Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

1990

Statistical mechanics of neural networks and combinatorial optimization problems

David L. Morabito

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Morabito, David L., "Statistical mechanics of neural networks and combinatorial optimization problems" (1990). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Rochester Institute of Technology
School of Computer Science and Information Technology

Statistical Mechanics Of Neural Networks And Combinatorial Optimization Problems

by

David L. Morabito

A thesis, submitted to
The Faculty of the School of Computer Science and Information Technology
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

1991

Approved by:

Professor Peter Anderson

Professor Stanislaw Radziszowski

Professor Jerome Wagner

PERMISSION TO REPRODUCE

Title of Thesis:

Statistical Mechanics of Neural Networks and Combinatorial Optimization Problems.

I _____ prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

David L. Morabito
9805 Union St.
Scottsville, N.Y. 14546

Date: 7/30/91

ACKNOWLEDGEMENTS

I want to thank my parents for all their love and support throughout my many years in school.

ABSTRACT

Local learning neural networks have long been limited by their inability to store correlated patterns. A common parameter used to specify the capacity of a network is $\alpha = p/N$, where p is the number of stored patterns and N is the number of neurons in the network. Using statistical mechanics I have found that a network using nonlocal learning rules has better retrieval qualities than a network using local learning rules. In particular, the analysis has determined that nonlocal learning rules give a maximum capacity of $\alpha_c = 1$, which is a significant increase over local learning rules in which the maximum capacity is never greater than $\alpha_c = 0.14$. Computer simulations for a nonlocal learning network are also presented.

A well known NP-complete problem is graph q-partitioning. The goal of q-partitioning is to partition the vertices into q equal parts, such that the number of intrapartition edges is minimized. A technique called the *replica method* has been previously developed to handle strongly frustrated physical systems. In this thesis it was applied to handle the graph q-partitioning problem. Final analytical results for the q-partitioning problem compare very favorably to known computer simulations.

KEY WORDS AND PHRASES

Associative recall - Different input stimulus will lead to a similar output stimuli.

Asynchronous dynamics Neurons are picked in a random sequence for updating. Updating of each neuron is dependent on the previous network state.

Attractor - A state, or a restricted set of states, which is dynamically reached by a large collection of input states after a long enough time.

Attractor Neural Network (ANN) A non-linear network whose long time behavior is governed by attractors.

Basin of attraction - Set of states which are attracted to the same attractor.

Ergodicity - A system will sample all of its possible states in a finite amount of time.

Excitatory - A synaptic connection that increases the likelihood that a neuron will fire.

Fixed-point - A network state that is continually repeated in the course of the dynamical evolution of the network.

Free-energy - A quantity that is decreased as a system follows some dynamical process.

Frustration Contradictory requests are made on a neuron. This is brought about by having both inhibitory and excitatory connections to the same neuron.

Inhibitory - A synaptic connection that reduces the likelihood that a neuron will fire.

Network state Instantaneous state of all neurons that make up the network.

Overlap - Measures the nearness of two N bit words.

Retrieval - Specification that a network has reached an attractor state. Will be signified by a special bursting of neurons.

Spurious state - Pattern that has not been stored by a network.

Synchronous dynamics - Updating of all neurons is based on the previous network state.

COMPUTING REVIEW SUBJECT CODES

G.2.1 Combinatorics

G.2.2 Graph Theory

TABLE OF CONTENTS

FIGURES

CHAPTER 1

INTRODUCTION

1.1. Motivation and Goals

1.1.1. Motivation

Biological neurons, though numerous in size, structure type and functionality, all "fire" a signal (voltage potential) down their axon if the sum of the inputs at their soma reaches the neurons "firing" potential. Since magnetic systems in nature exhibit many of the features of neural systems, the tools and concepts used in studying magnetic systems should also be applicable to the study of neural networks. To a neurobiologist this may seem to be a useless endeavor. In Physics the major tool for studying many-body problems (including an infinite number of bodies) is statistical mechanics. If we were actually trying to model the dynamics of some system it would be necessary to consider the actual structure of the system. On the other hand, if we were concerned with the long time behavior of a system it is not necessary to consider the structure. Given the rules for the interaction of one piece of the system with another piece of the system we can effectively compute the long time behavior of the system. In other words the actual structure of the system, be it a biological neuron or a magnetic spin system, is hidden inside the interaction rule.

1.1.2. Goals

Hopfield, using this approach, analyzed a local learning neural network [Hopfield82]. The particular magnetic system used was an Ising spin glass [Pathria80, Huang62, Kubo65, Feynman62]. The Ising model for a spin glass is a very naive model. The first topic of this thesis is to model a neural network that uses nonlocal learning instead of local learning. The second topic of this thesis is to use statistical mechanics to solve the graph q-partitioning problem analytically.

1.2. Neurophysiological Background

The subject of neural networks can be approached from two different angles: the first is concerned with massive parallel computers. Usually this area is not concerned with questions of biological plausibility. Mainly, questions of computational speed and efficiency are addressed. The second category is a study of model networks that exhibit many of the characteristics of a biological neural system. There are six criteria that will be considered important in any theory of a neural network [Amit89]:

- Biological plausibility – simply the requirement that the elements composing the network not be outlandish from a physiological point of view.
- Associativity – impression that many similar inputs are basically collapsed on a prototype for purposes of cognition and manipulation – a picture viewed from any angle and in different light and shading represents a single individual.
- Parallel processing – the articulation of the tension between the slow basic cycle-time of neuronal processes and the impressive speed with which the system as a

whole reacts to tasks that are prohibitive to high-speed serial computers.

- Emergent behavior – stands for an input-output relation which is rather unlikely (non-generic) given the system's elements.
- Freedom from homunculi – eternal goal of eliminating little external observers which could assign meaning to outputs.
- Potential for abstraction – the ability of a network to operate similarly on a variety of inputs that are not simply associated in form but are classified together only for the purpose of the particular operation.

It is important that we know the stages that an individual neuron goes through to determine if it will fire or not:

(1) A neuron either is in a state of dormancy or in a firing state. In the firing state an action potential is effectively propagated unchanged down the axon.

(2) When the action potential reaches the end of the axon, neuro-transmitters are emitted in the post-synaptic cleft.

(3) When the neuro-transmitters reach the membrane of the post-synaptic neuron, ionic current is allowed to enter the post-synaptic neuron. The amount of current is a measure of the efficacy of the synapse.

(4) The ionic current generates a Post Synaptic Potential (PSP). (Note: Throughout the rest of this thesis the PSP at a particular neuron, labelled by i, will be represented by h_i). This potential is gradually reduced as the PSP travels toward the soma. A PSP can be either excitatory (i.e. increasing the probability of the neuron firing) or inhibitory (i.e. decreasing the probability of the neuron firing).

(5) A neuron will fire (i.e. transmit an action potential – indicated below by $\sigma = 1$) if the sum, calculated at the soma, of all incoming PSP's is greater than the neuron's threshold potential. If the sum of PSP's is less than the threshold, the neuron is dormant (i.e. it does not transmit an action potential – indicated below by $\sigma = 0$).

1.3. Formal Neurons and Perceptrons

We can formally specify the condition for the neuron to fire in the manner prescribed in the Section 1.2:

$$h_i = \sum_{j=1}^{N} J_{ij}\, \sigma_j \tag{1.1}$$

where the J_{ij}'s are called the synaptic efficacies. The output of neuron i depends on the value of h_i in comparison with the threshold potential at neuron i (T_i). Symbolically, we can represent the spiking condition as:

$$\sigma_i' = \begin{cases} 1 & \text{if } h_i > T_i \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

Thus $\sigma_i' = 1$ indicates that the neuron will spike (fire an action potential) while $\sigma_i' = 0$ indicates that the neuron is dormant. Using that fact that 0 and 1 can be used to represent the values true and false, McCulloch and Pitts [Pitts43] proposed that a collection of neurons could be connected into a temporal sequence at the end of which there would be a single output neuron whose output would be the value of the logical expression imprinted on the network at the beginning of the temporal sequence. Refer to Figure 1.1 for the construction of the exclusive or of A and B by a collection of formal neurons.

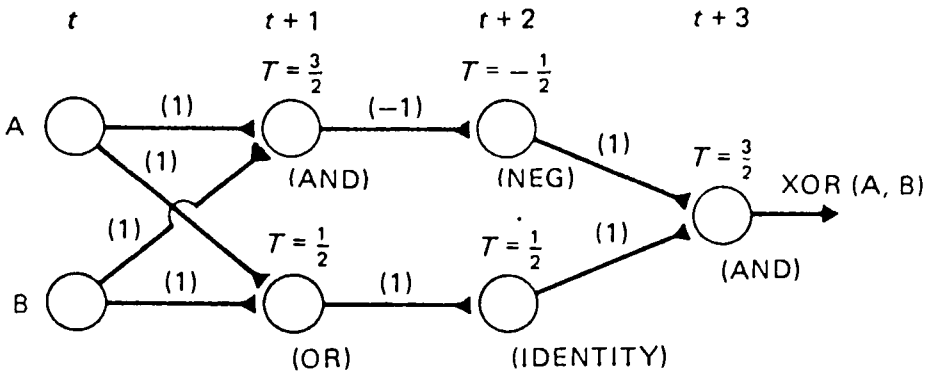


Figure 1.1: The construction for the exclusive or (xor by the route $[A+B]\cdot[\neg[A\cdot B]]$ [Amit89].

Realizing that any true neuron will have approximately 10^4 input synaptic connections, Rosenblatt [Rosenblatt62] proposed that a formal neuron can have any number of inputs. There is still only one output neuron but we now have any number of inputs into that output neuron. A perceptron is a linear threshold device (i.e. no feedback of any type is present in the network) just as a formal neuron is. The best that can be accomplished is a two-fold classification of input values. A network that has N input values can accommodate 2^N different input patterns (i.e. patterns of 1's and 0's). Since the output neuron can generate either a 1 or a 0 we can classify the input into two classes. One class represents all input that generates a 1 output while the other represents all input that generates a 0 output.

Nothing restricts us from forming a network of more than one perceptron. In this scheme we can generate a classification scheme that could be used in character recognition (see Figure 1.2). With N inputs and now Q output neurons we can classify all

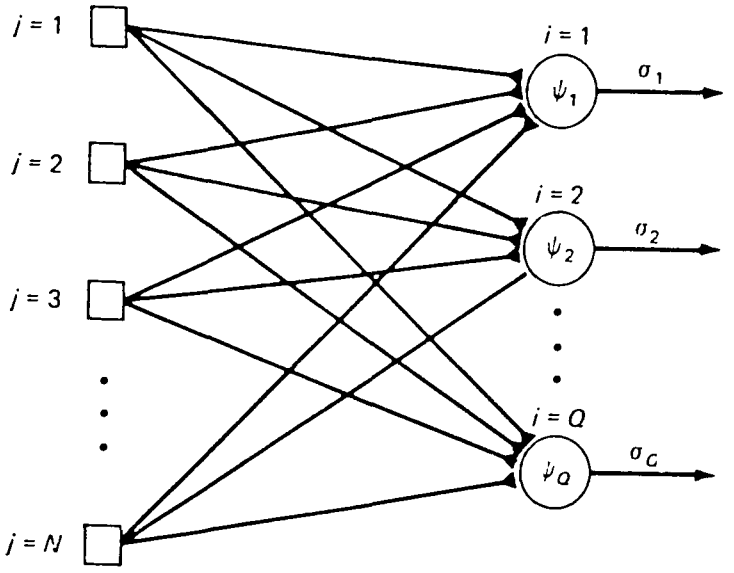


Figure 1.2: A single layer multi-perceptron [Amit89].

the input patterns into classes depending on the output pattern that they produce. The number of classes and the actual pattern each class corresponds to is determined by the value for the synaptic efficacies.

1.4. Attractor Neural Networks (ANN)

If we close a multi-perceptron upon itself we have a network where the output axons are identified also as input axons – this is a closed system that is not an input-output system (see Figure 1.3). In such a system once a network state is repeated once, it will be repeated indefinitely. These states are called the "attractors" of the dynamics. The idea behind the concept of an attractor is that when a network is prepared in a state close to an attractor state it will, through the dynamics of the network, progress to the attractor state (i.e. the initial state is attracted to the attractor state). The following notation will be used. T_i is the threshold of the i^{th} neuron and $\sigma_i(t+1)$ is the state of the neuron at time $t + 1$. We can give an explicit representa-

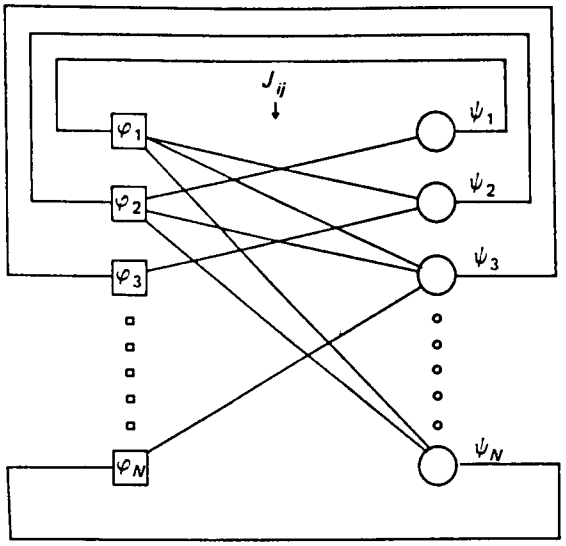


Figure 1.3: A multi-perceptron closed on itself to form an ANN [Amit89].

tion for σ_i as:

$$\sigma_i(t+1) = \begin{cases} 1 & \text{if } h_i(t+1) - T_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

In other words this means that if σ_i is 1 the neuron is firing, while if σ_i is 0 the neuron is dormant. For a graphical example of a 5 neuron network see Figure 1-4.

1.4.1. ANN Assumptions

ANN's have the following assumptions that have immediate neurobiological meaning.

- The individual neurons have no memory.

  This is manifest in the fact that no thresholds vary in time (i.e. no modification due to past happenings).

- A single neuron may have any number of excitatory and inhibitory synapses emanating from its axon.

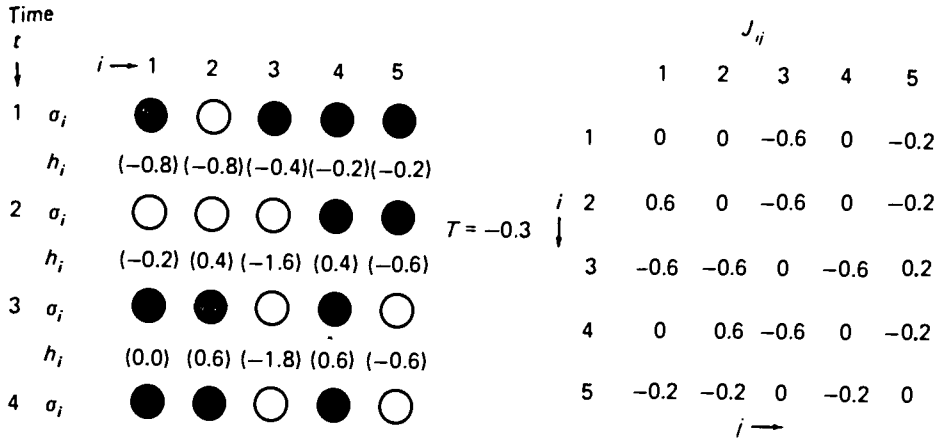


Figure 1.4: Dynamics of a five neuron network with thresholds $T_i = -0.3$ and a connection matrix J_{ij}. White neurons are resting, black ones have spiking axons. Note that following $t = 3$ all states will be identical. This is an example of a fixed-point attractor [Amit89].

In physics when there are competing tendencies throughout the system we say that the system is frustrated. Frustration is the underlying reason why a network will have many energy minima each of which corresponds to a stored pattern. The whole idea behind frustration will be treated in much more depth at a later time.

- The synaptic connections are all symmetric.

  This is formally written as $J_{ij} = J_{ji}$. This is an indispensable tool in the study of ANN's but it has no biological foundation. This would be worrisome to biologists, but it has been formally shown [Amit89] that when the symmetry restraint is lifted no major modifications are seen in the behavior of the network.

- The dynamics of the network is asynchronous.

This is a reasonable assumption to make since in a true neural system there does not seem to be any notion of a master clock indicating when each neuron in the network should update. Each neuron chooses to fire or not to fire within a preset time frame (after this time the PSP at the soma gradually is degraded). Also, when physical separations are included in the analysis it is apparent that updates of nearby neurons will only effect the neuron that is choosing to fire if those nearby neurons are within a certain length.

1.5. Significant Network States

A network state is defined to be the collection of the individual states (i.e. is the neuron firing or dormant) of each neuron. Thus a network state is made up of N numbers where each number can be 0 (dormant) or 1 (firing). In an ANN not all network states are significant. Shortly, we will see that only fixed-point sequences will have any dynamical significance. Before discussing fixed-points we must develop some terminology. Consider the fact that the state of a network of N neurons is specified by giving N values each of which can be only 0 or 1. Geometrically, these values can be viewed as labelling the vertices of an N-dimensional hypercube (see Figure 1.5). In this light a fixed-point can be defined as the sequence in which all network states remain at the same vertex in the hypercube (Note: The collection of vertices of the hypercube is sometimes called the space of network states.). All of this is important but we are striving for a way to represent the following three concepts:

- emergence

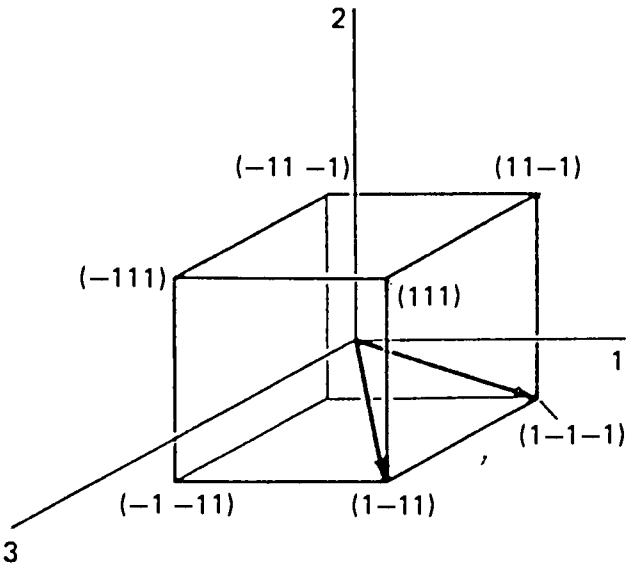


Figure 1.5: Geometrical representation of the distance between pairs of network states [Amit89].

- generation of meaning
- self-recognizability, i.e., freedom from homunculus

Even though the network state is unchanged, it does not mean that the neurons are dormant. On the contrary, when a network is in a fixed point, the average firing rate for a neuron is at its maximum. This fact alone allows for self-recognizability through the calculation of the mean neural firing rate. Thus, fixed points will be identified with cognitive events.

1.6. Spontaneous Computation vs. Cognitive Processing

At this stage of work in neural networks there exists two general classes of networks both of which are used when studying abstract functioning of the brain (i.e. pattern recognition, classification, shape and motion perception). These two classes are feed-forward networks and ANN's. Some of the workers using feed-forward networks are Widrow (multi-perceptron adaptive classifier – [Widrow60]), Willshaw (correlation

model – [Willshaw69]), and Kohonen and Palm (error reducing projections – [Kohonen84]). Feed-forward networks are further categorized as input systems since for each network input state there is an output state that can only be categorized as good or bad after looking at its content. In these cases we say that spontaneous computation has taken place (see Figure 1.6). Hence, in such networks the notion of a fixed point is meaningless. In actuality, all network states would correspond to a cognitive event. Thus, in many feed-forward networks the notion of an internal homunculus cannot be discarded.

In contrast, Little [Little74] and especially Hopfield [Hopfield82] have introduced the notion of Attractor Neural Networks. The major difference between feed-forward and dynamical attractor networks is that in an ANN there is not necessarily a meaningful output state for each input state. As discussed earlier the output states are meaningful when the network is in an attractor state of which a fixed point is only the simplest version.

The major study concerning ANN's deals with their use as associative memories. Acting as an associative network the attractor states will correspond with the stored memories "learned" by the network. Using the same idea, we can turn an ANN into a computational network. What is required for an ANN to be used for a computation is that the result of the computation corresponds to an attractor of the network. Thus, the notion of temporary results is not necessary or tenable. The following common problems can be "solved" by the use of ANN's:

(1) Traveling salesman

(2) Matching problem

(3) Graph partitioning problem

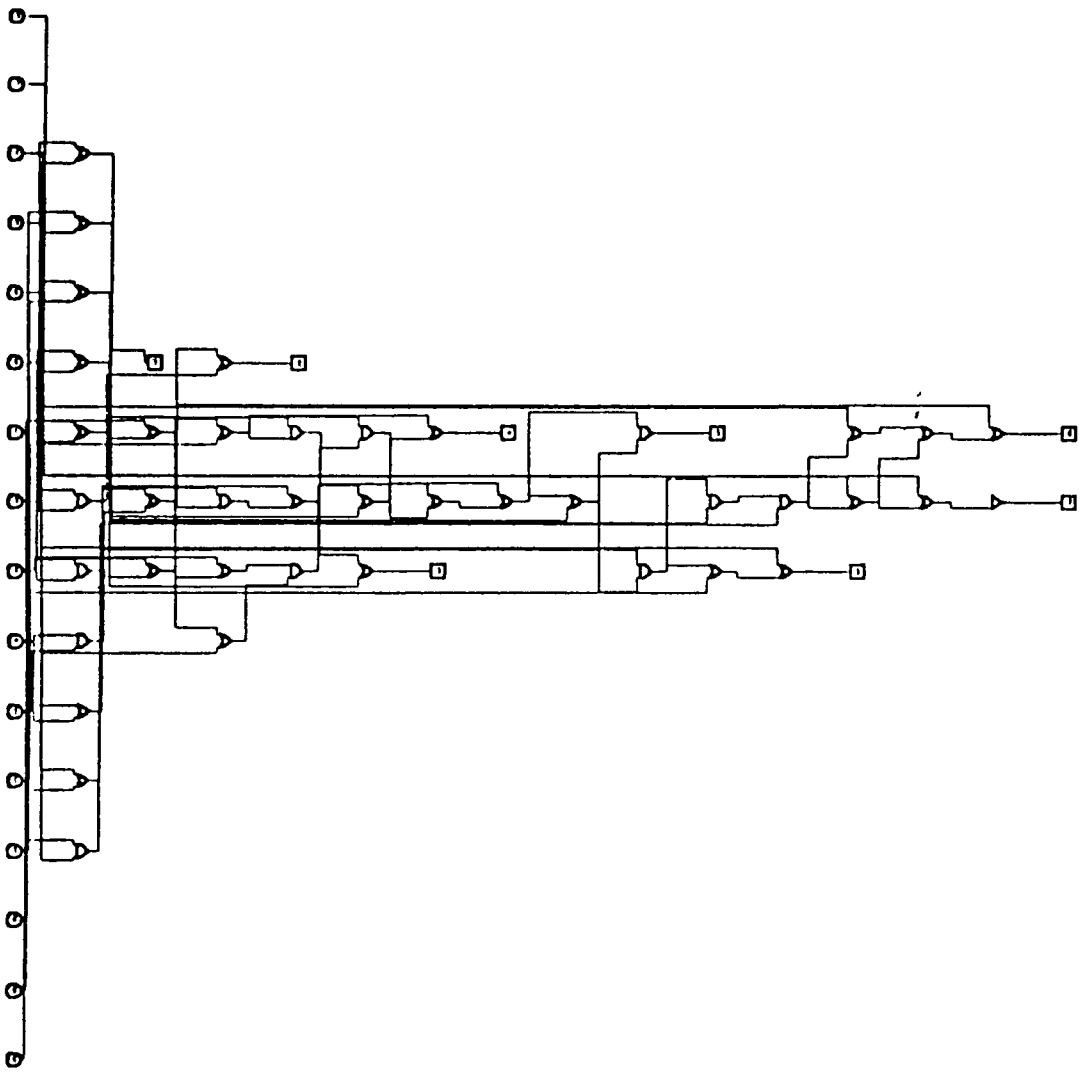


Figure 1.6: A network which spontaneously computes the sum of two 8-bit words, the top and the bottom input units (open circles on left) and represents the sum on the 8 output units (full squares on right). The internal elements are logical gates with two inputs and several outputs. The full specification of the network includes a description of the connectivity as well as the choice of logical function at each gate [Amit89].

CHAPTER 2

SPIN GLASSES

2.1. Introduction

Before discussing spin glasses it may be instructive to discuss some of the basic similarities between random magnetic systems and neural networks [Gutfreund85]. Any magnetic system can be naively viewed as a collection of magnetic spins S_i where $S_i = \pm 1$. The effect of all the spins in the system (except the i^{th}) contributes to the creation of a local field at site i. It is this local field that determines the value of S_i (assuming that no external magnetic field is present). The value of the local field can be calculated from a sum of exchange interactions acting on each spin in the system. Specifically,

$$h_i = \sum_{j=1}^{N} J_{ij} S_j \tag{2.1}$$

The J_{ij} are called the exchange interaction. Throughout this chapter it will be assumed that $J_{ij} = J_{ji}$ and $J_{ii} = 0$. The Hamiltonian (i.e. the energy of the system) is given by the expression:

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} J_{ij} S_i S_j \tag{2.2}$$

Notice that in equation (2.2) if the values for all the spins are simultaneously changed in sign, the Hamiltonian is unchanged (we say that the Hamiltonian is symmetric to global spin-flips).

2.1.1. Simple Model

In the simplest case assume that $J_{ij} \equiv J$ and J > 0. What are the phases of this system? First, a few technical points must be clarified. Glauber dynamics gives the following probability for spin i to take on the value S_i at "temperature" T:

$$Pr(S_i) = \frac{exp(h_i S_i/T)}{exp(h_i/T) + exp(-h_i/T)} \tag{2.3}$$

When N (= number of spins in the system) is large and the J_{ij} are of order 1, the local fields become proportional to N (refer to equation (2.1)). In equation (2.3) it is apparent that the dynamics of the system is governed by h_i/T. Use equation (2.3) to calculate the probability of a spin being flipped when the system is in a ground state configuration. (The following section will show that there are two ground states in each of which the spins are completely aligned when the temperature is low.) We expect in the noiseless limit (i.e. when the temperature is 0) that this probability will be identically zero. In equation (2.3) assume that the ground state configuration is when all the spins are +1 and we are looking for the probability of flipping one of the spins to −1. Equation (2.3) becomes:

$$Pr(-1) = \frac{exp(-JN/T)}{exp(JN/T) + exp(-JN/T)} \tag{2.4}$$

Assuming that T is independent of N, then we calculate from equation (2.3) that in the thermodynamic limit ($N \rightarrow \infty$):

$$Pr(S_i) = 0 \tag{2.5}$$

Thus the system will always appear noiseless. To remove the dependency of the local fields on N, all that is necessary to be done is to replace the exchange interaction with J/N. Now, equation (2.4) is independent of N and thus non-zero for non-zero T.

With this in mind, let us rewrite equation (2.1) as:

$$h_i = \frac{J}{N}\sum_{j=1}^{N} S_j \rightarrow \frac{J}{N} M\{S\} \equiv Jm\{S\} \tag{2.6}$$

Where the curly brackets indicate that the quantity is a function of all N spins. Also, M is the total magnetization of the system and m is the magnetization per spin of the system.

2.1.1.1. Ground States

Let us consider the situation when the temperature is zero. At this stage it must be understood that only at $T = 0$ will the minima of E be the true minima of the system. At temperatures greater than zero the minima of the free energy will be the true minima of the system. Without going into details at this point, the following results will be true also at temperatures that are small compared to J. With the negative sign in equation (2.2) and $J > 0$ notice that there will be two ground states (i.e. the states with lowest energy): all the spins +1 and all the spins −1. This is called ferromagnetic ordering. As is easily seen from equation (2.2), whenever two spins are parallel (i.e. $S_i = S_j$) the contribution to the sum is negative. In conjunction with the negative sign for E, this lowers the energy by $-J/(2N)$. Thus, when all pairs are parallel, we ascertain a ground state energy of $-J/2$. In general, the situation is the following: at high temperatures (relative to J) the entropy (measure of the order in the system − lower entropy → more order in the system) dominates over the energy while at low temperatures (again relative to J) the exact opposite occurs. In physical terms the high temperature phase is reflected in the fact that each spin in the system is, on average, up (+1) as often as it is down (−1).

2.1.1.2. Neuronal Language

Attractors of a neural network were discussed in Chapter 1 Section 1.4. In the language of neural networks we have for this simple model two attractors: the attractor with $S_i = +1$ for all i and the attractor with $S_i = -1$ for all i. Let us define an overlap function M as:

$$M = \sum_{i=1}^{N} S_i \, S_i' \tag{2.7}$$

This represents the "nearness" of two N bit numbers. We see immediately that the M, which was defined in the last section, is the overlap of the state {S} with the attractor with all spins pointing up (i.e. $S_i' = +1$).

How can we directly connect this to a neural network? When we associate $S_i = +1$ with the firing state and $S_i = -1$ with the dormant state we can use equation (2.6) to calculate the PSP at any neuron. Also, choosing the sign of J_{ij} as negative we get the inhibitory or competitive effect so important in the working of any plausible neural network.

2.2. Random Ising Magnetic System

The Ising model that Hopfield used is a simplistic model of magnetism but nonetheless, the model has led to tremendous insights into the study of large numbers of strongly interacting systems. Some insights are symmetry breaking, cooperative phenomena, order parameters, disorder parameters, critical exponents, and symmetry restoration [Amit89]. Of all magnetic systems, we will be most concerned with spin glasses. These are magnetic systems of a special nature. Because of conflicting influences between spins, a spin glass does not exhibit long-range ferromagnetic or

antiferromagnetic ordering. It is believed that a new type of ordering prevails in which the spins of the system are aligned randomly [Binder86].

What makes spin glasses so interesting for the study of neural networks? There are two main reasons for this:

(1) Multitude of energy minima – necessary for the operation of an associative memory.

(2) Conflicting neural activity – necessary for modeling mixtures of excitatory and inhibitory synapses.

This multitude of energy minima exists because spin glasses in many situations are frustrated systems (see Figure 2.1). Frustrated systems arise when one spin, because of its interactions with other spins, is required to be in two different states. In Figure

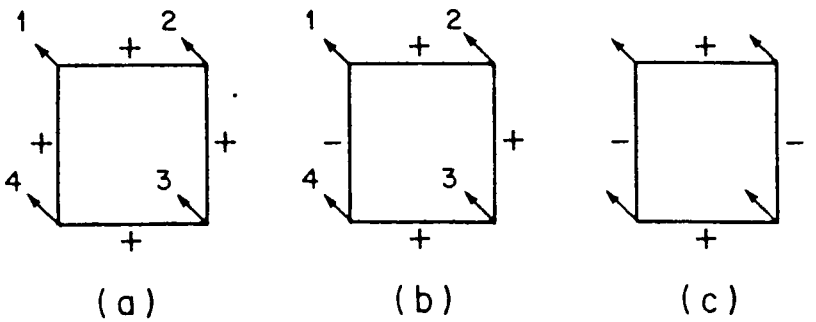


Figure 2.1: Three sets of four spins arranged on squares. The numbering is only for labelling purposes. (a) a ferromagnetic square; (b) a frustrated square - three aligning and one anti-aligning interactions; (c) a non-frustrated square - two aligning and two anti-aligning interactions [Amit89].

2.1 a "+" labelling an edge (bond) represents the fact that the interaction of the bond tries to align the spins while a "–" sign means that the bond tries to anti-align the spins. In Figure 2.2 you will find the calculation for the energy of each of the possible states of the four spin system. Notice how there is only one ground state for the non-frustrated systems (a and c) while there are 2 ground states for the frustrated system (b). (Note: We do not consider the state that has all of its spin values reversed in the count of ground states - they are ground states because of the global spin-flip symmetry of the energy function.) Thus, we see with a very simple system how frustration generates multiple ground states. Notice also that the ground state of the frustrated system is larger than the ground state of a corresponding non-frustrated system. One last comment should put this all into perspective. As is well known from biology, neurons can have excitatory and/or inhibitory connections. By allowing the J_{ij}'s to be

	(a)	(b)	(c)
$+1+1+1+1$	-4	-2	0
$+1+1+1-1$	0	-2	0
$+1+1-1+1$	0	0	0
$+1-1+1+1$	0	0	0
$-1+1+1+1$	0	-2	0
$+1+1-1-1$	0	-2	$+4$
$+1-1+1-1$	4	$+2$	0
$+1-1-1+1$	0	$+2$	-4

Figure 2.2: Eight of the sixteen states for the three networks in Figure 2.1. The first column lists the states with the order of the spins corresponding to the numbering on the squares from left to right. The columns labelled (a),(b) and (c) correspond to the different parts of Figure 2.1 [Amit89].

both positive and negative we can create a frustrated system. Immediately it is seen that the inhibitory and excitatory connections will be modeled by having positive and negative synaptic connections in our neural networks. This will in turn lead to a frustrated neural system.

2.3. Previous Work and Problem Specification

Now we are ready to restate our ideas in the language of neural networks. The simplest way to do this is to describe briefly some of the past work done. Hopfield's model [Hopfield84] was the first to extract the analogy between magnetic systems (spin glasses in this case) and neural networks. His approach was to form a differential equation that, when solved, gives the state of each neuron at any given time. Hopfield analyzed a model network using zero temperature (i.e. noiseless) Glauber dynamics. Little [Little74], 10 years before, analyzed a model using finite temperature Glauber dynamics. In each of these models we define the Hamiltonian (i.e. the energy function) as:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij}\, S_i\, S_j \tag{2.8}$$

As before $J_{ii} = 0$ and $J_{ij} = J_{ji}$. This is the Hamiltonian for an infinite range spin glass where each spin interacts through the random exchange J_{ij}. Both models when analyzed used local learning [Gutfreund85]. Local learning rules specify that the exchange interactions J_{ij} (i.e. the synapses) are modified only through the activity of neuron i and neuron j but not on any extended group of neurons.

Recently, it has been discovered that nonlocal learning rules lead to error free associative recall [Kanter87]. Unfortunately, when dealing with spin glasses,

traditional equilibrium statistical mechanics (i.e. the description of systems at or very near thermodynamic equilibrium) is not adequate. Normal systems at a finite temperature (in neuronal language, a system with noise) are ergodic. Ergodic behavior condemns the system to sample ALL of its possible states in a finite amount of time irrespective of its initial state. Hence, with ergodic behavior the operation of an associative memory fails. There are two possible ways out of this dilemma: the first is that a system exhibits non-ergodic behavior for a time long enough for associative recall to take place. Secondly, we can have cooperative behavior that will restrict the possible network states to a limited set of states indefinitely. In the case of spin glasses we see both of the above situations. We shall call states that relax on time scales beyond plausible observation times, metastable (i.e. they will relax to the minimum value of the free-energy but only after a long time).

2.4. Theoretical and Conceptual Development

The technique that will be used throughout this thesis is called the "replica method". This is a very sophisticated technique that will require much elaboration before it can be properly justified, used and analyzed. Let us begin our discussion by studying a system that is described by a set of statistical variables S_i. Also, assuming the system is random, we will need to specify a set of variables $\{x\}$ that describe the randomness of the system. If the fluctuation time for the random variables is less than the observation time, then the system progresses to a state of thermal equilibrium. In this case the usual techniques in equilibrium statistical mechanics can be used. Effectively, this means that normal averaging techniques can be carried out over the

random variables of the system. For example, for the free-energy and the partition function (i.e. the function from which we can derive ALL thermodynamic properties) we have:

$$F = -T \ ln[Z\{x\}]_{av} \tag{2.9}$$

$$Z[\{x\}] = Trace_S \ exp[-E\{x, S_i\}/T] \tag{2.10}$$

In the above formulas, T stands for the temperature while the Trace (Tr in future equations) is over all possible spin configurations. (Note: Later we will see that the free-energy is our "energy-function" in the noisy case.) The above average is called an annealed average. To reiterate, an annealed average is only possible to perform if the random variables participate in the dynamics of the system in the same manner as the spins. As was discussed earlier, spin glasses relaxation times are much greater than the observation times. This makes the above averaging technique inappropriate. The proper averaging technique is called quenched averaging. In this scheme, the random variables each take on a unique value as the statistical variables (i.e. the spins) fluctuate. The question arises as to what should be averaged. In thermodynamics we have two types of variables: intensive and extensive. An intensive variable is independent of the size of the system while extensive variables are dependent on the "size" of the system. By considering the following system [Brout59] we shall see that the size of the system plays a dominant role in the averaging process. Now consider that a large system is broken up into a collection of macroscopic systems each of which has its own set of random variables. Also, assume that the interaction between each subsystem is negligible. We can then calculate the (normalized) average of any extensive variable for the whole system by averaging over each of the subsystems. As the

number of subsystems grows, the above average will approach the average of an extensive variable over all choices of {x}. Let us use the magnetization per spin M as an example:

$$M\{x\} - [M]_{av} \rightarrow 0 \qquad (\text{for } N \rightarrow \infty) \tag{2.11}$$

Any system that satisfies the above equation for any set of random variables is called self-averaging.

I state without proof that if we calculate the free-energy density f (i.e. the free energy per spin) in this case we get a Gaussian distribution with width $N^{1/2}$ [Binder86]:

$$P(f) \propto \exp\{-N(f - [f]_{av})^2/(2(\Delta f)^2)\} \tag{2.12}$$

If the annealed average of f is calculated then we get:

$$F_{ann} = -\frac{T}{N} \ln[Z]_{av} = [f]_{av} + (\Delta f)^2/T \tag{2.13}$$

It would be expected that $\ln[Z]_{av}$ would give $[f]_{av}$. But in this case we get an additional term. How can all this be viewed physically? We know that averaging over the partition function will not work. For statistical mechanics to be of any use, we need a free-energy function that will give us the ability to calculate the thermodynamic properties of the system. But for this to be true, no large fluctuations of the free-energy function can take place. So the proper quantity to be averaged is the free-energy function, NOT the partition function. On a technical note, this is where the problems begin. Averaging the partition function is a reasonable job but when dealing with the logarithm of the partition function (i.e. the free energy) the job becomes much more difficult. A way around this is to use what is called the replica method. More on this later.

Let us return to the concept of ergodicity. We saw earlier that if ergodicity is present an associative memory cannot function. As is well known in Physics, any system, when noise ($T > 0$) is present, will be ergodic. We discussed the possible solutions to this problem earlier. In spin glasses many states can be found that have the minimum value of the free-energy (these will be our retrieval states for a neural network). Most systems that exhibit broken ergodicity have low-lying states that are related by some symmetry of the Hamiltonian (i.e. the energy function). This is not the case with spin-glasses. They exhibit what is called accidental degeneracy which is manifested from randomness and frustration. Before discussing the "replica method", there is one last important concept to discuss. Some means of labelling the multitude of thermodynamic ground states must be found. In Physics the most common way of labelling these states is by giving the average magnetization. In neuronal language this would be the overlap between the network state and a stored pattern.

The replica method is based on the formula [Binder86, Amit89]:

$$\lim_{n \to 0} \frac{Z^n - 1}{n} = lnZ \tag{2.14}$$

Immediately, one can see that the difficulty in averaging lnZ is replaced by averaging Z^n. Technical difficulties arise when trying to average LHS (Left Hand Side) when $n = 0$. To reach the limit, an analytic continuation must be performed on the LHS which is a function of n. A well known fact from statistical mechanics is that a collection of n identical independent systems can be represented by a partition function which is Z^n (Z is then the partition function of a single system). These n systems are the replicas. Let us explicitly write the general partition function of a system of spins.

Assume that the energy of the system is E({x},{S}), where {S} represents the spins of the system and {x} represents a particular set of random variables.

$$Z^n(\{x\}) = (Tr_S \; exp\,[-\beta E(\{x\},\{S\})])^n$$

$$= Tr_{S^1} \; exp\,[-\beta E(\{x\},\{S^1\})]) \; \cdots \; Tr_{S^n} \; exp\,[-\beta E(\{x\},\{S^n\})]$$

$$= Tr_{S^1,\ldots,S^n} \; exp\,[-\beta \xi(\{x\},\{S^1, \ldots, S^n\})], \tag{2.15}$$

where

$$\xi(\{x\},\{S^1,\ldots,S^n\}) = \sum_{\alpha=1}^{n} E(\{x\},\{S^\alpha\}) \tag{2.16}$$

This is the energy of a system of n • N spins (i.e. each replica is a collection of N spins). Once an explicit representation is known for the partition function then the averaging over the random variables is carried out. At this time two paths could be taken. The first is to continue with a general discussion of the replica method; the other to relegate any further discussion of the method until the actual calculations. The latter approach will be adopted here.

One of the goals of this thesis is to understand the workings of a neural network that uses nonlocal learning. Thus, a nonlocal learning rule will be introduced and then analyzed to determine its behavior as an associative memory. Another goal of this thesis is to give a statistical mechanical treatment to a common combinatorial optimization problem. Here graph q-partitioning will be analyzed using the full power of statistical mechanics and the replica method. Lastly, extensions of the use of statistical mechanics to Ramsey numbers, graph coloring, and other optimization problems will be addressed.

Earlier, it was stated that we can use an overlap parameter to distinguish retrieval states (i.e. states with minimal free-energy). In general situations, more than one parameter (generally called order-parameters) will be needed to specify retrieval states. By generating the free-energy function, we shall be able to determine the set of order-parameters (actually we will determine the equations for the order-parameters). These equations are the equations for the extrema of the free-energy.

During the analysis we will have to be conscious of the presence of not only fast noise (i.e. noise generated by non-zero temperature) but also of slow noise (i.e. noise generated by small random overlaps of the pattern being retrieved, with all other patterns stored in the network). The replica method can be used to determine the size of the basin of attraction around each pattern stored in the network. These sizes are used to determine which network states, in the presence of noise, will be attracted to the stored patterns – associative recall.

A quantity that will be important throughout is $\alpha = p/N$, where p is the number of stored patterns in the network and N is the number of neurons in the network. It will be seen that there is a value α_c above which no retrieval takes place (i.e. system is ergodic). Below this value, spurious states, as well as stored patterns, will arise. But, as more noise is introduced, these spurious states will be destabilized and thus the system will have relatively good retrieval properties. So, as should be apparent, a study of a network must be undertaken in the realm of no noise ($T \to 0$ or $\beta = 1/T \to \infty$) and when ($T \neq 0$ or $\beta \neq 0$).

CHAPTER 3

ANN AND NONLOCAL LEARNING

3.1. Learning Rules

As discussed earlier, nonlocal learning rules deal with the activity of a collection of neurons not just two neurons. There seem to be two major arguments against using nonlocal learning rules: (1) no strong evidence has been found indicating that such nonlocal effects are seen in biological systems and (2) their introduction complicates any analytical study of an ANN. When dealing with artificial networks, any concern about biological plausibility can be disregarded. Even though no evidence for nonlocal learning has at present surfaced, it seems reasonable that there could be situations dealing with more abstract operations of a biological neural system which could involve some type of nonlocal learning. All local learning networks have difficulty learning correlated patterns. The overlaps between patterns can be specified by use of the matrix:

$$C_{\mu\nu} \equiv N^{-1} \sum_{i=1}^{N} \xi_i^{\mu} \, \xi_i^{\nu} \tag{3.1}$$

where μ and ν each range from 1 to p. The $C_{\mu\nu}$ generate an internal static noise which is the main reason that local learning rules encounter difficulty with correlated patterns. These overlaps effectively flip spins away from the original pattern (i.e. original pattern is randomized). Let us look at the case where p is finite while N is very large ($\rightarrow \infty$ in the thermodynamic limit) and the patterns are uncorrelated. The cumu-

lative overlap is (choosing pattern 1 arbitrarily)

$$C_{total} = N^{-1} \sum_{i=1}^{N} \xi_i^1 \sum_{\nu=1}^{p} \xi_i^\nu$$

$$= N^{-1} \sum_{i=1}^{N} \sum_{\nu=1}^{p} \xi_i^1 \xi_i^\nu \tag{3.2}$$

Equation (3.2) is a sum of Np bits of +1 and −1. Since the stored patterns are uncorrelated, this is a one-dimensional random walk of Np steps. It is well known in Physics that the mean square distance of a random walk from a preset origin is XL where X is the number of steps and L is the step length. So, in this case we obtain that the mean square distance of the walk is $O(\sqrt{p/N})$. In the case that $\alpha = p/N$ is finite (i.e. p and N both large) there exists a value α_c above which there is a breakdown of associative recall. In all Hopfield style of networks with local learning rules, α_c is always less than 0.14. This means that in a network of 100 neurons only 14 patterns could be stored and retrieved with any certainty.

3.2. Personnaz Model

Personnaz [Personnaz85] discussed nonlocal learning rules that can be used to suppress the effects of overlaps between correlated patterns. The model he discussed was based on the solution of the stability equation

$$\sum_j J_{ij} \xi_i^\mu = \lambda \xi_i^\mu \tag{3.3}$$

where i ranges from 1 to N and μ ranges from 1 to p. These equations guarantee the stability of an arbitrary set of stored patterns. The solution of Personnaz was the following ($\lambda = 1$)

$$J_{ij} = N^{-1} \sum_{\mu,\nu} \xi_i^\mu \, \xi_j^\nu \, (C^{-1})_{\mu\nu} \tag{3.4}$$

where $C_{\mu\nu}$ is the overlap matrix as defined earlier.

3.3. Noiseless Dynamics

In the noiseless case a particular network state (i.e. the state of N spins) is considered stable if the spins are each parallel to their respective local fields. To be precise, the following relationships must hold:

$$S_i = sgn(h_i) \tag{3.5}$$

$$h_i = \sum_j J_{ij} \, S_j \tag{3.6}$$

Let us now check the stability of the patterns in this model,

$$\begin{aligned} \sum_j J_{ij} \, \xi_j^\mu &= \sum_j N^{-1} \sum_{\alpha,\nu}^{p} \xi_i^\alpha \, \xi_j^\nu \, (C^{-1})_{\alpha\nu} \, \xi_j^\mu \\ &= N^{-1} \sum_{\alpha,\nu}^{p} \xi_i^\mu N C_{\mu\nu} (C^{-1})_{\alpha\nu} \\ &= \sum_\alpha \xi_i^\alpha \, \delta_{\alpha\mu} \\ &= \xi_i^\mu \end{aligned} \tag{3.7}$$

This shows that the stored patterns are eigenvectors of the synaptic matrix with eigenvalue of 1. Any pattern that is orthogonal to these patterns has an eigenvalue of 0. Now, to check the stability of a stored pattern, assume that the network is in the state ξ^μ. Equation (3.5) now becomes,

$$\xi_i^\mu = sgn\left[\sum_j J_{ij} \, \xi_j^\mu\right] \tag{3.8}$$

Thus all stored patterns are stable states of the dynamics. In other words, once the network reaches one of these states, it will never leave the state. This is only true in

the noiseless case.

3.4. Self-Coupling in Personnaz Model

A sufficient condition for the existence of an energy function is that the interactions are symmetric [Amit89]. The necessity of an energy function is easily justified. Given an energy function, the network is guaranteed to asymptotically drift to a fixed point attractor. Inherent though unsaid in the condition is that J_{ii} must be 0. This requirement is found to be true in the noiseless case and the noisy case. The simplest example of this is in the noiseless asynchronous dynamics case. Asynchronous dynamics involves the updating of only one spin per unit time interval. Consider the energy function:

$$E = -\frac{1}{2} \sum_{i,j,i \neq j} J_{ij} \, S_i \, S_j \tag{3.9}$$

where the J_{ij}'s are arbitrary. Let us calculate the change in E when a single spin S_k is changed from S_k to $-S_k$.

$$\Delta E = S_k \sum_{j,j \neq k} J_{jk} S_j + S_k \sum_{j,j \neq k} J_{jk} S_j \tag{3.10}$$

Remember, in Physics the dynamics of a system must be such that the energy of the new state must be lower than the preceding state (at $T \neq 0$ this must be supplemented). From equation (3.10), we see that the change will occur only if the first term on the right is negative (i.e. the local field at site k is anti-parallel to the spin). Since the J_{ij} are completely arbitrary, there is no a priori way of knowing the sign of the second term on the right. Thus a spin flip may not reduce the energy as required. If, however, $J_{kj} = J_{jk}$ E will be reduced on any spin flip and we can properly associate E with the energy of the system. Thus, symmetric synaptic connections is a sufficient

condition for the existence of an energy function. At this point it is easy to see why we require $J_{ii} = 0$. Assuming that $J_{ii} \neq 0$, then we would have a term in the expression for ΔE that contained S_k^2. If this term were present, it would automatically be non-zero since S_k^2 is always 1. Thus the sum would no longer represent the local field at S_k. The importance of all of this is that any learning rule that has non-symmetric synaptic coefficients cannot be described by an energy function and thus the analytical techniques of statistical mechanics cannot be used.

It must be understood that a network with self-coupling is viable and can be studied through both simulations and analytical techniques. If the goal is to discard networks which do have self-coupling, then a dynamic reason must be specified. Now in any neural network we are concerned with the basin of attraction of a pattern. The concept of a basin of attraction is the following:

> A basin of attraction around a stored pattern represents the set of states near a stored pattern that will be attracted to the stored pattern by the dynamics of the network.

Now in the case of the Personnaz model, the following will show that the effect of self-coupling is to reduce the size of the basin of attractions of the stored patterns. Consider the case where the network is in a state that corresponds to a stored pattern $\{\xi_i^\mu\}$ except for one spin S_1. Then equation (3.5) reduces to:

$$
\begin{aligned}
S_1 &= sgn(h_1) \\
&= sgn\left[\sum_{j(\neq 1)} J_{1j}\, \xi_j^\mu + J_{11}\, S_1\right] \\
&= sgn\left[\sum_{j} J_{1j}\, \xi_j^\mu - J_{11}\, \xi_1^\mu + J_{11}\, S_1\right]
\end{aligned}
$$

$$= sgn\left[\xi_1^\mu - J_{11}\,\xi_1^\mu + J_{11}\,S_1\right] \tag{3.11}$$

The rule for constructing synapses allows us to rewrite equation (3.4) in a much more instructive form. First, calculate the mean of J_{ii}.

$$\begin{aligned}
\ll J_{ii} \gg &= N^{-1}\sum_i J_{ii} \\
&= N^{-1}\sum_{\mu,\nu} (C^{-1})_{\mu\nu}\left[\sum_i \xi_i^\mu\,\xi_i^\nu\right]N^{-1} \\
&= N^{-1}\sum_{\mu,\nu} (C^{-1})_{\mu\nu}\,N\,C_{\mu\nu}N^{-1} \\
&= \alpha
\end{aligned} \tag{3.12}$$

At this stage we cannot use this result in equation (3.4) without proving that any variation from the average is small (for large N).

$$\begin{aligned}
\Delta J_{ii} &= \sqrt{|\ll J_{ii}^2 \gg - \ll J_{ii} \gg^2|} \\
&\propto \sqrt{|N^{-3} - N^{-2}|}
\end{aligned} \tag{3.13}$$

Thus the fluctuations around the average value of α are $O(N^{-1})$. So for large N, the fluctuations are small and we can replace J_{ii} with its average value in equation (3.4). Equation (3.4) now becomes:

$$S_1 \propto sgn\left[(1-\alpha)\,\xi_1^\mu + \alpha\,S_1\right] \tag{3.14}$$

Looking at equation (3.14) we immediately see that if $\alpha > 1/2$ the configurations $S_1 = \xi_1^\mu$ and $S_1 = -\xi_1^\mu$ are stable. Thus, the maximum capacity of the Personnaz model to perform as an associative memory is $\alpha_c = 1/2$. An analysis similar to the one just undertaken would show that even below α_c, the basins of attractions are reduced when J_{ii} is non-zero.

3.5. Non-local Learning Model with an Energy Function

By adding the restriction that $j \neq i$ in equation (3.6), our work will be greatly simplified since we can now write an energy function for the network. Our energy function will be:

$$E = -\frac{1}{2}\sum_{i,j} J_{ij}\ S_i\ S_j \tag{3.15}$$

where

$$J_{ij} = N^{-1}\sum_{\mu,\nu} \xi_i^{\mu}\ \xi_j^{\nu}\ (C^{-1})_{\mu\nu}$$

$$C_{\mu\nu} = N^{-1}\sum_{i} \xi_i^{\mu}\xi_i^{\nu}$$

As with the Personnaz model, the following relationships hold for the attractors of the network:

$$S_i = sgn(h_i) \tag{3.16}$$

$$h_i = \sum_{j,j\neq i} J_{ij}\ S_j \tag{3.17}$$

As discussed earlier, at finite temperature (T) the energy function cannot be used to find the attractors of the network (i.e. the ground states). The proper function is called the free energy, which contains not only the energy function, but the entropy and the temperature. So, at finite temperature the general formula for the free energy is:

$$F = -\beta\ \ln Tr_{\{S_i\}}\ exp(-\beta E), \qquad \beta = 1/T \tag{3.18}$$

Notice how the expression for the energy has the J_{ii} term while the equation for h_i does not. Calculating as done for the Personnaz model we find that

$$-\frac{1}{2}\sum_{i} J_{ii} = -\frac{1}{2} N\ \alpha \tag{3.19}$$

As is well known, the energy can be rescaled by a constant value without changing the

dynamics of the problem. To formally eliminate the diagonal terms from the energy, we specify the synaptic coefficient as $J_{ij}(1 - \delta_{ij})$. In the thermodynamic limit ($N \to \infty$) we can rewrite this as $J_{ij} - \alpha\delta_{ij}$. Consider the eigenvalue equation discussed earlier for the Personnaz model. This equation rewritten using the new form for J_{ij} is

$$\sum_j (J_{ij} - \alpha\delta_{ij})\, \xi_j^\mu = (1 - \alpha)\, \xi_i^\mu \tag{3.20}$$

Thus, stored patterns are eigenvectors of the synaptic matrix with eigenvalue 1 α, while patterns orthogonal to them have eigenvalue of $-\alpha$.

3.5.1. Connection with Hopfield Model

Let us make the following decomposition for S_i:

$$S_i = \sum_\mu a_\mu\, (\xi_i^\mu + 1) + \delta S_i \tag{3.21}$$

The pattern $\{\delta S_i\}$ is orthogonal to all embedded patterns. In other words:

$$\sum_i \delta S_i\ \xi_i^\mu = 0 \tag{3.22}$$

Now a measure of the order in the system is conventionally given by

$$m_\mu = N^{-1} \sum_i S_i\ \xi_i^\mu \tag{3.23}$$

These are used to parametrize the attractor distributions at finite temperature. Traditional nomenclature calls the m_μ's order parameters. From a neural network standpoint, they represent the average overlap of the network with a stored pattern. When the number of patterns is much less than the number of neurons and the learning is local, this is enough information to parametrize what are called attractor distributions. When nonlocal learning is used, these overlap parameters must be modified before

they can be properly used. Unfortunately, at memory saturation (i.e. $p \simeq N$) even this will not suffice.

Before continuing with the comparison of a Hopfield network and a nonlocal learning ANN, let me clarify the idea of an attractor distribution. Throughout this thesis we have always referred to a single attractor state (or its simplest version – a fixed point). The notion of a single attractor state is only proper at zero temperature. At non-zero temperatures the probability of a spin flipping away from a stable state is non-zero (it is zero at zero temperature). We find in this situation a collection of states close to a stored pattern with non-zero probability. Each of these states is characterized by the fact that given the (average) overlap of the network state with a stored pattern, we find that the network is trapped in the neighborhood of the stored pattern. This collection of states is called an attractor distribution.

Since overlaps are tangible quantities, let us see if we can find a relationship between the m_μ and the a_μ. Crudely, we can think of the a_μ as the network overlaps with the stored **ORTHOGONAL** patterns and not the original patterns. Multiply equation (3.21) by ξ_i^μ and then sum over all i. We obtain the following:

$$\sum_i S_i \, \xi_i^\nu = \sum_i \sum_\mu a_\mu \, \xi_i^\mu \, \xi_i^\nu + \sum_i \delta S_i \, \xi_i^\nu \tag{3.24}$$

Using equation (3.22), the right-hand side of equation (3.24) becomes:

$$\sum_i \sum_\mu a_\mu \, \xi_i^\mu \, \xi_i^\nu \tag{3.25}$$

Now using equation (3.23) and equation (3.25), equation (3.24) becomes:

$$m_\nu N = \sum_i \sum_\mu a_\mu \, \xi_i^\mu \, \xi_i^\nu \tag{3.26}$$

Notice that the right hand side can be rewritten in a form that uses the overlap matrix

C defined in equation (3.1). Thus we obtain:

$$m_\nu N = \sum_\mu a_\mu N \; C_{\mu\nu} \tag{3.27}$$

Now, multiply both sides of equation (3.26) by $(C^{-1})_{\beta\nu}$ and then sum over ν. Thus we can rewrite equation (3.26) as:

$$\begin{aligned}\sum_\nu (C^{-1})_{\beta\nu} m_\nu &= \sum_{\mu,\nu} a_\mu \, (C^{-1})_{\beta\nu} \, C_{\nu\mu} \\ &= \sum_\mu a_\mu \delta_{\beta\mu} \\ &= a_\beta \end{aligned} \tag{3.28}$$

With the relationship between a_μ and m_μ established, let us rewrite the local field equation h_i and the energy function in terms of the "overlap" parameters.

$$\begin{aligned} h_i &= \sum_{j, j\neq i} J_{ij} \; S_j \\ &= \sum_j J_{ij} \; S_j - J_{ii} \; S_i \\ &= \sum_j N^{-1} \sum_{\mu,\nu} \xi_i^\mu \, \xi_j^\nu \, (C^{-1})_{\mu\nu} \, S_j \\ &= \sum_{\mu,\nu} \xi_i^\mu \, (C^{-1})_{\mu\nu} \, m_\nu \\ &= \sum_\mu \xi_i^\mu \, a_\mu - J_{ii} \; S_i \\ &\approx \sum_\mu \xi_i^\mu \, a_\mu - \alpha \, S_i \\ &= \sum_\mu \xi_i^\mu \, a_\mu - \alpha(\sum_\mu a_\mu \, \xi_i^\mu + \delta S_i) \\ &= (1 - \alpha) \sum_\mu \xi_i^\mu \, a_\mu - \alpha \, \delta S_i \end{aligned} \tag{3.29}$$

Now, let us rewrite the energy function:

$$E = -\frac{1}{2}\sum_{i,j} J_{ij}\, S_i\, S_j$$

$$= -\frac{1}{2}\sum_{i} h_i\, S_i$$

$$= -\frac{1}{2}\sum_{\mu} \xi_i^{\mu}\, a_{\mu}\, S_i - \sum_{i} J_{ii}\, S_i^2 \tag{3.30}$$

Using equation (3.23) we can finally write:

$$E = -\frac{N}{2}\sum_{\mu} a_{\mu}\, m_{\mu} \tag{3.31}$$

The corresponding equations in the Hopfield case are the following:

$$h_i = \sum_{\mu} \xi_i^{\mu}\, m_{\mu} - \alpha\, S_i \tag{3.32}$$

$$E = -\frac{N}{2}\sum_{\mu} (m_{\mu})^2 \tag{3.33}$$

Immediately, one can see that equations (3.29) and (3.30) are an advantage over (3.32) and (3.33). Consider the situation where $S_i = \xi_i^1$. In the case of the m_{μ}'s we obtain $m_{\mu} - \delta_{1\mu} = (C^{-1})_{1\mu}$ which is $O(1/\sqrt{N})$, no matter how the patterns are correlated. On the other hand, $a_{\mu} = \delta_{1\mu}$ regardless of the correlation of the patterns.

3.5.2. Global Minima of E

Consider the local fields of the configuration $S_i = \xi_i^{\mu}$. They are:

$$h_i = \xi_i^{\mu}\,(1 - J_{ii}) \simeq \xi_i^{\mu}\,(1 - \alpha) \tag{3.34}$$

For stability of all the patterns we must have $J_{ii} \leq 1$. We will now prove that $0 \leq J_{ii} \leq 1$.

$$\sum_{j} J_{ij}^2 = N^{-2} \sum_{j} \sum_{\mu,\nu} \sum_{\gamma,\delta} \xi_i^{\mu}\, \xi_j^{\nu}\, \xi_i^{\gamma}\, \xi_j^{\delta}\, (C^{-1})_{\mu\nu}\, (C^{-1})_{\gamma\delta}$$

$$= N^{-1} \sum_{\mu,\nu} \sum_{\gamma,\delta} \xi_i^{\mu}\, \xi_i^{\gamma}\, C_{\nu\delta}\, (C^{-1})_{\mu\nu}\, (C^{-1})_{\gamma\delta}$$

$$= N^{-1} \sum_{\mu,\nu} \sum_{\gamma} \xi_i^{\mu} \xi_i^{\gamma} \delta_{\nu\gamma} (C^{-1})_{\mu\nu}$$

$$= N^{-1} \sum_{\mu,\nu} \xi_i^{\mu} \xi_i^{\nu} (C^{-1})_{\mu\nu}$$

$$= J_{ii} \tag{3.35}$$

Since this relationship holds for $i=1,2,...,N$ we have:

$$J_{ii} - J_{ii}^2 = \sum_{j,j\neq i} J_{ij}^2 \geq 0 \tag{3.36}$$

Immediately from equation (3.36) we have that $0 \leq J_{ii} \leq 1$ for $i=1,2,...,N$. Combining this with equation (3.34) we see that the patterns will be stable for all $p < N$. Let us consider the following formulation for E. Define the quantity to be:

$$\Delta = \left[\sum_i (\delta S_i)^2 \right]^{1/2} \tag{3.37}$$

This is the Euclidean distance between the vector (N-dimensional) $\{S_i\}$ and the p-dimensional subspace spanned by the patterns. Squaring equation (3.21) and then summing over i we obtain:

$$\sum_i S_i^2 = \sum_i ((\sum_{\mu} a_{\mu} \xi_i^{\mu} + \delta S_i)(\sum_{\nu} a_{\nu} \xi_i^{\nu} + \delta S_i)) \tag{3.38}$$

$$= \sum_i (\sum_{\mu} a_{\mu} \xi_i^{\mu} \sum_{\nu} a_{\nu} \xi_i^{\nu} + \sum_{\mu} a_{\mu} \xi_i^{\mu} \delta S_i + \sum_{\nu} a_{\nu} \xi_i^{\nu} \delta S_i + (\delta S_i)^2)$$

Using equations (3.22) and equation (3.28) we obtain:

$$N = \sum_i (\sum_{\mu,\nu} a_{\mu} a_{\nu} \xi_i^{\mu} \xi_i^{\nu} + (\delta S_i)^2) \tag{3.39}$$

Multiply equation (3.28) by $\xi_i^{\mu}\xi_i^{\nu}$ and sum over ν and i to obtain:

$$\sum_{\nu} \sum_i a_{\nu} \xi_i^{\mu} \xi_i^{\nu} = \sum_{\nu} \sum_i \sum_{\beta} (C^{-1})_{\nu\beta} m_{\beta} \xi_i^{\mu} \xi_i^{\nu}$$

$$= N \sum_{\nu} \sum_{\beta} (C^{-1})_{\nu\beta} m_{\beta} C_{\mu\nu}$$

$$= N m_{\mu} \tag{3.40}$$

After all this we can finally give the final form of equation (3.39):

$$N = N\sum_{\mu} a_{\mu}\, m_{\mu} + (\delta S_i)^2 \tag{3.41}$$

Notice how the first term on the right is the expression for the energy function. Thus, we can rewrite the energy function in terms of the Euclidean distance between the space of network states and the space of patterns.

$$E = -\frac{N}{2} + \frac{\Delta^2}{2} \tag{3.42}$$

What has this reformulation given us? It immediately shows that the p patterns are global minima of E (they all have $\Delta = 0$). Unhappily, states that are linear combinations of the patterns, are also global minima of E (i.e. these are spurious states of the dynamics). Kanter [Kanter87] has shown that the probability for large N for the occurrence of a state that is a linear combination of patterns vanishes as $N \rightarrow \infty$ as:

$$P \sim 4\left[\frac{p}{3}\right]\left[\frac{3}{4}\right]^N$$

The last topic to be discussed with this formulation is to find the value for α_c. Our sole change of the Personnaz model is to remove the $j = i$ term in the sum for the local field h_i. Let us redo the calculation for finding α_c We can then immediately rewrite equation (3.14) without the J_{11}

$$S_1 = sgn[(1 - \alpha)\xi_1^{\mu}] \tag{3.43}$$

We readily see that if $S_1 = -\xi_1^{\mu}$ that for $\alpha < 1$ the network will flow to a state where $S_i = \xi_i^1$. Thus the maximum capacity of the memory is given by $\alpha_c = 1$.

3.6. Simulations of Network

The C code for a nonlocal learning neural network can be found in the Appendix of this thesis. I refer you there for further information on the network.

CHAPTER 4

REPLICA METHOD AND GRAPH PARTITIONING

4.1. General Information

It is well known that spin glasses on random lattices with finite connectivity are closely associated with combinatorial optimization problems such as graph partitioning [Fu86, Mezard85, Kanter87]. There is considerable formal understanding of infinite-ranged (i.e. all spins interact with all other spins) spin glasses. Many physical and conceptual problems still arise but they are not relevant to this discussion. Finite connectivity is just a statement that a spin is connected to only a subset of all spins. A random lattice is specified by a collection of interactions that are random in sign and generally different in magnitude. In the case of an associative memory, as discussed in Chapter 3, these random interactions would give rise to the stored patterns. It is the purpose of this chapter to solve the q-partitioning problem.

4.2. Problem Specification

In general, graph q-partitioning (i.e. a graph is partitioned into q equal parts) is a NP-complete problem, i.e. no known algorithm can guarantee to find the absolute optimal solution in polynomial time. The particular problem to be studied in this chapter is the following: take a graph specified by $G(V,E)$, where $|V| = N$ is an integral multiple of q. The graph is then to be partitioned into q equal groups of vertices such that the intergroup edges (i.e. connection between two vertices) are minim-

ized. In this case an Ising spin system will not be used. A Potts spin system will be utilized instead. The energy function in this case is:

$$E = -\frac{2}{q}\sum_{i<j} J_{ij}(q\,\delta_{s_i s_j} - 1) \tag{4.1}$$

where δ is called the Kronecker delta symbol and has the following property:

$$\delta_{s_i s_j} = \begin{cases} 1 & \text{if } s_i = s_j \\ 0 & \text{if } s_i \neq s_j \end{cases} \tag{4.2}$$

The J_{ij}'s represent the randomness of the system. In the graph partitioning case we specify each edge by J_{ij} such that:

$$J_{ij} = \begin{cases} \text{J} & \text{with probability } p \\ 0 & \text{with probability } 1 - p \end{cases} \tag{4.3}$$

This probability distribution can be explicitly written as:

$$p(J_{ij}) = p\,\delta(J_{ij} - J) + (1 - p)\,\delta(J_{ij}) \tag{4.4}$$

In this case δ is the Dirac delta function which is defined as follows:

$$\delta(x - a) = \begin{cases} 0 & \text{if } x \neq a \\ \infty & \text{if } x = a \end{cases} \tag{4.5}$$

The Dirac delta function is more properly defined in terms of an integral equation, such as:

$$\int f(x)\delta(x - a)dx = f(a) \tag{4.6}$$

So far we have dealt with very formal concepts. We should not lose touch with the fact that $J_{ij} = J$ represents the presence of an edge in the graph. Before we can continue we must specify some condition (constraint) that must be satisfied for the graph to be partitioned into equal parts. It must be understood that the S_i are not binary

spins as discussed earlier. They will be considered to be one of the q-roots of unity. In particular choose:

$$S_i \in \{exp(2\pi in/q) \mid n = 0, 1,..., q-1\} \tag{4.7}$$

With this representation for S_i we can specify the constraint as:

$$\sum_i S_i^r = 0 \quad \text{for } r = 1, 2,..., q-1 \tag{4.8}$$

4.3. Cost Function

For a Statistical Mechanics treatment to be fruitful we must determine a cost function to be minimized and its connection to the energy function. We already know that we want to minimize the number of connections between partitions. It will turn out that this is easily translated into a formal cost function. Let us rewrite the energy function taking care to specify all possible situations for the combinations of i and j. The expanded energy function is:

$$\begin{aligned} E = -(q-1)&\left[\sum_{i,j\in V_1} + \cdots + \sum_{i,j\in V_q}\right] J_{ij}/q \\ +&\left[\sum_{\substack{i\in V_1\\ j\in V_2}} + \cdots + \sum_{\substack{i\in V_1\\ j\in V_q}} + \cdots + \sum_{\substack{i\in V_q\\ j\in V_1}} + \cdots + \sum_{\substack{i\in V_q\\ j\in V_{q-1}}}\right] J_{ij}/q \end{aligned} \tag{4.9}$$

Since the only contribution to the sums will come when $J_{ij} = J$ we can write:

$$E = \frac{-p(q-1)JN(N-1)}{q} + 2CJ \tag{4.10}$$

where C is defined as:

$$C = \frac{E}{2J} + \frac{N(N-1)p(q-1)}{2q} \tag{4.11}$$

In each of the sums that make up C we are adding up connections (edges) between partitions. Thus, if we minimize C we have succeeded in minimizing the interpartition

edges.

4.4. Free Energy Expression

Starting with equation (2.14) we can specify the average of Z^n where Z is the "one"-particle partition function. For the average we have:

$$\overline{Z^n} = Tr'_{\{S^\alpha\}} \prod_{i<j}^{N} \int dJ_{ij} p(J_{ij}) exp(-\beta \sum_{\alpha=1}^{n} J_{ij}(\delta_{S_i^\alpha S_j^\alpha} - 1)) \tag{4.12}$$

Using the probability distribution for the interactions, equation (4.4), we obtain:

$$\overline{Z^n} = Tr'_{\{S^\alpha\}} \prod_{i<j}^{N} \left[p \exp\left[\frac{2\beta J}{q} \sum_{\alpha=1}^{n} (q\,\delta_{S_i^\alpha S_j^\alpha} - 1) \right] + (1-p) \right]$$

$$= (1-p)^{N(N-1)/2} Tr'_{\{S^\alpha\}} \exp \sum_{i<j}^{N} \ln \left[1 + p_0 \exp\left[\frac{2\beta J}{q} \sum_{\alpha=1}^{n} (q\,\delta_{S_i^\alpha S_j^\alpha} - 1) \right] \right] \tag{4.13}$$

where p_0 is defined as $p/(1-p)$. Remember at each stage of the analysis the constraint, equation (4.8), must be satisfied. To indicate this the trace has had a prime appended to it. At this stage the expression for the average of the partition function is still too cumbersome. To reduce the complexity of the partition function and to reach the point where we can explicitly use the constraint, expand the exponential and the natural logarithm into their power series form. I will detail the steps here since the techniques and philosophy are an integral part of the analysis. For simplicity define:

$$x = \frac{2\beta J}{q} \sum_{\alpha=1}^{n} (q\,\delta_{S_i^\alpha S_j^\alpha} - 1) \tag{4.14}$$

Expand the natural logarithm and then the exponential in their power series form:

$$\ln(1 + p_0\, exp(x)) = \sum_{l=1}^{\infty} \frac{(-1)^{l+1} {p_0}^l}{l} exp(lx)$$

$$= \sum_{l=1}^{\infty} \frac{(-1)^{l+1} {p_0}^l}{l} \left[1 + \sum_{k=1}^{\infty} \frac{l^k x^k}{k!} \right]$$

$$= ln(1 + p_0) + \sum_{k,l=1}^{\infty} \frac{(-1)^{l+1} {p_0}^l}{l} \frac{l^k x^k}{k!} \tag{4.15}$$

For the average of the partition function we now have:

$$\overline{Z^n} = (1-p)^{N(N-1)/2} \exp\left\{ \sum_{i<j}^{N} \left[ln(1 + p_0) + \sum_{k,l=1}^{\infty} \frac{(-1)^{l+1} p_0^l}{l} \frac{l_k x^k}{k!} \right] \right\}$$

$$= (1-p)^{N(N-1)/2} \exp\left\{ \frac{N(N-1)}{2} ln(1 + p_0) \right\}$$

$$\times \exp\left\{ \sum_{i<j}^{N} \left[\sum_{k,l=1}^{\infty} \frac{(-1)^{l+1} p_0^l}{l} \frac{l^k}{k!} x^k \right] \right\} \tag{4.16}$$

If we define the following coefficients we can simplify the formula:

$$c_k = \sum_{l=1}^{\infty} \frac{(-1)^{l+1}}{k} p_0^l \frac{l^k}{k!} \tag{4.17}$$

If x is now written out explicitly the following formula is obtained:

$$\overline{Z^n} = (1-p)^{N(N-1)/2} \exp\left\{ \frac{N(N-1)}{2} ln(1 + p_0) \right\}$$

$$\times \exp\left\{ \sum_{k=1}^{\infty} \left[\frac{2\beta J}{q} \right]^k c_k \sum_{i<j}^{N} \left[\sum_{\alpha=1}^{n} (q \delta_{S_i^\alpha S_j^\alpha} - 1) \right]^k \right\} \tag{4.18}$$

Now from the properties of the natural log and the exponential functions we have:

$$(1-p)^{N(N-1)/2} \left\{ \frac{N(N-1)}{2} ln(1 + p_0) \right\} = 1 \tag{4.19}$$

Remember that the whole purpose of these rewritings is to put the average of the partition function into a form where the constraint can be explicitly applied. Let us modify

the sum on i and j to include all combinations of i and j. By multiplying by 1/2 we can easily adjust for the inclusion of terms when $i > j$. The analysis is slightly more involved to remove the self-coupling terms (i.e. when $i = j$). The correction term is calculated using the following idea: calculate the general term when $i = j$ and change the sign of the exponent. This term is easily seen as:

$$Correction\_term = \exp\left\{-\frac{N}{2}\sum_{k=1}^{\infty}\left[\frac{2\beta J}{q}\right]^k (q-1)^k n^k\right\} \tag{4.20}$$

We are now ready to write a form of $\overline{Z^n}$ in which the effect of the constraint and the limit as $n \rightarrow 0$ can be easily seen:

$$\overline{Z^n} = Tr'_{\{S^\alpha\}} \exp\left[\frac{-N}{2}\sum_{k=1}^{\infty}(\beta J)^k c_k n^k\right]$$

$$\times \exp\left\{\sum_{k=2}^{\infty}\frac{1}{2}\left[\frac{2\beta J}{q}\right]^k c_k \sum_{i,j}^{N}\left[\sum_{\alpha=1}^{n}(q\,\delta_{S_i^\alpha S_j^\alpha} - 1)\right]^k\right\} \tag{4.21}$$

A variation, but completely equivalent version of equation (4.8), can be used to simplify equation (4.21). This version is the following:

$$\sum_{i,j}^{N}\left[\delta_{S_i^\alpha S_j^\alpha} - \frac{1}{q}\right] = 0 \tag{4.22}$$

The relationship between the energy of the system and the free-energy of the system is:

$$F = E - \frac{S}{\beta} \tag{4.23}$$

where S is the entropy of the system. In the limit that $\beta \rightarrow \infty$ we immediately see that the free-energy F and the energy E are equal. Thus we can rewrite our cost function in the following form:

$$C = \frac{N(N-1)p(q-1)}{2q} + \lim_{\beta\to\infty}\frac{F}{2J} \tag{4.24}$$

No matter what form we have the free-energy in, to calculate any values we must, in the final analysis, calculate a sum over all possible spin configurations (i.e. perform the trace). There is a spin representation in which we can replace the delta function with a sum over spin variables. This representation is:

$$q\,\delta_{S_iS_j} = 1 + \sum_{r=1}^{q-1}(S_iS_j^*)^r \tag{4.25}$$

The formula for the free-energy in terms of the partition function is:

$$F = -\frac{1}{\beta}\ln Z \tag{4.26}$$

If we use equation (2.14) we obtain:

$$-\beta F = \lim_{n\to 0}\frac{1}{n}\left[Tr'_{\{S^\alpha\}}\exp\left\{\sum_{k=2}^{\infty}\frac{1}{2}\left[\frac{2\beta J}{q}\right]^k c_k\sum_{i,j}^{N}\left[\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}S_i^\alpha S_j^{\alpha *}\right]^k\right\} - 1\right] \tag{4.27}$$

Notice that the first exponential in equation (4.21) is 1 in the limit when $n \to 0$.

It was necessary in Chapter 2 to rescale J so as to guarantee that the local fields would not be dependent on N. In that case, from a formal standpoint this guaranteed a sensible thermodynamic limit (i.e. $N \to \infty$). Since c_k is $O(1)$ with regard to N, a rescaling of $J \to J/\sqrt{N}$ will lead to a sensible limit. Each term in the sum making up the free-energy formula will be of $O(1/N^{k/2})$. Therefore in the thermodynamic limit the first term in the series will dominate. In this case:

$$c_2 = \frac{1}{2}p_0(1+p_0)^{-2} = \frac{1}{2}p(1-p) \tag{4.28}$$

The double sum on i and j can also be rewritten as:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}(S_i^{\alpha}S_j^{\alpha^*})^r\right]^2 = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}(S_i^{\alpha}S_j^{\gamma^*})^r\sum_{\gamma=1}^{n}\sum_{r'=1}^{q-1}(S_i^{\gamma}S_j^{\gamma^*})^{r'}$$

$$= \sum_{\gamma=1}^{n}\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}\sum_{r'=1}^{q-1}\sum_{i=1}^{N}\sum_{j=1}^{N}(S_i^{\alpha})^r(S_i^{\gamma})^{r'}(S_j^{\alpha^*})^r(S_j^{\gamma^*})^{r'}$$

$$= 2N\sum_{\substack{\gamma=1\\ \alpha<\gamma}}^{n}\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}\sum_{r'=1}^{q-1}\left|\sum_{i=1}^{N}(S_i^{\alpha})^r(S_i^{\gamma})^{r'}\right|^2 + \gamma{=}\alpha\_term \tag{4.29}$$

where

$$\gamma{=}\alpha\_term = N(q-1)K^2p(1-p) \tag{4.30}$$

Combining equation (4.29) and (4.30) we obtain for the free-energy:

$$-\beta F = N(q-1)K^2p(1-p)$$

$$+\lim_{n\to 0}\frac{1}{n}\left\{Tr'_{\{S^{\alpha}\}}\exp\left[\frac{2NK^2p(1-p)}{N}\sum_{\substack{\gamma=1\\ \alpha<\gamma}}^{n}\sum_{\alpha=1}^{n}\sum_{r=1}^{q-1}\sum_{r'=1}^{q-1}\left|\sum_{i=1}^{N}(S_i^{\alpha})^r(S_i^{\gamma})^{r'}\right|^2\right]-1\right\} \tag{4.31}$$

where

$$K = \frac{\beta JN^{1/2}}{q} \tag{4.32}$$

Notice how the calculation of the trace will still involve the constraint as specified in equation (4.8) or equivalently in equation (4.22). Rewriting $\overline{Z^n}$ by the use of Gaussian transformations and Lagrange multipliers, Fu and Anderson [Fu86] have shown that constraints like equation (4.8) are irrelevant at $T = 0$. Thus we can remove the prime on the trace.

Our stated goal is to minimize the cost function which in this case represents the minimizing of the cross partition edges. As briefly stated in Chapter 3 an analytic continuation must be performed before the limit as $n \to 0$ is taken. This is necessary since we must reduce n through non-integral values before 0 is reached. I will state

the results here without further elaboration. The Edwards-Anderson parameter ($Q_{\alpha\gamma}$) gives a measure of "order" in the system. It is a symmetric matrix (i.e. $Q_{\alpha\gamma} = Q_{\gamma\alpha}$) with a zero diagonal. Thus the order of the system will be parametrized by $\frac{1}{2}n(n-1)$ parameters. As we approach 0 from above, Q is acting on a space of negative dimension when $n < 1$. The eigenvalues of the Hessian matrix (i.e. the second derivative of the free-energy with respect to $Q_{\alpha\gamma}$) indicate the stability of a solution of the equations for the extrema of the free-energy. If the eigenvalues are positive then the solution is stable. In the case of negative dimensions the role of positive and negative eigenvalues becomes reversed. Thus to find a minimum of the free-energy we will have to look for a maximum of the (Gaussian) transformed free-energy expression. The free-energy expression can be written as:

$$\frac{\beta F}{N} = -(q-1)K^2 p(1-p) + Max\{f(Q)\} \tag{4.33}$$

f(Q) is obtained in the following manner. The Gaussian transformation is:

$$\exp\left[\frac{A}{2}O^2\right] = C\int_{-\infty}^{\infty} dx \exp\left[-\frac{x^2}{2A} + xO\right] \tag{4.34}$$

The Edwards-Anderson order parameter $Q_{\alpha\gamma}$ is defined as:

$$Q_{\alpha\gamma} \equiv \frac{1}{N}\sum_{i=1}^{N} S_i^{\alpha} S_i^{\gamma} \tag{4.35}$$

$$Q_{\alpha\gamma} = \begin{cases} Q & \text{if } \alpha = \gamma \\ 0 & \text{otherwise} \end{cases} \tag{4.36}$$

Introducing $Q_{\alpha\gamma}$ as the auxiliary (i.e. x in equation (4.34)) variable in a Gaussian transformation we obtain after much algebra:

$$f(Q)=\lim_{n\to 0}\frac{1}{n}\left\{\frac{q-1}{8K^2p(1-p)}\sum_{\alpha<\gamma}Q_{\alpha\gamma}^2-\ln Tr_{\{S^\alpha\}}\exp\left[\sum_{\alpha<\gamma}Q_{\alpha\gamma}^2\sum_{r=1}^{q-1}(S^\alpha S^\gamma)^r\right]\right\} \tag{4.37}$$

When we have $Q_{\alpha\gamma}$ the same for all α and γ we say that we have replica symmetry. If they were different we would say that we have replica symmetry breaking. Even though replica symmetry breaking will lead to a more precise answer, this paper will only concern itself with the replica symmetric solution.

Taking the limit as $n \to 0$ and using saddle point expansion (along with a considerable amount of Algebra) the following expression is obtained for the cost function:

$$C = \frac{N^2p(q-1)}{2q} + \frac{u_qN^{3/2}[p(1-p)]^{1/2}}{2} \tag{4.38}$$

The portion of the ground state energy of a "Potts" spin glass that is independent of p and N is given by u_q and has the following calculated values:

q	$-u_q$
2	0.80
3	1.00
4	1.10

Comparing with the simulations in the case of $N = 99$, $p = 0.8$, and $q = 3$, the following is obtained:

$$\frac{C_{sim}}{C} = \frac{200.9477}{197.0075} = 1.02 \tag{4.39}$$

Thus the calculated result is nearly equal to the simulated result. This result should expect to become better and better as N increases since all the formulas are only proper in the limit as $N \to \infty$.

CHAPTER 5

CONCLUSIONS

5.1. State Of The Subject

Simulated annealing was developed as a means to calculate numerically the ground state energy of physical systems. Unfortunately, much trial and error needs to be performed before the technique gives fruitful results. This thesis has analyzed a particular neural network using nonlocal learning rules. The behavior of the network was studied in the zero temperature (noiseless) realm but with the techniques used to "solve" the q-partitioning problem in Chapter 4, it could be studied in all temperature realms. Furthermore, physical systems that are used to model neural networks are teaming with energy minima. To accurately describe such a situation, replica permutation symmetry must be broken. Only in this situation will more detailed information be gained about a network, especially at transition points.

In Chapter 4 we undertook to "solve" the q-partitioning problem using the full power of the replica method. A solution by the replica method is not guaranteed to be optimal, but it is, at worst, near optimal (in NP complete cases). The q-partitioning problem could be solved, only because a proper rescaling of J could be performed that guaranteed a proper thermodynamic limit. Similar energy functions can be written for graph coloring problems (including Ramsey number calculations) such as:

$$E = \sum_{i<j}^{N} J_{ij}\delta_{S_iS_j} \tag{5.1}$$

Unfortunately, all energy functions similar to the one in equation (5.1) are plagued with one major problem. A calculation of $Z^{\overline{n}}$ generates an expression in which all k terms contribute (remember in the q-partitioning case we discarded all terms except the $k = 2$ term). Simply stated, we find ourselves in a situation in which no known technique will give a rescale of J that will lead to a proper thermodynamic limit and, thus, no solution, optimal or not, can be found. Learning how to tune the energy functions, and then rescale J in such situations, needs to be attacked from both a physical and mathematical standpoint. It is my hope that this thesis has shown the power and utility of statistical mechanics when dealing with large, highly-interacting systems of particles.

5.2. Future Work

The analysis of the neural network in Chapter 4 was done at $T = 0$. Future analysis could be undertaken to study the following questions:

(1) Are the stored patterns stable to simultaneous flips of many spins?

(2) What is the effect of non-zero temperature on the stability of stored patterns?

(3) How does non-zero temperature effect the size of the basin of attractions?

(4) What are the nature of spurious states other than the ones which are linear combinations of the stored patterns?

All of these questions can be analytically studied as well as simulated by a computer program.

The nonlocal learning rule specified in this thesis is not unique. Nonlocal networks that exhibit memory blackout, instead of memory loss, can be generated. It is also possible to modify this nonlocal idea to simulate hierarchical networks in which low-lying attractors are deepened by strengthening certain synapses, and eliminating others. Lastly, a topic that is only now being addressed, is how to use ANN's to perform abstract calculations. For example, how would an abacus be built?

BIBLIOGRAPHY

[Amit89]
D.J. Amit, Modeling Brain Function, Cambridge University Press, Cambridge, 1989.

[Binder86]
K. Binder and A.P. Young, Spin glasses: Experimental facts, theoretical concepts and open questions, Reviews of Modern Physics, Vol. 58, No. 4, 801(October 1986).

[Brout59]
R. Brout, Phys. Rev. 115, 824.

[Feynman62]
R.P. Feynman, Statistical Mechanics, The Benjamin/Cummings Publishing Company, Inc., Reading, 1962.

[Fu86]
Yaotian Fu and P.W. Anderson, Application of statistical mechanics to NP-complete problems in combinatorial optimisation, J. Phys. A, 19(1986) 1605-1620.

[Gutfreund85]
H. Gutfreund and D.J. Amit, Spin-glass models of neural networks, Physical Review A, Vol. 32, No. 2, 1007(August 1985).

[Hopfield82]
J.J. Hopfield, Neural networks and physical systems with emergent selective computational abilities, Proc. Natl. Acad. Sci. USA, 79, 2554(1982).

[Hopfield84]
J.J. Hopfield, Proc, Natl. Acad. Sci. USA B 1, 3088(1984).

[Huang62]
K. Huang, Statistical Mechanics, John Wiley, New York, 1963.

[Kanter87]
I. Kanter and H. Sompolinsky, Associative recall of memory without errors, Vol. 35, No. 1, 380(January 1987).

[Kohonen84]
T. Kohonen, Self-Organization and Associative Memory (Springer-Verlag, Berlin, 1984).

[Kubo65]
R. Kubo, Statistical Mechanics, Interscience Publishers, New York, 1965.

[Little74]
W.A. Little, Math. Biosci. 19, 101(1974).

[Mezard85]
M. Mezard and G. Parisi, Replicas and optimization, J. Physique Lett., 46(1985) L-771 - L-778.

[Pathria80]
R.K. Pathria, Statistical Mechanics, Pergamon Press, New York, 1980.

[Pitts43]
W.A. Pitts and W.S. McCulloch, A logical calculus of the ideas immanent in neural nets, Bull. Math. Biophys., 5, 115(1943).

[Rosenblatt62]
F. Rosenblatt, Principles of Neurodynamics (Spartan Books, Washington DC, 1962).

[Widrow60]
B. Widrow and M.E. Hoff, Adaptive switching circuits, IREWESCON Convention Record, 4, 4-96(1960).

[Willshaw69]
D.J. Willshaw, O.P. Buneman and H.C. Longuet-Higgins, Nonholographic associative memory, Nature, Lond., 222, 960(1969).

APPENDIX

NETWORK SIMULATIONS

The following is a description of the results from simulations of a nonlocal learning neural network that learned 30 Chinese characters. The characters are in block 16 form.

Out1 contains the output of the network with an initial network configuration which is a distortion of the 1^{st} Chinese character.

Out5 contains the output of the network with an initial network configuration which is a distortion of the 5^{th} Chinese character.

Out7 contains the output of the network with an initial network configuration which is an extreme distortion of the 7^{th} Chinese character.

Out10 contains the output of the network with an initial network configuration which is a distortion of the 10^{th} Chinese character.

Out30 contains the output of the network with an initial network configuration which is a distortion of the 30^{th} Chinese character.

The Chinese characters that we hope to retrieve in the above simulations are in the files chinese1, chinese5, chinese7, chinese10 and chinese30. The files in this appendix are printed in the following order:

chinese1
out1
chinese5
out5
chinese7
out7
chinese10
out10
chinese30
out30
makefile
replica.h
replica.c
attractor.h
net_utils.c
random.h
dynamics.c

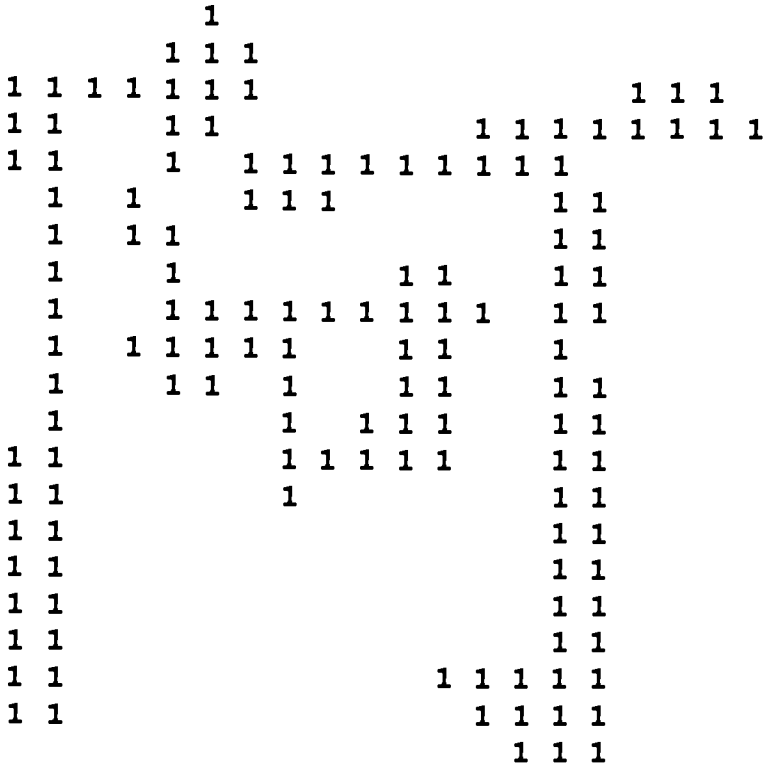
update_utils.c
report.c
init_utils.c
pattern.c
inverse.c

The network is run in the following way. There is a file called *spin_config* which contains the initial configuration of the network. The patterns to be learned must each reside in a separate file. There is no restriction on the file names of the patterns. They will be read into the network by including them as arguments at run time. The system is run and compiled in the following manner:

```
make replica
replica pat1 pat2 pat3 pat4 pat5 ...
```

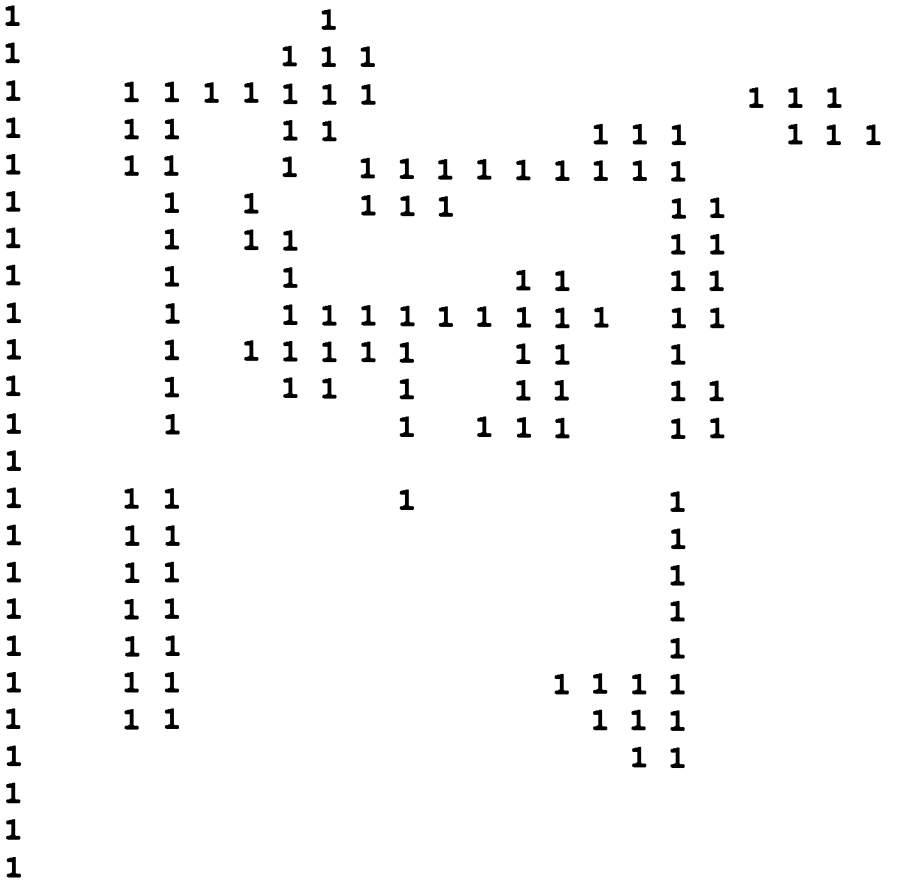
All output of the network is sent to standard output.

CHINESE 1



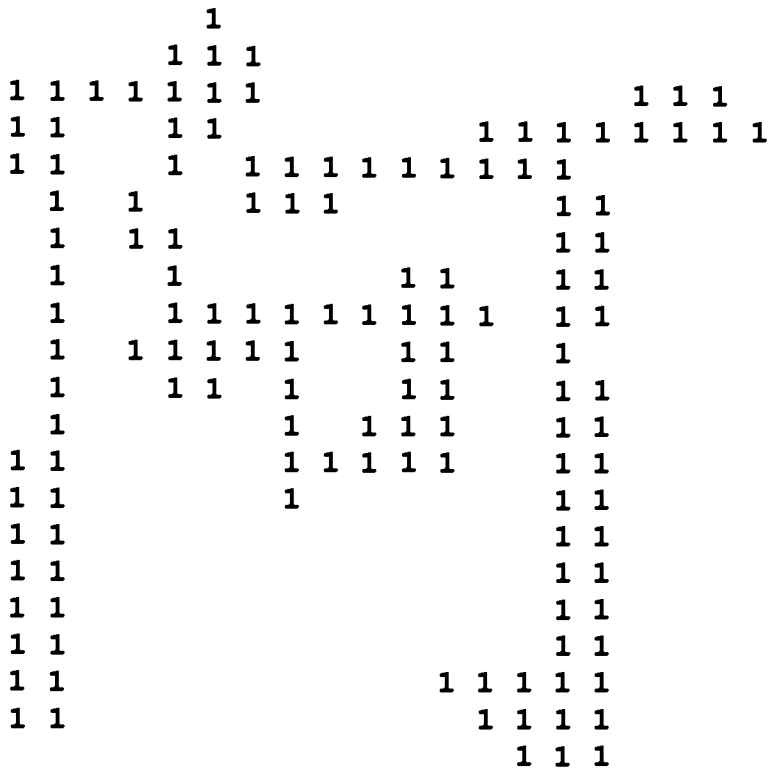
OUT 1

Initial Network State



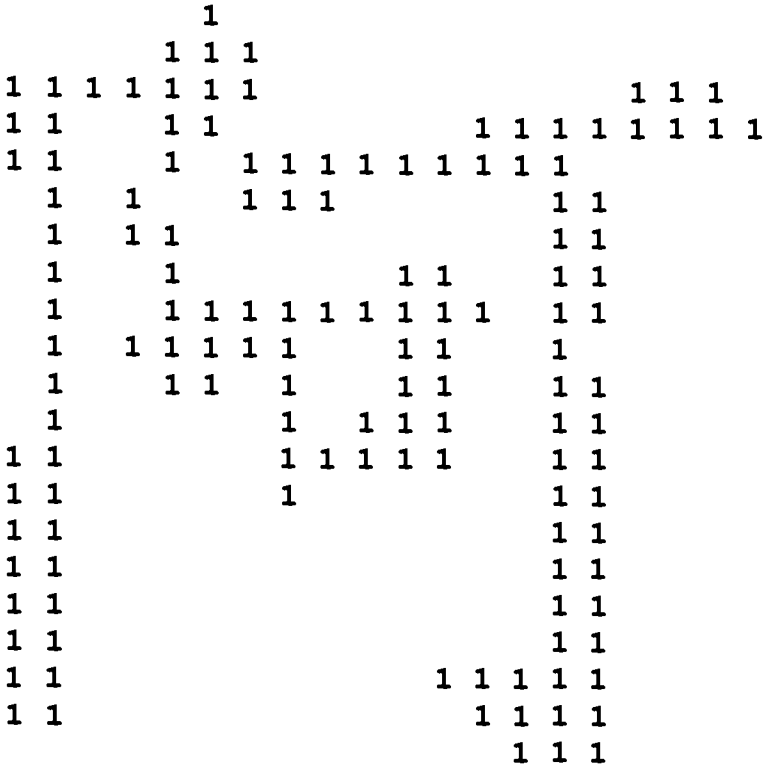
OUT 1

Network State After 1 Iteration(s)



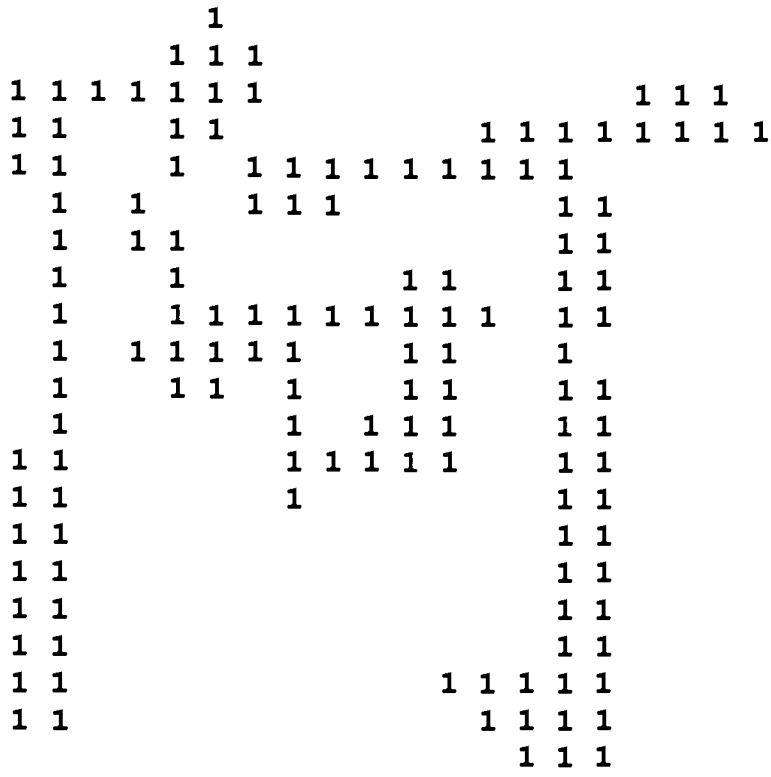
OUT 1

Network State After 2 Iteration(s)



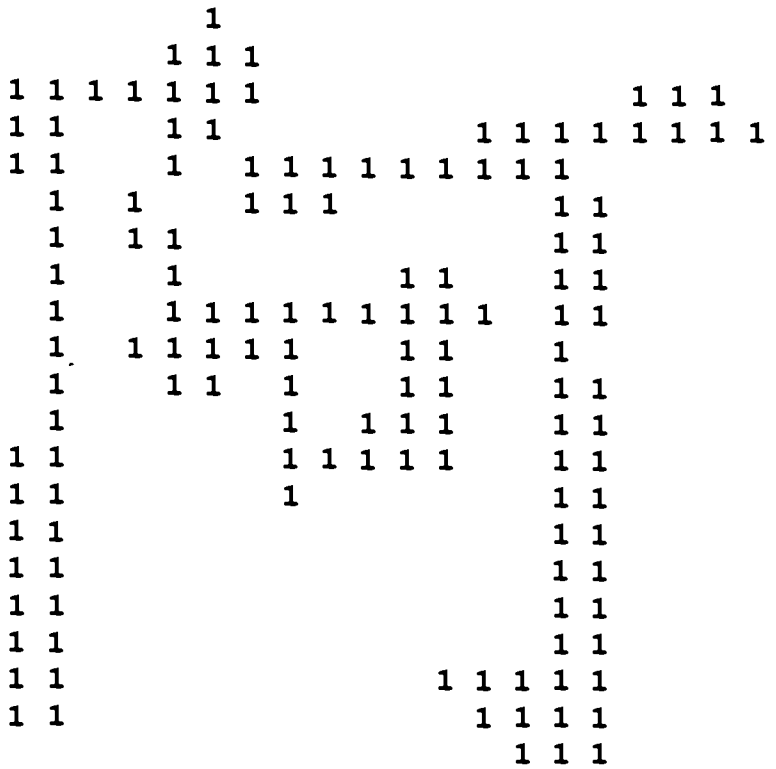
OUT 1

Network State After 3 Iteration(s)



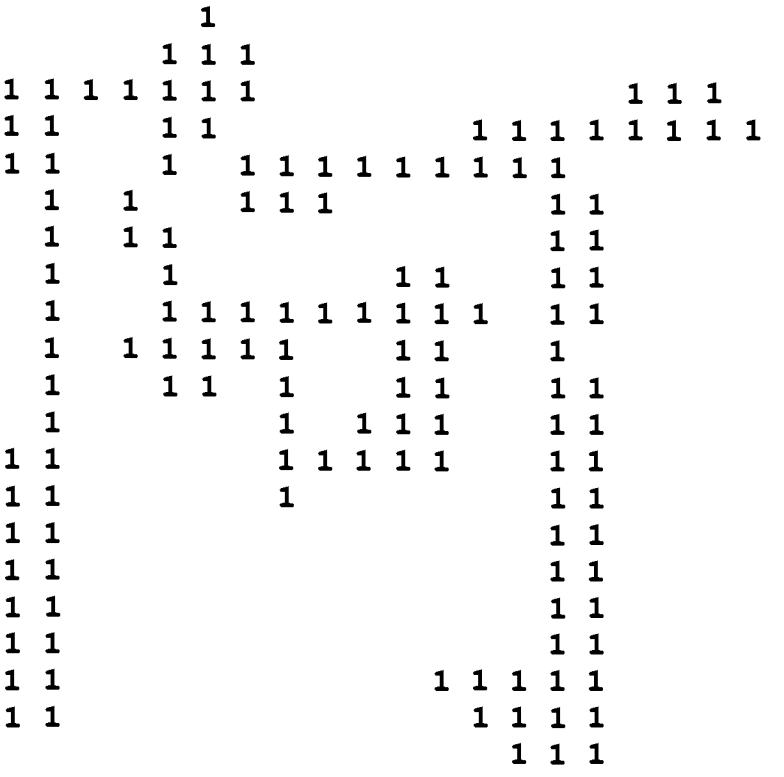
OUT 1

Network State After 4 Iteration(s)



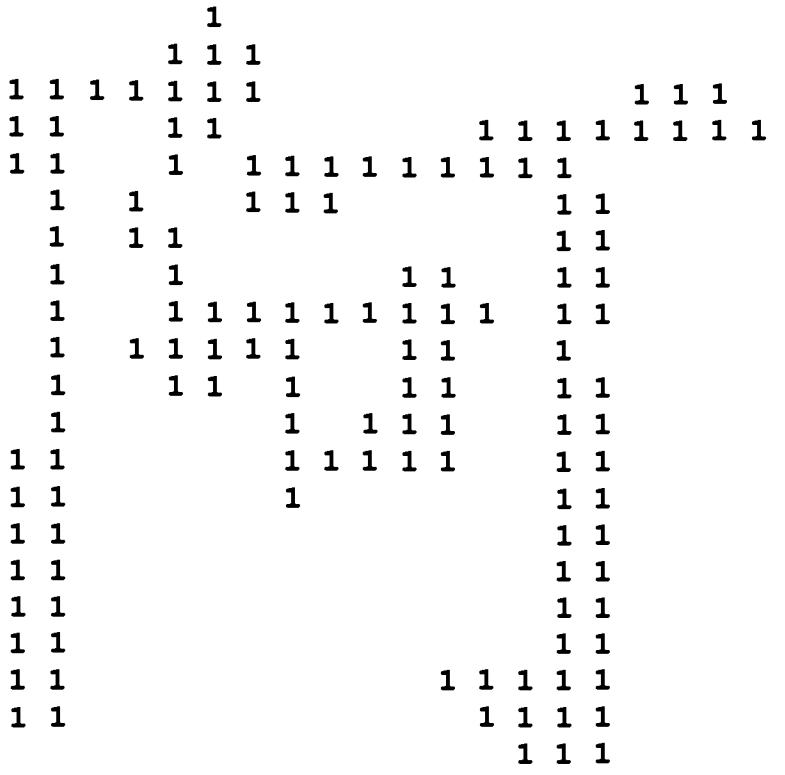
OUT 1

Network State After 5 Iteration(s)



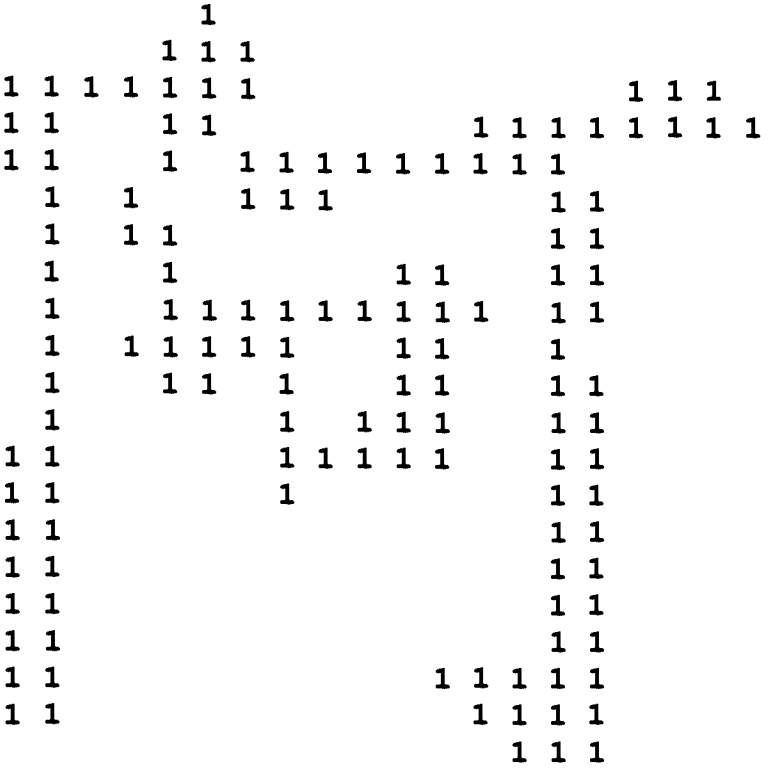
OUT 1

Network State After 6 Iteration(s)

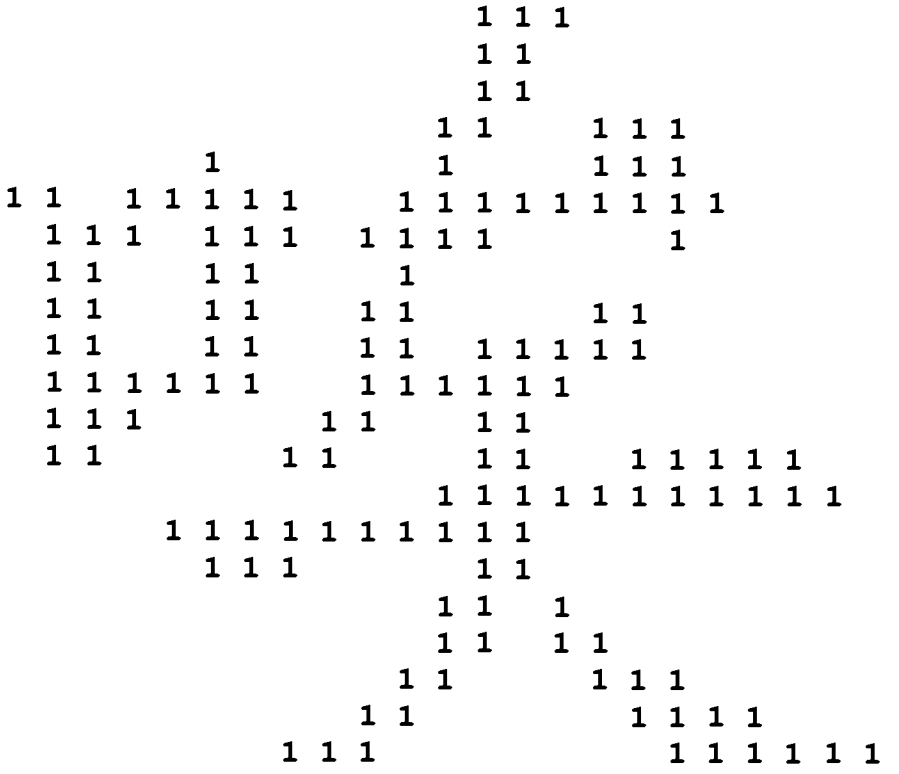


OUT 1

Attractor State

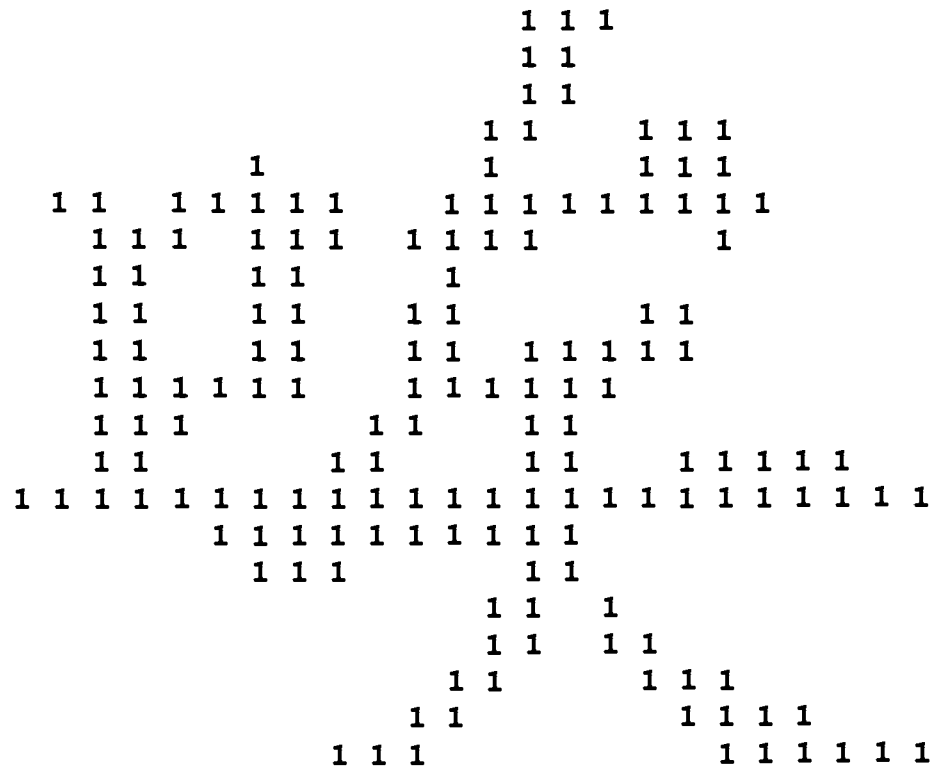


CHINESE 5



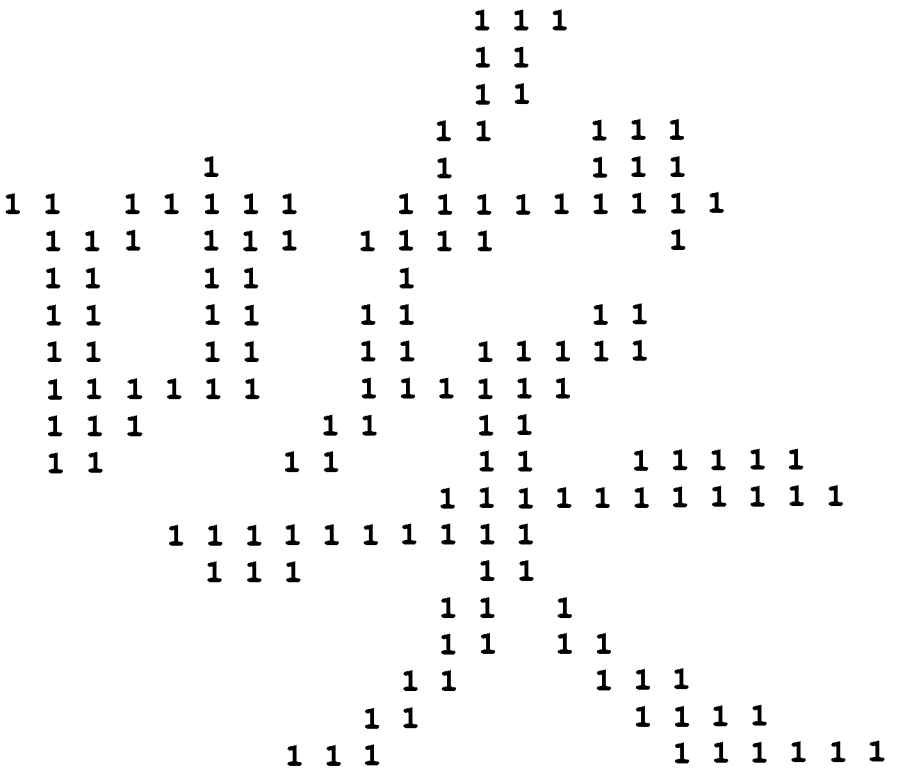
OUT 5

Initial Network State



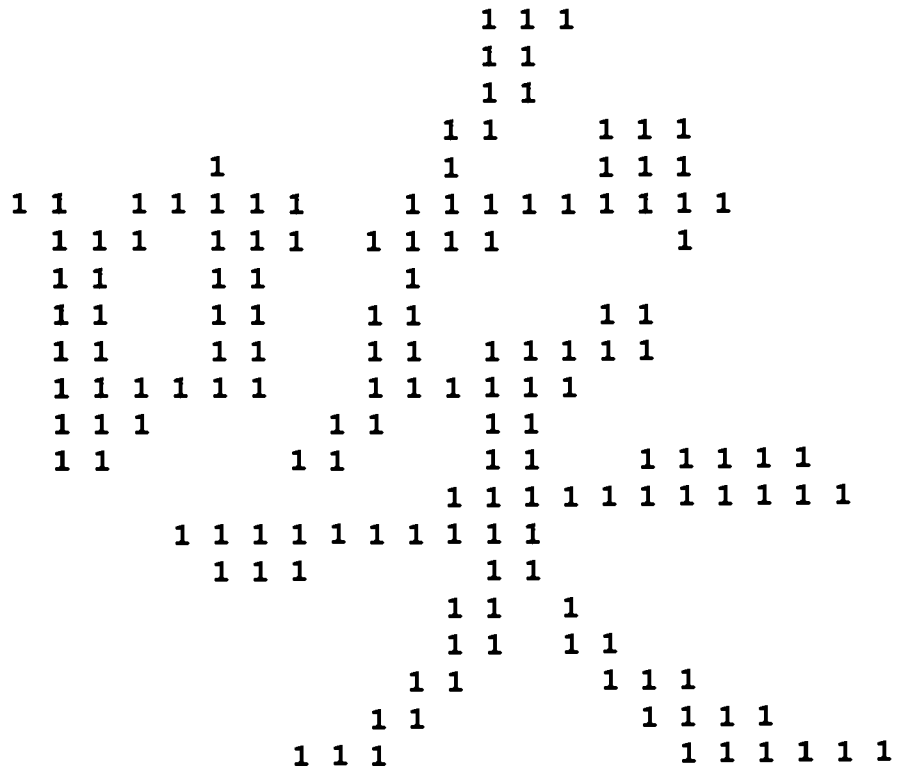
OUT 5

Network State After 1 Iteration(s)



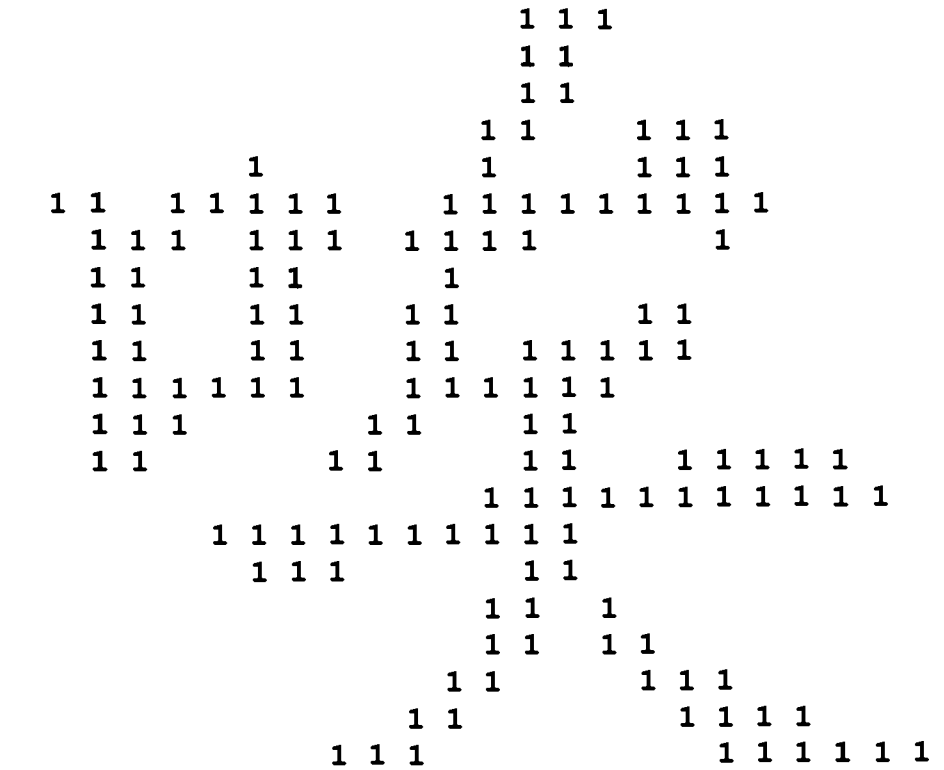
OUT 5

Network State After 2 Iteration(s)



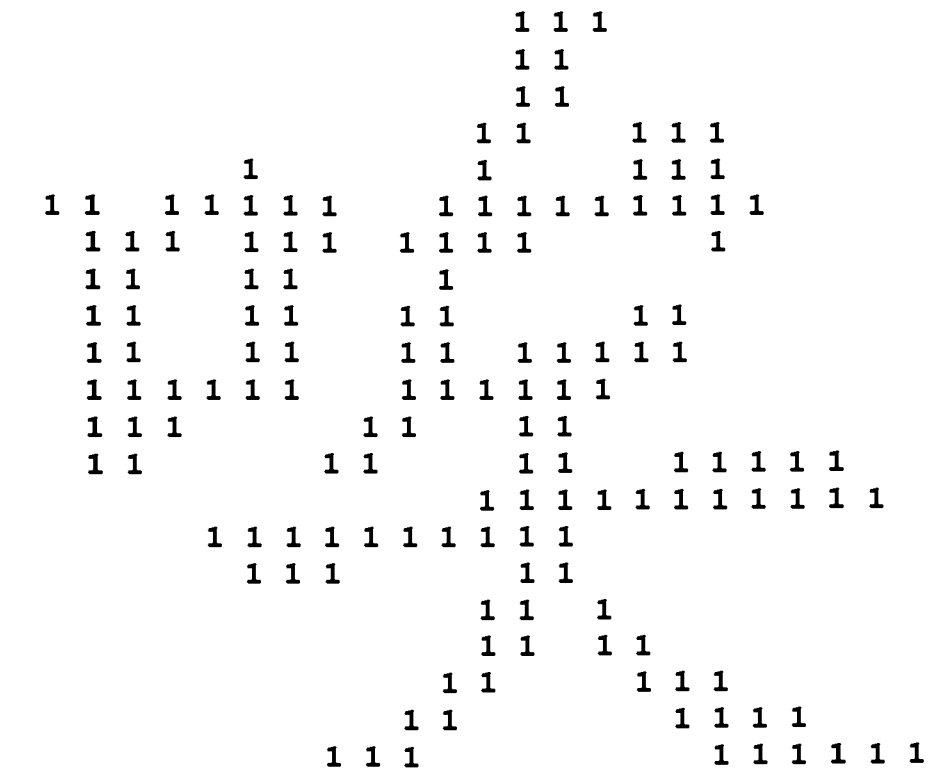
OUT 5

Network State After 3 Iteration(s)



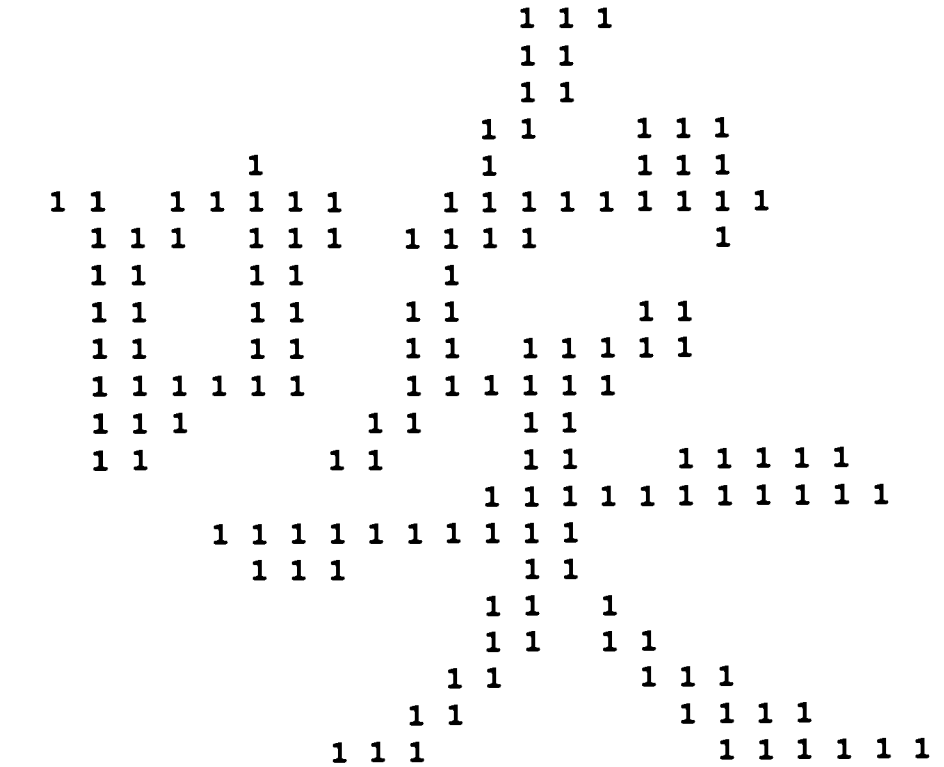
OUT 5

Network State After 4 Iteration(s)



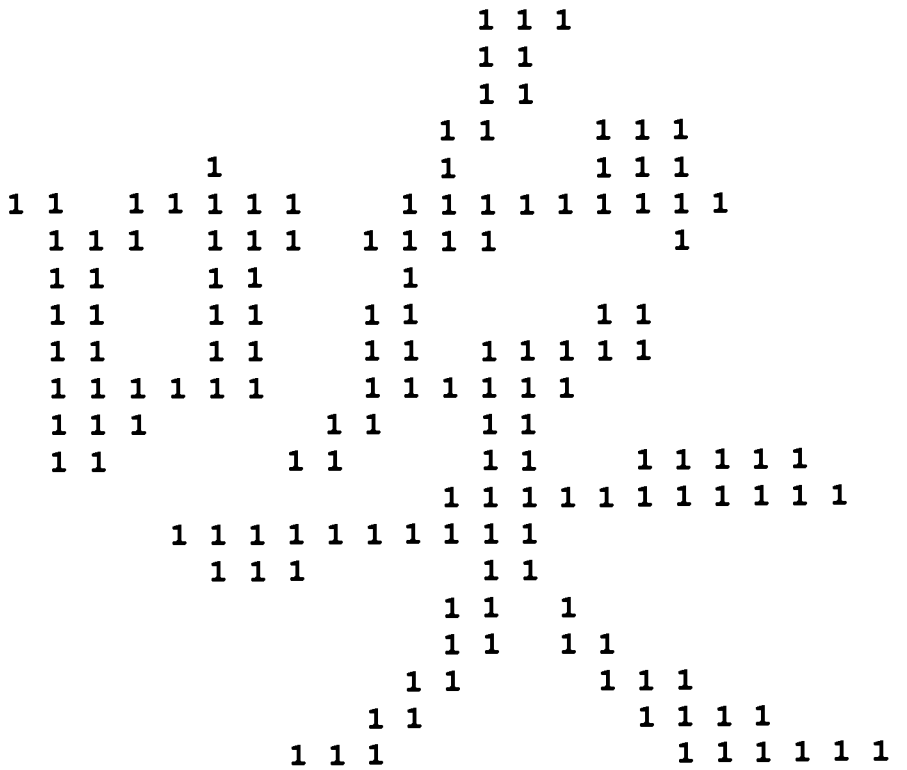
OUT 5

Network State After 5 Iteration(s)



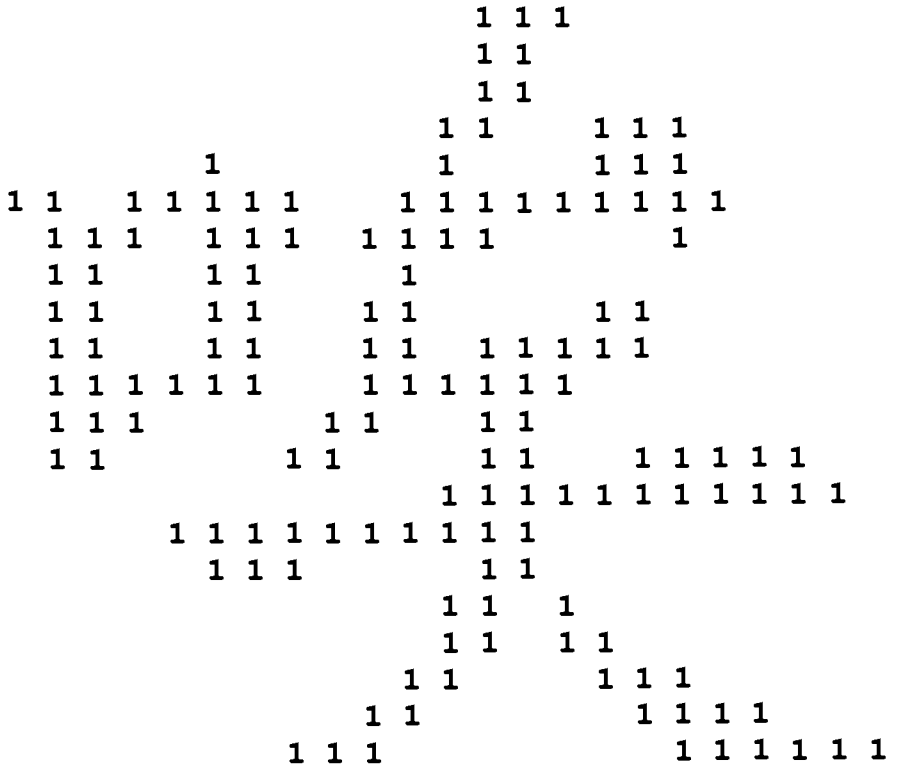
OUT 5

Network State After 6 Iteration(s)

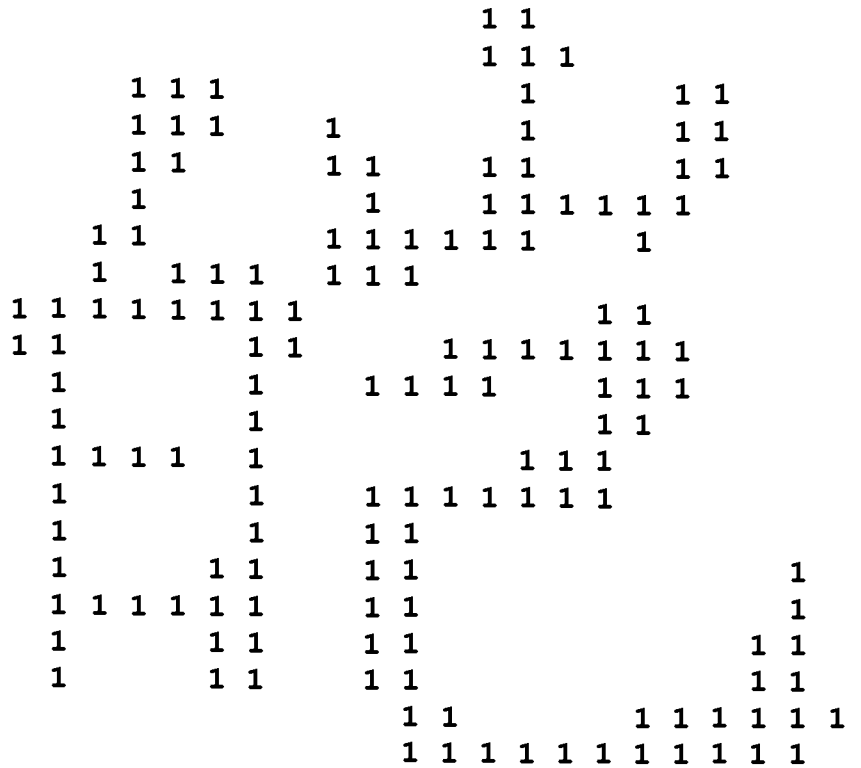


OUT 5

Attractor State

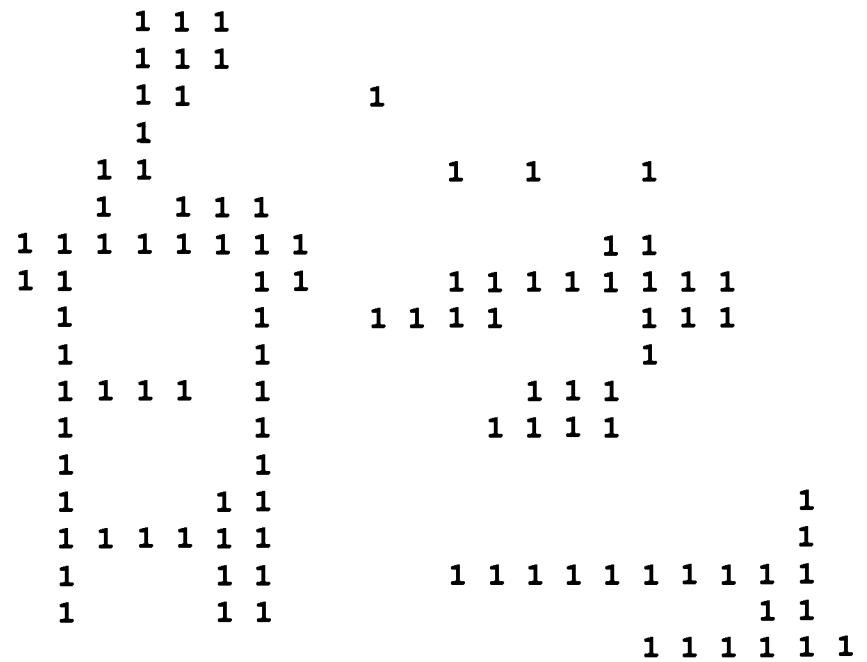


CHINESE 7



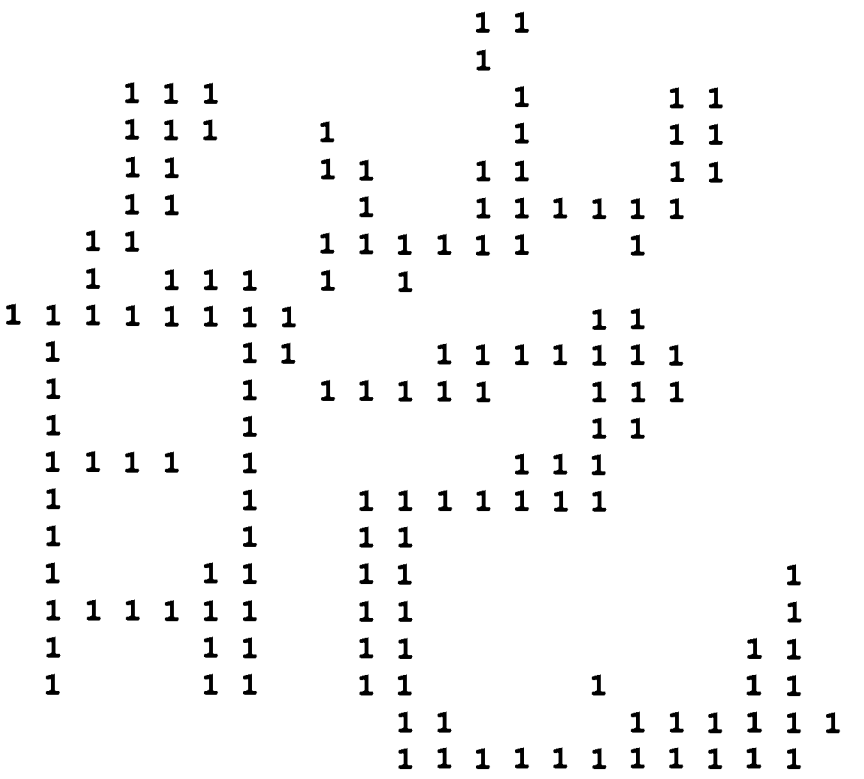
OUT 7

Initial Network State



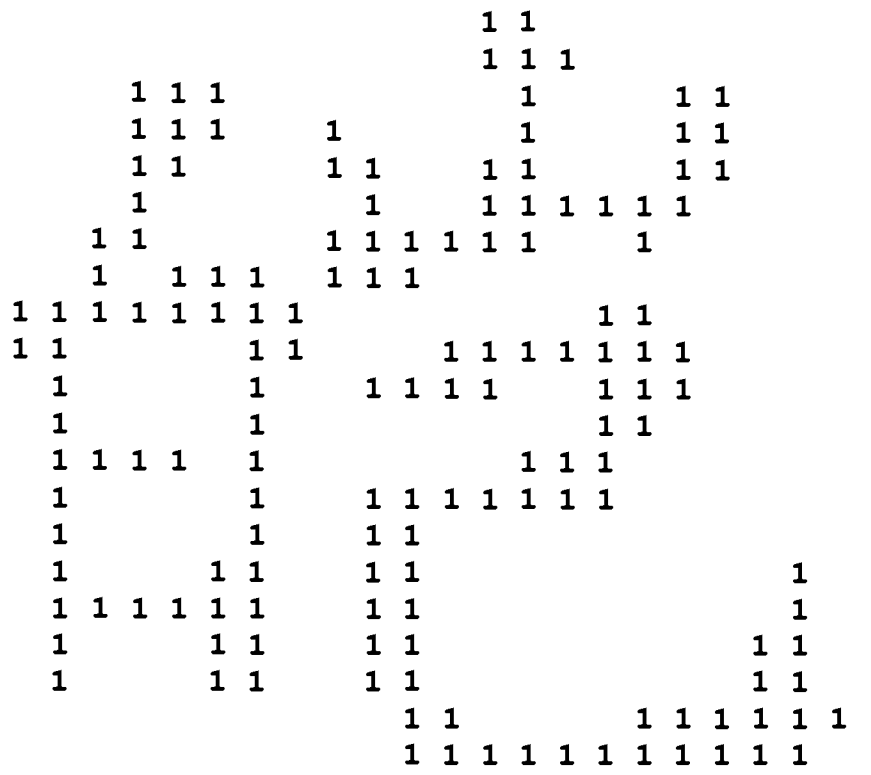
OUT 7

Network State After 1 Iteration(s)



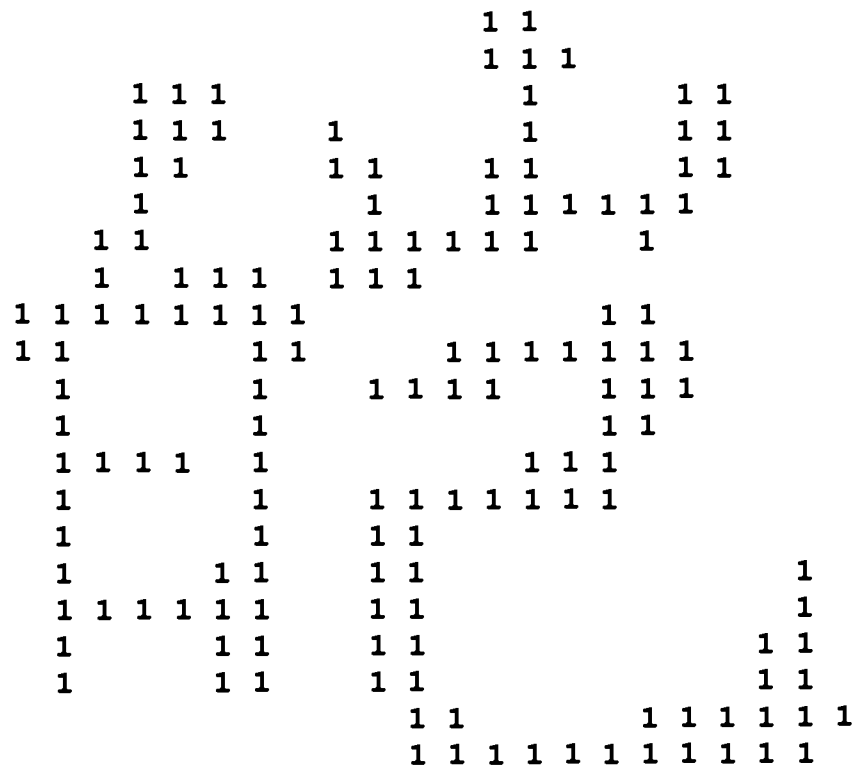
OUT 7

Network State After 2 Iteration(s)



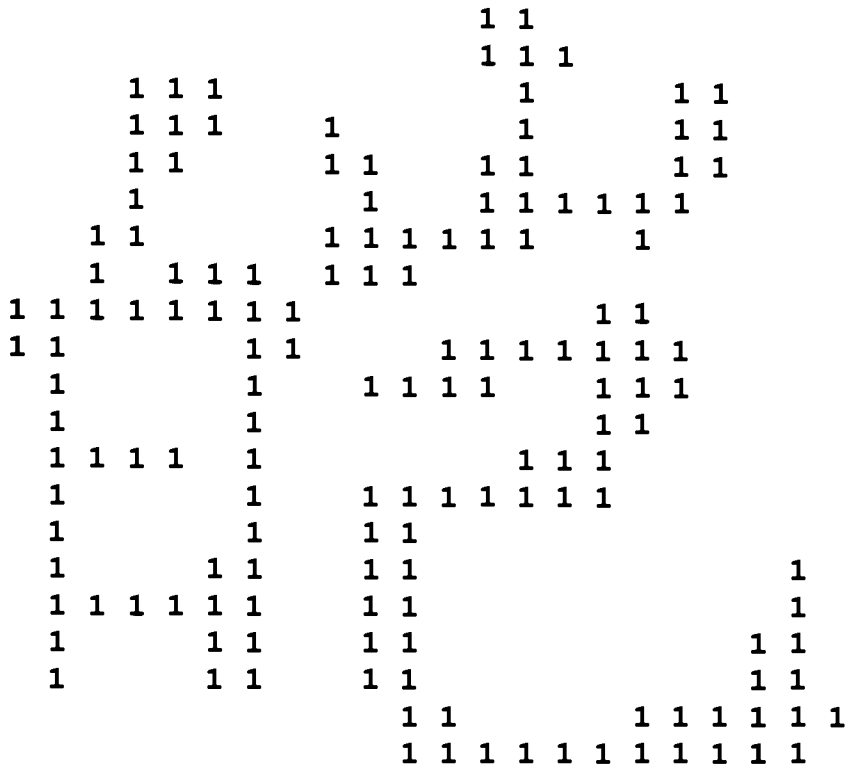
OUT 7

Network State After 3 Iteration(s)



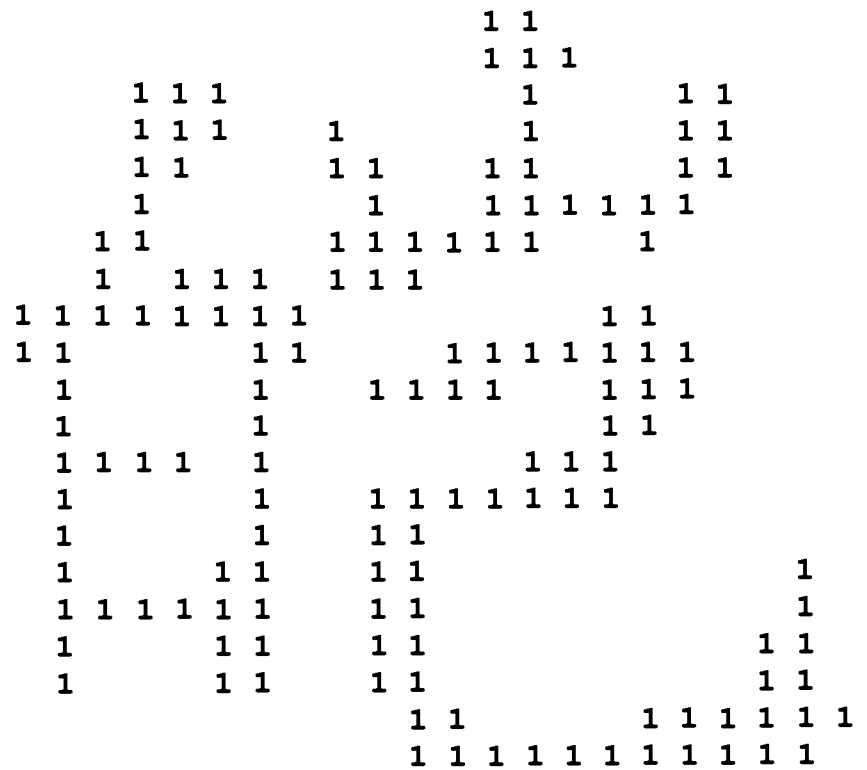
OUT 7

Network State After 4 Iteration(s)



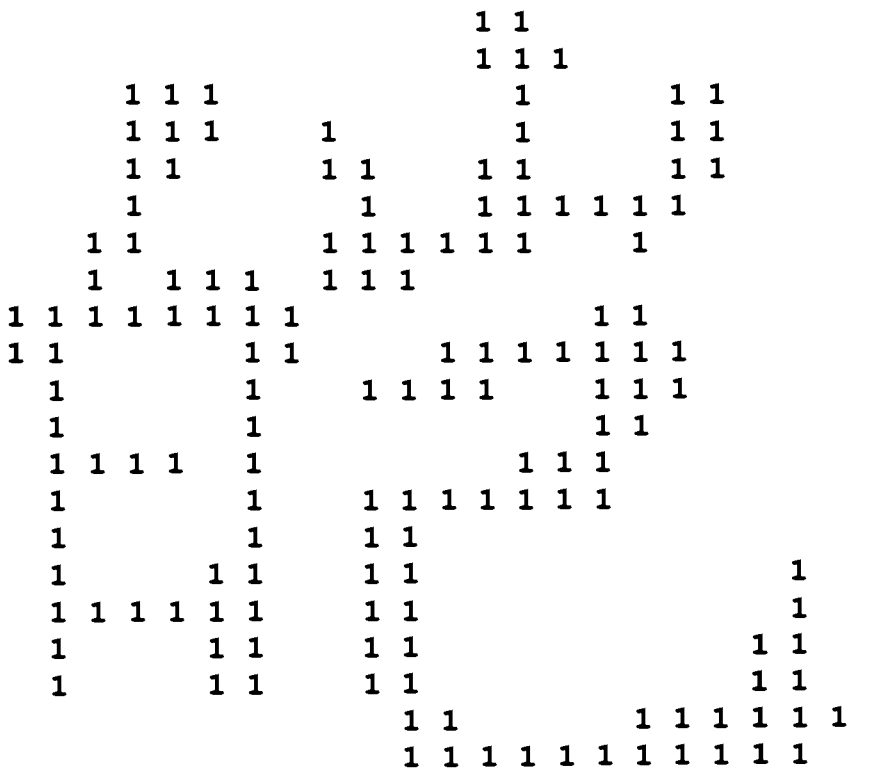
OUT 7

Network State After 5 Iteration(s)



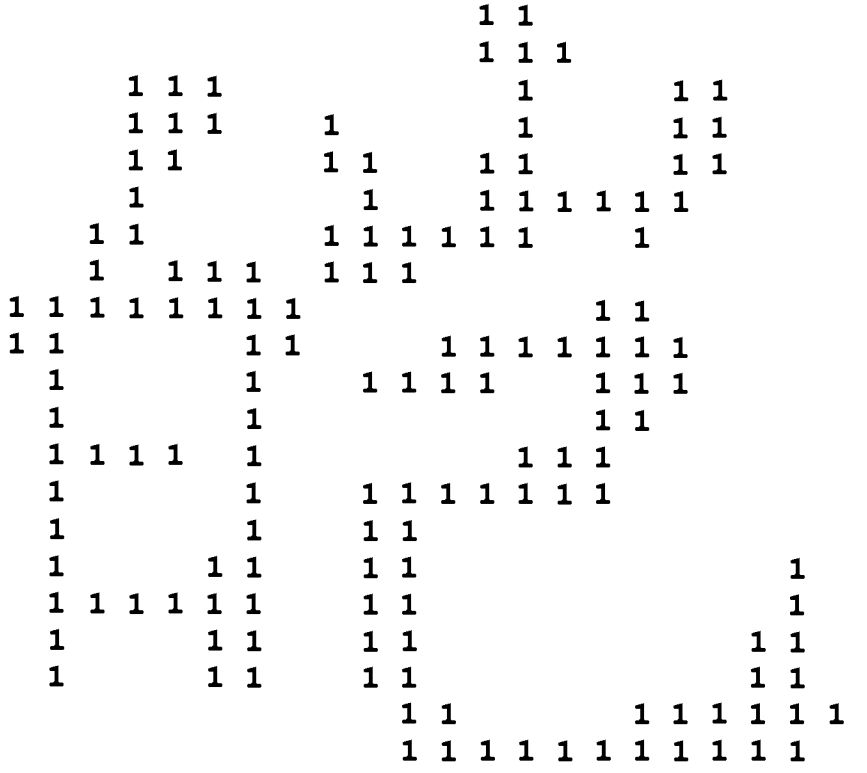
OUT 7

Network State After 6 Iteration(s)

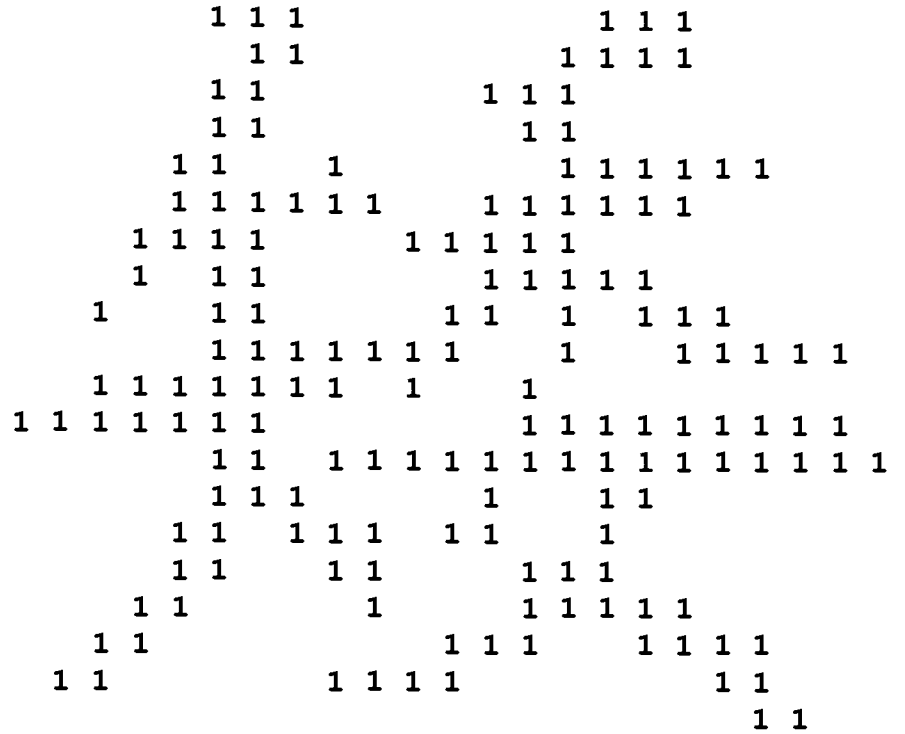


OUT 7

Attractor State

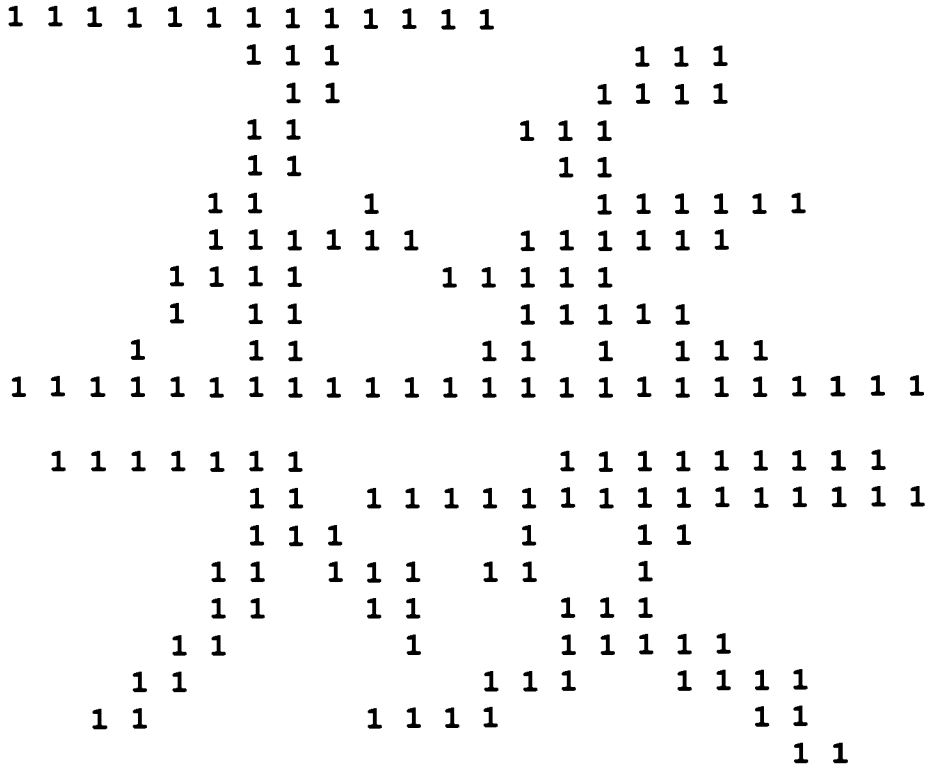


CHINESE 10



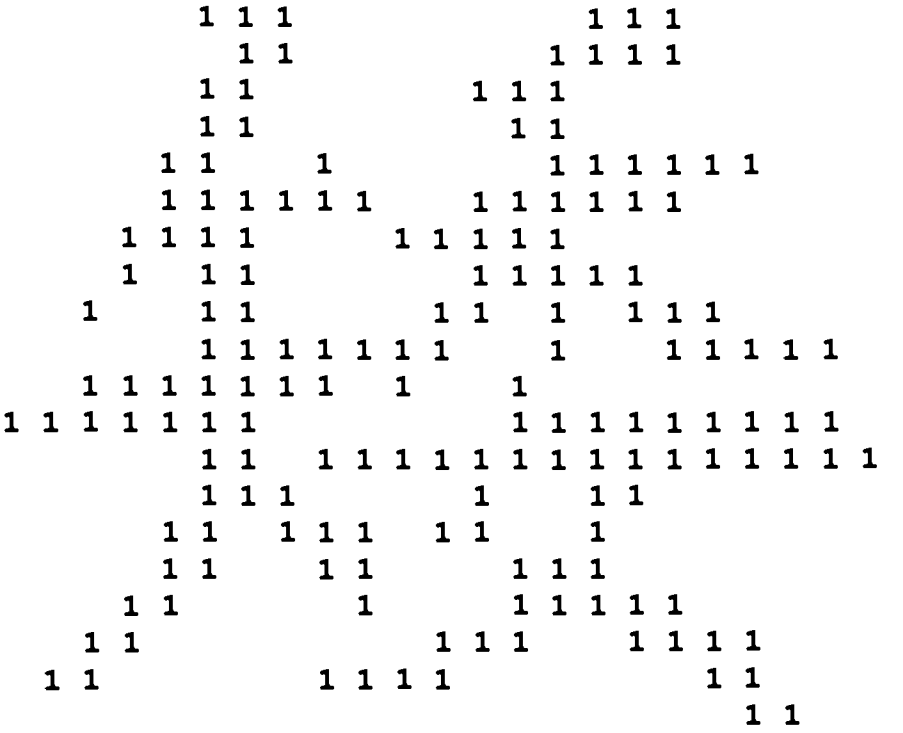
OUT 10

Initial Network State



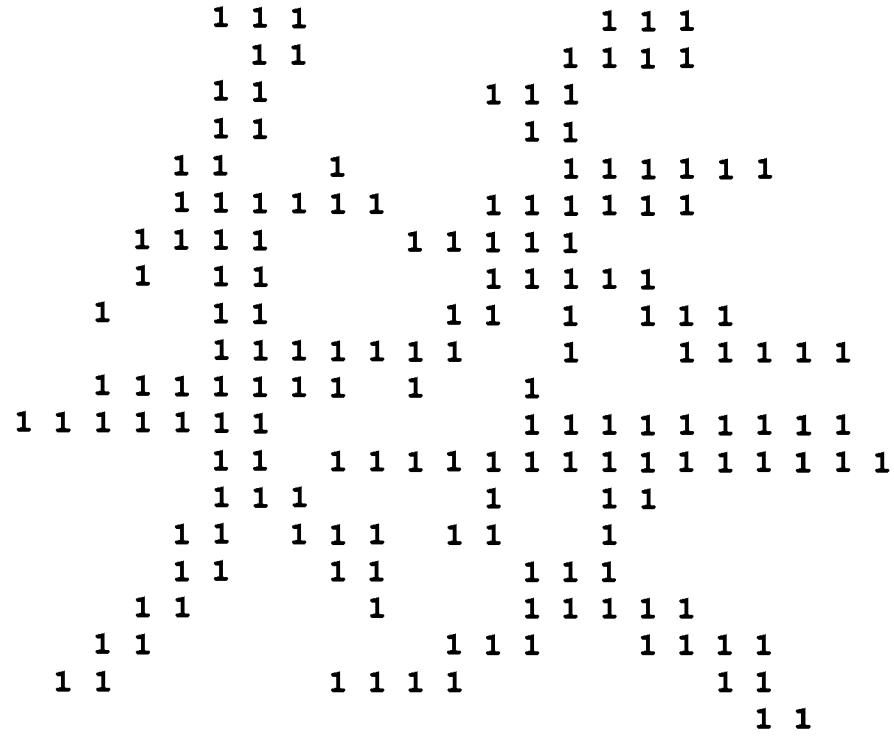
OUT 10

Network State After 1 Iteration(s)



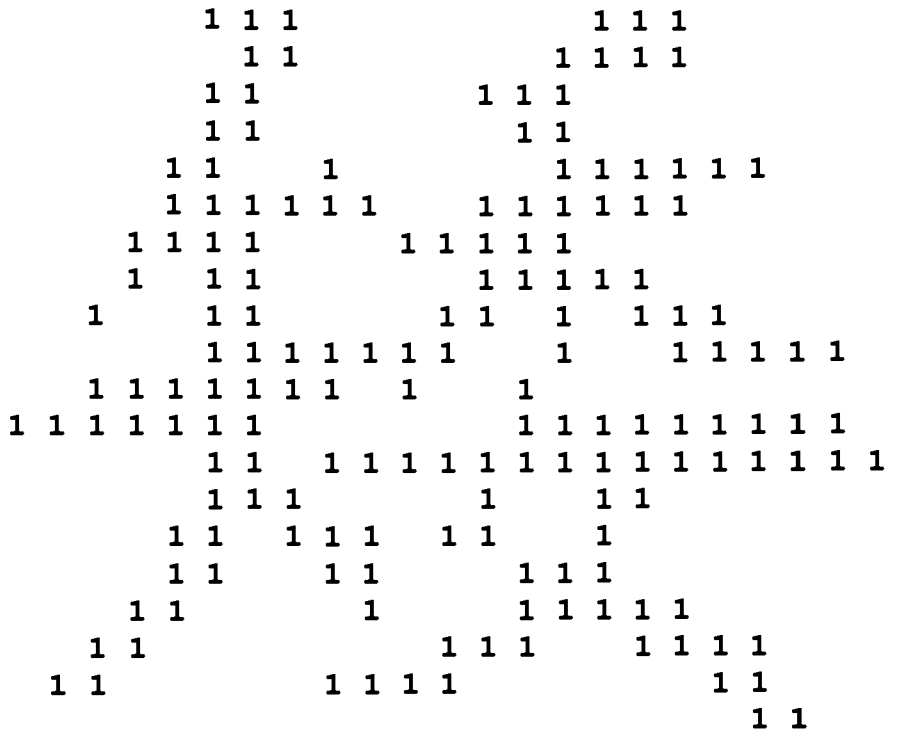
OUT 10

Network State After 2 Iteration(s)



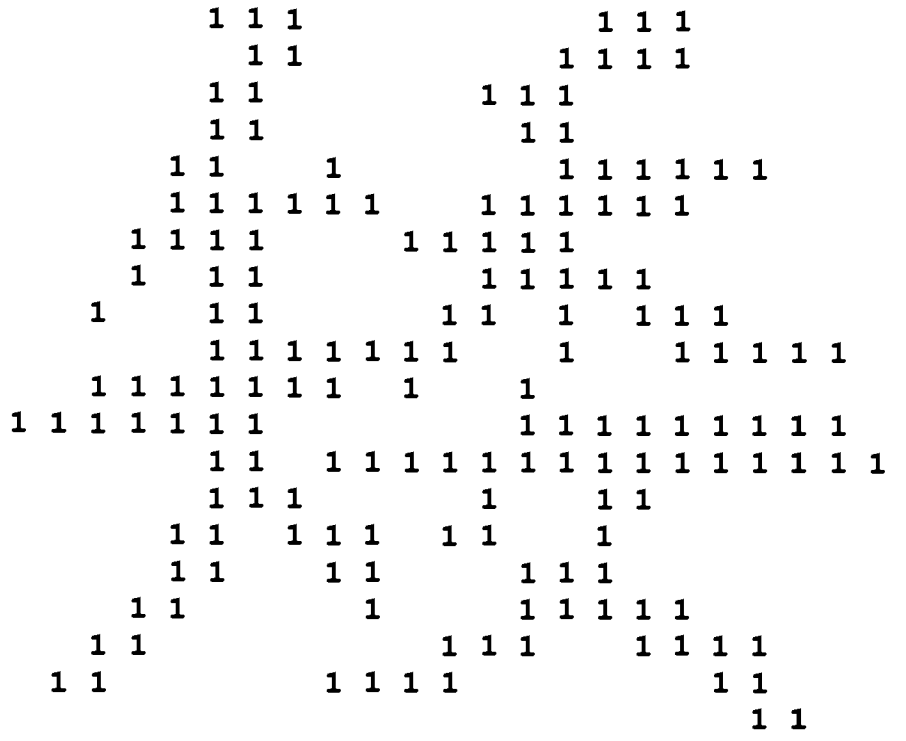
OUT 10

Network State After 3 Iteration(s)



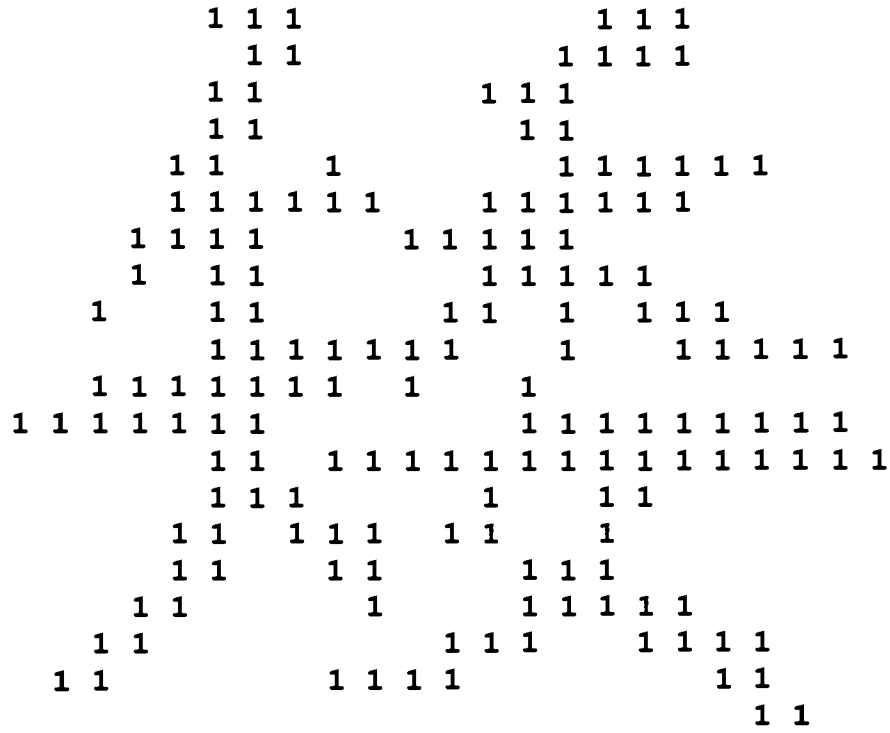
OUT 10

Network State After 4 Iteration(s)



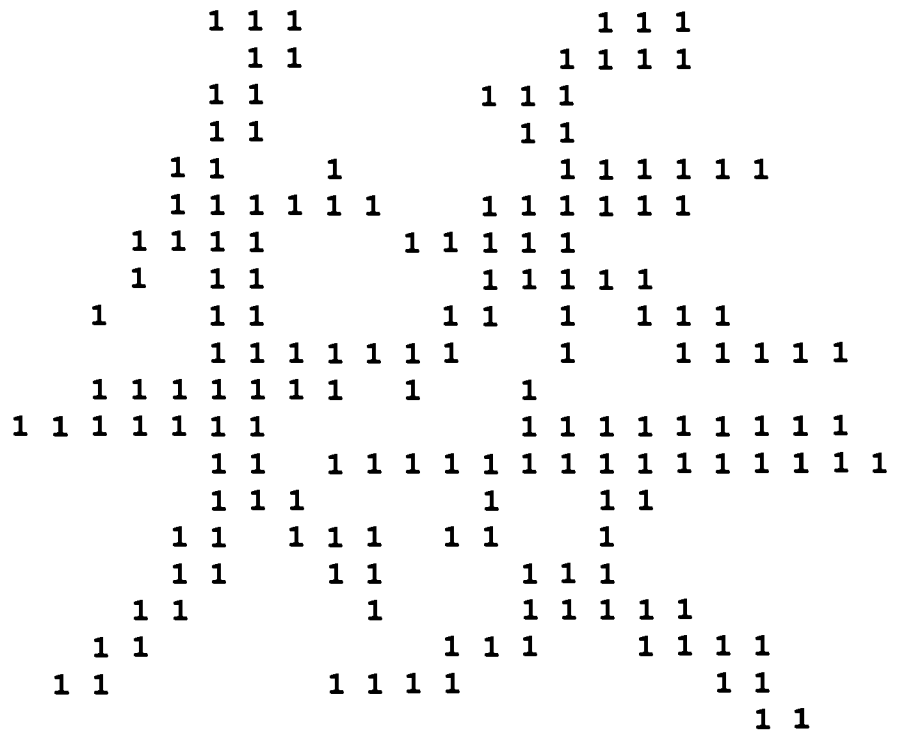
OUT 10

Network State After 5 Iteration(s)



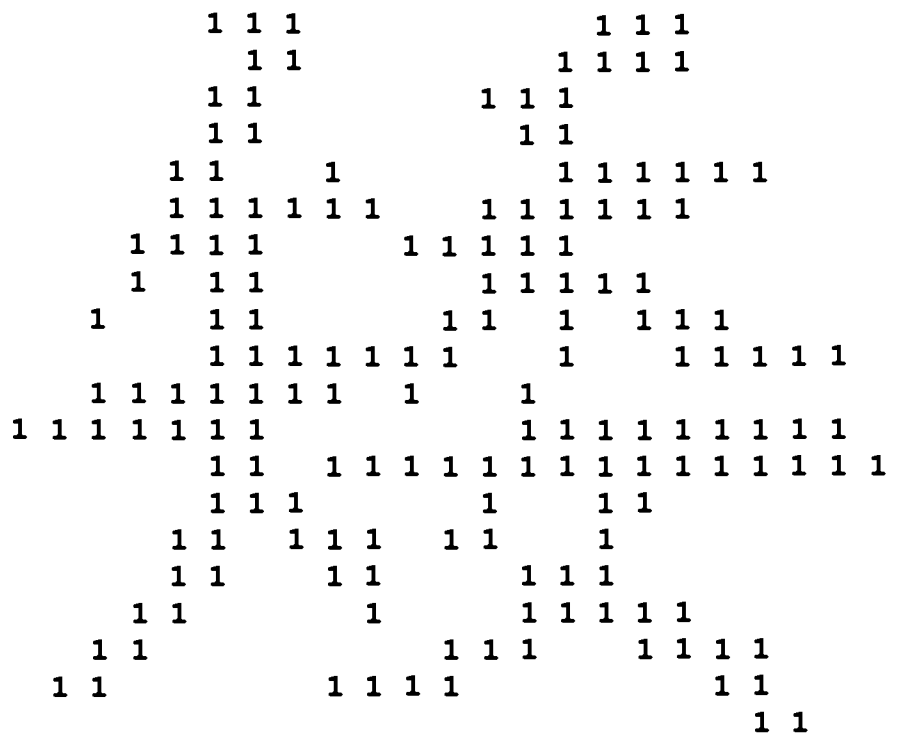
OUT 10

Network State After 6 Iteration(s)

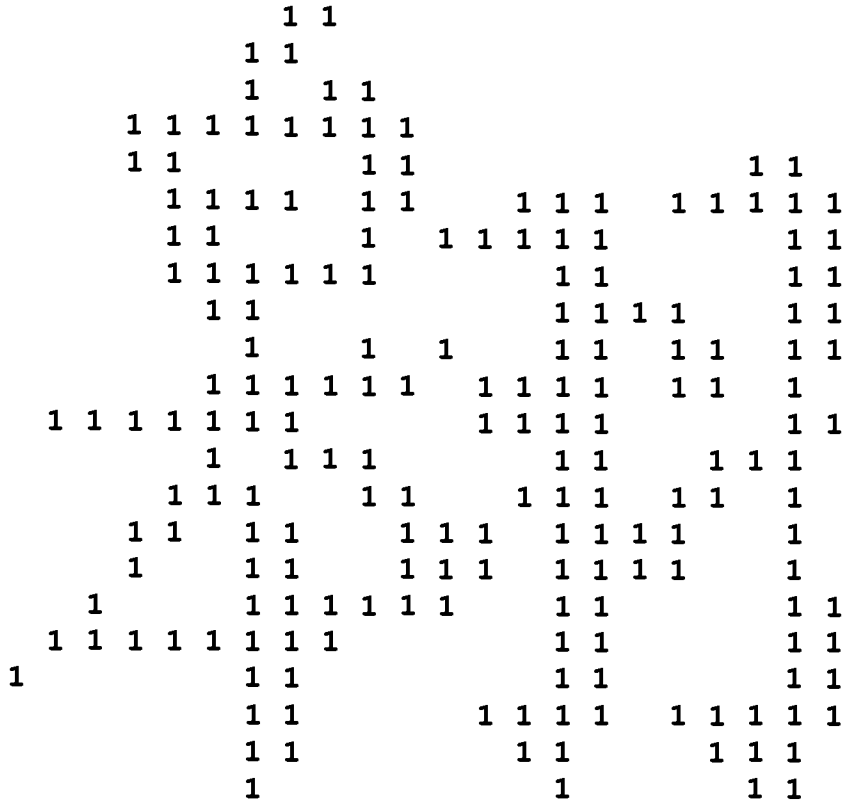


OUT 10

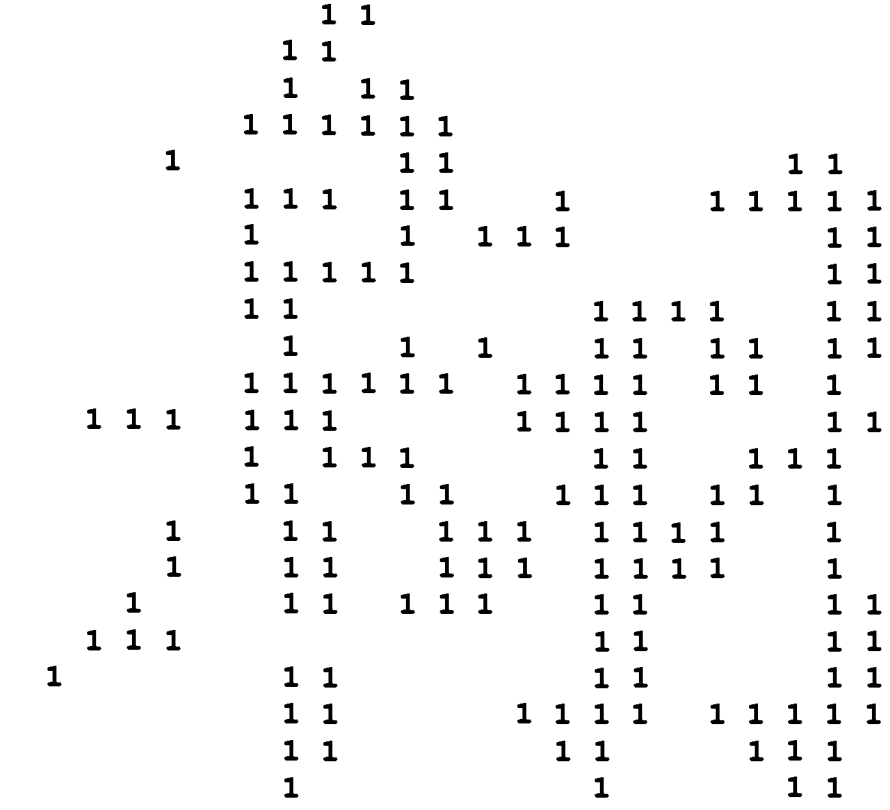
Attractor State



CHINESE 30

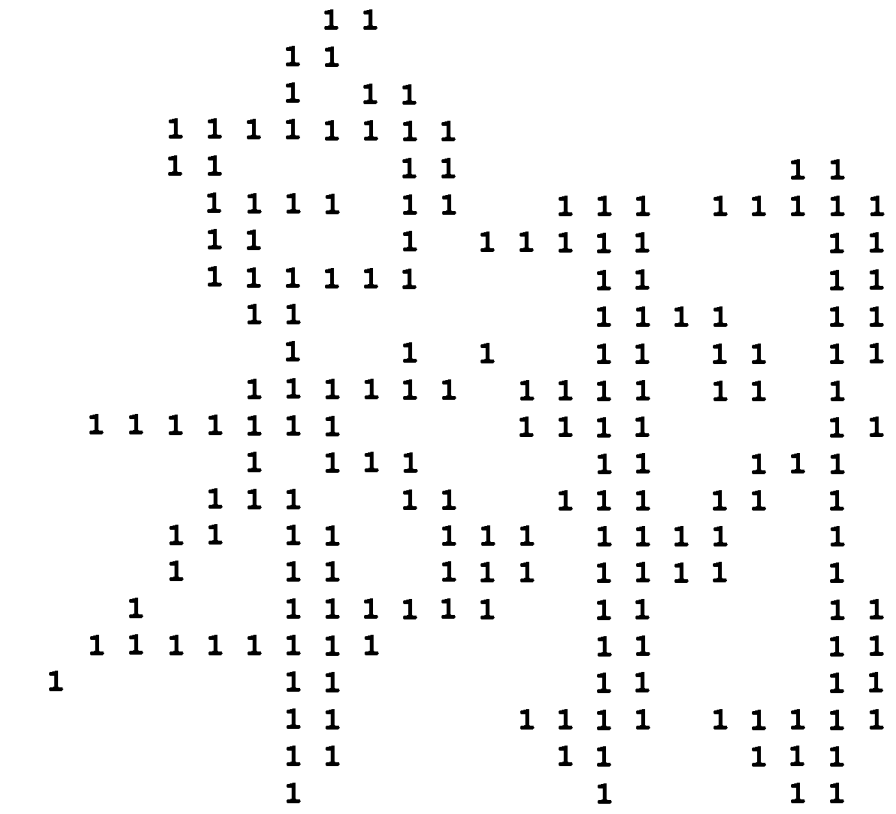


Initial Network State



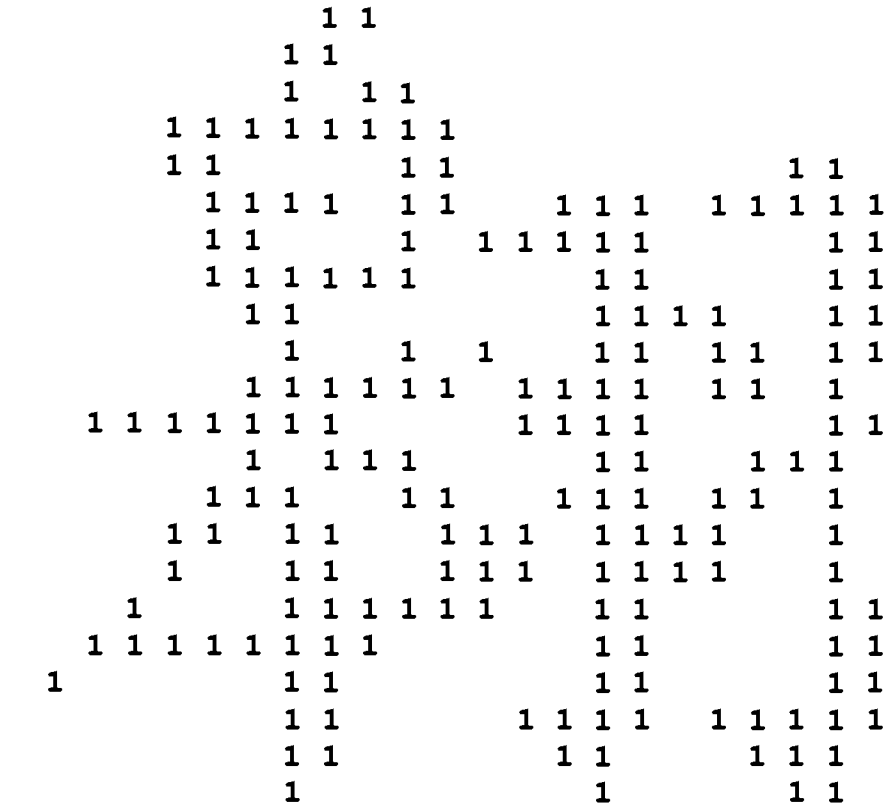
OUT 30

Network State After 1 Iteration(s)



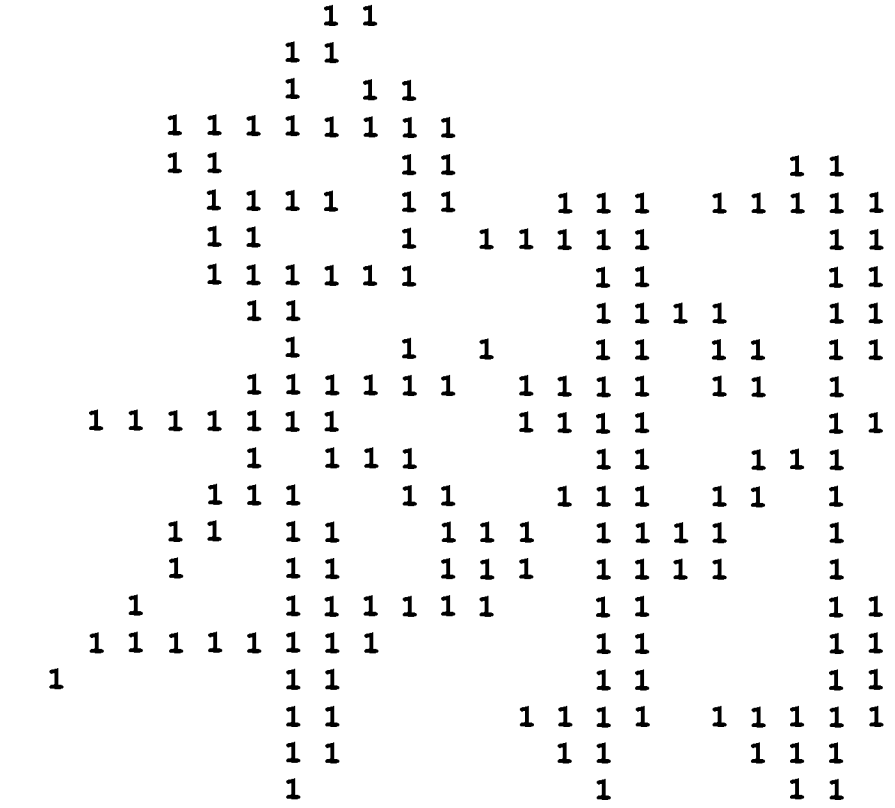
OUT 30

Network State After 2 Iteration(s)



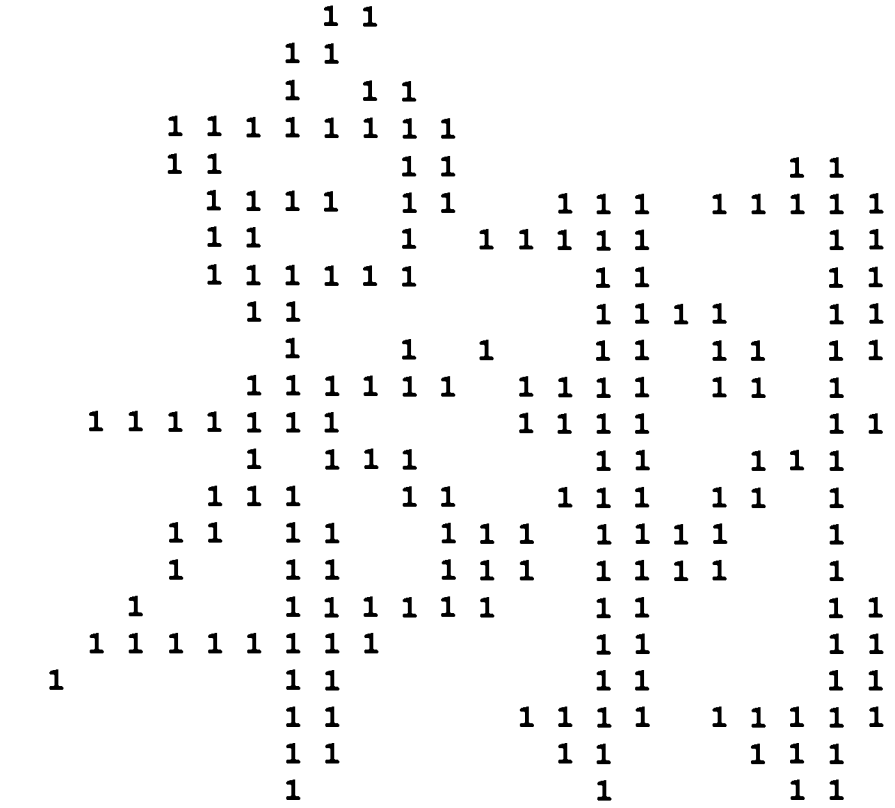
OUT 30

Network State After 3 Iteration(s)



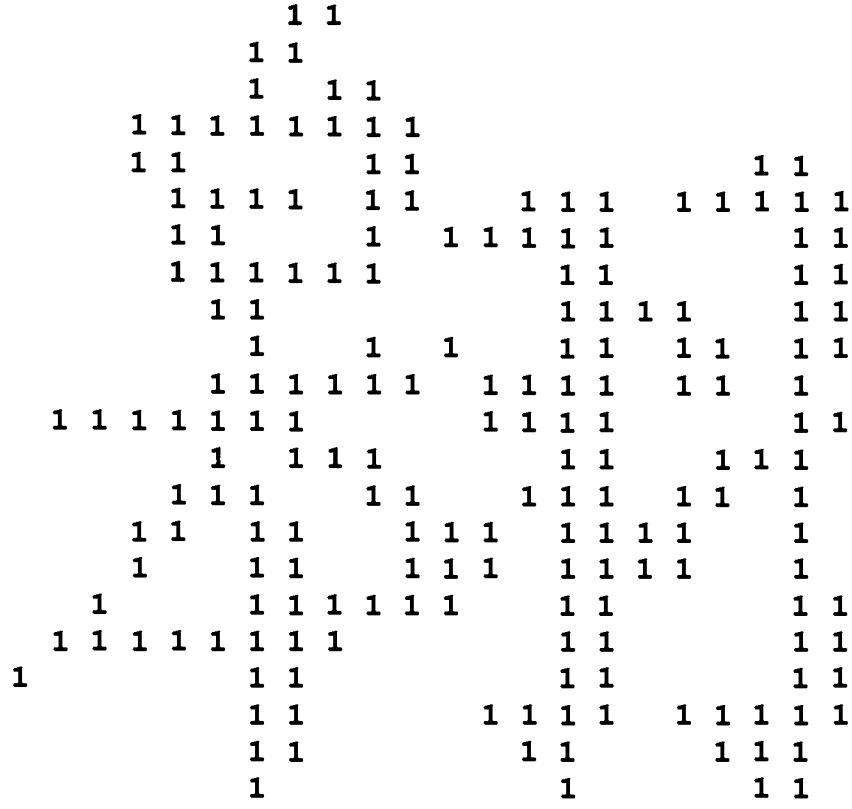
OUT 30

Network State After 4 Iteration(s)



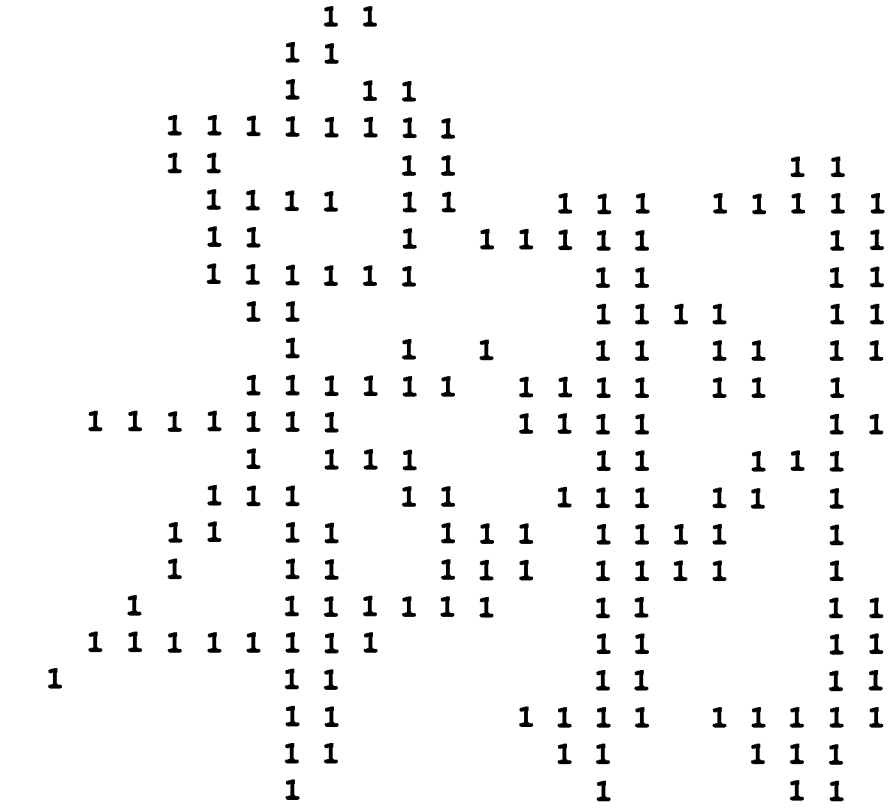
OUT 30

Network State After 5 Iteration(s)



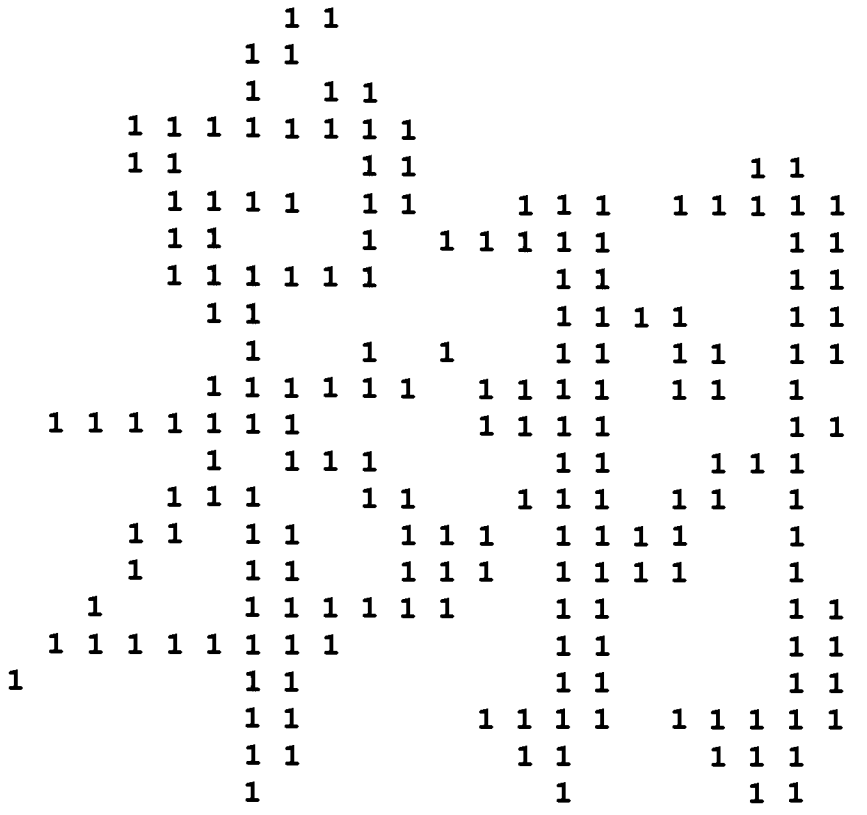
OUT 30

Network State After 6 Iteration(s)



OUT 30

Attractor State



```
#Title:  makefile
#Author: David L. Morabito

CFLAGS = -g

LIBFLAGS = -lm

CFILES1 = dynamics.c update_utils.c init_utils.c net_utils.c replica.c
CFILES2 = pattern.c report.c inverse.c

OBS1 = dynamics.o update_utils.o init_utils.o net_utils.o replica.o
OBS2 = pattern.o report.o inverse.o

inverse.o:
        cc -c $(CFLAGS) inverse.c

report.o:
        cc -c $(CFLAGS) report.c

pattern.o:
        cc -c $(CFLAGS) pattern.c

dynamics.o:
        cc -c $(CFLAGS) dynamics.c

update_utils.o:
        cc -c $(CFLAGS) update_utils.c

init_utils.o:
        cc -c $(CFLAGS) init_utils.c

net_utils.o:
        cc -c $(CFLAGS) net_utils.c

replica.o:
        cc -c $(CFLAGS) replica.c

replica : $(OBS1) $(OBS2)
        cc $(CFLAGS) -o replica $(OBS1) $(OBS2) $(LIBFLAGS)

depend:
        grep '^#include' ${CFILES1} ${CFILES2} | grep -v '<' | \
              sed 's/:[^"]*"\([^"]*\)".*/: \1/' | \
              sed 's/\.c/.o/' | sed 's,../[a-z]*/,,' | \
        awk ' { if ($$1 != prev) { print rec; rec = $$0; prev = $$1; } \
                else { if (length(rec $$2) > 78) { print rec; rec = $$0; } \
                       else rec = rec " " $$2 } } \
              END { print rec } ' > makedep
        echo '$$r makedep' >>eddep
        echo '/^# DO NOT DELETE THIS LINE/+1,$$d' >eddep
        echo '$$r makedep' >>eddep
        echo 'w' >>eddep
        cp makefile makefile.bak
        ed - makefile < eddep
        rm eddep makedep

# DO NOT DELETE THIS LINE - make depend uses it

dynamics.o: random.h replica.h
update_utils.o: replica.h
init_utils.o: replica.h
net_utils.o: replica.h attractor.h
replica.o: replica.h
pattern.o: replica.h
report.o: replica.h
inverse.o: replica.h
```

```
/* Title: replica.h
   Author: David L. Morabito

   Purpose: Define constants used to determine network size and the
            structure of the network itself.
*/

#define N1            24 /* Number of rows in neuron matrix */
#define N2            24 /* Number of columns in neuron matrix */
#define NUM_NEURONS   N1 * N2
#define NUM_PATTERNS 30
#define DYNAMICS      1  /* 1 for synchronous and 0 for asynchronous updating */
#define UP            1
#define DOWN         -1
#define SEED          2  /* Initial value for the seed of rand() */
#define RETRIEVAL_LEVEL 0.97 /* Percentage of neurons that must match for
                                two consecutive states to be considered the
                                same.
                             */

enum boolean {true, false};

/* Define network structure */
typedef struct
{

   int    spins[NUM_NEURONS];
   double synapses[NUM_NEURONS][NUM_NEURONS];

} net;

/* Define linked list structure to hold patterns */
struct patterns
{

   int pattern[NUM_NEURONS];
   struct patterns *next;

};

typedef struct patterns patterns;

/* Declare that pattern list will be global */
extern char *pattern_names[NUM_PATTERNS];
```

```
/* Title: replica.c
   Author: David L. Morabito

   Purpose: Main program to run the network simulation.
*/

#include <stdio.h>
#include "replica.h"

char *pattern_names[NUM_PATTERNS];

main(argc, argv)
int argc;
char *argv[];
{

   enum boolean attractor = false;
   net *network_ptr;
   int i;
   float **Cinv;
   double temperature = 2.0;
   patterns *pattern_list = NULL;
   enum boolean check_for_attractor();
   void update(), report_attractor(), print_state();
   patterns *initialize_network();
   float **matrix();
   int iteration = 0;

   /* Read in the pattern names from the parameter list. */
   for (i = 1; i < argc; ++i)
      pattern_names[i] = argv[i];

   /* Allocate memory for the network */
   network_ptr = (net *)malloc(sizeof(net));

   /* Allocate memory for the inverse of the overlap matrix/ */
   Cinv = matrix(1,NUM_PATTERNS,1,NUM_PATTERNS);

   /* Initial network and get input pattern list. */
   pattern_list = initialize_network(network_ptr, pattern_list, Cinv);

   printf("Initial Network State0);
   print_state(network_ptr);

   while (attractor != true)
   {
      /* Update the network at specified temperature. */
      update(network_ptr,temperature);

      ++iteration;
      printf("Network State After %d Iteration(s)0, iteration);
      print_state(network_ptr);
```

```
      /* Check to see if the network is in an attractor state */
      attractor = check_for_attractor(network_ptr);
   }

   /* Determine what the attractor state is and print out the corresponding
      pattern.
   */
   printf("Attractor State0);
   report_attractor(network_ptr, pattern_list, Cinv);

}
```

```
/* Title: attractor.h
   Author: David L. Morabito

   Purpose: To define the parameter that will determine the number of
            times that a network must stay in a state for that state
            to be considered an attractor.
*/

#define CYCLES 5
```

```
/* Title: net_utils.c
   Author: David L. Morabito

   Purpose: These are routines that initialize, check and print out
            information about the network state.
*/

#include <stdio.h>
#include "replica.h"
#include "attractor.h"

/* Initialize the network (i.e. get initial neuron values) and learn the stored
   patterns (i.e. calculate the synapses).
*/
patterns *initialize_network(network_ptr, pattern_list, Cinv)
net *network_ptr;
patterns *pattern_list;
float **Cinv;
{

   void init_spins();
   patterns *learn();
   int i;

   init_spins(network_ptr -> spins);
   pattern_list = learn(network_ptr -> synapses, pattern_list, Cinv);

   return pattern_list;

}

/* Check to see if the network has reached an attractor state.  Must
   be the same state for 5 comparisions.
*/
enum boolean check_for_attractor(network)
net *network;
{

   int i, num_neurons = NUM_NEURONS;
   static number_calls = 0, match = 0;
   double count = 0;
   enum boolean first_state = false, is_attractor = false;
   static net *old_net = NULL;

   if (number_calls == 0)
   {
      first_state = true;
      number_calls = 1;
   }

   if (first_state == true)
   {
      is_attractor == false;
      if ((old_net = (net *)malloc(sizeof(net))) == NULL)
         printf("NO SPACE0);
      else
         *old_net = *network;
   }
   else
   {
      for (i = 0; i < NUM_NEURONS; ++i)
```

```
      {
         if (network -> spins[i] == old_net -> spins[i])
            count = count + 1;
      }
      free(old_net);
      if ((old_net = (net *)malloc(sizeof(net))) == NULL)
         printf("NO SPACE0);
      else
         *old_net = *network;
   }
   if (count / num_neurons >= RETRIEVAL_LEVEL)
      match = match + 1;
   if (match >= CYCLES)
      is_attractor = true;

   return is_attractor;

}

/* Calculate the new network state using either synchronous or asynchronous
   dynamics.
*/
void update(network_ptr, temperature)
net *network_ptr;
double temperature;
{

   void synchronous_dynamics(), asynchronous_dynamics();

   if (DYNAMICS == 1)
      synchronous_dynamics(network_ptr, temperature);
   else
      asynchronous_dynamics(network_ptr, temperature);

}

/* Once the network has reached an attractor print out the attractor
   pattern.
*/
void report_attractor(network_ptr, pattern_list, Cinv)
net *network_ptr;
patterns *pattern_list;
float **Cinv;
{

   int i, pattern_num;
   double overlaps[NUM_PATTERNS];
   double order_parameters[NUM_PATTERNS];
   int find_corr_pattern();
   double calculate_overlaps();
   double calculate_order_parameters();
   void print_pattern();
   patterns *tmp_ptr;

   tmp_ptr = pattern_list;
   for (i = 0; i < NUM_PATTERNS; ++i)
   {
      overlaps[i] = calculate_overlaps(network_ptr, pattern_list);
      pattern_list = pattern_list -> next;
   }
   for (i = 0; i < NUM_PATTERNS; ++i)
```

```
   {
      /* order parameters specify the overlap with the stored orthogonal
         patterns not the original input patterns.
      */
      order_parameters[i] =  calculate_order_parameters(overlaps, i, Cinv);
   }

   /* Look for stored pattern with highest overlap value. */
   pattern_num = find_corr_pattern(order_parameters) + 1;

   for (i = 0; i < pattern_num - 1; ++i)
      tmp_ptr = tmp_ptr -> next;
   print_pattern(tmp_ptr);

}

/* Print out the pattern that corresponds to the attractor that the network
   is in.
*/
void print_state(network_ptr)
net *network_ptr;
{
   int j, count = 0, values_per_line;

   values_per_line = N1;
   for (j = 0; j < NUM_NEURONS; ++j)
   {
      if (count == values_per_line)
      {
         printf("0);
         count = 0;
      }
      if (network_ptr->spins[j] != -1)
         printf("%d ", network_ptr->spins[j]);
      else
         printf("  ");
      count = count + 1;
   }
   printf("0);

}
```

```
/* Title: random.h
   Author: David L. Morabito

   Purpose: To define the maximum random number generated by the rand()
            function.  Random numbers are needed for asynchronous
            random updating of the network.
*/

#define RAND_MAX  2147483647
```

```
/* Title: Dynamics.c
   Author: David L. Morabito

   Purpose: These routines are used to determine the updating of each
            individual neuron in the network.
*/

#include "random.h"
#include "replica.h"

/* Calculate the PSP at a particular position in the network. */
double calculate_local_field(pos, network)
int pos;
net *network;
{

   int j;
   double sum = 0.0;

   for (j = 0; j < NUM_NEURONS;  j++)
   {
      /* Can't include the effect of the neuron itself */
      if (j != pos)
         sum = sum + network -> synapses[pos][j] * network -> spins[j];
   }
   return sum;

}

/* Determine if the state of the neuron should be changed. */
double probability(T, local_field, spin)
double T;
double local_field;
int spin;
{

   double tanh();

   return (1.0 / 2.0) * (1 - spin * tanh((1/T) * local_field));

}

/* When doing random asynchronous updating determine which neuron should
   be updated next.
*/
int random()
{

    return 1 + (int)(NUM_NEURONS * rand() / (RAND_MAX+1.0));;

}
```

```
/* Title: update_utils.c
   Author: David L. Morabito

   Purpose: These routines perform the updating of the network.  Either
            synchronous updating (all neurons updated simultaneously) or
            asynchronous updating (neurons are choosen at random for
            updating) can be performed.
*/

#include <stdio.h>
#include "replica.h"

void synchronous_dynamics(network_ptr, temperature)
net *network_ptr;
double temperature;
{

   int i, j;
   double transition_probability, local_field;
   net network;

   double probability(), calculate_local_field();

   for (i = 0; i < NUM_NEURONS; ++i)
   {
      /* Save current network state so that synchronous update can be done
         one neuron at a time.
      */
      network.spins[i] = network_ptr -> spins[i];
      for (j = 0; j < NUM_NEURONS; ++j)
         network.synapses[i][j] = network_ptr -> synapses[i][j];
   }
   for (i = 0; i < NUM_NEURONS; ++i)
   {
      local_field = calculate_local_field(i, &network);
      transition_probability = probability(temperature, local_field,
                                           network_ptr -> spins[i]);

      /* If the transition probability is greater than 1/2 then flip the
         state.
      */
      if (transition_probability >= 0.50)
         if (network_ptr -> spins[i] == UP)
            network_ptr -> spins[i] = DOWN;
         else
            network_ptr -> spins[i] = UP;
   }

}

void asynchronous_dynamics(network_ptr, temperature)
net *network_ptr;
double temperature;
```

```
{

   int i, spin_pos;
   static int seed = SEED;
   double transition_probability, local_field;

   int random();
   double probability(), calculate_local_field();

   srand(seed);
   for (i = 0; i < NUM_NEURONS; ++i)
   {
      /* spin_pos represents the number of the neuron to be updated */
      spin_pos = random();
      local_field = calculate_local_field(spin_pos, network_ptr);
      transition_probability = probability(temperature, local_field,
                                           network_ptr -> spins[spin_pos]);

      if (transition_probability >= 0.50)
         if (network_ptr -> spins[spin_pos] == UP)
            network_ptr -> spins[spin_pos] = DOWN;
         else
            network_ptr -> spins[spin_pos] = UP;
   }
   seed = spin_pos;

}
```

```
/* Title: report.c
   Author: David L. Morabito

   Purpose: These routines determine the relationship (similarity) between
            the current network state and the stored patterns.
*/

#include <stdio.h>
#include "replica.h"

/* Calculate the overlap (similarity) of the current network state and
   one of the input patterns.
*/
double calculate_overlaps(network, pattern_list)
net *network;
patterns *pattern_list;
{

   int i, sum = 0, number_neurons = NUM_NEURONS;

   for (i = 0; i < NUM_NEURONS; ++i)
   {
      sum = sum + network -> spins[i] * pattern_list -> pattern[i];
   }
   return (1 / (double)number_neurons) * (double)sum;

}

/* Calculate the overlap (similarity) of the current network state and
   one of the stored patterns.
*/
double calculate_order_parameters(overlaps, pattern_number, Cinv)
double *overlaps;
int pattern_number;
float **Cinv;
{

   int i;
   double sum = 0.0;

   for (i = 0; i < NUM_PATTERNS; i++)
   {
      sum = sum + (double)Cinv[pattern_number+1][i+1] * overlaps[i];
   }
   return sum;

}

/* Search for the stored pattern with the largest overlap. */
int find_corr_pattern(overlaps)
double overlaps[];
{
```

```
   int i = 0, j, loc;
   double max;

   max = overlaps[i];
   loc = i;
   for (j = i + 1; j <= NUM_PATTERNS - 1; ++j)
      if (max < overlaps[j])
      {
         max = overlaps[j];
         loc = j;
      }

   return loc;

}

/* Print the pattern corresponding to the attractor state. */
void print_pattern(pattern_ptr)
patterns *pattern_ptr;
{
   int j, count = 0, values_per_line;

   values_per_line = N1;
   for (j = 0; j < NUM_NEURONS; ++j)
   {
      if (count == values_per_line)
      {
         printf("0);
         count = 0;
      }
      if (pattern_ptr -> pattern[j] != -1)
         printf("%d ", pattern_ptr -> pattern[j]);
      else
         printf("  ");
      count = count + 1;
   }
   printf("0);

}
```

```
/* Title: init_utils.c
   Author: David L. Morabito

   Purpose: These routines initialize the neurons (spins) in the network
            and calculate the synapses (exchange interactions).
*/

#include <stdio.h>
#include "replica.h"

/* Determine the initial network configuration. */
void init_spins(spins)
int spins[NUM_NEURONS];
{

   int i;
   FILE *spin_config, *fopen();

   spin_config = fopen("spin_config", "r");
   for (i = 0; i < NUM_NEURONS; ++i)
   {
      fscanf(spin_config, "%d", &spins[i]);
   }
   fclose(spin_config);

}

/* Read in the patterns to be stored and then calculate the synapses from
   those patterns (i.e. learn those patterns).
*/
patterns *learn(synapses, pattern_list, Cinv)
double synapses[][NUM_NEURONS];
patterns *pattern_list;
float **Cinv;
{

   int i, j, p;
   patterns *add_pattern();
   int pattern[NUM_NEURONS];
   void get_pattern();
   double get_synapses();
   double synap_value;

   for (p = 0; p < NUM_PATTERNS; ++p)
   {
      get_pattern(pattern);
      pattern_list = add_pattern(pattern_list, pattern);
   }
   get_inverse(pattern_list, Cinv);
   for (i = 0; i < NUM_NEURONS; i++)
      for (j = 0; j < NUM_NEURONS; j++)

/* Commented code is for the network to learn the patterns.  To reduce
```

```
   the time of execution the values of the synapses have been stored
   in another file and then read in.

         if (i != j)
            synapses[i][j] = get_synapses(pattern_list, i, j, Cinv);
         else
            synapses[i][j] = 0;
*/
         {
            scanf("%lf", &synap_value);
            synapses[i][j] = synap_value;
         }

   return pattern_list;

}
```

```
/* Title: pattern.c
   Author: David L. Morabito

   Purpose: These routines read in the patterns to be stored and place
            them in a linked list for future processing.
*/

#include <stdio.h>
#include "replica.h"

/* Read in a pattern. Each pattern will be found in a separate file. */
void get_pattern(pattern)
int pattern[NUM_NEURONS];
{

   int i;
   static int count = 1;
   FILE *fopen(), *file;

   file = fopen(pattern_names[count], "r");
   for (i = 0; i < NUM_NEURONS; ++i)
      fscanf(file, "%d", &pattern[i]);

   count = count + 1;

   fclose(file);

}

/* Add pattern to linked list. */
patterns *add_pattern(list, pattern)
patterns *list;
int pattern[NUM_NEURONS];
{

   int i;

   if (list == NULL)
   {
      list = (patterns *)malloc(sizeof(patterns));
      for (i = 0; i < NUM_NEURONS ; ++i)
         list -> pattern[i] = pattern[i];
      list -> next = NULL;
   }
   else
      list -> next = add_pattern(list -> next, pattern);

   return list;

}

/* Calculate the synapses from the patterns read in along with the inverse
   of the overlap matrix of the input patterns. */
```

```
double get_synapses(pattern_list, row, col, Cinv)
patterns *pattern_list;
int row;
int col;
float **Cinv;
{

   int number_neurons = NUM_NEURONS, count1, count2;
   double sum = 0.0;
   patterns *temp_ptr, *tmp_ptr;

   count1 = 1;
   temp_ptr = pattern_list;
   while (temp_ptr != NULL)
   {
      count2 = 1;
      tmp_ptr = pattern_list;
      while (tmp_ptr != NULL)
      {
         sum = sum + (temp_ptr -> pattern[row] * tmp_ptr -> pattern[col]
                                 * (double)Cinv[count1][count2]);
         tmp_ptr = tmp_ptr -> next;
         count2++;
      }
      temp_ptr = temp_ptr -> next;
      count1++;
   }

   return (1 / (double)number_neurons) * (double)sum;

}
```

```
/* Title: inverse.c
   Author: David L. Morabito
           (Some code used from book: Numerical Recipes in C - Vetterling)

   Purpose: These are utility routines that are used to calculate the
            inverse of the overlap matrix C.
*/

#include <stdio.h>
#include <math.h>
#include "replica.h"

/* Numbers smaller than TINY are considered zero. */
#define TINY 1.0e-20;

/* Calculate the inverse of a matrix. */
get_inverse(pattern_list, inv)
patterns *pattern_list;
float **inv;
{

    int i, j;
    float **mat;
    void matinv();
    float **matrix();
    void generate_overlap_matrix();

    mat = matrix(1,NUM_PATTERNS,1,NUM_PATTERNS);

    generate_overlap_matrix(pattern_list, mat);

    matinv(mat, NUM_PATTERNS, inv);

}

/* Actually perform the inverse calculation. */
void matinv(a, n, y)
float **a;
int n;
float **y;
{

   float d, *col;
   int i, j, *indx;
   void lubksb(), ludcmp();
   int *ivector();
   float *vector();

   indx = ivector(1, n);
   col = vector(1, n);
   ludcmp(a, n, indx, &d);
   for (j = 1; j <= n; j++)
   {
```

```
        for (i = 1; i <= n; i++)
           col[i] = 0.0;
        col[j] = 1.0;
        lubksb(a, n, indx, col);
        for (i = 1; i <= n; i++)
            y[i][j] = col[i];
    }

}

void lubksb(a, n, indx, b)
float **a;
int n;
int *indx;
float b[];
{

    int i, ii = 0, ip, j;
    float sum;

    for (i = 1; i <= n; i++)
    {
       ip=indx[i];
       sum = b[ip];
       b[ip] = b[i];
       if (ii)
          for (j = ii; j <= i-1; j++)
             sum -= a[i][j]*b[j];
       else if (sum)
          ii = i;
       b[i] = sum;
    }
    for (i = n; i >= 1; i--)
    {
       sum = b[i];
       for (j = i+1; j <= n; j++)
         sum -= a[i][j]*b[j];
       b[i] = sum/a[i][i];
    }

}


void ludcmp(a, n, indx, d)
float **a;
int n;
int *indx;
float *d;
{
    int i, imax, j, k;
    float big, dum, sum, temp;
    float *vv, *vector();
    void nrerror(), free_vector();
```

```
vv = vector(1,n);
*d = 1.0;
for (i = 1; i <= n; i++)
{
   big = 0.0;
   for (j = 1; j <= n; j++)
      if ((temp = fabs(a[i][j])) > big)
         big = temp;
   if (big == 0.0)
      nrerror("Singular matrix in routine LUDCMP");
   vv[i] = 1.0/big;
}
for (j = 1; j <= n; j++)
{
   for (i = 1; i < j; i++)
   {
      sum = a[i][j];
      for (k = 1; k < i; k++)
         sum -= a[i][k]*a[k][j];
      a[i][j] = sum;
   }
   big = 0.0;
   for (i = j; i <= n; i++)
   {
      sum = a[i][j];
      for (k = 1; k < j; k++)
         sum -= a[i][k]*a[k][j];
      a[i][j] = sum;
      if ((dum = vv[i]*fabs(sum)) >= big)
      {
         big = dum;
         imax = i;
      }
   }
   if (j != imax)
   {
      for (k = 1; k <= n; k++)
      {
         dum = a[imax][k];
         a[imax][k] = a[j][k];
         a[j][k] = dum;
      }
      *d = -(*d);
      vv[imax] = vv[j];
   }
   indx[j] = imax;
   if (a[j][j] == 0.0)
      a[j][j] = TINY;
   if (j != n)
   {
      dum = 1.0/(a[j][j]);
      for (i = j+1; i <= n; i++)
         a[i][j] *= dum;
```

```
        }
    }
    free_vector(vv, 1, n);

}

void free_vector(v, nl, nh)
float *v;
int nl;
int nh;
{

    free((char *)(v+nl));

}

void nrerror(error_text)
char error_text[];
{

    void exit();

    fprintf(stderr, "%s0, error_text);
    fprintf(stderr, "...now exiting to system...0);
    exit(1);

}

float *vector(nl, nh)
int nl;
int nh;
{

    float *v;
    void nrerror();

    v = (float *)malloc((unsigned)(nh-nl+1)*sizeof(float));
    if (!v)
       nrerror("allocation failure in vector()");
    return v-nl;

}

int *ivector(nl, nh)
int nl;
int nh;
{

    int *v;
    void nrerror();

    v = (int *)malloc((unsigned)(nh-nl+1)*sizeof(int));
    if (!v)
```

```
      nrerror("allocation failure in ivector()");
   return v-nl;

}

float **matrix(nrl, nrh, ncl, nch)
int nrl;
int nrh;
int ncl;
int nch;
{

   int i;
   float **m;

   m = (float **)malloc((unsigned)(nrh-nrl+1)*sizeof(float));
   if (!m)
      nrerror("allocation failure 1 in matrix()");
   m -= nrl;

   for (i = nrl; i <= nrh; i++)
   {
      m[i] = (float *)malloc((unsigned)(nch-ncl+1)*sizeof(float));
      if (!m[i])
         nrerror("allocation failure 2 in matrix()");
      m[i] -= ncl;
   }

   return m;

}

void generate_overlap_matrix(pattern_list, mat)
patterns *pattern_list;
float **mat;
{

   int sum = 0, number_neurons = NUM_NEURONS, count1, count2, i;
   patterns *temp_ptr, *tmp_ptr;


   count1 = 1;
   temp_ptr = pattern_list;
   while (temp_ptr != NULL)
   {
      count2 = 1;
      tmp_ptr = pattern_list;
      while (tmp_ptr != NULL)
      {
         for (i = 0; i < NUM_NEURONS; i++)
         {
            sum = sum + (temp_ptr -> pattern[i] * tmp_ptr -> pattern[i]);
         }
```

```
            mat[count1][count2] = (float)sum / (float)number_neurons;
            sum = 0;
            tmp_ptr = tmp_ptr -> next;
            count2++;
         }
         temp_ptr = temp_ptr -> next;
         count1++;
      }

}
```