September 15, 2001

Event-driven Simulation of Large Networks

of Spiking Neurons

*Delorme A, Thorpe S

CERCO, Faculté de medecine, 133 route de Narbonne, 31062 Toulouse Cedex, France

arno@salk.edu, thorpe@cerco.ups-tlse.fr

* Corresponding author: arno@salk.edu

Computational Neurobiology Laboratory, Salk Institute for Biological Studies, 10010 Torrey Pines

Road, CA 92107, La Jolla, USA

(858) 453-4100 x1124 office

(858) 587-0417 fax

http://www.cnl.salk.edu/~arno

Short title

Modeling Large Networks of Spiking Neurons

Classification number

7.05.Mh

Abstract   (170 words; expected 200)

Many biological neural network models face the problem of scalability because of the limited computational power of today's computers. Thus, it is difficult to assess the efficiency of these models to solve complex problems such as image processing. Here, a new technique to tackle this problem is presented: event-driven computation. Only the neurons that emit a discharge are processed and, as long as the average spike discharge rate is low, millions of neurons and billions of connections can be modeled. We describe the underlying computation and implementation of such a mechanism in SpikeNET, our neural network simulation package. The type of model one can build is not only biologically compliant, it is also computationally efficient as 400,000 synapses can be updated per second on a standard desktop computer. In addition, for large networks, we can set very small time steps (less than 0.01 ms) without significantly increasing the computation time. As an example, this method is applied to solve complex cognitive tasks such as face recognition in natural images.

1. Introduction   (4742 words)

There are currently a large number of different software packages that can be used for simulating neural networks. Many have been designed for simulating networks of artificial neurons and make no attempt to model the detailed biophysics of neurons. The underlying units have no structure, and their outputs typically consist of a single continuous value (often in the range 0 to 1 or from -1 to +1). While such systems have been widely used, and have applications in a wide range of engineering and financial areas, few would regard them as being useful as tools for the computational neuroscientist.

At the other end of the spectrum, sophisticated programs exist such as GENESIS and NEURON, which are good for performing detailed biophysical simulations that take into account factors like the dendritic structure and complex channel kinetics (Hines and Carnevale 1997, Bower and Beeman 1998), but the level of detail makes it difficult to simulate very large networks.

In this article, we describe SpikeNET, an object oriented neural network simulation package written in C++ code that lies between these two extremes. It is sufficiently biologically realistic to examine the role of temporal properties such as synchronous or asynchronous spiking in neurons, and yet sufficiently simple to allow real-time simulation of large-scale networks of neurons.

We first detail the implementation of spike propagation in SpikeNET. Then, we will analyze the implications for biological modeling and we will finally focus on a large-scale model of retinotopically organized groups of neurons.

2. Methods: SpikeNET

SpikeNET was initially developed at our laboratory in 1994 to process natural images using large networks of integrate-and-fire neurons. We set up the basic rules for the propagation of spikes. Spikes propagate in an event-driven way: at each time step, only activated neurons are processed. Neurons are organized in retinotopical homogenous maps, which allow the definition of a common basis for the

neurons receptive field (RF) that can be shifted depending on the neuron location. These two features form the basis of the computational power of SpikeNET compared to other neural simulator packages.

We have implemented various applications using SpikeNET ranging from face detection and recognition in natural images (Delorme *et al* 1999, Van Rullen *et al* 1998) to unsupervised learning in primary visual cortex (Delorme *et al* 2001) and motion processing in extrastriate visual areas (Paquier, Delorme and Thorpe, unpublished manuscript). SpikeNET has been mainly used to model feed-forward propagation in the visual system but, it is also well suited to simulate any network that can be organized into topological maps. Though not designed for this purpose, SpikeNET can also be used to model randomly connected networks of neurons.

The basic objects in SpikeNET are two-dimensional arrays – that we will call neuronal maps - of relatively simple leaky integrate-and-fire neurons. Each unit is characterized by a small number of parameters: a membrane potential, a threshold, and (in some cases) a membrane time constant. When an afferent neuron fires, the weight of the synapse between the two neurons is added to the target neuron's potential, and we test to see whether the neuron's potential has exceeded the threshold. If so, the neuron's potential is reset (by subtracting the threshold) and the neuron is added to the list of neurons that have fired during the current time step. Propagation of activity within SpikeNET involves sending lists of spikes between neuronal arrays as illustrated in figure 1. The event-driven nature of spike propagation is one of the reasons for the efficiency of SpikeNET as a modeling system. One of the advantages of this approach is that increasing the number of time steps has relatively little effect on processing time.

Insert figure 1 around here

2.1. Neuronal units

Leaky integrate-and-fire (IF) neurons (Lapicque 1907) stand in between abstract and biological models. For biological modeling, until recently, only conductance based Hodgkin-Huxley models were used (Hodgkin and Huxley 1952). The recent gain in interest for IF neurons is due to their simplicity facilitates mathematical analysis of populations of spiking neurons (Gerstner 2000). They are also computationally more efficient than Hodgkin-Huxley neurons (Hansel *et al* 1998). Moreover, as a biological modeling tool, IF neurons with minor modifications can model very accurately single compartmental Hodgkin-Huxley neurons (Destexhe 1997). Recently, Jaffe and Carnevale (1999) also provided evidence that the dendritic architecture of neurons tends to normalize synaptic integration, so that the single compartment simplification of neurons is not unreasonable.

We will now give a more rigorous definition of IF neurons and describe their implementation in SpikeNET. The membrane potential evolves below the firing threshold q according to:

$$C\frac{dV}{dt} = -g_l(V - V_l) + I_{syn}(t) \tag{1}$$

where the g_l and V_l parameters are the conductance and the reversal potential of the voltage dependant leak current. I_{syn} is the synaptic current due to the action of other neurons of the network and C the capacitance of the membrane. Whenever the membrane potential V reaches the threshold q a spike is fired and V is instantaneously reset to the resting membrane potential.

Under quiet conditions, leak currents make the membrane potential of real neurons converge toward an equilibrium state. Updating the membrane potentials of all neurons at each time step would result in a dramatic increase in computation time – especially when treating million of neurons - and we would lose the efficiency gained by the event-driven propagation of spikes. An alternate solution is to update membrane potentials each time the neurons are stimulated by a presynaptic spike. In the absence of stimulation (and assuming no sub-threshold dynamics), membrane potential decays do not

need to be calculated because they cannot lead to a discharge. Moreover, knowing the membrane time

constant $t = C / g_l$, the time lag between two incoming spikes allow us to calculate the decrease of the

membrane potential V between current time t and time of the last update t_{last} :

$$V(t) = V(t_{last})e^{\frac{-(t-t_{last})}{t}}$$ (2)

For a given neuron, synaptic current can be seen as a weighted sum in equation (3) or can

incorporate changes in membrane conductance for higher biological plausibility in equation (4):

$$I_{syn}(t) = \sum_{i \in A} W_i f(t - t_{spike}(i))$$ (3)

or

$$I_{syn}(t) = -\sum_{i \in A} \overline{g}_i (V - V_i) f(t - t_{spike}(i))$$ (4)

where W_i and $t_{spike}(i)$ are respectively the connection strength with the input neuron and its date of

discharge i (A being the set of afferent neurons). V_i and $\overline{g}_i$ are the reversal potential and

conductance of the synapse of input neuron i (this formula covers both excitatory and inhibitory

synapses). The implementation of f in SpikeNET in its simplest form is a step function being always 0

except at 0 (i.e. the discharge occurrence) where it has value 1 during one time step (if there is no

current leak in the output neuron, the result in equation (3) is independent of the time step value). For

more biological plausibility, f can implement a simple (or double) exponential decay function or more

complex non-linear behavior.

Most neurons are only affected by incoming spikes from their afferents. However, for certain

"input" cells, for example to retinal ganglion cell, we determined spike timing by a direct calculation that

depends on the stimulus. Electrophysiological studies have shown that X-cell behavior (i.e. parvocellular input to the visual system) could be approximated using a simple delay integrate-and-fire neuron where the input current is proportional to the activation value at this location (Reich *et al* 1998). In the context of retinal ganglion cells, we can perform a local contrast extraction ("Mexican-hat" convolution) on the image, and this value can be used to calculate the latency of the unit's spike - the earliest latencies correspond to those cells for which the value of the convolution is highest, whereas longer latencies correspond to lower activation levels (figure 2). To discretize this process in time, one has then to sort these activation values to determine the order into which the neurons will discharge. This sorting procedure represents a lot of computer time for large input arrays so we used a pseudo-sorting algorithm by regular sampling of the activation values and using a sampling rate high enough to obtain the exact order.

Insert figure 2 around here

2.2. Neuronal projection in SpikeNET

The event-driven propagation of spikes imposes constraints on modeling: we first need a mechanism to transform afferent synaptic weights into efferent ones. Instead of neurons integrating their receptive fields as in standard simulator packages, event-driven propagation of activity means that it is the spiking neurons that update their target neurons. For homogeneous projections between neuronal array maps of equal sizes this is achieved by mirroring the convolution (figure 3). The terms homogeneous projections mean that, inside a neuronal array map, neurons share the same synaptic weights while still processing separate zones of their input space. For example in the primary visual cortex of mammalian, it is generally assumed that all the neurons that detect the same orientation at different locations of the visual field have similar properties. Thus, in a computer implementation, it is possible to use the same synaptic pattern of connectivity for all these neurons. On the other hand, if each neuron in a neuronal

map has its own receptive field and synaptic weights – in the non-homogeneous receptive field case - we must reconstruct lists of output synapses for each neuron individually. We will mainly focus on homogeneous RF, which are easier to manipulate and allow us to model far more connections than non-homogeneous RF as we will see later on. In real neural systems, topographical maps might be of different magnification and size. When considering input and output neuronal array maps of different sizes, the mechanism that convert RFs into efferent synaptic weights becomes considerably more complicated. We designed an algorithm for performing such projections but we won't enter the technical details of its implementation.

Insert figure 3 around here

We also want to emphasize that, for computational purposes, all zero weights were removed. The computation cannot be seen as matrix multiplications but rather as an application of several one-dimensional arrays, each one representing consecutive values in the pattern of synaptic weights along the x-axis. This structure was used (*i*) to speed up computation for non-square receptive fields and (*ii*) for future development using highly efficient vectored-array computation such as OPENGL, and multimedia instruction sets such as MMX, SSE or ALTIVEC.

2.3. Propagation of spikes and Parallel applications

We will now describe the dynamics of spike propagation in the network, which is the core of the algorithm. We also show how the algorithm can be extended for parallel applications using asynchronous communications between several processors, thus preventing most synchronization problems.

The computation is very simple and only involves transfers of lists of spikes. As described previously, neurons are organized in topologically organized two-dimensional arrays – neuronal maps.

These maps are first pooled in a loop at each time step. Then, every map is processed sequentially checking whether spikes were triggered in its afferent maps at the previous time step. If so, input spikes are propagated in the output maps and output neuron membrane potentials that have been modified are checked to see whether the threshold has been exceeded. Whenever a map finishes its processing, it is removed from the loop. The time step propagation finishes when the loop collapses. Figure 4 illustrates the propagation in a simple model when one of the neuronal maps is processed on a remote computer.

The propagation is map centered as the processing maps just pick up the information they need – spikes locations – in the other maps. Input maps and network maps are purely passive: they simply store spikes lists to be provided to other processing maps. If the information is not available (i.e. for a network map that has not receive yet its list of spikes from the network) then the processing for this map is skipped but the map remained in the loop to be processed next turn. Accesses from the network are asynchronous to this loop, which means that it could occur at any time and would just suspend the current computation without affecting it.

Though it might seem complex at a first glance, this algorithm is especially simple: neuronal maps at each time steps only pick up lists of spikes from their afferent and propagate them. We only organized neuronal maps into a loop that collapses at each time step in the case of a parallel implementation to allow that processor jump over some node that are not ready yet.

Insert figure 4 around here

2.4. Learning algorithms

Implementing learning algorithms imposes constraints in SpikeNET: some learning mechanisms might update synaptic weights of a neuron depending on the history of input synaptic spikes to this neuron. For instance, recent works in neurophysiology showed that synapse plasticity depends on the relative

timing between the presynaptic and the postsynaptic spikes (Markram *et al* 1997). Thus we need to keep the reference of every synapse that has been activated: a convenient way to do that is to keep the reference of the pre-synaptic spikes. By keeping previous presynaptic times of discharge, SpikeNET can easily reconstruct the history of each synapse and update synaptic parameters depending on this history. We have implemented this mechanism in a simulation for the emergence of orientation selectivity in the mammalian primary visual cortex (Delorme *et al* 2001).

3. Results: performance of SpikeNET

SpikeNET has been designed to be computationally efficient. One of its advantages comes from the efficient use of RAM. Since the number of parameters per neuron is kept low, each neuron can require as little as 16 bytes of memory, depending on the type of numerical precision required. More importantly, the use of shared synaptic weights in homogeneous projections means that one set of weights can be used for all the neurons in an array. The use of shared weights for neuronal maps is not biologically unrealistic: for instance in the visual system, many neurons have similar selectivity at different positions in the visual field. As a result it is perfectly reasonable to simulate networks with tens of millions of neurons and billions of synapses on standard desktop computers.

Insert figure 5 around here

3.1. Real time computation speed

The main advantage of SpikeNET is computation speed and it's potential application to real time modeling. For simple integrate-and-fire neurons, SpikeNET can update roughly 20 million connections per second (figure 5), even when using a sensitivity parameter to modulate the effect of each synaptic input (using a standard 266 MHz Macintosh PowerPC 750 processor). This is sufficient to model a network of 400,000 neurons in real time, using a time step of 1 ms (assuming 49 connections per

neuron, and an average firing rate of 1 spike per second, a value which is a reasonable estimate for the average firing rate of cortical neurons). Note that with a more conventional neural network simulation approach one has to recalculate every unit at every time step, and so the same computational power would only allow 20,000 connections to be calculated per millisecond, which with 49 connections per neuron would limit real-time simulation to around 400 neurons.

Typically, the complexity of the spike propagation algorithm is $q(N)$, N being the number of synapses updated per second. What is critical is that this complexity does not depend on the time step value as long as there are enough synapses updated per time step ($N_{dt}>>1$). Increasing the time resolution from 1 ms to 0.1 ms has virtually no effect on computation time, since the number of spikes that are propagated does not change. For a large network of 1 million neurons discharging at 1hz and connected to 10,000 neurons each, 10^{10} connections are processed per second and one can thus get time steps as low as 10^{8} s (assuming $N_{dt}=100$) without a significant impairment of computation time.

Performance is clearly optimal with shared weights, but even when each neuron has its own set of weights (which obviously increases RAM usage considerably), speed only drops by a factor of around 2. Adding decays using equation (2) to simulate the leaky nature of the synaptic integration process adds roughly 30-40% to the computation time.

Rather than entering abstract details about the variation of the performances of SpikeNET as a function of parameter modifications, we present the result of an application in figure 6 (Delorme and Thorpe 2001). The simulation represents the propagation of a feed-forward wave of spikes in a visual system like architecture. The model contains about 32 million neurons and 245 billion connections and still, it runs with a reasonable time on a standard desktop computer (about 30 minutes on a 266 MHz Macintosh PowerPC 750 processor). In comparison, other biological modeling softwares can hardly simulate small groups of about 10,000 neurons on vectored-array Cray processors.

Insert figure 6 around here

3.2. Parallel application

Although running SpikeNET on a standard desktop machine is already reasonably quick, as described previously, the very nature of SpikeNET makes it an ideal candidate for implementation on parallel hardware. The factor that usually prevents large-scale use of parallel hardware is the amount of communications needed between processors. For many problems, one sees little speed up once the computation has been split between more than 4 or 8 processors. However, with SpikeNET, the only information that needs to be transferred between processors is the lists of spikes. The format used by SpikeNET means that the identity of each neuron which fired can be transmitted using only around 1-2 bytes, and so even a network with 10 million neurons firing at an average of one spike per second could be simulated in real time without saturating the bandwidth of a cluster of processors linked by conventional fast-Ethernet technology at 100 Mbytes per second. We implemented SpikeNET onto a cluster of 2 Linux machines. We used the implementation for parallelization that was described in the previous paragraphs, using asynchronous UNIX interruption and TCP/IP to communicate between the processor. In preliminary simulations, we observed typically a 90% efficacy of parallelization using Fast-Ethernet (100% would be perfect parallelization). However higher levels of parallelization need to be assessed in more details and we are working on developing broadcast techniques such as UDP to minimize the network load.

3.3. Accuracy of the algorithm

Concerning the accuracy of spike integration and membrane potential variation, SpikeNET's error is of the order of the time step value (dt) but it could be made lower with a few modifications. When a neuron fire between time t and $t+dt$, a local error on firing time of order dt is generated because firing occurs systematically at time $t+dt$, assuming no delay in the connection. First it leads to an error on membrane potential at time $t+dt$ and another one on the excitatory post-synaptic potential delivery date. Hansel *et al* (1998) proposed to interpolate membrane potential variations between time *t* and *dt*

in order to calculate a better approximation for the date of discharge of neurons. The global error is then of order dt^2. In SpikeNET, the reset of membrane potential of neurons can easily be modified to fit this better approximation.

4. Discussion

We showed how SpikeNET was able to simulate efficiently networks with large number of neurons. Though the simplicity and homogeneity of the algorithm seems appealing, one can question the biological plausibility of using IF neurons to model real neuron behavior. We will try to justify some of the simplification we made here and also look at possible extensions of the integrate-and-fire approach.

4.1. The IF model is a good approximation for network behavior

Leaky IF neurons cover all of the currently known neural network behaviors. First, oscillations such as alpha rhythms, similar to those recorded in the brain have been observed in populations of IF neurons (Liley *et al* 1999). Second, Golomb *et al* (1999) were even able to find a connectivity schemes into which IF neurons behave as bistable units, which may be the basis of short-term memory. Finally, a model of hippocampus using IF neurons can behave as an associative memory (Samsonovich and McNaughton 1997) and converge to stable attractors (Hopfield and Herz 1995).

While IF neural networks can model macroscopic behavior of real neural systems, they have also been used to simulate the behavior of individual neurons. For instance, leaky IF neurons have been used to model accurately real neuron spike discharges in the lateral geniculate nuclei (Reich *et al* 1997, Reich *et al* 1998) and the authors argue that any further simplification of this model was unable to capture the underlying dynamic of spike discharge.

At a low frequency of discharge, IF neurons can be made indistinguishable from Hodgkin-Huxley neurons by including the rough dynamics of voltage dependant channels. Destexhe (1997) has shown that adding the bimodal intrinsic properties of voltage dependant channels, which become active only if

the membrane potential cross a threshold (and modeling continuous changes of these parameters in real Hodgkin-Huxley models during the action potential), can mimic in great accuracy real Hodgkin-Huxley dynamics. In comparison, simple IF models are more imprecise because they lack the duration of the action potential. Adding these properties in SpikeNET would be straightforward as the transient activation of channels occurs at the same time as the discharge onset. Following a discharge, the reset of the neuron potential would be calculated depending on the time lag separating the current update with the next one (in a process similar to equation (2)).

The basic cellular model can also be made more interesting by including a sensitivity parameter that modulates the effect of incoming action potentials. We have used this feature to implement the rank-order coding scheme originally proposed by Thorpe (1990). According to this scheme, before each image propagation, the sensitivity parameter is initially fixed at 1.0 and decreases by a fixed percentage with each incoming impulse, resulting in a progressive desensitization of the post-synaptic neuron which can be thought of in terms of fast shunting inhibition in the visual system (Delorme et Thorpe, in preparation). This mechanism was implement in an extended version of equation (3) (for more details see Delorme and Thorpe 2001, Van Rullen *et al* 1998). The net result of this mechanism is that activation is maximal only when the spikes arrive in the order of the weights - with the highest weight synapses being activated first.

Many other parameters might be calculated in very short amount of time, even calcium concentration for instance. Compartmental neurons can also be modeled and be updated each time a neuron receives a spike. As long as the solution is deterministic, it can be calculated using the difference in update time between two incoming spikes (in a process similar to equation (2)). If the global analytical solution for a set of differential equations governing the neuron behavior cannot be calculated from the time step difference, the solution must be calculated using derivative methods. Thus between each stimulation of the neuron, we would have to compute the underlying dynamics using standard techniques such as a first order (Euler) or a slightly modified second order (Runge-Kutta) integration algorithm. A limitation of SpikeNET is that all the process in the neurons must be

either discharge locked or delayed discharge locked, which mean that spike can not be triggered by sub-threshold dynamic.

4.2. Towards implementation of delays and synaptic dynamics

In real neural systems and models, a spike can stimulate efferent neurons with different delays: some synapses being activated just after the spike discharge and some others being activated later. A simple way of implementing such delays is to postpone the propagation of lists of spike. In SpikeNET, delays must be discretized in term of time steps but this simplification is not restrictive: as we will see, even a large increase in the number of time steps does not impair the computation speed. Moreover, standard time steps values used in SpikeNET range from 1 ms to 0.01 ms which is on the order of accuracy needed to model these mechanisms. As illustrated in figure 7 delayed propagation would be associated to stored lists of spikes from previous time steps.

Synaptic models can also have complex internal dynamics (Tsodyks *et al* 1998) that go beyond the simple full open or closed synapses. For instance, we might want to implement alpha synapses, whose conductance decreases exponentially after transient rises. Dynamic synapses might also switch between different states. To implement these mechanisms, we must discretize them in time, the maximum sampling rate being the number of time steps per second. As shown in figure 7, this is equivalent to propagating many excitatory post-synaptic potentials at different delays, i.e. propagating the same spikes at different times and with different synaptic weights. As for delayed synaptic propagation, standard time steps are of sufficient accuracy for these mechanisms. Delayed propagation might not take additional time to compute but the simulation of complex synaptic dynamics strongly impair computation speed, the speed being virtually be proportional to the number of events in synapses.

Insert figure 7 around here

4.3. Comparison with other event-driven software packages

Comparing SpikeNET to other neural simulation packages is quite hard. Such packages typically either model detailed neuronal mechanism using standard techniques (for a review that is a bit outdated but which still reflects the state of biological modeling software see De Schutter 1992), or they deal with abstract entities that can hardly be linked to real neuron firing rates. SpikeNET is one of only a few simulators that lie between these two extremes.

The map-centered processing technique we used prevents most of problems inherent to synchronization of parallel program flows. The alternate solution implemented in the software SpikeLAB (Graßmann and Anlauf 1999) would have been to propagate list of spikes in an event-driven way: as soon as lists of spikes would be generated they would be propagated through the network. List of spikes would be organized in a stack filled initially with input lists of spikes. Each time a map generates a list of spike, the spikes are piled up into the stack to be propagated and popped out when they have touched all their targets. However, this raises the problem of the order of treatment of maps: when a map is connected to many others, to which map should the spikes be propagated first? The other problem with this implementation would be that two maps with recurrent excitations might indefinitely excite each other thus blocking the computation and there must some controls to counteract these side effect behavior. For parallel application, this kind of algorithm proved to raise strong synchronization problems for small groups of even less than 100 neurons (Graßmann, personal communication). A refinement of this approach is presented in Mattia and Del Guidice (2000). However the serial algorithm presented in this paper seems rather complex and requires sorting of spikes at each time step. This pure event-driven spike propagation offers the advantage of simplicity but appears to be less well adapted for real applications.

The map-centered processing solution we developed holds other advantages for parallel applications. First, processing on cluster of processors preserves the structure of the single processor

propagation; computation in cluster of processors is implemented using a new kind of map – network maps that provide lists of spikes just as an input map would do - so the main loop remain blind to this minor change (figure 4). Because of the limited bandwidth between clusters of processors, spike propagation must be uniformly distributed in time. It is actually the case with the map-centered solution we implemented: because of the sequential processing of maps, lists of spikes are sent continuously to the network in an asynchronous way. Finally, this structure helps to keep spikes from previous time steps that we will use for learning and biological modeling purposes.

Because of their computational power and their biological relevancy, large network of integrate-and-fire neuron using event-driven computation are bound to be popular for future understanding of real neural networks behavior. We have showed how such an implementation could be done and have highlighted some of the benefits of using it. We believe our approach, only based on time labeled lists of spikes is much more simple and straightforward. The use of homogeneous connections and neuronal maps have also allowed us to scale up the network to million of neurons and billion of connections which is far beyond what is currently achievable by any other neural network package.

Although primarily designed as a tool for modeling biological neural networks, the level of performance obtained with SpikeNET is such that in a variety of tasks, processing architectures developed using SpikeNET can perform at least as well and in many cases substantially better than more conventional image processing techniques (Delorme and Thorpe 2001).

Acknowledgement

References

Bower J M and Beeman D (1998) *The book of GENESIS: Exploring realistic neural models with the GEneral SImulation System* (New York: Springer-Verlag)

De Schutter E 1992 A consumer guide to neuronal modeling software *Trends in Neurosci.* **15** 462-4

Delorme A, Perinnet L and Thorpe S 2001 Network of integrate-and-fire neurons using Rank Order Coding B: spike timing dependant plasticity and emergence of orientation selectivity *Neurocomputing* **38-40** 539-45

Delorme A and Thorpe S 2001 Face recognition using one spike per neuron: resistance to image degradation *Neural Netw.* in press

Delorme A, Van Rullen R and Thorpe S J (1999) Rapid object recognition based on asynchronous feed-forward processing *Perception* **28 (suppl.)** 128-9

Destexhe A 1997 Conductance-based integrate-and-fire models *Neural Comput.* **9** 503-14

Gerstner W 2000 Population dynamics of spiking neurons: fast transients, asynchronous states, and locking *Neural Comput.* **12** 43-89

Golomb D and Ermentrout G B 1999 Continuous and lurching traveling pulses in neuronal networks with delay and spatially decaying connectivity *Proc Natl Acad Sci U S A* **96** 13480-5

Grassmann C and Anlauf J K 1999 Fast digital simulation of spiking neural networks and neuromorphic integration with SPIKELAB *Int. J. Neural Syst.* **9** 473-8

Hansel D, Mato G, Meunier C and Neltner L 1998 On numerical simulations of integrate-and-fire neural networks *Neural Comput.* **10** 467-83

Hines M L and Carnevale N T 1997 The NEURON simulation environment *Neural Comput.* **9** 1179-209

Hodgkin A L and Huxley A F 1952 A quantitative description of membrane current and its application to conduction and excitation nerve. *J. Physiol. (Lond.)* **117** 500-44

Hopfield J J and Herz A V 1995 Rapid local synchronization of action potentials: toward computation with coupled integrate-and-fire neurons *Proc. Natl. Acad. Sci. U S A* **92** 6655-62

Jaffe D B and Carnevale N T 1999 Passive normalization of synaptic integration influenced by dendritic architecture *J. Neurophysiol.* **82** 3268-85

Lapicque L 1907 Recherches quantitatives sur l'éxitation électrique des nerfs traitée comme une polarisation *J. Physiol. Pathol. Gen.* **9** 620-35

Liley D T, Alexander D M, Wright J J and Aldous M D 1999 Alpha rhythm emerges from large-scale networks of realistically coupled multicompartmental model cortical neurons *Network: Comput. Neural Syst.* **10** 79-92

Markram H, Lubke J, Frotscher M and Sakmann B 1997 Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs *Science* **275** 213-5

Mattia M and Del Guidice P 2000 Efficient Event-Driven Simmulation of Large Networks of Spiking Neurons and Dynamical Synapses *Neural Comput.* **12** 2305-29

Reich D S, Victor J D and Knight B W 1998 The power ratio and the interval map: spiking models and extracellular recordings *J. Neurosci.* **18** 10090-104

Reich D S, Victor J D, Knight B W, Ozaki T and Kaplan E 1997 Response variability and timing precision of neuronal spike trains in vivo *J. Neurophysiol.* **77** 2836-41

Samsonovich A and McNaughton B L 1997 Path integration and cognitive mapping in a continuous attractor neural network model *J. Neurosci.* **17** 5900-20

Thorpe S J (1990) Spike arrival times : a highly efficient coding scheme for neural networks *Parallel processing in neural systems* ed R Eckmiller, G Hartman and G Hauske (Amsterdam: Elsevier)

Tsodyks M, Pawelzik K and Markram H 1998 Neural networks with dynamic synapses *Neural Comput.* **10** 821-35

Van Rullen R, Gautrais J, Delorme A and Thorpe S 1998 Face processing using one spike per neurone *Biosystems* **48** 229-39

Figure 1. Organization of SpikeNET. SpikeNET redirects lists of spikes between different groups of neurons organized in two-dimensional neuronal arrays. Since only a small percentage of cells fire in each time-step, communications between neuronal groups are kept to a minimum.

Figure 2. Basic behavior of an integrate-and-fire neuron. The latency of its discharge depends on the strength of the stimulation. With strong stimulation, the neuron will reach its threshold quickly whereas with weak stimulation the latency will increase.

Figure 3. (*a*) Illustration of a homogeneous receptive field for an output neuron. (*b*) Transformation of a RF into a projection field for an input neuron using event-driven propagation. The convolution of synaptic weight must be mirrored. (*c*) Illustration of a spike propagation, the gray levels of the output neurons representing different synaptic weights (taken as random in this case).

Figure 4. Illustration of network computation for one processor. The neural groups named "Retina", "Map1" and "Map2" are local to the processor whereas the map named "network Map" is remote. (*a*) Connection between neuronal maps. Arrows indicate direction of projections for the neurons in each maps. (*b*) Flow of information in a map centered view. At each processing time step, only "Map2" and "Map3" perform computations based on the information available in other maps. A remote computer that sends its results at each time step computes the "network Map". This information is received in an asynchronous way by the local processor, meaning that there is no waiting delay associated with the communication. (*c*) Detailed view of (*b*). At each time step *t*, all the maps are organized in a loop. Each time box for each map represents a list of spike (for simplicity we use *dt*=1). The computation loop shrinks as each map completes its computation and is removed from the loop (network maps don't really do any computation on the local computer but they were inserted in the loop for better readability). Finally the loop collapses

when all the maps have finished to process their input spikes. At this point the algorithm proceeds to the next time step, starting with a new computation loop.

Figure 5. Result of a simulation in real time with a variable number of integrate-and-fire neurons and a variable number of synapses ($dt = 1$ ms). Neurons implements equation (3) with no decay and the function f simulating fast shunting inhibition as explained in the text. For simplicity, the model is only two layers of neuronal maps of the same size, the first layer emitting spikes and the second one integrating these spikes (for the neuron count, only the second layer is taken into account). We fixed the number of synaptic connection and determined the number of neurons that one can simulate in real time at 1Hz. We considered two conditions, one in which each neuron had a separate set of synaptic weight and the other one into which every neuron had the same set of synaptic weights but shifted depending on the position of the neuron in the neuronal map (for individual weights, we were not able to go above a certain number of synapses because of the memory limitations of our computer). In both cases the relation is roughly linear, which implies that the computation time only depends on the number of synapses updated. For shared weights, the computation is more efficient and reaches a maximum at 49 synapses per neuron because of the optimal use of cache memory by the processor.

Figure 6. Propagation of a set of 400 face images (belonging either to the learning base or one of the two test bases). We only present the global result of the network superimposed with the input image. The size of the image was 910x700 pixels, which requires a network containing roughly 32 million neurons and 245 billion connections. For a correct detection, a neuron selective to a particular face must discharge within a 4x4 region located at the center of the face. Black squares indicate correct recognition and white ones false recognition. Despite the size of the network, the simulation could be completed in about 30 minutes of CPU time on a modest desktop computer

(Macintosh G3, 266 MHz). The network also shows high resistance to noise and image degradation. Adapted from (Delorme and Thorpe 2001).

Figure 7. (*a*) Propagation with delayed synapses. Each square represents a neuron and the upper neuron projects toward the lower map of neurons. On the left, when the upper neuron spikes, all the connection are processed at the current time step. On the right however, some synapses will be processed at future time steps, thus introducing a delay in the synapse propagation. (*b*) Complex synaptic dynamics. Propagating a synaptic weight at different delays (gray bars) can approximate synapses with complex conductances (dark curve); brightest bars will be processed later than darkest ones.
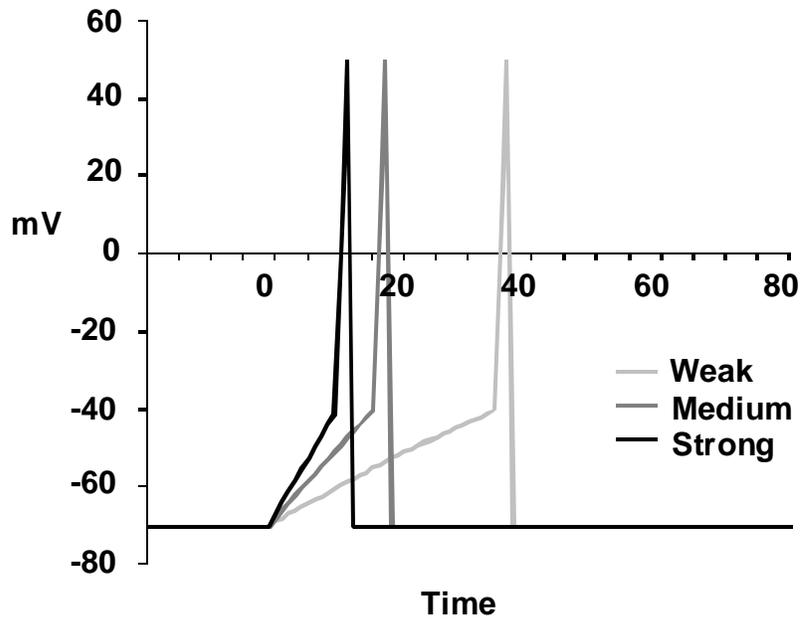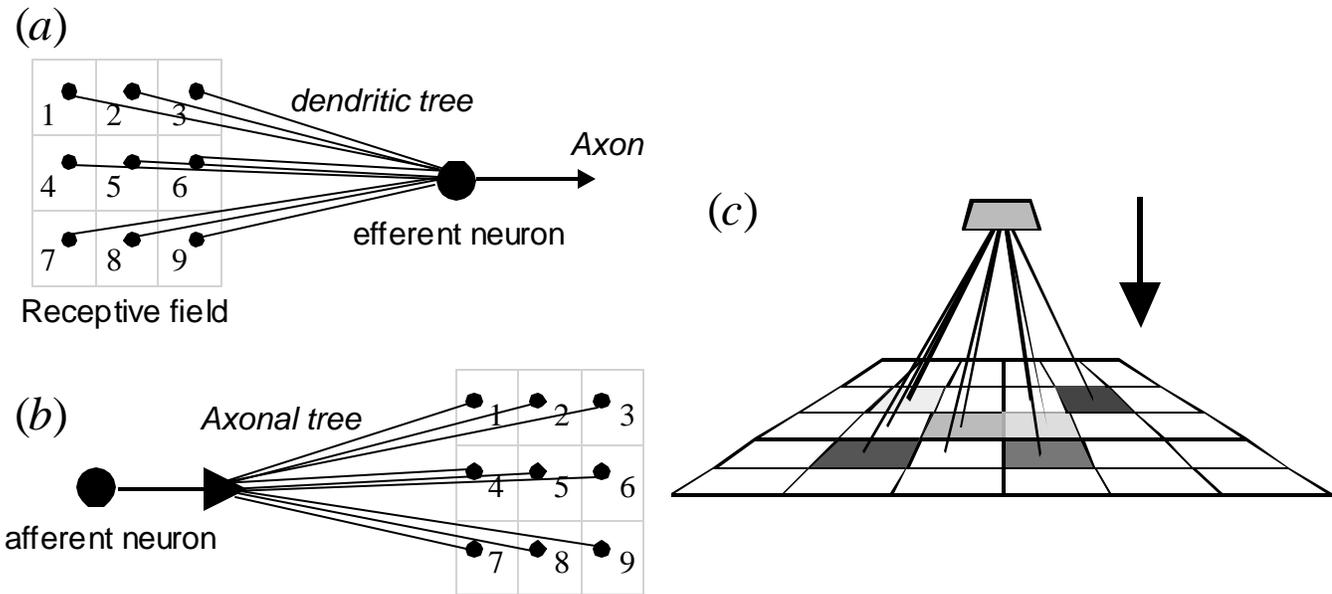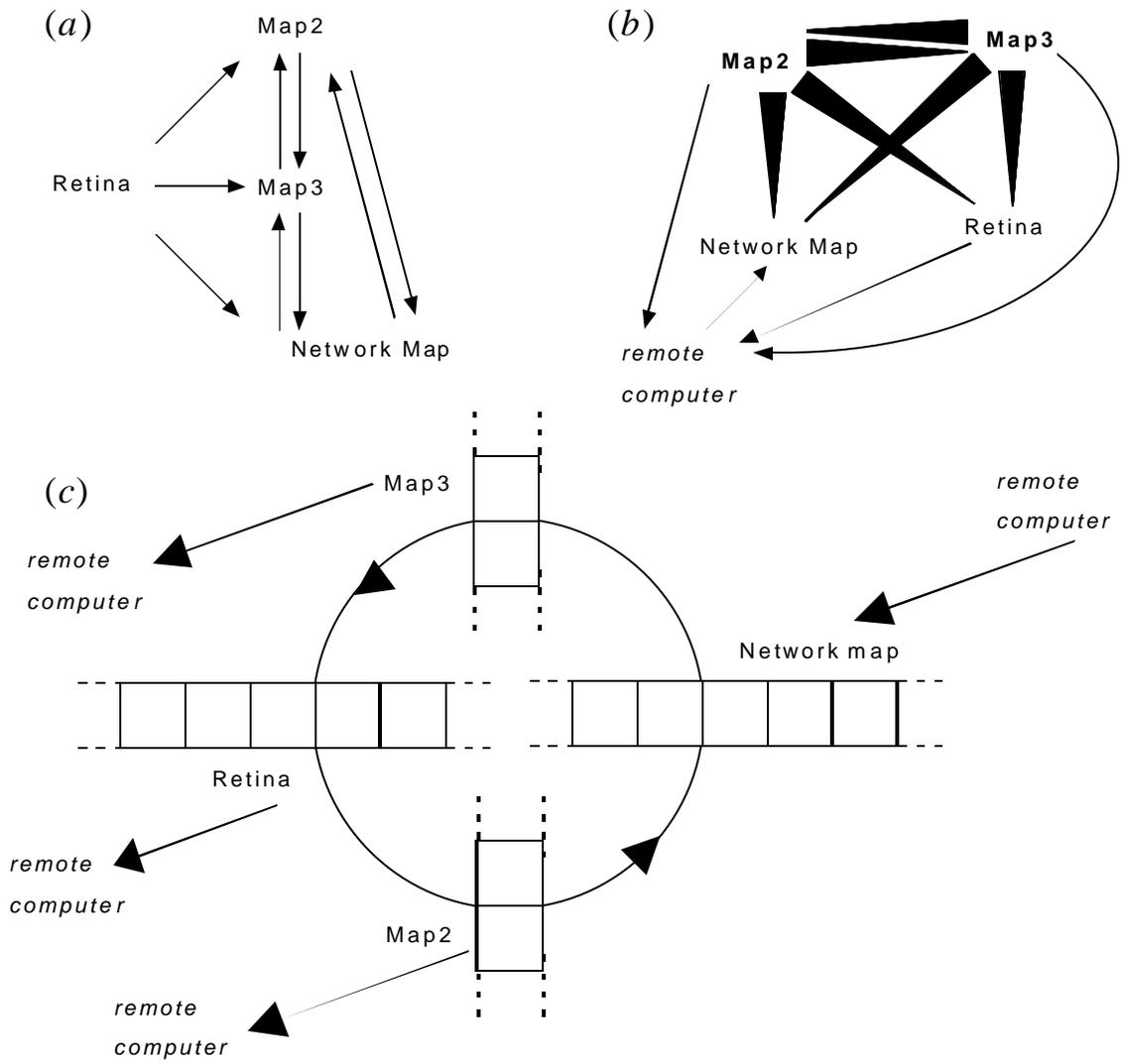
Figure 1.

Figure 2.

(*a*)

1 2 3
4 5 6
7 8 9

dendritic tree

Axon

efferent neuron

Receptive field

(*b*)

Axonal tree

afferent neuron

1 2 3
4 5 6
7 8 9

(*c*)

Figure 3.

(*a*)

Map2

Retina → Map3

Network Map

(*b*)

Map2   **Map3**

Network Map   Retina

remote computer

(*c*)

Map3

remote computer

remote computer

Retina

Network map

remote computer

Map2

remote computer
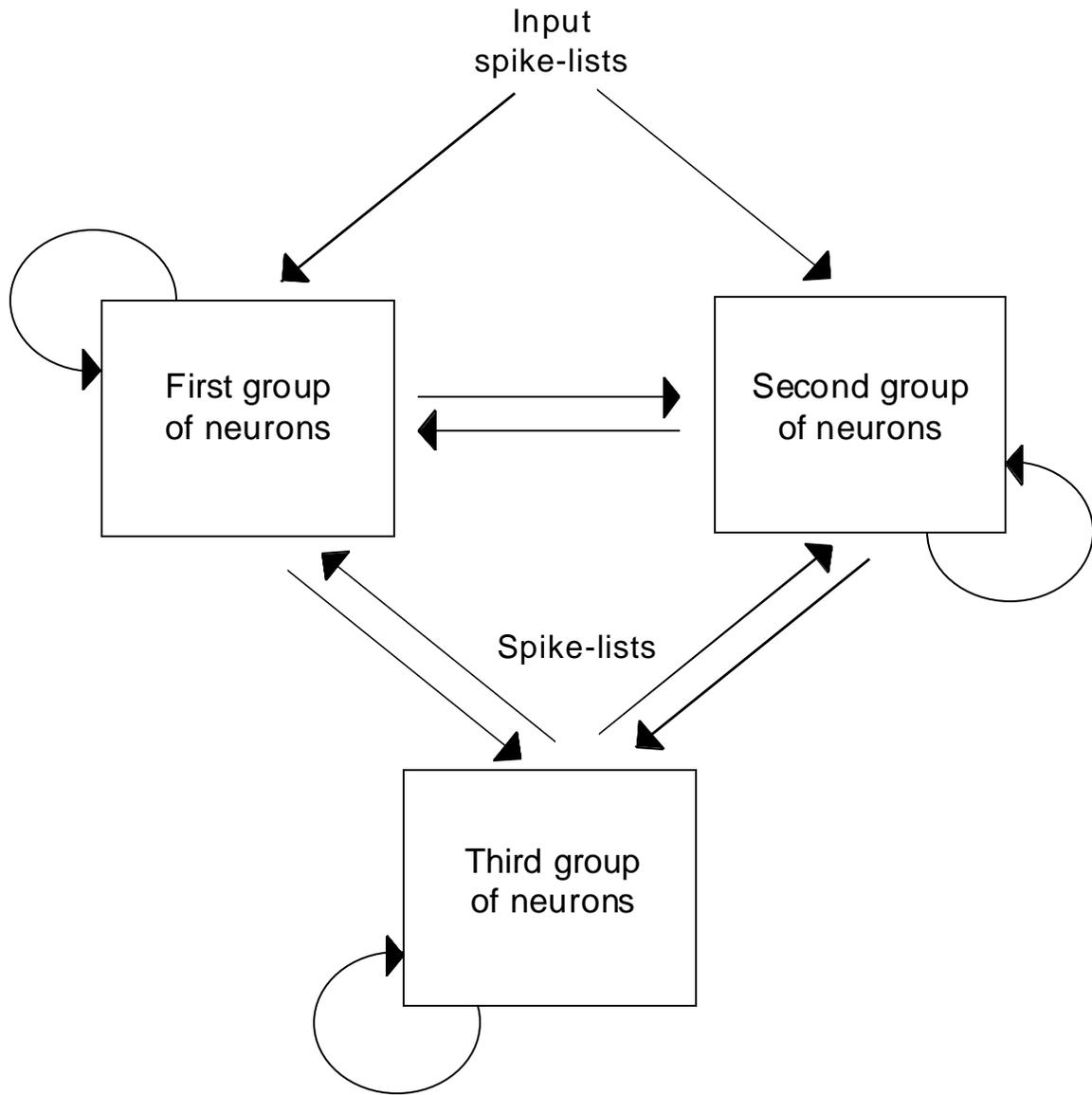
Figure 4.

Figure 5.

Figure 6.

(*a*)

(*b*)

Figure 7.