Applications musicales de cartes auto organisatrices :

jit.robosom

Mémoire

présenté en vue de l'obtention du Diplôme d'Études Musicales
en composition électroacoustique
au Conservatoire National de Région de Nice
réalisé au CIRM Centre National de Création Musicale
le 4 juin 2004

par Robin Meier

1 Remerciements

Je tiens à remercier :

Frédéric Voisin

Michel Pascal

Le CIRM et particulièrement François Paris

Laphroaig

2 Sommaire

2.1 Table des Illustrations

3 Introduction

Ce mémoire à pour objet de compléter et de documenter le travail que j'ai réalisé dans la recherche et le développement de jit.robosom. jit.robosom est une abstraction pour MaxMSPJitter[1] qui implémente un réseau de neurones artificiels (RNA) du type carte auto organisatrice.

Ce document est divisé en trois grandes parties :

- « L'intelligence artificielle » qui veut établir le contexte historique et scientifique du travail que j'ai mené
- « Réseaux de neurones artificiels » qui explique d'une manière générale les concepts et le fonctionnement de ce domaine de l'intelligence artificielle
- « jit.robosom » qui documente l'abstraction MaxMSPJitter et montre avec plusieurs exemples des applications possibles de cette nouvelle technologie

Cela est complété par les annexes et un CD-ROM contenant le programme et plusieurs exemples cités dans ce travail[2].

J'ai commencé ma recherche sur les RNA en été 2003 en Suisse, après avoir rencontré Frédéric Voisin[3], au CIRM[4] au mois de mai de la même année. Frédéric Voisin est le responsable du projet de recherche Neuromuse[5] du CIRM. Ce projet a pour objet l'étude et le développement de réseaux neuromimétiques pour la composition musicale, la synthèse sonore et l'analyse de données musicales.

En août 2003 j'ai fini ma première implémentation d'un RNA en Max (sans objets MSP[6] ou Jitter[7]) : « SOFM1 » et « SOFM2 ». Ces abstractions ont été publiées sur Internet en septembre[8].

Toujours en août 2003 j'ai commencé à collaborer avec Frédéric Voisin dans le cadre du projet Neuromuse, ce qui m'a amené à faire une présentation le 20 octobre 2003 lors du forum[9] de l'IRCAM.

Enfin, en novembre j'ai commencé à programmer jit.robosom, qui est le sujet principal de ce mémoire.

---

[1] http://www.cycling74.com
[2] Le contenu de ce CD-ROM est également accessible sur Internet http://robin.meier.free.fr et http://www.neuromuse.org/downloads/index.html#robosom
[3] http://www.fredvoisin.com/fv/index.html
[4] http://www.cirm-manca.org
[5] http://www.neuromuse.org
[6] MSP permet le traitement du signal et de l'audio en particulier
[7] Jitter permet le traitement de matrices et de la vidéo en particulier
[8] http://robin.meier.free.fr/neurob/
[9] http://forum.ircam.fr/

4 L'intelligence Artificielle

La définition de l'intelligence artificielle (IA) est controversée. Elle peut être envisagée selon deux points de vue complémentaires[10] :

- L'un concerne la simulation par un ordinateur des mécanismes de l'intelligence (humaine) pour tester un modèle ou une théorie ; c'est une démarche des sciences cognitives ;
- L'autre concerne les efforts faits pour doter un ordinateur de capacités habituellement attribuées à l'intelligence humaine, comme l'acquisition de connaissances, la perception, le raisonnement, la prise de décision, etc.

Ces deux approches sont largement complémentaires, l'avancée de l'une nourrissant l'avancée de l'autre.

Du fait de cette liaison étroite entre ces deux approches, l'IA se trouve à la croisée de différentes sciences comme les mathématiques, l'informatique, la biologie, la neurologie, la psychologie, etc.

L'expression « Artificial Intelligence » a été forgée en août 1956 lors d'une conférence à Dartmouth (USA) réunissant des personnalités clés de l'IA comme John McCarthy, Marvin Minsky et d'autres. Déjà, certains chercheurs s'opposaient à ce terme, plus particulièrement au mot « intelligence ». La remise en cause de ce terme est soumise encore de nos jours à de vives discussions. C'est toujours en 1956 qu'apparaissent les premiers programmes d'IA : le `Logic Theorist` un démonstrateur de théorèmes en logique, un jeu d'échecs ainsi que LISP, le langage de programmation phare de l'IA.

Mais cette naissance a été précédée d'une longue période de gestation qu'on peut faire remonter jusqu'à l'Antiquité. Dans l'histoire plus récente, on peut citer parmi les chercheurs les plus importants

- Georg Boole au XIX, inventeur de l'algèbre booléen qui est la base de toute arithmétique informatique
- Gödel et Church (1930) qui ont démontré qu'il existe une classe de problèmes dont la solution n'est pas algorithmique[11] (cf. chap. prochain)
- et enfin Alan Turing avec l'invention de la « machine de Turing » et de la « machine universelle » en 1936.

La machine de Turing est une machine extrêmement simple, mais qui permet (en termes de logique) de faire tout ce que peut faire un ordinateur d'aujourd'hui. Elle est le fondement, « l'Adam » de tous les ordinateurs.

C'est aussi Alan Turing qui plus tard a proposé un test permettant de décider si une machine est intelligente ou non. Le but de ce test est de distinguer un humain d´un ordinateur en posant des questions en aveugle. Selon Turing, un ordinateur peut

[10] Haton, J.P et Haton, M.C. : L'intelligence Artificielle ; PUF, p. 3
[11] Haton, J.P et Haton, M.C. : L'intelligence Artificielle ; PUF, p. 5

être considéré comme « intelligent » si l´interrogateur humain est incapable au bout de cinq minutes de déterminer lequel de ses deux sujets est la machine[12].

4.1 Approche algorithmique

Un algorithme est une suite finie d'opérations qui sont effectuées dans un ordre précis. Un algorithme sert à accomplir une tâche précise.

Une recette de cuisine est un exemple d'algorithme. Tous les pas à effectuer sont décrits précisément et dans l'ordre à respecter ; il n'y a pas de recette de cuisine composée d'un nombre d'opérations infini (sauf peut-être dans la cuisine sri lankaise). Si un problème nécessite un algorithme à un nombre infini d'opérations, ce problème est considéré comme incalculable[13].

L'approche algorithmique est celle de tous les ordinateurs. Un programme / un logiciel est en essence comme une grande recette décrivant des tâches précises et (normalement) l'ordinateur effectue ces instructions d'une manière très rapide.

Comme les opérations elles-mêmes ainsi que leur ordre sont toujours précis, un algorithme est complètement déterministe. Un algorithme fait toujours exactement la même chose de la même manière. Traduit dans un environnement temps réel musical c'est un système qui attend une entrée précise (p.ex. une note précise) pour déclencher une réponse prédéfinie.

L'application d'un algorithme pour jouer aux échecs par exemple consisterait à essayer tous les mouvements possibles et calculer le cas le plus avantageux. Comme les cas possibles sont trop nombreux, il est impossible de tout calculer dans un temps raisonnable.

Il faut mentionner que dans le monde physique, le déterminisme d'un algorithme est relatif. Même si théoriquement un algorithme est complètement déterministe, en pratique ce n'est pas le cas comme les recherches sur des systèmes dynamiques et chaotiques l'ont montré. Une fonction (un algorithme) récursive aussi simple que $x_{n+1} = x_n^2 + c$ donne des résultats non prévisibles si elle est itérée sur un ordinateur. Cette même fonction peut donner des résultats complètement différents sur deux ordinateurs différents après quelques itérations (c.f. Annexes 7.7). La cause de ce comportement est l'imprécision du calcul numérique, qui aussi petite soit elle peut rapidement rendre chaotique et imprévisible l'exécution d'un algorithme simple.

4.2 Systèmes experts

Une autre approche de l'IA est celle des systèmes experts, qui dans les années 70 était le champ principal de la recherche sur l'IA.

Un système expert consiste en une ou plusieurs bases de connaissances préalablement introduites par un expert humain. À partir de ces connaissances, le

---

[12] http://en.wikipedia.org/wiki/Turing_test et
http://www.cognition.ens.fr/~guerry/philosophie/dea/memoire_html.html
[13] http://en.wikipedia.org/wiki/NP-complete

système expert peut tirer des conclusions à l'aide d'un moteur d'inférences. Ce moteur d'inférence ou interpréteur est un système algorithmique qui fonctionne avec des opérateurs logiques comme « et », « ou », etc.

Un exemple d'un système expert est la reconnaissance d'une fleur à l'aide d'un livre (la base de connaissances) :

- Si la fleur est rouge et a 4 pétales, alors c'est un coquelicot avec une probabilité de 50%
- Si de plus elle se trouve dans un champ de blé, c'est un coquelicot à 95%.
- Pour en être sûr à 100% vérifiez qu'il y a 4 sépales.[14]

On voit que les systèmes experts peuvent être très efficaces pour des applications très spécialisées dans lesquelles on a déjà acquis beaucoup de connaissances, p.ex. pour diagnostiquer certaines maladies. Mais si on veut créer des systèmes plus généralistes la quantité de savoir (et souvent de sens commun) est abondante. Comme les connaissances dans la base de connaissances sont figées, l'approche des systèmes experts est plus adaptée pour simuler un géologue qu'un enfant de cinq ans.[15]

Il y a, outre l'approche algorithmique et celle des systèmes experts une troisième approche de l'IA : l'approche connexionniste. Dans les années 70 on a mis de côté la recherche sur cette approche, notamment la recherche sur les réseaux de neurones artificiels (RNA), à cause d'un article de Minsky et Papert[16], qui en 1969 ont montré des limites des RNA. Cela tombait dans une phase où l'armée américaine était en train de repartager les budgets de recherche et du coup la plus grande partie des investissements pour l'IA a été mise dans les systèmes experts. Mais à partir de 1975 des nouvelles architectures de RNA ont été proposées et les limites montrées par Minsky et Papert ont pu être surmontées[17]. Ce sont cette approche et son application à la musique qui constituent le sujet principal de ce mémoire.

[14] http://membres.lycos.fr/villemingerard/Logique/IAexpert.htm
[15] http://membres.lycos.fr/villemingerard/Logique/IAexpert.htm
[16] Minsky, M. et Papert, S. : Perceptrons ; Cambridge, MA, MIT Press
[17] c.f. chap. 5.2.2 Le neurone artificiel, figure 5.

5 Réseaux de neurones artificiels

5.1 Connexionnisme

La troisième approche de l'IA se résume dans le connexionnisme. Le connexionnisme est une voie des sciences cognitives. Les sciences cognitives, comme mentionné plus haut, sont les sciences du raisonnement, de l'intelligence et sont très inter-disciplinaire en faisant appel à la fois à la psychologie, la neurobiologie, la linguistique, la philosophie etc.

À l'intérieur donc des sciences cognitives se situe le connexionnisme, à la charnière entre la modélisation biologique, les systèmes dynamiques et le traitement du signal.

Il se fixe pour objet un réseau constitué de neurones formels dont les états évoluent au cours du temps en fonction de leurs entrées. Le fonctionnement individuel des neurones est assez simple, analogue à un petit relais électrique. Les caractéristiques des neurones et les valeurs des liaisons inter-neurones (liens synaptiques) définissent ce réseau de neurones artificiels [18].

5.2 Concept / Fonctionnement des RNA

5.2.1 Le neurone biologique

Les RNA s'inspirent du traitement de l'information par le cerveau. Après la découverte de la fibre nerveuse - le neurone - en 1700 on connaît depuis 1950 l'organisation des neurones : ce sont des unités indépendantes en contiguïté les unes avec les autres, créant ainsi un réseau extraordinairement complexe : 30 à 100 milliards de neurones, un milliard de milliards de connexions possibles, 600 millions de synapses par mm^3 !

L'intégrité d'un organisme vivant nécessite l'activité coordonnée des cellules qui le composent. La communication intercellulaire est assurée par deux grands systèmes : le système hormonal et le système nerveux. Ce dernier transmet l'information au moyen de prolongements cellulaires, les neurones.

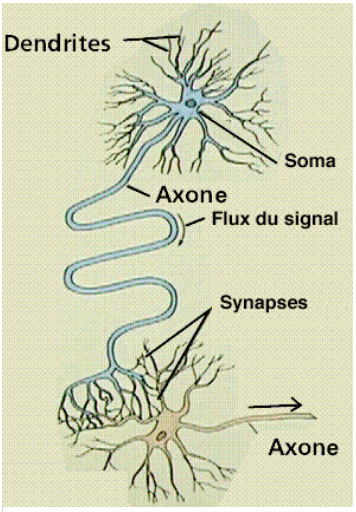

Fig. 1: Schéma de deux neurones en contiguïté

Le neurone est responsable de l'émission et de la propagation du message nerveux.

C'est une cellule « excitable », qui transmet et propage des signaux électriques en fonction des informations qu'elle reçoit.

C'est aussi une cellule « sécrétrice » très particulière, dont le produit de sécrétion est le neurotransmetteur. La sécrétion, très focalisée et dirigée uniquement vers les

[18] http://esm2.imt-mrs.fr/~dauce/these/node38.html

cellules avec lesquelles le neurone est connecté, se fait au niveau des synapses qui sont entre l'axone et les dendrites[19].

C'est grâce à des milliards de milliards de connexions entre les neurones que ce système devient très puissant en permettant notamment le calcul simultané (parallèle) de plusieurs opérations, à l'inverse de l'approche algorithmique qui exécute une opération après l'autre.

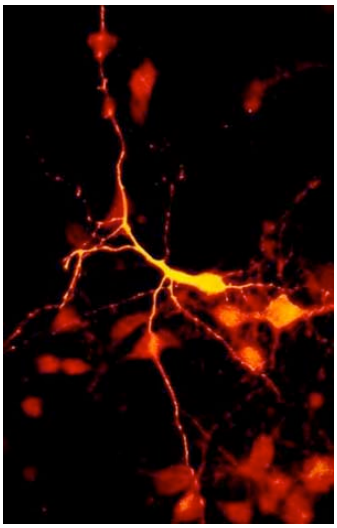

Fig. 2 : Image par immunofluorescence d'un neurone

Pour que ce système soit capable d'apprendre, il faut une modification durable du couplage entre les neurones. Ce couplage repose sur l'efficacité de la liaison synaptique, c'est-à-dire la facilité avec laquelle le neurone pré-synaptique peut influencer l'activation du neurone post-synaptique.

Les premières hypothèses sur l'adaptation synaptique ont été formulées par Hebb en 1949[20] :

« Quand un axone de la cellule A est assez proche pour exciter une cellule B et quand, de façon répétée et persistante, il participe à son activation, un certain processus de croissance, ou un changement métabolique s'installe dans une cellule ou dans les deux tel que l'efficacité de A, en tant qu'elle est une des cellules qui active B, est augmentée. »[21]

En 1973, l'intuition de Hebb est confirmée expérimentalement par la mise en évidence d'un processus de potentiation à long terme dans l'hippocampe[22]. En excitant électriquement un chemin d'activation existant, ces auteurs constatent que celui-ci tend à se renforcer, c'est-à-dire que l'excitation ultérieure du même chemin

[19] http://neurobranches.chez.tiscali.fr/neurophy/leneurone.html
[20] Hebb, D. : The Organization of Behavior ; Wiley, New York
[21] Traduit dans Pelissier, A., Tête, A. : Sciences cognitives textes fondateurs ; PUF
[22] Bliss, T.V.P., Lomo, T. : Long-lasting potentiation of synaptic transmission in the dendate area of the anasthetized rabbit following stimulation of the perforant path ; Journal of Physiology 232 (1973)

tend à produire une réponse plus intense. Ce phénomène se maintient de plus sur une longue durée[23].

5.2.2 Le neurone artificiel

S'inspirant donc de ce système naturel, les réseaux de neurones artificiels sont des réseaux d'unités indépendantes (neurones) travaillant parallèlement. Elles sont reliées entre elles par des nombreuses connexions (synapses) et peuvent ainsi traiter des informations / résoudre des problèmes complexes. Les unités peuvent être des unités formelles dans un logiciel – c'est-à-dire une simulation – ou bien des processeurs. Il existe en effet des ordinateurs neuroniques constitués de centaines de processeurs simples et interconnectés.

Comme dans le cerveau, les neurones peuvent être organisés en couches. Ils existent des couches d'entrée, des couches de sortie et des couches cachées.

Comme la loi de Hebb l'a montré pour les neurones biologiques, les synapses très actives augmentent leur efficacité. Dans l'informatique, cet effet se traduit par des poids qui désignent l'efficacité des connexions entre les neurones artificiels.

Fig. 3 : Un RNA (jit.robosom) de 16 neurones dont l'activation (la réponse) est visualisée par niveaux de rouge

[23] http://esm2.imt-mrs.fr/~dauce/these/node26.html

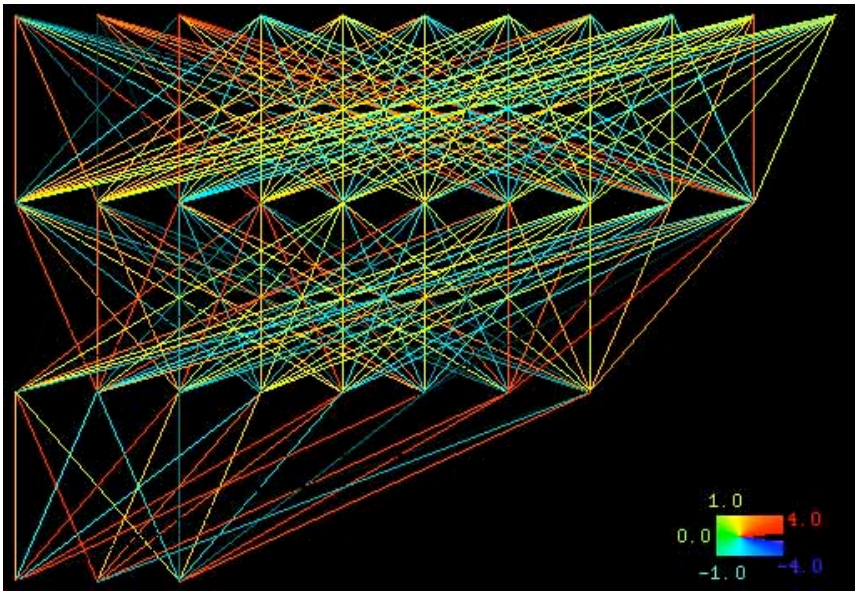

Fig. 4 : Visualisation d'une architecture de RNA qui s'appelle Multi Layered Perceptron (MLP) : les nœuds étant les neurones et les traits les synapses. Les couleurs chaudes indiquent une activation élevée. Ce RNA a 11 neurones en entrée et 3 neurones en sortie. Il a 2 couches cachées.

Lorsque l'on « met en marche » un RNA une valeur est assignée à chaque neurone d'entrée. Ces stimuli peuvent être donnés par un humain, provenir de capteurs, d'un autre logiciel, etc. Ces valeurs sont ensuite envoyées aux neurones voisins à travers les synapses en les multipliant par le poids de connexions. Une synapse avec un poids faible va donc diminuer l'activation d'un neurone, alors qu'une synapse avec un poids fort l'augmentera. La valeur reçue par chaque neurone dans la deuxième couche est la somme des valeurs propagées par les synapses menant à ce neurone. L'activation du neurone est ensuite modélisée par une fonction de transfert appliquée à la somme des entrées (en général, une fonction non linéaire de type sigmoïde ou logistique). Ainsi un stimulus se propage dans le RNA jusqu'à ce qu'il arrive à la couche de sortie et soit transformé en une réponse.

Par exemple un RNA de type Multi Layered Perceptron (MLP) (figure 5) est capable d'apprendre et de résoudre la table logique de la fonction logique « ou exclusif » (XOR) :

ENTRÉE 1	**ENTRÉE 2**	**RÉPONSE**
FAUX (0)	FAUX (0)	FAUX (0)
FAUX (0)	VRAI (1)	VRAI (1)
VRAI (1)	FAUX (0)	VRAI (1)
VRAI (1)	VRAI (1)	FAUX (0)

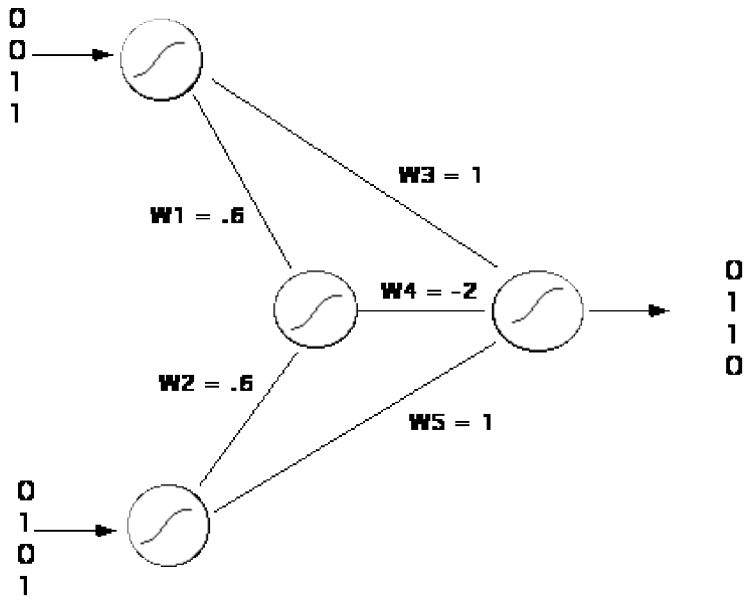

Fig. 5 : Un RNA qui a 2 neurones à l'entrée, 1 neurone dans la couche cachée et 1 neurone à la sortie. Les poids (weights) sont données (Wn = x). La fonction de transfert est sigmoïde.

Ce MLP montre qu'un RNA est capable de résoudre un problème non trivial, démontrant ainsi que les limites théoriques qu'ont évoquées Minsky et Papert en 1969 (c.f. chap. 4.2 Systèmes experts) ne pouvaient pas être généralisées à toutes les architectures de RNA possibles. Il s'agit cependant de savoir comment conduire l'apprentissage du réseau ; pour le MLP il aura fallu attendre jusqu'à l'année 1986[24].

5.3 L'apprentissage

Dans le travail avec les RNA la première étape est toujours l'initialisation du réseau, c'est-à-dire l'initialisation des poids synaptiques. D'habitude cela se fait avec des petites valeurs aléatoires. Lors de la deuxième étape - l'apprentissage - les poids aléatoires vont converger de plus en plus vers des valeurs utiles pour la résolution d'un certain problème. On peut dire qu'au début un RNA ne sait rien et qu'au fur et à mesure d'un apprentissage, il acquiert des connaissances.

L'apprentissage est un trait fondamental des RNA et les distingue radicalement d'autres architectures informatiques. Seul l'apprentissage permet de transmettre des connaissances non formalisées, quelque chose qui n'est pas possible avec un algorithme par exemple.

Pour résoudre un problème avec un algorithme ou avec un système expert il est nécessaire de le formaliser complètement. Ceci requiert déjà de grandes connaissances de la part des humains qui vont développer ce système.

Ce n'est pas nécessaire avec des RNA. L'apprentissage – l'acquisition de connaissances – par le réseau lui-même, permet de transmettre des connaissances à l'aide d'exemples. On donne quelques photos d'une personne à apprendre au RNA et après il est capable de reconnaître cette personne même si elle a changé !

[24] http://www.ccs.fau.edu/~bressler/EDU/CogNeuro/Perceptrons_hbtnn.htm

Or par exemple, qu'est-ce qui fait qu'on reconnaît un portrait de Pierre Boulez ? Comment formalise-t-on cela ?

5.3.1 Apprentissage supervisé

L'apprentissage porte sur les poids synaptiques : il renforce les poids synaptiques dans les cas favorables, et les atténue dans les cas défavorables. Les nouveaux couplages modifient les configurations des poids synaptiques, et orientent la réponse du réseau. Celle-ci est ainsi contrainte par l'apprentissage[25].

On utilise l'apprentissage supervisé quand on connaît la réponse souhaitée à un certain stimulus. On présente avec chaque stimulus la réponse correspondante et le RNA compare la réponse qu'il donne avec celle qui est souhaitée. Le résultat de cette comparaison est un vecteur d'erreur dont une fraction est réinjectée dans les différentes couches du réseau pour adapter les poids, afin de donner la bonne réponse. On désigne la quantité de l'erreur qui est renvoyée dans le réseau par le « learning-rate » ou taux d'apprentissage.

Cette technique est appelée « backpropagation ». Elle est utilisée dans des RNA type Perceptron ou Multi Layered Perceptron (MLP).

5.3.2 Apprentissage non supervisé

Une autre stratégie d'apprentissage est l'apprentissage non supervisé. Ce type d'apprentissage est utilisé dans l'architecture de RNA SOM, comme je l'ai implémenté pour MaxMSPJitter[26].

Ce type d'apprentissage repose sur des mécanismes d'auto-organisation et peut aussi être considéré comme un apprentissage par compétition.

D'abord, on recherche le neurone gagnant, c'est-à-dire un neurone dont la réponse correspond le plus à un stimulus donné - on peut parler d'une sorte de résonance. Après cette désignation, on modifie les poids synaptiques de ce neurone gagnant en comparant son activation avec le stimulus et le rendant plus similaire à celui-ci par itérations successives.

Souvent, on modifie non seulement les poids synaptiques du neurone gagnant mais aussi ceux des neurones voisins. On désigne un rayon autour du gagnant à l'intérieur duquel tous les poids seront changés. En conséquence on peut observer l'activation de groupes de neurones plutôt que des neurones isolés. À chaque stimulus correspond, en fin d'apprentissage, une région entière, créant ainsi géographie fonctionnelle du réseau.

On applique souvent une correction moins élevée aux neurones voisins qu'au neurone gagnant. La fonction la plus utilisée pour décrire la réduction de la correction par rapport à l'accroissement de la distance entre neurone gagnant et les neurones voisins est la fonction du « chapeau mexicain ».

---

[25] http://esm2.imt-mrs.fr/~dauce/these/node45.html

[26] c.f. chap. 6 jit.robosom

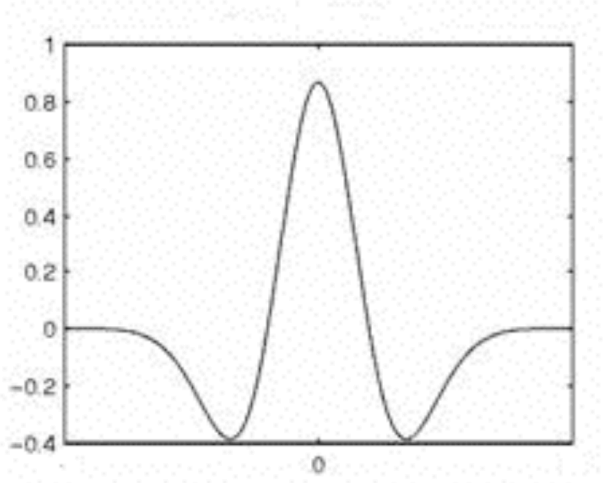

Fig. 6 : Fonction « chapeau mexicain ». Verticalement le taux d'apprentissage, horizontalement le placement « physique » des neurones (neurone gagnant au centre).

Souvent on commence l'apprentissage avec un rayon assez large pour laisser le SOM s'organiser d'une manière approximative. Au fur et à mesure de l'apprentissage, on diminue le rayon pour une adaptation du SOM de plus en plus précise.

5.3.3 Initialisation

Comme il a déjà été mentionné ci-dessus, l'initialisation des poids se fait normalement avec des petites valeurs aléatoires. Il y a cependant une autre technique qui consiste à donner une direction au RNA dès l'initialisation. Si on connaît déjà plus ou moins l'état final qu'on veut que le réseau atteigne, on peut initialiser les poids de manière à ce qu'ils favorisent cet état.

Pour une catégorisation[27] de couleurs avec un SOM[28] p.ex. on peut accélérer l'apprentissage en favorisant une certaine solution dès l'initialisation[29] :

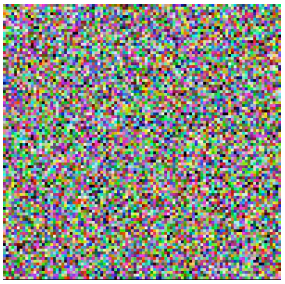

Fig. 7 : Initialisation aléatoire – on ne sait pas où et comment les couleurs seront classées

[27] c.f. chap. 6.2.1 Catégorisation
[28] c.f. chap. 6 jit.robosom
[29] http://davis.wpi.edu/~matt/courses/soms/

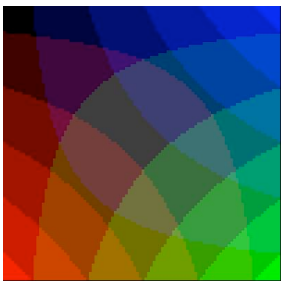

Fig. 8 : Initialisation guidée – on favorise un certain classement

5.3.4 L'importance de l'encodage

L'encodage des données est pour un RNA ce qui est pour nous la façon dont nous percevons notre environnement. C'est la perception du monde extérieur qui influence profondément les réponses d'un RNA comme elle influence nos actions aussi. Il suffit d'étudier les changements dans la peinture de Claude Monet au fur et à mesure de sa maladie aux yeux ou même des changements sociologiques après la publication de photos de la terre vue de l'espace ou de prisonniers en Irak. Cet effet ne se limite évidemment pas à la perception visuelle. Il peut arriver qu'un compositeur électroacoustique qui n'entend plus bien des fréquences aiguës, va faire un mixage où celles-ci sont trop présentes.

En travaillant sur l'encodage des données, on travaille directement sur la perception d'abord du RNA mais finalement aussi sur la nôtre. Quand on encode un objet musical par une FFT, on fait déjà implicitement un choix par rapport à la perception. On aurait peut-être pu encoder la même séquence comme une suite de hauteurs et de durées ou juste des amplitudes globales. Le choix est un choix compositionnel : qu'est-ce qu'on veut faire entendre ?

Quand on veut faire apprendre à reconnaître la voyelle « A » à un RNA et qu'on décide de présenter les données d'une FFT, il reste toujours la question de ce qu'on va présenter comme « A » ? Parlé, chanté, par différentes personnes ou toujours par la même, toujours la même hauteur, on ne présente que quelques fenêtres de la FFT ou la durée complète du « A »…

Pour chaque application, ces questions doivent à nouveau être posées. La réussite d'un apprentissage dépend presque toujours de la manière dont on présente les choses, un peu comme à l'école…

6 jit.robosom

jit.robosom est une encapsulation[30] MaxMSPJitter d'une carte auto-organisatrice ou self-organizing map (SOM) comme proposé par Teuvo Kohonen[31].

Les SOM sont utilisés pour visualiser et interpréter des données à grand nombre de dimensions, p.ex. des mesures climatiques quotidiennes. Chaque jour on enregistre les valeurs de centaines de paramètres, chaque paramètre correspondant à une dimension. Un SOM peut être utilisé pour interpréter les données de chaque jour et visualiser cette catégorisation sur une carte où les données sont organisées par similitude. Des journées climatiquement similaires proches les unes aux autres et inversement[32].

Un autre exemple de données à grand nombre de dimensions est un piano à 88 touches. Cela correspond à un espace à 88 dimensions où l'on désigne pour chaque dimension une vélocité. Un SOM est capable de faire une classification et une projection de cet espace dans un espace à deux dimensions[33].

J'avais programmé et publié sur Internet mon premier SOM qui n'utilisait que des objets Max (sans objets MSP) en août 2003. Puis en Novembre 2003 lors d'une conférence sur les RNA, j'ai décidé d'en programmer une nouvelle version, cette fois-ci en utilisant les objets Jitter. Les deux avantages principaux de Jitter par rapport à Max dans ce projet sont la vitesse et la taille des vecteurs.

Comme dans un SOM la plupart de calculs à effectuer sont matriciels, Jitter - qui est optimisé pour le traitement de la vidéo - peut effectuer ces calculs plus rapidement que les objets Max.

De plus la taille maximale d'une liste dans Max est de 256 éléments. Il est donc quasiment impossible d'envoyer des vecteurs à plus de 256 dimensions dans un SOM. Cette limite n'existe pas dans Jitter.

Même si d'autres implémentations de SOM existent, l'abstraction que j'ai faite est la première pour MaxMSPJitter. Pouvoir s'en servir à l'intérieur de MaxMSPJitter facilite beaucoup l'utilisation dans un contexte musical. Le fait que ce soit une abstraction permet aussi de faire des changements du « code » à la volée pour ainsi expérimenter de nouvelles architectures.

Un inconvénient de cette implémentation est le fait que MaxMSPJitter n'est pas adapté à ce type de problème. Certaines opérations de calcul matriciel n'étant pas optimisés dans MaxMSPJitter, jit.robosom est très lent, comparé a des implémentations en C p.ex.[34]

---

[30] en anglais : « abstraction ». Programme-objet autonome d'un environnement de programmation
[31] c.f. Annexes 7.4 SOM formule Kohonen et 7.5 jit.robosom pseudo code
[32] c.f. Annexes 7.7 Catégorisation de Typhons
[33] c.f. chap. 6.2.1 Catégorisation
[34] c.f. chap. 6.3 Développements Futurs

6.1 Documentation du patch

Une documentation détaillée des éléments utilisés dans jit.robosom est à l'intérieur du programme sous forme de commentaires.

Ici seront discutés seulement les éléments nécessaires à l'utilisation et au contrôle de la carte auto-organisatrice jit.robosom.

6.1.1 Arguments

1er argument : Le nom du SOM. Cette dénomination permet d'envoyer des messages à une instance de jit.robosom sans connecter la boîte de messages à l'objet jit.robosom lui-même. C'est aussi une méthode efficace d'organisation, quand on travaille avec plusieurs SOM en même temps.

2ème argument : « number of neurones » Définit la taille du SOM. Comme dans cette version de jit.robosom toutes les cartes sont carrées ce nombre doit être un nombre carré.

3ème argument : »number of dimensions » Définit le nombre de dimensions du SOM et ainsi la taille des stimuli et la taille des réponses.

6.1.2 Inlets et Outlets

jit.robosom a quatre inlets et quatre outlets :

1er inlet : « learn on / off » Permet de mettre en marche l'apprentissage ou de l'arrêter. Cela permet p.ex. de voir comment le SOM classe une certaine entrée, sans qu'il adapte ses poids synaptiques.

2ème inlet : « radius » (rayon) Dans cette implémentation de jit.robosom la rayon maximal est 1. pour une explication du rayon c.f. chap. 5.3.2 Apprentissage non supervisé.

3ème inlet : « input-vector » C'est ici qu'on envoie les stimuli en forme de matrices jitter pour le SOM. La taille des matrices doit correspondre au nombre de dimensions du SOM, désigné comme 3ème argument de jit.robosom.

4ème inlet : « learning-rate » Le taux d'apprentissage (c.f. chap. 5.3.1 Apprentissage supervisé)

1er outlet : « winning neuron » Les poids synaptiques du neurone gagnant en forme d'une matrice jitter qui logiquement a la même taille que les stimuli.

2ème outlet : « number of winning neuron » Le nombre attribué au neurone gagnant. Dans une carte de 3 sur 3 neurones p.ex., la numérotation est comme suit :

```
1 2 3
4 5 6
7 8 9
```

3ème outlet : « Error of winning neurone » Le taux d'erreur du neurone gagnant. Le taux d'erreur correspond à la distance euclidienne entre le stimulus et le neurone gagnant. La distance euclidienne se calcule ainsi :

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Un taux d'erreur faible indique que le neurone gagnant et le stimulus se ressemblent fortement.

4ème outlet : « Epoch » : un cycle complet d'apprentissage qui consiste en

- Une présentation d'un stimulus
- Une désignation du neurone gagnant
- Une modification des poids synaptiques

6.1.3 Messages de contrôle

« reset » : initialise les poids du SOM. À faire avant chaque apprentissage.

« temp $1 » : monte la température de la valeur spécifié par la variable $1. Valeur par défaut = 0 ; c.f. chap. 6.2.3.1 Températures et chap. 6.2.3.1.1 temp

« learntemp $1 » : monte « learntemp » de la valeur spécifiée par la variable $1. Valeur par défaut = 0 ; c.f. chap 6.2.3.1 Températures et chap. 6.2.3.1.2 learntemp

« init_range $1 » : Définit la borne supérieure des valeurs aléatoires qui sont utilisées pour l'initialisation des poids synaptiques du SOM. Valeur par défaut = 0.02 (la borne inférieur est toujours 0). Si les valeurs envoyées au SOM ne sont pas comprises entre 0. et 1. mais entre 0. et 127. p.ex. il peut être utile de monter init_range (jusqu'à 2. p.ex.)

« version » : imprime dans la fenêtre « Max » des informations sur la version du SOM utilisé, ainsi que les valeurs « init_range », « temp » et « learntemp ».

6.1.4 Matrices

Comme dans Jitter les matrices peuvent avoir des noms, il est possible de récupérer le contenu de certaines matrices en dehors de l'abstraction jit.robosom. Deux matrices en particulier sont utiles dans le travail avec les SOM.

« jit.matrix $1-küde » : $1 doit être remplacé par le nom du SOM (1er argument). Cette matrice contient tous les poids synaptiques du SOM. Cela peut être utile quand on veut visualiser toutes les données qui sont dans le SOM comme dans

l'exemple « 2 classification images », où dans le patcher « show_me_the_map_ !!! » tous les neurones sont visualisés[35].

« jit.matrix $1-error » : $1 doit être remplacé par le nom du SOM (1er argument). Cette matrice contient les erreurs (distances euclidiennes) pour chaque neurone. Elle est mise à jour une fois par cycle d'apprentissage. Un exemple d'application se trouve dans le patch « erroranalysis » qui est utilisé dans l'exemple « 4 reconnaissance spectrale »[36].

6.2 Applications

La liste des applications citées ci-dessous n'est pas exhaustive. La recherche de nouvelles applications est au centre de mon intérêt pour ce projet.

6.2.1 Catégorisation

La catégorisation des données à grand nombre de dimensions dans un espace à 2 dimensions est l'application classique des SOM. En fonction des données à catégoriser on peut aussi utiliser des espaces de représentation à 1-, 3- ou plus de dimensions.

Un exemple de cette application est donné dans le patch fourni avec jit.robosom « 3 classification d'images ». J'ai choisi ici des images à la place de sons pour le rendre plus démonstratif à l'écrit.

[35] c.f. chap. 6.2.1 Catégorisation
[36] c.f. chap. 6.2.2 Reconnaissance

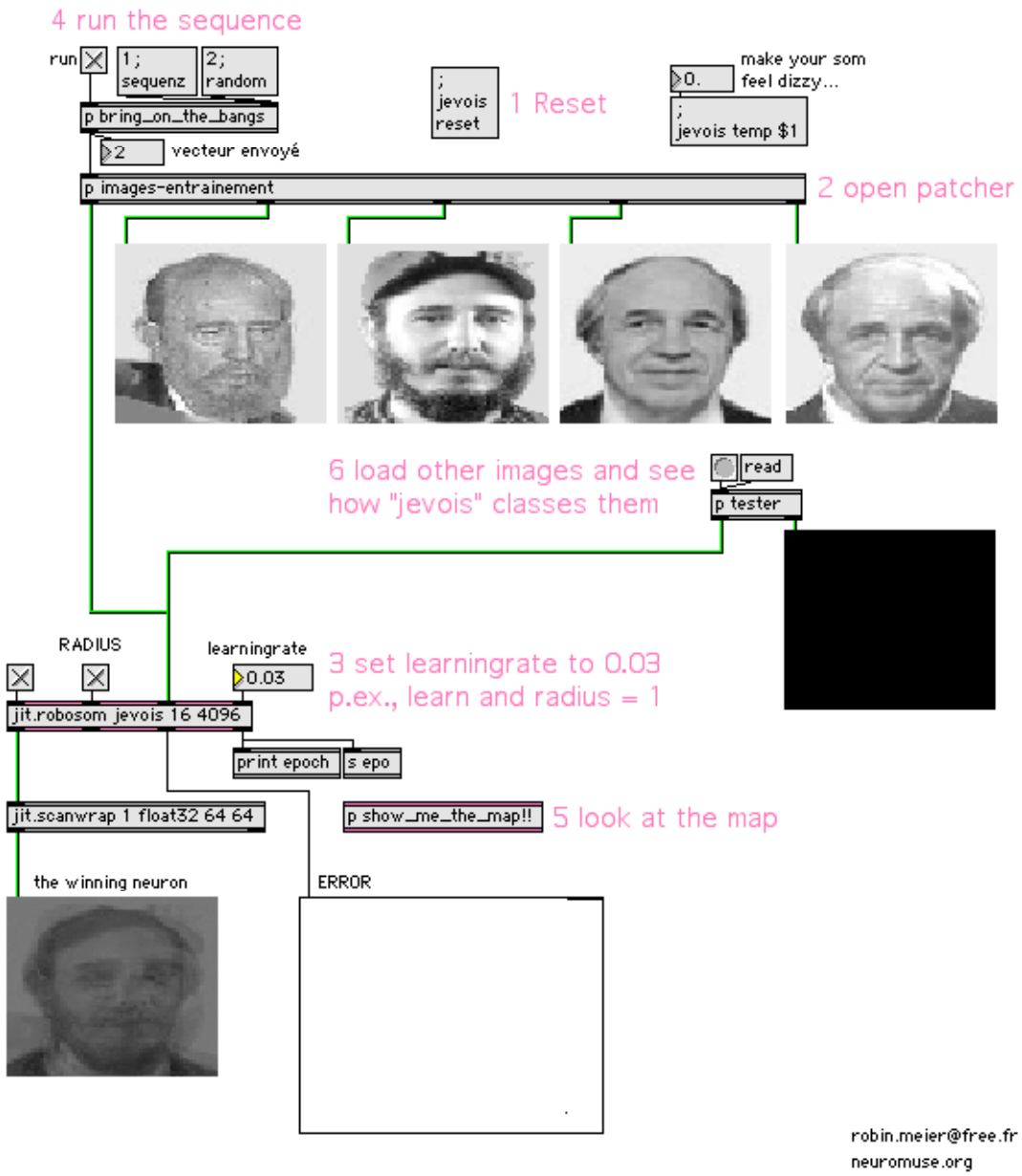

Fig. 9 : 3 classification images.pat

Dans cet exemple, quatre images différentes sont présentées à l'objet SOM nommé `jit.robosom jevois 16 4096`. Il les catégorise en 16 classes des vecteurs à 4096 éléments et visualise sa catégorisation dans le patcher `show_me_the_map !!`

Fig. 10 : Contenu du patcher « show_me_the_map_ !!! » : La visualisation de tous les neurones qui permet d'analyser la catégorisation du SOM.

On remarque dans figure 10 comme des images similaires sont placées proches les uns des autres vice-versa. Dans cet exemple, on a donné huit images à catégoriser. Pour bien distinguer ce nombre d'images un SOM de 16 neurones et presque trop petit.

Cette capacité des SOM à catégoriser des données peut être utile dans plusieurs applications musicales :

En faisant une catégorisation d'accords sur une dimension p.ex. on peut faire des enchaînements harmoniques où l'ordre des accords est déterminé par le SOM en fonction de la similarité. En faisant la catégorisation sur 2 dimensions, on peut après « se promener » librement sur la carte proposée par le SOM sans passer brutalement d'un accord à un autre qui ne ressemble pas au premier. Il se peut même que si la taille du SOM est plus grande que le nombre d'accords présentés, le SOM aille créer des accords intermédiaires entre deux accords qui n'existaient pas dans la collection d'accords présentés.

Il est possible de faire la même chose avec des spectres (Frédéric Voisin est en train, au moment où j'écris ce texte, de faire une catégorisation de sons multiphoniques), des gestes de danseurs, etc.

Il y a également la catégorisation de mélodies qui peut être utile du point de vue de la musicologie comme à l'université de Jyväskylä en Finlande où l'on utilise des SOM pour catégoriser une archive de env. 9000 mélodies populaires[37]. Bien sûr de telles recherches peuvent servir également d'aide à la composition.

Pour créer des gammes de toutes sortes – à mon avis un élément important de la composition - cet outil peut être très efficace.

6.2.2 Reconnaissance

La reconnaissance est étroitement liée à la catégorisation. Une fois qu'un SOM à catégorisé des données, il est aussi capable de les reconnaître. Il est même capable de reconnaître ou –plus proprement dit – de classer des données qu'il ne « connaissait » pas avant, qui ne lui n'ont pas été présentées lors de l'apprentissage.

On peut dire qu'un SOM (et c'est valable pour les humains aussi je pense) a reconnu une photo de Pierre Boulez qu'il n'a jamais vue, quand il la met dans la même catégorie qu'une photo de Pierre Boulez qu'il connaissait déjà.

Dans des cas où un SOM n'arrive pas à classer un stimulus avec certitude il peut donner des réponses approximatives. Cela revient à dire que le stimulus correspond à quelque chose entre deux neurones ou régions.

Un exemple de reconnaissance est fourni avec jit.robosom dans le patch `4 reconnaissance spectrale.` Ce patch permet de reconnaître des voyelles en temps réel.

[37] Toiviainen & Eerola, A Method for Comparative Analysis of Folk Music Based on Musical Feature Extraction and Neural Networks, http://www.jyu.fi/musica/essen/map.html

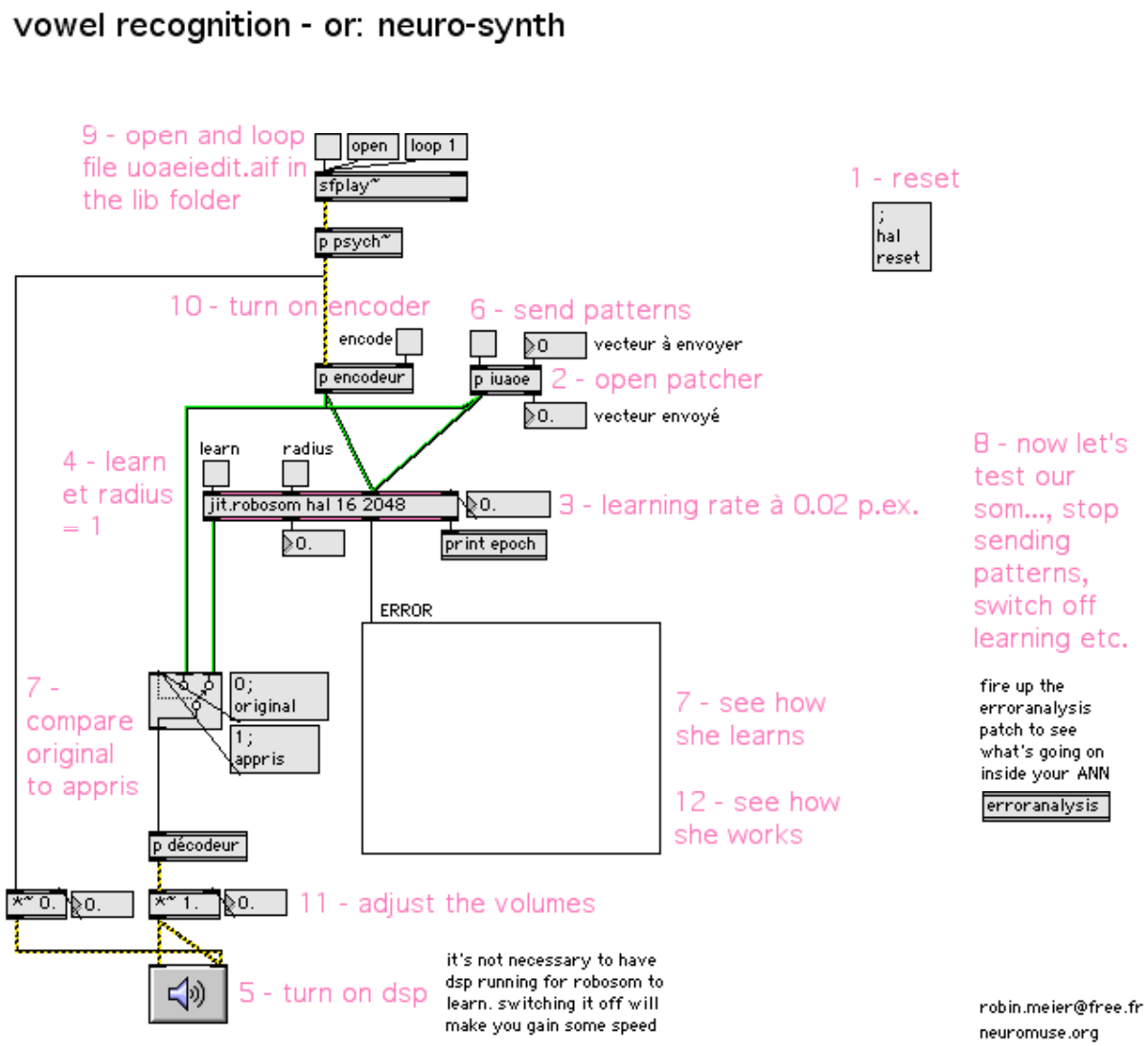

Fig. 11 : Le patch « 4 reconnaissance spectrale»

Premièrement on fait une classification de cinq voyelles, encodées en 4 fenêtres successives d'une FFT à 512 bandes. Cela donne des vecteurs à 2048 dimensions qu'on donne à « écouter » à un SOM de 16 neurones. Pour faciliter l'utilisation du patch, j'ai déjà fait un encodage des fichiers sons et l'ai sauvegardé en format Jitter jxf.

Lors de l'apprentissage, on peut déjà écouter la réponse du SOM. Comme, par correction récursive, les neurones s'adaptent de plus en plus aux stimuli les réponses du SOM vont ressembler de plus en plus aux stimuli. Ces réponses sont rendues audibles par une FFT inverse (patcher `décodeur`).

On peut également suivre visuellement l'évolution des poids synaptiques en ouvrant le patch `erroranalysis`. On voit alors comme les activations de neurones se déplacent au fur et à mesure de l'apprentissage, pour finalement donner à chaque voyelle sa place sur la carte (c.f. Annexes 7.3 Reconnaissance de Voyelles). Un film au format mp4 de l'évolution des poids synaptiques lors de l'apprentissage de quatre voyelles est disponible sur le site Internet du projet Neuromuse : http://www.neuromuse.org/txt/tutorial/Hal04.html .

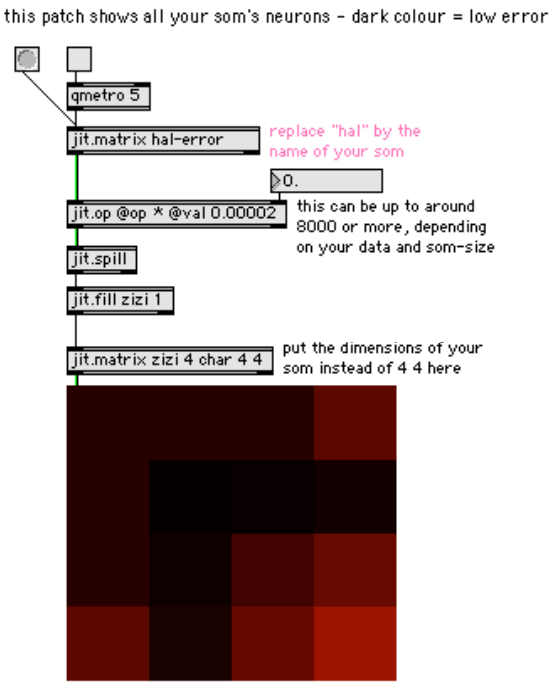

Fig. 12 : Visualisation des poids synaptiques lors d'un apprentissage dans le patch « erroranalysis ». Les couleurs sombres indiquant une erreur faible et donc un poids élevé.

Une fois qu'un taux d'erreur assez faible est atteint, on peut tester la reconnaissance du SOM en lui faisant reconnaître les voyelles dans l'enregistrement audio d'un chanteur.

La reconnaissance de voyelles est plus généralement une reconnaissance de spectres. Cela permet une multitude d'applications dont une serait éventuellement la reconnaissance d'accords. Il n'y a à ma connaissance actuellement pas de suiveur de hauteurs qui permette de reconnaître plus que trois hauteurs en même temps (et déjà trois hauteurs posent des problèmes). Il est certainement possible de reconnaître des accords plus complexes avec un SOM, si on fait apprendre ces accords au SOM auparavant. Mais je pense qu'il n'est pas forcément impossible de généraliser cette reconnaissance. On pourrait p.ex. utiliser 88 SOM et faire apprendre une note de piano à chacun. En analysant la réponse et le taux d'erreur de chaque SOM, on pourrait peut-être déterminer quelles notes ont été jouées.

Tout le problème est dans l'encodage et la conduite de l'apprentissage.

6.2.2.1 Suivi de partition

Comme la capacité des SOM en reconnaissance est assez impressionnante et fiable, c'est une technologie qui peut très bien être utilisée pour des suiveurs de partition.

Contrairement aux techniques de suivi de fréquences utilisées habituellement, qui montrent plus de difficulté dans le suivi des extrêmes graves que dans les aigues

par exemple (c.f. objet fiddle~[38] dans MaxMSP), la reconnaissance au moyen de SOM réagit avec la même acuité, quelque soit la fréquence, parce qu'elle ne repose pas sur des formalisations ou des généralisations.

Ainsi on peut imaginer qu'il est possible par l'apprentissage, de créer des suiveurs de partition spécialisés pour un certain musicien et une certaine partition. Ces suiveurs peuvent même évoluer lors de chaque concert et devenir de plus en plus efficaces.

6.2.2.2Reconnaissance à travers bruit

Un autre type de reconnaissance consiste à ne retenir d'une suite de données que les parties qui sont pertinentes. Cela est montré dans le patch « 1 reconn image bruitée ».

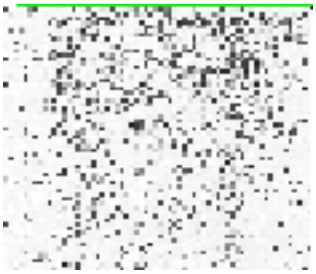

Fig. 13 : Image à laquelle du bruit à été ajouté

Comme les poids synaptiques d'un SOM ne s'adaptent que lentement si le taux d'apprentissage est faible (0.01 dans l'exemple) ils ne retiennent que les éléments pertinents ou, dans ce cas, statistiquement plus probables. L'image d'origine apparaît peu à peu à travers le bruit :

Fig. 14 : L'image comme elle a été retenue par le SOM (taux d'apprentissage 0.005)

[38] c.f. http://www-crca.ucsd.edu/~tapel/software.html

Fig. 15 : Image d'origine (Egon Schiele)

6.2.3 Génération

La fonction initiale d'un SOM est de reproduire des données qu'il a apprises et catégorisées. Comme on a vu plus haut, cela peut déjà servir à la composition. Mais il y a d'autres techniques qui consistent à modifier les poids synaptiques eux-mêmes et créer ainsi des nouveaux matériaux à partir de connaissances apprises.

Cela peut se faire en temps réel et est donc particulièrement apte à être utilisé au concert.

On pourrait considérer cette manière de générer du matériau musical comme une extension des techniques de composition stochastique. Mais, malgré toute ressemblance, il y a quelques différences importantes, dont l'illustration la plus poignante est que les RNA permettent l'apprentissage et donc une transmission de connaissances non formalisées.

D'ailleurs il ne faut pas oublier que l'encodage et le décodage influencent profondément la classification et la génération du contenu[39]. L'encodage d'une séquence musicale avec une FFT va entraîner des résultats complètement différents d'un encodage en hauteur, intensité et durée (p.ex. MIDI).

6.2.3.1 Températures

La notion de température dans le contexte des RNA vient d'une technique nommée « simulated annealing » ou « recuit simulé ». Utilisée dans certains types d'algorithmes cette technique s'est inspirée de la métallurgie où l'on réchauffe et refroidit du métal pour éviter des défauts. De la même manière le « recuit simulé » peut, au niveau des RNA favoriser l'apprentissage. Dans ce domaine-là, le « recuit simulé » consiste à introduire une certaine quantité de bruit dans les connexions synaptiques – on augmente la température.

Dans `jit.robosom` j'ai, pour le moment,implémenté deux types de températures : `temp` et `learntemp`.

Il est fort probable que j'implémente d'autres types de températures dans des développements futurs. Je pense que c'est un moyen formidable pour introduire des comportements dynamiques dans les RNA. Il existe déjà plusieurs thèses qui vont

[39] c.f. chap. 5.3.3 L'importance de l'encodage

dans cette direction[40]. En généralisant ces « accès au chaos » on peut imaginer des nouvelles architectures de RNA dynamiques et plus autonomes. Ce sujet fait partie des recherches du projet Neuromuse.

6.2.3.1.1 Le paramètre « temp »

`temp` permet d'introduire une certaine quantité de bruit dans les connexions synaptiques (la matrice des poids). Ceci est réalisé avec une simple addition de valeurs aléatoires aux poids synaptiques. La valeur `r` pour `temp` désigne les bornes des valeurs aléatoires qui sont `-r` et `r` [41].

Normalement utilisée pendant la phase de l'apprentissage cette température permet d'éviter des minima locaux. Il s'agit de minimiser l'erreur entre ce qui est présenté et ce qui est reproduit. Un SOM peut « croire » avoir trouvé le minimum d'erreur, mais se trouver en fait dans un minimum local et être ainsi pris dans une sorte de piège (c.f. schéma). Pour faire ressortir le SOM et lui faire trouver le minimum global on augmente la température afin d'ajouter du bruit et puis on la baisse lentement, ce qui va permettre au SOM de réorganiser les poids synaptiques et d'atteindre un minimum global.

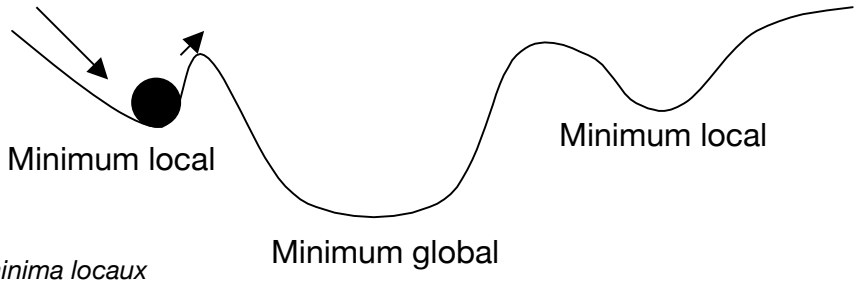

Fig. 16 : minima locaux

On peut aussi se servir du paramètre `temp` pour générer du matériau musical. On peut faire apprendre p.ex. un accord et en élevant la température le SOM va de plus en plus oublier ce qu'il a appris pour jouer des accords de plus en plus éloignés (par rapport au calcul de distance du SOM – distance euclidienne[42]) pour finir dans un état chaotique non prévisible[43].

6.2.3.1.2 Le paramètre « learntemp »

`learntemp` fonctionne différemment : ici ce sont les distances entre les neurones et l'entrée présentée qui sont multipliées par des valeurs aléatoires comprises entre

---

[40] Christophe Philemotte, « Etude des réseaux neuronaux en tant que systèmes dynamiques chaotiques », http://iridia.ulb.ac.be/~cphilemo/tfe/tfe.html
et Emmanuel Daucé, « Adaptation dynamique et apprentissage dans de réseaux de neurones récurrents aléatoires », http://esm2.imt-mrs.fr/~dauce/these/
[41] c.f. sous-patch « incorporate_into_küde » dans jit.robosom
[42] c.f. chap. 6.1.1 Inlets et Outlets
[43] c.f. Annexes 7.1 Interpolations d'accords

`1 - r'` et `1 + r'` [44].

Par conséquent les neurones n'apprennent pas tous à la même vitesse. Dans une image par exemple, il y peut avoir des régions qui s'adaptent plus rapidement et d'autres qui sont plus lentes.

J'ai introduit ce type de température, pour éviter des courbes d'apprentissages trop linéaires. Musicalement il me paraît souvent plus intéressant d'avoir des parcours non linéaires et donc plus surprenants tout en gardant une direction générale préétablie.

Dans le cas d'interpolations[45], `learntemp` permet de ne pas passer linéairement de A à B mais par détours et fournit ainsi des résultats plus variés et plus riches.

Une autre spécificité que j'ai remarquée en faisant des essais avec `learntemp`, c'est qu'en élevant à la fois `temp` et `learntemp` le taux d'erreur va varier autour d'une certaine valeur pendant un temps indéfini et, à un moment non-prévisible va « exploser » et atteindre pendant quelques dizaines de cycles d'apprentissages à des valeurs d'erreurs qui peuvent être des multiples de 10'000 ou plus de la valeur stable précédente.

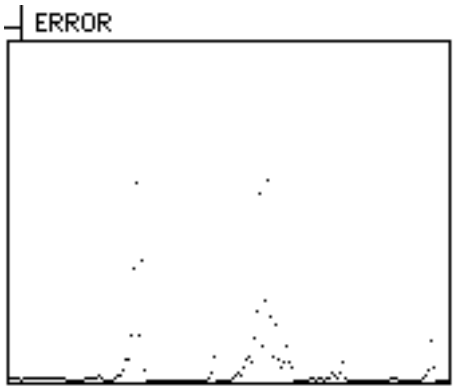

Fig. 17 : Plusieurs pics ou « explosions » du taux d'erreur
Learnrate = 0.02, temp = 4, learntemp = 100

Musicalement ces pics peuvent être intéressants, car ils entraînent des changements soudains et extrêmes dans le matériau généré.

6.2.3.2Interpolation

Le fait que les SOM apprennent par auto adaptation peut être exploité pour faire des interpolations.

L'apprentissage en lui-même est déjà une sorte d'interpolation entre un état chaotique / aléatoire et un état ordonné[46]. Ce mécanisme peut aussi être mis en

[44] c.f. sous-patch « gewichtsanpasser2 » dans jit.robosom
[45] c.f. chap. 6.2.3.2 Interpolation
[46] c.f. Annexes 7.1 Interpolations d'accords

œuvre pour interpoler entre différents états ordonnés. Cela est démontré dans le patch `2 chord morph`, qui permet de faire des interpolations entre des accords.

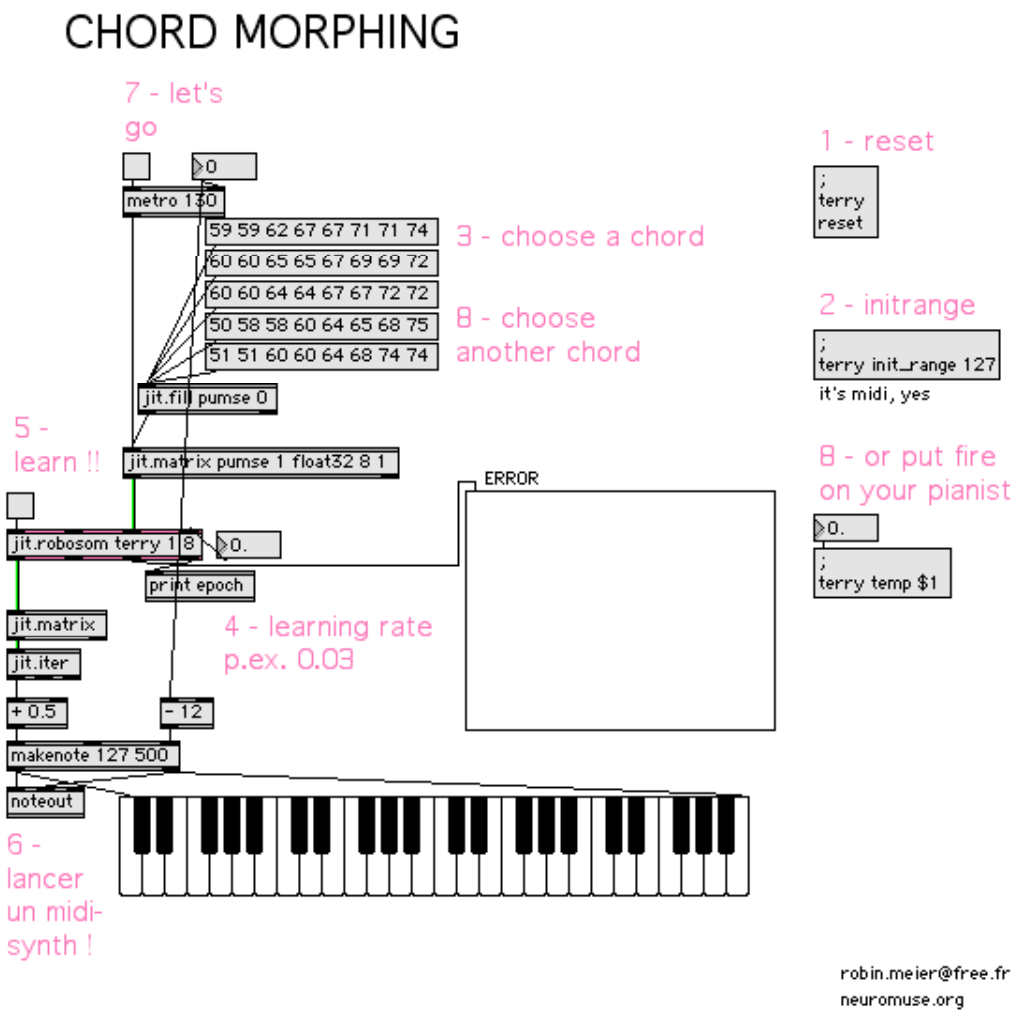

Fig. 18 : Patch « 2 chord morph »

En utilisant un SOM constitué d'un seul neurone, on peut d'abord faire apprendre un premier accord A (interpolation aléatoire vers accord A). Puis, à partir de cet accord A on peut faire apprendre un deuxième accord B (interpolation d'accord A vers accord B).

Normalement ces interpolations sont plutôt logarithmiques, du fait de la règle d'apprentissage qui consiste à adapter les neurones au stimulus en additionnant une partie de l'erreur (la distance entre stimulus et réponse).

Pour varier la manière dont le SOM interpole, on peut se servir des deux températures. Dans ce cadre `learntemp` se révèle d'être particulièrement efficace. Comme learntemp multiplie l'erreur par des valeurs aléatoires, il peut – dans l'exemple des accords – avoir des notes qui arrivent plus vite sur la note cible et d'autres qui mettent plus de temps. Il peut même avoir des notes qui dépassent la note cible et y reviennent.

L'interpolation est une technique très riche pour la musique. Le concept de continuité qui peut être trouvé dans beaucoup d'œuvres de la musique contemporaine en est témoin. La possibilité de donner quelques points fixes, quelques bornes et ensuite explorer les espaces entre ces points est une expérience intéressante. On peut imaginer interpoler avec facilité des spectres, des presets pour des systèmes de synthèse (ou autres) très complexes, des rythmes, des séquences mélodiques, etc.

6.2.3.3Réinjection

Une autre technique que j'ai trouvé pour générer du matériau musical consiste à réinjecter la sortie d'un SOM dans sa propre entrée. C'est une technique que j'ai développée dans ma pièce électronique présentée pour cet examen du diplôme d'études musicales du 4 juin 2004.

On fait apprendre p.ex. une petite séquence mélodique dont chaque note est encodée en trois dimensions :

hauteur (0-127), vélocité (0-127), Δ_t (0 – n)

où Δ_t est le temps entre 2 notes successives. Il est éventuellement utile de le multiplier par une constante pour être plus ou moins dans le même domaine de valeurs comme la hauteur et la vélocité.

Le principe est le suivant : la réponse d'un SOM à un neurone qui traite des données à trois dimensions passe par une ligne de retard de n cycles d'apprentissages. Après cette ligne de retard, la réponse du SOM est réinjectée dans lui-même.

Pour créer une boucle stable on met le taux d'apprentissage à 1.0 (100%). Si on veut que la séquence se développe, on change le taux d'apprentissage : le réseau n'arrive plus à reproduire exactement ce qui lui est présenté et produit alors une variation / mutation de la séquence initiale.

On peut aussi augmenter légèrement la température pour créer des variations qui se propageront et se développeront dans le temps à cause da la structure cyclique du principe de départ.

Une autre technique consiste à additionner à l'entrée réinjectée des nouveaux vecteurs. En montant la quantité du nouveau vecteur on peut diriger le développement autonome du SOM vers un nouvel état donné, créant ainsi un mélange entre interpolation et développement autonome.

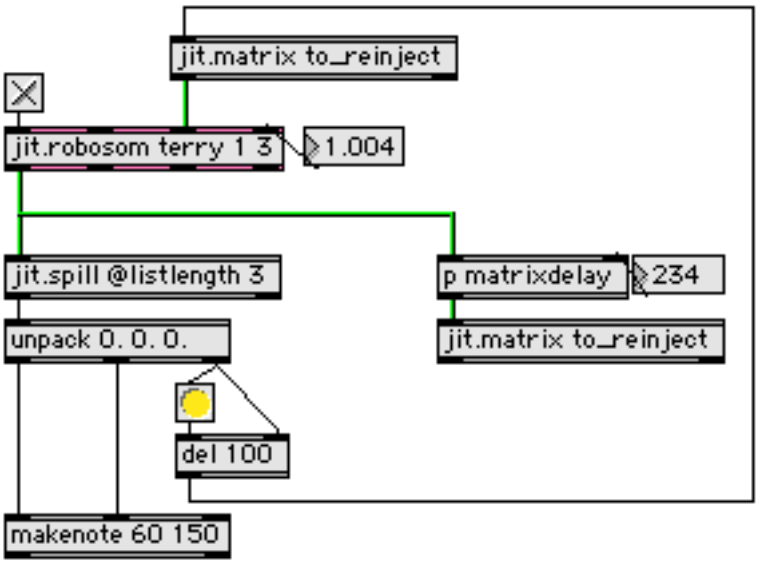

Fig. 19 :Photo d'écran d'un SOM à réinjection

On voit dans le patch ci-dessus comme la sortie de `Terry` est réinjecté après un délai de 234 cycles d'apprentissage. Chaque cycle est déclenché Δ_t millisecondes après le précédent (implémenté par le `del 100` après `unpack 0. 0. 0.`)

6.3 Développements futurs et conclusion

Parmi les priorités du développement figure une généralisation du rayon d'apprentissage dans jit.robosom. Actuellement, seul un rayon de 1 ou 0 est possible, ce qui rend difficile le travail avec des SOM d'une taille supérieure à 81 neurones. Ce problème devrait être résolu cet été.

Au moment où j'écrivais ces lignes Frédéric Voisin venait de programmer une première version d'un SOM en LISP. L'architecture du LISP, qui est très différente de l'architecture MaxMSPJitter, permettra de faire certains types d'expériences avec plus de facilité que dans MaxMSPJitter. En particulier les processus récursifs sont beaucoup plus faciles à réaliser en LISP. De plus je suppose que le SOM en LISP sera plus rapide que jit.robosom. Ce qui restera à faire par contre, est de créer un accès aussi simple que pour jit.robosom. Dans un environnement audio / midi / vidéo comme MaxMSPJitter il est très facile de travailler avec des données sous forme musicale ce qui n'est pas forcément le cas pour le LISP.

Il s'agit aussi de porter en code C jit.robosom. Le C est un langage de programmation qui doit être compilé. C'est-à-dire que le code doit être traduit par un programme en langage-machine. Une fois cette traduction faite, le code n'est plus éditable. Cela gèlera en quelque sorte l'état de développement actuel de jit.robosom. Le grand avantage par contre de cette traduction en langage-machine, est que le programme peut être effectué jusqu'à cent fois plus vite. Le code C peut être utilisé pour faire des objets MaxMSPJitter, des objets PureData, des applications « stand-alone », etc.

Un autre projet de recherche et d'application est la construction d'un électro-encéphalogramme (EEG)[47]. Un EEG permet de mesurer l'activité électrique du cerveau en placent des électrodes sur le scalp. Il est ainsi possible de mesurer l'activité des différentes aires cérébrales.

Un des buts de la construction du EEG est l'utilisation comme capteur pour la création musicale. Pierre Henry et Roger Lafosse ont déjà commencé à explorer ce domaine avec « cortical art ». Ce qui est nouveau dans l'approche que j'ai choisie, c'est que l'analyse des données, obtenues avec le EEG sera faite avec des RNA. Miguel Nicolelis de la faculté de neurobiologie à l'université Duke (Durham, USA) a réussi à faire contrôler un bras robotique par un singe en mesurant l'activité cérébrale de ce dernier[48]. On a d'abord mesuré et donné à apprendre à des RNA l'activité cérébrale lorsque le singe a saisi des objets placés devant lui. Une fois cet entraînement des RNA fait, le bras robotique effectuait les mêmes mouvements en même temps que le singe. Encore plus surprenant : au bout d'un moment le singe a compris qu'il pouvait contrôler le bras robotique comme il pouvait contrôler son propre bras. Autrement dit, le singe s'est emparé du bras robotique comme si c'était une partie de son corps. Ainsi le singe a réussi à contrôler le bras robotique sans bouger son propre corps.

Il serait étonnant d'en arriver à de tels résultats avec un EEG construit par nous-mêmes. Mais je suis certain qu'on pourra à l'aide des RNA arriver à contrôler effectivement des systèmes musicaux et ouvrir ainsi des nouveaux accès à la création musicale. Même si la qualité de mesure et le nombre d'électrodes du EEG que nous allons construire n'est pas à la hauteur d'un EEG professionnel, ce projet peut très bien nous permettre des collaborations avec des laboratoires spécialisés.

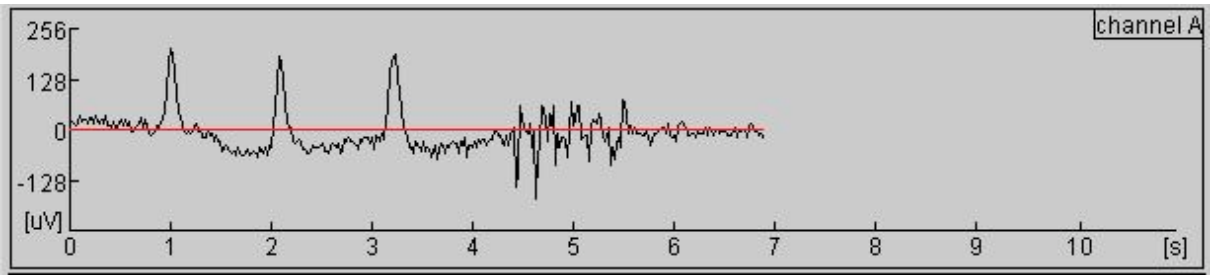

Fig. 20 : courbe de tension électrique d'un sujet clignant 3 fois les yeux et ensuite grinçant les dents

Ainsi, de nombreux sujets au sein des RNA ont pu être explorés dans le cadre de ce mémoire. Je pense qu'un lien étroit entre la recherche connexionniste sur les SOM et la composition musicale a pu être établi. J'espère que cette recherche va continuer à enrichir mon langage musical et va enfin ouvrir de nouveaux horizons non seulement pour moi mais aussi pour tous ceux qui sont intéressés par ce sujet.

[47] http://openeeg.sourceforge.net/
[48] http://dukemednews.org/news/article.php?id=7100&index=2

7 Annexes

7.1 Interpolations d'accords

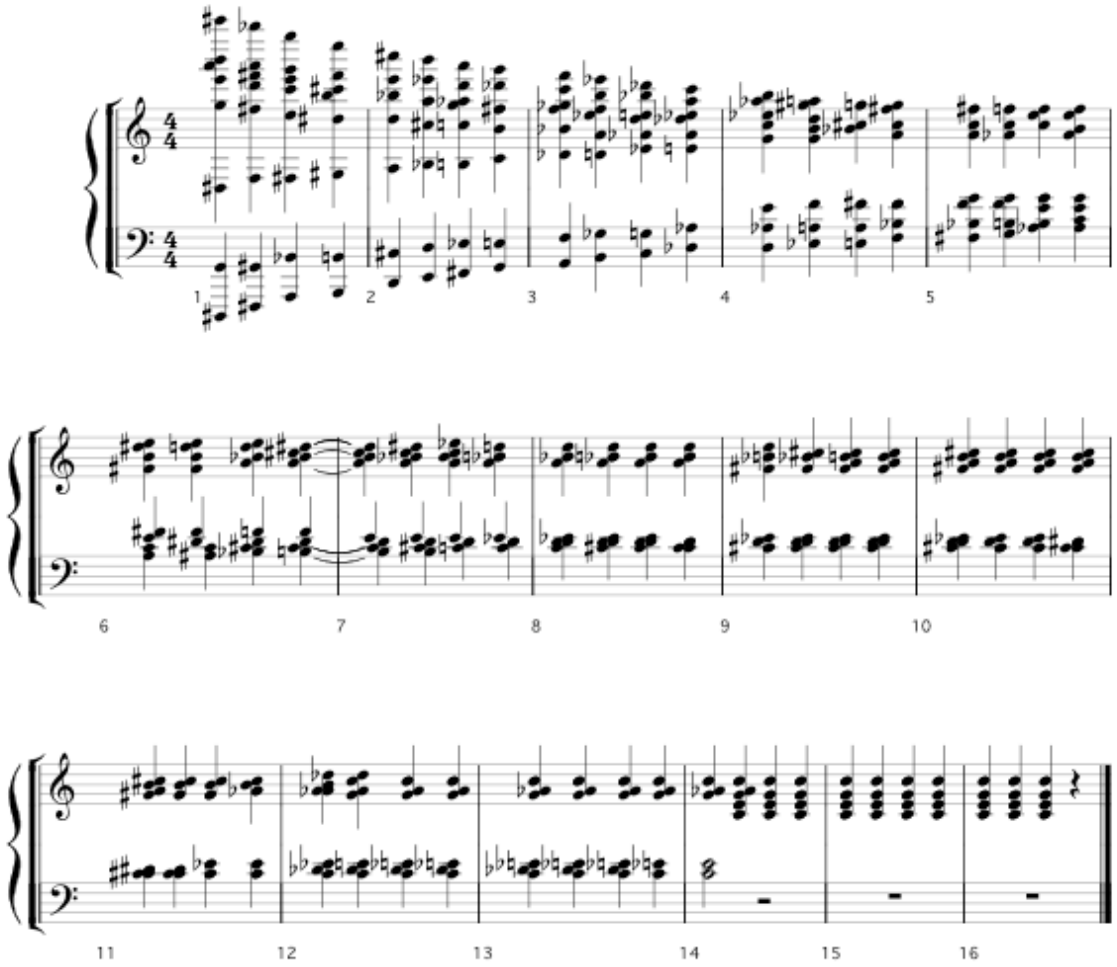

Fig. 21 : Interpolation entre état chaotique et un accord de Do Majeur. Temp et learntemp = 0 . ; taux d'apprentissage = 0.03

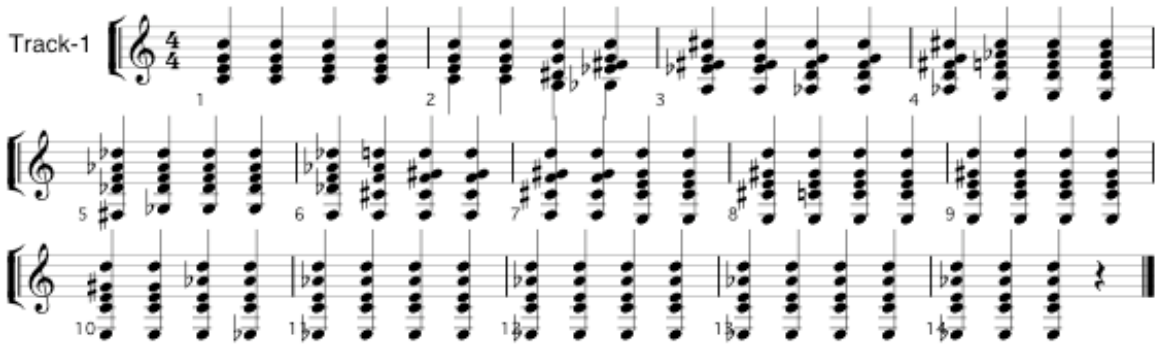

Fig. 22 :Interpolation entre un accord de Do Majeur et un accord non classé. Temp et learntemp = 0. ; taux d'apprentissage = 0.03

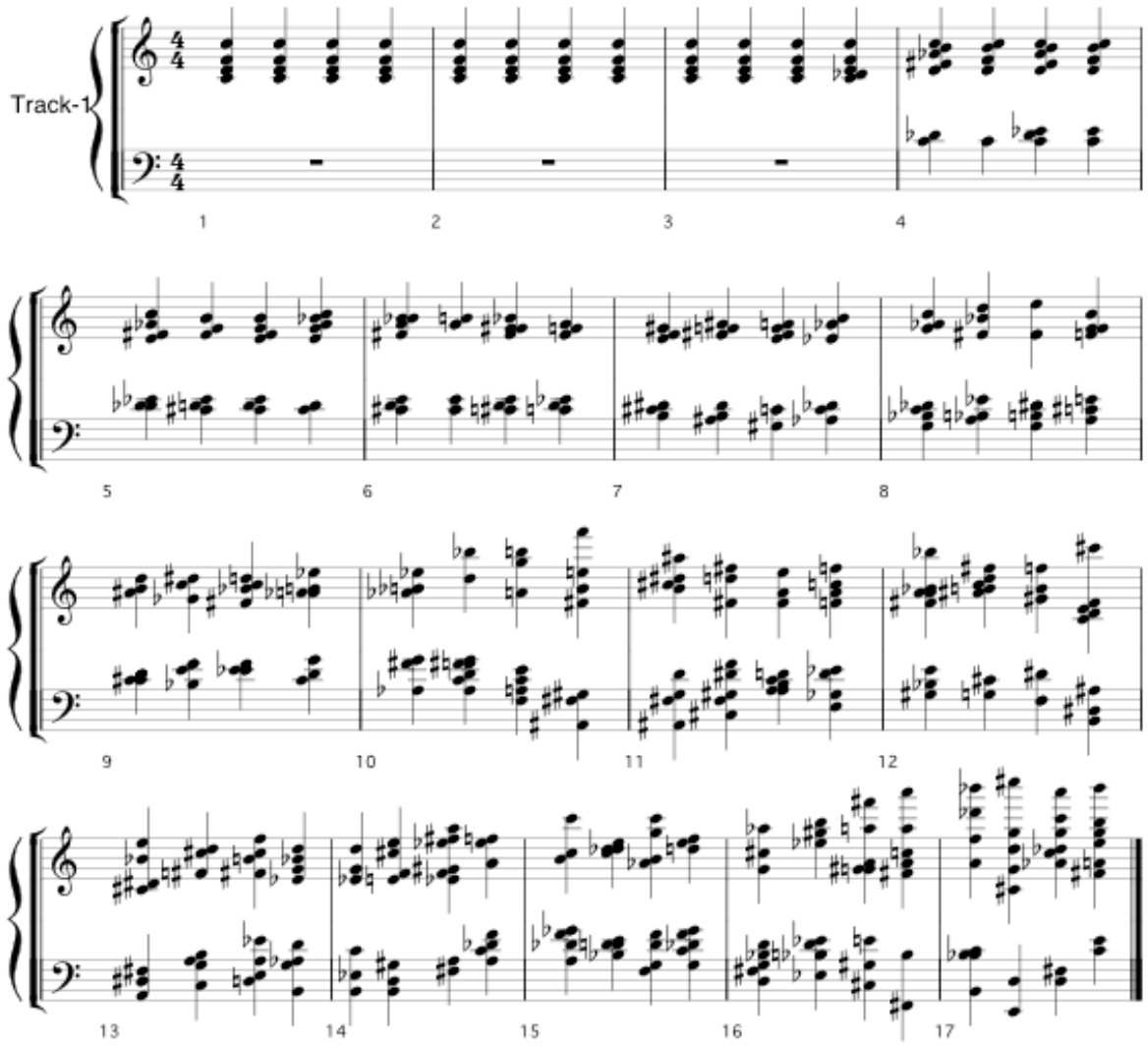

Fig. 23 : Interpolation entre un accord de Do Majeur et un état chaotique. Augmentation de temp de 0. à 4.

7.2 Reconnaissance de voyelles

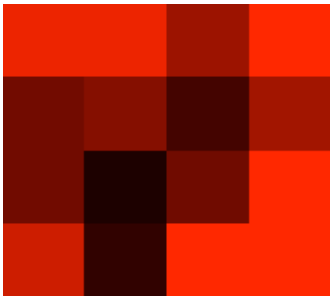

Fig. 24 : Activation du SOM lors de la voyelle « U »
Le centre de l'activation se trouve en bas à gauche

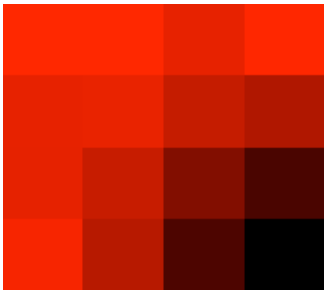

Fig. 25 : Activation du SOM lors de la voyelle « O »
Le centre de l'activation se trouve en bas à droite

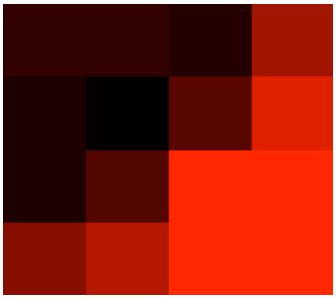

Fig. 26 : Activation du SOM lors de la voyelle « I »
Le centre de l'activation se trouve en haut à gauche

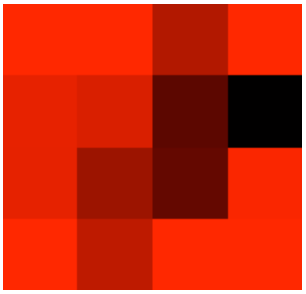

Fig. 27 : Activation du SOM lors de la voyelle »E »
Le centre de l'activation se trouve en haut à droite

Fig. 28 : Activation du SOM lors de la voyelle « A »
Le centre se trouve près du « U » mais n'est pas très développé

7.3 SOM formule Kohonen

D'abord le neurone gagnant est désigné.

t = 1,2,... égale le cycle d'apprentissage. Pour chaque stimulus x(t), le neurone gagnant « c » est désigné par la condition

$$\forall i, ||\mathbf{x}(t) - \mathbf{m}_c(t)|| \leq ||\mathbf{x}(t) - \mathbf{m}_i(t)||.$$

Après, les poids synaptiques du neurone gagnant c = c(x) ou des neurones à l'intérieur du rayon sont modifiés :

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(\mathbf{x}),i}(\mathbf{x}(t) - \mathbf{m}_i(t)).$$

$h_{c(\mathbf{x}),i}$ est la fonction du « voisinage », le rayon. C'est une fonction décroissante de la distance entre neurone i et neurone c sur la carte[49].

[49] http://www.cis.hut.fi/research/som-research/som.shtml; traduction : Robin Meier

7.4 jit.robosom pseudo code

w = poids (weights)

```
PROCEDURE Kohonen-apprentissage
      initalisation de tous les poids
      REPEAT
            choix arbitraire d'un stimulus m
            déterminer neuron-gagnant z pour stimulus m
            FOR tout les neurons j DO
                  IF dist(j, z) <= r THEN
                        h(jz) := e( - ( pow (dist(j,z) ) ) / ( pow(2r) ) )
                        w(ij) := w(ij) + n * h(jz) * ( m(i) - w(ij) )
                  END IF
            END FOR
            diminuer le taux d'apprentissage n et le rayon r
      UNTIL n = 0 OR r = 0 OR nombre d'itérations atteint
END Kohonen-apprentissage
```

La fonction h(jz) (rayon) n'a pas pu être implémentée correctement encore en MaxMSPJitter, mais elle est utilisée dans l'implémentation du SOM en LISP et sera implémentée également en C.

7.5 jit.robosom code source

Code source de jit.robosom version du 23.04.04 pour MaxMSPJitter v.4.3.2 pour MacOS10.3 :

```
max v2;
#N vpatcher 1 44 1025 654;
#P origin 53 1;
#P window setfont "Sans Serif" 9.;
#P newex 836 93 80 196617 s $1-learntemp;
#P message 539 415 50 196617 dim \$1;
#N vpatcher 222 362 846 567;
#P window setfont "Sans Serif" 9.;
#P newex 149 52 80 196617 r $1-learntemp;
#P newex 124 32 55 196617 r $1-temp;
#P newex 99 72 60 196617 pak 0. 0. 0.;
#P message 44 73 54 196617 0.02 0. 0.;
#P newex 44 33 45 196617 loadbang;
#P inlet 99 32 15 0;
#P newex 99 141 45 196617 print $1;
#P message 99 97 314 196617 Version du 23.05.04 - robin meier
- neuromuse.org \, init-range \$1 \, temperature \$2 \,
learntemp \$3 \, ------------------------------;
#P connect 3 0 4 0;
#P connect 2 0 5 0;
#P connect 4 0 0 0;
#P connect 5 0 0 0;
#P connect 0 0 1 0;
#P connect 6 0 5 1;
#P connect 7 0 5 2;
#P pop;
#P newobj 274 113 62 196617 p print_info;
#P comment 674 550 294 196617 avec l'aide précieuse de
frédéric voisin - merci beaucoup fred !;
#P newex 673 93 55 196617 s $1-temp;
#N comlet winning neuron;
#P outlet 162 216 15 0;
#P newex 513 237 30 196617 s $1;
#P newex 60 38 164 196617 r $1;
#P newex 327 370 90 196617 route dims kats;
#P newex 591 186 65 196617 prepend kats;
#P newex 513 186 68 196617 prepend dims;
#P newex 432 417 27 196617 t b f;
#N vpatcher 69 338 469 638;
#P inlet 168 101 15 0;
#P inlet 146 101 15 0;
#P inlet 129 101 15 0;
#P outlet 129 179 15 0;
#P window setfont "Sans Serif" 9.;
```

```
#P newex 129 125 27 196617 gate;
#P message 168 125 32 196617 set 0;
#N counter;
#X flags 0 0;
#P newobj 129 154 66 196617 counter;
#P comment 201 148 83 196617 welches jahr haben wir heute?;
#P connect 5 0 3 0;
#P connect 3 0 1 0;
#P fasten 2 0 1 0 173 147 134 147;
#P connect 1 0 4 0;
#P connect 6 0 3 1;
#P connect 7 0 2 0;
#P pop;
#P newobj 256 260 82 196617 p epoch_counter;
#P newex 21 67 988 196617 route int version init_range reset
temp learntemp;
#N comlet radius (1 / 0);
#P inlet 70 251 15 0;
#P newex 99 291 27 196617 gate;
#N comlet epoch;
#P outlet 256 283 15 0;
#P newex 82 220 78 196617 t i i b;
#N vpatcher 229 58 940 700;
#P origin 0 -58;
#P window setfont "Sans Serif" 9.;
#P comment 535 481 100 196617 ça va venir \, bientôt! chapeau
mexicain et tout \, ouais...;
#P newex 500 29 30 196617 r $1;
#P newex 38 577 27 196617 - 1;
#P newex 81 92 27 196617 + 1;
#P newex 143 62 27 196617 0.02;
#P newex 241 152 27 196617 +;
#P newex 420 134 27 196617 - 1;
#P newex 298 409 27 196617 + 3;
#P newex 327 409 27 196617 + 3;
#P newex 108 448 45 196617 split 0 8;
#P newex 122 402 92 196617 expr $i1 + $i2 - 1;
#P newex 108 384 93 196617 expr $i1 + $i2 + 1;
#P newex 271 304 27 196617 - 3;
#P newex 300 304 27 196617 - 3;
#P newex 81 343 45 196617 split 0 8;
#P newex 95 297 91 196617 expr $i1 - $i2 - 1;
#P newex 81 279 92 196617 expr $i1 - $i2 + 1;
#P newex 279 194 27 196617 + 1;
#P newex 279 174 27 196617 - 3;
#P newex 239 194 27 196617 * 1;
#P newex 239 174 27 196617 /;
#P newex 66 237 45 196617 split 0 8;
#P newex 409 87 27 196617 sqrt;
#P newex 500 62 142 196617 route kats;
#P outlet 165 106 15 0;
#P newex 165 83 27 196617 0.01;
```

```
#P newex 182 59 34 196617 * 0.5;
#P inlet 182 38 15 0;
#P newex 81 128 40 196617 t b i b;
#P newex 38 552 472 196617 split 0 8;
#P outlet 38 601 15 0;
#P newex 80 212 27 196617 - 1;
#P newex 66 194 27 196617 + 1;
#P newex 52 176 27 196617 - 1;
#P inlet 81 41 15 0;
#P newex 38 158 27 196617 + 1;
#P comment 398 251 95 196617 achtung: kats 1-16;
#P comment 220 57 118 196617 learning-rate * 0.5 = learning-
rate for radius;
#P comment 535 419 100 196617 i know \, it's messy. the day
i'm lucid i'll find some smarter way of doing the radius
stuff.;
#P connect 10 1 3 0;
#P connect 17 0 9 0;
#P connect 5 0 9 0;
#P connect 3 0 9 0;
#P connect 24 0 9 0;
#P connect 29 0 9 0;
#P connect 9 0 36 0;
#P connect 36 0 8 0;
#P connect 10 1 5 0;
#P connect 16 0 3 1;
#P connect 10 1 6 0;
#P connect 7 0 17 0;
#P connect 6 0 17 0;
#P connect 16 0 5 1;
#P connect 10 1 7 0;
#P connect 4 0 35 0;
#P connect 35 0 10 0;
#P connect 10 1 22 0;
#P connect 23 0 24 0;
#P connect 22 0 24 0;
#P connect 21 0 17 1;
#P connect 10 1 23 0;
#P connect 26 0 24 1;
#P connect 19 0 17 2;
#P connect 10 1 27 0;
#P connect 28 0 29 0;
#P connect 27 0 29 0;
#P connect 25 0 24 2;
#P connect 10 1 28 0;
#P connect 31 0 29 1;
#P connect 30 0 29 2;
#P connect 11 0 34 0;
#P connect 10 0 34 0;
#P connect 16 0 22 1;
#P connect 10 2 13 0;
#P connect 34 0 14 0;
```

```
#P connect 13 0 14 0;
#P connect 16 0 23 1;
#P connect 11 0 12 0;
#P connect 12 0 13 1;
#P connect 16 0 27 1;
#P connect 16 0 28 1;
#P connect 33 0 18 0;
#P connect 18 0 19 0;
#P connect 10 1 33 0;
#P connect 16 0 18 1;
#P connect 16 0 19 1;
#P connect 32 0 33 1;
#P connect 21 0 26 0;
#P connect 19 0 20 0;
#P connect 20 0 21 0;
#P connect 16 0 26 1;
#P connect 16 0 20 1;
#P connect 21 0 31 0;
#P connect 19 0 25 0;
#P connect 16 0 31 1;
#P connect 16 0 25 1;
#P connect 19 0 30 0;
#P connect 16 0 30 1;
#P connect 15 0 16 0;
#P connect 16 0 32 0;
#P connect 37 0 15 0;
#P connect 15 0 9 2;
#P pop;
#P newobj 99 342 55 196617 p radius2d;
#P newex 111 457 27 196617 gate;
#P newex 82 369 27 196617 t i i;
#P button 387 91 15 0;
#P comment 267 184 100 196617 "p findmin" sort le vecteur
gagnant.;
#N comlet winner-vector;
#P outlet 128 430 15 0;
#N comlet ERROR amt.;
#P outlet 186 188 15 0;
#N vpatcher 277 44 876 720;
#P window setfont "Sans Serif" 9.;
#P comment 63 504 165 196617 ajoute du bruit dans les
connexions;
#N vpatcher 10 59 575 350;
#P window setfont "Sans Serif" 9.;
#P comment 304 255 238 196617
http://en.wikipedia.org/wiki/Simulated_annealing;
#P comment 326 74 205 196617 inspiré par la technique "recuit
simulé" - "simulated annealing";
#P comment 326 44 205 196617 ajoute des valeurs aléatoires
autour de 0. à tout les poids synaptiques;
#P comment 326 26 205 196617 adds random values around 0. to
all weights;
```

```
#P outlet 21 233 15 0;
#P inlet 21 57 15 0;
#P newex 21 205 95 196617 jit.op @op +;
#P newex 279 152 50 196617 / 2.;
#P button 106 88 15 0;
#P newex 106 179 183 196617 jit.op @op - @val 0.;
#P newex 106 132 168 196617 jit.op @op * @val 0.;
#P newex 264 108 75 196617 r $1-temp;
#P newex 106 108 150 196617 jit.noise 1 float32 @dim $3 $2;
#P connect 7 0 6 0;
#P connect 6 0 8 0;
#P fasten 7 0 4 0 26 80 111 80;
#P connect 4 0 0 0;
#P connect 0 0 2 0;
#P connect 2 0 3 0;
#P connect 3 0 6 1;
#P connect 1 0 2 1;
#P connect 1 0 5 0;
#P connect 5 0 3 1;
#P pop;
#P newobj 22 504 39 196617 p temp;
#P comment 415 142 112 196617 this patcher is about jitter-
acrobatics. how to replace 1 line in a matrix of many lines;
#P newex 291 408 79 196617 route dims kats;
#P newex 291 323 91 196617 r $1;
#P comment 415 69 100 196617 "incorporate into küde" met le
vecteur recalculé dans küde;
#N comlet er;
#P inlet 198 63 15 0;
#P newex 22 538 162 196617 jit.matrix $1-küde;
#P newex 37 228 54 196617 jit.matrix;
#P newex 336 233 205 196617 jit.matrix $1-küde;
#P inlet 37 195 15 0;
#P newex 266 432 61 196617 pak dim 0 0;
#P newex 198 116 53 196617 t b b b i;
#P number 198 92 35 9 0 0 0 3 0 0 0 221 221 221 222 222 222 0
0 0;
#P newex 37 263 27 196617 t l b;
#P message 54 321 63 196617 usedstdim 1;
#P message 336 376 63 196617 usedstdim 0;
#P newex 336 355 20 196617 t b;
#P newex 240 363 27 196617 int;
#P newex 160 432 90 196617 pak dstdimend 9 0;
#P newex 49 432 98 196617 pak dstdimstart 0 0;
#P newex 22 466 165 196617 jit.matrix 1 float32 9 9 @thru 0;
#P connect 12 0 0 0;
#P connect 1 0 0 0;
#P connect 2 0 0 0;
#P fasten 5 0 0 0 341 402 27 402;
#P fasten 6 0 0 0 59 357 27 357;
#P fasten 7 0 0 0 42 341 27 341;
#P fasten 9 0 0 0 203 149 27 149;
```

```
#P connect 10 0 0 0;
#P connect 0 0 20 0;
#P connect 20 0 14 0;
#P connect 11 0 13 0;
#P connect 9 1 13 0;
#P connect 13 0 7 0;
#P connect 7 1 6 0;
#P connect 3 0 1 2;
#P connect 15 0 8 0;
#P connect 8 0 9 0;
#P connect 18 0 2 1;
#P connect 9 3 3 0;
#P connect 3 0 2 2;
#P connect 17 0 18 0;
#P connect 18 0 10 1;
#P connect 18 1 10 2;
#P connect 9 2 12 0;
#P connect 12 0 4 0;
#P connect 4 0 5 0;
#P pop;
#P newobj 21 505 118 196617 p incorporate_into_küde;
#N vpatcher 20 74 470 374;
#P window setfont "Sans Serif" 9.;
#P comment 199 89 162 196617 tells us _who_ has smallest
error;
#P newex 92 111 63 196617 unpack 0. 0.;
#P inlet 150 27 15 0;
#P newex 191 55 70 196617 pak max 1. 0.;
#P newex 123 55 65 196617 pak min 0. 0.;
#P newex 92 87 105 196617 jit.findbounds float32;
#P outlet 145 141 15 0;
#P inlet 92 54 15 0;
#P comment 264 58 143 196617 min and max = smallest error;
#P connect 1 0 3 0;
#P fasten 4 0 3 0 128 79 97 79;
#P fasten 5 0 3 0 196 79 97 79;
#P connect 3 0 7 0;
#P connect 7 1 2 0;
#P connect 6 0 4 1;
#P fasten 6 0 5 1 155 48 226 48;
#P pop;
#P newobj 82 188 99 196617 p findmin;
#P newex 327 396 50 196617 pak 1 1;
#P newex 539 435 213 196617 jit.matrix $1-karli 1 float32 $3;
#P newex 327 553 184 196617 jit.matrix $1-küde;
#P newex 327 418 63 196617 prepend dim;
#P newex 327 488 132 196617 jit.op @op * @val 0.02;
#B color 5;
#P newex 327 444 100 196617 jit.noise 1 float32 5;
#B color 5;
#P comment 29 477 79 196617 switch learning on and off;
#N comlet learn on / off;
```

```
#P inlet 21 21 15 0;
#P newex 21 457 88 196617 gate;
#N vpatcher 568 248 1007 618;
#P origin -254 -261;
#P window setfont "Sans Serif" 9.;
#P comment 307 283 105 196617 "gewichtsanpasser" adds the
difference between input-vector and output-vector to the
output-vector;
#P comment 83 235 31 196617 temp;
#P comment 23 179 92 196617 stimulus - réponse;
#P comment 283 34 100 196617 "gewichtsanpasser" calcule les
nouveaux valeurs pour le vecteur gagnant;
#N vpatcher 10 59 527 380;
#P origin 60 313;
#P window setfont "Sans Serif" 9.;
#P comment 96 156 148 196617 valeurs aléatoires autour de 1.;
#P comment 327 108 169 196617 instead of adapting itself
linearly \, learntemp will make certain parts of the winning
neuron adapt themself faster and other slower. Gives
interesting results when doing interpolation p.ex.;
#P newex 84 200 100 196617 jit.op @op + @val 1.;
#P outlet 30 249 15 0;
#P newex 30 223 64 196617 jit.op @op *;
#P inlet 30 56 15 0;
#P newex 257 153 50 196617 / 2.;
#P button 84 87 15 0;
#P newex 84 178 183 196617 jit.op @op - @val 0.;
#P newex 84 131 168 196617 jit.op @op * @val 0.;
#P newex 242 107 80 196617 r $1-learntemp;
#P newex 84 107 134 196617 jit.noise 1 float32 @dim $3;
#P comment 327 89 169 196617 induit du bruit dans
l'apprentissage.;
#P connect 7 0 8 0;
#P connect 8 0 9 0;
#P connect 7 0 5 0;
#P connect 5 0 1 0;
#P connect 1 0 3 0;
#P connect 3 0 4 0;
#P connect 4 0 10 0;
#P connect 10 0 8 1;
#P connect 2 0 3 1;
#P connect 2 0 6 0;
#P connect 6 0 4 1;
#P pop;
#P newobj 117 233 62 196617 p learntemp;
#P newex 117 269 152 196617 jit.op @op +;
#N comlet lern-index;
#P inlet 219 182 15 0;
#P newex 117 200 113 196617 jit.op @op * @val 0.02;
#P button 243 150 15 0;
#N comlet winner-number;
#P inlet 117 67 15 0;
```

```
#N vpatcher 221 172 611 455;
#P window setfont "Sans Serif" 9.;
#P newex 168 42 27 196617 + 1;
#P outlet 76 242 15 0;
#P button 62 37 15 0;
#P button 121 128 15 0;
#P inlet 168 21 15 0;
#P newex 157 159 105 196617 jit.matrix $1-küde;
#P newex 60 60 45 196617 loadbang;
#P newex 144 80 27 196617 - 1;
#P message 144 128 65 196617 splitpoint \$1;
#P newex 60 216 42 196617 jit.split;
#P newex 60 86 31 196617 t 1 3;
#P number 192 66 35 9 0 0 0 3 0 0 0 221 221 221 222 222 222 0
0 0;
#P message 192 103 65 196617 splitpoint \$1;
#P message 60 113 59 196617 splitdim \$1;
#P newex 95 185 42 196617 jit.split;
#P comment 268 40 100 196617 ce sous-patch cherche dans küde
(la matrice des poids synaptiques) la ligne / le neurone qui a
gagné et l'affiche par le outlet.;
#P connect 13 0 9 0;
#P connect 9 0 5 0;
#P connect 5 0 2 0;
#P connect 2 0 6 0;
#P fasten 7 0 6 0 149 151 65 151;
#P connect 1 0 6 0;
#P connect 6 1 14 0;
#P connect 10 0 1 0;
#P fasten 3 0 1 0 197 151 100 151;
#P fasten 2 0 1 0 65 151 100 151;
#P connect 4 0 12 0;
#P connect 4 0 8 0;
#P connect 8 0 7 0;
#P connect 12 0 10 0;
#P connect 11 0 15 0;
#P connect 15 0 4 0;
#P connect 5 1 4 0;
#P connect 4 0 3 0;
#P pop;
#P newobj 259 149 72 196617 p qui_a_gagné;
#P newex 117 176 62 196617 jit.op @op -;
#P button 117 89 15 0;
#P newex 117 111 99 196617 jit.matrix $1-karli;
#P outlet 117 292 15 0;
#P comment 63 203 51 196617 learnrate;
#P connect 6 0 3 0;
#P connect 3 0 2 0;
#P connect 2 0 4 0;
#P connect 4 0 8 0;
#P connect 8 0 11 0;
#P connect 11 0 10 0;
```

```
#P connect 10 0 1 0;
#P fasten 5 0 4 1 264 171 174 171;
#P connect 9 0 8 1;
#P fasten 6 0 5 0 122 85 264 85;
#P connect 7 0 5 0;
#P connect 5 0 10 1;
#P pop;
#P newobj 99 401 103 196617 p gewichtsanpasser2;
#N vpatcher 198 80 930 642;
#P origin 267 80;
#P window setfont "Sans Serif" 9.;
#P comment 388 442 92 196617 the smallest error;
#P comment 514 206 154 196617 to determine the no. of
iterations to calculate the sum of each line;
#P comment 12 127 200 196617 make karli the same size as küde.
this is done for multiplication of the two matrices;
#P comment 16 520 228 196617
http://en.wikipedia.org/wiki/Euclidean_distance;
#P comment 16 480 100 196617 c'est la distance euklidienne qui
est calculé:;
#P newex 57 41 79 196617 route dims kats;
#P newex 57 21 176 196617 r $1;
#P comment 16 404 100 196617 ici sont mesurés les "distances"
entre vecteurs présentés et vécteurs dans küde (matrix des
poids);
#P newex 382 209 131 196617 expr (log($f1)/log(2.)) - 1;
#P newex 214 207 40 196617 t b b l;
#P newex 352 231 40 196617 uzi 4;
#N comlet smalles value;
#P outlet 332 467 15 0;
#P newex 332 439 53 196617 jit.3m;
#P outlet 310 467 15 0;
#N vpatcher 156 198 608 516;
#P window setfont "Sans Serif" 9.;
#P comment 65 240 228 196617
http://en.wikipedia.org/wiki/Euclidean_distance;
#P outlet 65 213 15 0;
#P newex 65 187 241 196617 jit.op @op +;
#P newex 65 163 133 196617 jit.op @op -;
#P newex 188 136 98 196617 jit.op @op * @val 2;
#P newex 188 115 64 196617 jit.op @op *;
#P newex 296 115 109 196617 jit.op @op pow @val 2;
#P newex 65 115 109 196617 jit.op @op pow @val 2;
#P inlet 296 58 15 0;
#P inlet 65 58 15 0;
#P connect 0 0 2 0;
#P connect 2 0 6 0;
#P connect 6 0 7 0;
#P connect 7 0 8 0;
#P connect 0 0 4 0;
#P connect 4 0 5 0;
#P connect 5 0 6 1;
```

```
#P connect 1 0 4 1;
#P connect 1 0 3 0;
#P connect 3 0 7 1;
#P pop;
#P newobj 214 158 187 196617 p la_formule_euklid;
#P button 310 386 15 0;
#P newex 310 405 196 196617 jit.matrix $1-error;
#P newex 214 182 161 196617 jit.matrix $1-error;
#P newex 244 328 143 196617 jit.matrix $1-error;
#P message 352 251 106 196617 jit_matrix $1-error;
#P newex 244 307 63 196617 jit.op @op +;
#P newex 244 270 199 196617 jit.demultiplex float32
@demultiplexdim 0;
#P newex 90 66 27 196617 - 1;
#P newex 57 66 27 196617 - 1;
#P newex 127 93 61 196617 pak dim 0 0;
#P window setfont "Fixedwidth Serif" 10.;
#P newex 7 92 110 1441802 pak dstdimend 0 0;
#P window setfont "Sans Serif" 9.;
#P newex 214 132 481 196617 jit.matrix $1-karli2d 1 float32 4
100 @usedstdim 1 @dstdimstart 0 0 @dstdimend 3 99;
#P button 356 71 15 0;
#P newex 391 95 141 196617 jit.matrix $1-küde;
#P newex 214 94 155 196617 jit.matrix $1-karli 1 float32 $3;
#N comlet prototypvektor;
#P inlet 391 52 15 0;
#N comlet inputvektor;
#P inlet 214 50 15 0;
#P comment 389 326 212 196617 the sum of each line - please \,
please cycling74 make an object which does this !!!!;
#P connect 26 0 27 0;
#P connect 27 0 9 0;
#P connect 9 0 7 1;
#P connect 27 1 10 0;
#P connect 10 0 7 2;
#P connect 27 0 8 1;
#P connect 27 1 8 2;
#P connect 1 0 3 0;
#P fasten 8 0 6 0 132 121 219 121;
#P fasten 7 0 6 0 12 121 219 121;
#P connect 3 0 6 0;
#P connect 6 0 18 0;
#P connect 18 0 15 0;
#P connect 15 0 23 0;
#P connect 23 2 11 0;
#P fasten 13 0 11 0 357 268 249 268;
#P connect 11 0 12 0;
#P connect 12 0 14 0;
#P fasten 11 1 12 1 343 296 302 296;
#P fasten 23 0 17 0 219 239 219 239 219 385;
#P connect 17 0 16 0;
#P connect 16 0 19 0;
```

```
#P connect 16 0 20 0;
#P connect 20 0 21 0;
#P fasten 23 1 22 0 234 227 357 227;
#P connect 22 0 13 0;
#P fasten 1 0 5 0 219 68 361 68;
#P fasten 27 0 24 0 62 61 387 61;
#P connect 24 0 22 1;
#P connect 5 0 4 0;
#P connect 2 0 4 0;
#P connect 4 0 18 1;
#P pop;
#P newobj 82 158 114 196617 p distanzmesser_euklid;
#P newex 513 144 53 196617 i $3;
#P newex 591 144 52 196617 i $2;
#N comlet learning rate;
#P inlet 192 314 15 0;
#N comlet trainingsvektoren;
#P inlet 82 135 15 0;
#P newex 387 113 136 196617 t b b;
#P comment 400 399 100 196617 randomisation @ init;
#P comment 779 488 101 196617 robin.meier@free.fr \;
robin.meier.free.fr \; neuromuse.org \; cirm-manca.org \; nice
\, 23.5.04;
#P comment 33 253 35 196617 radius;
#P connect 11 0 32 0;
#P connect 38 0 32 0;
#P fasten 32 0 10 0 26 444 26 444;
#P connect 10 0 20 0;
#P connect 4 0 8 0;
#P connect 8 0 19 0;
#P connect 19 0 28 0;
#P fasten 28 0 25 0 87 259 87 259;
#P connect 27 0 25 0;
#P connect 31 0 30 0;
#P connect 30 0 27 0;
#P connect 25 1 9 0;
#P connect 9 0 10 1;
#P fasten 32 0 26 0 26 437 116 437;
#P connect 28 1 30 1;
#P connect 9 0 22 0;
#P connect 25 0 26 1;
#P fasten 26 0 20 1 116 492 134 492;
#P fasten 5 0 27 1 197 334 149 334;
#P connect 19 0 40 0;
#P connect 8 1 19 1;
#P connect 8 1 21 0;
#P fasten 27 1 9 1 149 391 197 391;
#P connect 5 0 9 1;
#P fasten 32 0 33 0 26 107 261 107;
#P connect 33 0 29 0;
#P fasten 32 2 43 0 352 101 279 101;
#P fasten 32 1 43 0 189 101 279 101;
```

```
#P fasten 28 2 33 1 155 248 297 248;
#P fasten 32 6 37 0 1004 359 332 359;
#P connect 37 0 18 0;
#P connect 18 0 15 0;
#P fasten 3 0 13 0 392 438 332 438;
#P fasten 34 0 13 0 437 438 332 438;
#P connect 15 0 13 0;
#P connect 13 0 14 0;
#P connect 14 0 16 0;
#P fasten 3 0 33 2 392 249 333 249;
#P connect 37 1 18 1;
#P fasten 32 3 24 0 515 87 392 87;
#P connect 24 0 3 0;
#P fasten 32 2 34 0 352 162 437 162;
#P fasten 34 1 14 1 454 483 454 483;
#P connect 3 1 7 0;
#P connect 7 0 35 0;
#P connect 35 0 39 0;
#P connect 36 0 39 0;
#P fasten 37 0 44 0 332 391 544 391;
#P connect 44 0 17 0;
#P fasten 3 1 6 0 518 138 596 138;
#P connect 6 0 36 0;
#P connect 32 4 41 0;
#P connect 32 5 45 0;
#P pop;
```

7.6 Catégorisation de Typhons

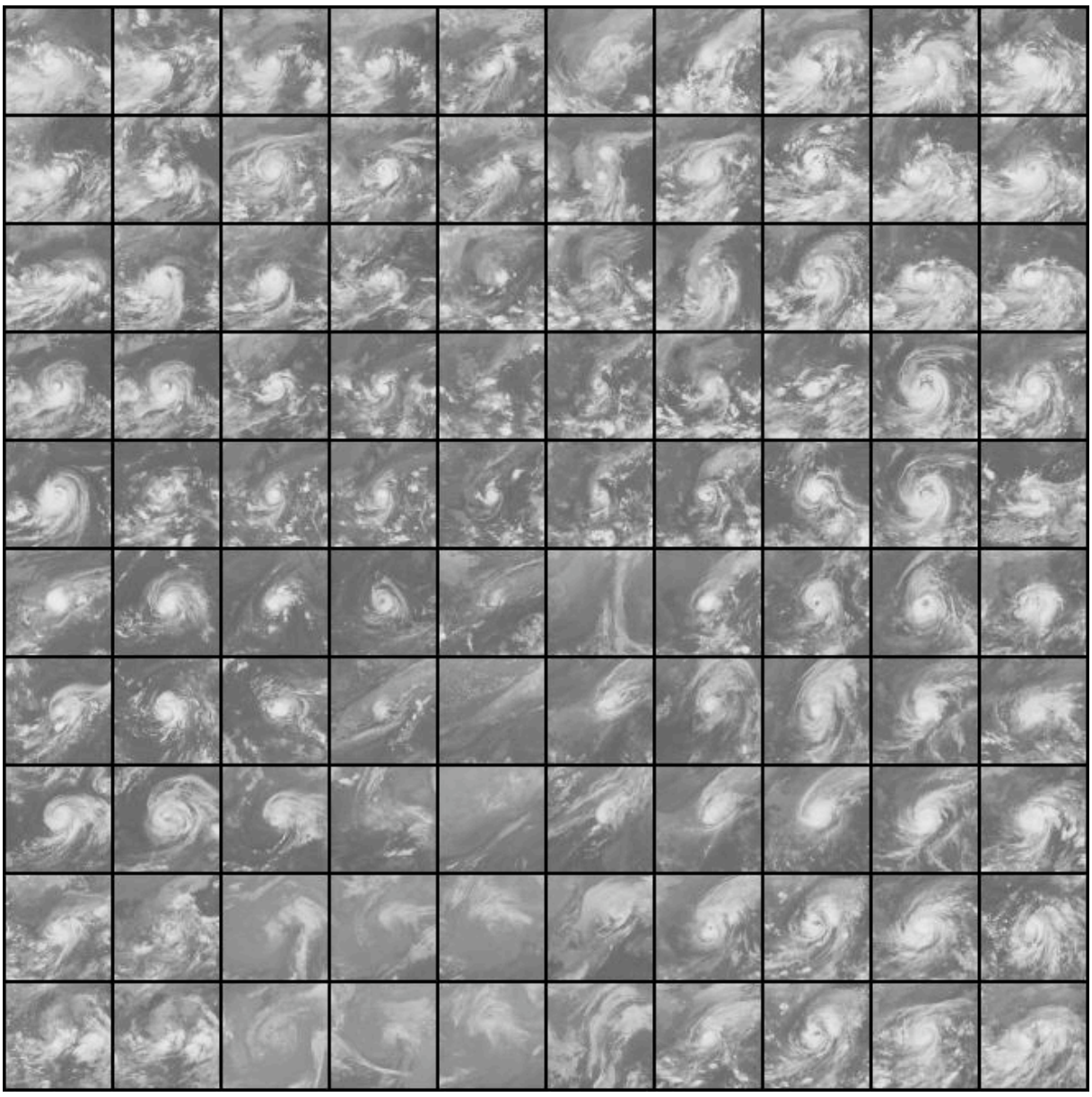

Fig. 29 : Ce SOM visualise la catégorisation de photos aériennes de Typhons. Sur le site, il est possible de cliquer sur chacune des photos pour accéder à une multitude de données sur chacun des Typhons (données à grand nombre de dimensions).

http://agora.ex.nii.ac.jp/digital-typhoon/start_som_wnp.html.en

7.7 $x^2 + c$

Voici les résultats de 37 itérations de $x^2 + c$ avec une précision de 64bit et avec une précision de 32bit. On voit qu'après deux itérations déjà, les résultats divergent légèrement. À partir de la 25ème itération par contre, les séries divergent totalement, montrant ainsi une grande sensibilité aux conditions initiales qui sont une divergence infinitésimale.

x = 0.5 et c = -2

Itération	64bit précision	32bit précision	différence
1	-1.75	-1.75	0
2	1.0625	1.0625	0
3	-0.87109375	-0.871094	2.5E-07
4	-1.241195679	-1.241196	3.21289E-07
5	-0.459433287	-0.459433	-2.87149E-07
6	-1.788921055	-1.788921	-5.46592E-08
7	1.20023854	1.200238	5.39803E-07
8	-0.559427448	-0.559428	5.52428E-07
9	-1.687040931	-1.68704	-9.30903E-07
10	0.846107103	0.846104	3.10254E-06
11	-1.284102771	-1.284107	4.22897E-06
12	-0.351080073	-0.351068	-1.20734E-05
13	-1.876742782	-1.876751	8.21797E-06
14	1.52216347	1.522195	-3.15301E-05
15	0.316981629	0.317078	-9.63709E-05
16	-1.899522647	-1.899462	-6.06468E-05
17	1.608186286	1.607955	0.000231286
18	0.58626313	0.585518	0.00074513
19	-1.656295543	-1.657169	0.000873457
20	0.743314925	0.746209	-0.002894075
21	-1.447482922	-1.443172	-0.004310922
22	0.09520681	0.082744	0.01246281
23	-1.990935663	-1.993153	0.002217337
24	1.963824815	1.972661	-0.008836185
25	1.856607905	1.891391	-0.034783095
26	1.446992912	1.577358	-0.130365088
27	0.093788487	0.488059	-0.394270513
28	-1.99120372	-1.761798	-0.22940572
29	1.964892253	1.103934	0.860958253
30	1.860801568	-0.781331	2.642132568
31	1.462582474	-1.389522	2.852104474
32	0.139147493	-0.069228	0.208375493
33	-1.980637975	-1.995208	0.014570025
34	1.922926789	1.980853	-0.057926211