Study of a neuromimetical network for recognizing complex sounds

X. Giannakopoulos
Computer science student (Xavier.Giannakopoulos@studi.epfl.ch)

7th semester project at the CIRC-EPFL
supervisors Hervé Dedieu, Jan Erikson and Arnaud Robert

Searching a way to design a network suited for a particular problem, I added functionalities to an existing neural network framework to be able to design neuromimetical learning structures. The goal was to develop stable spatiotemporal patterns as a response to auditive stimuli, using a cochlear preprocessing. The constructed network exhibits stable response and learning behaviour, but the existence of stable spatiotemporal patterns linked to the input stimuli is not quantitatively analyzed and documented.

1 Table of contents

1 Table of contents 2

2 Biological sources 3
2.1 Auditive system 3
2.2 Spatio-temporal patterns 4
2.3 Synchronization 5
2.4 Synapse learning 6

3 Structure of the elements 7
3.1 Neuron Model 7
3.2 Layering system 8
3.2.1 General features of connections 8
3.2.2 Feed-forward connections 9
3.2.3 Self-connections 10
3.3 Learning 11
3.3.1 Local part 11
3.3.2 Global part 12
3.3.3 Triggering method 13

4 Programs 14
4.1 Structure declarations part 14
4.1.1 LecteurModel.................................. 14
4.1.2 Detecteur model 15
4.1.3 Layering system 15
4.1.4 Connections 15
4.2 Method declarations part 16
4.2.1 Learning 16
4.2.2 Weights histogram 17
4.3 Code structure 18
4.3.1 LecteurModel.................................. 18
4.3.2 Layering system 18
4.3.3 Learning system 19
4.3.4 Weights histogram 20

5 Results 21
5.1 Tuning the parameters 21
5.1.1 Fixed parameters 21
5.1.2 Tuned parameters 22
5.1.3 Parameter tables 22
5.2 Comments on the results 23
5.2.1 Sensitiveness and border effect 23
5.2.2 Learning effects 23

6 Conclusion 25
6.1 Ideas coming next 25

7 Bibliography 27

8 Annexes 29

This project can be seen from two points of view. On one hand it is a temptative try to see what can be done for auditive recognition by a new class of neuronal methods. On the other hand, I see it as a mean to know how useful the models of spiking neurons applied to real problems can be. First, I must say this is a six months project, and I had to take the straight line on many subjects that would normally require more care in a more furnished work. I apologize for this and hope that the work done will still be useful.

Part 2 is a review of the models from the biological field that shaped the network and its components. Part 3 covers choices made for the elements at different levels of the modeling. Part 4 describes the features as implemented in the extensions to the SLI language. Part 5 gives the parameters used for the simulations and comments the results.

2 Biological sources

In this part, I will be gross, since I will restrict my purpose to the facts, the models and the data that served in the following work, putting quickly aside the things examined but that I finally didn’t used.

2.1 Auditive system

When I consider the auditive pathway :

- mechanical part
- cochlea
- ciliate cells
- auditive nerve
- cochlear nuclei
- inferior colliculus
- thalamus
- cortex

there exist no model, not even the hope to arrive soon to a global model of the mechanical and neuronal structures involved.

Highly parallel processing has been shown to exist at a low level, as well as separate circuitry for verbal and non-verbal sounds. But even the system until the first relay into the cortex is to date beyond modeling, because of the many loops, sub-systems and multiple non-linear effects (among other problems).

A model for the mechanical part and the cochlea exists (AIM, cf. [5]). And a model for the ciliate cells and the auditive nerve is developed at the EPFL.

In both models, only a part of what is currently known is implemented, but the system provides realistic output.

I have to do recognition, and it is not plausible that advanced recognition (more than feature extraction) occurs at the level of the cochlear nuclei. On the other hand, it has no biological meaning to feed a network based on cortical structures with data from an auditive nerve simulator.

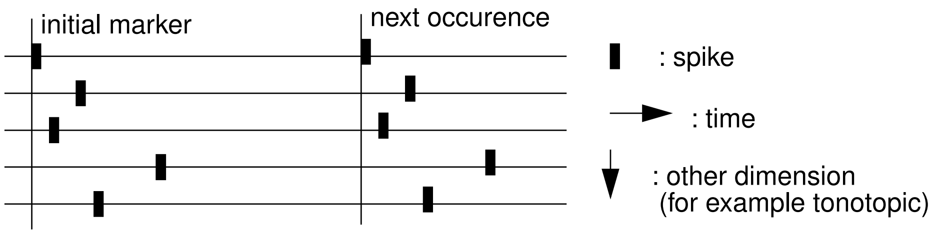
I must conclude that the gap between the biological direct inspiration and the problem is still too big, and that the network will be a theoretical network, not situated anywhere.

I will however continue to feed the network with output from the auditive nerve simulator, because I think that the preprocessing done by neuronal method will be more significant to a subsequent network than a mathematical method like formants analysis or Fourier transforms.

2.2 Spatio-temporal patterns

A main hypothesis of this work is that in the inputs from auditive stimuli exist spatio-temporal patterns approximatly at the level of cochlear nuclei that are few dependent from the speaker. The goal is to be able to recognize theses spatio-temporal patterns and to associate them with logatoms.

FIGURE 1. Spatio-temporal pattern



I will now cite the different models I found that cope with spatio-temporal patterns.

I started by looking at the synfire chains developed by Abeles and others, which seems very promising.

These are a complex construction of what could explain the observation of multi-electrode recordings. A central concept in the synfire chain is the node, a group of neuron that fires synchronously and activates the next node. The fact that a node is made of many neurons, added to the many converging/diverging links from one node to the next assures the stability of transmission for the waves of activity. For a complete description of the synfire chains, see [1] and [2].

There is however a problem that made me leave the synfire chain (for this time at least). From the moment that a neuron is activated as part of a node transmitting a wave of activity, his particular synchronization has no meaning by itself, it becomes 'noise' on the information conveyed by the entire wave. This supposes first that one should find a way to cope with the many nodes present in a network (so as to retrieve and insert information), given that a neuron can participate to several nodes. More than that, I should have found a mean of building these nodes and the chains of nodes in an unsupervised manner. The only article I know that tell about forming synfire chains in an unsupervised manner [14] points out that it needs control mechanism that grow rapidly.

I must recognize that the perspective of linking this with the particularities of the auditory system seemed to me too hazardous for the moment. Since then, more articles have been published about the learning capabilities of synfire chains and I should reconsider the question.

Anyway, the work done on synfire chains is of first importance in the comprehension of fine temporal structures in the cortex, and models able to perform a task should not be far away.

On another subject, I have a great debt to another group working in the same field, since the conceptors of the programming framework I used, SYNOD, actually work on synfire chains and designed the framework for experimenting hypothesis on synfire chains.

There is another model suggested in two articles for vowel recognition ([16] and [11]). The neuronal treatment is carried by a DYSTAL network (see [6]). DYSTAL is interesting because it is based on correlations in the inputs and is a temporal network.

I couln't use DYSTAL because of several reasons. First, it uses supervised learning. The notion of patch also seems to me a serious twist to the biological inspiration, because of the combinatorial number of patches necessary to associate the inputs, although the author claims for a neuronal equivalent for the network in general.

The use that is made of the DYSTAL network in the articles presupposes also the use of the formants and of a special type of non-linear filter, that was incompatible with the idea of using a cochlear pre-processing.

There is also the fact that DYSTAL is a commercial product. This is a serious drawback in a semester project since it means programming everything from scratch.

Nevertheless, the idea of extracting a particular feature from the data before doing association seems a good idea that is perhaps lacking in my work, since I'm doing the association with a straight network and no particular filtering.

2.3 Synchronization

Spatio-temporal patterns are based on fixed delays between spikes, and therefore the pattern is a general case of multiple synchronization.

There is an abundant literature about the importance of precise delays for the functioning of systems building perception, a good example is to be found in the bat ([18] Figure 11) for early perception, and the concept of cells assemblies in the cortex uses the synchronization to offer a solution to the binding problem.

My main source about synchronization in neural networks is [17]. There stands "Because neurons have a firing threshold and because synaptic potentials have a finite duration, signal transmission probability depends critically on the temporal structure of afferent input activity, synchronous input being the most effective of all possible constellations.(p364)".

I will consider that the network must code synchronizations and that the ability to assemble synchronizations together is sufficient to fulfill the goal presented above. The main tool to detect synchronization are the delays placed on connections.

2.4 Synapse learning

It is traditionnally considered that the learning in artificial neural networks is done through the weights. The biological equivalent to the weight is the synaptic strength (so it is in most cases). Here, synaptic strength means the peak post-synaptic current, thus is expressed in [pA]. I will consider that learning is purely synaptic.

From a succession of articles from a team with W. Singer ([7], [8], [9] and [10]), I drew the conclusion that long-term potentiation of the synapses occurs as a combination of conditions, using special chemical reactions, and is highly dependent of synchronization among the pre-synaptic neurons and between pre- and post-synaptic neurons, and that two thresholds must be crossed before the change take place. The crossing of only one threshold induces synapse weakening

I didn't modelized this type of synapse because I should have also modelized the inhibiting system that plays a first role in activating the potentiation system, and that it seemed to me too much work to be completed. I will only keep from these conclusions that learning occurs with the synchronization as a main parameter.

3 Structure of the elements

I will now describe the elements that compose the network, starting with the smaller level and building the global schema.

Many elements come directly from SYNOD. I didn't touched especially the simulation kernel. I will describe the elements I added, reviewing other elements for understanding purposes.

3.1 Neuron Model

The neuron model is the simplest I found ready to use in SYNOD. Its simplicity helps to draw some theoretical conclusions about the behaviour of the network and reduce the calculation time.

Lapique Model is a very simple integrate-and-fire neuron described in [21], the dynamical equation is :

$$\frac{dU}{dt} = \frac{EPSC + Iext}{C} - \frac{U}{\tau} \qquad \text{(EQ 1)}$$

and the neuron makes a spike when the membrane potential added to a gaussian

TABLE 1. Parameters of the Lapique Model

Parameter	Definition	Unit
U	Membrane potential	mV
τ	Decreasing factor of the dynamical equation	ms
t	Network time	ms
EPSC	Current arriving from the presynaptic neurons. This current has passed through the connections	pA
Iext	current from neurons not represented in the network (background activity)	pA
C	membrane capacity	pF
theta0	Threshold to cross to make a spike	mV
sDev	Standard deviation of the membrane noise	mV
t_ref_h	Refractory time where a neuron can't spike	simulation step

noise of deviation sDev crosses the threshold. After a spike, the neuron can't fire for the refractory time.

Crucial parameters are the refractory time, since it defines the behaviour of the saturated network, the threshold for controlling the sensibility, and the membrane noise and external current for controlling the activity without stimuli. These are the parameters that I tuned in the simulation runs.

There is a way of linking approximately the neuron model and the connections by calculating the minimal number of post-synaptic potentials necessary to trigger a spike (without the effect of the differential equation). This means to search :

$$\frac{n \cdot EPSC \cdot dt}{C} \geq theta0 \text{ and thus } n \geq \frac{C \cdot theta0}{EPSC \cdot dt} \quad \text{(EQ 2)}$$

When fixing the number of connections arriving to a neuron, one must look at the quotient between this number and the n calculated above because this quotient indicates the sensitivity of the network.

3.2 Layering system

I added to SYNOD a system to assemble many neurons into layers, and then to draw connections between layers, that are expressed into connection between individual neuron in an automatic manner.

By layer, I mean an ordered set of neuron, contiguous in my implementation, and fixed. Every layer has a number that is unique. Inside a layer, a neuron has a local index. Since the neurons are ordered, the layer do have a topology.

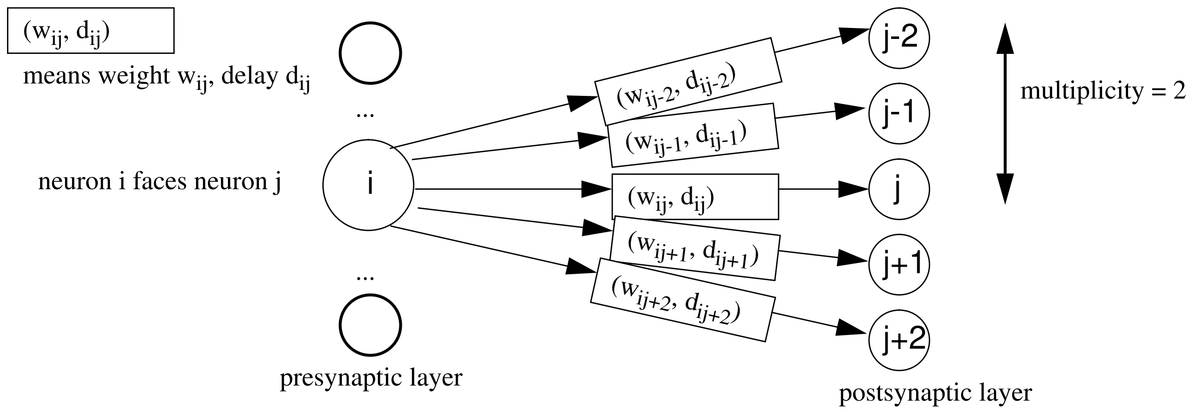
Between layers, I draw connections of the wanted type. By connection, I mean a way of drawing neuron-to-neuron connections between two layers. In a mathematical way, connections could be described as sets of neuron-to-neuron connections. Every connection is named with a unique number.

There are two types of connections, a feed-forward connection and a self-connection. The feed-forward connection is used to pile layers in a network, while the self-connection serves to build recurrent networks. I will now describe in more details how the connections are made. Remember that in SYNOD all the connections are made from the presynaptic neuron to the postsynaptic neurons (neurons ‘send’ connections). This appears graphically as a doublet on the line that connect the two neurons.

3.2.1 General features of connections

All connections are conic in my implementation, that is that a neuron on presynaptic side is only connected with a neuron on postsynaptic side that faces it and to the adjacent neuron on both sides according to the multiplicity of the connection. To define are the concept of facing neurons and the multiplicity

FIGURE 2. Conic connections



The borders are a delicate question for the connections. I chose to preserve first the number of connections that arrive in a neuron, which determines how many current this neuron can receive, and therefore how much it will spike. To preserve this I had to face post-neuron(0) with pre-neuron(multiplicity) and not with pre-neuron(0), and the same at the end of the layer (see Figure 3).

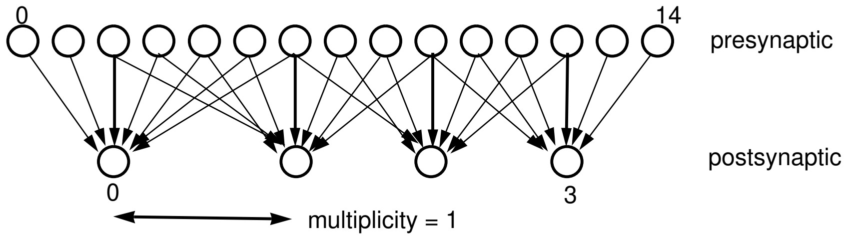
As a consequence, the borders of the presynaptic layer have less importance, since they send less connections. I can solve the problem by making the layer bigger that what is really interesting. The borders then have no particular meaning. Nevertheless, I have to take care that the border irregularities do not corrupt the dynamic of the network.

The concept of face-to-face neurons has a precise meaning. When two layers are connected, I have to put

$nbofNeurons(preSynaptic) - (2 \cdot multiplicity)$ face to $nbofNeurons(postSynaptic)$

and I do this by an integer division and by spreading the remain to distribute the facing neurons as regularly as possible.

FIGURE 3. Spacing the connections



What seem important to me here is that in a connection, the smaller layer is stretched to fit the bigger, and that the overlapping between cones is not constant, as seen in Figure 3, although the difference between minimum and maximum overlap is 1.

3.2.2 Feed-forward connections

In feed-forward connections, the presynaptic and postsynaptic layers are in principle different, but don't have to be. Feed-forward connections are useful to implement backprop-like structures.The values of the weights and delays are calculated by mean of 3 additional parameters (along with the multiplicity(mul) that belongs to connections in general), named base weight (W), min delay(minD) and max delay(maxD). Weights are constant, delays are linearly spaced between min and max.:

$$w_{ij} = W \text{ and } d_{i,\,j+k} = maxD - (maxD - minD) \cdot \frac{|k|}{mul} \quad \textbf{(EQ 3)}$$

The delays are not touched by learning. The reason to space them is to allow a temporal pattern to activate a neuron with shifted spikes. In a network where all delays are constant, the only way to be aware of correlation between spikes is the decay time of the membrane potential. Only different delays allow the precise detection of synchronization. I also had to decide between regular and random weights. Random weight seem to me not necessary since the spiking of a neuron already has a random component. The network will then have no trend to a degenerate solution, that is the main reason for putting random weights at the beginning of artificial neural networks algorithms.

3.2.3 Self-connections

Self-connections loop over a layer the same way feed-forward connections do between two layers. The same algorithm for spacing is used. The values for the connections are different however. Delays are fixed at a given value. Weights are calculated as a function of the distance between the two neurons (mexican hat):

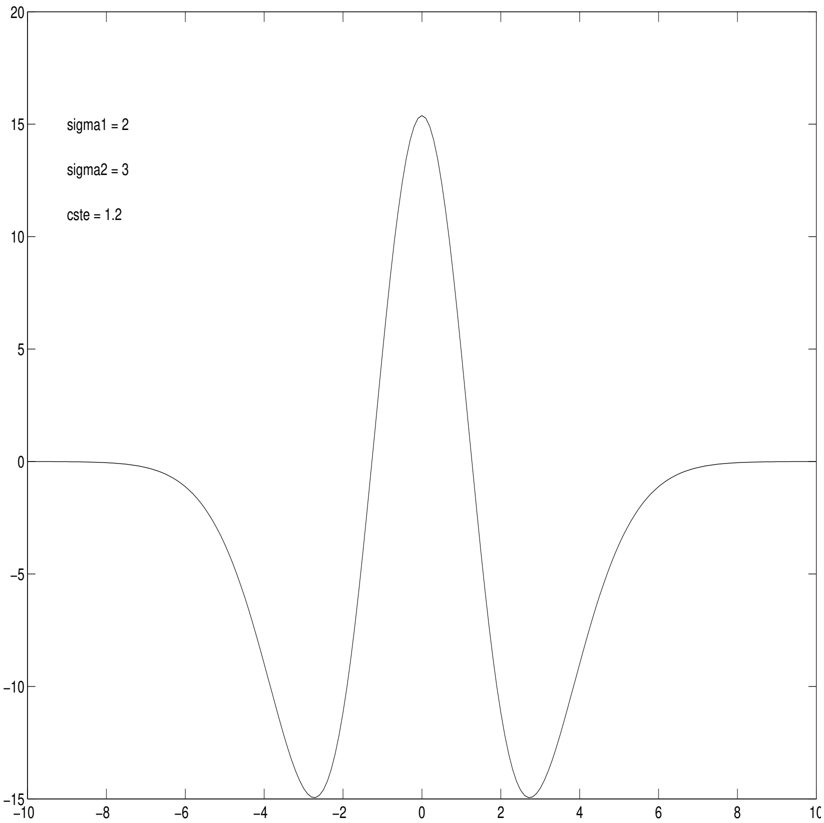
$$w_{i,\,i+k} = \frac{w_0}{f(0)} \cdot f(k) \qquad \textbf{(EQ 4)}$$

with

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma_1} \cdot e^{\frac{-x^2}{2\sigma_1^2}} - cste\frac{1}{\sqrt{2\pi}\sigma_2} \cdot e^{\frac{-x^2}{2{\sigma_2}^2}}$$

I designed this function with a type of curve in mind like Figure 4,

FIGURE 4. Curve of weight as a function of distance



but other behaviour are easy to obtain, like a saturating one with ($\sigma1 = 4$, $\sigma2 = 2$, cste = 0.48 and $w_0 = 5$).

3.3 Learning

Learning is a sensible point in my work. I must first express the regret that I didn't had some insight for this part from W. Gerstner, that teaches at the EPFL, and whose work is not used here, since I knew too late that he was working on this field (see [12]).

I did my best, though, to include my conclusions and to make things run.

Synchronization is the key of the learning process. Hereupon are based the weight updates. The learning algorithm has a local part and a global part. The global part tells where and when to make a weight update, the local part tells how to update the weights, once a local event has been defined.

3.3.1 Local part

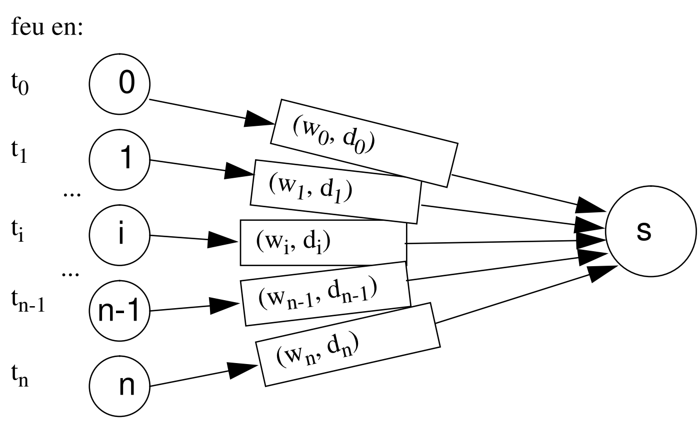
A local event includes all the inputs to the learning equations. A post-synaptic spike is the main part of a local event that is fully described by :

- the time of post-synaptic spike, t_s,
- the times of pre-synaptic spikes, from neurons n_i, that drove the post-synaptic spike, t_i,
- the delay of the connections from the pre-synaptic neurons, d_i,
- the strength of these connections, w_i,

The synchronization is expressed by $\Delta_t = (t_s - (t_i + d_i))$, $\Delta_t = 0$ meaning that the post-synaptic neuron fires just when the EPSP has arrived.

Not all the connected neuron are considered but only those who activated the synapse to s in the neighbourhood of t_s. An exact interval is not computable because of the dynamical nature of the differential equation.

FIGURE 5. Local event



Then, update is calculated by separating the pre-synaptic neurons in two sets, using a criteria :

$$B^p = \{i | b_i - \hat{b} \geq 0\} \text{ and } B^m = \{i | b_i - \hat{b} < 0\} \quad \textbf{(EQ 5)}$$

$$\text{with } b_i = p_i \Delta_i; \quad \Delta_i = t_s - (t_i + d_i); \quad \hat{b} = \frac{1}{n}\sum_i b_i \quad \textbf{(EQ 6)}$$

Conservation of the total weight arriving to neuron s is important, so the learning step will only take weight from one side to put it on the other side. Let α be the learning factor. I want to add a total of αP to the synchronized neurons, taken from the others (P is the sum of the weights of the activated synapses arriving to s). I use the B^p and B^m to make the partition. That is,

$$p_i = \left(1 + \frac{\alpha P}{P^p}\right) p_i (i \in B^p) \text{ and } p_i = \left(1 - \frac{\alpha P}{P^m}\right) p_i (i \in B^m) \quad \textbf{(EQ 7)}$$

$$\text{with } P^p = \sum_{i \in B^p} p_i \text{ and } P^m = \sum_{i \in B^m} p_i$$

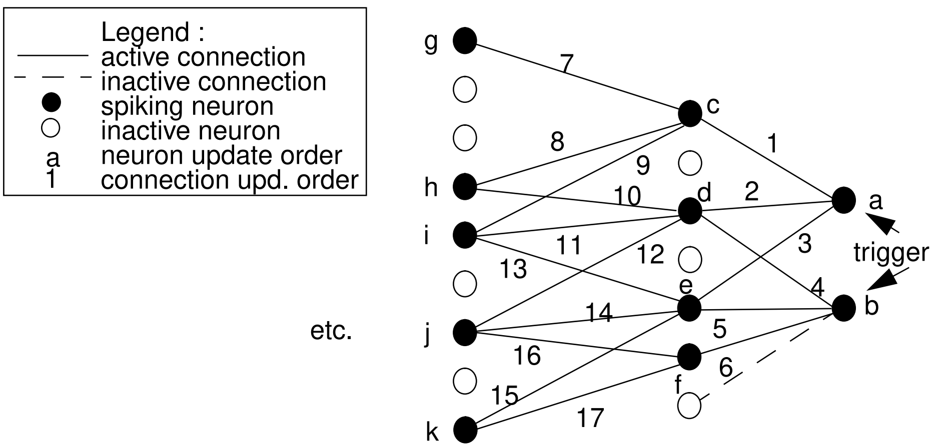
In case I have $P^m < \alpha P$ I only substitute αP with P^m in (Eq. 7) and the second part equals 0, meaning that the synapse is wiped out.

3.3.2 Global part

Learning is a recursive process, and each neuron i will become the s in a following recursive local event. It has no meaning however to update many times the same neuron because it has helped to activate more that one neuron in the following layers.

In other words, a learning step is the reinforcement of a partial graph of connections that has generally not a tree-structure. The problem is to walk through this partial graph without updating two times the same neuron, given that the reinforcement must be uniform on all the connections of the partial graph.

FIGURE 6. Illustration of the global part



Using the fact that the graph is layered, I can find the sets to update recursively by taking only the union of the learning neurons designated by the next layer.

Let the layers be sets of neurons :

$$L_j = \{n_i\}, j = 1, 2, \quad , l; i = 1, 2, \quad , n \qquad \text{(EQ 8)}$$

and the learning sets subsets of the layers :

$$LS_j \subseteq L_j \qquad \text{(EQ 9)}$$

The activating set of a neuron is the set of neurons that drive the spike :

$$Act_{jn_s} = \{n_i | \Delta_i \in [0, \Delta_{max}]\} \subseteq LS_j \qquad \text{(EQ 10)}$$

Then I take

$$LS_{j-1} = \bigcup_{n_s \in LS_j} Act_{jn_s} \qquad \text{(EQ 11)}$$

And I can define all the subsequent sets in a recursive manner from a first set on the last layer.

3.3.3 Triggering method

My idea is that learning is a discrete event that is done when something interesting occurs. I need a criterion to decide when is something interesting occuring, and that is my triggering method. This method should be calculated on the basis of the output of the network, since only the output can be interesting or not (I put aside the case where partial treatments are used).

I did a very simple system, since the trigger are contiguous neurons that synchronously fire in a given interval of time, say 1 ms for example. There must be more than a given threshold.

I thing that being able to trigger through non contiguous groups could be very interesting. Maybe in a future work.

4 Programs

I will now describe the code that performs the simulations described in the previous part and implants the structures. This part should also serve as a guide to the user of SYNOD/SLI and of what I added.

There are very good references for SLI, produced by the creators like [3] for the usage, and [4] for the internals. I will not discuss what is strictly SLI, unless it is necessary for understanding what I have added.

When speaking of the version of SYNOD/SLI that I designed and compiled, I will write SYNODa to reference my version of the code.

I did not add any new type to the language, but mainly functions, and some new instances of existing types. For the functions, you can find in Annexe B a description with the same format as used in [3].

I divide a SLI program in three parts :

- Structure declarations : number of neurons, types, connections
- Method declarations : Recording, Learning, Histogram
- Trials : the simulation loop

I will precise in which part a command should go, because as SLI is interpreted, there are dependencies to respect in the succession of the instructions.

The new instances I added are :

- LecteurModel, a new type of neuron model
- Lecteur, a new type of neuron
- DetecteurModel, a new type of neuron model
- Detecteur, a new type of neuron
- LearnMonitor, a type of Recording Method
- LearnUpdater, a type of Manipulation method
- WeightsHistogram, a type of Recording method

I will now go through the groups of modifications I did.

4.1 Structure declarations part

4.1.1 LecteurModel

When I took the source of SYNOD, there was no mean to bring externally recorded stimuli (as list of spikes) to a network. To do this, I added a new neuron, whose spikes are those recorded in an external file. Since this neuron can be connected with the others like a simulated neuron, it provides a simple way to feed the network in any point with complex stimuli.

The description of a particular binding with external stimuli must enclose the description of the external stimuli, the correspondence of the time basis and the correspondence between external and internal neurons.

As I implemented it, the description is in a .gdf file, thus providing the same file type as the output files, the neurons are given by their index number; the time basis cannot be stretched, but only shifted by a fixed number of time steps and the correspondence between internal and external neurons is made by attaching an external index to a neuron in the network.

The steps to follow (structure declaration part) are :

- Create the LecteurModel (CreateLecteurModel).
  Here are also given the input file and the time shift.
- Create the Lecteur neurons (CreateNeuron with proper modelid).
- Give each Lecteur neuron its external index (LecteurAssocier).

The neurons will then ask automatically for spikes to the model.

4.1.2 Detecteur model

The detecteur model is just another name for a second Lapique model. For implementation reasons, SYNOD cannot have the same model in two versions with two sets of configuration parameters (the models are static). It was interesting to test network models with two types of neurons, one as an integrator, one as a coincidence detector. I did another type to allow this but there are more elegant ways of doing that (next version). Detecteur model is used exactly like the Lapique model, see the SYNOD manual for description of Create - Delete functions, and Annexe B for SetDetecteurParameters. This is handled in the structure declaration part.

4.1.3 Layering system

The layers are continuous in this implementation, meaning that there is no layer broken into pieces. Layers inside SYNODa have a unique number (it is to the programmer to ensure this uniqueness is kept). This layer index is chosen at the creation and given each time a reference to a layer is needed.

It is important to understand that layers have no effect on the simulation itself, like grouping the neurons or so. They are only a useful abstraction to describe the connectivity in a shorter way.

To create a layer, just give the first and last neuron index to function CreateCouche. This command should go in the structure declaration part, after the creation of the neurons.

4.1.4 Connections

Connections also have a unique number (given by the user). They are created with all their parameters and cannot be changed after creation. The weights, if updated by the learning algorithm, will differ from the value set at creation, but anyway they cannot be retrieved inside SYNOD (but see Weights Histogram). The fact that there exist a connection between given layers will remain true until deletion of the layer system (DeleteConnections). As layers, the connections do not intervene into the simulation process itself. In other words, connections are a descriptive mean.

The creation of the connections occurs at the time the function CreateConnection is called. The user must ensure that at this time, the network, the concerned neurons and layers exist, for example, at the end of the structure declaration part. After the call, the connection exists and every neuron has the proper individual connections.

You can make cycles in the connections, but the mixing with the learning will not run. Theoretically, the learning global algorithm needs the graph to be layered. Otherwise, everything is possible. Remember that you have the self-connections to design recurrent networks.

When creating a self-connection, you must know which parameters for $\sigma1$ and $\sigma2$ will deliver the wanted function for the weights. Calculating the parameters from function's characteristics was too complicated.

4.2 Method declarations part

4.2.1 Learning

The learning system is divided in three pieces, following the SYNOD distinction between recording methods, that collect data from the state of the network, and manipulation methods, that change the state in an active way. So I did a recording method LearnMonitor, that remembers the recent history of the network (spikes), and decides when to make a learning step, and a manipulation method LearnUpdater that performs the learning step provoked by the LearnMonitor. I said three pieces because there is another object called SsCoucheT that implements the propagation rule of the learning and that is invisible to the SYNODa user.

I can use a breadth-first search algorithm to update one layer entirely before going to the precedent layer (I go from the last to the first layer) and to avoid updating many times the same connections. For each local event, the t_i are found by examining the recent history of the neurons connected to s.

If the nearest spike is too far, the neuron is considered inactive and not involved in this local event.

Then the local event is treated, meaning that the weights are updated and the driving neurons added to the list of neurons to update.

The process stops on the first layer.

Each time a learning step occurs, the learning is disabled for a short time (a kind of refractory time).

I put 4 parameters to control the learning process, two for the LearnMonitor, two for the LearnUpdate.

For the LearnMonitor are the size of the minimal contiguous spikes that will trigger a learning step and the refractory time. For the LearnUpdate are the alpha parameter that tells how much of the weights will be moved and the Δ_{max}, explained in Section 3 on page 7.

To design a learning experiment you must do the following in the method declaration part :

- create a LearnMonitor (CreateLearnMonitor)
- attach it to a recording session (AddMethod)
- Create a LearnUpdate (CreateLearnUpdate)
- attach it to a manipulation session (AddMethod)

You can switch the learning on and off with the SwitchLearning function, 0 meaning off, 1 meaning on.

4.2.2 Weights histogram

To have an idea of the weight distribution, I added a method that counts the connections in bins at fixed intervals. The parameters to give are the interval of weights to examinate, the number of bins for the count and the interval for spacing the samplings. Each method has an attached file. This method will put in the attached file one line per sampling, containing one integer for each bin.

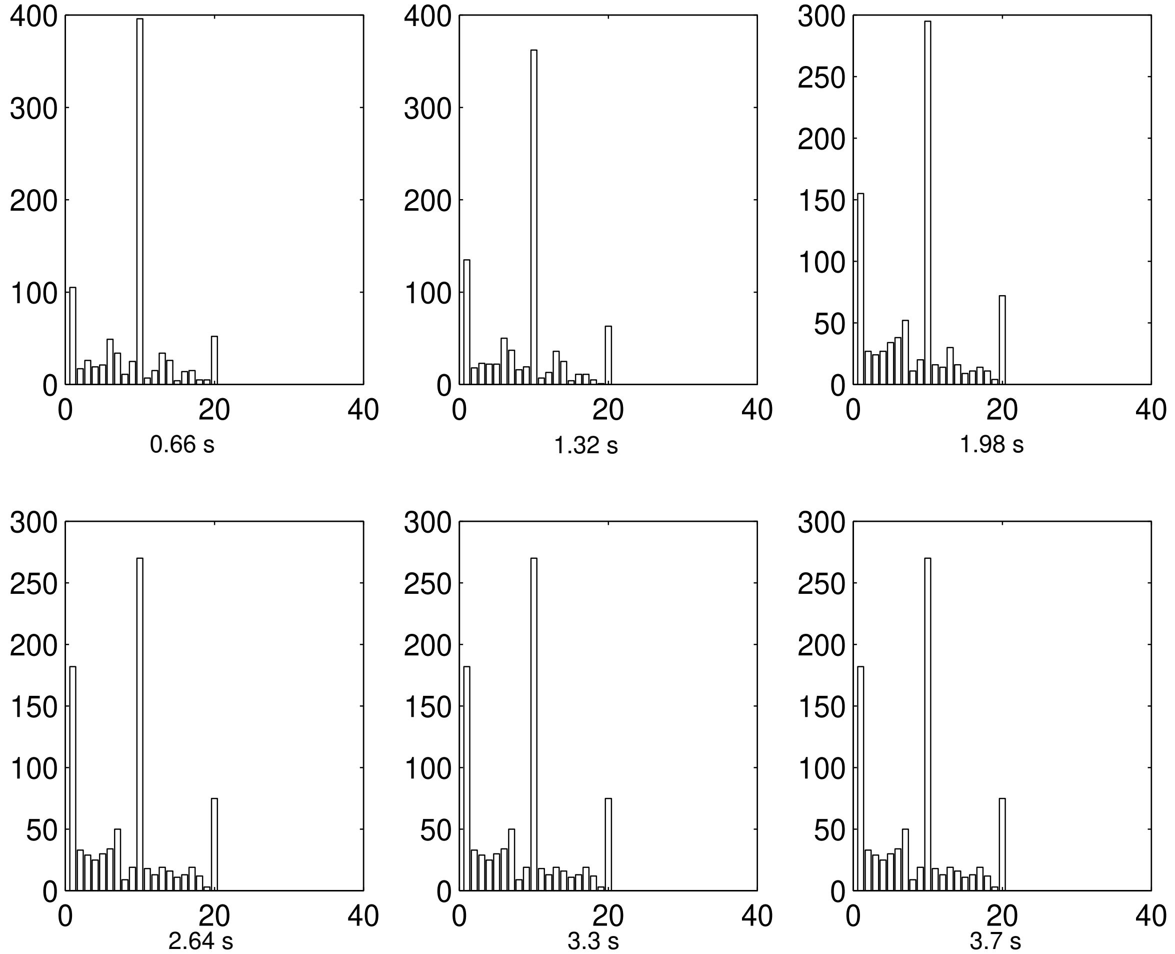
You must therefore :

- create the method (CreateWeigthsHistogram)
- attach it to a recording session (AddMethod)
- attach a file to it (SetFile)

to use this method.

The result looks like that (under Matlab):

FIGURE 7. Weight histogram from Figure 14 (20 bins from 0.0 to 100.0)



4.3 Code structure

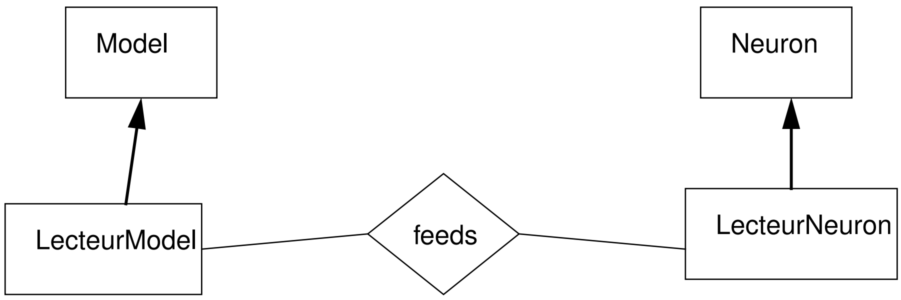
All the programs are in C++. I inserted the additional mechanisms into the SYNOD framework as new objects, inheriting from the existant structure whenever possible.

All the commentaries inside the code are in french, since I programmed with variable names in this language. Practically, couche stands for layer, connexion for connection, the rest should be self-explaining. The packages are the same as in previous section.

4.3.1 LecteurModel

First, the LecteurModel makes use of the distinction between a neuron model and the neurons themselves, using following object model :

FIGURE 8. LecteurModel objects



Basically, the lifetime of a LecteurModel can be conceived as :

LecteurModel := Creation.PutFile.[BufferCycle]

BufferCycle := FillBuffer.[GiveTiming]

There is no specially complicated piece. One should care at the timing and the proper matching between external time references and internal time, so that a spike described in the input file occurs at the same time step as it is written.

The files concerned are lecteur.h and lecteur.cc in (SOURCE)/xavier

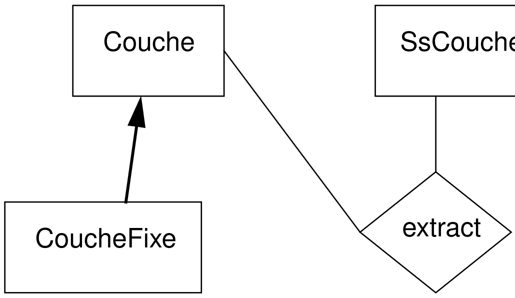
4.3.2 Layering system

The layering system did not exist before. I choose to implement it first as a descriptive mean, that is that the proper connections are made with SYNOD means between each neurons and are seen like that by the simulation kernel. Only at creation and special manipulations, some functions operate on the entire layer and update each particular connection.

The implementation is very simple, since I only allow contiguous layers, that need one start index and one end index. The layer has a copy of the neuron pointers, taken from the global neuron vector of the network.

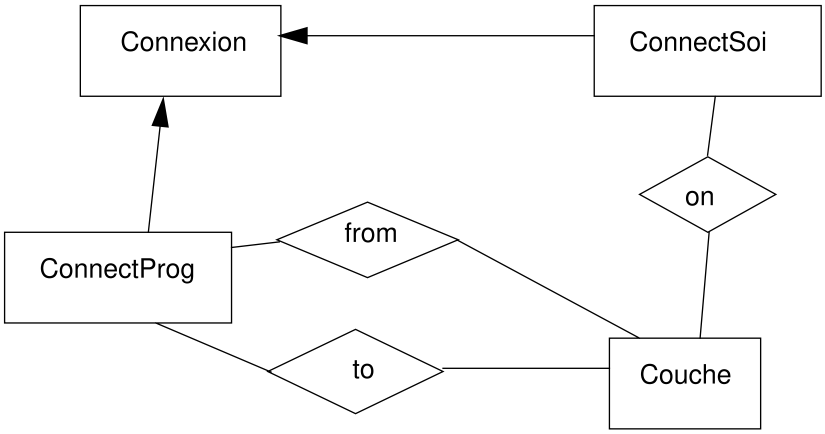
The SsCouche class is also a very basic implementation with just an iterator. It is derived in learning (see below).

FIGURE 9. Layer objects



The connections are made quite the same way, with an abstract class at top and just one basic implementation for each type of connection.

FIGURE 10. Connection objects



I designed a container class for all the layers, named SystemeCouches, that stores the objects in SLI and transform layer indexes into layers.

4.3.3 Learning system

The learning system, as I said before, is divided in three. One part collects the information and detects the learning triggers, one part lauches a learning step on the detected trigger and the third part implements the learning step with correct algorithm, on request from second part.

VoitApprend. The first part derives from the recmethod. The class has a vector for each neuron that contains the recent history of the spikes (class RapidVec). Using the history, it detects the triggers very easily by looking the differences in spiking between a potential flat zone (trigger) and the next neuron.

ApprendMnp. The second part derives from the manipulation method. The method tests for an existing trigger, gets it and lauches a QueueSsCoucheT on it.

SsCoucheT. The third part derives from SsCouche, and has access to the intrinsics of the Couche class. A instance contains all the neurons that triggered a particular spike in the post-synaptic neuron. The weights are update at object creation. After creation, the object serves as a buffer for remaining neurons to treat that will feed the queue. The QueueSsCoucheT is a specialized container class with a Lisp-like structure.

4.3.4 Weights histogram

The object derives from the recmethod. It simply waits for regular intervals and counts, when activated, the connections in bins and dumps the result in his standard output file. The existence of the InvNet object, that presents the network as connected from post-synaptic to pre-synaptic neurons makes the work easy.

5 Results

I did not (by far) explore all the possibilities of parameter tuning and network design offered by SYNODa.

I did notice some properties that affect the behaviour of the network. The problem is that I have to date no law that will predict that a given network has correct behaviour, even for such basic properties as total gain.

5.1 Tuning the parameters

My first goal was to obtain a network without learning, with good sensibility. Tuning the parameters is a fastidious work, and I will now try to group the parameters to explain in which order and for which reason I did the choices.

- Neuron parameters : for a complete description, see [21]
  theta 0 : spiking threshold
  sDev : noise standard deviance
  C : membrane capacity
  tau : membrane time constant
  t_ref : refractory time
- Network parameters :
  number of neurons
  number of models
  shape of the network
  maximum delay time
  computation step size (see [3]p 31)
  observation step size
- Connection parameters
  type of connection
  multiplicity
  delays
- Learning parameters
  alpha
  maxDistance

5.1.1 Fixed parameters

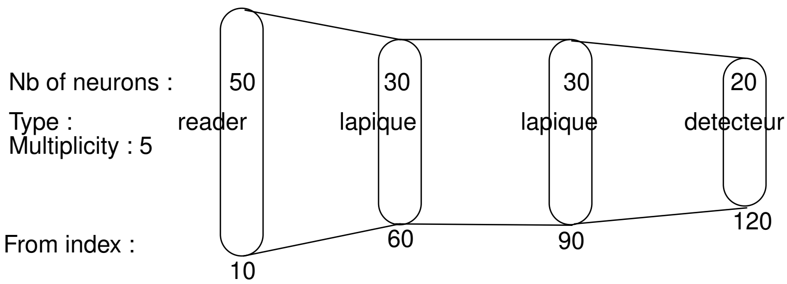
First, I took some parameters as I found them in the SYNOD package, considering that they have a biologically plausible value and that I had no reason to change them. These are C = 3ms (neuron parameter), the maximum delay time, 10 ms, the computation step size, 0.1ms and the observation stepsize, 0.1 ms (network parameters).

I choose not to put background noise nor external noise, and the membrane noise has two values, one for each type of neuron. The integrator neuron has higher membrane noise to allow a wider range of input energy to activate the network.

The number of neurons in the network is constant in SYNOD. The number of neuron models is also a constant. For me, the number of neurons is driven by the shape of the network and by the created layers. I had three neuron models, one for reading from a file, one as an integrator neuron (Lapique model) and one as a coincidence detector (Detecteur model).

After some tries, I designed a very simple network with 4 layers of decreasing size connected in a simple chain. I tried with and without self-connection on the last layer.

FIGURE 11. Network shape



5.1.2 Tuned parameters

The connections are the most important variable parameters. They are the object of the learning algorithm.

The multiplicity must be chosen such that on one hand, it allows a stimulus on the input layer to have an effect everywhere on the output layer, and on the other hand that a fraction of the arriving synapse is able to trigger a post-synaptic spike. I had a multiplicity of 5 and a post-synaptic current of 45 pA.

The delays are delicate to tune. If they are all the same, there is no detection of synchronization possible. If they are too big, a strong border effect occurs. I did it with a range from 2 ms to 5 ms.

5.1.3 Parameter tables

Neuron parameters

Type neurone	Seuil	C membrane	tau	bruit (σ)	t. réfractaire
Intégrateur	8 mV	3 pF	5 ms	1.8 mV	3 ms
Détecteur	8 mV	3 pF	2.5 ms	1 mV	3 ms

Connection parameters

Type	feed-forward
Multiplicity	5
PS peak current	45 pA
minimum delay	2 ms
maximum delay	5 ms

Learning parameters

size of synchronous set	>=2
time without learning	5 ms
alpha	0.2
Δ_{max}	5 ms

5.2 Comments on the results

All the simulation data are found in Annexe A.

5.2.1 Sensitiveness and border effect

The network is sensitive to vertical shifts of the input pattern. Compare Sim12 and Sim13, whose input is 5 lines higher. In this case, the vertical shift of the input is found in the output, the group that is in Sim12(0.44, 125) is found at Sim13(0.44, 1), except when a wave of activity is on a border. Then the activity will be lost, like in Sim13(1.2, 120-130) that does not appear in Sim12.

5.2.2 Learning effects

About Sim14.

There are lost cells (72 73 for example) and saturating cells (132 133). Since the weights that arrive to a neuron are conserved, if a synapse draws all the sum, the neuron will be completely dependent of this connection (for exemple between 106 and 74 73).

Several connections are wiped-out by the algorithm. This behaviour is not useful, since a saturated cell can convey no information, neither can a cell that doesn't fire. I should put limits on the weights to avoid great disparities. On the histogram on Figure 7, the fact that the bin for weights 100 or more contains round 80 connections indicates a drift of the values (remember that 2 connections only with weight 120 can activate a neuron).

The discrimination after learning between pattern B Sim15(2.7,all), pattern D Sim16(3.7,all) and pattern E Sim16(3.4, all) is not clear, but the patterns are very similar, and I should look into the microstructure on more than one presentation to have a meaning about that.

About vowel AA on Sim18.

It is quite clear that the group in (0.28, 138) has been learned, as it appears in (0.51, 138) and in (0.77, 138). This group is contiguous, but if the spikes round (0.38,125), that appear in (0.61, 125) can be considered as markers, I shoud also take the sparse groups in consideration.

6 Conclusion

6.1 Ideas coming next

There are some remarks that come out from me trying to understand the behaviour of this network, about what could be done next.

- The learning by steps has the advantage of learning quickly. Every learning step is important and fixes some connections. The network converges rapidly, but I can expect that the generality of this method is quite poor. One solution would be to try a continuous algorithm, very near from a Hebbian rule (for example see [12]). The problem is then to ensure that the weight distribution will not degenerate and that the sum of weights will be kept stable (otherwise, the general gain of the network will diverge).
- For the same problem, a less substantial change would be to take another triggering method, for example to say that if the number of neurons firing in the last layer in a given interval size crosses a threshold, a learning step occurs. This would be nearer to the concept of cell assemblies (see [17], p354). But then the retrieval of meaningful information becomes harder.
- The network shape I used is very simple, perhaps too simple. It could be extended in at least two directions :

  First, to double the layers and make two pathways of information with eventual links between the two paths. This can be seen as an ‘hemispheric’ approach, as in [15].

  Second, to make layer larger and to organize it as a recurrent network (for example a cellular network, idea from Arnaud). I tried it with the self-connections, and it seems interesting to extract features from the input, but the tuning is difficult.
- Retrieval of meaningful information is a big problem with this kind of network, and the search of a criterion function that would be able to associate output spike trains with discrete characteristics is a major goal. Apart from making the ouptut clearer, it would allow to test the learning abilities on a quantitative basis.
- On the same point, I think that a post-processing is almost unavoidable. To keep on a neuronal line, a Kohonen classification net could be used, with an algorithm modified for time-dependent data. I saw something interesting in [13].

I have the feeling that the network I studied is only a piece of what can be attained. In fact, in every part that I treated, there is another improvement at hand. The problem is to find a guideline. And in fact, some theoretical conclusions would be very useful. This would be the next thing I would do, and I will I think.

One word last

Here ends my (hard) work. I am quite convinced that the pulse neuron model will have soon a great importance, not only because it is closer to the biological model, but also because of its interesting properties and first its fundamentally temporal nature, that is useful in many real world problems.

I did sure not bring any tangible result, but it is by weeding that this quite new field will become more familiar. And the road stretches far away.

I would like to thank some people, my supervisors, Markus Diesmann und Mark-Oliver Gewaltig, Stéphanie Clarke, Anne Bellmann, Anne-Claude Cottier and my neighbours for pulling me up in times of hard work.

7 Bibliography

[1] M. Abeles (1981), “Corticonics : Neural Circuits of the Cerebral Cortex”, Cambridge University Press

[2] M. Abeles, E. Vaadia, H. Bergman, Y. Prut, I. Haalman and H. Slovin (1993) “Dynamics of neuronal interactions in the frontal cortex of behaving monkeys”, Concepts in neuroscience, 4:131-158

[3] Markus Diesmann, Marc-Oliver Gewaltig, Ad Aertsen (1995), “SYNOD: An Environment for Neural Systems Simulations, Language Interface and Tutorial”, Rehovot and Bochum

[4] Markus Diesmann, Marc-Oliver Gewaltig, Ad Aertsen (1995),“SYNOD: An Environment for Neural Systems Simulations, Concepts and C++ Implementation Details”, Rehovot and Bochum

[5] Roy D. Patterson, Mike AllerHand, Michael Akeroyd, Jay Datta (1995) “Auditory Image Mode,l Release 7.1 : Read me first”, Medical Research Council, Applied Psychology Unit, Cambridge. Ftp : ftp.mrc-apu.cam.ac.uk/pub/aim

[6] D.L. Alkon, K.T. Blackwell, G.S. Barbour, A.K. Rigler, and T.P. Vogl (1990), “Pattern-Recognition by an Artificial Network Derived from Biologic Neuronal Systems”, Biol. Cybern. 62:363-376

[7] Susanne Bröcher, Alain Artola, and Wolf Singer (1991), “Intracellular injection of Ca++ chelators blocks induction of long-term depression in rat visual cortex”, Proc. Natl. Acad. Sci. USA, 89:123-127

[8] A. Artola, S. Bröcher & W. Singer (1989), “Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex”, Nature, 347:69-72

[9] Alain Artola & Wolf Singer (1987), “Long-term potentiation and NMDA receptors in rat visual cortex”, Nature, 330:649-652

[10] Andreas Kleinschmidt, Mark F. Bear, Wolf Singer (1987), “Blockade of “NMDA” receptors disrupts experience-dependent plasticity of kitten striate cortex”, Science, 238:355-357

[11] J. Rouat and M. Garcia (1995). "A prototype speech recogniser based on associative learning and nonlinear speech analysis", accepted for publication to the International Joint Conference on Artificial Intelligence, Workshop on Computational Auditory Scene Analysis.

[12] Wulfram Gerstner, Richard Kempter, J. Leo van Hemmen & Hermann Wagner (1996), “A neuronal learning rule for sub-millisecond temporal coding”, Nature,

383:76-78

[13] Josef Göppert, Wolfgang Rosenstiel (1995), "Neurons with Continuous Varying Activation in Self-Organizing Maps", in From Natural to Artificial Neural Computation, Lecture notes in Computer Science, 930:416-426, Springer-Verlag

[14] John Hertz and Adam Prügel-Bennett (1996), "Learning short synfire chains by self-organization", Network, 7:357-363

[15] Theodor Landis (1996), "Bilateral control in the human brain and robotic systems", FNRS No 32-37819.93 Intermediate report

[16] Jean Rouat (1993), "Nonlinear operators for speech analysis" in Visual Representations of Speech Signals, ed. M. Cooke, S. Beet and M. Crawford, J. Wiley and Sons. 335-340

[17] Wolf Singer (1993), "Synchronization of cortical activity and its putative role in information processing and learning", Annu. Rev. Physiol., 349-374

[18] Nobuo Suga (1990), "Cortical Computational Maps for Auditory Imaging", Neural Networks, 3:3-21

[19] Harvey M. Sussman (1995), "Neural Coding of Relational Invariance in Speech : Human Language Analogs to the Barn Owl", Neural coding of invariance, 631-642

[20] Harvey M. Sussman (1986), "A Neuronal Model of Vowel Normalization and representation", Brain and Language, 28:12-23

[21] Tuckwell, H.C. (1988), "Introduction to theoretical neurobiology Volume1", Cambridge Univerity Press

8 Annexes

A. Simulation Data

Here are gathered all the simulation data.

All the time diagrams are displays of the neuron spikes, one small line for each spike, with the time on the first axis and the neuron index (as in Figure 11) on the second. Times are in seconds. Lines separate each layer. Under the last line are markers for the beginning of the stimuli and for learning steps when learning is enabled.

Referring to a point in the datas, I will indicate the number of the figure and the position in terms of (time, index), like Sim12(0.25, 35) for the first input pattern on page 30.

For the signification of the arrows and signs, refer to the comment.

FIGURE 12. Artificial data, 1 presentation in order ABCDE, without learning, on lines 20-40

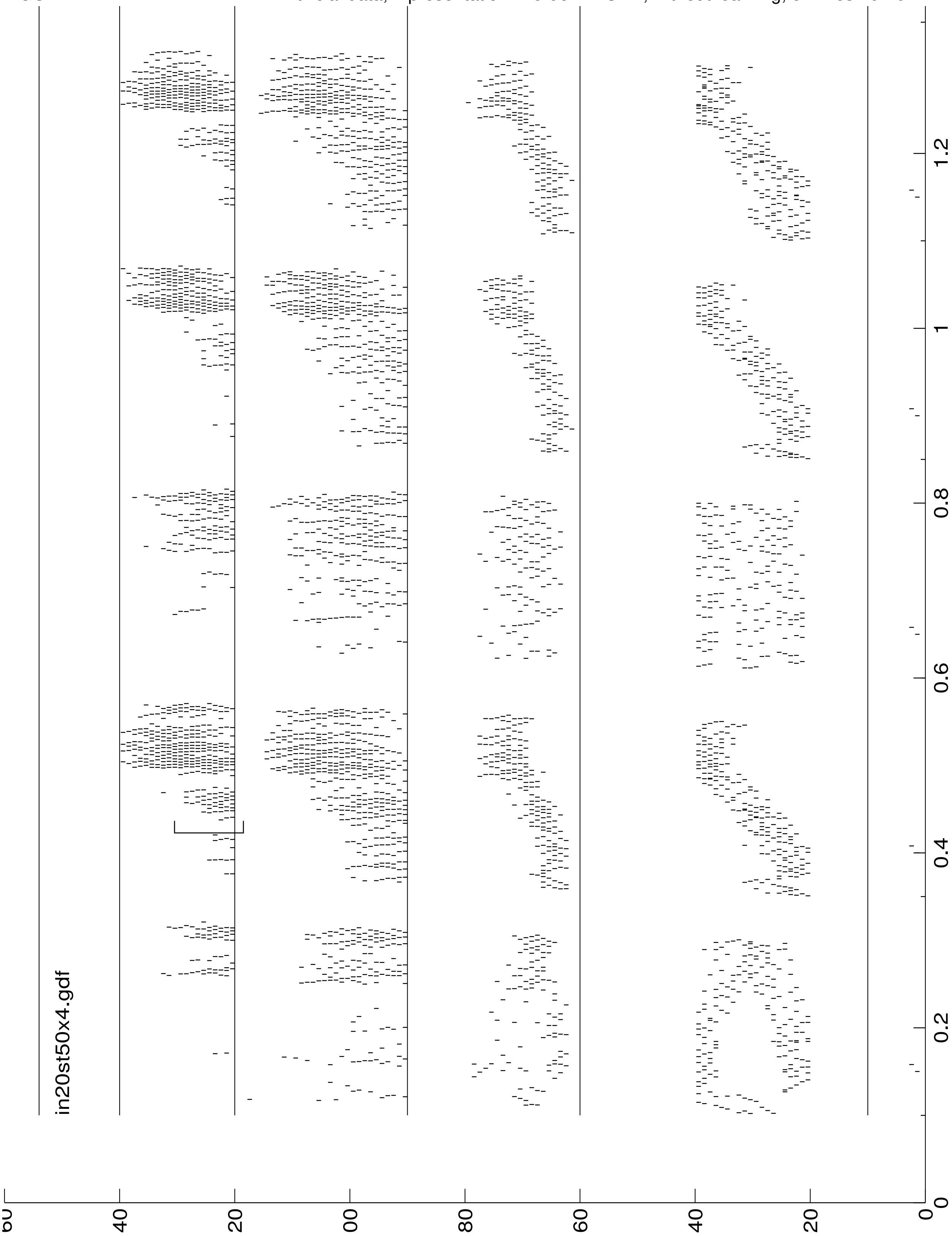


FIGURE 13. Artificial data, 1 presentation in order ABCDE, without learning, lines 25-45

FIGURE 14. Artificial data, 3 presentations, 2 with learning, 1 without

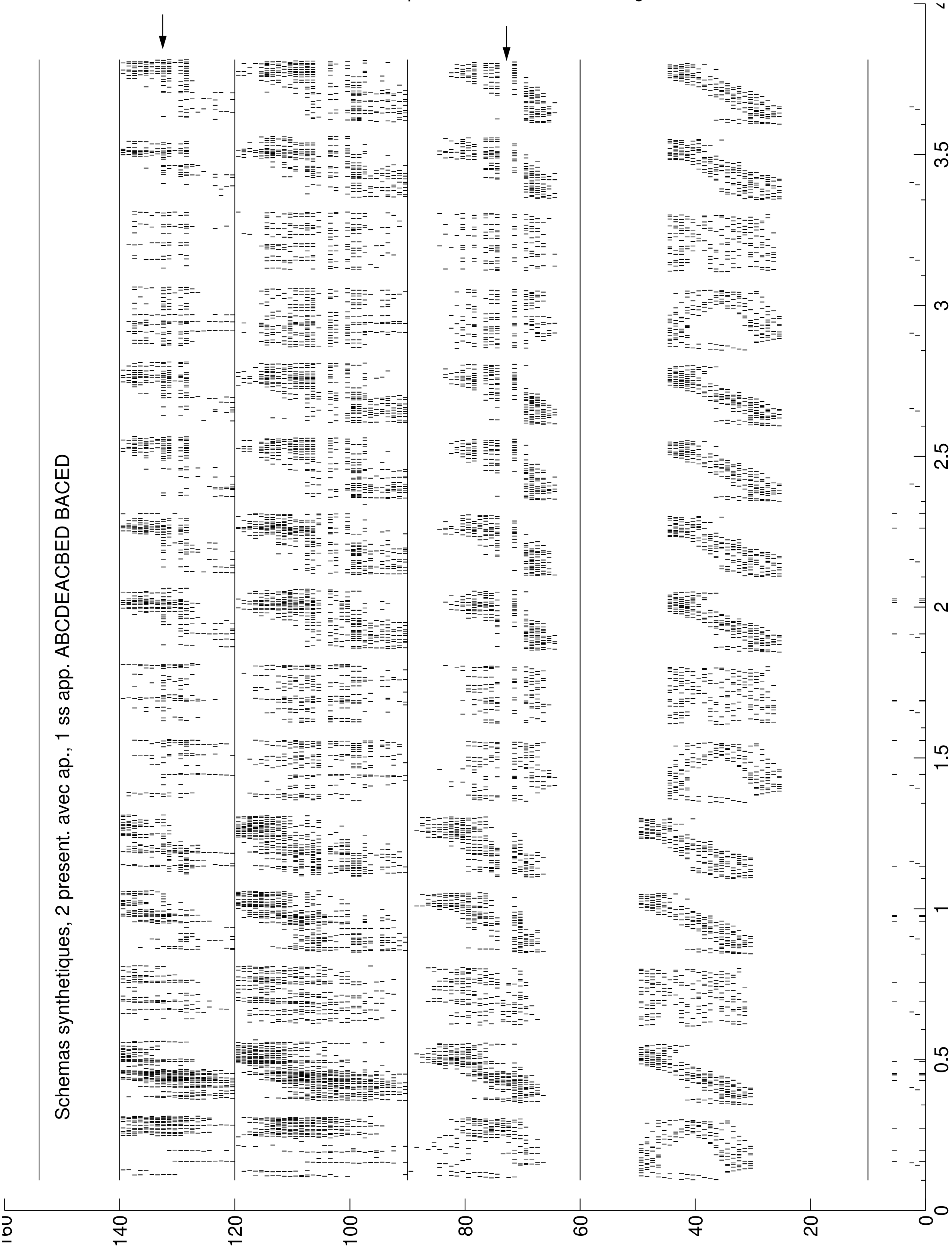


FIGURE 15. Artificial data after learning, zoom from Figure 14 on inputs A and B

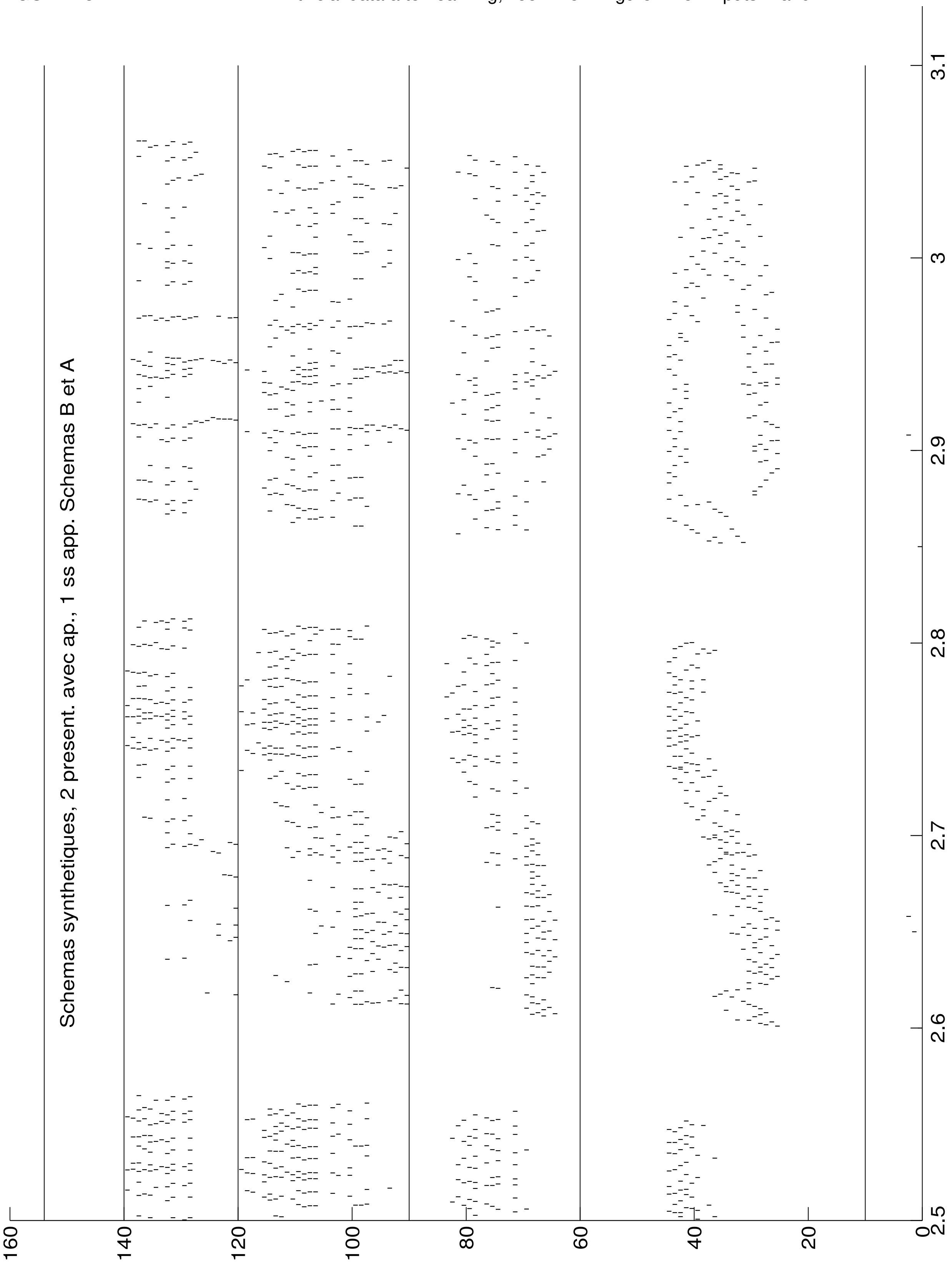


FIGURE 16. Artificial data after learning, zoom from Figure 14 on inputs C, E and D

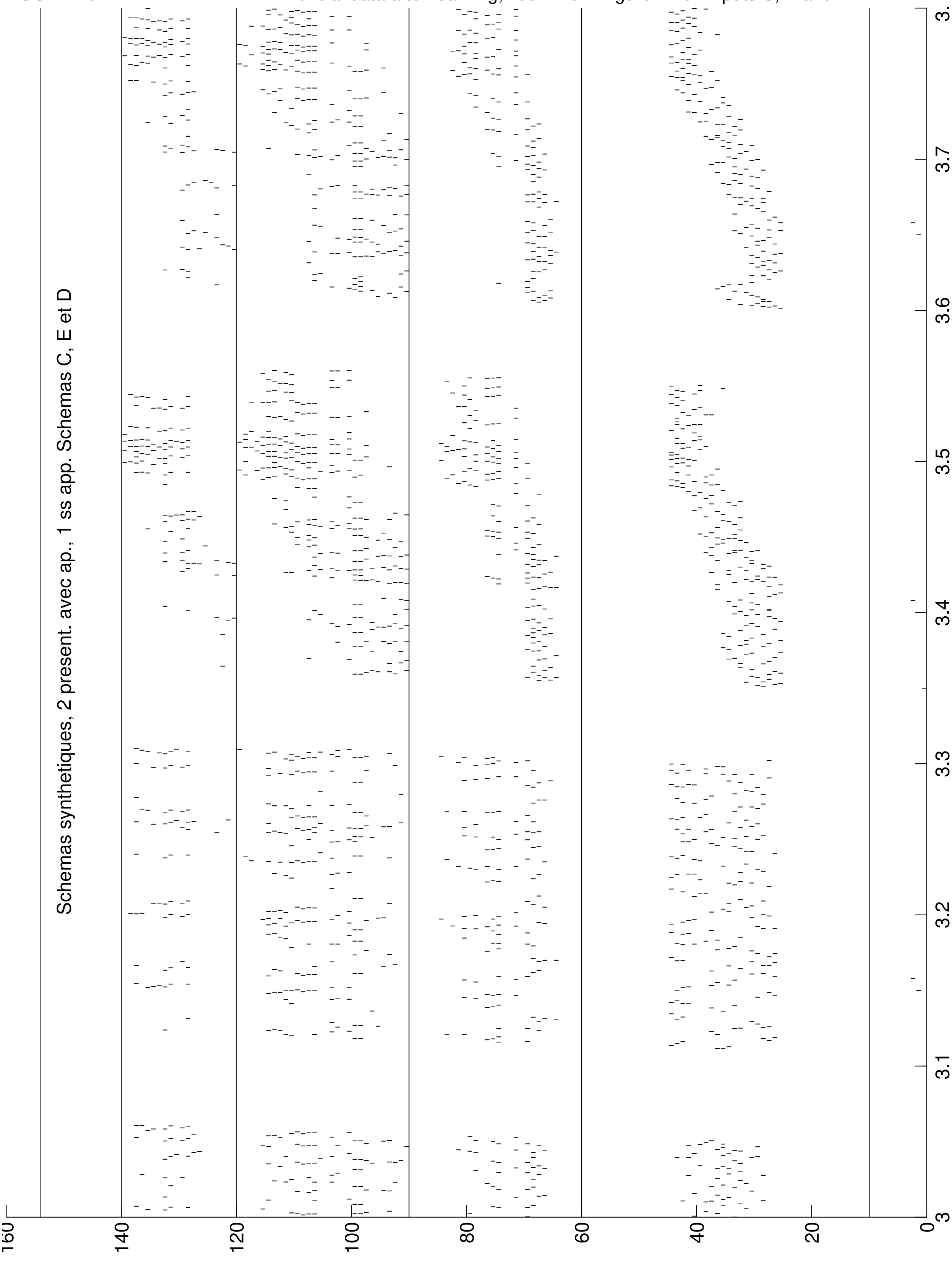


FIGURE 17. Vowels, without learning, 12 samples, lines 20-50

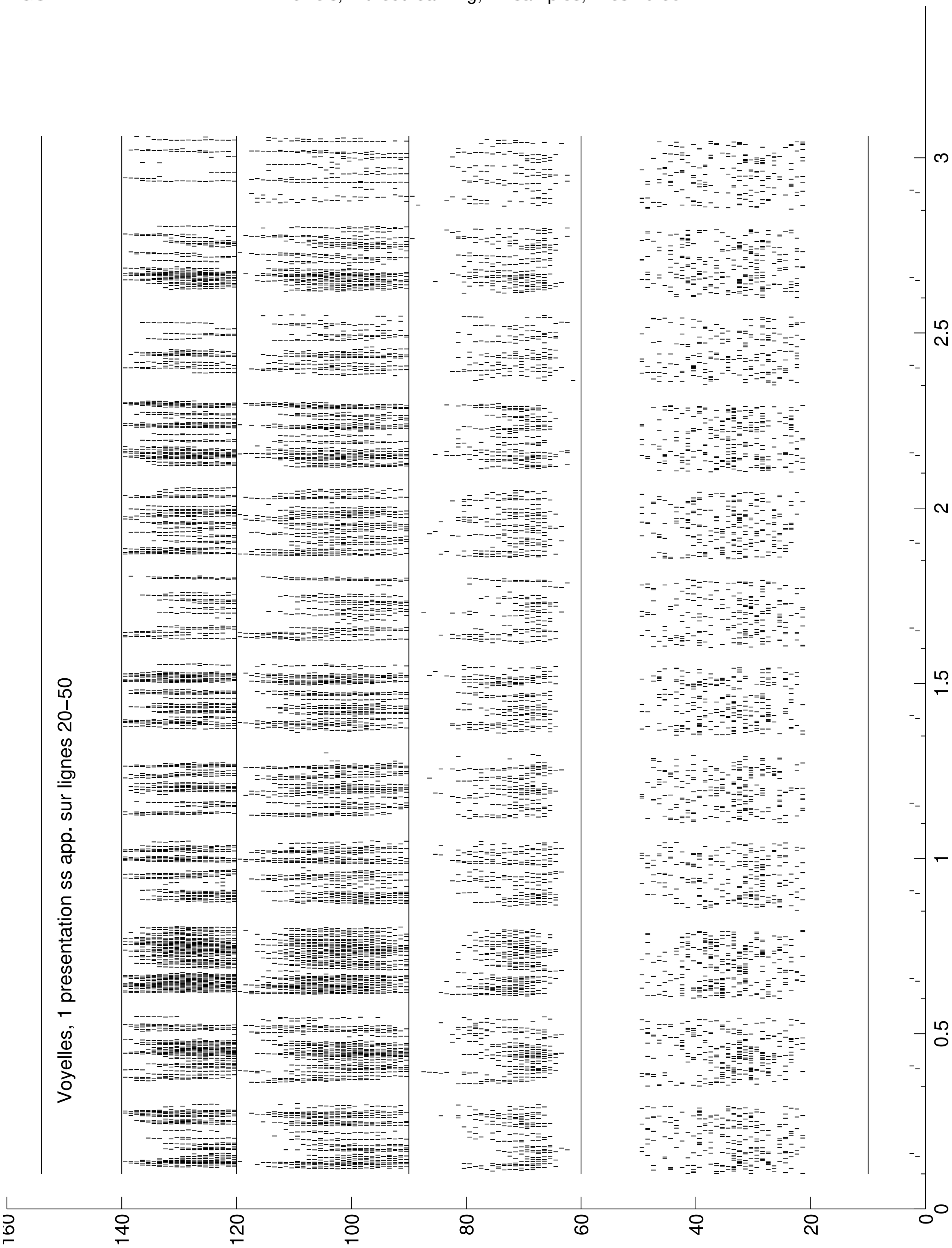
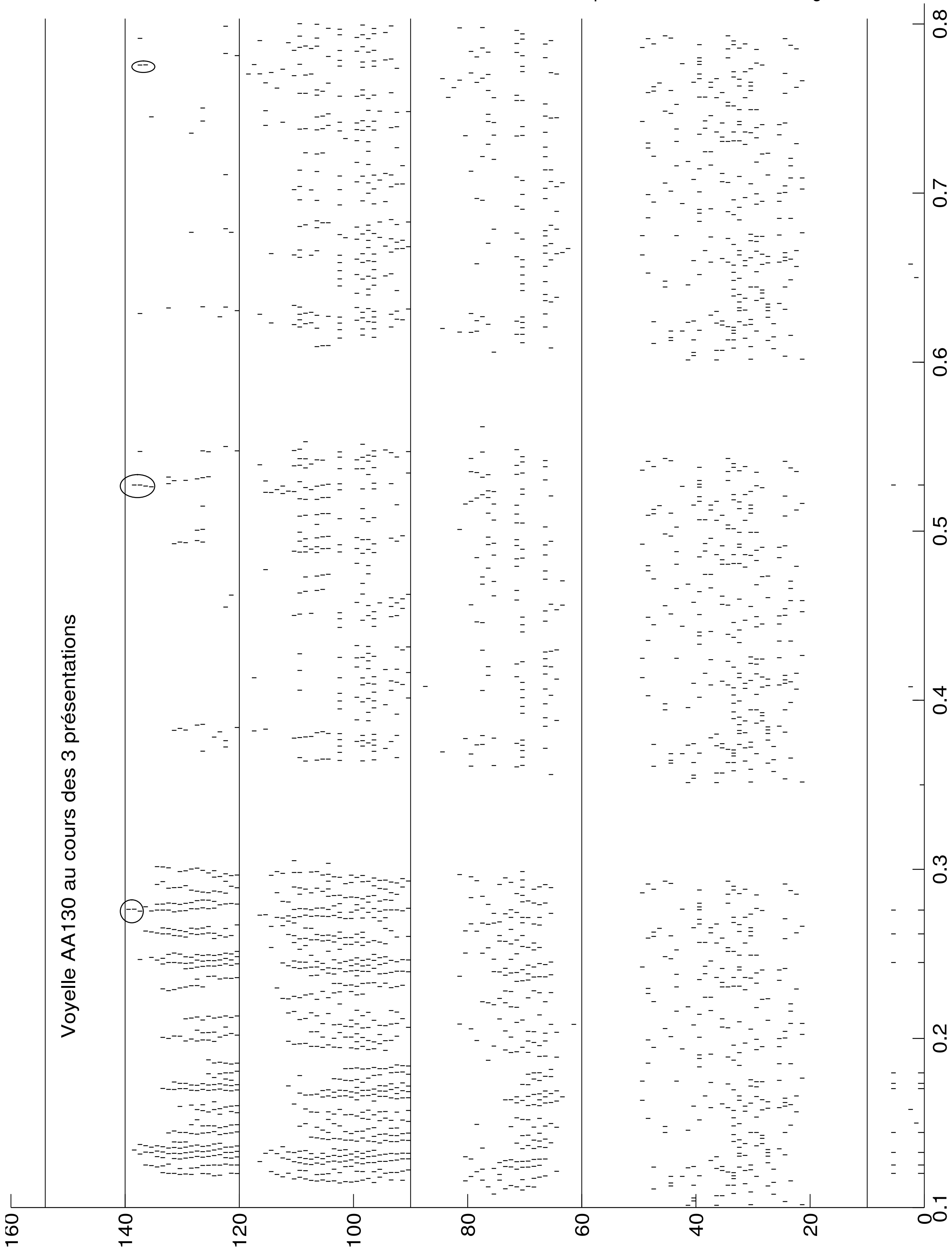


FIGURE 18. AA vowel, at different moments from a presentation: 2 with learning, 1 without



B. Additional function descriptions

CreateLecteurModel

stack levels	1
param.->res.	modelid ->

Description: Create the Reader neuron model. The reader neuron allows you to put spikes from a result file as input for the network.

SetLecteurModel

stack levels	2	1	
param.->res.	decalage	nomfichier	->

Description: Initialize the reader neuron model. Open the specified file to extract the spike timing, with the specified shift. Initialise le Modele du neurone lecteur en ouvrant le fichier de reprise des donnees spécifié, avec le décalage spécifié.

LecteurAssocier

stack levels	2	1	
param.->res.	refInterne	refExterne	->

Description: Associate a reader neuron (given by his number in the Network) with an event number in the file given to the reader model. Associe un neurone (donné par sa référence dans le réseau refInterne) avec un numéro d'événement dans le fichier donné au modèle lecteur (donné par refExterne).

DumpNetwork

stack levels	
param.->res.	->

Description: Print out information about the network and the models.

CreateCouche

stack levels	3	2	1	
param.->res.	noCouche	indexDébut	indexFin	->

Description: Create a bank with the neurons numbered from indexDébut to indexFin (inclusive). The bank will be designed by number noCouche, destroying every precendent bank. Crée une couche avec les neurones allant de indexDébut à indexFin compris, sous le numéro noCouche. Toute couche portant ce numéro est détruite.

CreateConnection

stack levels	7	6	5	4	3	2	1	
param.->res.	noConnexion	coucheSource	coucheDest	largeur	delaiMin	delaiMax	poids	->

Description: Create a conic connection between two banks, meaning that each neuron from the destination bank receives (2*largeur+1) connections from the source bank, coming from a cone centred ‘in front’ of this neuron on the source bank. The (arbitrary) definition of ‘in front’ is such that the bigger bank extends over the smaller. The connection is numbered with noConnexion. Crée une connexion qui portera le numéro noConnexion entre la couche source et la couche destination, désignées par leur numéro. La connexion est conique, ce qui signifie que chaque neurone de la couche destination reçoit exactement (2*largeur+1) connexions de la couche source d’une zone située ‘en face’ de lui sur la couche source. La signification de ‘en face’ est telle que les neurones de la couche la plus petite se répartissent sur la couche la plus grande. Il est évident que si la couche de départ est plus petite que la couche de destination, certains neurones auront les mêmes entrées (avec des délais et poids potentiellement différents toutefois).

CreateSelfConnection

stack levels	8	7	6	5	4	3	2	1	
param.->res.	noConnexion	couche	large	delai	poids	sigma1	sigma2	rapport	->

Description: Create a connection looping over a layer, with a constant delay and weights distributed with a ‘mexican hat’ peaking at poids, made out of two normal curves of variance sigma1 and sigma2, that is, a weight at distance d of the source is calculated with
W = C * (norm(0, sigma1) - rapport*morm(0, sigma2)).
Crée une connection auto-excitatrice, avec délai constant et poids distribués selon la distance entre l’origine et la destination de la connexion, selon une fonction ‘chapeau mexicain’ calculée selon la formule ci-dessus.

DeleteConnections

stack levels	
param.->res.	->

Description: Delete all the existing connections. To call once before exiting. Cette fonction détruit toutes les connexions et doit être appelée une fois avant de sortir.

SetLapiqueParameters

stack levels	5	4	3	2	1	
param.->res.	seuil	Sdev	capa	tau	refractaire	->

Description: Allows to set the parameters of the Lapique model. For the meaning of the parameters, see the docs of SYNOD. Permet la modification des paramètres du modèle de neurone Lapique. Cf les docs de SYNOD pour la signification des paramètres.

CreateLearnMonitor

stack levels	2	1	1
param.->res.	tau	seuilPlat ->	learnmonitor

Description: The learning needs two components, one to collect the states of the network and keeping a piece of past from the ongoing simulation. This function create this first component. L'apprentissage nécessite de connaître le passé à court terme du réseau en cours de simulation. Le moniteur garde cette trace.

DeleteLearnMonitor

stack levels	1	
param.->res.	learnmonitor	->

Description: Delete a learn monitor. Same as the other methods for usage. Détruit un collecteur de données d'apprentissage.

CreateLearnUpdate

stack levels	3	2	1	1
param.->res.	cstTemps	alpha	learnmonitor ->	weightupdate

Description: This function create the method that affect the connections following the learning algorithm. It needs a learnmonitor to get informations about the recent spikes. Alpha is the learning factor of the algorithm, cstTemps is the maximal distance of search. See the report for details about the parameters. Création d'une méthode algorithme d'apprentissage, qui va modifier les poids lorsque le collecteur de données donne le signal d'apprentissage. Alpha est le facteur d'apprentissage, cstTemps la distance maximale d'un potentiel pré-synaptique significatif. Pour la signification des paramètres voir le rapport.

DeleteLearnUpdate

stack levels	1	

param.->res.	weightupdate ->

Description: Delete a weight updater. Same as the other methods for deleting manip methods. Détruit une méthode algorithme d'apprentissage.

CreateWeightsHistogram

stack levels	4	3	2	1		1
param.->res.	maxValue	minValue	nbBins	intervals	->	recordmethod

Description: Create a histogram recorder. Every 'intervals', this methods dumps the distribution of weights in the method file. The distribution is made of 'nbBins' bins that cover the [minValue, maxValue] interval. Un enregistreur de distribution des poids est créé, avec nbBins répartis sur l'intervalle [minValue, maxValue]. L'enregistrement a lieu tous les 'intervals' pas d'observation.

SwitchLearning

stack levels	3	2	1	
param.->res.	learnmonitor	weightupdater	state	->

Description: Start (state = 1) or stop (state = 0) the learning algorithm. Met en marche (state = 1) ou arrête (state = 0) l'apprentissage.

CreateDetecteurModel

Description: This is the same function as CreateLapiqueModel, but for the Detecteur neuron model. Même fonction que CreateLapiqueModel, s'appliquant au modèle de neurone Detecteur.

SetDetecteurParameters

Description: This is the same function as SetLapiqueParameters, applied to the Detecteur neuron model. Même fonction que SetLapiqueParameters, appliquée au modèle de neurone Detecteur.