A Formal Model for Definition and Simulation of Generic Neural Networks

Article *in* Neural Processing Letters · April 2000

DOI: 10.1023/A:1009678528953 · Source: dx.doi.org

3 authors:

Miguel Atencia
University of Malaga
55 PUBLICATIONS **390** CITATIONS
SEE PROFILE

Gonzalo Joya
University of Malaga
84 PUBLICATIONS **545** CITATIONS
SEE PROFILE

Francisco Sandoval
University of Malaga
362 PUBLICATIONS **2,154** CITATIONS
SEE PROFILE


Neural Processing Letters **11:** 87–105, 2000.

A Formal Model for Definition and Simulation of Generic Neural Networks

M. A. ATENCIA[1], G. JOYA[2] and F. SANDOVAL[2]
[1]*Departamento de Lenguajes y Ciencias de la Computación, E.T.S.I. Informática, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain. E-mail: matencia@lcc.uma.es*
[2]*Departamento de Tecnología Electrónica, E.T.S.I. Telecomunicación, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain*

Abstract. This paper presents the definition of a formal data structure, which assists in the characterization of any neural paradigm, with no restriction, including higher-order networks. Within this model, a neural network is mathematically described by specifying some static parameters (number of neurons, order) as well as a set of statistical distributions (which we call the network 'dynamics'). Once a concrete set of distributions is defined, a single algorithm can simulate any neural paradigm. The presented structure assists in an exhaustive and precise description of the network characteristics and the simulation parameters, providing us with a unified criterion for comparing models and evaluating proposed systems. Though not presented here, the formal model has inspired a software simulator, which implements any system defined according to this structure, thus facilitating the analysis and modelling of neuronal paradigms.

Key words: abstract data type, artificial neural networks, backpropagation, computer simulation, data structure, higher-order, Hopfield paradigm, system modelling

1. Introduction

Any proposed neural solution should fulfil some requirements: easy simulation, quantitative evaluation and possibility of comparison with other alternatives. However, each paradigm is specified by a different terminology, sometimes with qualitative descriptions of the network characteristics; with respect to simulations, they often depend upon ad-hoc programming and the implementation details are seldom completely specified. These two facts obstruct the evaluation and comparison among paradigms. To avoid these problems, we consider that it is necessary to have a formal structure available, which grants the description of both the topological and dynamical characteristics of any neuronal paradigm. The specification of this structure must be accomplished with a precise language, which includes mechanisms for the representation of parallelism and synchronization processes, typical in neural networks. Also, this structure must be easily implemented through a simulation software, which is desirable to ease the following processes:

- Modification or extension of the model without reprogramming.
- Interactive evaluation of every small modification of the model. Obtainment of information about the behaviour of the model.
- For instance, in the case of recurrent networks, we may be interested in observing the stability (or instability) of the network.

This paper presents a data structure that fulfils the mentioned requirements. The specification of the network dynamical properties is performed by means of probability distributions; each network is described by a specific choice of the set of distributions. We name the network state 'configuration'; the configuration evolution is obtained as the application of an algorithm, with a fixed definition, but whose result depends on the concrete set of distributions that the network dynamics comprises. Unlike other models [12, 13] we have aimed to achieve maximum generality in the neuronal paradigms that may be represented by the formal data structure. We have also implemented a software package [1] that simulates the configuration evolution starting from the static (order, number of neurons) and dynamic (probability distributions) properties of the network. Contrary to other simulators [2] these properties need not to be chosen among a previously established set, but they are defined by the modeller.

In Section 2 we define the characteristics of a neural network that are relevant for our work: configuration and dynamics. The configuration is the set of variables that embody all the information about the network state at a given instant. The network dynamics is a set of functions and probability distributions that determine the dynamical behaviour of the network, that is, given a configuration, the dynamics governs the process of obtaining a new configuration. In Section 3, the network evolution is described as the change of its configuration, according to an algorithm that is applicable to every network; however, the concrete behaviour of this algorithm depends on the concrete network dynamics. The algorithm comprises several processes, which, in order of complexity, are: activation, step, simulation phase and experiment. An activation implies firing (and possibly change of state) of a single neuron. A step is a sequence of activations, and represents a simultaneous change of state of a set of neurons. A simulation phase is a succession of steps, and is the largest process that a network may undergo, as long as there are no changes in weights (apart from their initialization). An experiment is a sequence of phases, and it may involve changes in weights, as well as in neuron states. These concepts are clarified with their application to two neuronal paradigms: the Hopfield model and the multilayer perceptron. Finally, Section 4 summarizes the conclusions and presents some fields of future work. In Appendices A.1 and A.2 the paradigms that have been used as an example are briefly described. A summary of notation is included at the end.

2. Neural Network: Configuration and Dynamics

The study of a neural network through its computer simulation suggests the following definition: a neural network is a dynamical system whose state evolves along a discrete temporal scale. This evolution will be formulated as successive transformation from a configuration into another, according to an algorithm. As it will be exposed in Section 3, even though this algorithm is fixed, its application makes use of some parameters (which are probability distributions) that are differently defined for each concrete network. In this way, any neural network may be mathematically defined by specifying its static parameters (number of neurons, order) and its dynamical properties (probability distributions). And the simulation of the network may be achieved with a single algorithmic scheme.

2.1. DEFINITION: CONFIGURATION

A neural network is a dynamical system whose state, at a given instant, is characterized by a 5-tuple C, named *configuration*.

$$C = \{n, m, \mathbf{s}, \boldsymbol{W}, a\}$$

where

n is named *number of neurons*, $n \in \mathbb{N}$

m is named *order*, $m \in \mathbb{N}$

s is a vector named *vector of states*, $\boldsymbol{s} = (s_0 \ldots s_{n-1})$, $\mathrm{s}_i \in \mathbb{R}$, $\forall i \in \{0 \ldots n-1\}$

$\boldsymbol{W}$ is called collection of weights, $\boldsymbol{W} = (W_0 \ldots W_{m-1})$ and W_j is the *collection of weights of order j*, which comprises n^j weights:

$$W_j = \left\{ \begin{array}{cccc} w_{00\underset{j}{\cdots}000} & w_{00\underset{j}{\cdots}001} & \cdots & w_{00\underset{j}{\cdots}00n-1} \\ w_{00\underset{j}{\cdots}010} & w_{00\underset{j}{\cdots}011} & \cdots & w_{00\underset{j}{\cdots}01n-1} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ w_{00\underset{j}{\cdots}0n-10} & w_{00\underset{j}{\cdots}n-11} & \cdots & w_{00\underset{j}{\cdots}0n-1n-1} \\ & & \vdots\vdots & \\ w_{00\underset{j}{\cdots}100} & w_{00\underset{j}{\cdots}101} & & w_{00\underset{j}{\cdots}1n-1n-1} \\ \vdots & \vdots & \vdots\vdots & \vdots \\ \vdots & \vdots & \vdots\vdots & \vdots \\ w_{n-1n-1\underset{j}{\cdots}n-10} & w_{n-1n-1\underset{j}{\cdots}n-11} & \cdots & w_{0n-1n-1\underset{j}{\cdots}n-1n-1} \end{array} \right\}$$

$w_{i_0 \ldots i_j} \in \mathbb{R}, \forall i_k \in \{0 \ldots n-1\}, k \in \{0 \ldots j\}$

Zero-order weights, members of the collection W_0, are often called bias o thresholds.

a is called latest activated neuron, $a \in \{0 \ldots n-1\}$

We will note the set of all possible configurations with symbol $\mathfrak{G}$.

2.2. NOMENCLATURE: PARAMETRIC PROBABILITY DISTRIBUTIONS

Given a set A, let θ be a probability distribution depending on a parameter x $\in$ A, and defined on a set B; that is, given an element $x \in$ A, θ assigns a probability $P_x(y)$ to each element $y \in B$. In these conditions, we can perform a drawing of lots among elements in B, so that each element $y \in B$ may be chosen with probability $P_x(y)$; with the symbol $\theta(x)$ we designate the **result of a concrete execution** of such drawing. With the symbol $\mathscr{P}$(B: A) we designate the class of all probability distributions on a set B, as long as they are dependent on a parameter $x \in$ A, that is, if $\theta \in \mathscr{P}$(B: A), then $\theta(x) \in$ B, where $x \in$ A. These drawing processes may be seen as the execution of an algorithm or, using terminology of the programming languages field, a procedure with parameters. If the distribution θ on the set B has no parameters, we will write $\theta \in \mathscr{P}$(B), and the execution of the drawing process produces a result $\theta(\,) \in$ B, which is equivalent to the execution of a procedure without parameters.

Moreover, the evaluation of a function f: A $\rightarrow$ B is also equivalent to the application of an algorithm which, given $x \in$ A, produces an element $y \in$ B, but now it is the case of a deterministic process, as the value $y = f(x)$ is the same whenever the algorithm is executed. Thus, given a function f: A $\rightarrow$ B we adopt the convention of being $f \in \mathscr{P}$(B: A), unifying the nomenclature between probability distributions (stochastic processes) and functions (deterministic processes).

For instance, the functions or probability distributions that characterize the dynamics of a neural network are usually dependent on the current configuration of the network, that is, they belong to $\mathscr{P}$(B: $\mathfrak{G}$) and the set B will be differently defined in each case. Using this nomenclature, we define the following auxiliary probability distributions which will be used below:

> $v \in \mathscr{P}$(I) is the uniform distribution on the interval I = [0,1]
> $\varphi \in \mathscr{P}(\{0, \ldots i-1\} : \mathbb{N})$ is the discrete probability distribution that, given $i \in \mathbb{N}$, assigns to each integer $0, \ldots i-1$ the probability $1/i$.

It is worth noting that in a parametric distribution, not only the probabilities but also the definition set may depend on the $\mathscr{P}$ parameter. That is, we may a have a distribution $\theta \in \mathscr{P}(\mathrm{B}_x$: A), where $x \in$ A, and $\mathrm{B}_x \neq \mathrm{B}_y$ if $x \neq y$. The above defined distribution φ is an example of this situation, as the set $\{0 \ldots i-1\}$ depends on the value $i \in \mathbb{N}$ that we are considering. We will not use any particular notation to remark this fact, that will be easily deduced from the context.

2.3. DEFINITION: DYNAMICS

The evolution of a neural net (transformation of a configuration into another one) is achieved by means of the application of a fixed algorithm, whose results depend on the evaluation of the probability distributions which have been defined for each concrete network. A data structure which comprises these distributions is named *network dynamics*. Given a neural network, its dynamics are described by the 8-tuple

$D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$. The distributions that compose D are described below; many of them use the current configuration of the network as a parameter, so from now on we will suppose a configuration $C \in \mathfrak{G}$, defined as $C = \{n, m, \boldsymbol{W}, \boldsymbol{s}, a\}$.

$\theta_{sl} \in \mathscr{P}(\{0 \ldots n-1\}: \mathfrak{G})$ is a probability distribution called **selection**. Given a configuration $C \in \mathfrak{G}$, $\theta_{sl}(C)$ will be used as a mechanism of choice of the neuron that must activate at that instant. For instance, if we define $\theta_{sl}(C) = (a+1)\ mod\ n$, neurons will fire sequentially in a cyclic way, as in the multilayer perceptron (Appendix A.2); with $\theta_{sl}(C) = \varphi(n)$, neurons will activate randomly, as in asynchronous feedback networks (Appendix A.1). (The distribution $\varphi(n)$ was defined in paragraph 2.2 as the uniform discrete distribution on the set $\{0 \ldots n-1\}$).

$\theta_{pt} \in \mathscr{P}(\mathbb{R}: \mathfrak{G} \times \{0 \ldots n-1\})$, is a probability distribution named neuron input or **potential**. When a neuron is selected for firing, let us call it i, $\theta_{pt}(C, i)$ will calculate the input to the activated neuron. For instance, in a first order network, a typical definition is $\theta_{pt}(c, i) = \sum_{j=0}^{n-1} w_{ij} s_j$. This is the case of examples in Appendices A.1 and A.2.

$\theta_{ac} \in \mathscr{P}(\mathbb{R}: \mathbb{R})$ is a distribution named **activation**. Once a neuron is selected for activation, the distribution θ_{ac} results in the new neuron state, having the input of the neuron as a parameter. Usually, θ_{ac} is a monotonically increasing function, e.g. $\theta_{ac}(x) = tanh(x)$, as in the examples of Appendices A.1 and A.2.

$\theta_{sn} \in \mathscr{P}(\{0, 1\}: \mathfrak{G})$ is a distribution which will be designated synchronism or **step decision**. θ_{sn} will be used to simulate synchronism, because in our model we only consider the activation of a single neuron at each instant, in order to ease the implementation in sequential computers. Simultaneous activation of several neurons is modelled by means of a *simulation step* (see paragraph 3.2 for a detailed description): neurons are firing in a sequence $\{i_1, i_2, \ldots i_k\}$; when such a configuration C is reached that the evaluation of $\theta_{sn}(C)$ results in the value 1, the step is concluded; this process produces an identical effect to the simultaneous activation of the set of neurons $\{i_1, i_2, \ldots i_k\}$. For instance, in an asynchronous network, no more than one neuron activates at each instant, and this may be simulated by defining a constant value for the *step decision* $\theta_{sn}(C) = 1$, as in a Hopfield network (Appendix A.1). In a layered network, a simulation step must be performed whenever every neuron in a layer has fired. This behaviour is modelled by defining,

$$\theta_{sn}(C) = \begin{cases} 0 & \text{if } layer(a) = layer((a+1) \bmod n) \\ 1 & \text{otherwise} \end{cases},$$

as in the multilayer perceptron shown in Appendix A.2. This example points out the tight interdependence between this distribution and selection θ_{sl}: a correct definition of synchronism is based on the sequential activation of previously defined neurons.

The remaining distributions, that also belong to the definition of network dynamics, correspond to simulation characteristics, rather than intrinsic properties of the network. These characteristics are often specified with a qualitative description, but we think that an accurate definition of them is necessary for a precise analysis and comparison among models.

$\theta_{in} \in \mathscr{P}(\mathbb{R}: \mathfrak{G} \times \{0 \ldots n-1\})$ is a distribution named **state initialization** or, simply, initialization. Given a configuration $C \in \mathfrak{G}$, neuron i will be set for the result of the evaluation of $\theta_{in}(C, i)$ whenever a *simulation phase* starts (see Paragraph 3.3). For instance, if we define $\theta_{in}(C, i) = \nu(\ )$ every neuron initializes its initial value randomly in the interval [0, 1], as in the Hopfield network of Appendix A.1 (according to paragraph 2.2, $\nu(\ )$ is a uniform distribution). In a backpropagation network, as the multilayer perceptron of Appendix A.2, we define

$$\theta_{in} = \begin{cases} p_i^j & \text{if } l\text{ayer}(i) = 0 \\ s_i & \text{otherwise} \end{cases},$$

where $P^j = (p_1^j \ldots p_{n_0}^j)$ is the pattern which is currently being presented to the network; i.e. neurons in the input layer hold the pattern, and any other neuron remains with the same value it reached at the end of the previous simulation phase.

$\theta_{te} \in \mathscr{P}(\{0, 1\}: \mathfrak{G})$ is a distribution named termination or **halt condition**. For each configuration $C \in \mathfrak{G}$ reached after a simulation step is completed, $\theta_{te}(C)$ is calculated: if (and only if) it results in the value 1, the simulation phase stops (see Paragraph 3.3). For instance, in Appendix A.1 we consider a Hopfield network whose neurons have values on the interval [0, 1] and the solutions (final state) must be in the set $\{0, 1\}$; the halt condition may be defined as every neuron state being less than ε distant from any of the extremes of the interval, which may be implemented as $\theta_{te}(C) = \min_i(dif(s_i))$, where

$$dif(s_i) = \begin{cases} 1 & \text{if } s_i < \varepsilon \text{ or } s_i > 1 - \varepsilon \\ 0 & \text{otherwise} \end{cases}.$$

If the network simulation stops when all neurons have been sequentially activated, we define

$$\theta_{te} = \begin{cases} 1 & \text{if } a = n - 1 \\ 0 & \text{otherwise} \end{cases}$$

as in the multilayer perceptron (Appendix A.2).

$\theta_{re} \in \mathscr{P}(\{0, 1\}: \mathfrak{G})$ is a distribution named **repetition condition**. For each configuration $C \in \mathfrak{G}$ reached after a simulation phase is completed, $\theta_{re}(C)$ is calculated: if (and only if) it results in the value 1, a new simulation phase is undertaken, at the beginning of which initialization of states and weights may be performed. In Paragraphs 3.3 and 3.4 the concepts *simulation phase* and *experiment* (sequence of simulation phases) are described in full detail. If the network simulation consists of a single simulation phase, e.g. Hopfield network of Appendix A.1, this distribution must be constantly defined as $\theta_{re}(C) = 0$. However, in a multilayer perceptron (Appendix A.2) the simulation must restart while training patterns remain, and, also, until the network is correctly trained for these patterns (except for an error ε). This

behaviour is implemented by defining

$$\theta_{re} = \begin{cases} 0 & \text{if no patterns remain and } Error < \varepsilon \\ 1 & \text{otherwise} \end{cases}.$$

θ_W is a tuple named **weight** initializations, defined as $\theta_W = \{\theta_{w_0}, \theta_{w_1} \ldots \theta_{w_m}\}$ where $\theta_{w_i} \in \mathscr{P}(\mathbb{R}: \mathfrak{G} \times \{0 \ldots n-1\}^{j+1}) \forall w_i$. The collection of distributions θ_W embodies the process that calculates the weights of a neural network starting from a configuration C, assigning to the j-order weights the value $w_{i_0 \ldots i_{j+1}} = \theta_{w_j}(C, i_0, \ldots i_{j+1})$. In a Hopfield network, weights are calculated just once, by identifying the function to be minimized with the energy function (see, for instance, Appendix A.1). On the other hand, in a network trained with backpropagation, weights are randomly initialized and are modified after every simulation phase, according to formulae shown in Appendix A.2. This latter example shows the need for a process that comprises several simulation phases, which we name *experiment*: as we will see in Paragraph 3.3, weight calculation is only accomplished at the beginning of a simulation phase, not during the phase, so there must exist several phases for a training process (which is a weight change mechanism) to be implemented.

3. Network Evolution: Transformation of Configurations

Once the number of neurons, the order, and the dynamics of a neural network have been defined, its evolution may be simulated by means of a fixed algorithm, consisting in the application of several processes that transform the network configuration. These processed are, ordered according to their complexity:

- Activation
- Simulation step
- Simulation phase
- Experiment

The algorithmic definition of these processes is permanent, but it involves the probability distributions that comprise the network dynamics. Thus, the result of the execution of this algorithm depends on the concrete neural network which we are simulating; moreover, the components of the dynamics are probability distributions, so successive executions of the simulation may produce different results. Our aim is a single algorithmic scheme that is able to simulate any neural network, whose behaviour is accurately defined.

3.1. ACTIVATION

We define an activation as a process that starts from an 'actual' configuration C^i and a 'working' configuration C^w, and results in a new configuration C^o which is identical to C^w, except for the state of a single neuron i. The neuron i that is being activated is picked according to the result of the selection distribution θ_{sl} which receives as a

parameter the working configuration; the state of neuron i is calculated by the potential θ_{pt} and the activation θ_{ac} using as a parameter the actual configuration C^i. This formalism is adopted so that the new neuron states, resulting from several successive activations, do not alter the actual configuration, instead they are stored in the working configuration. When a simulation step is completed, the actual state vector is updated, producing the same result as the simultaneous activation of several neurons (see Paragraph 3.2 below for definition of step and examples).

DEFINITION: ACTIVATION
Let:

D be a dynamics, $D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$
C^i be a configuration, $C^i = \{n, m, \boldsymbol{W}, \boldsymbol{s^i}, a^i\}$, $\boldsymbol{s^i} = (s_j^i)$
C^w be a configuration, $C^w = \{n, m, \boldsymbol{W}, \boldsymbol{s^w}, a^w\}$, $\boldsymbol{s^w} = (s_j^w)$

An **activation**, $A_D(C^i, C^w)$, is defined as the process of obtaining a new configuration: $A_D(C^i, C^w) = C^\circ = n, m, \boldsymbol{W}, \boldsymbol{s^o}, a^o, \boldsymbol{s^o} = (s_j^o)$, where

$$a^o = \theta_{sl}(C^w)$$

$$s_j^o = \begin{cases} \theta_{ac}(\theta_{pt}(C^i, a^w)) & \text{if } j = a^o \\ s_j^w & \text{if } j \neq a^o \end{cases}$$

3.2. SIMULATION STEP

A simulation step is a formal concept that aims to represent the simultaneous activation of several neurons, a parallel process, in a sequential computer or programming language. A simulation step may be defined as the successive computation of the state of one or more neurons, starting from an initial configuration C, but the calculated states do not influence the calculation of the state of subsequent neurons. Instead, the calculated states are stored in successive configurations C', C''... which may be called 'working configurations'. Only at the end of the step, a resulting configuration with every updated state is obtained.

The dynamical description of a neural network must determine the sets of neurons that activate simultaneously at each instant. In this work, this is accomplished by means of functions, so the definition of step results in the following formal mechanism: the step condition θ_{sn} is evaluated after every activation; if it produces the value 1, the step finishes; otherwise, a new activation occurs. In this way, the different classes of networks may be represented with adequate definitions of θ_{sn} and the selection distribution θ_{sl}, for instance:

- Asynchronous networks: $\theta_{sn}(C) = 1$ for every C: every activation involves a step. The result is equivalent to the asynchronous activation of the neurons. It is the case of Hopfield network, as in Appendix A.1.

- Synchronous networks:

$$\theta_{sn}(C) = \begin{cases} 1 & \text{if } a = n-1 \\ 0 & \text{if } a \neq n-1 \end{cases}$$

$$\theta_{sl}(C) = (a+1) \bmod n$$

Neuron activation occurs in a cyclic way, but new states pass to the output only after every cycle, that is, with our terminology, a step is completed after the activation of neuron $n-1$. The same result is obtained if all neurons activate simultaneously.

- Layered networks:

$$\theta_{sn}(C) = \begin{cases} 1 & \text{if } a \text{ is the last neuron of a layer} \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_{sl}(C) = (a+1) \bmod n$$

Neuron activation occurs in a cyclic way, but unless the previous case, a simulation step is completed whenever all neurons in a layer have been activated. The same result is obtained if all neurons in a layer activate simultaneously. This is the definition used for the multilayer perceptron in Appendix A.2.

DEFINITION: STEP
Let:

D be a dynamics, $D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$
C^i be a configuration, $C^i = \{n, m, \boldsymbol{W}, \boldsymbol{s^i}, a^i\}$, $\boldsymbol{s^i} = (s^i_j)$

A **simulation step**, $\mathrm{S_D}(C^i)$, is defined as the process of obtaining a new configuration: $\mathrm{S_D}(C^i) = C^o = n, m, \boldsymbol{W}, \boldsymbol{s^o}, a^o, \mathbf{s}^o = (s^o_j)$, by means of the following sequence, that finishes when the evaluation of the step condition results in value 1:

$$\begin{aligned} C^1 &= \mathrm{A_D}(C^i, C^i) & \theta_{sn}(C^1) &= 0 \\ C^2 &= \mathrm{A_D}(C^i, C^1) & \theta_{sn}(C^2) &= 0 \\ &\cdots \\ C^{p-1} &= \mathrm{A_D}(C^i, C^{p-2}) & \theta_{sn}(C_{p-1}) &= 0 \\ C^p &= \mathrm{A_D}(C^i, C^{p-1}) & \theta_{sn}(C_p) &= 1 \end{aligned}$$

The obtained configuration is $\mathrm{C}^o = \mathrm{C}^p$.

3.3. SIMULATION PHASE

A simulation phase is a process that transforms a configuration into another and consists of two stages: a initialization procedure, and a sequence of simulation steps that finishes when either:

- The network has reached a stable state, or
- The evaluation of the halt condition θ_{te} results in the value 1.

A network is considered to reach a stable state if every neuron has been selected at least once in an activation process, but there has not been any modification of the state vector. We will use some previous definitions.

Given the step $S_D(C)$, with working configurations C^1, C^2, ... C^p, the subset of $\{0, \ldots n-1\}$ that includes the subindices of neurons that have been activated at least once, is called **activated set**, and is represented by the symbol $AC(S_D(C))$:

$$\mathrm{AC}(\mathrm{S_D}(C)) = \bigcup_{j=1}^{p} \{a^j\}, \text{ where } C^j = \{n, m, \boldsymbol{W}, \boldsymbol{s^j}, a^j\}$$

Given the step $S_D(C)$, and a set $E_0 \subset \{0 \ldots n-1\}$ (activated set before the step), the activated set after the step is called **stable-activated set**, $SA(S_D(C), E_0) \subset \{0 \ldots n-1\}$, and it is defined as the union of E_0 and the activated set of $S_D(C)$, if the step has not produced a modification of the network state; otherwise, the stable-activated set is the empty set, that is:

$$\mathrm{SA}(\mathrm{S_D}(C), \mathrm{E_0}) = \begin{cases} \varnothing & \text{if } \mathbf{s} \neq \mathbf{s}' \\ \mathrm{E_0} \cup \mathrm{AC}(\mathrm{S_D}(C)) & \text{if } \mathbf{s} = \mathbf{s}' \end{cases}$$

If the only criterion to stop the simulation in a feedback network is convergence to a stable state, the halt condition may be defined constant $\theta_{te} = 1$. On the other hand, in feedforward only networks, where the concept of stability is not applicable, the specification of θ_{te} allows for the definition of the criterion to stop the simulation. For instance, in the multilayer perceptron of Appendix A.2, we have defined $\theta_{te} = 1$ if and only if every neuron has been activated.

DEFINITION: SIMULATION PHASE
Let:

D be a dynamics, $D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$
C^i be a configuration, $C^i = \{n, m, \boldsymbol{W}, \boldsymbol{s^i}, a^i\}$, $\boldsymbol{s^i} = (s^i_j)$

A **simulation phase**, $P_D(C^i)$, is defined as the process of obtaining a new configuration $\mathrm{P_D}(\mathrm{C}^i) = C^o = \{n, m, \boldsymbol{W}, \boldsymbol{s^o}, a^o\}$, $\boldsymbol{s^o} = (s^o_j)$, through:

(1) An initialization stage resulting in a configuration $C^1 = \{n, m, \boldsymbol{W^1}, \boldsymbol{s^1}, a^1\}$, $\boldsymbol{W^1} = (W^1_j)$, $j = 0 \ldots m$, $W^1_j = (w^1_{i_0 \ldots i_j})$, $i_k = 0 \ldots n-1$, $\boldsymbol{s^1} = (s^1_i)$
   (a) Weight initialization: $w^1_{i_0 \ldots i_j} = \theta_{w_j}(C^i, i_0, \ldots i_{j+1})$
   (b) State initialization: $s^1_j = \theta_{in}(C^i, j)$

(2) A sequence of simulation steps that finishes when either the network has reached a stable state (that is, the stable-activated set comprises every neuron) or the evaluation of the halt condition results in value 1:

$$
\begin{aligned}
&\mathrm{E}_1 = \varnothing \\
&C^2 = \mathrm{S_D}(C^1), \quad \mathrm{E}_2 = \mathrm{SA}(\mathrm{S_D}(C^1), \mathrm{E}_1), && \mathrm{E}_2 \neq \{0 \ldots n-1\} \text{ and } \theta_{te}(C^2) = 0 \\
&C^3 = \mathrm{S_D}(C^2), \quad \mathrm{E}_3 = \mathrm{SA}(\mathrm{S_D}(C^2), \mathrm{E}_2), && \mathrm{E}_3 \neq \{0 \ldots n-1\} \text{ and } \theta_{te}(C^3) = 0 \\
&\ldots \\
&\mathrm{C}^{p-1} = \mathrm{S_D}(\mathrm{C}^{p-2}), \ \mathrm{E}_{p-1} = \mathrm{SA}(\mathrm{S_D}(\mathrm{C}^{p-2}), \mathrm{E}_{p-2}) && \mathrm{E}_{p-1} \neq \{0 \ldots n-1\} \text{ and } \theta_{te}(\mathrm{C}^{p-1}) = 0 \\
&\mathrm{C}_p = \mathrm{S_D}(\mathrm{C}^{p-1}), \ \mathrm{E}_p = \mathrm{SA}(\mathrm{S_D}(\mathrm{C}^{p-1}), \mathrm{E}_{p-1}) && \mathrm{E}_p \neq \{0 \ldots n-1\} \text{ or } \theta_{te}(\mathrm{C}^p) = 1
\end{aligned}
$$

The obtained configuration is $\mathrm{P_D}(\mathrm{C}^i) = C^o = C^p$

3.4. EXPERIMENT

The transformations processes defined so far, allow for the complete representation of neuronal paradigms which do not involve change in weights, such as the Hopfield network. The models that include some learning mechanism must be simulated using a new process, which we call experiment. An experiment is defined as a sequence of one or more simulation phases; at the end of every phase, the repetition condition θ_{re} is evaluated and, if it returns value 1, a new phase is executed. For instance, in the multilayer perceptron (Appendix A.2) a simulation phase is executed whenever a new training pattern is presented. Thus, new simulation phases must be performed while patterns exist. Moreover, if all patterns have been presented and the error is significant yet, a new presentation of all patterns must be accomplished. Taking into account these considerations, the following definition is adopted:

$$
\theta_{re} = \begin{cases} 0 & \text{if no patterns left and } \mathit{Error} < \varepsilon \\ 1 & \text{otherwise} \end{cases}
$$

At the beginning of each simulation phase, as mentioned above, initialization of weights and states may be performed. This can be used to implement models that involve modification of some parameters of the network, such as the slope of the activation function [8, 9].

DEFINITION: EXPERIMENT
Let:

D be a dynamics, $D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$
NN be a neural network with n neurons and order m.

We define an experiment, and denote $\mathrm{E_D}(n, m)$, as the obtainment of a final configuration $\mathrm{E_D}(n, m) = C^o = \{n, m, \boldsymbol{W}, \boldsymbol{s}^o, a^o\}, \boldsymbol{s}^o = (s_j^o)$, through the following procedure, that finishes when the evaluation of the repetition condition results in value 0:

$C^1 = \{n, m, \boldsymbol{W^1}, \boldsymbol{s^1}, a^1\}$, where

$$\boldsymbol{s^1} = (s_j^1), s_j^1 = 0, j = 0 \ldots n-1$$
$$\boldsymbol{W^1} = (W_j^1), j = 0 \ldots m,\ W_j^1 = (w_{i_0 \ldots i_j}^1),\ w_{i_0 \ldots i_j}^1 = 0,\ i_k = 0 \ldots n-1$$
$$a^1 = 0$$

$$C^2 = \mathrm{P_D}(C^1), \qquad \theta_{re}(C^2) = 1$$
$$C^3 = \mathrm{P_D}(C^2), \qquad \theta_{re}(C^3) = 1$$
$$\ldots$$
$$C^{p-1} = \mathrm{P_D}(C^{p-2}) \qquad \theta_{re}(C^{p-1}) = 1$$
$$C^p = \mathrm{P_D}(C^{p-1}) \qquad \theta_{re}(C^p) = 0$$

The obtained configuration is $\mathrm{E_D}(n, m) = C^o = C^p$.

4. Conclusions

We have presented a model for definition of neural networks with a comprehensive scope. No restriction is a priori imposed on the network features. A neural network is completely and precisely described by specifying:

- Number of neurons.
- Order.
- Dynamics, which is a set of probability distributions or functions. Its definition characterizes the behaviour of the described paradigm.

The evolution of any network is formally described by means of a fixed algorithm, which is composed of these processes:

- Activation, which corresponds to the firing of a single neuron.
- Simulation step, which is a sequence of one or more activations.
- Simulation phase, which is a sequence of usually many steps.
- Experiment, which is a sequence of one or more phases.

The result of this algorithm depends on the concrete definition of the distributions which the network dynamics comprises. The specification of the network characteristics through a set of mathematical objects is a straightforward and precise method, which eases a complete and objective description of the network, so facilitating its simulation and interactive study, as well as its evaluation and comparison with other models. This open and flexible structure eases the design of hybrid systems with non-neuronal elements, such as neuro-fuzzy and stochastic systems, or inference networks [10] based upon classical artificial intelligence. This formal structure has inspired the implementation of a simulation software [1] that directly performs the simulation of any neural network defined according to the formal model.

Moreover, the simulator includes some additional facilities, such as modification of the network parameters along the simulation, or storing temporal results.

Some directions of future work include:

- Constructive–destructive learning, with the implementation of mechanisms for the modification of topological features of the network (number of neurons and order) that we have considered fixed in this paper.
- Application of the formal model and the simulator to the analytic and statistic study of several neuronal paradigms, specially convergence and stability analysis in Hopfield networks [7].
- Simulation software enhancements, adopting object oriented programming and handling networks of greater size (both order and number of neurons). Moreover, a parallel implementation of the simulator is under study.

Appendices: Examples of Definition of Neuronal Paradigms

A.1. HOPFIELD NETWORK FOR THE SOLUTION OF THE TRAVELLING SALESMAN PROBLEM (TSP)

In a continuous Hopfield network [4] neurons are activated randomly and asynchronously. When neuron i is activated, its state s_i is calculated according to the expression:

$$u_i \sum_{j=0}^{n-1} w_{ij} s_j - w_i; \qquad s_i = g(u_i)$$

where n is the number of neurons, w_{ij} is the weight of the connection from neuron j to neuron i, w_i is the bias or threshold of neuron i and g is a sigmoid-like function, such as the hyperbolic tangent. The network has the following energy function:

$$E = -\sum_{i,j=0}^{n-1} w_{ij} s_i s_j + \sum_{i=0}^{n-1} w_i s_i$$

In the Travelling Salesman Problem (TSP) there are a set of p nodes (cities), linked pairwise by paths of some length. The solution of the problem is a minimal length path that travels through all nodes once and only once, going back to the initial node. This problem, like other optimization problems [6] is NP-complete [3] and its traditional solutions are computationally complex. A first order Hopfield network for the solution of TSP has been proposed which usually provides an acceptable solution, though not always optimum [5]. The network comprises $n = p^2$ neurons arranged in a square matrix with p rows and p columns. When the network reaches a valid solution, one and only one neuron in each row and column is on (value 1) and the rest is off (value 0). The network state is mapped into a solution in the following way: if the neuron at column i and row x is on, it represents that the node $x$is the ith node inside the path. The energy function that the network must

minimize, corresponding to minimum length paths, is

$$E = \frac{A}{2}\sum_x \sum_i \sum_{j\neq i} S_{xi}s_{xj} + \frac{B}{2}\sum_i \sum_x \sum_{x\neq y} s_{xi}s_{yi} + \frac{C}{2}\left(\sum_x \sum_i s_{xi} - p\right)^2 + \frac{D}{2}\sum_x \sum_{y\neq x} \sum_i d_{xy}s_{xi}\left(s_{y,i+1} + s_{y,i-1}\right)$$

where (x, i) is the neuron at row x and column i, s_{xi} is the state of such neuron, A, B and C are positive constants and d_{xy} is the length of the path from city x to city y. To simplify the notation, subindices have been represented modulo n, that is, $s_{yn} = s_{y0}$. When this function is compared with the network energy, the following values for weights are obtained:

$$w_{xi,yj} = -A\delta_{xy}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{xy}) - C - Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1})$$

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

where $w_{xi,yj}$ is the weight of connection from neuron (x, i) to neuron (y, j); and biases are computed by:

$$w_{xi} = -Cp$$

In [5], the following values for constants are proposed: A = B = 500, C = 200, D = 500.

According to the formal model described in this paper, the Hopfield network for the solution of TSP with p nodes is defined as:

$$n = p^2, \; m = 1$$

$$\mathrm{D} = \{\theta_{sl}, \theta_{pl}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$$

$$\theta_{sl}(C) = \varphi(n), \qquad \text{where } C = \{n, m, \boldsymbol{W}, \boldsymbol{s}, a\}$$

$$\theta_{pt}(C, i) = \sum_{j=0}^{n-1} w_{ij}s_j, \qquad \text{where } C = \{n, m, \boldsymbol{W}, \boldsymbol{s}, a\}, \boldsymbol{s} = (s_i), \; i = 0 \ldots n-1$$

$$\theta_{ac}(x) = tanh(x)$$

$$\theta_{sn}(C) = 1$$

$$\theta_{in}(C, i) = v(\,)$$

Simulation finishes when a discrete state is (almost) reached:

$$\theta_{te}(C) = \min_i (dif(s_i)), \text{ where } dif(s_i) = \begin{cases} 1 & \text{if } s_i < \varepsilon \text{ or } s_i > 1 - \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_{re}(C) = 0$$

$$\theta_W = \{\theta_{w_0}, \theta_{w_1}\}$$
$$\theta_{w_0}(C, i) = -200\ p$$
$$\theta_{w_1}(C, i, j) = -500\delta_{xy}(1 - \delta_{zt}) - 500\delta_{ztj}(1 - \delta_{xy}) - 200 - 500d_{xy}(\delta_{t,z+1} + \delta_{t,z-1})$$

where $x = \text{node}(i)$, $z = \text{stage}(i)$, $y = \text{node}(j)$, $t = \text{stage}(j)$, is the transformation of indices of neurons to the bidimensional arrangement that represents the obtained path.

A.2. MULTILAYER PERCEPTRON WITH BACKPROPAGATION LEARNING

Neurons in the multilayer perceptron arrange in layers and neurons in each layer are activated simultaneously. When a neuron i at layer p is activated, its new state is computed according to the expression:

$$u_i \sum_j w_{ij} s_j - w_i; \qquad s_i = tanh(u_i)$$

where the index j in the sum comprises only neurons in layer $p - 1$. The weight values are computed through the backpropagation algorithm: several training patters are presented to the network and after every presentation the weights of the connections to the neurons in the output layer are calculated as:

$$\delta_i = (1 - s_i^2)(d_i - s_i); \qquad \Delta w_{ij} = \mu \delta_i s_j$$

where d_i is the correct value of neuron i for the presented pattern and μ is a parameter called *learning rate*. To compute the connections to neurons in intermediate layers, the error is 'back propagated':

$$\delta_i = (1 - s_i^2) \sum_l w_{li} \delta_l; \qquad \Delta w_{ij} = \mu \delta_i s_j$$

where, if neuron i belongs to layer p, the sum in l includes only neurons in layer $p + 1$. In [11], the value $\mu = 0{,}05$ is suggested.

In connection to the introduced formal model, the simultaneous activation of the neurons of one layer is modelled as a simulation step and the activation of all neurons in the network after the presentation of one pattern is a simulation phase.

The complete definition of a multilayer perceptron with backpropagation learning is as follows:

$$n = \sum_{j=0}^{k-1} n_j,$$

where n_j is the number of neurons in layer j y k is the number of layers.

$$m = 1$$
$$\mathrm{D} = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$$

Let C be a configuration, $C = \{n, m, \boldsymbol{W}, \boldsymbol{s}, a\}$

$$\theta_{sl}(C) = \begin{cases} a+1 & a+1 \quad \text{if} \quad a < n-1 \\ n_0 & \text{otherwise} \end{cases}$$

Note that neurons in the first layer (input layer) are never activated.

$$\theta_{pt}(C, i) = \sum_{j=0}^{n-1} w_{ij} s_j - w_i$$
$$\theta_{ac}(x) = tanh(x)$$
$$\theta_{sn}(C) = \begin{cases} 0 & \text{if } layer(a) = layer((a+1) \bmod n \\ 1 & \text{otherwise} \end{cases}$$

$$\theta_{in}(C, i) = \begin{cases} p_i^j & \text{if } layer(i) = 0 \\ s_i & \text{otherwise} \end{cases}$$

where $P^j = (p_1^j \ldots p_{n_0}^j$ is the input pattern currently being presented to the network.

The simulation phase finishes when the last neuron is activated:

$$\theta_{te}(C) = \begin{cases} 1 & \text{if } a = n-1 \\ 0 & \text{otherwise} \end{cases}$$

A new simulation phase is executed if either the pattern presentation has not finished,

or after the presentation the error is significant yet:

$$\theta_{re}(C) = \begin{cases} 1 & \text{if no patterns left and } Error < \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

$$\theta_W = \{\theta_{w_0}, \theta_{w_1}\}$$

$$\theta_{w_0}(C, i) = \begin{cases} 0 & \text{if } layer(i) \neq \text{layer}(j) + 1 \\ 2v - 1 & \text{if } layer(i) = layer(j) + 1 \text{ and this is the first simulation phase} \\ w_i + 0,05\delta_i(-1) & \text{if } layer(i) = layer(j) + 1 \text{ and this is not the first simulation phase} \end{cases}$$

$$\theta_{w_0}(C, i, j) = \begin{cases} 0 & \text{if } layer(i) \neq \text{layer}(j) + 1 \\ 2v - 1 & \text{if } layer(i) = layer(j) + 1 \text{ and this is the first simulation phase} \\ w_{ij} + 0,05\delta_i\delta_j & \text{if } layer(i) = layer(j) + 1 \text{ and this is not the first simulation phase} \end{cases}$$

$$\text{where } \delta_i = \begin{cases} (1 - s_i^2)(O_i^j - s_i) & \text{if } l\text{layer}(i) = k - 1 \text{ (output layer)} \\ (1 - s_i^2) \sum\limits_{\substack{l \\ layer(l) = \\ layer(i)+1}} w_{li}\delta_l & \text{if } l\text{layer}(i) \neq k - 1 \end{cases}$$

Summary of Notation

C	Configuration of a neural network. It is a 5-tuple: $C = \{n, m, \boldsymbol{s}, \boldsymbol{W}, a\}$.
$\mathfrak{G}$	Class of all possible configurations.
$\mathscr{P}$(B: A)	Class of all probability distributions on the set B, depending on a parameter x ∈ A.
$\mathscr{P}$(B)	Class of all probability distributions on the set B.
$\theta(x)$	If $\theta \in \mathscr{P}$(B: A) and x ∈ A, $\theta(x)$ is the result of the execution of a drawing process among the elements of B, according to the distribution θ with parameter x. If $\theta \in \mathscr{P}$(B), $\theta(x)$ is the result of the execution of a drawing process among the elements of B, according to the distribution θ.
$v(\,)$	Uniform random variable on the interval I = [0,1]. Note that $v \in \mathscr{P}$(I).
$\varphi(i)$	Discrete random variable that, for a given $i \in \mathbb{N}$, assigns to each integer $0, \ldots i-1$ the probability $1/i$. Note that $\varphi \in \mathscr{P}(0 \ldots i-1\}: \mathbb{N})$.
D	Dynamics of a neural network, which comprises the following probability distributions: $D = \{\theta_{sl}, \theta_{pt}, \theta_{ac}, \theta_{sn}, \theta_{in}, \theta_{te}, \theta_{re}, \theta_W\}$.
A_D	Activation process on a network with dynamics D.
S_D	Simulation step on a network with dynamics D.
AC	Activated set.

SA Stable-activated set.
P_D Simulation phase on a network with dynamics D.
E_D Experiment process on a network with dynamics D.

Acknowledgements

This work has been partially supported by Spanish Comisión Interministerial de Cienca y Technología (CICYT), Project No. TIC98-0562.

References

1. Atencia, M. A.: An arbitrary order neural network design and simulation environment, graduating work, Dpto. Tecnología Electrónica, Univ. Málaga (Spain), (in Spanish), 1997.
2. García del Valle, M., García, C., López, F. J. and Acevedo, I.: Generic neural network model and simulation toolkit, In: Mira, Moreno-Díaz and Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, Lecture Notes in Computer Science, No. 1240, Springer-Verlag, (1997), pp. 313–322.
3. Garey, M. R. and Johnson, D. S.: *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, (1979).
4. Hopfield, J. J.: Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Nat. Acad. Sci. U.S.A.,* **81** (1984), 3088–3092.
5. Hopfield, J. J. and Tank, D. W.: Neural computation of decisions in optimization problems, *Biological Cybernetics*, **52** (1985), 141–152.
6. Joya, G., Atencia, M. A. and Sandoval, F.: Application of high-order Hopfield neural networks to the solution of diophantine equations, In: A. Prieto (ed.), *Artificial Neural Networks*, Lecture Notes in Computer Science, No. 540, Springer-Verlag, (1991), pp. 395-400.
7. Joya, G., Atencia, M. A. and Sandoval, F.: Associating arbitrary-order energy functions to an artificial neural network. Implications concerning the resolution of optimization problems, *Neurocomputing*, **14** (1997), 139–156.
8. Joya, G., Atencia, M. A. and Sandoval, F.: Hopfield Neural Network applied to optimization problems: some theoretical and simulation results, In: Mira, Moreno-Díaz and Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, Lecture Notes in Computer Science, No. 1240, Springer-Verlag, (1997), pp. 556–565.
9. Joya, G.: Contributions of high order artificial neural networks to the design of autonomous systems. Doctoral Thesis, Dpto. Tecnología Electrónica, Univ. Málaga. Servicio de Publicaciones e Intercambio Científico de la Universidad de Málaga, (in Spanish), 1997.
10. Mira, J., Herrero, J. C. and Delgado, A. E.: A Generic formulation of neural nets as a model of parallel and self-programming computation, In: Mira, Moreno-Díaz and Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, Lecture Notes in Computer Science, No. 1240, Springer-Verlag, (1997), pp. 195–206.
11. Müller, B. and Reinhardt, J.: Learning Boolean Functions with Back-Propagation, *Neural Networks. An Introduction*, Springer-Verlag, (1990), pp. 222–227.

12. Santos, J., Cabarcos, M., Otero, R. P. and Mira, J.: Parallelization of connectionist models based on a symbolic formalism, In: Mira, Moreno-Díaz and Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, Lecture Notes in Computer Science No. 1240, Springer-Verlag, (1997), pp. 304–312.
13. Strey, A.: EpsiloNN – A specification language for the Efficient Parallel Simulation of Neural Networks, in Mira, Moreno-Díaz and Cabestany (eds.), *Biological and Artificial Computation: From Neuroscience to Technology*, Lecture Notes in Computer Science No. 1240, Springer-Verlag, (1997), pp. 714–722.