ELEKTRİK, VOL.1, NO.1, February 1993, ©TÜBİTAK EMO

The Random Neural Network Model *

Erol Gelenbe †
Ecole des Hautes Etudes en Informatique
Université René Descartes (Paris V)

Abstract

The purpose of this article is to survey results concerning a new neural network model, the random network, which we have introduced in [1] and extended and generalized in [2,3], and to present some new results concerning its application to combinatorial optimisation. In this model "negative" or "positive" signals circulate, modelling inhibitory and excitatory signals. These signals can arrive either from other neurons or from the outside world, they are summed at the input of each neuron and constitute its signal potential. The state of each neuron in this model is its signal potential, while the network state is the vector of signal potentials at each neuron. If its potential is positive, a neuron fires, and sends out signals to the other neurons of the network or to the outside world. As it does so its signal potential is depleted. We have shown [1] that in the Markovian case, this model has product form, i.e. the steady-state probability distribution of its potential vector is the product of the marginal probabilities of the potential at each neuron. The signal flow equations of the network, which describe the rate at which positive or negative signals arrive to each neuron, are non-linear, so that their existence and uniqueness is not easily established except for the case of feedforward (or backpropagation) networks. In [2], the relationship between this model and the usual connectionist (formal) model of neural networks is discussed, and two sub-classes of networks are examined: balanced, and damped networks and stability conditions are examined in each case. In practical terms, these stability conditions guarantee that the unique solution can be found to the signal flow equations and therefore that the network has a well-defined steady-state behaviour. In [3] the model is extended to the case of "multiple signal classes" dealing with networks in which each neuron processes several streams of signal simultaneously. We indicate how the random network can be used to implement the well-known XOR example. We then apply it to solve approximately an NP-hard combinatorial optimization problem, the graph covering problem. We also suggest a hardware implementation of the model.

1. Introduction

We model a network of n neurons in which *positive and negative* signals circulate. Each neuron accumulates signals as they arrive, and can fire if its total signal count at a given instant of time is positive. Firing then occurs at random according to an exponential distribution of constant rate, and it sends signals out to other neurons or to the outside of the network. In this model, each neuron i of the network is represented at any time t by its input signal potential $k_i(t)$, which we shall simply call the potential.

Positive and negative signals have different roles in the network; positive signals represent excitation, while negative signals represent inhibition. A negative signal *reduces by 1* the potential of the neuron to which

* This research was supported by the Distributed Algorithms Section of C3-CNRS (French National Program on Parallelism and Concurrency)

† Author's address : Ecole des Hautes Etudes en Informatique, Université René (Paris V), 45 rue des Saints, Peres, 75006 Paris.

it arrives (i.e. it "cancels" an existing signal) or has no effect on the signal potential if it is already zero, while an arriving positive or excitation signal *adds 1* to the neuron potential. The potential at a neuron is constituted only by positive signals which have accumulated, which have not yet been cancelled by negative signals, and which have not yet been sent out by the neuron as it fires.

This is a simplified representation of biophysical neural behaviour, which is shown in Fig. 1 (see for instance [5] or [7]). In this representation, at time $t = 0$ a neuron is excited; at time T (typically T may be of the order of 50 milliseconds) it fires a train of impulses along its axone. Each of these impulses is practically of identical amplitude (represented in our random model by 1). Some time later (say around $t = T + \tau$) the neuron may fire another train of impulses, as a result of the same excitation. Even when the neuron is not excited, it may send out impulses at random. Our model represents firing of an excited neuron by a train of impulses or unit signals; the $+1$ or -1 value is tied to the interpretation (excitation or inhibition) given to the arriving signal at the receiving neuron.

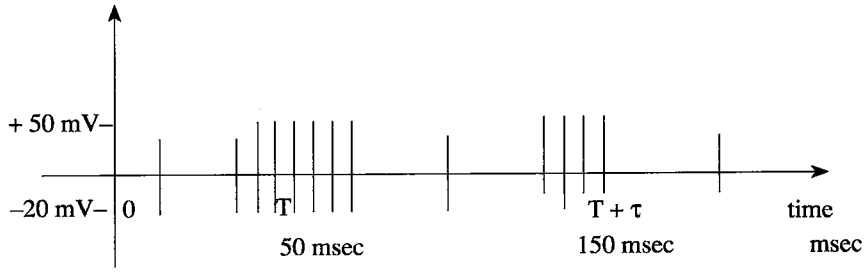

Figure 1. Representation of biophysical neuron output signal after excitation at time $t = 0$

Signals can either arrive to a neuron from the outside of the network (exogenous signals) or from other neurons. Each time a neuron fires, a signal leaves it depleting the total input potential of the neuron. A signal which leaves neuron i heads for neuron j with probability $p^+(i,j)$ as a positive (or normal) signal, or as a negative signal with probability $p^-(i,j)$, or it departs from the network with probability $d(i)$. Let $p(i,j) = p^+(i,j) + p^-(i,j)$; it is the transition probability of a Markov chain representing the movement of signals between neurons.

We shall assume that $p^+(i,i) = 0$ and $p^-(i,i) = 0$; though the former assumption is not essential we insist on the fact that the latter indeed is to our model; this assumption excludes the possibility of a neuron sending a signal directly to itself. Clearly we shall have

$$\sum_j p(i,j) + d(i) = 1 \quad \text{for} \quad 1 \leq i \leq n.$$

A neuron is capable of firing and emitting signals if its potential is strictly positive.

We assume that exogenous signals arrive to each neuron in Poisson streams of positive or negative signals.

In [1] we show that the Markovian version of this network, with positive signals which arrive to the i-th neuron according to a Poisson process of rate $\Lambda(i)$, *iid* exponential neuron firing times with rates $r(1), \ldots, r(n)$, and Markovian movements of signals between neurons, has a product form solution. That is, the open network's stationary probability distribution can be written as the product of the marginal probabilities of the state of each neuron.

Thus *in steady-state the network's neurons are seemingly independent though they are in fact coupled* via the signals which move from one neuron to the other in the network.

The model we propose has a certain number of interesting features:

- it appears to represent more closely the manner in which signals are transmitted in a biophysical neural network where they travel as voltage spikes rather than as fixed signal levels,

- it is computationally efficient in the feed-forward case, and whenever network stability can be shown,
- it is easy to simulate, since each neuron is simply represented by a counter; this may lead to a simple hardware implementation which is described in Section 7,
- it is closely related to the connexionist model and it is possible to go from one model to the other easily,
- it represents neuron potential and therefore the level of excitation as an integer, rather as a binary variable, which leads to more detailed information on system state; furthermore the neuron is interpreted as being in the "firing state" if its potential is positive.

As one may expect from existing models of neural networks [4], the signal flow equations which yield the rate of signal arrival and hence the rate of firing of each neuron in steady-state are non-linear. Thus in [1] we were able to establish their existence (and also a method for computing them) only in the case of feed-forward networks, i.e. in networks where a signal cannot return eventually to a neuron which it has already visited either in negative or positive form. This of course covers the case of back-propagation ornetworks [2]. In this paper we are able to establish existence and uniqueness for so-called "balanced and quasi-balanced" networks as well as for the class of "dissipative" networks, and uniqueness of the solution in the general case whenever the solution exists.

In this paper we shall first recall briefly the main results proved in [1] concerning the *random neural network model.*

We then illustrate the relation of the model to the usual connexionist network. We illustrate it with a network which computes the exclusive OR (XOR) function. We then present the manner in which the random network model can be used to solve approximately some hard combinatorial problems.

Two special classes of networks [2] which have internal feedback and for which we provide stability results and compute the network state are then characterized. These are the balanced networks, and the quasi-damped networks. The first class includes all networks for which the steady-state behaviour of each neuron is the same. The second class includes all networks in which the positive signal flow rates to neurons are small enough that the network would be stable even if internal negative signals are removed. The generalization of the model to "multiple class signals" [3] is also discussed; in this new development, it is suggested that neural networks which process simultaneously several different activities be considered. We apply our new model to the approximate solution of the graph covering problem, which is NP-hard, and compare the results obtained to previous approaches using simulated annealing, and to the existing greedy algorithm. Finally we suggest a simple hardware realization of random networks.

2. General Properties of the Random Network Model

In this section we shall recall the main theoretical result concerning the random network model.

The following property states that the steady-state probability distribution of network state can always be expressed as the product of the probabilities of the states of each neuron. Thus the network is seemingly composed of independent neurons, though this is obviously not the case. Let $k(t)$ be the vector of signal potentials at time t, and $k = (k_1, \ldots, k_n)$ be a particular value of the vector. We are obviously interested in the quantity $p(k,t) = \text{Prob}[k(t) = k]$.

Let $p(k)$ denote the stationary probability distribution

$$p(k) = \lim_{t \to \infty} \text{Prob}[k(t) = k]$$

if it exists.

Theorem 1. [1] Let

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)], \tag{1}$$

where the $\lambda^+(i), \lambda^-(i)$ for $i = 1, \ldots, n$ satisfy the following system of non-linear simultaneous equations:

$$\begin{aligned} \lambda^+(i) &= \sum_j q_j r(j) p^+(j,i) + \Lambda(i), \\ \lambda^-(i) &= \sum_j q_j r(j) p^-(j,i) + \Lambda(i), \end{aligned} \tag{2}$$

If a unique non-negative solution $\{\lambda^+(i), \lambda^-(i)\}$ *exists to equations* (1), (2) *such that each* $q_i < 1$, then:

$$p(k) = \prod_{i=1}^{n} [1 - q_i] q_j^{k_i} \tag{3}$$

We omit the proof, which can be found elsewhere [1]. Since $\{k(t) : t \geq 0\}$ is a continuous time Markov chain it satisfies the usual Chapman-Kolmogorov equations; thus in steady-state it can be seen that $p(k)$ must satisfy the following global balance equations:

$$\begin{aligned} p(k) \sum_i [\Lambda(i) &+ (\lambda(i) + r(i)) 1[k_i > 0]] \\ &= \sum_i [p(k_i^+) r(i) d(i) \\ &+ p(k_i^-) \Lambda(i) 1[k_i > 0] + p(k_i^+) \lambda(i) \\ &+ \sum_j \{p(k_{ij}^{+-}) r(i) p^+(i,j) 1[k_j > 0] \\ &+ p(k_{ij}^{++}) r(i) p^-(i,j)) \\ &\quad + p(k_i^+) r(i) p^-(i,j) 1[k_j = 0]\}] \end{aligned} \tag{4}$$

where the vectors used in (4) are defined as follows:

$$\begin{aligned} k_i^+ &= (k_1, \ldots, k_i + 1, \ldots, k_n) \\ k_i^- &= (k_1, \ldots, k_i - 1, \ldots, k_n) \\ k_{ij}^{+-} &= (k_1, \ldots, k_i + 1, \ldots, k_j - 1, \ldots, k_n) \\ k_{ij}^{++} &= (k_1, \ldots, k_i + 1, \ldots, k_j + 1, \ldots, k_n) \end{aligned}$$

and $1[X]$ is the usual characteristic function which takes the value 1 if X is true and 0 otherwise. Theorem 1 is proved by showing that (3) satisfies this system of equations.

The computational simplicity of this result is illustrated by the useful consequence indicated below.

Corollary 1.1. The probability that neuron i is firing in steady-state is simply given by q_i of (1) and the average neuron potential in steady-state is simply $A_i = q_i/[1 - q_i]$.

By Theorem 1 we are guaranteed a stationary solution of product form provided the non-linear signal flow equations have a non-negative solution. The following result can be easily established (though this was not done in [1]).

Theorem 2. If the solution to (1), (2) exist with $q_i < 1$, then they are unique.

Proof. Since $\{k(t) : t \geq 0\}$ is an irreducible and aperiodic Markov chain, if a positive stationary solution $p(k)$

exists, then it is unique. By Theorem 1, if the $0 < q_i < 1$ solution to (1), (2) exist for $i = 1, \ldots, n$, then $p(k)$ is given by (3) and is clearly positive for all k. Suppose now that for some i there are two different q_i, q_i' satisfying (1), (2). But this implies that for all k_i, $\lim_{t\to\infty} P[k_i(t) = 0]$ has two different values $[1 - q_i]$ and $[1 - q_i']$, which contradicts the uniqueness of $p(k)$; hence the result.

Let us now turn to the existence and uniqueness of the solutions $\lambda^+(i), \lambda^-(i)$, $1 \leq i \leq n$ to equations (1), (2) which represent the average arrival rate of positive and negative signals to each neuron in the network. An important class of models is covered by the following result concerning *feedforward networks.*

We say that a network is *feedforward* if for any sequence $i_1, \ldots, i_s, \ldots, i_r, \ldots, i_m$ of neurons, $i_s = i_r$ for $r > s$ implies

$$\prod_{v=1}^{m-1} p(i_v, i_{v+1}) = 0$$

Theorem 3. [1] If the network is *feedforward*, then the solutions $\lambda^+(i), \lambda^-(i)$ to equations (1), (2) exist and are unique.

We shall omit the proof of this result which can be found in [1]. Its significance lies in the fact that it covers the usual backpropagation networks.

□

Let us now examine the similarity relationship of our model with the usual connexionist model [4].

2.1. Analogy with classical connexionist networks

A connexionist network [4] is a set of n neurons each of which computes its *state* $y(i)$ using a sigmoid function $y(i) = f(x(i))$ where $x(i) = (\Sigma_j\, w_{ji} y(j) - \Theta_i)$ is the *input signal* composed of the weighted sums of the states $y(j)$ of the other neurons of the network; the w_{ji} are the weights and Θ_i is the threshold. In its simplest form $f(\,)$ is the unit step function. The set of weights and the set of thresholds completely characterise the network.

An analogy between the usual model of neural networks and the model introduced in this paper, which we henceforth call *the random network*, can be constructed.

Each neuron is represented by a neuron of the random network. The threshold of neuron i is represented by a flow of negative signals to the neuron so that $\lambda(i) = \Theta_i$.

Consider the non-output neuron i; it is represented by the random neuron i whose parameters are chosen as follows:

$$d(i) = 0, \;\; r(i)p^+(i,j) = w_{ij} \text{ if } w_{ij} > 0$$

and

$$r(i)p^-(i,j) = |w_{ij}| \text{ if } w_{ij} < 0$$

Summing over all j, the firing rate $r(i)$ of the non-output "random" neuron i is chosen:

$$r(i) = \sum_j |w_{ij}|$$

Finally, for each output random neuron i set $d(i) = 1$, and assign some appropriate value to $r(i)$.

To introduce into the random network the external parameters or inputs to the neural network we can use the arrival rates of positive signals $\Lambda(i)$. Assume that the external signals to the formal neural network are binary. If the input signal to neuron i is 0 or if neuron i is not connected to an external signal we can set $\Lambda(i) = 0$. If the external signal to neuron i has the value 1, we set $\Lambda(i) = \Lambda$. Here Λ can be chosen so as to obtain the desired effect at the output neurons.

All the parameters of the random network are chosen from those of the formal network, except for the firing rates of the output neurons, and the input rate of positive signals.

The state $Y = (y_1, \ldots, y_n)$ of the formal neural network, where y_i is 0 or 1, is simulated by the state probabilities of the random neurons.

Consider $\lim_{t\to\infty} P[k_i(t) > 0]$ which (as a consequence of Corollary 1.1) is simply q_i; we associate y_i to q_i. Thus we have for some "cut-point" $1 - \alpha$,

$$[y_i = 0] \iff q_i < 1 - \alpha;$$
$$[y_i = 1] \iff q_i \geq 1 - \alpha.$$

A finer (i.e. more detailed) way of differentiating between highly excited and relatively unexcited neurons is via the average neuron potential $A_i = q_i/[1 - q_i]$ as will be seen in the example given below.

In an arbitrary neural network, i.e. one which does not have the feedforward structure, this procedure could also be used for establishing the random network. We could also use the $d(i)$ at each neuron i in order to represent the loss currents which are known to exist in biological neural systems [5], and take $r(i)d(i)$ to be the rate of loss of electric potential at a neuron if we wish to include this effect in the model.

Example 1. A simple example often given to illustrate the behaviour of formal neural networks is the network for the XOR (exclusive OR) function [4]. We shall present the equivalent random model representation for this network.

In Figure 2(a) we show a formal neural network which receives two binary inputs x_1, x_2 and which produces $y(x_1, x_2)$ the boolean function XOR. It is composed of four neurons; each neuron is numbered (1 to 4) and the synaptic weights are indicated on arcs between neurons. The threshold of each neuron is assumed to be 0. On Figure 2(b) we show the random network analog corresponding to Figure 2(a).

According to the rules we have given for constructing the random network analog, we have:

- $\lambda(i) = 0$ for $i = 1, \ldots, 4$ because all thresholds are 0;
- $r(1) = r(2) = 2$, $r(3) = 1.1$, $r(4) = r$, as yet undetermined;
- $p^+(1,3) = p^+(2,3) = 0.5$, $p^-(1,4) = p^-(2,4) = 0.5$, $p^+(3,4) = 1$;
- $d(1) = d(2) = d(3) = 0$, $d(4) = 1$.

Recall that according to the rules proposed in Section 2.1, we choose a value $\Lambda(i) = \Lambda$ to represent $x_i = 1$, and $\Lambda(i) = 0$ to represent $x_i = 0$. Set Λ large enough to saturate neurons 1 and 2, i.e. any $\Lambda > 2$.

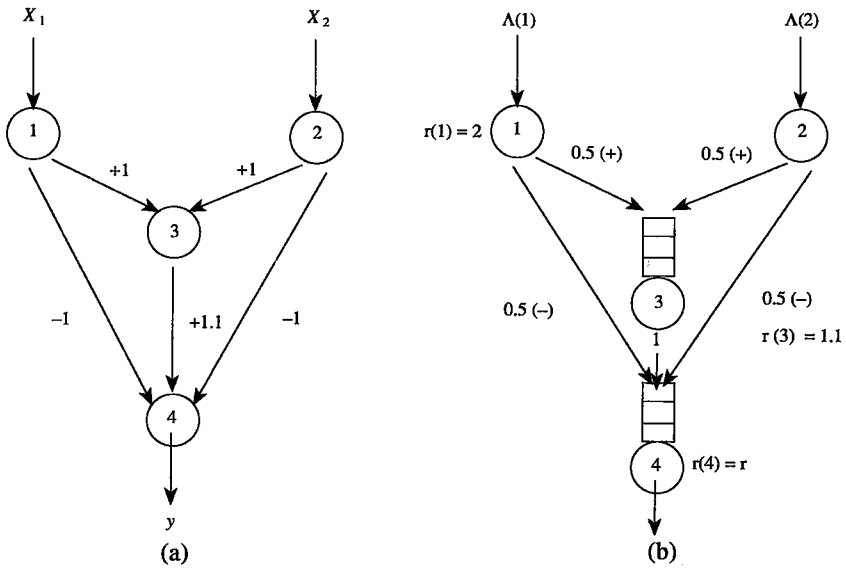

Figure 2. a) A connexionist network for the Boolean XOR function, b) The random network analog of the formal neural network shown in Figure 2(a) ; here, $p^+(1,3) = p^+(2,3) = 0.5, p^+(3,4) = 1$, and $p^-(1,4) = p^-(2,4) = 0.5$

q_4 is the analog of the output y of the connexionist network of Figure 2(a). Notice that:

$$q_4 = \begin{cases} 0, & \text{if } \Lambda(1) = L(2) = 0, \\ 1.1/[r+2], & \text{if } \Lambda(1) = \Lambda(2) = \Lambda > 2, \\ 1/[r+1], & \text{if } \Lambda(1) = \Lambda,\ \Lambda(2) = 0 \text{ or vice-versa}. \end{cases}$$

Setting $\alpha = 0.85$ and $r = 0.1$ we see that we obtain the XOR function with this network, since we have $q_4 = 0.909$ when $\Lambda(1) = \Lambda$, $\Lambda(2) = 0$ or vice-versa, and $q_4 = 0.5238$ if $\Lambda(1) = \Lambda(2) = \Lambda$ for any $\Lambda > 2$. In fact we may choose any $1 - \alpha$ such that $1.1/[r+2] < 1/[r+1]$. The capacity of the network to represent the XOR function becomes even more apparent if we consider the average potential of neuron 4; we then have $A_4 = 11.222$ when $\Lambda(1) = 2, \Lambda(2) = 0$ or vice-versa, and $A_4 = 1.0999$ if $\Lambda(1) = \Lambda(2) = \Lambda$ for any $\Lambda > 2$. Thus, the average neuron potential can also be used very effectively to discriminate between the output states of the neural network since we have a ratio greater than 10 between the average potential of neuron 4 for the input equivalent to (0, 1) or (1, 0) and the input equivalent to (1, 1).

3. Balanced Neural Networks

We now consider a class of networks whose signal flow equations have a particularly simple solution. We shall say that *a network with negative and positive signals is balanced* if the ratio

$$\delta_i = [\sum_j q_j r(j) p^+(j,i) + \Lambda(i)]/[\sum_j q_j r(j) p^-(j,i) + \lambda(i) + r(i)]$$

is identical for any $i = 1, \ldots, n$. This in effect means that all the q_i are identical.

Theorem 4. The signal flow equations (1), (2) have a (unique) solution if the network is balanced.

4. Damped and Dissipative Networks

We say that a random network is *dissipative* if $p^+(j,i) = 0$ for all i, j.

Notice that a dissipative network will, in general, need to have some positive exogenous signal arrivals to be of any interest. However in such a network a neuron may only send negative signals to another neuron.

Theorem 5. [2] If the network is dissipative and $r(i) + \lambda(i) > \Lambda(i)$ then the signal flow equations (1), (2) have a unique solution with $q_i < 1$.

In fact such networks are a special case of damped networks which will be discussed below.

We shall say that a network is *damped* if $p^+(j,i) \geq 0$, $p^-(j,i) \geq 0$ with the following property: $r(i) + \lambda(i) > \Lambda(i) + \Sigma_j r(j)p^+(j,i)$ for all $i = 1, \ldots, n$.

Theorem 6. [2] If the network is damped then the customer flow equations (1), (2) always have a unique solution with $q_i < 1$.

Proof. The proof is by construction of the n-dimensional vector homotopy [6] function $H(q,x)$ for a real number $0 \leq x < 1$. Let us define the following n-vectors:

$$q = (q_1, \ldots, q_n),\ F(q) = (F_1(q), \ldots, F_n(q))$$

where

$$F_i(q) = [\sum_j q_j r(j) p^+(j,i) + \Lambda(i)]/[\sum_j q_j r(j) p^-(j,i) + \lambda(i) + r(i)]$$

Clearly, the equation we are interested in is $q = F(q)$ which, when it has a solution in $D = [0,1]^n$, yields the appropriate values of the q_i for Theorem 1 to be applicable in order to compute the stationary solution of the network. Notice that $F(q) : R^n \to R^n$. Notice also that $F(q) \in C^2$. Consider the mappings $F(q) : D \to R^n$. Clearly, we are interested in the interior points of D since we seek solutions $0 < q_i < 1$. Write $D = D^0 \cup \delta D$ where δD stands for the boundary of D, and D^0 is the set of interior points. Let $y = (y_1, \ldots, y_n)$ where

$$y_i = [\sum_j r(j) p^+(j,i) + \Lambda(i)]/[\lambda(i) + r(i)]$$

By assumption $y_i < 1$ for all $i = 1, \ldots, n$. Now define

$$H(q,x) = (1-x)(q-y) + x(q - F(q)), \quad 0 \leq x < 1.$$

Clearly $H(q,0) = q - y$ and $H(q,1) = q - F(q)$. Consider the set

$$H^{-1} = \{q : q \in D,\ H(q,x) = 0 \text{ and } 0 \leq x < 1\}.$$

We can show that H^{-1} and δD have an empty intersection, i.e. as x is varied from 0 towards 1 the solution of $H(q,x)$ if it exists does not touch the boundary of D. To do this assume the contrary; this implies that for some $x = x^*$ there exists some $q = q^*$ for which $H(q^*, x^*) = 0$ and such that $q_i^* = 0$ or 1. If $q_i^* = 0$ we can write

$$-(1-x^*)y_i - x^* F_i(q^*) = 0,$$

or

$$x^*/(1-x^*) = -y_i/F_i(q^*) < 0 \Rightarrow x^* < 0,$$

which contradicts the assumption about x. If on the other hand $q_i^* = 1$, then we can write

$$-(1-x^*)(1-y_i) - x^*(1 - F_i(q^*)) = 0,$$

or

$$x^*/(1-x^*) = -(1-y_i)/(1-F_i(q^*)) < 0 \Rightarrow x^* < 0$$

because $(1-y_i) > 0$ and $0 < F_i(q^*) < y_i$ so that $(1-F_i(q^*)) > 0$, contradicting again the assumption about x. Thus $H(q,x) = 0$ cannot have a solution on the boundary δD for any $0 \leq x < 1$. As a consequence, applying Theorem 3.3.1 of [4] (which is a Leray-Schauder form of the fixed-point theorem), it follows that $F(q) = q$ has at least one solution in D^0; it is therefore a unique solution as a consequence of Theorem 2.

5. Multiple Class Networks

Consider a neural network which generalises the model presented in [1, 2]. It is composed of n neurons and receives exogenous positive (excitatory) and negative (inhibitory) signals, as well as endogenous signals exchanged by the neurons. As in [1, 2] excitatory or inhibitory signals are sent by excited neurons to other neurons in the network, or to the outside world. The arrival of an excitatory signal increases the potential of a neuron by 1, while an inhibitory signal's arrival decreases it by one. A neuron is excited if its potential is positive; it then fires at exponentially distributed intervals sending excitatory signals of *different classes*, or inhibitory signals, to other neurons or to the outside of the network. This model has been introduced in [3]. It represents a neural network which processes several streams of information in parallel, which appears to be a sensible generalization of existing models.

Positive signals may belong to several *classes*. In this model, the potential at a neuron is represented by the vector $\underline{k}_i = (k_{i1}, \ldots, k_{iC})$ where k_{ic} is the value of the "class c potential" of neuron i, or its "excitation level in terms of class c signals". The total potential of neuron i is $k_i = \Sigma_{c=1,C} k_{ic}$.

Exogenous positive signals of class c arrive to neuron i in a Poisson stream of rate $\Lambda(i,c)$, while exogenous negative signals arrive to it according to a Poisson process of rate $\lambda(i)$. Thus negative signals belong to a single class. When a positive signal of class c arrives to a neuron, it merely increases k_{ic} by 1. When a negative signal arrives to it, if $k_i > 0$ the potential is reduced by 1, and the class of the potential to be reduced is chosen at random: with probability k_{ic}/k_i it is of class c for any $c = 1, \ldots, C$. As with single class networks discussed above, a negative signal arriving to a neuron whose potential is zero has no effect on its potential.

When its potential is positive ($k_i > 0$), neuron i can fire; the potential depleted is of class c with probability k_{ic}/k_i in which case the neuron fires at rate $r(i,c) > 0$. In the interval $[t, t+\Delta t]$, neuron fires, depletes by 1 its class c potential, and sends to neuron j a class ξ positive signal with probability:

$$r(i,c)(k_{ic}/k_i)p^+(i,c;j,\xi)\Delta t + o(\Delta t)$$

or a negative signal with probability:

$$r(i,c)(k_{ic}/k_i)p^-(i,c;j)\Delta t + o(\Delta t)$$

On the other hand, the probability that the depleted signal is sent out of the network, or that it is "lost" is

$$r(i,c)(k_{ic}/k_i)d(i,c)\Delta t + o(\Delta t)$$

The $\{p^+(i,c;j,\xi), p^-(i,c;j), d(i,c)\}$ are the transition probabilities of a Markov chain with state-space $\{1,\ldots,n\} \times \{1,\ldots,C\} \times \{+,-\}$ representing the movement of signals in the network, and for (i,c), $1 \leq i \leq n, 1 \leq c \leq C$:

$$\sum_{j,\xi} p^+(i,c;j,\xi) + \sum_j p^-(i,c;j) + d(i,c) = 1.$$

Notice that if the network only contains a single class of positive signals ($C = 1$) then we simply have the model presented previously in [1, 2].

The complete state of the network is represented by the vector (of vectors) $\underline{k} = (\underline{k}_1, \ldots, \underline{k}_n)$. Under the above assumptions, the process $\{\underline{k}(t), t \geq 0\}$ is Markovian, and we shall denote by $p(\underline{k}, t) \equiv P[\underline{k}(t) = \underline{k}]$ the probability distribution of its state. Its behavior is then described by the Chapman-Kolmogorov equations:

$$\begin{aligned}
dp(\underline{k},t)/dt \;=\; & -p(\underline{k},t)\sum_{(i,c)}[\Lambda(i,c) \\
& + (\lambda(i) + r(i,c)(k_{ic}/k_i)] \\
& + \sum_{(i,c)}\{p(\underline{k}+e_{ic},t)r(i,c)((k_{ic}+1)/(k_i+1))d(i,c) \\
& + p(\underline{k}-e_{ic},t)\Lambda(i,c)1[k_{ic}>0] \\
& + p(\underline{k}+e_{ic},t)\lambda(i)((k_{ic}+1)/(k_i+1)) \\
& + \sum_{(i,\xi)}(p(\underline{k}+e_{ic}-e_{j\xi},t)r(i,c)((k_{ic}+1)/(k_i+1))p^+(i,c;j,\xi)1[k_{j\xi}>0] \\
& + p(\underline{k}+e_{ic},t)r(i,c)((k_{ic}+1)/(k_i+1))p^-(i,c;j)1[k_j=0] \\
& + p(\underline{k}+e_{ic}+e_{j\xi},t)r(i,c)((k_{ic}+1)/(k_i+1))((k_{jc}+1)/(k_j+1))p^-(i,c;j))\}
\end{aligned}$$

where we have used the notation:

$$\begin{aligned}
\underline{k}+e_{ic} \;&=\; (\underline{k}_1,\ldots,(\underline{k}_{i1},\ldots,k_{ic}+1,\ldots,k_{iC}),\ldots,\underline{k}_n) \\
\underline{k}+e_{jc} \;&=\; (\underline{k}_1,\ldots,(\underline{k}_{j1},\ldots,k_{jc}+1,\ldots,k_{jC}),\ldots,\underline{k}_n)
\end{aligned}$$

$$\begin{aligned}
\underline{k}+e_{ic}-e_{j\xi} \;&=\; (\underline{k}_1,\ldots,(k_{i1},\ldots,k_i+1,\ldots,k_{iC}) \\
&\quad\;,\ldots,(k_{j1},\ldots,k_\xi-1,\ldots,k_{iC}) \\
&\quad\;,\ldots,\underline{k}_n) \\
\underline{k}+e_{ic}+e_{j\xi} \;&=\; (\underline{k}_1,\ldots,(k_{i1},\ldots,k_i+1,\ldots,k_{iC}) \\
&\quad\;,\ldots,(k_{j1},\ldots,k_\xi+1,\ldots,k_{iC}) \\
&\quad\;,\ldots,\underline{k}_n)
\end{aligned}$$

These vectors are defined only if their elements are non-negative.

The stationary solution of the model described above has product form, i.e. its steady-state probability distribution is the product of the marginal probabilities of each neuron.

Theorem 7. [3] Let $\underline{k}(t)$ be the vector representing the state of the neural network at time t, and let $\{q_{ic}\}$ with $0 < \Sigma_{(i,c)} q_{ic} < 1$, be the solution of the system of non-linear equations:

$$q_{ic} = \lambda + (i,c)/[r(i,c) + \lambda^-(i)], \tag{5}$$

$$\begin{aligned}
\lambda^+(i,c) \;&=\; \sum_{(j,\xi)} q_{j\xi} r(j,\xi) p^+(j,\xi;i,c) + \Lambda(i,c), \\
\lambda^-(i) \;&=\; \sum_{(j,\xi)} q_{j\xi} r(j,\xi) p^-(j,\xi;i) + \lambda(i).
\end{aligned} \tag{6}$$

Then the stationary solution $p(\underline{k}) \equiv \lim_{t\to\infty} P[\underline{k}(t) = \underline{k}]$ exists and is given by:

$$p(\underline{k}) = \prod_{i=1}^{n}(k_i!)G_i \prod_{c=1}^{C}[(q_{ic})^{k_{ic}/k_{ic}!}], \tag{7}$$

where the G_i are appropriate normalising constants.

6. Use of the Network Model for Graph Covering, a Combinatorial Optimization Problem Which Appears in Image Processing

Certain combinatorial problems arise in a routine manner in image processing applications. One such problem is minimum graph covering. It is an NP-complete problem, so that efficient algorithms for its exact solution when the number of nodes of the graph is very large, are not available.

The formal problem can be stated as follows. Let G be a graph with K nodes $\mathbf{K} = \{1, \ldots, K\}$ and edges denoted by (u, v) where u, v are nodes of the graph. A *cover* of G is a subset C of $\mathbf{K}$ such that for each edge (u, v) in G, either u or v is in C. A *minimum cover* of G is a set C^* such that the number of nodes in C^* is no larger than the number of nodes in any cover C of $G : |C^*| \leq |C|$.

In intuitive terms, if an image is represented by a set of interconnected straight lines, then the minimum cover will yield a representation of the image with a minimum number of components. In the transposition of the image to the graph representation, the straight lines of the image can be represented by vertices of G, while lines which intersect at an angle would give rise to an edge in G.

The minimum graph covering problem can also be formulated so that each node of the graph G can carry a weight, and this weight is included in the "size" of each cover C. We summarize and compare here the solution of the minimum cover problem using four main approaches:

- The exact solution E; in this case all covers are enumerated and a minimum cover is selected. For graphs chosen at random, we have rapidly discovered that this approach was too time consuming to implement on a workstation when the number of nodes of G exceeded $K = 50$.

- Simulated annealing SA; this is a simulated "neural network" approach based on the paper of Hopfield and Tank [9].

- Heuristic solution using a known approach, the Greedy Algorithm GA. The idea here is to first select the node of G which has the highest degree (number of neighbours), and to include it in C^+ the cover being generated; the node and all its adjacent edges and terminal nodes is then removed from the graph and the procedure is repeated until all nodes have been removed, or placed in C^+.

- Heuristic solution using the Random Network discussed in this paper. We now describe this part in some greater detail.

6.1. Random network (RN) solution to the minimum graph covering problem

The random network used to solve the problem is set up in the following manner. For each node i of G, we construct two neurons $N(i)$ and $n(i)$. The objective is to have $N(i) = 1$ and $n(i) = 0$ if $i \in C^*$, and $N(i) = 0$ and $n(i) = 1$ in the opposite case. The network parameters are chosen as follows:

- $\Lambda(N(i)) = D(i)$, the degree of node i
- $\Lambda(N(i)) = 1$ for all i
- $r(N(i)) = 2K$
- $r(n(i)) = D(i)$
- $p^-(N(i), n(i)) = 1$ for all i
- $p^+(n(i), N(j)) = 1/D(i)$ if j is a neighbour of i

- $\lambda(N(i)) = 0, \lambda(n(i)) = 0$ for all i.

It can be easily seen that the network considered is hyperstable, so that the stationary solution of the network state exists. It is obtained by an iterative numerical solution of equations (1), (2).

The algorithm used for obtaining a "quasi-minimal" cover C^- using the random network is as follows:

1. Solve for the $q(N(i)), q(n(i)),\ i = 1, \ldots, K$ from the above model.
2. Place in C^- the node, say i^*, for which $q(N(i))$ is largest, i.e. closest to 1.
3. Remove i^* and all its neighbours from G, and reinitialize the above model without i^*.
4. Return to 1) if there is more than one node in the remaining graph; otherwise include in C^- and end the procedure.

This algorithm uses the random network as a decision making tool in the greedy algorithm. Notice that step 1) of the algorithm, which is the most time-consuming part, can be accelerated and executed in parallel with up to $2K$ processors.

6.2. Experimental comparison

Let us now compare results obtained with the four methods described above, using randomly generated graphs. We compare both the quality of the results obtained, and the total computation time of the method on the same workstation.

A family of random graphs is defined by two parameters (K, π), where π is the probability that there is an arc between any two nodes of the graph. For $K = 20$ and 50 we tabulate results and indicate: the fraction of cases where the method found the minimum cover (**mini**), the average number of nodes in excess of the minimum cover (**exc**), and the computation time **T**. For $K = 100$ the exhaustive search of the minimum cover for each graph was impossible to carry out. Thus for $K = 100$, **exc** stands for the average number of nodes in excess of the smallest cover found among the four methods tested. For each (K, π), the number of graphs chosen at random is 25, and the values shown are average values over this set.

$K = 20$	$\pi = 0.5$	$\pi = 0.25$	$\pi = 0.125$
Method	**(mini,exc , T)**	**(mini ,exc , T)**	**(mini,exc , T)**
SA	100% , 0 , 43s	88% , 0.12 , 35s	96% , 0.04 ,27s
GA	72% , 0.28 , 0s	68% , 0.40 , 0s	84% , 0 , 0s
RN	100% , 0 , 3s	96% , 0.04 , 0s	100% , 0 , 0s
//-RN (Optimistic)	-,-,0.075s	-,-, 0s	-,-, 0s

$K = 50$	$\pi = 0.5$	$\pi = 0.25$	$\pi = 0.125$
Method	**(mini,exc , T)**	**(mini ,exc , T)**	**(mini,exc , T)**
SA	72%,0.28 , 43s	56% , 0.60 , 2'05"	56% , 0.48 , 1'30"
GA	16% , 1.20 , 0s	24% , 1.28 , 0s	12% , 1.52 , 0s
RN	52% , 0.52 , 32s	60% , 0.4 , 20s	80% , 0.2 , 9s
//-RN (Optimistic)	-,-, 0.32s	-,-, 0.2s	-,-, 0.09s

$K = 100$	$\pi = 0.5$	$\pi = 0.25$	$\pi = 0.125$
Method	**(mini,exc , T)**	**(mini ,exc , T)**	**(mini,exc , T)**
SA	68% , 0.36 , 10'	52% , 0.72 , 2'20"	32% , 1.32 , 4'20"
GA	20% , 1.28 , 0s	0% , 2.04 , 0s	4% , 2.72 , 0s
RN	60% , 0.52 , 3'40"	48% , 0.56 , 2'00"	56% , 0.52 , 1'10"
//-RN (Optimistic)	-,-, 1.1s	-,-, 0.6s	-,-, 0.35s

We notice that for large π, Simulated Annealing (SA) provides better results; however the computational cost is generally incompatible with "fast" or real-time applications. For smaller values of π, which correspond to more realistic image processing applications since the graphs considered are less dense, the Greedy Algorithm (GA) provides poor results, while the Random Network (RN) algorithm provides substantially better results than SA. GA is the lowest cost algorithm in terms of computation time, but is much less effective than the RN algorithm. In the best of cases, using parallelism, the RN algorithm can *potentially* become as fast as GA.

In order to examine the situation for sparsely connected graphs we show below results concerning a simulation of 25 graphs with $K = 100$, and $\pi = 0.0625$.

$K = 100$	$\pi = 0.0625$
Method	**(mini , exc, T)**
SA	12% , 1.80 , 3'10"
GA	4% , 2.88 , 0"
RN	76% , 0.36 , 35"
//-RN (Optimistic)	-,-, 0.178"

Here Simulated Annealing seldom finds the optimum solution, and its computation time is very high. The conventional Greedy Algorithm (GA) is ineffective. The Random Network solution provides the best results at higher computational cost than GA, but much lower than SA.

At this point of the evaluation, we dispose of a reasonable general comparison of these methods. Any further comparison of these techniques would have to be based on graphs which describe some class of applications in a more accurate manner.

7. A Simple Hardware Implementation of Random Networks

In addition to being an analytical tool for the study of neural networks via the results provided in Theorems 1 to 6, random networks can also be implemented in hardware as shown in Figure 3.

Each neuron is simulated using a Counter and a Random Number Generator (RNG). The latter is used to simulate the exponential delays associated with the signal emission process, as well as to handle the signal routing probabilities. The interconnection network receives signals, which can be viewed as short packets of data containing the signal's polarity and its destination, and routes the signal to the appropriate output line connected to the neuron to which it is addressed. Alternatively, all routing information and signal polarity (including the random generation of the destination) can be handled by the interconnection network. This network is programmable so as to represent different neural networks. External signal sources can be implemented simply as permanently activated counters which constantly generate signals at their output. Obviously, as with any highly parallel machine implementation, the complexity of the system is concealed in the interconnection network.

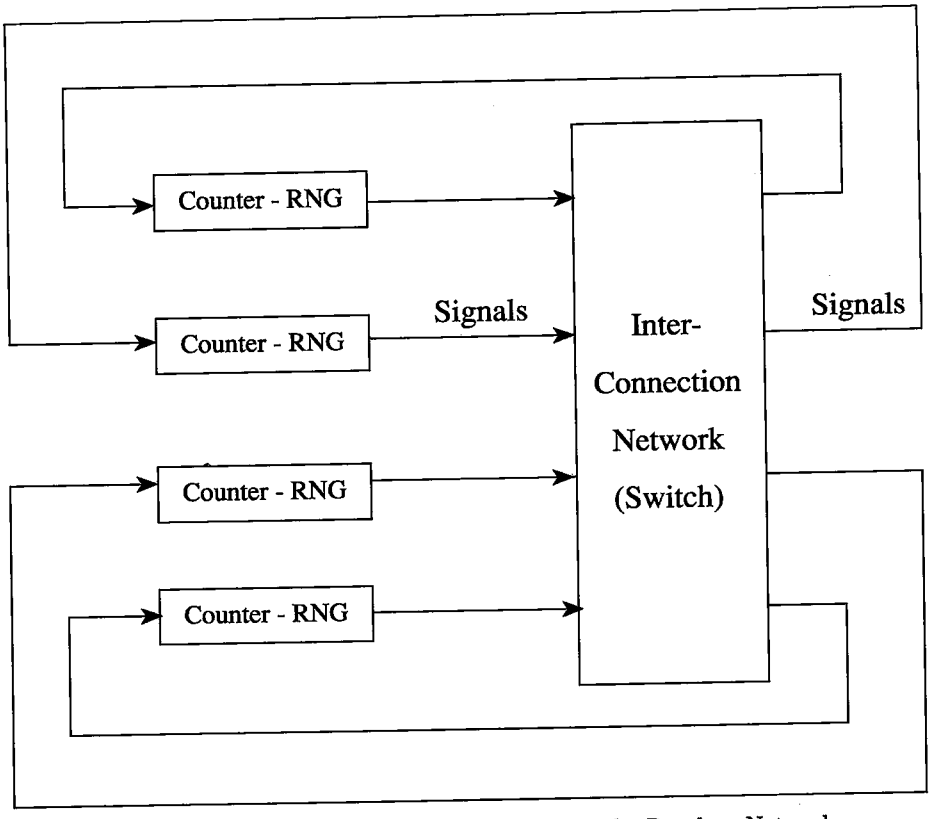

Figure 3. A possible hardware implementation of a Random Network

Acknowledgements: The author gratefully acknowledges the partial support for this research of Pôle Algorithmique Répartie, C3 CNRS, the French National Program in Distributed Computing.

REFERENCES

[1] Gelenbe, E. "Random neural networks with negative and positive signals and product form solution", *Neural Computation*, Vol.1, No.4, 1989.

[2] Gelenbe, E. "Stability of the random neural network model", *Neural Computation*, Vol.2, No.2, 1990.

[3] Fourneau, J.M., Gelenbe, E. "Random neural networks with multiple classes of signals", submitted for publication.

[4] Rumelhart, D.E., McClelland, J.L. and the PDP Research Group "Parallel distributed processing Vols. I and II", Bradford Books and MIT Press, Cambridge, Mass., 1986.

[5] Kandel, E.C., Schwartz, J.H. "Principles of Neural Science", Elsevier, Amsterdam, 1985.

[6] Garcia, C.D., Zangwill, W.I. "Pathways to Solutions, Fixed Points, and Equilibria", Prentice Hall, Englewood Cliffs, N.J. (1981).

[7] Sejnowski, J.T. "Skeleton fields in the brain" in Hinton, G.E., Anderson, J.A. Ed. Parallel Models of Associative Memory, Lawrence Elbaum Associates Publishers, Hillsdale, N.J., 1981.

[8] Gelenbe, E., Koubi, V. "Note on random networks and combinatorial optimization", in preparation.

[9] Hopfield, J.J., Tank, D.W. "Neural computation in combinatorial optimization problems", *Biological Cybernetics*, Vol.52, 141-152 (1985).