Available online at www.sciencedirect.com
ScienceDirect
Journal of Physiology - Paris 00 (2011) 1–10
Journal of Physiology Paris

No clock to rule them all

Wahiba Taouali, Thierry Viéville, Nicolas Rougier, Frédéric Alexandre
INRIA Cortex
http://cortex.loria.fr

Abstract

The hallmark of most artificial neural networks is their intrinsic parallelism where each unit is evaluated concurrently to other units in a distributed way, thus using *asynchronous* mechanisms. However, this notion of asynchronous computation is polymorphic and far from being obvious, as asserted by the huge and somehow contradictory literature on this topic. Facing up the related pragmatic issues, the goal of this article is twofold. On one hand, it precisely clarifies basic key notions related to asynchronous distributed computations. On the other hand, a few practically usable methods and quantitative bounds are made explicit.

Key words: Neural Networks, Central Clock, Dynamic Neural Fields, Asynchronous Computation.

1. Introduction

Brain is mostly event driven ! When a spike is sent along an axon, its processing is taken into account at the level of the dendrite (and later integrated in the soma), when arriving at the synaptic cleft. This triggers a complex chain of bio-chemical processings allowing the signal to be processed via the synapse and to be sent further along the dendrite, etc. There is consequently no need of any kind of central clock (or centralized signal) to coordinate such processing. This is truly distributed and asynchronous. And these properties are enforced anytime over the entire network (a.k.a. the brain). However, this does not prevent some synchronization to happen between neurons as it has been reported in a number of works, but this occurs only on the basis of local interactions without the need of any kind of central clock nor supervisor. If we now turn to the artificial neural networks paradigm, we realize that this asynchronicity property is hardly enforced in any model. Most numerical methods used to simulate the underlying differential evolution equations require an implicit central clock. Even the most complex and elaborated spiking neuron models are doomed with such considerations if they do not rely on explicit event-driven simulations[1]. This is the case for synchronous computations, in which a central mechanism updates each unit (e.g. neuron model) at the same clock-time but less obvious is the fact that this is also the case for some asynchronous paradigms, in which, for instance, a central mechanism randomly draws without replacement the units to sample at a given regular clock time, in order to simulate asynchronous computation. The knowledge of which unit has been sampled or not must be centralized. More generally, usual strategies require a complete information about each part of the current state in order to deliver from a centralized locus the signal of the next step. Furthermore, this implies the random sampling events to occur at very regular times. It is thus implicitly assumed that the time is global to the whole system. Computation is distributed, but the computation time and clock remain centralized.

What are the consequences ? At the computational level, this means that if we are using a multi-processor architecture, processors that finish their task early have to wait, doing nothing, until others finish. The more

[1] See e.g. http://mvaspike.gforge.inria.fr/ for such implementation

synchronization points are set, the more performance degrades. At the system dynamics level, such regular updates may induce spurious synchronization mechanisms. At the biological modeling level, we must assume the existence of a global "universal" clock which is a reasonable approximation for small dynamical systems, but less obvious when considering several cortical maps in interactions with complex connection delays.

The goal of this article is thus to consider literature in relevant domains, namely cellular automata [15, 14, 16, 4] and parallel computations [5, 6], and to make the link with computational neuroscience. More specifically, we will introduce the dynamic neural fields theory that offer a very general computing framework for the modeling of cortical phenomena and since this theory is continuous in both space, time and values, we will pay special attention at the discretization procedure since we aim at using results from the discrete event systems specification to simulate discrete time systems and approximate, as closely as desired, differential equation systems. At this stage, we do not aim at giving a complete state-of-the-art on asynchronous models, but rather to show how a general construct, called here *fully asynchronous paradigm*, provides a constructive answer to asynchronous computation, especially in the particular case of artificial neural networks.

2. General framework

2.1. Dynamic Neural Fields

Let us target generalized neural fields with delayed connection strength that are *tissue level models that describe the spatio-temporal evolution of coarse grained variables such as temporal synchronization or firing rate activity in populations of neurons* as explained by [13]. Such neural fields were primarily introduced by [25] and [2] and are generic enough to give account on a great number of models from the computational neuroscience community. We will consider a single *network* made of several *units* with *spatial connections* between them. The evolution of the activity V of such a network is described by the following differential equation, which may write:

$$\frac{\partial V(\mathbf{x},t)}{\partial t} = -\frac{1}{\tau(\mathbf{x})}\,V(\mathbf{x},t) + h(\mathbf{x},t) + s(\mathbf{x},t) + \int_{\mathcal{M}} d\mathbf{y} \int_0^{+\infty} d\eta\; W(\mathbf{x},\mathbf{y},\eta)\,\sigma_{\mathbf{y}}\left(V(\mathbf{y},t-\eta)\right) \qquad (1)$$

where $\mathbf{x}$ denotes a location onto the manifold M, t is time, $V(\mathbf{x},t)$ denotes the membrane potential of a neural population at point $\mathbf{x}$ and time t, τ is the temporal decay of synapses, i.e., represents a *leak*, η is the transmission delay, W is the (spatio-temporal) synaptic function, $s(\mathbf{x},t)$ is the input received at position $\mathbf{x}$ and h is the mean neuron threshold.

In this framework, $\sigma_{\mathbf{y}}()$ is a smooth sigmoid function that relates the unit state to the mean firing rate. As an extension of this *analog* input/output non-linearity, steep sigmoid profiles can also be introduced in order to encounter for mesoscopic events triggered by some unit state. Furthermore, one mesoscopic unit is not necessarily represented by a scalar value $V(\mathbf{x},t)$, whereas a vectorial state vector could be taken into account. Providing that the leak, now a matrix, is diagonalizable, this representation trivially decomposes in a vector of equation of the form of (1), so that we can keep considering (1), in our context, without loss of generality.

Some dynamic neural field (DNF) models assume that the information velocity is unbounded, thus neglect transmission delays (i.e., $W(\mathbf{x},\mathbf{y},\eta) = w(\mathbf{x},\mathbf{y})\,\delta(\eta - 0)$, with only an "instantaneous" connection). Some consider an unchanged transmission, with delay (e.g., $W(\mathbf{x},\mathbf{y},\eta) = W(\mathbf{x},\mathbf{y})\,\delta(\eta - \frac{|\mathbf{x}-\mathbf{y}|}{v})$ for a propagation at a velocity v). The present formulation subsumes these various cases, taking into account the spatio-temporal nature of the connection.

2.2. Discrete Neural Fields

At this point, it is important to note that such a model possesses three distinct levels of continuity, namely: *space*, *time* and *value* and since such a system cannot be solved analytically in the general case, we have to use numerical methods to solve the corresponding discretized differential equations system. Here we consider the spatial discretization level as an *a priori* and neglect the value discretization (except in subsection 3.2, where it is discussed). This allows us to concentrate on the temporal level. We can now rewrite the equation as the following discrete set of units evolution, indexed by a finite set M:

$$\frac{\Delta V_i(t)}{\Delta t} = -L_i\,V_i(t) + I_i(t) + \sum_{\substack{j\in M \\ k\in\{k_{\min},k_{\max}\}}} W_{ijk}\,\sigma_j\left(V_j(t-k\,\Delta T)\right) \qquad (2)$$

Here $V_i(t) \equiv V(\mathbf{x}_i,t)$ denotes the related mesoscopic state at the sampled location $\mathbf{x}_i$ (namely the spatial average of the membrane potential, for the neural population at that point and time). Furthermore, $L_i \equiv \frac{1}{\tau(\mathbf{x}_i)}$ is the "leak" related to the unit time-constant (namely the average membrane temporal decay of synapses in a

mean-field approach), while $\sigma_j()$ is the same function defined as in (1), and W is the connection strength function. Weights are indexed by the spatial indexes and by the delay index k, i.e. each possible delay between $k_{\min}$ and $k_{\max}$ is taken into account, sampled at ΔT. Finally $I_i(t) \equiv h(\mathbf{x}_i, t) + s(\mathbf{x}_i, t)$ encounters for both the received input $s(\mathbf{x}_i, t)$ and activity threshold $h(\mathbf{x}_i, t)$.

The former equation (1) corresponds to the *original* dynamic neural field (see, e.g. [18] for a recent review) and the latter equation (2) corresponds to a discrete neural assemblies of neurons and we are only going to consider this latter system in the following.

A key point here is the distinction between the *simulation* sampling time Δt (i.e., the time at which the continuous system is discretized) and the *modeling* sampling time ΔT (i.e., the discrete time chosen to model the dynamical system). Choosing to sample $\int_0^{+\infty} d\eta$ by a sum $\sum_{k=k_{\min}}^{k_{\max}}$ at $t = k\,.\Delta T$ is a modeling choice, whereas calculating the $\frac{\partial V(\mathbf{x},t)}{\partial t}$ at some rate of Δt is a simulation issue. Model delays ΔT are simulated at sampling times Δt. Indeed, if considering synchronous computations this distinction is meaningless (the obvious choice is $\Delta t = \Delta T$), whereas it is required for asynchronous computations, as made explicit in the sequel.

2.3. From synchronous to asynchronous computations

In this aforementioned context, synchronous computations would refer to the standard numerical method used to solve a set of ordinary differential equations. After choosing a temporal resolution Δt, any value $V_i(t + \Delta t)$ is evaluated according to $V_i(t)$ and $\Delta V_i(t)/\Delta t$ using one of the numerous implicit or explicit available methods (Euler, Runge-Kutta [20], etc.). Since any value $V_i(t)$ may depend on $V_j(t)$, it is important to update any $V_j(t)$ only once all values $V_i(t + \Delta t)$ are known. This may require the synchronization from a centralized control, signaling units that are allowed to update their *public* state. Even if not all states are required at each step, as for example with the Gauss-Seidel method [5], the central clock is still needed (because we need to ensure exactly one evaluation for any unit), thus the computation remains macroscopically synchronous in this sense.

A fully asynchronous system therefore implies to circumvent such global or centralized clock and to let the system operate under fully distributed control. At a computational level, this would mean that each processor is an independent unit with a local notion of time and is thus updated separately. From a more biological point of view, this would reflect several ideas:

- biological delays related to the cortical map topography, with three facets:
  - fixed delay related to known connection length
  - dynamic delays related to on-going processing or transmission
  - random delays related to uncertainty or lack of knowledge
- local computation effects such as adaptive asynchrony, i.e. the fact that a unit adapts its state with parsimony: the more its value is stable, the less its change has to be output rapidly;
- mesoscopic events such as activity synchronization, rhythms, or sudden activity change.

At this modeling level, where parallel processing is a key aspect of neural computation, we may have different run times and units are not supposed to wait for each other. Additionally, transmission delays contribute to desynchronize the exchanged information. As described here, both phenomena are obvious to take into account in a "fully asynchronous" paradigm. In other words, asynchronism is not only an implementation issue, it is also a modeling issue.

2.4. The Asynchronous model

Following [19], let us propose the following asynchronous computation paradigm. Each processing implementing equation (2) for an index i is referred to as unit.

At a given sampling time of index t, only a subset of arbitrarily chosen units $U(t)$ is evaluated. This very simple scheme includes synchronous relaxation (i.e., $U(t)$ contains all the units), serial or deterministic Gauss-Seidel relaxation (i.e., $U(t)$ contains one unit at each update, each cannot be updated more than once before the whole system is updated), other asynchronous schemes (e.g., $U(t)$ contains one or more units, randomly drawn with or without replacement), etc..

In addition, each connection between two units i and j is supposed to have an implementation delay $\Delta_{ij}(t)$ constant or variable: the information $V_j(t)$ is available to the unit i only after such a delay, indeed different from the simulation delay Δt and modeling delay ΔT.

The key point is the following: each unit does not compute one sample $V_i(t)$ given some updated values, but *calculates an approximation of the whole trajectory*

$\{V_i(t), 0 \leq t\}$, given some delayed knowledge from the connected units $\{\hat{V}_j(t), 0 \leq t < t_{ij}(t), j \in M\}$:

- Initially, each unit only knows its initial value $V_i(0)$, given *a priori*, and must have specified an initial value of the connected units trajectories (e.g., assuming $V_j(t) \simeq V_j(0)$, as best knowledge when nothing is calculated).
- Then, each unit starts estimating an approximation of a part of its related trajectory, up to some time t_i: $\{V_i(t), 0 \leq t < t_i\}$, and communicates the knowledge asynchronously to other units.
- When receiving some knowledge, it updates it own approximation, and so on.

The paradigm thus requires (i) the initial values to match the expected initial values for all units and (ii) each unit to be able to solve the initial value problem specified in (2) (i.e. to calculate a convergent approximation of the solution trajectory). Surprisingly enough, no additional restriction is placed on the implementation. Very clearly here, modeling and simulation times are entirely unlinked. Though this seems to be an intractable paradigm thanks to the specific form of the equation, we are going to make explicit that the estimation process can very efficiently be implemented in this case. The original framework is a bit more general (thus still interesting for generalizations of the present framework) but the present work is precisely to make it specific to the present framework.

3. Before asynchronism: discretization issues

Here space discretization is taken as an a priori, while we discuss time and value discretization issues in this section.

3.1. Time discretization issue

Since symbolic resolution of differential equations such as (2) is not always possible, the evolution of the system can be approximated using numerical integration.

A toy example

Let us first consider the very simple case of a linear constant approximation of the system (2) with initial condition $V_j(0)$, in the particular case where leak L_j, connection strength W_{jk0} and current input I_j are constant, without transmission delays, while $\sigma_j(u) = u$ (see [1] for a discussion of the kind of "sigmoid" profiles usually used). This writes in vectorial form:

$$\frac{\Delta V(t)}{\Delta t} = -\mathbf{W}\,\mathbf{V}(t) + \mathbf{I},$$

with

$$\mathbf{W} \stackrel{\text{def}}{=} \begin{pmatrix} L_1 & -W_{120} & \cdots \\ -W_{210} & L_2 & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}, \; \mathbf{I} \stackrel{\text{def}}{=} \begin{pmatrix} I_1 \\ I_2 \\ \cdots \end{pmatrix}, \quad (3)$$

and its regular sampling forward Euler discretization writes: $\mathbf{V}[i+1] = \mathbf{V}[i] - \Delta t\, \Delta V(t)/\Delta t$, at $t = i\Delta t$.

Here, we have to assume that the system is contracting, i.e., that real part of the eigenvalues of $\mathbf{W}$ are strictly positive, otherwise the system does not converge towards a stable solution, and the Euler-forward approximation method is not expected to converge towards a continuous solution (see e.g. [20] for these elementary notions). In words this means that leak is strong enough with respect to the weights in order to induce the system convergence, see [1] for a detailed study in the case of discrete neural fields. More precisely, on an eigendirection (i.e. in the direction of an eigenvector of the matrix), the linear equation is decoupled from the others and the leak (either a real or a complex value) corresponds to the opposite of the eigenvalue, with solution either as damped oscillations or as an exponential vanishing profile. We do not have to assume that weights are symmetric, but that the matrix $\mathbf{W}$ is diagonalizable, which is always the case up to a negligible singular set, not taken into account here. In such a simple case,[2] both the continuous

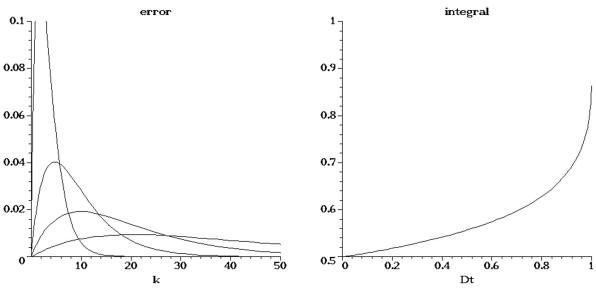

Figure 1: *Left view*: The normalized bias temporal profile between the continuous scheme and its discrete approximation, drawn here for $\Delta t/\tau_j = [0.05, 0.1, 0.2, 0.5]$ from the flattest to the sharpest curve respectively. *Right view*: The integral of the bias along the trajectory as a function of $\Delta t/\tau_j$, making explicit that the cumulative bias is never negligible even for very small leak, while it diverges for large leak. See text for details.

scheme and its discrete approximation starting from the same initial value converge towards the same fixed point (which can be found in all textbooks), but not though the same trajectory (which is surprisingly not studied in text books up to our best knowledge). More precisely the bias in an eigendirection of the **W** matrix is proportional to $V_j(0) - I_j\,\tau_j$, where $1/\tau_j$ is the eigenvalue (i.e., the leak) in this direction, and follows a double exponential profile, only function of $\Delta t/\tau_j$, as illustrated in Fig. 1. In words, the higher $\Delta t/\tau_j \in [0, 1[$ (this boundaries corresponding to the convergence interval), the higher the bias magnitude, but the quicker the bias between both methods vanishes. The highest leak thus determines the maximal bias, the smallest leak the maximal duration of bias.

It is a counter-intuitive and very important result to notice that large leaks (i.e. small time-constants) indeed accelerate convergence, but with the drawback to generate large errors during the first iterations. This means that the system dynamics may very easily switch from one attractor to another, at the beginning of the trajectory, even in such a very simple case.

The general case: consequences

In more general cases, there is no cute closed-form formula, as in the previous toy example, but it is obvious to infer to which extends the same kind of bias occurs.

In the non-constant case (i.e., leak, weights or currents vary with time), the previous results generalize considering bounds of the, now variable, leaks. In the non-linear case, the previous results generalize bounding the non-linear function $\sigma_j()$ by the corresponding maximal slope linear function. See, e.g. [10] for a review of such tools. More precisely, if the system is hyperbolic, the same condition applies on the Jacobian of the system at any time and state value.

If the input variable $I_i(t)$ is not constant, we are thus in the situation where we track the true solution, and this is qualitatively equivalent to be reset at a new initial state at each step. As a consequence for small $\Delta t/\tau_j$ the system is expected to be always in a transient biased state with first iteration large errors, whereas for large $\Delta t/\tau_j$ the system is always going to have a non-negligible lag with respect to the true solution. There is clearly a trade-off to find out, but always impaired by an incompressible bias.

As a consequence, we learn something from the present tiny analysis, even in more general cases: the method never converges at the trajectory level. In any case, the cumulative bias is never negligible, with a lower bound for small leak, while diverging for large leaks. This shows the very important difference between the fact that the discretization methods *converge at least* toward the expected fixed point and the fact that the simulation trajectory is unbiased. In the toy example, unbiasness never occurs, except in the singular case where $V_j(0) = I_j\,\tau_j$. Where stands the "mistake" ? At the implementation level, it stands on the simple fact that we are stick to synchronous computations.

When asynchronism solves the situation

At the implementation level, we obviously need several sampling times for a given simulation time in order to make the discrete approximation converge towards the continuous one, which is often not taken into consideration.

At the modeling level, the present mistake stands on the belief that a pertinent model of the reality has to be a "continuous" model, its discretization being a kind of second-class implementation detail. This is definitely wrong when modeling digital computational systems, but this is also questionable for microscopic neural models (see, e.g., [11] for a discussion on biologically plausible generalized integrate and fire neuron models) and mesoscopic neural map models (see, e.g. [23] for a discussion at this modeling scale), the key question being "what do we want to learn" from the model or its simulation.

Now let us reconsider equation (2) in the asynchronous computation paradigm. We have at time t, estimations of other units $\{\hat{V}_j(t)\}$, and estimations of the input $\hat{I}_i(t)$, and have missing values specified as a first approximation, as discussed in section 2.4. We thus can approximate:

$$\hat{J}_i(t) = \hat{I}_i(t) + \sum_{\substack{j\in M \\ k\in\{k_{\min},k_{\max}\}}} W_{ijk}\,\sigma_j\left(\hat{V}_j(t - k\,\Delta T)\right)$$

[2] In the scalar case, an explicit closed-form is automatically derived from a few lines of, e.g., `maple` symbolic code:

```
eq := D(V)(t) = -W * V(t) + b:  assume(0 < W, W < 1):
# Continuous solution, assuming t0 = 0
s_c := dsolve(eq, V(0) = V0, V(t));
# Euler approximate integration, assuming delta_t = 1
s_e := rsolve(subs(eq, t = k, V(k + 1) = V(k) + Dt * D(V)(t)), V(0) = V0, V);
# Bias analysis
err_k := simplify(factor((subs(s_c, t = Dt * k, V(t))
- subs(s_e, V(k))) / (V0 - i / W)), Dt * W = c);
```

while the result is straightforward to apply to the eigenvalue decomposition of the **W** matrix. Furthermore, in the scalar case, if the Euler approximation is used with $Dt = (1 - exp(-W\,\Delta t))/W = \Delta t - W/2\Delta t^2 + O(\Delta t^2)$ in numerical scheme, the bias is canceled, which is not generalizable in the vectorial case since it depends on the leak value.

and are left with solving:

$$\frac{\Delta V_i(t)}{\Delta t} = -L_i\, V_i(t) + \hat{J}_i(t)$$

with the obvious closed form solution:

$$\hat{V}_i(t) = V_i(0)\, e^{-L_i t} + \int_0^t d\eta\, e^{-L_i(t-\eta)}\, \hat{J}_i(\eta) \qquad (4)$$

which is straightforward to iteratively approximate up to some adjustable time t, at any degree of precision, and easy to update as soon as better estimations of $\hat{J}_i(t)$ are available. Several further implementation choices are to be specified up to this point, but it is clear that all ingredients are there to design an asynchronous implementation of (2).

3.2. Value discretization issues

A Discrete Dynamic System (DDS) is a finite set of elements, each taking a finite number of states evolving in a discrete time, by mutual interactions. In [21], a book dedicated to the analysis of the temporal dynamics of such systems, Robert proposes a general framework, i.e., "a chaotic discrete iteration mode", to analyze such systems. The studied system is a cellular automaton with N Boolean cells (a finite number of states), which may correspond, in a neural network, to an active neuron state (spiking) or a silent state. Each unit[3] is influenced by a subset of units that are involved in its update.

Indeed, though a computer variable takes a finite number of values, the DDS results do not apply to floating-point number representations, because the order of magnitudes of the results are not usable in practice. Furthermore, DDS are not sufficient for modeling the targeted systems, i.e. DNF, that represent physical quantities, thus are best modeled by differential equations with continuous values. However, depending on the level of modeling, DDS could encounter for qualitative behaviors and we include them in the discussion in the sequel.

4. Convergence and guaranty of computation

Let us now report the proof of the convergence of asynchronous algorithms. We are in a *bounded asynchronous framework*, i.e. two main assumptions are required:

- *bounded delays*. The delays should be bounded by a uniform finite constant $\Delta_{ij} \leq D < +\infty$. All sampling delays are thus finite.

- *non starvation condition*. There is a maximal uniform bound B, between two updates for a given unit. Each unit should provide an update at least once during all interval of length B.

Different versions of these assumptions have been assumed in asynchronous computational algorithms like in [12]. Therefore, with some additional conditions on the differential equations system (specially the leak function), uniform convergence at a geometric rate is proved. Let us precisely state the result.

Let us write $\mathbf{W}$ the weight matrix defined in (3) in the case without transmission delay and when $\sigma_j(u) = u$. If transmission delays occur, $\mathbf{W}$ is now a tensorial quantity, and standard methods to rewrite the nth-order system as an augmented dimensional 1st-order system are standard, $\mathbf{W}$ being a matrix in the augmented system. If the non-linearity $\sigma_j(u)$ is present, using differential inequalities, the non-linear system has to be bounded by a linear system of weight $\mathbf{W}'$. This is non trivial and beyond the scope of this short paper, and we refer to [19] for a full development. Despite the technical difficulties, the underlying idea is quite simple: the weight has to be multiplied somehow by the highest slope of the non-linearity, i.e. $W'_{ijd} \simeq \sigma'_j(0)\, W_{ijd}$ for a sigmoid profile. Therefore, though not trivially, the following result holds in the most general case.

Based on this, if we refer to Mitra's work, our differential equation is a particular case of the proposed framework[4]. Applying Proposition 2 of the paper, uniform convergence at a geometric rate in the asynchronous mode occurs when, in our case, $\mathbf{I}-|\mathbf{W}|$ is an M-matrix (i.e. a matrix whose off-diagonal entries are less than or equal to zero, with eigenvalues whose real parts are positive). Since $\mathbf{W}$ is a symmetric positive matrix, thus diagonalizable, with positive eigen-values, $\mathbf{I} - |\mathbf{W}|$ is a M-matrix a soon as these eigen-values are smaller than one. As a consequence, we can conclude that asynchronous computation convergence in such a neural network is guaranteed and that the long-term average rate geometric convergence (per update) is not less than:

$$\frac{1}{\tau_{\min}} \leq \frac{1}{D+B} \qquad (5)$$

[3] More precisely, starting from an initial state at $t = 0$, at each time step each unit updates its state function involving the states of the corresponding subset of elements, using any specific iteration mode (parallel, serial or chaotic).

[4] It corresponds to equation (6.7i) of [19], with $\mathbf{D} = \mathbf{I}$ the (identity matrix) and $\mathbf{B} = \mathbf{W}$.

where $1/\tau_{\min}$ is the spectral radius of $\mathbf{W}$, i.e., the largest (here positive) eigen-value corresponding to the smallest combined leak of the system. This is a very precise and effective result, *with a usable number*.

The main problem, even if the system is finite and time is discrete, is not only to ensure the convergence to a stable state (this obtained here, if the system is contracting), but to guaranty the computation is going to converge to the right result. Here is the power of the Mitra result. In the non-linear case, Mitra has thus to assume that the non-linear functions are continuous (but not necessarily smooth) and bounded by a linear contracting function, to obtain the same result. In our case, this means that the non-linear, so-called sigmoid function $\sigma()$ can not be a step function, but any usual continuous profile is convenient.

Convergence when values are discretized

A step further, let us discuss what can be stated, when values are also discretized. In the numerical analysis community, the reference book in both continuous and discrete context is Bertsekas and Tsitsiklis book, chapters 6 and 7 [6], released in 1989. It presents some algorithms and gives sufficient conditions for convergence for general nonlinear problems and necessary conditions for linear problems. In the special case of discrete dynamic systems without transmission delays, the main result presented in [21, 3], is that if a system is pseudo-periodic, it converges at most after N pseudo-periods. A pseudo-period corresponds to a sequence of events in which each unit is updated at least once. So with only N synchronization points the system convergence is guaranteed. But when we introduce transmission delays and continuous variables, such an assumption is no more valid, thus does not ensure the convergence.

An illustrative example

Let us consider for example a two-layer network, each layer being of size $N \times N$ units. An input layer corresponding to the constant current entry is feeding an output layer. Each unit of index i of the output layer receives its input I_i from the input layer with respect to a receptive field (a Gaussian connection). We suppose that there is neither lateral connection nor feedback in the input map. Lateral connections in the output map are excitatory, decreasing with distance and the input is inhibitory (the reverse connectivity scheme is also suitable). It means that each neuron in the output layer is excited by its neighbors and inhibited by the neurons in its receptive field. The resultant activity in the output map is a difference of Gaussian which is a computation scheme very used in dynamic neural networks.

So, in this case, for each unit of the $M = N \times N$ units, equation (2) writes with weights $W_{i=(u,v)\ j=(u',v')\ d}$ indexed in function of the 2D geometry of the network and V_i stands for the membrane potential, W_{ijd} is a positive symmetric tensor in this case, and I_i is the constant current input, projection from the input layer. One instantiation, without delayed weight, would be to choose:

$$W_{ij0} = a_+ e^{-\frac{|i-j|^2}{v_+^2}} - a_- e^{-\frac{|i-j|^2}{v_-^2}}$$

then as a side result of the work reported in [1], we can state:

$$\frac{1}{\tau_{\min}} \simeq \sqrt{\pi}\,(a_+ v_+ - a_- v_-)$$

the approximation being to neglect the finite side effect of the corresponding 2D map. Then, combined with (5), we have a concrete bound on asynchronous convergence.

As soon as the *sampling* and *simulation* times are not mixed, and the dynamic neural field dynamics attractor is a fixed point, attained with or without damped oscillations, any general asynchronous relaxations schemes with neither unbounded transmission delays, nor starvation, uniformly converge at a geometric rate. This result has obviously no reason to hold for more complex dynamics, especially in chaotic cases, since even a negligible error is going to make the discrete approximation diverge exponentially fast, without any chance to redress the error by a bounded number of asynchronous relaxations. In the case of a periodic stable attractor, though the previous formalism can not be applied as it is, it seems reasonable to assume that the asynchronous relaxations would be able to maintain the discrete approximation scheme at a bounded error of the continuous exact solution. Thanks to this fundamental result, we can consider the synchronous/asynchronous dynamical neural fields simulation dilemma, (i.e., the fact that authors often wonder whether they can efficiently simulate such a dynamical system using an asynchronous scheme) as solved in such a context.

5. A generic asynchronous implementation

Let us finally consider a last aspect of asynchronous computation, i.e. not the fact that we want to simulate a continuous system in an asynchronous way, but the fact that we want to simulate a system *intrinsically asynchronous* in a sequential or distributed computational device. This covers two fundamental aspects.

On one hand, each unit has a local clock, so the state evolution depends entirely on the occurrence of asynchronous mechanisms over time. This means there are delays between units, with unpredictable exact values. On the other hand, there is another semantics related to anachronism, i.e. the fact that information associated to input and output is defined by both a value and the time at which the value is issued. In other words, the information is defined through temporal events.

Concerning biologically plausible models, the event-driven computation scheme has been mainly developed for spiking neuron models. Such models are not addressed here [5]. Nevertheless, this scheme could be certainly extended at a more mesoscopic level, as that of cortical columns, modeled by dynamical neural fields, as developed here. The dedicated simulation tool is an event-based neuron simulation kernel as proposed by, e.g., [22] based on the well-known Discrete Event System Specification (DEVS) framework (for a comparative review see [7]), very easy to simulate on a single processor.

Very concretely, each unit is defined by three functions: (i) The next event function provides an estimation of a lower-bound of the future time, where the next event time is going to be fired. (ii) Two update functions define how the unit update its state when either its own internal event, or another incoming external event is issued. The DEVS allows to show that given this specification, a complete asynchronous system can be simulated, using a simple calendar queue of events, on any device (see, e.g., [9] for details). This allows not only to simulate asynchronous calculations on a centralized system, but also to *mix* both kinds of implementation.

In brief, we propose to not only use event-based unit simulation at the microscopic level, targeting spiking-neurons, but also at the mesoscopic level, modeling the emergence of temporal events (synchronization or more complex mesoscopic spiking patterns, dynamic change triggering, etc..).

An illustrative example

Let us instantiate this general discussion, through an illustrative example. In a recent work [23] a model has been designed that performs global competition, only using local connections, with diffusion of the inhibition throughout the network. This is far quicker to have a few local interactions when computing activity within the network and this makes the model a real candidate for distributed computations. We have re-implemented this mechanism considering asynchronous sampling via a minimal event-based simulation kernel[6], which obviously works since the system is still contracting when using asynchronous sampling, as discussed previously. This has been numerically experimented, as shown in Fig. 2, with the obvious heuristic to have the local sampling period roughly proportional to the state value variation (parsimonious principle), with a strong robustness with respect to the related parameters (modifying the asynchronous paradigms changes the transitional values, slightly influences the convergence speed, but does not modify the final result).

This numerical result corresponds to trivial implementation of (4), so that the convergence is really obtained thanks to asynchronous paradigm. We have numerically verified on this simulated example that we could approximate the true solution at any precision (which was obviously expected anyway). Though this is not informative to report in details, we have been able to reproduce all qualitative DNF input/output functions (*filtering* of the output bump shape, *selection* of the output bump among several input bumps, *tracking* of a moving input bump at the output level, *remanence* of the output bump after the partial or total suppression of the input bump, see [1] for details). We have been able to verify that providing that (5) is verified, the computation was guaranteed in all observed cases while if far away from this bound, it is expected to fail. It is however still an open point to verify whether this bound is numerically a thick one, in all interesting case. Though this is only a preliminary result that opens large perspectives on new asynchronous paradigms for discrete neural field implementations.

6. Discussion and conclusion

This work aimed at addressing the following key-points:

- Making the difference between the discretized model *sampling* time and the implementation *simulation* time (several implementation steps may be use to iteratively estimate one sample of time, whereas a closed-form expression may provide the result after several sampling times in one simulation time).

[5] see [17] for an introduction and [11, 8] for a recent theoretical analysis and general discussion in link with these aspects

[6] Code available at `http://enas.gforge.inria.fr`, while `EnaS` is the general purpose large-scale event-based multi-scale simulator at the edge of the state of the art.

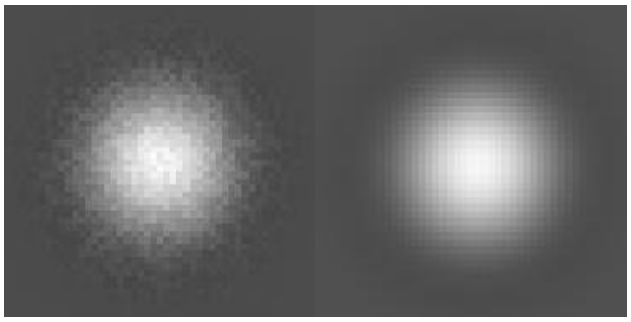

Figure 2: An example of asynchronous sampling of such maps (event-based implementation), applying convergence criteria derived here. We have numerically verified the conjecture that the present results apply when using asynchronous sampling. *Left view*: intermediate result, the fact asynchronous sampling yields randomization is visible. *Right view*: final result, after convergence.

- Calculating the bias between a continuous stable trajectory and its discretized approximation, all along the trajectory and given a sampling time (not only be sure that either the asymptotic targets are the same or that everything is perfect if the sampling is infinitesimally small).
- Making explicit the goals of using synchronous/asynchronous mechanisms at both the modeling level (asynchronous evaluation mechanisms avoid generating spurious synchronizations not present at the modeling level) and the implementation level (simulation on coarse or fine grain parallel processing clusters to multiply the calculation capability).
- Specifying whether not only the calculation but also the time is distributed (global time versus local time), i.e. whether a discrete clock dynamic system versus a discrete event dynamic system is considered.
- Deriving, in the case of the general family of dynamic neural field distributed and interconnected units, quantitative bounds that guaranty the convergence of the implementation calculations towards the modeling expected solution.

Addressing all these issues in such a short review would have been unrealistic, whereas a major but rather unnoticed work [19] on asynchronous computing, addresses these issues at a very general and deep level. The goal of this paper was thus to apply these results to the case of DNF computations and provide the complements in order to make these results directly usable.

Making the explicit distinction between sampling times and simulation times allowed us to review how well-established asynchronous evaluation methods can be efficiently used for dynamic neural fields simulation; as soon as reasonable assumptions are verified, fast convergence and unbiasedness are guaranteed. In return, as we explained in the previous section, dynamic neural field theory provides a fruitful playground for the study of asynchronous evaluation schemes. For example, in [24], it has been shown (numerically) that such an asynchronous evaluation method leads to new stable solutions that are functionally different from the continuous case. When presented with two identical stimuli at different locations, the field is able to stabilize itself on either one of the two stimuli because of the perturbation that lead the system away from a very unstable equilibrium state (like would also do any kind of noise). However, this new state, that has been shown to be very stable, can be also easily proved not to be a solution of the continuous equation of the field. What is thus the relevancy of such a continuous description if we are to evaluate it using numerical asynchronous equations ? Ideally, we wish we could have an equivalent continuous asynchronous description but unfortunately, this is not yet the case in the field of mathematics. We should then take extra precaution when describing a system using continuous equations and wonder if we are really simulating what we advertised in the definition of the system. Particularly, at the mesoscopic modeling level, it may be worthwhile to use an event-based paradigm instead of a clock-based one, as it is a well-defined paradigm which takes into consideration that not only the processing but also the timing are fully distributed.

From a more cognitive point of view, this study reveals the implicit presence of a central clock in a number of models and thus the implicit presence of a grand supervisor (a.k.a. central executive, homunculus, etc.) orchestrating the overall activity of the model. While this may be acceptable in most models that do not care about this parasitic presence, it is hardly acceptable if a model pretends to vanquish the curse of the homunculus.

Acknowledgment.. Partially supported by the ANR MAPS & the ANR KEOpS projects.

References

[1] Alexandre, F., Fix, J., Rougier, N., Viéville, T., 2009. Algorithmic adjustment of neural field parameters. Research Report RR-6923, INRIA.

[2] Amari, S., 1977. Dynamic of pattern formation in lateral-inhibition type neural fields. Biological Cybernetics 27, 77–88.
[3] Bahi, J., Contassot-Vivier, S., 2002. A convergence result on fully-asynchronous discrete-time discrete-state dynamic networks. Research report, AND Team, LIFC, IUT de Belfort-Montbliard, Belfort, France.
[4] Barret, C., Reidys, C., 1999. Elements of a theory of computer simulation i: Sequential ca over random graphs. Applied Mathematics and Computation 98, 241.
[5] Bertsekas, D., Tsitsiklis, J., 1991. Some aspects of parallel and distributed iterative algorithms - a survey. Automatica 27, 3–21.
[6] Bertsekas, D., Tsitsiklis, J., 1997. Parallel and Distributed Computation: Numerical Methods. Athena Scientific.
[7] Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J., Diesmann, M., Morrison, A., Goodman, P. H., Jr., F. C. H., Zirpe, M., Natschläger, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A., Boustani, S. E., Destexhe, A., 2007. Simulation of networks of spiking neurons: a review of tools and strategies. Journal of Computational Neuroscience 23 (3), 349–398.
[8] Cessac, B., Paugam-Moisy, H., Viéville, T., 2010. Overview of facts and issues about neural coding by spikes. J. Physiol. Paris 104 (1-2), 5–18.
[9] Cessac, B., Rochel, O., Viéville, T., 2009. Introducing numerical bounds to improve event-based neural network simulation. Research Report RR-6924, INRIA.
URL http://hal.inria.fr/inria-00382534/en/
[10] Cessac, B., Samuelides, M., 2007. From neuron to neural networks dynamics. EPJ Special topics: Topics in Dynamical Neural Networks 142 (1), 7–88.
[11] Cessac, B., Viéville, T., jul 2008. On dynamics of integrate-and-fire neural networks with adaptive conductances. Frontiers in neuroscience 2 (2).
[12] Chazan, D., Miranker, W., Viéville, T., 1969. Chaotic relaxation. Linear Algebra and its Applications 2, 199–222.
[13] Coombes, S., 2006. Neural fields. Scholarpedia 1 (6), 1373.
[14] Fates, N., 2009. Asynchronism induces second order phase transitions in elementary cellular automata. Journal of Cellular Automata 4, 21–28.
[15] Fates, N., Morvan, M., 2005. An experimental study of robustness to asynchronism for elementary cellular automata. Complex Syst. 16, 1–27.
[16] Garcia, L., Jarrah, A., Laubenbacher, R., 2006. Sequential dynamical systems over words. Applied Mathematics and Computation 174, 500–510.
[17] Gerstner, W., Kistler, W., 2002. Spiking Neuron Models. Cambridge University Press.
[18] Grimbert, F., march 2008. Mesoscopic models of cortical structures. Ph.D. thesis, Nice.
[19] Mitra, D., 1987. Asynchronous relaxations for the numerical solution of differential equations by parallel processors. SIAM J. Sci. Stat. Comput. 8 (1), 43–58.
[20] Press, W., Flannery, B., Teukolsky, S., Vetterling, W., 1988. Numerical Recipes in C. Cambridge University Press.
[21] Robert, F., 1994. Les systèmes dynamiques discrets. Vol. 19. Springer.
[22] Rochel, O., Martinez, D., 2003. An event-driven framework for the simulation of networks of spiking neurons. In: Proc. 11th European Symposium on Artificial Neural Networks. pp. 295–300.
[23] Rougier, N., 2006. Dynamic neural field with local inhibition. Biological Cybernetics 94 (3), 169–179.
[24] Rougier, N., Vitay, J., 2006. Emergence of attention within a neural population. Neural Networks 19 (5), 573–581.
[25] Wilson, H., Cowan, J., 1973. A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. Kybernetic 13, 55–80.