose entries are 0 at empty sites and are the labels of the neurons at occupied sites (neurons are labeled in the order in which they are created). Another matrix "*neuron_type*" $N(i,j)$ has entries 1,-1, or 0, depending on whether the site is occupied by an excitatory or inhibitory neuron, or vacant.

We further require that by their mere presence, the neurons create a potential field which determines the direction of new axons. Such axons may terminate in new neurons ("sprouting") or at neurons already present ("connecting"). Inhibitory neurons have only the second choice.

The potential will be a function of the state of the system at a given time. We define this "state" as follows, by the neurons which exist at this time and their properties. The neuron with label i is at a location (site) $(x_i, y_i) \in Z^2$, has activity (membrane potential) $u_i(t)$ and a "number of free synapses", $s_i(t)$. It also has a number of free axons, $a_i(t)$, less than or equal to its generation counter $g_i = a_i(t_*)$ (t_* is the time of creation of the i-th neuron). The neuron is excitatory if $N(x_i, y_i) = 1$, inhibitive if $N(x_i, y_i) = -1$. We

sometimes use the shorthand $N_i = N(x_i, y_i)$.

The state of the system at time t, when N neurons have been created, is then given by the set of quadruples $\{(x_i, y_i, s_i(t), N_i); i = 1, \ldots, N\}$. Notice that neither the membrane potential, nor the number of free axons, nor the existing connections play a part in the definition of "state".

Modifications of our model could certainly include these other characteristics into the definition of "state". However, the current potential depends only on the variables defined above. We tried various candidates for potentials; the one which seemed to lead to satisfactory patterns is the following:

$$V(x, y) = \sum_{i=1}^{N} \frac{S/2 - s_i + 1}{1 + |x - x_i| + |y - y_i|} \tag{2.1}$$

Note that a neuron at site (x_i, y_i) will make a negative contribution to the potential while it has many free synapses; as its number of free synapses, s_i, decreases, this potential will gradually increase and eventually become positive.

If our potential were a potential in the usual physical sense, forces between various neurons would act in the direction of the gradient of the potential. We loosely follow this idea in using the potential to give directions for sprouting or connecting. To do this, we need one more ingredient, namely the idea that neurons will look for sites to sprout to, or connection partners, in a neighborhood rather than far away. We implicitly work under the assumption that there is a natural scale of length for the typical connection, i.e., that the axon cannot exceed a certain length. This length defines a "window" around each neuron already in place. We used windows of the type

$$|x - x_j| + |y - y_j| \leq d_e, \quad |x - x_j| + |y - y_j| \leq d_i$$

where d_i is the window size for inhibitive neurons, d_e the size for excitatory neurons. Following biological evidence, we took $d_i > d_e$ ("long-range inhibition, short-range excitation"). If the neuron at site x_j, y_j is inhibitive, it can connect to another neuron inside its window. If it is excitatory, it can connect to another neuron in its (smaller) window, or it can sprout a new

neuron to an empty site inside this window. The details of these sprouting and connecting routines are described in what follows.

2.3. The sprouting and connecting routines. These procedures are outlined here in detail, but they are also the content of the flow chart in Figure 2 (see Appendix 3). In the program, the routines are contained in the subroutine "*alter_net*". Assume that we have arrived at a certain net after a number of steps, and that the activity of some of the existing neurons, either by external or internal stimulation, is above the threshold. We then loop through all the neurons and make the following tests and choices:

A. For the $i-$th neuron, check whether the activity is above the threshold. If no, go to the next neuron, If yes, go to B.

B. For the i-th neuron, check whether $a_i = 0$. If yes, go to the next neuron. If $a_i \geq 1$ and the neuron is inhibitive, go to C. If $a_i = 1$ and the neuron is excitatory, go to E. If $a_i > 1$ and the neuron is excitatory, go to D.

C. If the i-th neuron is inhibitive, it will look for the place inside its window which is occupied by a neuron with at least one free synapse and where the potential is *maximal.* Neurons to which our "current neuron" is already connected are disregarded in this search. From the way that our potential was defined, the inhibitive neuron will typically be targetted towards neurons which already have many synaptic connections and are therefore likely to be highly active. If there are several sites which have the same maximal potential and are occupied by connection candidates, the neuron chooses one randomly for connection. Then go to F.

D. If the i-th neuron is excitatory, it will look for the sites inside its window which have *minimal* potential and are either empty or occupied by a neuron with at least one free synapse, and without a connection from the i-th neuron. If there is more than one site in this set, a random choice is made. If the chosen site is empty, the neuron will sprout a new neuron to the site; then go to G. If the chosen site is occupied, neuron i will connect to the neuron which already exists there; then go to F.

As the neuron sprouts or connects, its number of free axons $a_i(t)$ is reduced by one.

E. If $a_i = 1$, then this excitatory neuron must connect. In doing so, it follows the routine from D, where empty sites are now disregarded.

F. For the connection, we simply postulate a connection between the current neuron and the chosen one. As for the connection strength, see 2.4. In the process, the $i-$th neuron's number of leftover axons (a_i) is reduced by one, and the number of leftover synapses s_j of the neuron receiving the new connection is also reduced by one.

G. Sprout. Just postulate a new neuron at the empty site, and set the connection weight as in 2.4. The number of free axons of the i-th neuron, a_i, is reduced by one. The new neuron is assigned a generation counter 1 less than its parent. The choice whether it be excitatory or inhibitive is made randomly, usually biased towards the excitatory type with a weight called "xratio" (a free parameter in the program).

Notice that the sprouting process will automatically come to an end with the last generation, and the structure of the net will have been created by its own activity, and hence, to some degree, by the inputs which were active during the process.

2.4. The connection weights. We used a connectivity function which creates a connection weight between the $j-$th (sending) and $i-$th (receiving) neuron as

$$C(i,j) = \pm\lambda \mathrm{sgm}(\mu u_i)\mathrm{sgm}(\mu u_j), \tag{2.2}$$

with the "+"-sign if j is excitatory, the "-"- sign otherwise. In terms of the firing rates v,

$$C(i,j) = \pm\lambda v_i v_j,$$

and we set the connection matrix entry

$$T_{ij} = C(i,j). \tag{2.3}$$

2.5. The dynamics of the system. The dynamics of the system happen on two different time scales: A fast one for the evolution of the individual neurons' membrane potentials, given by equations (2.1), and a slow one for alterations of the net, involving the growth process, the creation of new connections and the setting of new connection strengths in accordance with (2.2-3). Also, all connection strengths already in existence get updated on the slow time scale. In practice, we solved Eqns. (2.1) by the Matlab routine for 20 time units, and then stopped to update the system with the routine "alternet". Any connections created at this time get the connection strength defined by (2.3), while the already existing connectivities are updated according to

$$T_{ij}(\text{new}) = (1-\epsilon)T_{ij}(\text{old}) + \epsilon C(i,j). \tag{2.4}$$

Here, ϵ is a free parameter which determines how rapidly the connection strengths change. Formula (2.4) is a generalized Hebb rule.

Finally, we add a mechanism to delete inactive connections and therefore make room for true alterations in the net. This mechanism is to kick in when the receiving neuron is firing at a low rate *and* when the connectivity is below a certain threshold. In the computer simulation, we chose the rule

H. If sgm$u_i < 0.01$ and if $|T_{ij}| < 0.05$ then set $T_{ij} = 0$ (connections with zero connection strengths are considered as severed). Then increase the numbers of free synapses of neuron i and the number of free axons of neuron j by one.

2.6. Input-output representation, graphical depiction of the net and of the potential.

This is done in three windows brought up by Matlab. All three windows are square with a given side length (a parameter which can be set freely; for our tests we chose side length 30. As the neurons are placed at grid points, there is room for maximal 900 neurons). The first window shows each neuron which has so far been created; the second window does the same, but it shows

in addition all the existing connections plus, by color, the activity of all the neurons at the particular time instant. The third window shows the potential as a color map. The only purpose of this map is to indicate in which direction the net will grow at a given time.

In the first two windows, we found it easiest to display the neurons as triangles. The purpose of the first window is to choose neurons which are to be fed external input, and to apply this input. Specifically, one can sketch a rectangle by downclicking the mouse button at a particular location (thereby setting a corner of the chosen rectangle), then move the cursor and eventually release it, thereby setting the corner on the other side of the diagonal of the rectangle. The rectangle will be shown in yellow. The input will then act on every neuron in the rectangle. The value of the input is between -10 and +10 and can be set by a slider, displayed at the bottom of this window.

A fourth display on the screen shows a little menu with the commands "Step", "Save", "Clear input" and "Quit". The latter two commands are self-explanatory. The first command starts a solution of the existing system of differential equations by the Matlab routine. After 20 time units, the system then goes through the routine "alternet". in this routine, new neurons and connections are produced; the net grows accordingly, and the display in windows 1 to 3 is automatically rescaled. The command "Save" enables the user to save the current evolution as a Matlab movie.

Almost all the relevant information is contained in window nr. 2, where the whole net and its activity is shown. We found it most convenient to display the connections as piecewise straight lines with one "kink". The idea behind this is that ordinary straight lines would make it very hard to see the connection structure after even the very first few steps (because connections could overlap each other), and it is difficult to produce curved connections. In order to minimize the chances that connections would fall on top of each other, we chose the following geometric construction: Divide the distance between the neurons to be connected by three. Along this imaginary line, at the point one third away from the sending (or sprouting) neuron, step to the left from the connecting line at a right angle, and a distance $1/\pi$ away.

The point reached is "the kink" in the connection. This procedure has the advantage that relative to the axis given by the two neurons, the "kink" has always a rational and an irrational coordinate (to the accuracy of the computer and the display, of course). It is therefore extremely unlikely that two connections fall on top of each other.

We emphasize that this rule has noting to do with biology; we just chose it because it works. Many other rules will do just as well.

As for colors, we chose green to display the connections coming from excitatory neurons, red for connections from inhibitory neurons. Connections which have been severed because of inactivity are displayed in dark blue. The colors are easily changed by choosing a different colormap from the Matlab library.

3. Results, and the Future.

We conducted extensive tests with the program and made the following observations.

3.1. In most cases, the net will grow rapidly if there is sufficient initial stimulation. The input leads to high activity of the mother neuron, to sprouting, to high activity of the next generation, which sprouts or reconnects, and so on. The process can end prematurely if too many inhibitive neurons are created early in the game. These must connect to other neurons, whose activity then gets inhibited to the point where they may not sprout new neurons anymore. High stimulation of such neurons can lead to new growth.

Typically, however, when the number of inhibitors is kept low, once a few neurons are in place, growth continues even when input is turned off. The chosen growth rule, which directs new excitatory (inhibitory) neurons to locations of lowest (highest) potential, appears to lead naturally to the formation of subnets (clusters). These clusters will usually form inhibitive connections to other clusters and back to the core. The reason is the longer range of inhibitive connections. It follows that the inhibitive neurons in the outlying clusters are the most sensitive neurons to inputs. Indeed, it was seen that these inhibitors, when they are active, inhibit the activity of the

core to a large degree, such that the majority of the core neurons shows low activity in the absence of any external inputs. This activity is not necessarily stationary: we saw periodic and likely chaotic behavior (the difference is not easily told from a color display on the screen), but most of the core neurons at any given time show low activity.

This changes if one or several of the inhibitors at the fringe receive an *inhibitive* input, which decreases the activity of this neuron. There is usually a characteristic response to this input; part of the core shows increased activity, and the generic response pattern there is either increased cyclic (or chaotic) activity, or convergence to a more active steady state. In this sense, the grown networks show clear information processing abilities.

This description indicates that *"xratio"* is one of the most important parameters. If there is too little inhibition, such mostly excitatory nets tend to freeze in high-activity states, whereas too much inhibition leads to a freeze in low activity states. Such nets still respond to input, but in a fairly trivial way—neurons directly affected by input, or directly connected to a neuron receiving input, show a response, but the rest of the net seems rather unaffected. It follows that a proper ratio between inhibitive and excitatory neurons is important.

Most of the tests which we conducted used the value xratio= .35.

3.2. The other most important parameter appears to be the "gain" μ. This is not very surpising, as the gain is well-known to influence the behavior of a neural network in sensitive ways. For low values of μ, the net tends to be in stable rest states for most of the time; inputs can force the net from one such steady state to another one, but it is questionable whether this is really an optimal way of information processing. First, it is not consistent with biological observations, where chaotic activity is seen in rest states and periodic attractors seem the typical responses to inputs. Second, it may well be that chaotic rest state activity increases the information capacity of a network in ways which we do not (yet) understand.

Be that as it may, for larger values of μ, (μ between 2 and 10) the characteristic rest states of our network are oscillatory (we do not really know

whether periodic or chaotic; it probably does not matter, the main advantage of both being access to large parts of phase space, with the ability to respond characteristically to many different types of input). The application of inputs to inhibitors in the clusters on the fringe usually pushes the activity of the core into a steady state. These states are sometimes stable, sometimes unstable, and sometimes metastable in the sense that they may persist in a slightly altered form when the input is removed, they may disappear and the system may become oscillatory again, or the system may move into another steady state. We observed all these.

For networks which have grown to more than about 50 neurons, it becomes hard to observe the response to a given input. Our impression is that nets which have grown to this size show more stable behavior. We speculate that the ground rules should be changed slightly as the size of the net passes certain thresholds in order to retain the sensitive behavior to inputs. For example, the connection range for inhibitive neurons could be increased; *"xratio"* could be increased; clustering of input neurons could be enforced to guarantee a stronger signal, etc.

3.3. Figures 3 to 5 in Appendix 3 show configurations of nets which grew in this way and displayed interesting behavior. We hope that the reader is inspired. Unfortunately, the journal format is inappropriate to show the dynamic behavior of such a net— in a typical experiment, we let a network grow to about 20-30 neurons and then respond to a variety of inputs. This required from anywhere of 50-100 frames (each frame corresponds to a step, in which the "*alter_net*" routine is called and in which the network equations are solved for about 20 time units). This requires little computing effort on a modern workstation, but it is impossible to present 50-100 pictures here (and this would still be only one single evolution). To remedy the situation, we produced 13 movies, which can be run using Matlab by anybody with access to a workstation and Matlab.

To obtain the movies and the program, the reader should use "netscape" and connect to the address

http://maxwell.math.uvic.ca/faculty/rillner/neural-net

the main program (in compressed form) is in "*netbrain*". By clicking on it, you are offered the option to save it in your active directory (you can give it any name you want, but netbrain is our choice). Similarly, you can transfer any of the movies 1-13. We consider the ones with nrs 10, 12 and 13 as the most interesting ones, not only because interesting activity is visible, but also because the location of inputs are visible in these later movies.

To run the program, follow these steps (these instructions are also available via the WWW):

1. Type "uncompress netbrain.tar.Z"
2. Type "tar xvf netbrain.tar"

These steps will create a directory netbrain, which contains all the subroutines wich comprise the brain program. To run it, type and enter

1. cd netbrain
2. matlab
3. brain

The windows described earlier in the paper should now appear on your screen, and you are ready to grow your own network. It will likely take a few attempts to produce an interesting one.

To run, e.g., movie nr. xx, type and enter the following commands:

1. uncompress moviexx.mat.Z
2. matlab
3. load moviexx
4. colormap(cool)
5. movie(M,1,1)

We present a short summary of the more interesting movies (in our opinion) with nrs. 10, 12 and 13 in Appendix 1. Figures 4 and 5 in Appendix 3 are the final frame from movies 10 and 12, respectively. Appendix 2 contains the program, which consists of a number of subroutines, all called in various places. The main program is called "brain", and if you want to run tests, you have to type "brain" after entering Matlab to start the menu-driven process.

The question arises what to do with all this in the future. Our objective in this project was to find generic rules which would lead to the growth of artificial networks with interesting properties, in particular with properties resembling those of primitive natural neural networks. We don't really know whether or not we have succeeded in doing this. What we did achieve was a demonstration that a few simple ground rules and the proper choice of a few parameters are sufficient to produce nets with behavior that at least mimicks properties seem in primitive brains. We have to be content with this, but our purpose in disseminating the results, the movies, and the program, have an ulterior motive: we hope that many readers with access to computers will run our program, will play with it, and will eventually begin to choose other values of the available parameters, and, finally, tinker with the rules themselves. It is our hope that our self-growing neural networks will become subject to this kind of computer evolution. If the objective discussed in this paper is realistic, then, we believe, there is a real chance that such experiments will eventually lead to truly exciting results.

Acknowledgement. This research was supported by grant nr. A 7847 of the Natural Sciences and Engineering Research Council of Canada, and by an internal research grant of the University of Victoria. The authors would like to thank Kelly Choo for his help in making the program, the movies and the instructions available to the public by placing them on R. Illner's home page in the World Wide Web. We are also grateful to Nikitas Dimopoulos and Patrice Snopkowski for helping us with the pictures.

REFERENCES

[A] Amit, *Modeling Brain Function: The World of Attractor Neural Networks,* Cambridge University Press, Cambridge, Mass. (1989)

[Co] Cottet, G.H., *Modèles de réaction-diffusion pour des réseaux de neurons stochastiques er déterministes,* C.R.A.S. Paris, 312-I, 217-221 (1991)

[Ed] Edwards, R., *Approximation of Neural Network Dynamics by Reaction-Diffusion Equations,* Math. Meth. Appl. Sci., to appear

[Ed2] Edwards, R., *Ph.D.-thesis,* University of Victoria, Victoria, B.C. 1994

[EIK] Evans, N., Illner, R., and Kwan, H., *On Information-Processing Abilities of Chaotic Dynamical Systems,* Journal Stat. Phys. 66, 549-561 (1992)

[Gr] Grossberg, S., *Nonlinear Neural Networks: Principles, Mechanisms and Architectures,* Neural Networks 1, 17-61 (1988)

[HKP] Hertz, J., Krogh, A., and Palmer, R., *Introduction to the Theory of Neural Computation,* Addison-Wesley (1991)

[Ha] Hayflick, L., *Human Cells and Aging,* Scientific American (1968)

[He] Hebb, D., *The Organization of Behavior,* Wiley, New York (1949)

[Li] Lippmann, R.P., *An Introduction to Computing with Neural Nets,* IEEE ASSP Magazine 4(2), 4-22 (1987)

[PBK] Priesol, A.J., Borrett, D.S., and Kwan, H.C., *Dynamics of a Chaotic Neural Network in Response to a Sustained Stimulus,* Technical Reports on Research in Biological and Computational Vision, No. RBCV-TR-91-38, Dept. of Computer Science, University of Toronto, 1991

APPENDIX 1: ABOUT THE MOVIES

USING MOVIES IN MATLAB

A movie of successive figures can be created in matlab using its built in functions. The information for frame j is saved in column j of a large matrix which is initialized before creating the movie. As a result, matlab allows variables (such as the matrix) to be saved so that it can be retrieved later and played back. Specifics on the matlab commands 'movie', 'getframe' and 'moviein' can be obtained from the matlab help files (help <command>). One drawback of the movies is that the range of values for the axes is not stored.

In the subroutines of the brain program, each time the net is altered (ie. network figure changes), the previous frame is saved. This allows for the complete history of a growing net to be recorded. Since the replay will not show the appropriate (x,y) coordinates of each neuron, it may be important to visualise when a neuron receives an input. The rectangular boxes which select neurons for input can be activated on the network figure (as well as the input field figure). This allows the rectangle to be saved in the previous movie frame. As a result, when playback occurs, the neuron which is to receive input in the next frame can be discerned. All frame saving and matrix updating is done by the brain program after each step.

To save a movie (eg. before quitting), clicking on the 'save' button will record all previous steps (note: not current figure) to the default file 'movie.mat' in the matrix M. This filename must be hardcoded in the program due to constraints by matlab, but can be changed by altering the 'saveit.m' file. A word of warning before saving is that these matricies can get very large and take up a huge amount of disk space (70 frames ~ 10 Meg.). However, these files can be compressed using the UNIX 'compress' command, which reduces size by 95+%.

To restore a saved movie, it must first be loaded in matlab. The commands described in the following section are described in much more technical detail in the document 'refguide.txt'. Any number of frames can be replayed at any chosen speed. To get an accurate movie, the colour scheme must be changed to 'cool' and to eliminate the false axes, these must subsequently be shut off. Note that before matlab plays a movie, it must first assemble the frames, so it plays through the entire movie once as it compiles the information. Once this has been completed, it will then proceed in playing the movie at the chosen speed for the number of repetitions specified.

QUICK REFERENCE GUIDE TO MOVIES AND NETWORK SIMULATOR

Matlab command	Result of action
movie(M,1,<fps>)	Plays movie matrix M once at <fps> frames per second
save <pn>/<fn> M	Saves the movie matrix M in a file in selected directory
load <pn>/<fn>	Loads all variables from your file
colormap(cool)	This defines our colour scheme. It is not the default.
axis('off')	Removes the figure axes
G=M(:,a:b);	Creates a new movie matrix G from frames a to b of M
G=M(:,c);	G contains only frame c of matrix M (eg. for printing)
G=M(:,[d,e,f,...]);	Matrix contains frames d,e,f,... in that order
capture(<fig>)	Use this command to copy a movie figure to a printable figure (which looks identical)
print -dpsc <fn>	Saves active figure to a postscript colour file. Note that this command will only work on a 'captured' figure window (uncaptured movie figures don't save properly)
uncompress <fn>.<ext>.Z	Decompresses file (eg. movie4.mat.Z) (or double click)
compress <fn>.<ext>	Compresses your file (saves ~97% of space for matricies)

<fps> frames per second
<pn> full path name of all directories
<fn> file name (no extension needed)
<ext> file extension
<fig> figure number of brain to be captured (eg. 1 for Figure No.1)

MATLAB NEURAL NETWORK TEST RUN - MOVIE10

Parameter	Value	Description
max_gen	5	Maximum number of generations of neurons allowed
max_synapse	5	Maximum number of synaptic connections per neuron
threshold	0.5	Limit of potential for sprouting or connecting
shift	0.5	Used in sigmoid by inhibitory neurons
xratio	0.35	Limit ratio for proportion of inhibitive neurons
lambda	2	Connectivity weight factor (gain)
mu	6	Gain in sigmoid function
d(excititory)	3	Maximum distance for an exitatory neuron's axon
d(inhibitory)	6	Maximum distance for an inhibitory neuron's axon

Frame Number(s)	Action	Description
	Input +3.0	
1-5	Grow	Initial growth from mother neuron
	Clear Input	
6-40	Grow/Observe	Network saturates and exibits chaos
	Input -10.0	Inhibit a group of three neurons
41-44	Observe	Network instantly reaches equilibrium
	Clear Input	(No box shown in frame 44)
45-52	Observe	Network returns to chaotic behaviour
	Input -10.0	Inhibit one neuron this time
53-62	Observe	Choas now occurs
	Clear Input	
63-64	Observe	Choas continues
	Input -3.0	Inhibit same neuron, but not as much
65-90	Observe	A nearly cyclic pattern seems to result

MATLAB NEURAL NETWORK TEST RUN - MOVIE12

Parameter	Value	Description
max_gen	7	Maximum number of generations of neurons allowed
max_synapse	6	Maximum number of synaptic connections per neuron
threshold	0.5	Limit of potential for sprouting or connecting
shift	0.5	Used in sigmoid by inhibitory neurons
xratio	0.35	Limit ratio for proportion of inhibitive neurons
lambda	2	Connectivity weight factor (gain)
mu	6	Gain in sigmoid function
d(excititory)	3	Maximum distance for an exitatory neuron's axon
d(inhibitory)	6	Maximum distance for an inhibitory neuron's axon

Frame Number(s)	Action	Description
	Input +3.0	
1-5	Grow	Initial growth from mother neuron
	Clear Input	
6-27	Grow	Network saturates into state A
	Input -10.0	Inhibit group of 3 neurons
28-35	Observe	Equilibrium is achieved
	Clear Input	
36-41	Observe	Network returns to state A
	Input -10.0	Inhibit a different group of 3 neurons
42-47	Observe	A steady state is reached
	Clear Input	
48-51	Observe	Network goes to new state B

Notes: This network does not grow outwards as much as expected. It seems to grow in a dense manner and exhibits an unusual number of severed connections.

MATLAB NEURAL NETWORK TEST RUN - MOVIE13

Parameter	Value	Description
max_gen	5	Maximum number of generations of neurons allowed
max_synapse	5	Maximum number of synaptic connections per neuron
threshold	0.5	Limit of potential for sprouting or connecting
shift	0.5	Used in sigmoid by inhibitory neurons
xratio	0.35	Limit ratio for proportion of inhibitive neurons
lambda	.2	Connectivity weight factor (gain)
mu	6	Gain in sigmoid function
d(excititory)	4	Maximum distance for an exitatory neuron's axon
d(inhibitory)	7	Maximum distance for an inhibitory neuron's axon

Frame Number(s)	Action	Description
	Input +3.0	
1-5	Grow	Initial growth from mother neuron
	Clear Input	
6-35	Grow	Network saturates and becomes chaotic
	Input -10.0	
36-40	Observe	More chaotic behaviour
	Clear Input	
41-46	Observe	Chaos continues
	Input -10.0	
47-51	Observe	A steady state is reached
	Clear Input	
52-56	Observe	Network returns to chaotic activity
	Input -10.0	Inhibit two periphery inhibitive neurons
57-61	Observe	Network reaches an equilibrium state
	Input -4.0	Reduce inhibition by 60%
62-71	Observe	Network now becomes chaotic
	Input +2.0	Now excite both neurons
72-75	Observe	A steady state is reached

Notes: This network has two inhibitive neurons on the periphery (RHS) and cause stimulus in the core (LHS) when they receive an input. It appears that for mild stimulus, chaotic behaviour is unaffected. When larger inhibitory or somewhat excititory inputs are introduced, the core instantly achieves an equilibrium state.

APPENDIX 2: THE PROGRAM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Alter_net
%   Alters the network according to the rules of our model.  For each neuron
%   in the network, check whether it is able to undergo a growth process and if
%   so, either try to form a connection to an existing neuron, or sprout a new
%   one.
%-------------------------------------------------------------------------------

% Constant specifying the window size inside which we connect or sprout;
global d;

%  Update the connection matrix - slow 'learning' process
learn;

%  Don't use the variable 'last' for the loop, as it can change inside the loop
%   as the number of neurons is increased.  So put the current number of
%   neurons in a new variable, 'endloop'.
endloop = last;

for this = 1:endloop

% Membrane potential must be greater than threshold....

  if u(this) >= threshold

%   also, this neuron must have the ability to put out new axonal
%    connections

    nelig = 0;

%    Give inhibitory neurons a longer range window for connections

    if (neuron_type(sites(this,1),sites(this,2)) == 1)
      d = 3;
    else
      d = 6;
    end

% Allow only connection (not sprouting) if this is the last connection this
%  neuron can make, or if it is inhibitory and can still make connections.
    if (axons_left(this) == 1) | ...
       ((axons_left(this) > 1) & ...
        (neuron_type(sites(this,1),sites(this,2)) == -1))
      minpot = Inf;
      i0 = sites(this,1);
      j0 = sites(this,2);

      for i = max(1,i0 - d) : min(SIZE,i0 + d)
%  Note: j can be allowed to go down to 1 if we don't display our scale in
%              the bottom row
        for j = max(2,j0 - d + abs(i-i0)) : min(SIZE,j0 + d - abs(i-i0))
% don't allow self connections
          if (i ~= i0) | (j ~= j0)

            if O(i,j) > 0

              if (T(O(i,j),this) == 0) & (synapses_left(O(i,j)) > 0)

%    Give inhibitory neurons a bias towards long range connections
```

```
            if (neuron_type(sites(this,1),sites(this,2)) == 1)
              pot = Vfield(i,j);

            else
               pot = -Vfield(i,j);
            end

            if pot < minpot

              nelig = 1;
              elig_site(nelig,1) = i;
              elig_site(nelig,2) = j;
              minpot = pot;

            elseif pot == minpot

              if pot < Inf

                nelig = nelig+1;
                elig_site(nelig,1) = i;
                elig_site(nelig,2) = j;

              end
            end

          end
        end
       end
      end
    end

    if nelig > 0

      x = rand;
      n = ceil(x*nelig);
      l = O(elig_site(n,1),elig_site(n,2));

      connect;
    end

  elseif (axons_left(this) > 1)

    minpot = Inf;
    i0 = sites(this,1);
    j0 = sites(this,2);

    for i = max(1,i0 - d) : min(SIZE,i0 + d)
      for j = max(1,j0 - d + abs(i-i0)) : min(SIZE,j0 + d - abs(i-i0))

        if O(i,j) == 0
          goahead = 1;

        elseif T(O(i,j),this) == 0
          goahead = 1;

        else goahead = 0;

        end

% don't allow self connections
        if i == i0 & j == j0
          goahead = 0;
        end

        if goahead == 1
```

```
%    Give inhibitory neurons a bias towards long range connections
              if (neuron_type(sites(this,1),sites(this,2)) == 1)
                pot = Vfield(i,j);
              else
                pot = -Vfield(i,j);
              end

              if pot < minpot
                nelig = 1;
                elig_site(nelig,1) = i;
                elig_site(nelig,2) = j;
                minpot = pot;

              elseif pot == minpot
                if pot < Inf
                  nelig = nelig + 1;
                  elig_site(nelig,1) = i;
                  elig_site(nelig,2) = j;
                end
              end
            end
          end
        end

        if nelig > 0
          x = rand;
          n = ceil(x*nelig);
          l = O(elig_site(n,1),elig_site(n,2));

          if l == 0
            sprout;
          else
            connect;
          end
        end
      end
    end
  end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Brain  --  main program in matlab
%
%  Grows a 2-dimensional neural network, by steps operated by button clicks,
%  and allowing changing of neural input interactively.  Starts from
%  a single neuron.
%  Three windows are displayed, showing the following information:
%    1.  Input field.
%    2.  Network (neurons,connections, activities).
%    3.  Potential Field

%-------------------------------------------------------------------------------
%  General notes on programming:
%   Variables from the main program that are used by functions must be declared
%     global both here and in the function itself.
%   Variables from the main program that are used in .m files that are not
%     functions need not be declared as globals as the .m file is pulled in as
%     if it were part of the main program.
%   The above three display windows are referenced by the 'figure' command in
%     matlab, the first being figure(1), the second, figure(2), the third,
%     figure(3).
```

```
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%  Declaration of matrices and vectors
%    Matlab works quicker if you pre-define matrix and vector sizes
%-------------------------------------------------------------------------

LOTS = 200;
SIZE = 30;

% Neural activity vector (column)
u = zeros(LOTS,1);

% External input vector (column)
global inpvec;
inpvec = zeros(LOTS,1);

% Array of line colours for displayed connection lines
linecol = zeros(LOTS,1);

% Generation number vector (column) -- attached to the neuron of the same index
generation = zeros(LOTS,1);

% Number of potential axonal connections (outgoing) remaining for a neuron
%  column vector  -- attached to the neuron of the same index
axons_left = zeros(LOTS,1);

% Number of potential synaptic connections (incoming) remaining for a neuron
%  column vector  -- attached to the neuron of the same index
synapses_left = zeros(LOTS,1);

% Connectivity matrix
global T;
T = sparse(LOTS,LOTS);

% Array of indices for connecting lines associated with each connection
connindex = zeros(LOTS,LOTS);

% Occupancy matrix
O = zeros(SIZE,SIZE);

% Neuron type
global neuron_type;
neuron_type = zeros(SIZE,SIZE);

%shifts vector
global shifts;
shifts = zeros(LOTS,1);

%Potential
Vfield = zeros(SIZE,SIZE);

% Display version of potential
potdisp = zeros(SIZE,SIZE);

% Sites
global sites;
sites = zeros(LOTS,2);

% Movie matrix
global M

% frame counter
global count;
```

```
% The last neuron created (and also the current number of neurons in the net).
global last;

%-------------------------------------------------------------------------
%  Initialize constants
%-------------------------------------------------------------------------

% Constant specifying the maximum number of generations of neurons allowed
max_gen = 5;

% Constant specifying size of neurons in display
neuronsize = .4;

% Constant specifying the maximum number of synaptic connections which a neuron
%  can get
global max_synapse;
max_synapse = 5;

% Threshold for sprouting or connecting -- a neuron's membrane potential
%  must exceed this to grow a new connection or a new neuron.
%  Currently fixed as a constant.
threshold = .5;

%shift used in sigmoid by inhibitory neurons
shift = 0.5;

%Initialising the random number
timearray = clock;
random_seed = timearray(6)*1000000;
rand('seed',random_seed);

% Random parameter 0<p<1 to determine whether a sprouted neuron is excitatory
%  or inhibitory
xratio = 0.35;

% Start movie matrix recording at first frame
count = 1;

%-------------------------------------------------------------------------
%  Initialize variables used throughout the program
%-------------------------------------------------------------------------

% Start with one neuron (the mother)
last = 1;

%  Location of first neuron
sites(1,1:2) = [SIZE/2,SIZE/2];
%  Index of first neuron
O(SIZE/2,SIZE/2) = 1;

%  Mother neuron is excitatory
neuron_type(SIZE/2,SIZE/2) = 1;

% number of connecting lines in display
nline = 0;

% Initialize arrays for the first neuron (the others contain all zeros from
%  above)
generation(1) = 1;
axons_left(1) = max_gen;
synapses_left(1) = max_synapse;
% updateS (change the potential)

   for i = 1:SIZE
     for j = 1:SIZE
```

```
      dist = abs(i - SIZE/2) + abs(j - SIZE/2);
      Vfield(i,j) = (max_synapse/2.0 - synapses_left(last) + 1)/(1+dist);
    end
  end

% Initialize the input for the mother neuron (this can be changed
%  interactively before starting the growth process)
  inpvec(1) = 3;

% Stepsize, initial time and end time for first integration step --
%  subsequently t0 and tf will be incremented by stepsize for each step.
stepsize = 2;
t0 = 0;
tf = stepsize;

%-------------------------------------------------------------------------
%  Define the three display window figures, with appropriate titles and
%   positions on the screen.
%-------------------------------------------------------------------------

h1=figure('Name','Input Field','NumberTitle','off','Position',...
          [50,600,550,400]);
caxis([-1.0,1.0])
colormap(cool)

h2=figure('Name','Network','NumberTitle','off','Position',...
          [650,600,550,400]);
caxis([0.0,1.0])
colormap(cool)

% Initialize movie matrix.  For movies of more than 10 frames, matlab will
%  automatically increase the size of matrix M.
M = moviein(1);

h3=figure('Name','Potential','NumberTitle','off','Position',...
          [650,150,550,380]);
%  scale potential between -1 and 1 (as it can be negative)
caxis([-1.0,1.0])
colormap(cool)

%-------------------------------------------------------------------------
%  Set up coordinates of first neuron's display triangle and draw it
%-------------------------------------------------------------------------

clear triX triY;
clear triXin triYin;
clear inboxX inboxY;

%  Coordinate matrix -- each column of this matrix gives the (x,y)
%   coordinates of one of the three corners of the triangle.
%   According to matlab syntax, C is defined row by row.
C = [SIZE/2 SIZE/2+neuronsize SIZE/2; SIZE/2 SIZE/2 SIZE/2+neuronsize];

%  Extract the rows of the coordinate matrix, i.e the x's in one vector, the
%   y's in another
xs = C(1,1:3);
ys = C(2,1:3);

%  triX, triY  store the coordinates of the display triangles for all
%   the neurons in a form usable by the patch command.  Here we put the three
%   x coords for the first neuron in the first column of triX, and similarly
%   for y's.
triX(1:3,1) = xs';
triY(1:3,1) = ys';
```

```
%  Activate the window for the network
figure(2);

%  Draw the first neuron's triangle, giving it the colour associated with
%   its activity
v = sgm(u);
patch(triX(1:3,1),triY(1:3,1),v(1)')

%-------------------------------------------------------------------------------
%  Now display the potential function
%-------------------------------------------------------------------------------

%  Activate appropriate display window
figure(3);

% Convert to display scale (-1 to 1)
potdisp = display_sgm(Vfield);

%  Put in scale along bottom row:
for i=1:SIZE
  potdisp(i,1) = -1.0 + 2.0*(i-1)/(SIZE-1);
end

%  Display the potential matrix
pcolor(potdisp');

%-------------------------------------------------------------------------------
%  Now display the input field
%-------------------------------------------------------------------------------

%  Initialize the active input window to the mother neuron only
corner1(1:2) = [SIZE/2 SIZE/2];
corner2(1:2) = [SIZE/2 SIZE/2];

%  Activate appropriate display figure and plot the input field
plot_input;

% Set up button press functions for changing input
   firstcorner = [...
      'pv = get(gca,''CurrentPoint'');',...
      'corner1(1:2) = pv(1,1:2);'];

   rectangle = [...
      'pv = get(gca,''CurrentPoint'');',...
      'corner2(1:2) = pv(1,1:2);',...
      'delete(ibox);',...
      'drawbox;'];

   set(h1,'WindowButtonDownFcn',firstcorner)
   set(h1,'WindowButtonUpFcn',rectangle)

% set same functions up for getting coordinates from network display window
   set(h2,'WindowButtonDownFcn',firstcorner)
   set(h2,'WindowButtonUpFcn',rectangle)

% Slider for changing input values
   input_slider = uicontrol(gcf,'Style','slider','Position',...
      [200 0 120 20],'Min',-10,'Max',10,'Value',0,'Callback','get_input');

%-------------------------------------------------------------------------------
%  Now put a menu in a new figure with pushbuttons for stepping through the
%   time steps in the growth process and quitting from the program.  These
%   calls also define the callback routines (.m files) to be executed on each
%   button selection.
%-------------------------------------------------------------------------------
```

```
% create a figure just for the menu
h4=figure('Name','Menu','NumberTitle','off','Position',...
        [100,250,370,40]);

% Put the menu in the menu figure window
figure(4);

% Define pushbuttons for 'step' and 'quit' functions
stepbutton = uicontrol('Style','Pushbutton','String',' Step ','Callback',...
                       'do_step','Position',[10,10,80,20]);
clear_input_button = uicontrol('Style','Pushbutton','String','Clear Input',...
                               'Callback','clear_input','Position',...
                               [100,10,80,20]);
savebutton = uicontrol('Style','Pushbutton','String',' Save ','Callback',...
                        'saveit','Position',[190,10,80,20]);
quitbutton = uicontrol('Style','Pushbutton','String',' Quit ','Callback',...
                       'finish','Position',[280,10,80,20]);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Clear_input
%   Clear all inputs (i.e. set them all to zero)
%-------------------------------------------------------------------------------

% set the slider back to zero
set(input_slider,'Val',0);

for k = 1:last
  inpvec(k) = 0;
end

% Plot the result in the display window
plot_input;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Connect
%   Attempts to form a connection from the current neuron to one of the others
%   that is eligible for receiving synaptic connections.
%
%-------------------------------------------------------------------------------

axons_left(this) = axons_left(this) - 1;
synapses_left(1) = synapses_left(1) - 1;

%  Set the strength of this connection based on the current membrane potential
%   of the two neurons.
T(1,this) = connectivity(u(1),u(this),sites(1,1:2),sites(this,1:2));

% updateC: Update the potential following the establishment of the new connection.

for i=1:SIZE
%  Note: j can go down to 1 if not using bottom row for display scale
  for j=2:SIZE
    dist = abs(i - elig_site(n,1)) + abs(j - elig_site(n,2));
    if (synapses_left(1) == 0)
      Vfield(i,j) = Vfield(i,j) - max_synapse/(2*(1+dist));
```

```
    else
      Vfield(i,j) = Vfield(i,j) + 1/(1+dist);
    end
  end
end

%-------------------------------------------------------------------------
%  Draw the line between the two on network display
%-------------------------------------------------------------------------

%  The appropriate coordinates are saved in the triX, triY arrays
%  Find the 'middle' point, p, for the connection display
p1 = [triX(1,this),triY(1,this)]';
p2 = [triX(1,1) triY(1,1)]';
p = middle(p1,p2);

nline = nline+1;
connindex(1,this) = nline;

if (T(1,this) > 0)

%  Excitatory connection (green)
  linecol(nline) = 'g';
else

%  Inhibitory connection (red)
  linecol(nline) = 'r';
end

lineX(1:3,nline) = [p1(1),p(1),p2(1)]';
lineY(1:3,nline) = [p1(2),p(2),p2(2)]';



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
%  Connectivity
%   Function defining the connection strength based on Hebbian-style rule for
%    joint activities.

function z = connectivity(x,y,site1,site2)
global neuron_type;

lambda = .2;

%  Hebb rule but excitatory or inhibitory depending on type of sending
%   neuron.
   z = lambda*sgm(x)*sgm(y) * neuron_type(site2(1),site2(2));



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------
%  Display_sgm
%   Sigmoid transfer function for display purposes (vector input)
%-------------------------------------------------------------------------

function v=display_sgm(u)

  v = tanh(0.25*u);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Do_step
%    Called by clicking on the Step button
%    Does the integration for stepsize seconds (from time t0 to tf) using the
%     Hopfield net equations.  Currently uses the matlab ode23 command.
%    After the integration, alter the net based on the rules defined.
%-------------------------------------------------------------------------------
%  -differential equation defined by the 'hopfield.m' file,
%  -initial condition given by current activity vector, u
%  -output is a matrix of activity vectors at many small time steps, we just
%     need the last one to put back into our activity vector
%-------------------------------------------------------------------------------

toler = 1.e-2;
[t,result] = ode23('hopfield',t0,tf,u,toler);
temp = size(result,1);
u(1:last) = result(temp,1:last)';

% increment the start and end times for next iteration
t0 = tf;
tf = tf + stepsize;

% save frame of movie before continuing
figure(2);
M(:,count) = getframe;
count = count + 1;

%  Now alter the network
alter_net;

%  Activate the network display window and redraw the activities
figure(2);

%clear the axes before redrawing
cla
colormap(cool)
caxis([0.0,1.0])

%  Display of all lines
for i = 1:nline
  line(lineX(1:3,i),lineY(1:3,i),'Color',linecol(i));
end

v=sgm(u);
patch(triX(1:3,1:last),triY(1:3,1:last),v(1:last)')

%  Display the potential (scale it between -1 and 1)
figure(3);
cla
colormap(cool)
caxis([-1.0,1.0])

%  Note: don't reset the bottom row of the display potential, as it contains
%    the display scale
potdisp(1:SIZE,2:SIZE) = display_sgm(Vfield(1:SIZE,2:SIZE));
pcolor(potdisp');

% Display the input field
plot_input
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Drawbox
%   Draw a box around the input window in the input figure.
%-------------------------------------------------------------------------------

% Make sure corner coordinates in right order
if corner1(1) > corner2(1)
  temp = corner1(1);
  corner1(1) = corner2(1);
  corner2(1) = temp;
end
if corner1(2) > corner2(2)
  temp = corner1(2);
  corner1(2) = corner2(2);
  corner2(2) = temp;
end
corner1 = round(corner1);
corner2 = round(corner2);

%  Assume the input figure is active
hold on

% draw lines around the window
inboxX(1:5) = [corner1(1)-0.5; corner2(1)+0.5; corner2(1)+0.5;...
               corner1(1)-0.5; corner1(1)-0.5];
inboxY(1:5) = [corner1(2)-0.5; corner1(2)-0.5; corner2(2)+0.5;...
               corner2(2)+0.5; corner1(2)-0.5];
ibox = line(inboxX(1:5),inboxY(1:5));



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Finish
%     Do whatever exit is desired.  Here we just quit matlab.
%-------------------------------------------------------------------------------

% Clean up and exit matlab

disp('Quitting brain program and exiting matlab...');
exit;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Get_input
%
%-------------------------------------------------------------------------------

% get the value of the slider
level = get(input_slider,'Val');

% get corner coordinates in right order
if corner1(1) > corner2(1)
  temp = corner1(1);
  corner1(1) = corner2(1);
  corner2(1) = temp;
end
if corner1(2) > corner2(2)
  temp = corner1(2);
```

```
    corner1(2) = corner2(2);
    corner2(2) = temp;
  end
  corner1 = round(corner1);
  corner2 = round(corner2);

  % set the input level for all neurons in current rectangle.
  for i = corner1(1) : corner2(1)
    for j = corner1(2) : corner2(2)
      if (O(i,j) > 0)
        inpvec(O(i,j)) = level;
      end
    end
  end

  % Plot the result in the display window
  plot_input;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------------------------------
%  Hopfield
%    Function defining the differential equations of the Hopfield network.
%    Used by matlab's differential equation solving commands.
%-----------------------------------------------------------------------------

% t is time, x is the vector of quantities which change (neural activities)
function xdot = hopfield(t,x)

% globals needed within function call
global T;
global inpvec;
global shifts;

% Need input as a vector as well as indexed by site.
xdot = -x + T * sgm(x-shifts) + inpvec;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------------------------------
%  Learn
%    Update the connection matrix based on current activity
%-----------------------------------------------------------------------------

% Define local constant -- small time scale for learning
eps = 0.01;

% First, get firing rates from membrane potentials
v=sgm(u);

for i = 1:last
   for j = 1:last
      if T(i,j) ~= 0.0
        T(i,j) = (1-eps)*T(i,j)+eps*connectivity(u(i),u(j),sites(i,1:2),sites(j,1:2));
% Inactive connections are severed (inactive = small firing rate ???)
        if (v(i) < 0.01)
          if (abs(T(i,j)) < 0.05)

%    set conection strength to 0 to indicate  no connection
            T(i,j) = 0.000000;
```

```
%    change line colour to blue to indicate 'severed'
            linecol(connindex(i,j)) = 'b';

%    make the receiving neuron open for another synaptic connection and make
%    it eligible again if it wasn't
            synapses_left(i) = synapses_left(i)+1;

%    make the sending neuron open for another axonal connection
            axons_left(j) = axons_left(j) + 1;
          end
        end
      end
   end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------------------------------------------------------
%  Middle
%   Find the 'middle' point for drawing connections between two neurons
%------------------------------------------------------------------------------

function[p] = middle(p1,p2)
dif = p2-p1;
i = [0,-1;1,0];
L = norm(dif);
p = p1 + dif/3 + i*dif/(L*pi);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%------------------------------------------------------------------------------
%  Plot_input
%   Plot the input in its display window
%------------------------------------------------------------------------------

%  Activate appropriate display window
figure(1);

%  Clear the figure before redrawing 13-Jan-1995
cla
colormap(cool)
caxis([-1.0,1.0])

for k = 1:last
  triXin(1:3) = [sites(k,1); sites(k,1)+neuronsize; sites(k,1)];
  triYin(1:3) = [sites(k,2); sites(k,2)+neuronsize];
  inputcol = display_sgm(inpvec(k));
  patch(triXin(1:3),triYin(1:3),inputcol(1)')
end

%  Now draw the box around the input window
drawbox;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Saveit
%    Matlab does not allow a user to actively enter a save file path and name,
%     so we are forced to hard code it here
%-------------------------------------------------------------------------------

%  Saves current movie variable M into a file called 'movie.mat'.

savevar = input('File will be saved as ''movie.mat'' y/n [y]: ','s');
if isempty(savevar)
  savevar = 'y';
end

if ((savevar == 'Y') | (savevar =='y'))
  save movie M;
  disp('File saved contains movie matrix M');
  disp('Total number of frames is:');
  disp(count);
else
  disp('File not saved');
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Sgm
%   Sigmoid transfer function (vector input)
%-------------------------------------------------------------------------------

function v=sgm(u)

mu = 6;

  v = 0.5 * (tanh(mu * u) + ones(size(u)));
```

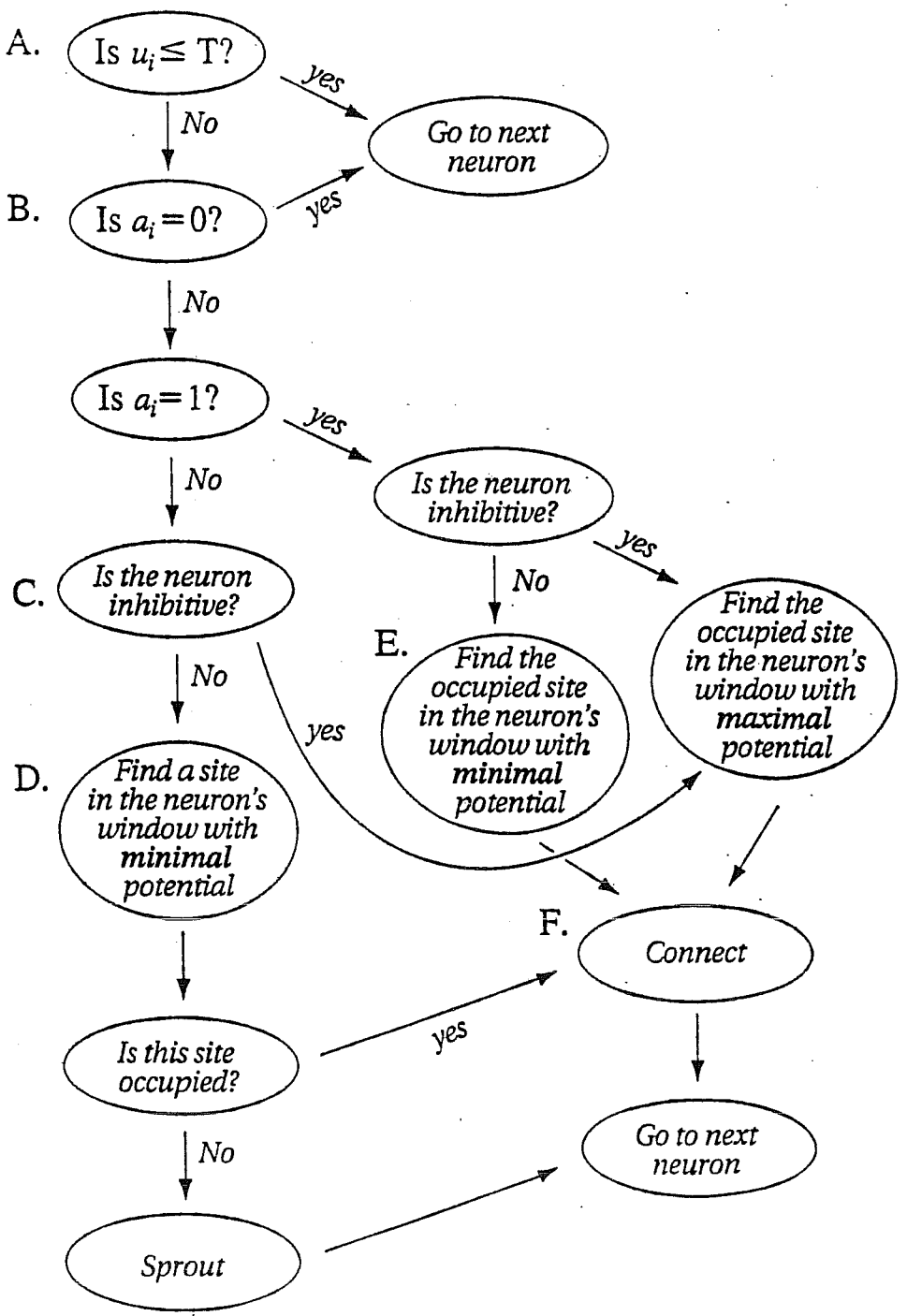
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-------------------------------------------------------------------------------
%  Sprout
%   Attempt to sprout a new neuron from the current neuron with a connection
%    from this one to the new one.
%-------------------------------------------------------------------------------

%  Increment the last neuron marker and fill out the connection matrix with a
%   new row and column full of zeros
last = last+1;

O(elig_site(n,1),elig_site(n,2)) = last;
sites(last,1:2) = elig_site(n,1:2);

% Assign the new neuron a type (excitatory (1) or inhibitory (-1))
temp = rand;
if temp > xratio
  neuron_type(elig_site(n,1),elig_site(n,2)) = 1;
else
  neuron_type(elig_site(n,1),elig_site(n,2)) = -1;

  shifts(last) = shift;
```

```
end

%  No input available for newly created neuron unless set manually...
inpvec(last) = 0;

%  Set the connection strength of the new connection
T(last,this) = connectivity(inpvec(this),u(this),sites(last,1:2),sites(this,1:2));

%  Initialize the new neuron:
%       0 membrane potential,
%       0 input if new group,
%       generation 1 up from its parent,
%       potential axonal connections decreasing with generation
%       potential synapses 1 less than max (it has one from parent)
u(last) = 0;

generation(last) = generation(this) + 1;
% axons_left(last) = max_gen - generation(last) + 1;
if generation(last) == max_gen
   axons_left(last) = 1;
else
   axons_left(last) = max_gen;
end
synapses_left(last) = max_synapse - 1;

%  Decrement the number of potential axonal connections remaining for the
%    current neuron.
axons_left(this) = axons_left(this) - 1;

% updateS (change the potential)

for i = 1:SIZE
%  Note: j can go down to 1 if not using bottom row for display scale
  for j = 2:SIZE
    dist = abs(i - elig_site(n,1)) + abs(j - elig_site(n,2));
    Vfield(i,j) = Vfield(i,j) + (max_synapse/2 - synapses_left(last) + 1)/(1+dist);
  end
end

%-------------------------------------------------------------------------------
%  Now, we locate the new neuron in space and draw its triangle and connection
%-------------------------------------------------------------------------------

%  Save the new neuron's coordinates
triX(1:3,last) = [sites(last,1);sites(last,1)+neuronsize;sites(last,1)];
triY(1:3,last) = [sites(last,2);sites(last,2);sites(last,2)+neuronsize];

%  The appropriate coordinates are saved in the triX, triY arrays
%  Find the 'middle' point, p, for the connection display
p1 = [triX(1,this),triY(1,this)]';
p2 = [triX(1,last) triY(1,last)]';
p = middle(p1,p2);

nline = nline+1;
connindex(last,this) = nline;

if (T(last,this) > 0)

%  Excitatory connection (green)
  linecol(nline) = 'g';
else

%  Inhibitory connection (red)
  linecol(nline) = 'r';
end
```

```
lineX(1:3,nline) = [p1(1),p(1),p2(1)]';
lineY(1:3,nline) = [p1(2),p(2),p2(2)]';
```

APPENDIX 3: FIGURES 2-5.

Figure 2. *Alternet*



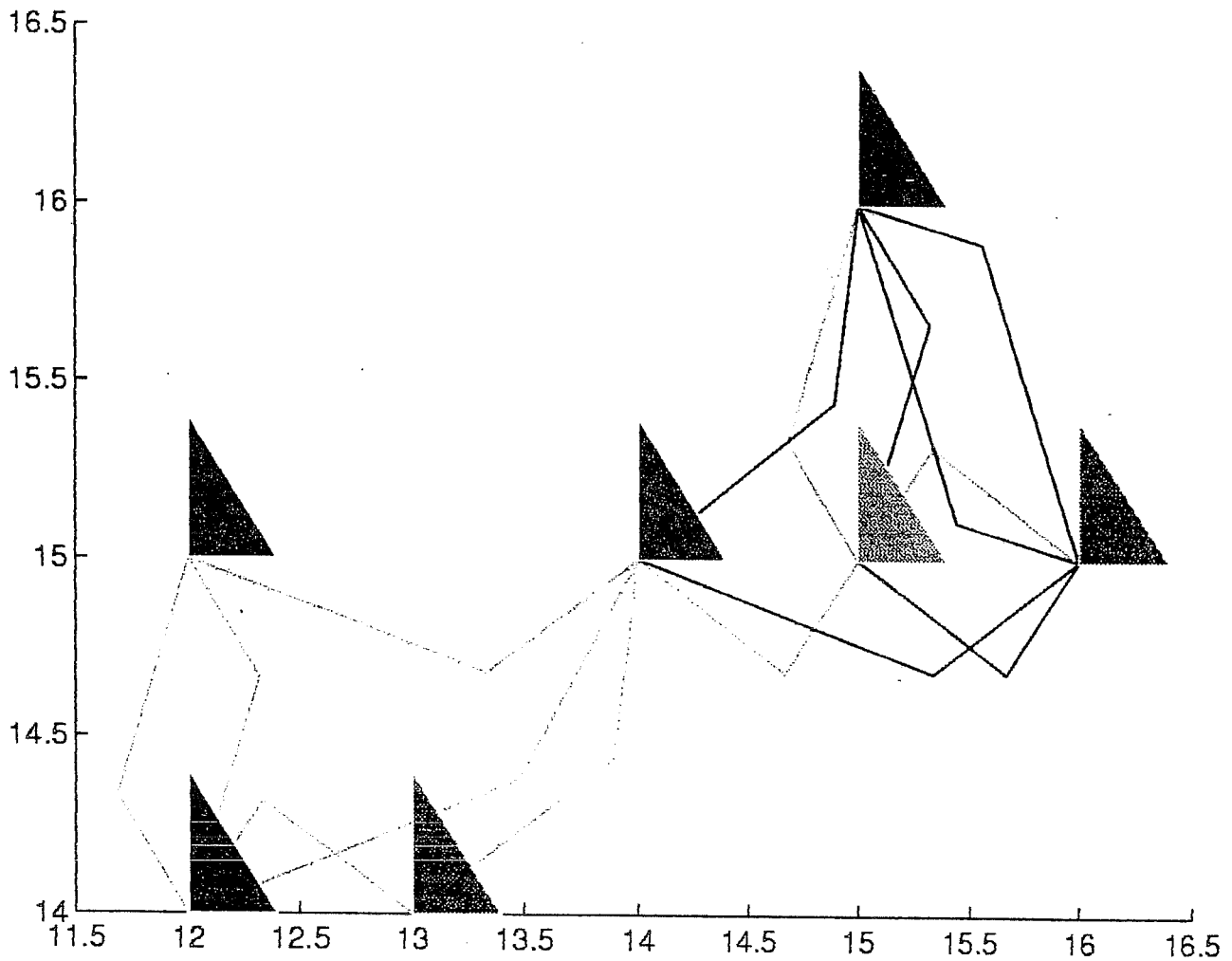


Figure 3: A typical network.

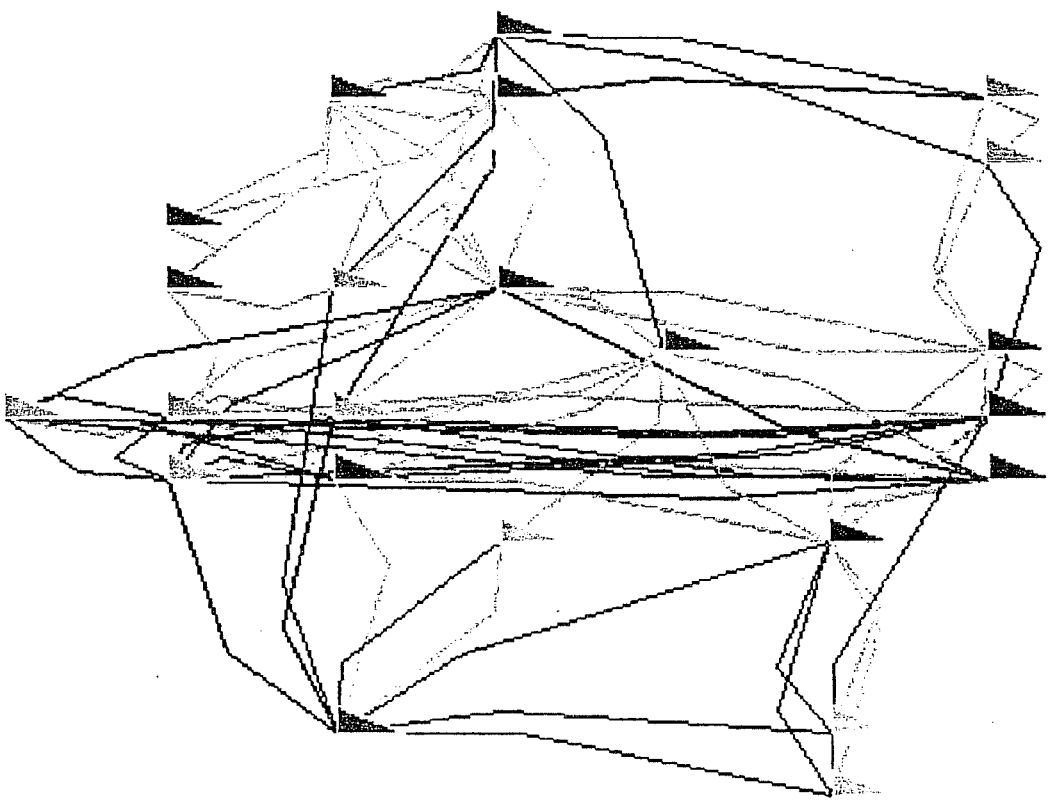


Figure 4: The final frame of movie 10.

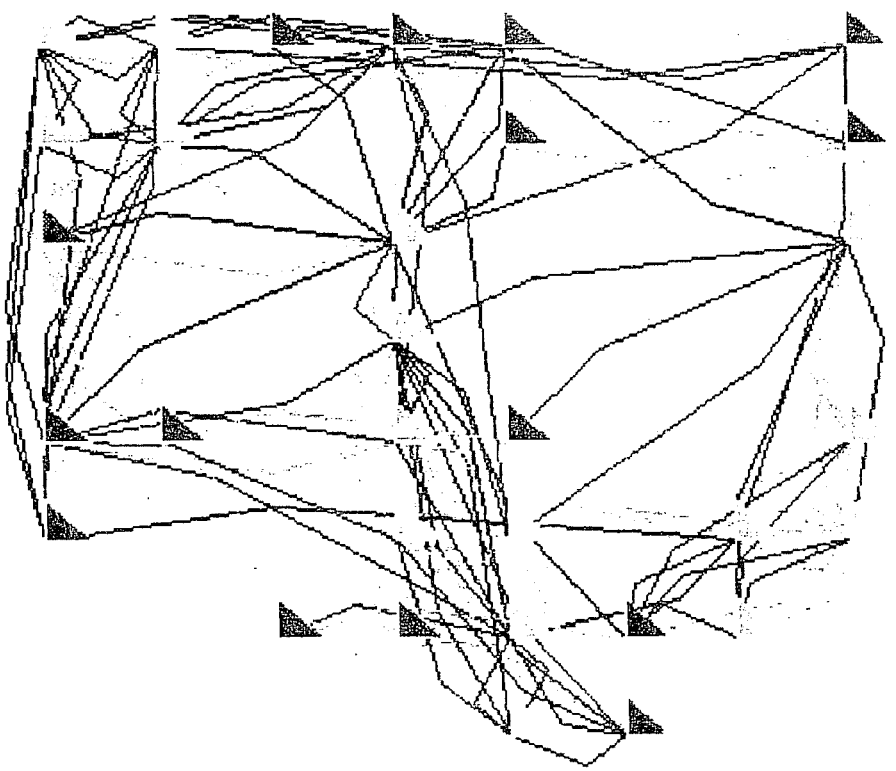


Figure 5: The final frame of movie 12.