Emergence of Oriented Circuits driven by Synaptic Pruning associated with Spike-Timing-Dependent Plasticity (STDP)

Javier Iglesias

Faculté des Sciences

Emergence of Oriented Circuits driven by Synaptic Pruning associated with Spike-Timing-Dependent Plasticity (STDP)

Emergence de circuits neuromimétiques orientés
sous l'effet de l'épissage associé à la plasticité synaptique
à modulation temporelle relative (STDP)

Thèse de doctorat

présentée à

la Faculté des Sciences de l'Université de Lausanne	l'Université Grenoble I Joseph Fourier Neurosciences - Neurobiologie

par

Javier Iglesias

Diplômé en Biologie
Université de Lausanne

Lausanne et Grenoble
2005

UNIVERSITE DE LAUSANNE

Faculté des sciences

Imprimatur

Vu le rapport présenté par le jury d'examen, composé de

Président	M. le Prof. François **Grize**
Directeur de thèse	M. le Prof. Alexandro **Villa**
Directeur de thèse	M. le Prof. Marco **Tomassini**
Experts	M. le Prof. Juan Manuel **Moreno Arrostegui**
	Mme le Dr Brigitte **Quenet**
	M. le Dr Jean-François **Vibert**

le Conseil de Faculté autorise l'impression de la thèse de

Monsieur Javier Iglesias

titulaire du Diplôme d'études approfondies en bioinformatique des universités de Genève et Lausanne

intitulée

Emergence de circuits neuromimétiques orientés sous l'effet de l'épissage associé à la plasticité synaptique à modulation temporelle relative (STDP)

Lausanne, le 28 septembre 2005

pour Le Doyen de la Faculté des Sciences

François Grize

M. le Prof. François Grize

to my parents

Maria Esther and Fernando

who know the price
of every single page

Emergence of Oriented Circuits driven by Synaptic Pruning associated with Spike-Timing-Dependent Plasticity (STDP)

Javier Iglesias

INFORGE – Institut d'Informatique et Organisation, Université de Lausanne

Laboratoire de Neurosciences Précliniques – INSERM U318, Université Grenoble 1

Massive synaptic pruning following over-growth is a general feature of mammalian brain maturation. Pruning starts near time of birth and is completed by time of sexual maturation. Trigger signals able to induce synaptic pruning could be related to dynamic functions that depend on the timing of action potentials. Spike-timing-dependent synaptic plasticity (STDP) is a change in the synaptic strength based on the ordering of pre– and postsynaptic spikes. The relation between synaptic efficacy and synaptic pruning suggests that the weak synapses may be modified and removed through competitive "learning" rules. This plasticity rule might produce the strengthening of the connections among neurons that belong to cell assemblies characterized by recurrent patterns of firing. Conversely, the connections that are not recurrently activated might decrease in efficiency and eventually be eliminated.

The main goal of our study is to determine whether or not, and under which conditions, such cell assemblies may emerge out of a locally connected random network of integrate-and-fire units distributed on a 2D lattice receiving background noise and content-related input organized in both temporal and spatial dimensions. The originality of our study stands on the relatively large size of the network, 10,000 units, the duration of the experiment, 10^6 time units (one time unit corresponding to the duration of a spike), and the application of an original bio-inspired STDP modification rule compatible with hardware implementation.

A first batch of experiments was performed to test that the randomly generated connectivity and the STDP-driven pruning did not show any spurious bias in absence of stimulation. Among other things, a scale factor was approximated to compensate for the network size on the activity. Networks were then stimulated with the spatiotemporal patterns. The analysis of the connections remaining at the end of the simulations, as well as the analysis of the time series resulting from the interconnected units activity, suggest that feed-forward circuits emerge from the initially randomly connected networks by pruning.

Émergence de circuits neuromimétiques orientés sous l'effet de l'épissage associé à la plasticité synaptique à modulation temporelle relative (STDP)

Javier Iglesias

INFORGE – Institut d'Informatique et Organisation, Université de Lausanne

Laboratoire de Neurosciences Précliniques – INSERM U318, Université Grenoble 1

L'élagage massif des synapses après une croissance excessive est une phase normale de la maturation du cerveau des mammifères. L'élagage commence peu avant la naissance et est complété avant l'âge de la maturité sexuelle. Les facteurs déclenchants capables d'induire l'élagage des synapses pourraient être liés à des processus dynamiques qui dépendent de la temporalité relative des potentiels d'actions. La plasticité synaptique à modulation temporelle relative (STDP) correspond à un changement de la force synaptique basé sur l'ordre des décharges pré– et post-synaptiques. La relation entre l'efficacité synaptique et l'élagage des synapses suggère que les synapses les plus faibles pourraient être modifiées et retirées au moyen d'une règle "d'apprentissage" faisant intervenir une compétition. Cette règle de plasticité pourrait produire le renforcement des connexions parmi les neurones qui appartiennent à une assemblée de cellules caractérisée par des motifs de décharge récurrents. A l'inverse, les connexions qui ne sont pas activées de façon récurrente pourraient voir leur efficacité diminuée et être finalement éliminées.

Le but principal de notre travail est de déterminer s'il serait possible, et dans quelles conditions, que de telles assemblées de cellules émergent d'un réseau d'unités *integrate-and-fire* connectées aléatoirement et distribuées à la surface d'une grille bidimensionnelle recevant à la fois du bruit et des entrées organisées dans les dimensions temporelle et spaciale. L'originalité de notre étude tient dans la taille relativement grande du réseau, 10'000 unités, dans la durée des simulations, 1 million d'unités de temps (une unité de temps correspondant à une milliseconde), et dans l'utilisation d'une règle STDP originale compatible avec une implémentation matérielle.

Une première série d'expériences a été effectuée pour tester que la connectivité produite aléatoirement et que l'élagage dirigé par STDP ne produisaient pas de biais en absence de stimulation extérieure. Entre autres choses, un facteur d'échelle a pu être approximé pour compenser l'effet de la variation de la taille du réseau sur son activité. Les réseaux ont ensuite été stimulés avec des motifs spatiotemporels. L'analyse des connexions se maintenant à la fin des simulations, ainsi que l'analyse des séries temporelles résultantes de l'activité des neurones, suggèrent que des circuits *feed-forward* émergent par l'élagage des réseaux initiallement connectés au hasard.

Adresses

Président
Prof. François Grize
École des Hautes Études Commerciales
Bâtiment de la Faculté des Sciences Humaines 1
Université de Lausanne
CH-1015 Lausanne

Rapporteur
Dr. Brigitte Quenet
École Supérieure de Physique et de Chimie Industrielles
Bâtiment C
10, rue Vauquelin
FR-75231 Paris Cedex 5

Rapporteur
Prof. Juan Manuel Moreno Aróstegui
Universitat Politècnica de Catalunya
Departament d'Enginyeria Electrònica
Edifici C4, Campus Nord, Gran Capità s/n
ES-08034 Barcelona

Codirecteur de Thèse
Prof. Marco Tomassini
INFORGE – Institut d'Informatique et Organisation
Collège Propédeutique 1
CH-1015 Lausanne

Codirecteur de Thèse
Prof. Alessandro E.P. Villa
Laboratoire de Neurosciences Précliniques – INSERM U318
Pavillon B - CHU
BP 217
FR-38043 Grenoble Cedex 9

Expert
Dr. Jean-François Vibert
B3E ESIM INSERM U707
Faculté de Médecine Saint-Antoine
27, rue Chaligny
FR-75571 Paris Cedex 12

Liste des publications

Iglesias J., Eriksson J., Pardo B., Tomassini M., Villa A.E.P. Stimulus-Driven Unsupervised Pruning in Large Neural Networks, *in proceedings of BV&Ai 2005*, Lecture Notes in Computer Science, LNCS 3704:59-68, 2005.

Moreno J.M., Eriksson J., Iglesias J., Villa A.E.P. Implementation of Biologically Plausible Spiking Neural Networks Models on the POEtic Tissue, *in proceedings of ICES 2005*, Lecture Notes in Computer Science, LNCS 3637:188, 2005.

Iglesias J., Eriksson J., Pardo B., Tomassini M., Villa A.E.P. Emergence of Oriented Cell Assemblies Associated with Spike-Timing-Dependent Plasticity, *in ICANN 2005*, Lecture Notes in Computer Science, LNCS 3696:127–32, 2005.

Iglesias J., Eriksson J., Grize F., Tomassini M., Villa A.E.P. Dynamics of Pruning in Simulated Large-Scale Spiking Neural Networks, *Biosystems*, 79(1-3):11-20, 2005.

Villa A.E.P., Tetko I.V., Iglesias J. Computer Assisted Neurophysiological Analysis of Cell Assemblies, *Neurocomputing*, 38-40:1025–30, 2001.

Villa A.E.P., Tetko I.V., Iglesias J., Filipov D. Transdisciplinary approach to scientific data analysis through Internet, in *Transdisciplinarity: Joint Problem-Solving among Science, Technology and Society?* (Häberli R., Scholz R.W., Bill A., Welti M. Eds.), Haffmans Sachbuch Verlag Zürich, 550-5, 2000

Liste des conférences et des présentations

European Conference on Evolvable Systems (ICES 2005), September, 12–14, 2005. presentation, Sitges (Barcelona), Spain.

International Conference on Artificial Neural Networks (ICANN 2005), September 11–15, 2005. Oral presentation, Warsaw, Poland.

6th International Neural Coding Workshop (NCWS 2005), August 23–28, 2005. Oral presentation, Marburg, Germany.

Federation of European Neuroscience Societies (FENS Forum 2004), July 10–14, 2004. Poster presentation, Lisboa, Portugal.

Workshop: Nonlinear dynamics and noise in biological systems, April 19–21, 2004. Oral presentation, Torino, Italy.

International School: Does noise simplify or complicate the dynamics of nonlinear systems?, April 13–17, 2004. Poster presentation, Torino, Italy.

Second International Conference on Multimedia and Information and Communication Technologies in Eduction (m-ICTE 2003), December 3–6, 2003. Poster presentation, Badajoz, Spain.

5th International Neural Coding Workshop (NCWS 2003), September 20–25, 2003. Poster presentation, Aulla, Italy.

EU Advanced Course in Computational Neurosciences 2003, August 11 - September 5, 2003. Oral presentation, Obidos, Portugal.

Federation of European Neuroscience Societies (FENS Forum 2002), July 13–17, 2002. Poster presentation, Paris, France.

Information Science Technology meeting (IST 2002), November 4–6, 2002. Poster presentation, Copenhaguen, Denmark.

4th International Neural Coding Workshop (NCWS 2001), September 10–14, 2001. Poster presentation, Plymouth, UK.

31st Annual General Meeting of the European Brain and Behaviour Society (EBBS 1999), September 29 - October 3, 1999. Poster presentation, Roma, Italy.

With logic, one can go from A to B.
With imagination, one goes anywhere.
– Albert Einstein

Preface

Óbidos, August 2003. It's a warm evening in the medieval village located north of Lisbon, Portugal. We are having a drink under a weeping willow after spending the day at the European Union Advanced Course in Computational Neuroscience. Rudy Guyonneau, a fellow student, suggests an intriguing theory:

> " What if the brain were just a silly device,
> equipped with a security mechanism
> that starts producing chaotic patterns of activity
> as soon as we try to reverse-engineer it? "

I have the conviction that we come closer to unraveling the neural code when we mutually exchange knowledge across the artificial barriers of scientific fields. Together, physiologists, mathematicians, chemists, engineers, ... can meet the bet of deducing how the brain performs, with the only help coming from the concepts and artifacts our own (collective) (cyber-)brains can produce. Only 116 years elapsed since Santiago Ramón y Cajal argued that nerve cells are independent elements, and 57 years since the announcement of the transistor invention by Bell Labs and there is plenty of room for progress in the field of computational neuroscience. The present work marks the beginning of my own modest contribution to this endeavour.

Acknowlegements

Beside the transdisciplinary approach of my research, I happen to be involved in a multicultural scientific and social environment. This acknowledgement section is therefore written in several languages: what is the sense of thanking people in a language they do not fully understand and that I do not master anyway?

To Prof. Alessandro E.P. Villa who has been my guide and tutor for the last few years, pushing me constantly to work harder and better. I hope I have deserved all the time and energy he spent helping me to find my way into Science.

To Prof. Marco Tomassini for proposing to a young biologist a position in a computer science research group, and thus giving me the opportunity to realize this transdisciplinary journey.

To Prof. François Grize, president of the jury, who challenged me to undertake the responsibility of the computing equipement for the group, and provided me the thrust and support needed to accomplish this task to the point that everyone thinks I am useless.

I would like to thank the jury members, Dr. Brigitte Quenet, Prof. Juan Manuel Moreno Arostegui, and Dr. Jean-François Vibert, for accepting to review and critically comment my work despite the tight schedule and the additional paperwork imposed by the joint Ph.D. programme. I am really proud that *you* recognized me as a colleague.

To Prof. L.B. Kier from the Virginia Commonwealth University, Richmond (VA), who happened to visit Prof. Tomassini during summer 2005, just in time to read, correct the spelling, and comment this document. He deserves many thanks for his contribution as well as the kind comments that came along with the corrections.

To Dr. Jan Eriksson, Dr. Yoshiyuki Asai, Dr. Roberta Sirovich and Dr. Sean Hill for their direct or indirect contributions to my scientific work. I would like to thank collectively the numerous scientists that commented my work during the workshops, poster sessions, and other scientific manifestations. My path has been perceptibly deflected at their contact.

The work discussed in this document would not have been possible without the help of an army of devoted and gifted coding hands, giving their work away for free. The resources of about 50 different open source projects have been put together to interact into this research project. I am indepted to the Open Source community as a whole, and I see no other way to thank it, than to continue to contribute my work, hoping that the culture of meritocracy and open standards will finally rule the world of software. Who knows? maybe other human fields of activity could be enlighted by that time.

À Madame Elisabeth Fournier Pulfer pour son aide dans les démarches administratives et son soutien tout au long de mon engagement à l'IIS/Inforge. Et c'est pas encore fini !

À Messieurs Michel Schüpbach, Julien Gianotti et Gill Cordey qui furent mes compagnons sur la Grande Galère des Ressources Informatiques de l'Institut. Ensemble, nous avons réussi, non sans mal, à faire d'une épave en perdition une petite embarcation grinçante mais téméraire. Une transformation périlleuse qu'il nous a fallu obtenir tout en naviguant. Qu'ils soient remerciés ici pour leur aide par le passé et pour continuer à faire flotter l'engin et péréniser ainsi mon investissement.

À tous ceux qui ont supporté (ou supportent encore) mes bruits incongrus dans le bureau commun : Laurent Bucher, Mathieu Perrenoud, Jonathan Jaccard, Dr. Mario Giacobini, Marc Mercier, Christian Darabos, ainsi que les autres assistants répartis dans les bureaux annexes du Collège Propédeutique 1 : Dr. Leonardo Vanneschi, Leslie Luthi et Denis Rochat. S'ils ont subi ma présence et mes absences ces dernières années, j'espère qu'ils sont conscients que nos échanges et leurs questions m'ont permis, d'une manière ou d'une autre, d'acquérir un savoir-faire informatique qui m'est précieux aujourd'hui, justifiant ainsi mon approche transdisciplinaire. Même si ce n'est pas évident au jour le jour, j'ai été ravi de me mettre à leur service pendant ces quelques années. J'aimerais remercier ici collectivement les équipes de l'U318 et de l'Inforge dans leur ensemble pour m'avoir accueilli.

À Monsieur Gérald Marcel Savary pour son incroyable patience face à un garnement indiscipliné et maladivement décalé. Les problèmes administratifs liés à la cotutelle furent admirablement démêlés par Madame Jeanine DiMarco du secrétariat de l'Ecole Doctorale Chimie et Sciences du Vivant, ainsi que Mesdames Elisabeth Perrin et Céline Legrenzi du secrétariat du Collège Doctoral de l'Université Grenoble 1 Joseph Fourier. Je n'aurais pu tracer mon chemin au travers de la jungle administrative française sans l'aide de ces trois anges gardiens. Qu'elles soient ici chaleureusement remerciées pour la patience qu'elles ont su montrer à mon égard.

À mon épouse, Denise Anja Hayward Iglesias, pour avoir relu et corrigé ce document à plusieurs reprises ; pour avoir suggéré les améliorations de structure et de forme nécessaires ; pour avoir supporté pendant quatre ans la vie complexe et futile, ponctuée de doutes et de coups de blues, du docteur en devenir ; pour avoir accepté de m'accompagner dans la vie inconfortable que nécessite le parcours académique. Merci de tout mon coeur pour savoir être là et pour me laisser en faire autant.

À mes amis les plus chers, Laurent Burgbacher et Jacques Beaud, qui continuent à me parler, à mon grand étonnement, malgré les "ça va pas être possible" qui n'ont pas manqué ces dernières années. Si je n'ai pas été suffisamment disponible, vous avez su, par votre présence, me donner l'élan nécessaire. Il est rare, semblerait-il, de rencontrer des personnes véritablement, fondamentalement talentueuses. Alors en rencontrer deux...

A mis padres, Maria Esther y Fernando Iglesias quien, año tras año, siempre me han acompañado en mis empresas las mas ambiciosas. No solo les debo la vida, claro, que también les debo de haberme enseñado a escoger las soluciones justas (usualmente complicadas), la abnegación, y el valor del trabajo bien hecho. Les doy las gracias por haberme transmitido estos valores, aunque no sean caracteres genéticos.

Javier Iglesias
Lausanne, 9 octobre 2005

Il est très difficile de trouver
un chat noir dans une pièce sombre,
surtout s'il n'y est pas.

– Stéphane Belloc

Résumé

L'élagage massif des synapses est une propriété générale des cerveaux des mammifères (Rakic et al., 1986). L'élagage commence peu avant la naissance et est complété avant l'âge de la maturité sexuelle. Les signaux déclanchants capables d'induire l'élagage des synapses pourraient être liés à des fonctions dynamiques qui dépendent du moment des potentiels d'action. La plasticité synaptique à modulation temporelle relative (*Spike-Timing Dependent synaptic Plasticity* – STDP) est un changement de la force des synapses basé sur l'ordre pré– et postsynaptique des potentiels d'action. La relation entre l'efficacité synaptique et l'élagage des synapses (Chechik et al., 1999; Mimura et al., 2003) suggère que les synapses les plus faiblement efficaces pourraient être modifiées et éliminées au moyen de règles «d'apprentissage» compétitives. Une telle règle de plasticité entraînerait le renforcement des connexions entre les neurones qui appartiennent à des groupes de cellules caractérisés par des motifs récurrents de décharges. Ces groupes particuliers seront définis «assemblées de cellules». Inversement, l'efficacité des connexions qui ne sont pas activées de façon récurrente pourrait décroître et ces connexions seraient finalement éliminées.

Le but principal de notre travail est de déterminer s'il est possible, et sous quelles conditions, que de telles assemblées de cellules puissent émerger d'un réseau d'unités *integrate-and-fire* distribuées sur une grille bidimensionnelle et connectées localement de manière aléatoire, alors qu'elles sont caractérisées par une activité spontanée et reçoivent des afférences organisées à la fois dans les dimensions spatiale et temporelle. L'originalité de notre travail se trouve dans la taille relativement grande du réseau (10'000 unités), la durée des expériences (10^6 unités de temps, où une unité de temps correspond à la durée d'un potentiel d'action), et à l'application d'une règle STDP originale bio-inspirée de modification des poids synaptiques compatible avec une implémentation matérielle (Eriksson et al., 2003).

Plasticité synaptique

Nous avons considéré de nombreuses hypothèses simplificatrices dans notre modèle, comme la présence de seulement deux types d'unités, leur dynamique *integrate-and-fire*, leurs distributions, ainsi que la dynamique de la fonction de transfert des synapses qui les connectent. En tenant compte de toutes ces approximations, nous avons observé que le réseau parvenait à un état d'équilibre lorsque les poids synaptiques étaient soit renforcés à leur valeur maximale, soit diminués à leur valeur la plus faible (figure 4.4). Nos résultats sont en accord avec la distribution bimodale des forces synaptiques observée avec d'autres modèles basés sur STDP (Chechik et al., 1999; Abbott and Nelson, 2000; Dayan and Abbott, 2001). Cet effet est interprété comme étant la démonstration qu'une règle STDP entraîne les neurones présynaptiques à entrer en compétition pour obtenir le contrôle de la production de potentiels d'action par la cellule postsynaptique. Il a été montré que cette compétition assurait cependant des afférences constantes au neurone postsynaptique (Abbott and Nelson, 2000; Song and Abbott, 2001).

De plus en plus d'indices suggèrent que les synapses à états ont un fondement biologique vraisemblable (voir Montgomery and Madison (2004) pour une revue). Le modèle de synapse que nous avons utilisé n'était cependant pas basé sur ces résultats expérimentaux. Nos niveaux d'activation sont issus d'une simplification requise pour l'implémentation matérielle discutée ailleurs dans ce document (section 7.2.1, p.109). Ils peuvent être interprétés comme étant une combinaison de deux facteurs : le nombre de boutons synaptiques entre les unités pré– et postsynaptiques et la modification de la conductance synaptique. Néanmoins, le travail expérimental et théorique entrepris autour des synapses à états nous aidera certainement à affiner notre simplification et à maintenir un degré de vraisemblance biologique dans le futur. Parmi les questions en suspens, la question de la dépendance de la régulation des récepteurs NMDA de l'état de la synapse reste ouverte. Si cette dépendance était avérée, un aspect métaplastique (Abraham and Bear, 1996) pourrait être ajouté à la plasticité synaptique à modulation temporelle relative.

La métaplasticité est définie comme la plasticité de la plasticité. Un modèle en cascade a été proposé dernièrement (Fusi et al., 2005) comme un mécanisme théorique pour la mémorisation des expériences quotidiennes. L'acquisition de tels souvenirs – comme lors de l'apprentissage au premier essai – requiert un haut degré de plasticité, alors que la rétention de ces mêmes souvenirs nécessite une protection contre les changements induits par l'activité et les expériences postérieures. Les auteurs définissent deux niveaux de force synaptique («faible» et «fort») caractérisés par différents degrés de plasticité. Chaque niveau est associé à une cascade de n états qui introduisent un intervalle de transitions possibles entre les deux niveaux synaptiques. Passer d'un état à un autre ne modifie en rien la force de la synapse, mais diminue la probabilité de changer de niveau, induisant une métaplasticité.

Il faut remarquer que la métaplasticité est liée de manière inhérente à l'apprentissage. Dans notre travail, nous avons simulé un élagage synaptique contrôlé par l'activité des cellules dans le cadre d'une expérience de développement. Néanmoins, un léger changement à notre modèle permettrait une transition graduelle de l'élagage à l'apprentissage tout en conservant le même principe de fonctionnement synaptique. Il serait alors possible de simuler les transitions de l'enfance à l'adolescence, et de l'adolescence à l'âge adulte. Dans la section 3.3.1, p.20, nous observons que la variable continue $L_{ji}(t)$ est utilisée pour contrôler les niveaux d'activation discrets $A_{ji}(t)$ à travers une règle STDP. Les limites définies par l'utilisateur $L_0 < L_1 < \cdots < L_{N-1} < L_N$ ont été utilisées pour déterminer les transitions entre les différents niveaux d'activation, avec une lente décroissance menant $L_{ji}(t)$ vers la limite inférieure, entraînant l'élagage de la connexion (figure 3.3c). Cette règle pourrait être modifiée continuellement vers une relation entre $L_{ji}(t)$ et $A_{ji}(t)$ de sorte que la décroissance de $L_{ji}(t)$ l'entraînerait vers la valeur $\frac{L_k - L_{k-1}}{2}$ (Eriksson et al., 2003). De cette façon, les synapses auraient tendance à conserver le niveau d'activation acquis et l'apprentissage deviendrait possible. Il a été observé qu'une telle règle était capable de maintenir des souvenirs en présence de bruit (Fusi, 2001). Dans le cas qui nous occupe ici, nous avons arbitrairement décidé de fixer $\Delta L_k = L_k - L_{k-1} = 20$ pour tous les niveaux attracteurs $[A_k]$. En plus de la transition proposée précédemment, la modification de ΔL_k en fonction du temps passé dans un attracteur particulier $[A_k]$, par exemple, pourrait introduire une métaplasticité dans notre modèle.

Il a été rapporté récemment que dans des cultures d'hippocampe et des tranches d'hippocampe, les enregistrements électrophysiologiques ont montré que le poids des synapses GABAergiques pouvait être modifié de façon persistente par la décharge répétée de la cellule postsynaptique dans les 20 ms qui précédent ou suivent l'activation de la synapse (Woodin et al., 2003). La détection et la modification de ces synapses par la coïncidence des potentiels d'action pré– et postsynaptiques permettent de moduler le niveau d'inhibition en fonction du moment des décharges. Une telle modification des connexions inhibitrices-excitatrices et inhibitrices-inhibitrices n'a pas été prise en compte dans notre modèle simplifié. Nous avons observé, au commencement des simulations, des périodes au cours desquelles, selon les conditions choisies, l'activité du réseau était saturée alors que le réseau était transitoirement composé d'un grand nombre de synapses au niveau d'activation le plus élevé (observez la figure 4.4a autour du temps de la première stimulation). Ceci suggère la nécessité d'une plus forte inhibition pendant cette période pour éviter autant de bouffées d'activité qui ne sont pas réalistes. Au cours de la simulation, alors que les connexions excitatrices-excitatrices sont éliminées du réseau, la force des connexions inhibitrices devrait probablement être adaptée afin de maintenir une balance dynamique entre excitation et inhibition.

Effet de la taille du réseau

Le choix d'une topologie en grille bidimensionnelle nous a permis d'étudier l'effet de l'augmentation de la taille du réseau de 10×10 à 100×100 unités. Il est intéressant d'observer que le nombre de synapses actives à la fin de la simulation est inférieur à 10% des synapses initialement présentes. Ce nombre ne varie que marginalement en fonction de la taille du réseau. L'effet des différentes initialisations du tirage de nombres pseudo-aléatoires est également limité. Nous avons observé que le nombre de synapses actives caractérisées par le niveau d'activation maximal peut transitoirement atteindre 50% du nombre de synapses initiales.

La mise en évidence d'un facteur d'échelle permettant d'ajuster l'activité du réseau en fonction de sa taille indique que de très grands réseaux ne seraient pas nécessaires pour que des circuits récurrents émergent. Ce résultat est en accord avec plusieurs découvertes liées à la théorie des similarités dynamiques (MacGregor et al., 1995).

Des «modules» interconnectés de 50×50 ou 60×60 unités immergés dans un réseau plus grand pourraient être plus efficaces pour recruter des synapses actives qui entreraient en compétition pour produire un potentiel d'action postsynaptique. La question de savoir si la capacité de grandes assemblées de neurones à «calculer» des tâches pouvait être en partie une conséquence de l'interaction collective d'un grand nombre de neurones très simples (Hopfield, 1982) a été posée. L'observation que la plus grande partie du cortex cérébral est composée de circuits locaux aux fonctions bien définies – les colonnes corticales (Lorente de No, 1949; Mountcastle, 1957; Douglas and Martin, 1991) et les hypercolonnes (Sur et al., 1980) – suggère que le pont entre les circuits simples et les propriétés complexes du cerveau serait dû à l'émergence spontanée de nouvelles capacités computationnelles du comportement collectif d'un grand nombre d'éléments simples de calcul.

Emergence de circuits

Un biais dans l'orientation géométrique des synapses produirait des effets importants sur la dynamique globale, tout comme il pourrait introduire des singularités dans la topologie du réseau. Ces singularités pourraient permettre la formation d'attracteurs présentant une balance altérée entre excitation et inhibition si elles étaient la conséquence d'activité afférente associée à un stimulus particulier. Si de tels attracteurs apparaissaient à cause uniquement de l'activité spontanée, ils pourraient masquer les propriétés liées à la nature des afférences. Nous avons observé que le renforcement de quelques synapses se faisait sans distorsion géométrique, ni en fonction de la direction, ni en fonction de la distance entre les unités pré- et postsynaptiques à la surface de la grille bidimensionnelle. L'épissage synaptique apparaît comme un processus homogène et isotrope dans tout le réseau. En présence d'activité spontanée aléatoire, l'implémentation de notre règle STDP est équivalente à un épissage aléatoire et n'introduit pas de biais.

Nous avons observé qu'en présence d'activité spontanée, un motif de stimulation spatiotemporel récurrent pouvait induire l'émergence d'assemblées de cellules orientées lorsqu'il était associé à un épissage entraîné par une règle STDP. Le processus d'épissage non supervisé, associé à des motifs de stimulation courts et stables, tendrait à organiser les unités en assemblées fortement interconnectées selon le modèle *feed-forward* à la suite des unités stimulées. Cependant, l'émergence de projections divergentes est plus difficile à observer que celle des projections convergentes.

La détection de motifs d'activité spatiotemporelle complexes dans l'activité des unités simulées suggère que des topologies en couches peuvent apparaître pendant le processus d'épissage. Les *synfire chains* (Abeles, 1991) sont des chaînes divergentes / convergentes de neurones se déchargeant de manière synchrone pour soutenir la propagation de l'information au travers d'un réseau neuronal feed-forward. Ce modèle est très efficace pour expliquer la transmission d'information temporelle précise dans les réseaux de neurones, mais les mécanismes qui soutiennent l'apparition de telles chaînes dans le cerveau mature n'ont jamais été profondément recherchés. Certains travaux ont utilisé des règles d'apprentissage *hebbiennes* (Bienenstock, 1995; Hertz and Prugel-Bennett, 1996a,b) ou STDP (Levy et al., 2001; Kitano et al., 2002) pour faire ressortir des structures semblables aux synfire chains dans des réseaux de taille relativement petite, générés aléatoirement. Les assemblées résultantes étaient composées de quelques (3-4) groupes de neurones se déchargeant en synchronie et en boucle, mais la distribution topologique des connexions n'a pas été discutée.

L'organisation non supervisée de neurones à décharges (*spiking neurons*) en assemblées a été récemment décrite dans une étude présentant des réseaux simulés de grande dimension, connectés par des projections qui s'adaptent selon une règle STDP (Izhikevich et al., 2004). 80'000 unités excitatrices et 20'000 unités inhibitrices (à comparer avec nos tailles de réseau de 100×100 et leurs distributions des connexions) ont été connectées sur une surface sphérique par 8,5 millions de connexions (sur nos grilles de 100×100, le nombre total de connexions était initialement de plus de 3 millions) en utilisant une règle de connexion locale, à laquelle s'ajoute une petite surface connectée à longue distance par les unités excitatrices. La structure spatiotemporelle des motifs émergeants de décharges a mis en évidence, lorsque les temps de conduction axonaux et une règle STDP sont incorporés au modèle, l'organisation spontanée, en présence de bruit, des neurones du réseau en assemblées, même en absence d'afférences corrélées. Des motifs de décharges récurrents ont également été observés entre les unités topologiquement proches les unes des autres. L'approche de ces auteurs (Izhikevich et al., 2004) et la nôtre sont similaires, en dehors du fait que leurs simulations n'avaient pas pour but l'étude de l'élagage synaptique, mais l'émergence des assemblées. Suite à la détection des motifs spatiotemporels d'activité, des listes d'unités produisant ces activités ont été déterminées et les circuits correspondants ont été reconstruits. Au cours de leurs simulations, l'émergence, la maintenance et la disparition de ces circuits ont été observées. Ces circuits étaient composés en moyenne de moins de 30 unités, et moins de 7.5% de toutes les unités faisaient partie de ces circuits.

L'étude réalisée par Izhikevich et al. (2004) met l'accent sur l'importance des délais de conduction que nous n'avons pas initialement considérés dans notre modèle. Des unités présynaptiques caractérisées par des vitesses de conduction axonale différentes peuvent provoquer une dépolarisation suffisante au niveau postsynpatique en fonction d'un certain motif temporel. Alors la décharge simultanée des unités présynaptiques ne permet pas de provoquer une décharge dans l'unité postsynaptique (Bienenstock, 1995). Au moyen d'un autre motif de décharges, les mêmes unités présynaptiques pourraient provoquer la décharge d'une autre unité postsynaptique. Nous considérons actuellement la possibilité d'ajouter à notre modèle des délais de conduction.

Il existe une tendance pour l'utilisation des méthodes liées à la théorie des graphes (voir Albert and Barabasi (2002) pour une revue) dans l'analyse des motifs de connexion neuronale (Sporns, 2002; Sporns et al., 2004). En particulier, la théorie des graphes a été appliquée avec succès aux données obtenues par imagerie fonctionnelle par résonance magnétique (fMRI) lors d'une tâche de tapotement des doigts (Chialvo, 2004), suggérant que les réseaux fonctionnels sont indépendants de l'échelle d'observation, c'est-à-dire que les nœuds fortement connexes sont connectés, en moyenne, avec d'autres nœuds fortement connexes, ce qui représente une propriété inattendue dans un système organisé hiérarchiquement. Dans l'état actuel des connaissances, la théorie des graphes a peu d'outils à offrir pour les problèmes liés à des réseaux dynamiques, orientés et pondérés comme les nôtres. Nous avons recherché un indice approprié pour estimer la qualité des circuits reconstruits comme ceux de la figure 4.8. Aucun des indices standards, comme le *cœfficient de clustering* (la fraction des connexions présentes entre les voisins d'un nœud par rapport au nombre maximal théoriquement possible) ou la longueur du trajet moyen (le nombre minimal de liens nécessaires pour connecter deux nœuds), n'était adapté à la mesure de réseaux convergents / divergents. Nous sommes encore à la recherche d'un indice approprié pour cette mesure.

Epissage synaptique

L'épissage synaptique massif qui se déroule au cours de l'enfance, après une phase de surcroissance synaptique, est une propriété intriguante du développement cérébral chez les mammifères. Peu d'études théoriques ont été menées pour déterminer l'avantage computationnel d'une stratégie de développement apparemment si dispendieuse. Dans un premier travail théorique, Chechik et al. (1998) ont suggéré que les performances d'apprentissage d'un réseau atteignaient leur niveau optimal si, sous des contraintes métaboliques limitant leur nombre et leurs forces, les synapses étaient d'abord surnuméraires avant d'être épissées. Deux «*organismes adultes*» disposant des mêmes ressources synaptiques peuvent stocker un nombre différent de souvenirs selon la manière dont ils ont acquis leurs densités synaptiques finales. Un organisme doté d'un excès de synapses soumis à un épissage synaptique sélectif pourrait donc stocker davantage de souvenirs qu'un autre adulte dont la densité synaptique a été fixée pendant l'enfance. Les simulations ont été réalisées en utilisant une règle de plasticité synaptique hebbienne, et une fonction d'épissage qui éliminait les connexions les plus faibles du réseau. Il a été démontré qu'un tel algorithme pouvait maintenir les performances du réseau.

La régulation neuronale est un mécanisme identifié expérimentalement qui régule la force des afférences synaptiques pour maintenir l'homéostase de la membrane du neurone postsynaptique (Turrigiano et al., 1998). Horn et al. (1998) ont suggéré que la régulation neuronale pourrait maintenir, en théorie, les performances d'apprentissage des réseaux subissant un épissage synaptique sélectif. Dans l'article de Chechik et al. (1999), les auteurs discutent du rôle de la régulation neuronale dans le maintien des projections postsynaptiques, en retirant les synapses les plus faibles et en modifiant les autres synapses en conséquence (voir Mimura et al. (2003) pour une discussion analytique de ces résultats). Il a été également démontré que STDP maintenait le champ des projections postsynaptiques (Abbott and Nelson, 2000; Song and Abbott, 2001). Pour cette raison, STDP serait une règle de modification de la force synaptique appropriée à la simulation de l'épissage synaptique.

Le darwinisme neuronal – également nommé théorie de la sélection des groupes neuronaux – est une théorie de populations du système nerveux, qui a pour but la compréhension de la signification de la variation et de la sélection dans le développement du cerveau (voir Edelman (1993) pour une revue). Selon cette théorie, le monde se caractérise perceptiblement, pour un organisme, par la conséquence de deux processus interactifs de sélection après variation. Le premier processus apparaît pendant les phases de développement embryonnaire et postnatal, lorsque les neurones adjacents tendent à être fortement interconnectés en «assemblées» de taille et de structure variables dénommés «groupes neuronaux». Le second processus consiste en l'altération des forces synaptiques suivant l'activité de l'individu, et la sélection des groupes neuronaux adaptatifs sur la base de la corrélation de leurs réponses. Dans l'Edelmanisme Neuronal (voir Crick (1989) pour une critique de la théorie), l'emphase est placée sur la «compétition» entre les groupes neuronaux sans référence au mécanisme d'épissage synaptique. Nous devons reconnaître que la théorie de l'Edelmanisme Neuronal produit des résulats qui mettent à l'épreuve nos propres résultats.

Effet des taux de décharges

Les résultats présentés ici suggèrent que des topologies en couches, compatibles avec des synfire chains, peuvent apparaître au cours d'un processus d'épissage synaptique non supervisé. Il n'a pas été possible de déterminer les conditions exactes permettant au mécanisme d'épissage synaptique de conduire à l'émergence de chaînes convergentes / divergentes balancées, c'est-à-dire présentant des k_{in} et k_{out} comparables.

Les unités fortement interconnectées (*SI*-units, voir section 4.2, p.34) présentaient des taux de décharges situés entre 30 et 40 décharges par seconde. Bien que cette valeur soit trop grande pour être biologiquement plausible dans le contexte d'un modèle de cortex cérébral, nous avons observé une corrélation négative entre le taux de décharge moyen et le k_{out}. Cette corrélation est inhérente à la règle de modification STDP standard. Comme représenté dans la figure 5.2a,b, une unité se déchargeant plus rapidement que les autres aura tendance à maintenir toutes ses entrées et à perdre toutes ses sorties. En conséquence, nous pouvons suggérer qu'à cause des taux de décharges excessivement élevés, STDP tend à être trop affecté par les différences de taux de décharges.

En tenant compte de cette observation concernant la dynamique de STDP, un scénario peut être proposé pour expliquer l'évolution des indices de connectivité k_{in} et k_{out}, comme présenté dans la figure 4.6. Il est possible que pendant les quelques premiers pas de la simulation, les unités excitatrices qui, par hasard, ont été dotées d'un grand nombre d'unités présynaptiques excitatrices commencent à se décharger avec une fréquence élevée, sous la seule action de leurs nombreuses entrées. La figure 5.2b suggère que ces unités renforceraient leurs entrées, augmentant d'autant leur taux de décharge. Parallèlement, la figure 5.2a suggère que les unités présentant un taux de décharge élevé ont tendance à perdre toutes leurs projections, ce qui correspond à la dynamique d'élagage observée pour presque toutes les *SI-units*. Ce scénario propose une explication possible à l'excès de convergence par rapport à la divergence observé dans les réseaux émergeants de notre travail.

Il a été observé sur des préparations de tranches de cortex cérébral visuel de rats (Froemke and Dan, 2002), que la contribution de chaque paire de décharges pré-/postsynaptiques à la modification synaptique dépendait non seulement de l'intervalle entre les paires, mais également du moment de la décharge précédente. L'efficacité de chaque décharge sur la modification synaptique était supprimée par la décharge précédente du même neurone intervenant dans un intervalle de plusieurs dizaines de millisecondes. Les auteurs suggèrent que le moment de la première décharge de chaque bouffée domine la modification synaptique, alors que les décharges additionnelles n'offrent qu'une contribution marginale. Sur la base de cette observation, un neurone à décharge présentant une règle de modification du poids synaptique inspirée de STDP pour la première décharge uniquement a été proposé dans un article théorique récent (Guyonneau et al., 2005). L'unité qui décharge recevait une vague provenant de 1'000 unités présynaptiques et apprenait à réagir plus rapidement à un motif afférent de décharges qui se répétait. La latence de la décharge postsynaptique tendait à se stabiliser autour d'une valeur minimale alors que les premières synapses étaient complètement renforcées et les suivantes complètement déprimées (reproduisant la distribution bimodale des efficacités), impliquant que la règle STDP donnait plus d'importance aux afférences rapides et répétées et ignorait les autres.

La figure 5.2c,d montre de quelle manière, en ne considérant que les premières paires de décharges, la proportion entre convergence et divergence pourrait être modifiée d'une manière balancée. Avec une telle règle STDP modifiée, l'impact des unités se déchargeant rapidement sur la perte des connexions sortantes pourrait être limité et pourrait faire disparaître complètement le problème lié aux unités à fort taux de décharge. Si nous considérons la possibilité de simuler des réseaux de tailles plus grandes, alors l'introduction d'une telle modification dans la règle STDP semble nécessaire.

Synfire chains

Le rôle des neurones inhibiteurs dans la stabilisation d'un réseau pourvu de synfire chains a été récemment proposé. Dans un travail théorique, Aviel et al. (2004) suggèrent la nécessité d'une inhibition dans un réseau balancé – un réseau où chaque unité reçoit un nombre égal d'excitations et d'inhibitions – pour éviter l'allumage spontané des synfire chains et pour éviter que l'activité du reste du réseau ne soit saturée par la décharge synchrone des couches des synfire chains figure 2.3. La balance de l'inhibition serait obtenue à chaque niveau des synfire chains par la connexion à un groupe «fantôme» d'unités inhibitrices (dénommé *shadow pool*), qui reçoivent les entrées convergentes de la couche précédente comme si elles faisaient partie de la couche suivante. De plus, ces unités inhibitrices ne se projettent pas de façon divergente sur la couche suivante mais localement à l'intérieur du réseau comme des interneurones ordinaires. La nécessité d'un tel mécanisme d'inhibition pour la propagation d'une vague d'activité au travers de la synfire chain pose la question du rôle des unités inhibitrices dans nos simulations. Nous avons observé une activité balancée dans notre réseau, ainsi que l'apparition d'une inhibition à la fin des stimuli, qui est dépendante de l'intensité de ceux-ci (comparez les figures 4.13 et 4.14). Ceci suggère que nous pourrions considérer des unités inhibitrices dans les circuits émergés.

L'analyse des corrélogrammes est souvent considérée comme un outil de grande valeur pour la déduction de la connectivité fonctionnelle (Abeles, 1982a). La figure 4.14b montre un pic asymétrique près du temps $t = 0$ qui pourrait être interprété comme une corrélation temporelle entre les unités 1234 et 7794, qui suggère une projection directe de l'unité 1234 sur l'unité 7794. Toutefois, cette projection n'existe pas dans la topologie du circuit (voir figure 4.8b). La courbe de corrélation pourrait être expliquée par un retard systématique dans les temps de décharge de l'unité 7794 par rapport à l'unité 1234. Une telle variabilité est néanmoins consistante avec l'activité synfire (Gewaltig et al., 2001). Dans tous les cas, des recherches supplémentaires sont requises pour déterminer si une activité synfire soutenue (Tetzlaff et al., 2004) pourrait apparaître dans de tels circuits inclus dans un plus vaste réseau. La caractérisation de l'état du réseau proposé par Brunel (2000) pourrait être utile pour cette analyse, mais les taux de décharge des unités excitatrices ne devraient pas dépasser les niveaux incompatibles avec STDP.

Contents

List of Figures

The First Law of Explanation
When you're explaining something to somebody
and they don't get it, that's not their problem,
it's your problem.

– Tim Bray

Chapter 1

Introduction

Résumé Ce Chapitre est une introduction au présent document. Son organisation générale s'articule autour de deux parties principales. La première (p.5) regroupe les informations neuroscientifiques liées à notre travail, ainsi que les résultats et la discussion des simulations. La seconde (p.67) présente les réalisations logicielles et matérielles.

The present work results from the collaboration between a neuroscience laboratory – INSERM U318, University of Grenoble 1, France – and a computer science department – INFORGE, University of Lausanne, Switzerland. Most of the work has been produced at the Swiss location with continuous input from the French side. The document was produced according to (partly incompatible) requirements formulated by both universities.

For the sake of readability, and despite the fact that they were undertaken synchronously, the computational and neuroscientific aspects of our investigation are discussed in two distinct parts. We will begin at page 5 with the neuroscientific part. An introduction to the neurobiological structures and concepts underpinning our research is presented in Chapter 2, p.5. In Chapter 3, p.15, the network, the leaky integrate-and-fire neuromime, and the synaptic models are treated with analytic details. The simulation results are analysed in Chapter 4, p.29 before being discussed in the Chapter 5, p.51.

Starting at page 67, the computational aspects are presented. The different pieces of software developed directly for – or in close relation with – this work are described in Chapter 6, p.67. An account is provided on the hardware (Chapter 7, p.103) that sustained the computational load for our simulations. The novel platform on which our model is expected to evolve in the future is also presented. A final conclusion (Chapter 8, p.113) closes the second part.

The electronic version of the document features hypertext links. Clicking on the section references or on the bibliographic citations will pop the appropriate page up. Clicking on the URLs will point your browser to the corresponding web page. Check the Appendix A, p.115 for details on how to get the electronic document.

Part I

Neuro(Informatics)

My brain?
It's my second favorite organ.

– Woody Allen

Chapter 2

Neurobiological perspective

Résumé Ce Chapitre introduit de façon succincte les structures biologiques et les concepts théoriques que nous avons cherché à modéliser. Certains aspects développementaux et fonctionnels du système nerveux central des mammifères sont abordés. Les notions de plasticité synaptique à modulation temporelle relative (STDP), d'épissage des synapses et de *synfire chains* sont présentées ici.

This Chapter presents the biological structures and the established theoretical concepts underpinning our modeling effort. We will discuss some aspects of the developmental process leading to the mature brain. The functional aspects of the cortical circuits will be presented, in particular the notions of spike-timing-dependent synaptic plasticity (STDP), synaptic pruning and synfire chains.

2.1 Brain development

2.1.1 Differentiation

The entire mammalian central nervous system is derived from the walls of the neural tube that is formed at an early stage of embryological development. The structures of the brain become more elaborate through the differentiation of the neural tube into three vesicles. The vesicles will further differentiate into the forebrain, the midbrain and the hindbrain, while the rest of the neural tube will differentiate into the spinal cord. We will focus on the development of the forebrain, that gives rise – among other structures – to the optic vesicles and the telencephalon. The telencephalon consists of the two cerebral hemispheres, the walls of which are the site of the proliferation and differentiation of the precursor cells into the neurons of the cerebral cortex.

The cellular development of the mammalian cerebral cortex follows a consistent pattern, with large pyramidal neurons of the lower layers taking laminar positions and differentiating earlier than neurons intended to be situated more superficially (Rakic, 1974). Cortical neurons originate in proliferative zones close to the ventricular surface and migrate to the cortical plate only after the final division of the precursor cells. Cells destined for the deep cortical positions are generated first, and the more superficial ones at progressively later times.

In the mature brain, the cerebral cortex appears as a layered structure (layers I-VI, Bear et al. (1996)) characterized by the changes in the density of cells and neuropil morphology. The thickness and definition of each layer varies from area to area of the cortex (Brodmann, 1909), but the layered structure is generally maintained (Douglas and Martin, 1991). The human cerebral cortex is a highly folded sheet of neurons at a density of *circa* 10^5 neurons per mm^2. Its thickness varies between 1 and 4.5mm, with an overall average of approximately 2.5mm (von Economo, 1929; Zilles, 1990). One half of these cells are pyramidal cells which are characterized by the distal connection of their axon (Abeles, 1991). The physiological ratio is about one inhibitory neuron for four excitatory neurons (Braitenberg and Schuez, 1998).

2.1.2 Synaptogenesis

There is experimental evidence that the cerebral cortex develops as a whole rather than regionally, as synaptogenesis proceeds concurrently in all cortical areas and layers. Simultaneous overproduction of a critical mass of synapses in each cortical area may be essential for their parallel emergence through competitive interactions between extrinsic afferent projections. Such competition has been observed between the projections of the two eyes during the formation of visual centers (Hubel et al., 1977; Rakic, 1981).

Genetic programs are assumed to drive the primordial pattern of neuronal connectivity through the actions of a limited set of trophic factors and guidance cues, initially forming excessive branches and synapses, distributed somewhat diffusely (Innocenti, 1995). Then, refinement processes act to correct initial inaccuracies by pruning inappropriate connections while preserving appropriate ones. The embryonic nervous system is refined over the course of development as a result of the twin processes of cell death and selective axon pruning. Apoptosis – genetically programmed cell death – and necrosis – pathologic or accidental cell death due to irreversible damage – are two rough mechanisms for refining embryonic connections. However, the creation of complex connectivity patterns often requires the pruning of only a selected subset of the connections initially established by a neuron.

It is generally agreed that changes in cortical function are associated with corresponding alterations in the density and arrangement of synaptic circuits. Pruning events at the neuromuscular junction – where the motor neuron synapses with the targeted muscle cell – may provide mechanistic insights into how this type of segregation could occur (Lichtman and Colman, 2000). Individual muscle fibers are initially contacted by many motoneurons, resulting in a highly overlapping pattern of innervation. Eventually, as a result of synaptic competition between asynchronous inputs, synapse pruning followed by branch pruning occurs, and the innervation pattern becomes strictly segregated so that no two motoneurons maintain a junction onto the same muscle fiber.

Quantitative analyses of synaptogenesis in the rat (Aghajanian and Bloom, 1967), the Rhesus monkey (Bourgeois and Rakic, 1993), and human (Huttenlocher, 1979) cortex have suggested a transient phase of high density of synapses during infancy. The rapid rate of synaptogenesis begins a few weeks after the end of neurogenesis and completion of neuronal migration. The density of synapses continues to increase during infancy and remains above adult levels. After a relatively short period of stable synaptic density, a pruning process begins: synapses are constantly removed, yielding a marked decrease in synaptic density. This process continues until puberty, when synaptic density stabilizes at adult levels which are maintained until old age. For the human brain, the peak level of synaptic density in childhood is 150 to 200% compared to adult levels, depending on the brain region. The changes in synaptic density are not the result of changes in total brain volume, but reflect true synaptic pruning. If experience alters synaptic density during development, it does so by causing selective survival of certain synapses, and not by regulating their initial formation.

The cerebral cortex has come to be associated with a remarkable capacity for functional and anatomical plasticity during pre– and postnatal development periods. The phenomenon of synaptic over-growth and pruning was found in humans (Huttenlocher, 1979), as well as in other mammals such as monkeys (Bourgeois and Rakic, 1993) and cats (Innocenti, 1995). It was observed through widespread brain regions including cortical areas (Bourgeois and Rakic, 1993;

Huttenlocher et al., 1982) and the projection fibers between hemispheres (Innocenti, 1995). For example, during infancy to adolescence of Macaque monkeys, an average of 5,000 synapses per second are lost in the striate cortex – the primary visual area – of both hemispheres (provided that they are being pruned equally over 24 hours cycle).

Adult patterns of neuronal connectivity develop from a transient embryonic template characterized by exuberant projections to both appropriate and inappropriate target regions. However, behavioral competence continues to increase beyond the stage of excess synapses. Pruning may also play a role in establishing topographic maps, as it can be seen in the retinotectal system (Nakamura and O'Leary, 1989). This suggests that full functional maturation may be related to synapse pruning and acquisition of synaptic efficiency at the molecular level. While there have been huge advances made in identifying the cellular and molecular events involved in initial axon guidance events, there has been much less progress in identifying the biological mechanisms involved in pruning.

2.1.3 Neurogenesis

There is increasing evidence that neurogenesis – the birth of new neurons – occurs in at least the olfactory bulb and the hippocampal dentate gyrus of the adult mammalian brain (see Emsley et al. (2005) for a review), against the dogmatic view of a static mature brain. In these regions, some molecular and cellular events required for neuronal development observed in the embryonic brain may appear at the adult stage under non-pathological conditions. Endogenous precursor cells have the ability to migrate to selected brain regions, differentiate into neurons, and functionally integrate the adult brain circuitry, developing mature electrophysiological activity. Newborn neurons are morphologically identical to surrounding neurons. Hippocampal neurogenesis can be modulated by physiological and behavioral extreme events such as stress, seizure, learning and exercise throughout adulthood, but declines with age. Despite these recent results, the function of neurogenesis in adult brains is still under investigation as it seems to represent a marginal phenomenon with respect to the massive cell death naturally occurring as a function of age. We did not include any neurogenesis-like process into our current model, but we are considering the opportunity to integrate it, at least partially, in terms of synaptogenesis.

2.2 Cortical micro-circuits

During the embryonic and postnatal development, adjacent neurons tend to be strongly interconnected in collectives of variable size and structure. In regions of the central nervous system where specific roles can be assigned to neurons, local mosaic arrangements that provide a natural basis for a functional arrangement are observed. These include ocular dominance columns (Rakic, 1981), blobs (Purves and LaMantia, 1990), and barrels (Rice and Van Der Loos, 1977).

Anatomical results provide quantitative local connectivity rules in the mature cortex, like in the rat visual cortex (Douglas and Martin, 1991; Hellwig, 2000). The probability for two pyramidal neurons to share a synapse decreases with distance in a Gaussian fashion. This rule ignores the axonal patches (Amir et al., 1993), extensive horizontal axons from pyramidal neurons in superficial layers that provide a substrate for lateral interactions across cortical columns. These connections are believed to link functionally similar regions (Yabuta and Callaway, 1998).

2.2.1 Synaptic efficacy

Synapses can change their strength in response to the activity of both pre– and postsynaptic cells (see next section). This property is assumed to be associated with learning, synapse formation and pruning. Alterations in the synaptic transmission can be roughly subdivided into two classes of mechanisms: long-term potentiation (LTP), and long-term depression (LTD). LTP is measured as a persistent increase in the amplitude of the excitatory postsynaptic potentials (EPSP), whereas LTD is measured as a persistent decrease in the amplitude of the EPSPs.

Recent works (see Montgomery and Madison (2004) for a review) suggest that the strength of the synapses may vary between discrete mechanistic states, rather than by adjusting their efficacy along a continuum. This type of synaptic plasticity has primarily been studied in the excitatory glutamatergic synapses of the brain, particularly in the hippocampus. Glutamatergic synapses use the glutamate released in the active zones of the presynaptic membrane for transduction into postsynaptic cell membrane depolarization. Among the glutamate receptors, AMPA and NMDA receptor subtypes play predominant roles in the excitatory synaptic transmission and plasticity (Bear et al., 1996). Experimental observations have led to the proposal that the AMPA receptor subtype mediates ion fluxes across the membrane during synaptic transmission, whereas

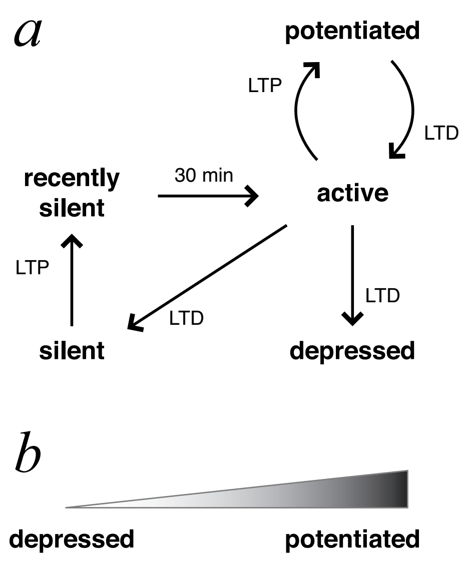


Figure 2.1: Schematic comparison of the state and the continuous synaptic models. ***(a):*** transitions between the synaptic states described in main text (modified from Montgomery and Madison (2002)); ***(b):*** continuous representation between depressed and potentiated synapses.

NMDA receptors primarily play a role in inducing or modulating synaptic plasticity of the AMPA-receptor-mediated transmission. We recapitulate here the five synaptic states that have been suggested (Montgomery and Madison, 2002) without entering into the molecular details (see figure 2.1 for an overview):

active state: Both AMPA- and NMDA-receptor-mediated responses are present.

potentiated state: Active synapses undergoing LTP enter this state. It is related to the active state, except for a different LTD molecular mechanism.

depressed state: Active synapses undergoing LTD enter this state. It is currently ill-defined, and it might differ little from the active state.

silent state: Synapses in this state are characterized by the lack of synaptic response at normal postsynaptic membrane potentials due to the absence of AMPA receptors in the postsynaptic membrane. NMDA receptors are present in the postsynaptic membrane, but they are subject to voltage-dependent Mg^{2+} blocking. Synapses in this state can be potentiated in the same way as active synapses, though.

recently silent state: Silent synapses undergoing LTP enter this state. It differs from the active state in that synapses cannot undergo LTD. The transition to the active state takes about one half hour after leaving the silent state.

The regulation of the dynamic transport (by *endocytosis* and *exocytosis*) of AMPA receptors into and out of the synaptic membrane (Malinow and Malenka, 2002) has been proposed as the mechanism behind discrete synapse state transitions. In most cases, the insertion and removal of AMPA receptors on the postsynaptic membrane is triggered by Ca^{2+} influx through the NMDA receptors. According to the discrete state model, previous studies may not have revealed the existence of the discrete states because they recorded activity in large populations of synapses, averaging out the properties of individual synapses (Montgomery and Madison, 2002). The synaptic heterogeneity is preserved because no single activity protocol can alter all synapses in the same way. In the continuous synaptic model, plasticity is coded only in the current strength of the synapse. The discrete states synapse model features an historical aspect that is absent from the continuous model. Depending on their previous state, synapses have different abilities to express synaptic plasticity, and use differing mechanisms to achieve it.

2.2.2 Spike-timing-dependent synaptic plasticity

Donald Hebb was the first to suggest a precise rule that might govern the synaptic changes (Hebb, 1949). He proposed that the efficiency of a connection from a pre– to a postsynaptic neuron is increased if the presynaptic neuron repeatedly or persistently contributes to firing the postsynaptic neuron. His hypothesis emphasized the role of causality between the pre– and postsynaptic spikes, but did not provide a rule for the decreasing of the synapse efficiency, nor did he address the issue of the effective time window.

Recent experiments suggest that both potentiation and depression obey the timing of pre– and postsynaptic spikes. Spike-timing-dependent synaptic plasticity (STDP) is a mechanism to explain the synaptic strength modification based on such spiking order, first observed by Bell et al. (1997). Different correlations have been observed in various preparations (see Roberts and Bell (2002) for a review). Some of them (figure 2.2a-c) are consistent with Hebb's proposal that pre– before postsynaptic spikes should increase the efficiency of the projection ("Hebbian" rules), but the reverse relation holds for others ("anti-Hebbian" rules, figure 2.2d,e).

The Antisymmetric Hebbian rule depicted in figure 2.2a has been observed in the mammalian cortex (Markram et al., 1997), and in cultured hippocampal neurons (Bi and Poo, 1998). It has been proposed to explain the origin of long-term potentiation (LTP), i.e. a mechanism for

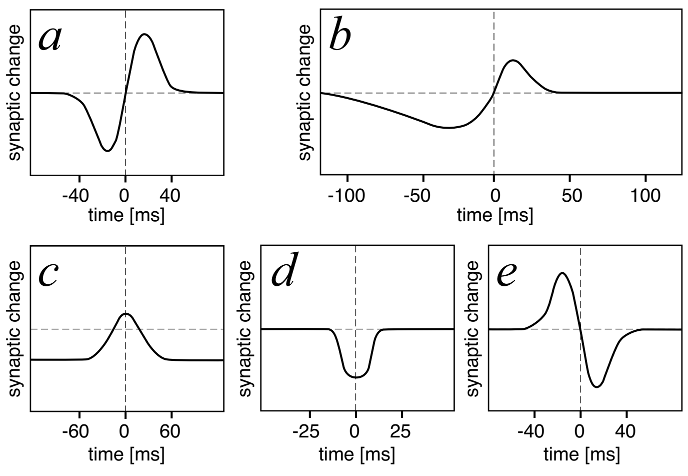


Figure 2.2: STDP rules where time indicates the interspike interval. Positive times correspond to the postsynaptic spike following the presynaptic spike; negative times correspond to the presynaptic spike following the postsynaptic spike. ***(a):*** antisymmetric Hebbian rule; ***(b):*** antisymmetric Hebbian rule with differential dynamics; ***(c):*** symmetric Hebbian rule; ***(d):*** symmetric anti-Hebbian rule; ***(e):*** asymmetric anti-Hebbian rule. Modified from Roberts and Bell (2002).

reinforcement of synapses repeatedly activated shortly before the occurrence of a postsynaptic spike (Kelso et al., 1986). It has also been proposed to explain long-term depression (LTD), which corresponds to the weakening of synapses strength whenever the presynaptic cell is repeatedly activated shortly after the occurrence of a postsynaptic spike (Karmarkar and Buonomano, 2002). This is the rule we used for our model.

The glutamatergic NMDA receptors were initially identified as the receptor site with all biological features compatible with LTP induced by coincident pre– and postsynaptic cell discharges (Wigstrom and Gustafsson, 1986). The involvement of NMDA receptors in timing-dependent long-term depression (tLTD) has been described (Sjostrom et al., 2003). Other investigations suggest that glutamatergic receptors with AMPA channels and GABAergic receptors may also undergo modifications of the corresponding postsynaptic potentials as a function of the timing of pre– and postsynaptic activities (Engel et al., 2001; Woodin et al., 2003). These studies suggest that several mechanisms, mediated by several neurotransmitters, may exist at the synaptic level for changing the postsynaptic potential, either excitatory or inhibitory, as a function of the relative timing of pre– and postsynaptic spikes.

The important consequences from changes in synaptic strength may produce for information transmission, and subsequently for synaptic pruning, have raised an interest to simulate the activity of neural networks with embedded synapses characterized by STDP (Lumer et al., 1997; Fusi et al., 2000; Hopfield and Brody, 2004).

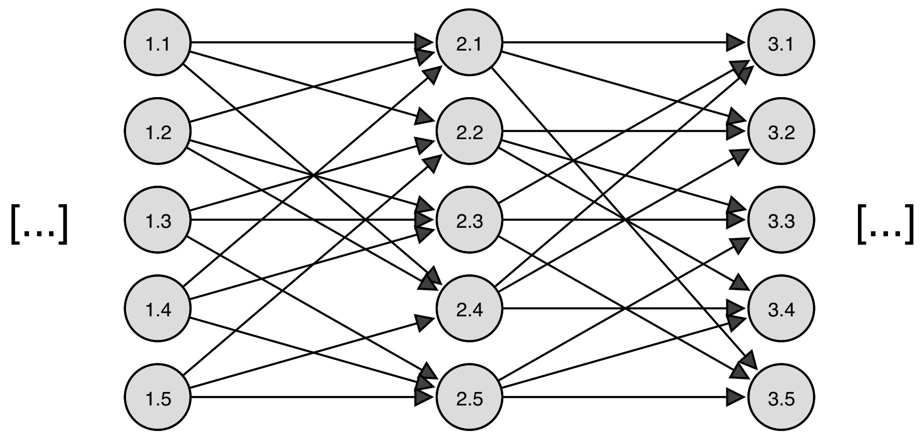


Figure 2.3: A schematic synfire chain. A synfire chain is characterized by its width (w), defined as the number of neurons in each pool of the chain, and its multiplicity (m), defined as the number of projections from a neuron in pool n to a neuron in pool $n+1$. The synfire chain is said to be incomplete if $m < w$, like in this example: $w = 5$ and $m = 3$. Note that individual neurons can appear in multiple pools of the same or different synfire chains.

2.2.3 Synfire chains

This section makes the connection between the biological and the modeling aspects of this research. *Synfire chains* are theoretical models (Abeles, 1991) for the transmission and processing of precisely timed information through the cerebral cortex. They suggest how precise timing can be sustained by means of pools of neurons linked together in a feed-forward chain. Waves of activity can propagate from pool to pool through convergent/divergent connections. It has been postulated that such a wave corresponds to an elementary cognitive event (Bienenstock, 1995). A prediction from this model is that simultaneous recording of activity of cells belonging to the same assembly and involved repeatedly in the same process should be able to reveal repeated occurrences of spatiotemporal firing patterns (Villa, 2000). Figure 2.3 is a convenient representation of a synfire chain section from which the topological distribution of the pools in the network has been removed, leaving only the logical chain. One has to imagine that neurons in the same pool are not necessarily located in an immediate neighbourhood.

The memory capacity of networks embedding synfire chains has been theoretically studied (Herrmann et al., 1995; Aviel et al., 2005). Many spatiotemporal patterns can be stored in a network. As each neuron can participate several times in the same synfire chain, or in several chains, it introduces a crosstalk noise that limits the network capacity. Among other tasks, synfire chains have been successfully applied to recognition of patterns (Arnoldi et al., 1999) and spatiotemporal sequences of spikes (Jin, 2004), as well as the modeling of compositionality[1] (Abeles et al., 2004; Hayon et al., 2005). Most studies have emphasized, analytically or numerically, the robustness, against noise, of the synchronous volleys propagation through isolated pre-wired synfire chains (Diesmann et al., 1999; Gewaltig et al., 2001; Tetzlaff et al., 2002; Yazdanbakhsh et al., 2002), or pre-wired synfire chains embedded in large networks (Mehring et al., 2003; Aviel et al., 2003; Tetzlaff et al., 2004).

If synfire chains are found in real brains, the successive synaptic connections between the pools must develop through some form of unsupervised process. The mechanisms that may underlie the appearance of synfire chains in the mature brain have received less attention. Some works have used the Hebbian learning rule (Bienenstock, 1995; Hertz and Prugel-Bennett, 1996a,b), or STDP (Levy et al., 2001; Kitano et al., 2002) to let synfire-like structures emerge out of relatively small randomly generated networks. The resulting assemblies were composed of a few (3-4) groups of neurons firing synchronously in loops, but the topological distribution of the connections was not discussed.

[1] The *principle of compositionality* states that the meaning of a complex expression is determined by the meanings of its parts, and by their relations.

Fifth Young's Rule
Mistakes are human,
but to really mess things up,
you must involve a computer.
– Anonymous

Chapter 3

Modeling

Résumé Tous les aspects du modèle simulé sont abordés dans ce Chapitre. Le réseau, le neurone, le bruit, ainsi que les règles d'adaptation et de plasticité sont décrits dans le détail de leurs formulations analytiques. La construction des stimuli utilisés pour les simulations présentées dans le Chapitre 4, p.29, est également discutée.

All aspects of the simulated model are discussed in this Chapter. The network, the neuron, the noise, as well as the adaptation and pruning rules are described in the details of their analytical expressions. At the end of the Chapter, the construction of the stimuli used for simulations described in Chapter 4, p.29 is presented.

3.1 Network model

3.1.1 Layout

The network is a 2D lattice folded as a torus to limit the edge effect where the units near the boundary receive less input. The size of the network varies between 10×10 and 110×110 units. Several types of units may be defined. In this study, we define two types, $q \in \{1, 2\}$. 80% of *Type I* ($q = 1$) units and 20% of *Type II* ($q = 2$) units are uniformly distributed over the network according to a space-filling quasi-random Sobol distribution (Press et al., 1992, Fig. 7.7.1). A unit of either type may project to a unit of either type, but self-connections are not allowed.

3.1.2 Connectivity

Each unit is assumed to be at the center of a relative 2D map, with coordinates $x = 0$, $y = 0$. The probability that another unit located at coordinates (x, y) receives a projection is provided by the following density function

$$G(x,y) = \alpha_{[q]} \cdot \exp\left(\frac{-2\pi \cdot (x^2 + y^2)}{\sigma^2_{[q]}}\right) + \phi_{[q]} \tag{3.1}$$

where $\alpha_{[q]}$ is a scaling factor for maximal probability of establishing a connection with the closest neighbours, $\sigma_{[q]}$ is a scaling factor for the width of the Gaussian shaped function, and $\phi_{[q]}$ is a uniform probability (Hill and Villa, 1997). The density function defining the probability of the connections is different for each type of unit and is illustrated in figure 3.1a,e. The values of the parameters used for the density functions are indicated in table 3.1, p.19. These values approximate the connectivity distribution and spatial extents within the cortex, but not the characteristic long distance patches.

The random selection of the target units is run independently for each unit of either type. An example of the spatial distribution of the projections of one *Type I* unit, and of one *Type II* unit, is illustrated for a 100×100 network in figure 3.1b, and figure 3.1f, respectively. In this example, the *Type I* unit (figure 3.1b) projects to 233 units and the *Type II* unit (figure 3.1f) projects to 537 units overall. For each unit, it is possible to illustrate the orientation of its connections in the 2D lattice by plotting the deviation from a perfect isotropic distribution in polar coordinates. In case of an isotropic distribution, the orientations would be illustrated by a circular line around the center. If this line is not circular, it shows that some orientations have been selected preferentially by chance, as it may occur in a random selection procedure. The orientations of the projections of the two example units are illustrated in figure 3.1c,g. It appears that, at the single unit level, a large degree of anisotropy exists in the connection topology.

Figure 3.1d shows the cumulative distribution of all connections established by *Type I* units projecting to either type in a 100×100 network. The histograms modes show that on average one unit of *Type I* is projecting to 50 units of *Type II* and to 190 units of *Type I*. Figure 3.1h illustrates the cumulative distribution of all connections established by *Type II* units and shows that on average one unit of *Type II* is projecting to 115 units of *Type II* and to 460 units of

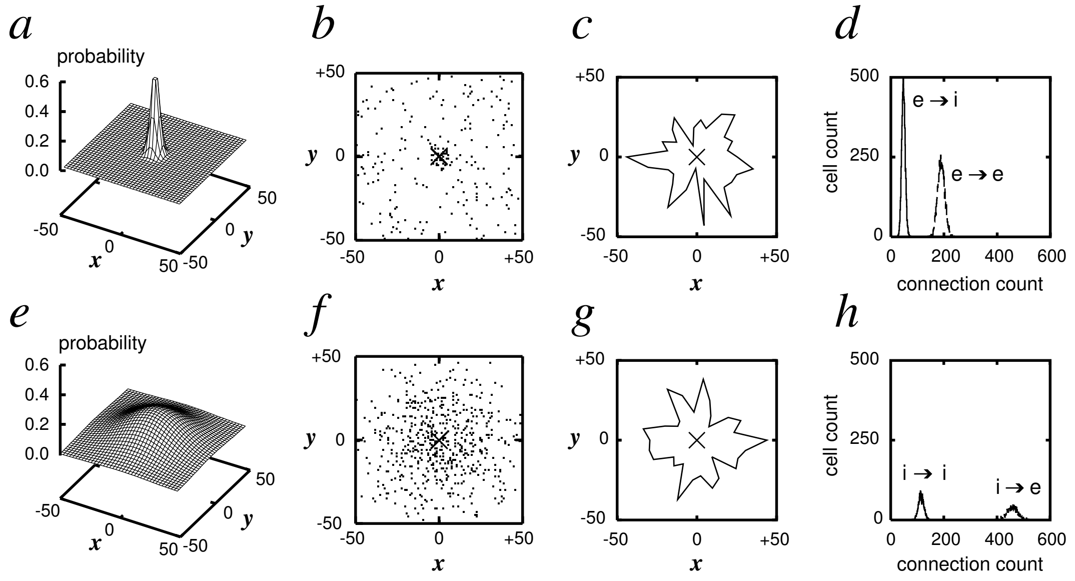


Figure 3.1: Main connectivity features for *Type I* unit (upper row) and *Type II* unit (lower row). ***(a,e):*** connectivity density function for a unit located at coordinates 0,0 on a 100 × 100 2D lattice; ***(b,f):*** example of two projecting units, one for each type, located at the center of the 2D map. Dots represent the location of a target unit connected by the projecting unit; ***(c,g):*** orientation map of the projections of the same example units with polar coordinates smoothed with a bin equal to 12°. A circular line would represent a perfect pattern of isotropic connections; ***(d,h):*** cumulative distributions of the connections. *Type I* are assumed to represent excitatory units (**e**→) and *Type II* inhibitory units (**i**→).

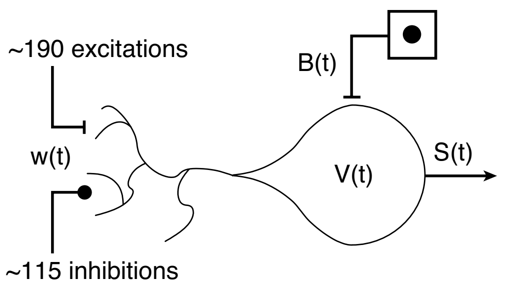


Figure 3.2: A sketch overview of the neuron model. See text for details on the membrane potential $V(t)$ and the neuron state $S(t)$ (section 3.2.1, p.18); the background activity $B(t)$ (section 3.2.2, p.19); and the postsynaptic potentials $w(t)$ functions (section 3.3, p.20). Connection numbers are mean values for a 100×100 network, before simulation begins.

Type I.

3.2 Neuromimetic model

3.2.1 Membrane potential

All units of the network are simulated by leaky integrate-and-fire neuromimes. At each time step, the value of the membrane potential of the i^{th} unit, $V_i(t)$, is calculated such that

$$\begin{aligned} V_i(t+1) &= V_{\mathrm{rest}[q]} + B_i(t) \\ &\quad +(1 - S_i(t))((V_i(t) - V_{\mathrm{rest}[q]})k_{\mathrm{mem}[q]}) \\ &\quad +\sum_j w_{ji}(t) \end{aligned} \tag{3.2}$$

where $V_{\mathrm{rest}[q]}$ corresponds to the value of the resting potential for the units of class type $[q]$, $B_i(t)$ is the background activity arriving to the i^{th} unit (see section 3.2.2, p.19), $S_i(t)$ is the state of the unit as expressed below, $k_{\mathrm{mem}[q]} = \exp(-1/\tau_{\mathrm{mem}[q]})$ is the time constant associated to the current of leakage for the units of class type $[q]$, and $w_{ji}(t)$ are the postsynaptic potentials of the j^{th} units projecting to the i^{th} unit (see section 3.3, p.20).

The state of a unit $S_i(t)$ is a function of the membrane potential $V_i(t)$ and a threshold potential $\theta_{[q]_i}$, such that $S_i(t) = \mathcal{H}(V_i(t) - \theta_{[q]_i})$. $\mathcal{H}$ is the Heaviside function, $\mathcal{H}(x) = 0 : x < 0$, $\mathcal{H}(x) = 1 : x \geq 0$. In addition, the state of the unit depends on the refractory period $t_{\mathrm{refract}[q]}$,

Variable	*Type I*	*Type II*	Short description
	0.80	0.20	proportion in network [%]
ϕ	0.02	0.00	uniform connection probability
α	0.60	0.20	Gaussian maximal probability
σ	10	75	Gaussian distribution width
P	0.84	-1.40	post synaptic potential [mV]
V_{rest}	-78	-78	membrane resting potential [mV]
θ_i	-40	-40	membrane threshold potential [mV]
t_{refract}	3	2	absolute refractory period [ms]
τ_{mem}	15	15	membrane time constant [ms]
τ_{syn}	40	40	synaptic plasticity time constant [ms]
τ_{act}	11000	11000	activation time constant [ms]

Table 3.1: Parameter list of the main variables used for both types of units for 100 × 100 networks. See text for details.

such that

$$S_i(t + \Delta t) = \frac{(t_{\text{refract}[q]} - \Delta t)}{t_{\text{refract}[q]}} \cdot S_i(t) \tag{3.3}$$

for any $\Delta t < t_{\text{refract}[q]}$. For a refractory period equal to 1 time unit, the state $S_i(t)$ is a binary variable. It is assumed that a unit can generate a spike only for $S_i(t) = 1$. The parameter values used for the simulations are listed in table 3.1, p.19.

3.2.2 Background activity

The background activity $B_i(t)$ is used to simulate the input of afferents to the i^{th} unit that are not explicitly simulated within the network. Let us assume that each type of unit receives $n_{\text{ext}[q_i]}$ external afferents. In the present study, we simplify by a setting that all units receive the same number of external projections and that all of them are excitatory. Specifically, we assume that $n_i \equiv n \equiv 50$ and that the postsynaptic potential generated by these external afferents is fixed at a value equal to $P_{[1,1]}$. In the current case (see table 3.1, p.19), each external afferent generates an excitatory postsynaptic potential equal to 0.84 mV.

We assume that the external afferents are correlated among themselves. This means that each time a unit is receiving a correlated input from 50 external afferents, its membrane potential is depolarized to an extent that will generate a spike. Such external input is distributed according to a Poisson process which is independent for each unit and with mean rate λ_i. The rate of external background activity is a critical parameter. In the present study, we have set the

Poisson input to a rate $\lambda_i = 5\ spikes/s$ for most simulations, but network activity was sustained with frequencies as low as $\lambda_i = 3\ spikes/s$.

3.3 Synaptic connection model

The postsynaptic potential w_{ji} is a function of the state of the presynaptic unit S_j, of the "type" of the synapse $P_{[q_j,q_i]}$, and of the activation level of the synapse A_{ji}. This is expressed by the following equation

$$w_{ji}(t+1) = S_j(t) \cdot A_{ji}(t) \cdot P_{[q_j,q_i]}. \tag{3.4}$$

Notice that the "type" of the synapse is a parameter that depends on the types of units in the network. In the current study, we assume that $P_{[1,1]}$, i.e. (*Type I* $\rightarrow$ *Type I*), and $P_{[1,2]}$ connections, i.e. (*Type I* $\rightarrow$ *Type II*), are of the same kind. A similar assumption was made for $P_{[2,1]}$ and $P_{[2,2]}$ connections. In order to maintain a balanced level of depolarization (excitatory) and hyperpolarization (inhibitory), the *Type I* unit was considered as excitatory and *Type II* as inhibitory. We set $P_{[1,1]} = P_{[1,2]} = 0.84mV$ and $P_{[2,1]} = P_{[2,2]} = -1.40mV$.

3.3.1 Adaptation

It is assumed *a priori* that modifiable synapses are characterized by activation levels $[A]$ with N attractor states $[A_1] < [A_2] < \cdots < [A_N]$. Activation levels of type $[1,1]$ synapses are *integer-valued levels* $A_{ji}(t)$, with $A_{ji}(t) \in \{[A_1] = 0, [A_2] = 1, [A_3] = 2, [A_4] = 4\}$. Index j is referred to as the presynaptic unit and index i as the postsynaptic unit. We assume that postsynaptic potentials generated by synapses of type $[1,1]$ correspond to synaptic currents mediated by NMDA glutamatergic receptors. These discrete levels could be interpreted as a combination of two factors: the number of synaptic *boutons* between the pre– and postsynaptic units and the changes in synaptic conductance as a result of Ca^{2+} influx through the NMDA receptors. In the current study, we attributed a fixed activation level (that means no synaptic modification) $A_{ji}(t) = 1$, to $exc \rightarrow inh$, $inh \rightarrow exc$, and $inh \rightarrow inh$ synapses.

A *real-valued variable* $L_{ji}(t)$ is used to implement the spike-timing-dependent synaptic plasticity (STDP) rule for $A_{ji}(t)$, with integration of the timing of the pre– and postsynaptic activities. The variables $L_{ji}(t)$ are user-defined boundaries of attraction $L_0 < L_1 < L_2 < \cdots < L_{N-1} < L_N$

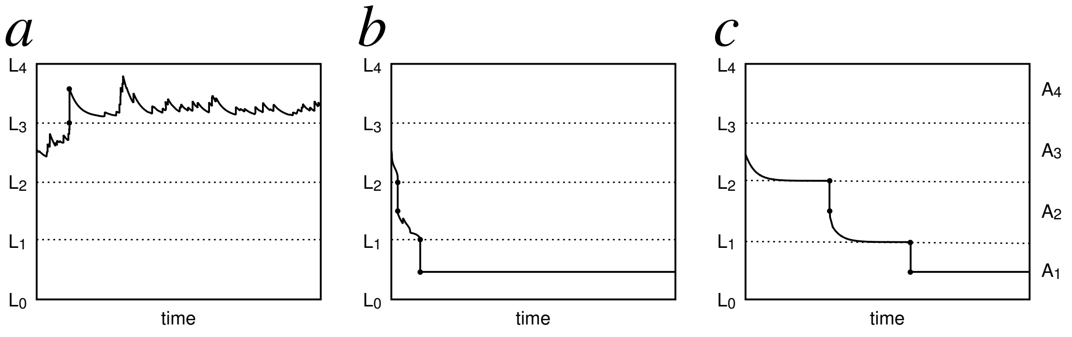


Figure 3.3: The *real-valued variable* L_{ji} is increased or decreased according to the STDP rule. If the value L_{ji} reaches one of the L_k user-defined boundaries, a jump occurs in the *integer-valued variable* $[A_k]$. In the beginning, all **(e→e)** synapses have been set at activation level $[A_3]$. ***(a):*** example of potentiation with an increase in synaptic strength that is stabilized on the long term; ***(b):*** example of depression with a fast decrease in synaptic activation level down to its minimal level, $[A_1] = 0$ which provokes the elimination of the synapse; ***(c):*** example of a synaptic link that is neither affected by potentiation nor by depression, but which efficacy decays down to $[A_1] = 0$ according to the time constant τ_{act}.

satisfying $L_{k-1} < [A_k] < L_k$ for $k = 1, \cdots, N$. This means that whenever $L_{ji} > L_k$, the activation variable A_{ji} jumps from state $[A_k]$ to $[A_{k+1}]$. Similarly, if $L_{ji} < L_k$ the activation variable A_{ji} jumps from state $[A_{k+1}]$ to $[A_k]$. Moreover, after a jump of activation level $[A]$ occurred at time t, the real-valued variable L_{ij} is reset to $L_{ij}(t+1) = \frac{L_k + L_{k+1}}{2}$ (see figure 3.3 for a graphical example).

STDP defines how the value of L_{ji} at time t is changed by the arrival of presynaptic spikes, by the generation of postsynaptic spikes and by the correlation existing between these events. On the generation of a postsynaptic spike (i.e. when $S_i = 1$), the value L_{ji} receives an *increment* which is a decreasing function of the elapsed time from the previous presynaptic spike at that synapse (i.e. when $S_j = 1$). Similarly, when a spike arrives at the synapse, the variable L_{ji} receives a *decrement* which is likewise a decreasing function of the elapsed time from the previous postsynaptic spike. This rule is summarized by the following equation: $L_{ji}(t+1) = L_{ji}(t) + (S_i(t) \cdot M_j(t)) - (S_j(t) \cdot M_i(t))$, where $S_i(t), S_j(t)$ are the state variables of the i^{th} and j^{th} units and $M_i(t), M_j(t)$ are inter-spike decay functions. $M_i(t)$ may be viewed as a "memory" of the latest inter-spike interval,

$$\begin{aligned} M_i(t+1) &= S_i(t) \cdot M_{\max[q_i]} \\ &\quad +(1 - S_i(t)) \cdot M_i(t) \cdot k_{\text{syn}[q_i]} \end{aligned} \tag{3.5}$$

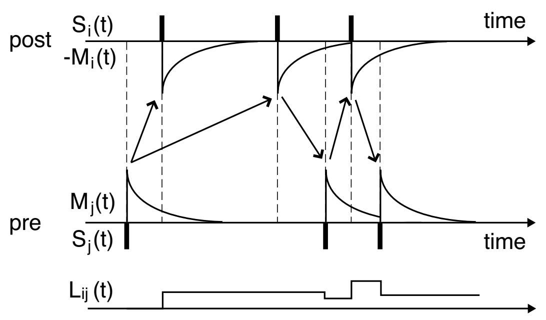


Figure 3.4: Graphical representation of the interactions between $S(t)$ and $M(t)$ variables in our STDP rule. Time flows from the left to the right. Presynaptic cell activity (j) appears on the bottom line, postsynaptic cell activity (i) on the top line. Bold ticks mark the occurrences of spikes ($S(t) = 1$) that correspond to the times when $M(t)$ is reset to $M_{\max}$. $M(t)$ is an exponentially decaying variable. When the spike order is pre– before postsynaptic (*up arrows*), the positive contribution of $S_i(t) \cdot M_j(t)$ can be read where the interrupted line crosses $M_j(t)$. When the spike order is post– before presynaptic (*down arrows*), the negative contribution of $S_j(t) \cdot M_i(t)$ can be read where the interrupted line crosses $M_i(t)$.

where $k_{\text{syn}[q_i]} = \exp(-t/\tau_{\text{syn}[q_i]})$ is the time constant associated to the memory, $\tau_{\text{syn}[q_i]}$ is the synaptic plasticity time constant characteristic of each type of unit, and $M_{\max[q_i]}$ was set $M_{\max[q_i]} = 2$ for all units of either type in this study. In the event where neither the pre– nor the postsynaptic unit should fire a spike, the real-valued variable will decay with a time constant $k_{\text{act}[q_j,q_i]} = \exp\left(-1/\tau_{\text{act}[q_j,q_i]}\right)$ characteristic for each type of synapse, such that the final equation is the following:

$$\begin{aligned} L_{ji}(t+1) &= L_{ji}(t) \cdot k_{\text{act}[q_j,q_i]} \\ &\quad +(S_i(t) \cdot M_j(t)) \\ &\quad -(S_j(t) \cdot M_i(t)). \end{aligned} \tag{3.6}$$

Figure 3.4 shows a graphical representation of the interaction between $S(t)$ and $M(t)$ variables.

In the present study, the differences between the user defined boundaries L_k were all equal, such that $\Delta L_k = L_k - L_{k-1} = 20$ for any attractor state $[A_k]$. In the beginning of the simulation, all modifiable synapses were set to activation level $[A_3] = 2$. Figure 3.3a illustrates a case when the synaptic link receives a potentiation determined by the STDP rule described above. The activation variable jumps from $[A_3]$ to $[A_4]$ and stabilizes at highest activation level. Figure 3.3b illustrates a case when the synapse is continuously depressed such that the activation variable jumps from $[A_3]$ to $[A_2]$, and then from $[A_2]$ to $[A_1]$ faster than its spontaneous decay, determined

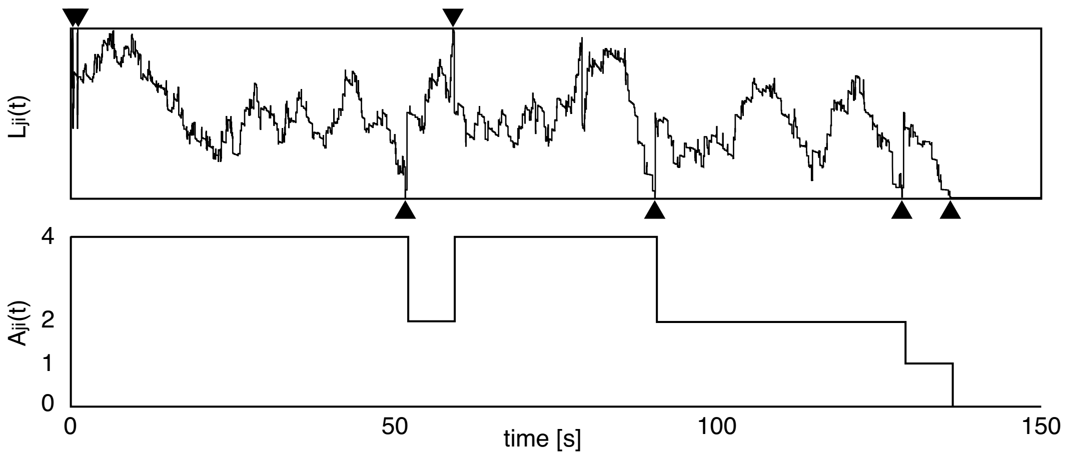


Figure 3.5: Co-evolution of $L_{ji}(t)$ and $A_{ji}(t)$ for a sample excitatory-excitatory connection during the first 150 seconds of a simulation. Each time $L_{ji}(t)$ outcomes the upper bound (*down-pointing triangles*), or the lower bound (*up-pointing triangles*), $A_{ji}(t)$ state changes accordingly, constrained by the possible values $\{0, 1, 2, 4\}$. At $t \simeq 137$ seconds, $A_{ji}(t)$ reaches the value of 0 and the connection is definitely pruned. L_{ji} and A_{ji} are dimension-less variables.

by time constant $k_{\mathrm{act}[q_j,q_i]}$. Figure 3.3c illustrates a case when the synapse is neither depressed nor potentiated, and the activation level spontaneously decays down to the minimal level.

3.3.2 Pruning

No generation of new projections is allowed in the present study, although specific rules could be defined to this purpose. Synaptic pruning occurs when the activation level of a synapse reaches a value of zero. This means that synaptic pruning may occur only for synaptic connections of type $[1, 1]$, which are also the most abundant, when the activation level A_{ji} decreases to its minimal value, i.e. $[A_1] = 0$. In this case, the synapse $[i, j]$ is eliminated from the network connectivity. Figure 3.5 shows side by side the evolution of L_{ji} and A_{ji} for a sample simulated connection.

3.4 Stimuli models

The first experiments discussed in the following Chapter involved networks that were not stimulated. The subsequent ones were stimulated with one of the two types of stimuli described here. The generic simulation layout was the following: from $t = 0$ to $t = 9$ ms, every unit received a small input on the membrane, randomly chosen between 0 to 30 mV, to "shake" the network and to provide a non-uniform initial state; from $t = 10$ to $t = 999$ ms, the network was left alone to relax before starting the experiment; from $t = 1,000$ ms to the end of the simulation, a stimulus was presented every 2,000 ms. The simulation duration was set to $5 \cdot 10^5$ or $1 \cdot 10^6$ time steps, 500 or 1,000 seconds respectively, after checking that the networks usually had reached a steady-state by that time. Overall, this represented 250 or 500 presentations of the stimulus along one simulation run. Note that we arbitrarily decided that one simulation time step corresponds to the duration of a spike: 1 ms. To ease the reading, we will ignore the difference between these two measures.

3.4.1 Simple spatiotemporal pattern

The stimulus was composed of vertical bars uniformly distributed over the 2D lattice surface, each bar being one lattice column wide (see figure 3.6a). At each time step during stimulus presentation, the bars were simultaneously moved one column to the right, such that each bar slipped over the entire surface of the network. See Appendix A, p.115 for animated sequences of this stimulus. Depending on the protocol needs, several parameters of this stimulus were modified:

size: The number of bars composing the stimulus was a function of the simulated network sizes: 9 bars for 90×90 networks, 10 bars for 100×100 networks, and 11 bars for 110×110 networks, such that the bars were always distant of 10 columns from one another and spanning over all the available surface.

duration: Three stimulus durations were used: 50 ms followed by 1,950 ms without any external input, 100 ms followed by 1,900 ms without any external input, and 200 ms followed by 1,800 ms without any external input.

input units: The stimulus was applied only to a fraction of the population formed by excitatory units; these units are called *input units*. The number of input units was expressed as a ratio (i.e. 3, 5, 7, or 10%) of the initial number of excitatory units. For a 100×100 network, 10% of input units corresponds to 800 input units, i.e. 10% of the 80% excitatory units of the 10,000 units. See below how the input units were chosen.

intensity: The stimulus applied to a particular input unit provoked a depolarization on its

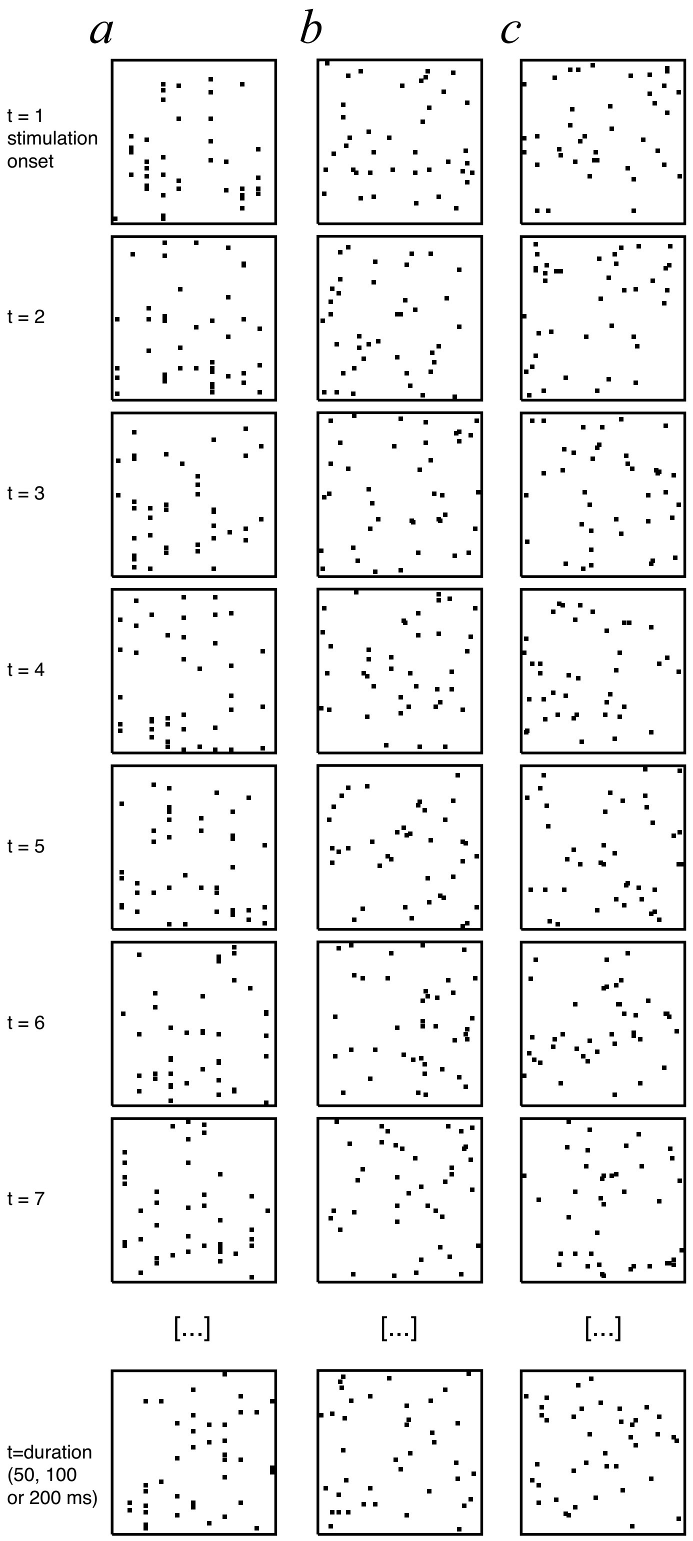


Figure 3.6: Stimuli models time decomposition. Squares represent 100 × 100 networks. Points mark the position of the input units receiving the stimulus at each time step – one time step per line. The 10 ms basic stimuli described in this section are looped to construct longer stimuli lasting 50, 100 or 200 ms, ***(a):*** simple stimulus, 5% input units. Note that the input units are aligned on 10 columns slowly moving to the right; ***(b):*** complex stimulus A, 5% input units; ***(c):*** complex stimulus B, 5% input units. See Appendix A, p.115 for animated sequences.

membrane with amplitudes equal to 0 (i.e. no stimulus), 30, 40, 50, and 60 mV. The stimulus intensity was selected in the beginning and was stable during a simulation run.

To summarize the stimulation procedure, let us consider the following example. For each of the input units – selected among the 80% of excitatory units – of a 100×100 network stimulated with a 100 ms stimulus, one stimulus presentation resulted in a sequence of 10 external inputs equally distributed in time every 10 ms. At the network level, each stimulus presentation corresponded to a spatiotemporal sequence characterized by 10 groups of 80 synchronously excited units stimulated 10 times during 10 ms. Note that the spatial distribution of the 10 groups was constrained by the bars pattern.

In addition to the stimulus parameters previously defined, and in order to control the effect of the stimulation, we introduced 3 protocols differing in the method used to choose the input units:

no stimulation: No input units were chosen at all. This condition corresponds to a stimulation of zero intensity;

random stimulation: At each stimulus presentation, the input units were randomly chosen, such that the input units changed at every new stimulus presentation;

fixed stimulation: The input units were selected in the beginning of the simulation and remained the same for every stimulus presentation.

3.4.2 Complex spatiotemporal pattern

This stimulus pattern is slightly more complex than the previous (see figure 3.6b and c). The timing components are related, but the spatial component is different. This pattern was used only once, for the experiment described in section 4.2.4, p.47. For this reason, it was only defined for 100×100 networks and used only with the intensities of 60 mV and 0 mV (control). Before the simulation started, two non-overlapping sets of 400 units were randomly selected from the 8,000 excitatory units, labeled sets A and B. Each set was randomly divided into 10 ordered groups of 40 units, $A = \{A_1, A_2, \dots, A_{10}\}$ and $B = \{B_1, B_2, \dots, B_{10}\}$. At each time step during a stimulus presentation, the 40 units of one group received a large excitatory input on their membrane, leading to their synchronous firing. The 10 groups of a set were stimulated following an ordered sequence, thus defining a 10 ms long spatiotemporal pattern (see Appendix A, p.115 for animations). These 10 ms patterns were then concatenated to compose 100 ms stimuli according to one of the six following protocols:

No stimulation: No input units were stimulated. This condition corresponds to a stimulation of zero intensity;

AA stimulation: $10\times$ pattern A in a row;

BB stimulation: $10\times$ pattern B in a row;

AB stimulation: $5\times$ pattern A followed by $5\times$ pattern B;

BA stimulation: $5\times$ pattern B followed by $5\times$ pattern A;

AB|BA stimulation: A random mixture of stimulus $5\times$ pattern A followed by $5\times$ pattern B, and stimulus $5\times$ pattern B followed by $5\times$ pattern A. Before each presentation, one of the two stimuli was randomly selected with equal probability.

Fett's Law

Never try to repeat
a successful first experiment.

– Anonymous

Chapter 4

Results

Résumé Les résultats des simulations sont présentés dans ce Chapitre. Une première série d'expériences a été réalisée dans le but de tester que la connectique tirée aléatoirement et l'épissage associé à la plasticité synaptique à modulation temporelle relative (STDP) ne présentaient pas de biais intrinsèques en absence de stimuli externes au réseau. L'espace des paramètres du modèle étant très important, seuls les résultats obtenus lors de l'exploration de certains d'entre eux sont rapportés ici. Ainsi, un facteur d'échelle a pu être approximé pour compenser l'effet de la taille du réseau sur l'intensité de l'activité. Les réseaux ont ensuite été stimulés avec les motifs spatiotemporels plus ou moins complexes décrits précédemment. L'analyse des connexions survivantes à la fin des simulations, ainsi que l'analyse des séries temporelles résultantes de l'activité des neurones fortement interconnectés, suggèrent que des circuits *feed-forward* émergent par épissage des réseaux initialement tirés au hasard.

Simulation results are presented in this Chapter. A first batch of experiments was performed to test that the randomly generated connectivity and the STDP-driven pruning did not show any spurious bias in absence of stimulation. The model parameter space is so vast, that only the results obtained for some of them are described here. Among other things, a scale factor was approximated to compensate for the network size on the activity. Networks were then stimulated with the spatiotemporal patterns discussed previously. The analysis of the connections remaining in the end of the simulations, as well as the analysis of the time series resulting from the interconnected units activity, suggest that feed-forward circuits emerge by pruning from the initially randomly connected networks. Each experiment is described in a separate section. A *parameters summary* box helps keeping track of the differences between the increasingly more complex simulation setups.

N	size	$\phi^*_{[1]}$ [%]	$P^*_{[1,1]}$ [mV]
1	10×10	9.28	2.36
2	20×20	5.84	1.64
3	30×30	4.46	1.35
4	40×40	3.68	1.19
5	50×50	3.17	1.08
6	60×60	2.81	1.01
7	70×70	2.53	0.95
8	80×80	2.32	0.90
9	90×90	2.14	0.87
10	100×100	2.00	0.84
11	110×110	1.88	0.81

Table 4.1: Scaled parameter values for each network size N. Note that the values for $N = 10$ correspond to those listed in table 3.1, p.19. See text for details.

4.1 Preliminary work

4.1.1 Size effect

parameters summary	
network size	10×10 to 100×100
background	10 sp/s
stimulus	none
scaled	yes

We investigated the pruning dynamics with networks of different sizes. The smallest network was defined by 10×10 units and the largest by 100×100 units, i.e. $(10 \times N)^2$, with $N \in \{1, \ldots, 10\}$. To compensate for the changes in the balance between excitation and inhibition induced by the change of size, we introduced the scaling factor

$$f = \sqrt[3]{\frac{10^4}{(10 \cdot N)^2}} \tag{4.1}$$

where N is the size as described before. The uniform probability for an excitatory unit to project to any other unit of the network $\phi_{[1]}$ was scaled according to

$$\phi^*_{[1]} = f \cdot \phi_{[1]}. \tag{4.2}$$

The level of postsynaptic depolarization for excitatory-excitatory synapses $P_{[1,1]}$ was scaled according to

$$P^*_{[1,1]} = \left(1 + \frac{f-1}{2}\right) \cdot P_{[1,1]}. \tag{4.3}$$

This way, small networks have a larger number of stronger excitatory connections to balance

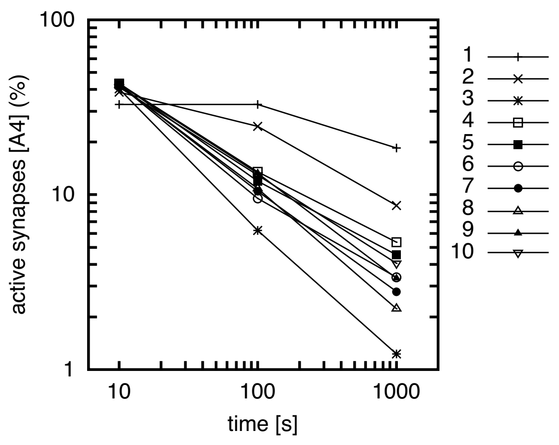


Figure 4.1: Pruning dynamics averaged over $n = 10$ simulations for each network size. With the proposed size scaling factor, the pruning dynamics are comparable for network sizes $N \in \{4, \cdots, 10\}$. Simulations for $N = 1$ and $N = 2$ saturated, suggesting that the scaled parameter values were too large for these two specific sizes.

the inhibitory connections when the simulation begins. Table 4.1, p.30 lists the scaled values of $\phi^*_{[1]}$ and $P^*_{[1,1]}$ for the network dimensions we simulated.

It is interesting to notice that the final ratio of active synapses ($R_{\max[A]}$) represented only few percents of the initial number of synapses (see figure 4.1). In addition, it is important to notice that the connections reaching the maximal activation level did not necessarily remain active until $t = t_{\text{steady}}$. Several synapses reached the activation level $[A_4]$ after some delay, and then decreased at variable speed to $[A_1] = 0$, before they were eventually eliminated. The network is expected to work in a range such that background activity is unable to create spurious attractors by STDP. This means that background activity alone should not create stable connections that would shape the topology of cell assemblies. The size of the network is critical if the goal is to detect the emergence of circuits embedded in a large network. Figure 4.1 shows that the ratio of active synapses with activation level equal to $[A_4]$ could be as high as 50% of all initial synapses. The final percentage of active units is much less variable and it is always less than 10% at t_{steady} for networks that did not saturate.

4.1.2 Seed effect

parameters summary	
network size	100×100
background	10 sp/s
stimulus	none
random number generator seeds	100

A simulation study that relies on large use of randomly generated numbers may fall into local minima or spurious attractors simply by chance. It was necessary to assess the effect of the random number generator seed on our simulations. The most critical effect of the randomization might occur in the very beginning, when the initial network topology is created according to the connection density functions for the different types

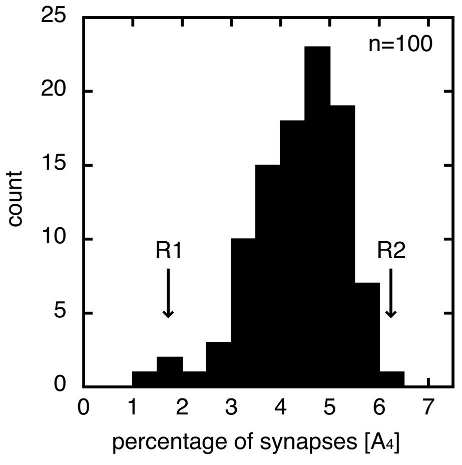


Figure 4.2: "Seed effect" on the number of synapses remaining after 1,000 seconds. $n = 100$ simulations were performed using different random number generator seeds. The distribution of $R_{\max[A]}$ at time $t = t_{\text{steady}} = 1,000$ seconds shows that, in the majority of the runs, synaptic pruning left 3.0 to 6.0% active synapses at the maximum level $[A_4]$ (bin=0.5).

of units. The very same simulation, with the parameter set described in table 3.1, p.19 was repeated 100 times using different random number generator seeds with the 100×100 units network size.

The choice of the seed had a significant impact on the value of $R_{\max[A]}$ at time t_{steady}, as it could vary in the range $[1.30, 6.03]\%$. However, as shown by the distribution of $R_{\max[A]}$ (figure 4.2), about 90% of these values were comprised between 3.0 and 6.0%. Moreover, we never observed cases with absence of convergence at delays as large as $t = 1,000$ seconds. This indicates the existence of a "seed effect", which does not cause changes in the overall dynamics of synaptic pruning.

A bias in the connections orientation could occur by random choices. In order to test this hypothesis, two cases of extreme values of $R_{\max[A]}$ observed in the distribution of figure 4.2 were selected. The first case corresponds to $R_{\max[A]}$ as low as $R_1 = 1.97\%$. The second case corresponds to $R_{\max[A]}$ as high as $R_2 = 6.03\%$. For both cases, we calculated the deviation from an isotropic connection (see figure 3.1c and g) for all active synapses, i.e. with an activation level not equal to $[A_1] = 0$. Then, we calculated an average deviation plot that corresponds to the mean of the orientations computed over all active synapses at given times t.

Figure 4.3a shows the evolution of the orientation map in the case R_1, when the network stabilizes with a low level of active connections. In this example, the number of active excitatory-excitatory connections n was: $n = 1,517,240$ initially, $n = 330,920$ at $t = 100s$, and $n = 172,503$ at $t = 200s$. The network eventually stabilized with $n = 30,864$ active synapses at $t = t_{steady} = 1,000$ seconds. The orientation map shows that the deviations from an isotropic distribution

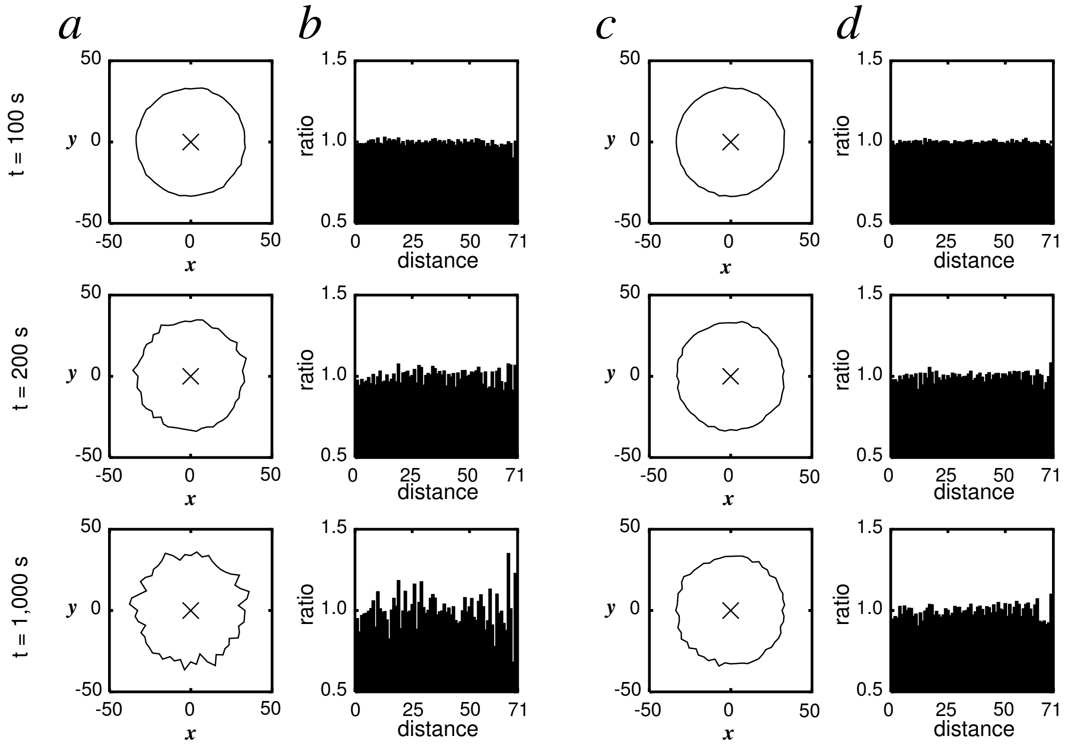


Figure 4.3: Random number generator seed effect on the active connections orientation and length. ***(a):*** Average orientation map. A circular line indicates an isotropic projection orientation of a unit ideally located in the center marked by a cross. This data corresponds to run R_1 from figure 4.2. The average deviation from isotropy for all active connections is plotted at various times. The last line shows the situation at t_{steady}: $30,864$ synapses remained active, all with active state $[A_4]$, representing 1.97% of the initial number of synapses at time $t = 0$. ***(b):*** Normalized histogram of the source-to-target length measured as the Euclidean distance in the 2D lattice for simulation run R_1. A flat line at $ratio = 1$ indicates that the distances are totally predicted by the modified 2D Gaussian density function described in section 3.1.2, p.16. ***(c):*** Average orientation map corresponding to run R_2 from figure 4.2. At t_{steady}, $93,346$ synapses remained active, all with active state $[A_4]$, representing 6.03% of the initial number of synapses at time t_0. ***(d):*** Normalized histogram of the source-to-target length measured as the Euclidean distance in the 2D lattice for simulation run R_2.

were equally distributed in all directions. Another factor that could be affected by the random choice is the distance from source-to-target (calculated as an Euclidean distance over the 2D lattice) of the remaining projections. The distance distribution histogram (figure 4.3b and d) was normalized with respect to the probability distribution of establishing a connection. In this normalized histogram, a ratio of 1 means that the count is perfectly determined by the probability distribution. In the case of R_1, we observed a tendency to some deviation from the original probability function, but this variance was the same for any source-to-target distance. In the case R_2, the initial number of synapses was $n = 1,512,634$ and $n = 93,346$ active synapses remained at t_{steady}. This analysis shows that the "seed effect" does not introduce significant biases either in the orientation, or in the length of the connections that were selected by pruning.

4.2 Stimulated networks

After checking the intrinsic properties of our model, we studied the effect of selected stimuli on pruning dynamics and distributions. The type of stimuli, as well as the experimental protocols were described in section 3.4, p.23.

Figure 4.4 shows the evolution of the proportions of excitatory-excitatory connections in each activation level during a simulation. We observed that, in presence of mere background activity, about 20% of the connections had been pruned ($A_{ji} = [A_0]$), and about 20% of the connections had reached the strongest state ($A_{ji} = [A_3]$) by the time of the first stimulus presentation ($t = 1$ second). The first stimulus presentation was followed by a period of network saturation of less than 1 second during which the proportions of connections did not change. After this period of artificial hyper-activity, the pruning resumed until $t = t_{steady} = 500$ seconds when the proportions reached a stable state. Among the remaining active synapses, defined by $A_{ji} \neq [A_0]$, almost all were characterized by the largest activation level.

In the end of the simulation, i.e. at $t = t_{steady} = 500$ seconds, we identified the pool of excitatory units that were still "alive", discarding all the input units. From this pool, a subset of units was selected on the basis of their connection pattern from and to the pool itself. Those units with at least three strong projections to ($k_{out} \geq 3$) and three projections from ($k_{in} \geq 3$) other units of the pool are dubbed *strongly interconnected units* (*SI*-units).

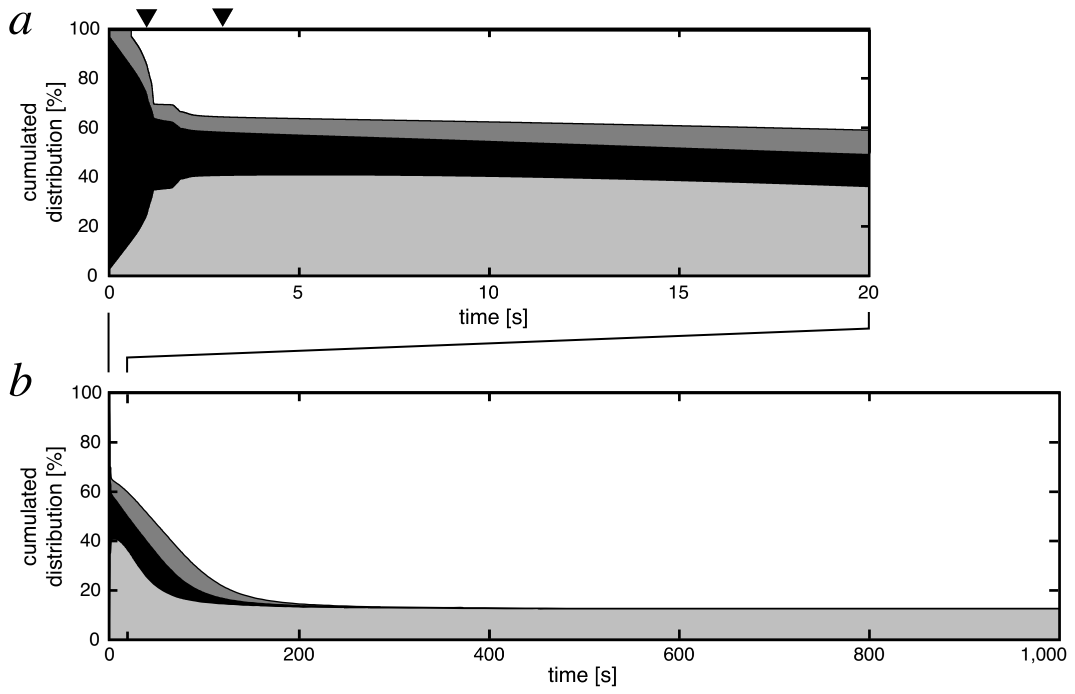


Figure 4.4: Evolution of the proportion of excitatory-excitatory connections at each activation level $[A]$ as a function of simulated time. ***(a):*** from $t = 0$ to $t = 20s$ (close-up view); ***(b):*** from $t = 0$ to $t = 1000s$; *white*: $[A_0]$; *dark gray*: $[A_1]$; *black*: $[A_2]$; *light gray*: $[A_3]$; *down-pointing triangles*: onset of first and second stimulus presentations. At $t = 0$, all connections ($n = 1,516,384$) are initialized at $A_{ji} = [A_2]$. The proportion of connections at levels $[A_1]$ and $[A_2]$ rapidly tend to 0. At $t = 500$ seconds: 87.29% $[A_0]$, 0.05% $[A_1]$, 0.09% $[A_2]$, 12.57% $[A_3]$. At $t = 1,000$ seconds: 87.59% $[A_0]$, 0.02% $[A_1]$, 0.04% $[A_2]$, 12.35% $[A_3]$. Network size: 100×100; stimulus duration: 100 ms; stimulus intensity: 60 mV; input units ratio: 10%; fixed stimulus protocol.

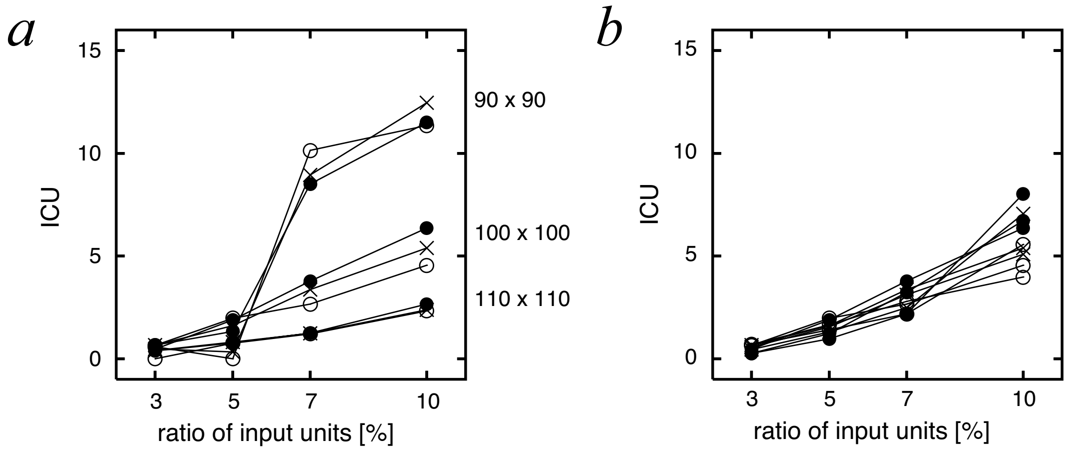


Figure 4.5: The index of connected units (ICU), i.e. the ratio between the number of input units and the number of *SI*-units, as a function of the ratio of input units, stimulus duration, and network dimensions. Labels of stimulus duration: ○: 50 ms; ×: 100 ms; •: 200 ms. ***(a):*** simulations performed with the parameter values listed in table 3.1, p.19 for all network sizes; ***(b):*** like *(a)* except for the size-specific scaled variables defined in table 4.1, p.30. Stimulus intensity: 60 mV; fixed stimulation protocol.

4.2.1 Stimulus duration

parameters summary	
network size	90×90 to 110×110
background	5 sp/s
input units ratio	3, 5, 7, 10%
stimulus pattern	simple
stimulus duration	50, 100, 200 ms
stimulus intensity	60 mV
scaled	yes, no

During the pruning process, only the modifiable connections that keep a sufficient level of STDP-driven activity can "survive". The first step for searching an oriented topology was to detect the *excitatory neighbourhood* of the *SI*-units. This neighbourhood corresponds to the set of excitatory units that send a projection to the *SI*-units, that receive a projection from the *SI*-units, or that both send and receive projections. Thus, this neighbourhood may also include input units, i.e. the units that are depolarized by the stimulus. The ratio between the number of input units belonging to the neighbourhood and the number of *SI*-units defines the *index of connected units* (ICU). The larger the ICU, the larger the influence of the input units on the *SI*-units.

The ICU was computed for different network dimensions, with and without scaled parameters, stimulus durations, and input units ratio for the fixed stimulus protocol. The results are gathered in figure 4.5. With an input units ratio equal to 3%, we observed that the value of ICU is almost equal to zero and independent of the other parameters, because the amount of stimulus delivered to the network is not large enough to distinguish a stimulus-driven pruning. Such pruning appears with 5% of input units and becomes clearly visible with 7 and 10% of input units. It is worth noting that a stimulus lasting 200 ms provoked an effect similar to a stimulus lasting 50 ms. Conversely, the “network size” effect is not interesting by itself, as it is consistent with the fact that the smaller the network, the larger the impact of a certain input units ratio. In addition, applying the parameter scaling factor introduced in section 4.1.1, p.30, almost canceled the size effect (compare figure 4.5a and b).

4.2.2 Circuit emergence

parameters summary	
network size	100×100
background	5 sp/s
input units ratio	10%
stimulus pattern	simple
stimulus duration	100 ms
stimulus intensity	60 mV
stimulus protocol	none, random, fixed

The evolution of the in– and out-degrees for the SI-units and their neighbourhood was studied as a function of the simulation duration for a 100×100 network. Three experimental protocols with the simple spatiotemporal stimulus were used: none, random and fixed stimulation. The state of the network was analysed at $t = 50$, $t = 200$ and $t = 500$ seconds (figure 4.6). In the beginning of a simulation, an average excitatory unit receives and sends projections to about 190 other excitatory units, i.e. $k_{in} = k_{out} \approx 190$ (see figure 4.6a-c). The variability comes from the connectivity probability distribution used to randomly set these connections. As no new connections are established during the simulation, k_{in} and k_{out} can only decrease under the pressure of the pruning process. Some units tend to loose their incoming connections first, others tend to loose their outgoing connections first. The existence of other processes combining different speeds for the loss of input and output connections results in the smear of points visible in figure 4.6d-f.

We observed that as soon as $t = 50$ seconds, corresponding to 25 stimulus presentations with the fixed stimulation protocol (figure 4.6f), the evolution of the neighbourhood units input and output degrees was different from the other two protocols. Plots for $t = 200$ and $t = 500$ seconds show that most units have $k_{out} \ll k_{in}$, which indicates that the pruning modified the topology of the connections and favored the emergence of the converging pattern. The comparison of these degrees between $t = 200$ and $t = 500$ s (figure 4.7a-c *vs.* figure 4.7d-f) shows that the tendency to loose outward projections continued during the last part of the simulation. In particular, notice that a large part of the neighbourhood population lost all its input connections ($k_{in} = 0$); these units “survived” only because the background noise maintained some of their outward connections timely tuned with the discharges of their targets.

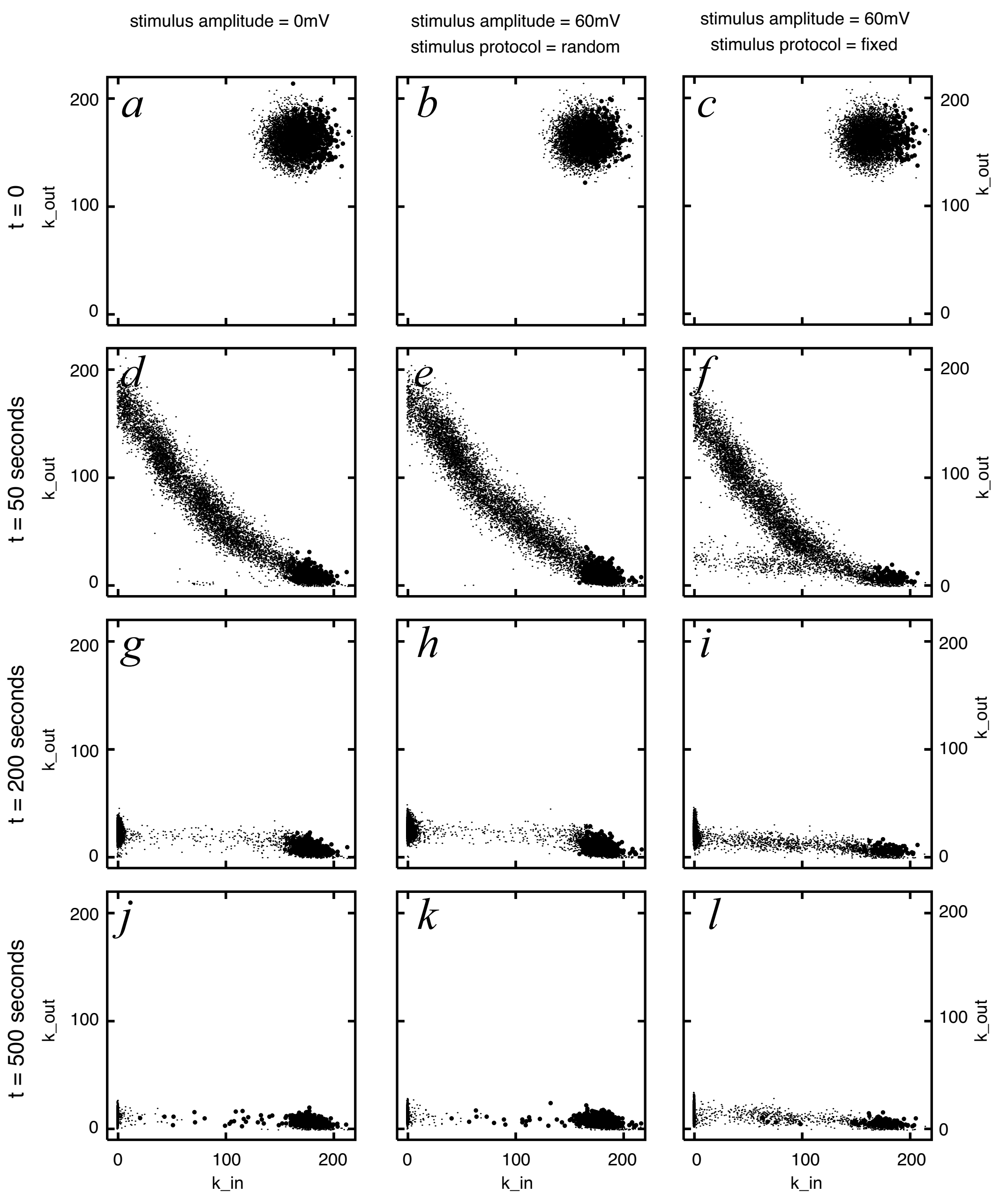


Figure 4.6: Evolution of the out-degrees (k_{out}) vs. in-degrees (k_{in}) for (•): *SI*-units; and (·): their neighbourhood units. ***(a, d, g, j):*** in absence of any input (362 *SI*-units, 6,954 neighbours); ***(b, e, h, k):*** random stimulation protocol (425 *SI*-units, 6,996 neighbours); ***(c, f, i, l):*** fixed stimulation protocol (123 *SI*-units, 6,762 neighbours). ***(a, b, c):*** initial situation at $t = 0$ is identical for all three protocols; ***(d, e, f):*** at $t = 50$; ***(g, h, i):*** at $t = 200$ seconds; ***(j, k, l):*** at $t = 500$ seconds; See figure 4.7 for a closer look to panels *(g-l)*. Network size: 100×100; stimulus duration: 100 ms; input units ratio: 10%.

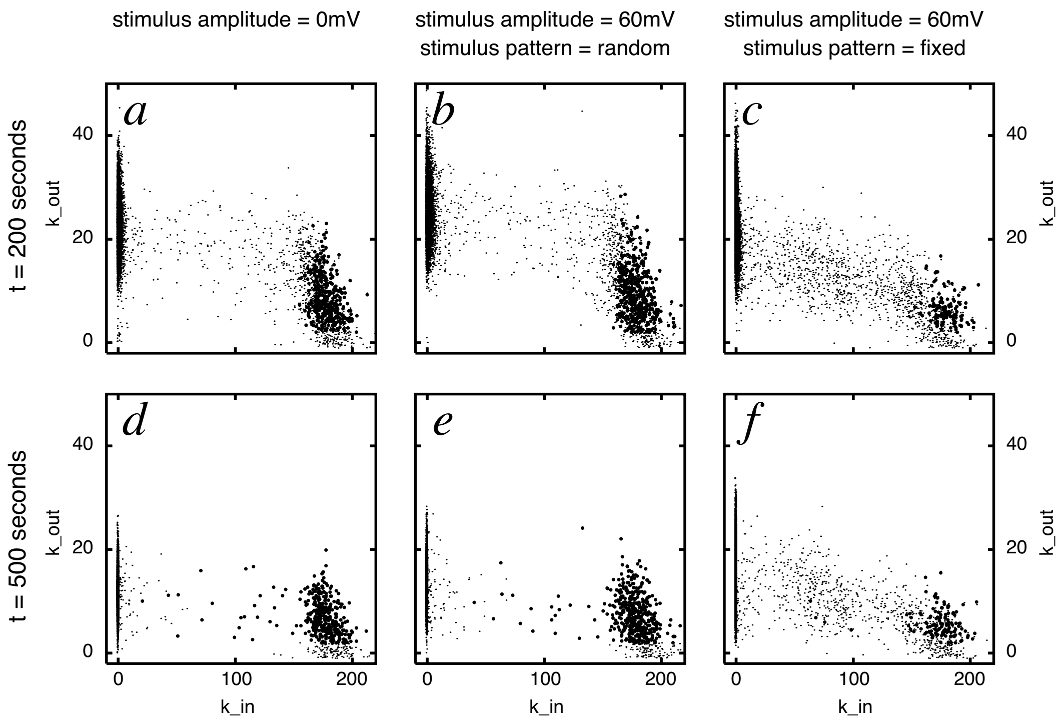


Figure 4.7: Evolution of the out-degrees (k_{out}) vs. in-degrees (k_{in}) for (•): *SI*-units; and (·): their neighbourhood units. These panels correspond to panels *(g-l)* in figure 4.6 with appropriately scaled axes. ***(a, b, c):*** $t = 200$ seconds; ***(d, e, f):*** $t = 500$ seconds. ***(a, d):*** in absence of any input; ***(b, e):*** random stimulation protocol; ***(c, f):*** fixed stimulation protocol. Network size: 100×100; stimulus duration: 100 ms; input units ratio: 10%.

Figure 4.7 shows that the distribution patterns for the random stimulation protocol (figure 4.7a,d) and in the absence of stimulation (figure 4.7b,e) are very similar. A random stimulus could not drive any significant effect, which was somehow expected, but it was necessary as a control experiment to detect any bias introduced in the simulation program. In the fixed stimulus protocol (figure 4.7c,f), we observed $n = 415$ units with $30 \leq k_{in} \leq 130$ at $t = 200$ s that are maintained at $t = 500$ s. There are only 26 units with these properties in the other two conditions. This population is composed of 407 input units belonging to the neighbourhood. These input units maintained a large k_{in}, because of the synchronization of their activity during the stimuli presentations. The vast majority of the input units ($> 85\%$) were presynaptic with respect to the *SI*-units, thus confirming that the topology organized towards a feed-forward converging pattern of connections.

The number of *SI*-units was smaller with the fixed stimulus and their $k_{in} \approx 180$ and $3 \leq k_{out} \leq 20$. It is important to notice that the distribution of the k_{in} of the *SI*-units did not change in time. In fact, the *SI*-units were characterized by an input pattern very close to the one they had when the simulation began. Their connectivity pattern, initially set at random, appeared to match some requirements for maintaining almost all the input connections. The interpretation is that the cell assembly formed by the *SI*-units was initially determined by chance, and maintained because of its internal connections when the pruning process started to select the active connections. The pruning let a particular network emerge, that was already embedded in the network at time $t = 0$. Different random seeds generated different populations of *SI*-units but the number of these units did not vary much as a function of the random seed.

Figure 4.8 shows the evolution of all the interconnections among a group of strongly interconnected units. The simulation lasted 500 seconds, corresponding to 250 simple spatiotemporal stimulus presentations. At $t = 500$ seconds, 49 *SI*-units were detected, connected by 69 projections (figure 4.8b). At $t = 0$ (figure 4.8a), these *SI*-units were connected by 98 connections randomly set according to the topographic rules described in section 3.1.1, p.15. At this point, connections have no feed-back or feed-forward directions. After 500 seconds of synaptic pruning driven by the stimulus and by STDP (figure 4.8b), an oriented topology emerged. According to this connection pattern, it was possible to determine a layered structure with a connection map characterized by feed-forward connections. Within this structure, a "layer" is virtual, in the

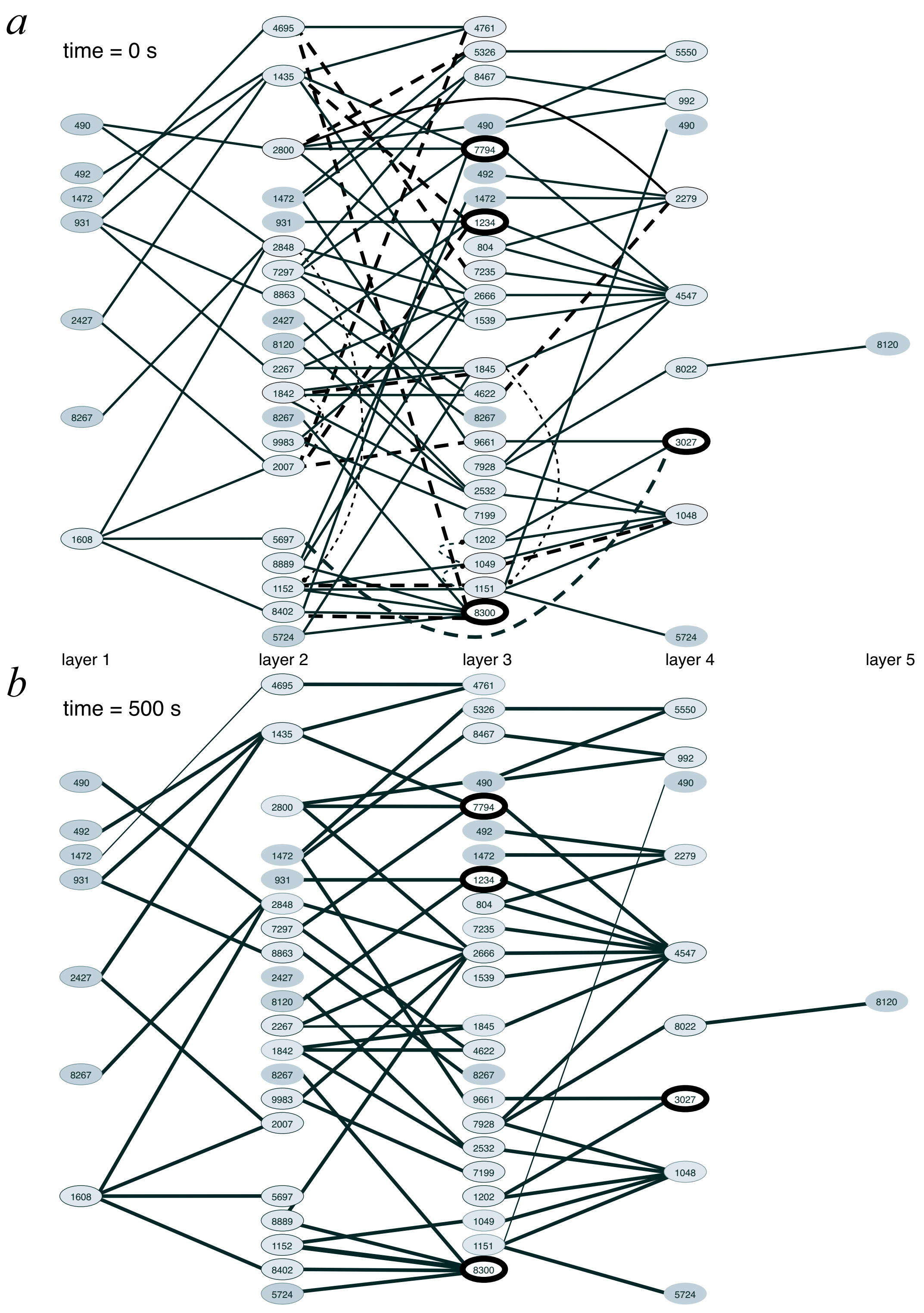


Figure 4.8: The 49 selected *SI*-units appear to be embedded into a layered circuit. ***(a):*** in the beginning of the simulation, 98 projections; ***(b):*** at $t = t_{steady} = 500$ seconds, 69 projections. Units are arranged in layers according to a best fit of their connections. *Plain lines*: projections going forward (left to right); *large interrupted lines*: projections going backward (right to left); *thin interrupted line*: projections to the same layer (up or down). Some units appear in more than one layer. Replicas are not surrounded with a black ellipse. Units surrounded by *bold ellipses* are further investigated in figures 4.13 and 4.14. Network size: 100×100; stimulus duration: 100 ms; stimulus intensity: 60 mV; input units ratio: 10%; fixed protocol.

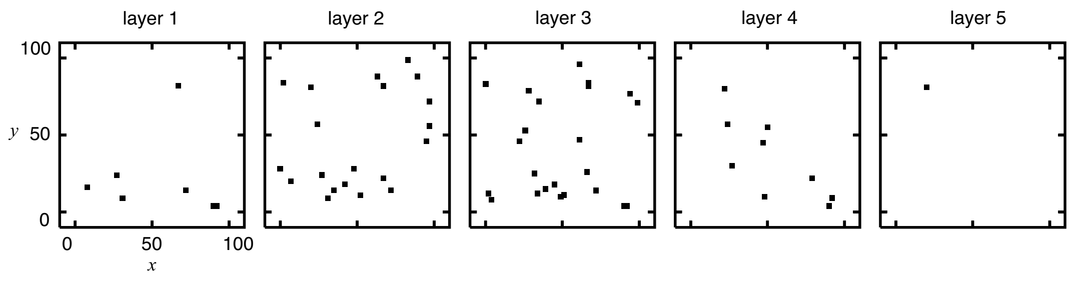


Figure 4.9: Each layer unit of figure 4.8 is represented as a dot at its position on the 2D lattice.

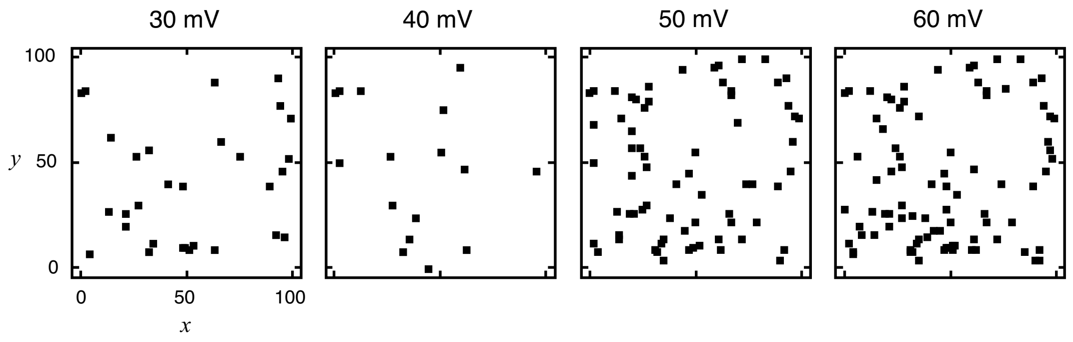


Figure 4.10: Example of the location of strongly interconnected units as a function of the stimulus-induced depolarization amplitude. Network size: 100×100; stimulus duration: 200 ms; input units: 10% of the excitatory units; fixed stimulation protocol.

sense that it groups a number of *SI*-units sharing feed-forward projections to the next "layer", and no topology is implied. The units belonging to each layer are scattered over the 2D lattice surface as shown in figure 4.9. The structure represented in figure 4.8 is not unique, and alternative circuits, very similar to the one presented here, could be determined. We have started to search for an efficient algorithm to detect the most plausible layered topology. At this stage, we rely on Graphviz[1] and hand editing, which is not totally satisfactory for our purpose. The 4 units surrounded by bold ellipses in figure 4.8 are further analysed in figures 4.13 and 4.14.

4.2.3 Stimulus intensity

The number of *SI*-units after 500 seconds depended on both the stimulation protocol and the amplitude of the stimulus-induced depolarization. In the *fixed stimulation* protocol condition, a small group of *SI*-units appeared with stimulus depolarizations as weak as 30 mV and their number increased with stronger stimuli (figure 4.10). Conversely, in the *random stimulation* protocol condition, the number of *SI*-units remained high and we did not observe a significant

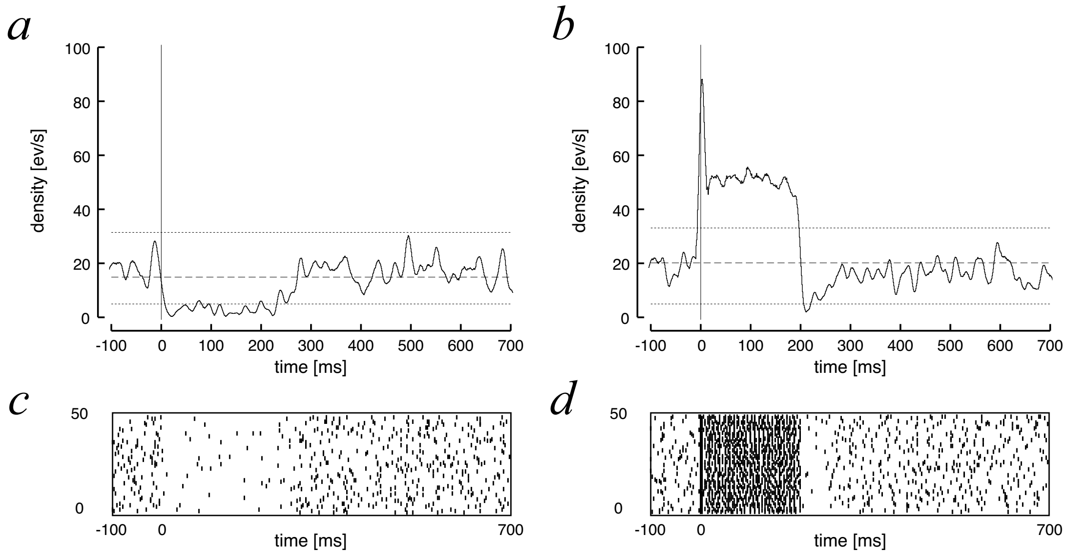


Figure 4.11: Response of two strongly interconnected sample units to 50 presentations of the fixed stimulation between time t = 450 and t = 500 seconds from the simulation start. ***(a, b):*** peri-event densities (PSTH) for the last 50 presentations of the stimulus, interrupted line: mean activity, dotted lines: confidence limits according to Abeles (1982b), Gaussian smoothing bin: $5ms$; ***(c, d):*** corresponding raster plots. Network size: 100×100; background activity: 5 sp/s; stimulus duration: 200 ms; stimulus intensity: 60 mV; ratio of input units: 10%; fixed stimulation protocol.

change in response to stimulus intensity.

parameters summary

network size	100×100
background	5 sp/s
input units ratio	10%
stimulus pattern	simple
stimulus duration	100, 200 ms
stimulus intensity	0, 30, 40, 50, 60 mV
stimulus protocol	fixed

The activity of all the SI-units was affected by the fixed stimulation presentation. Figure 4.11 shows the response of two SI-units to a simple spatiotemporal stimulus (200 ms), using the fixed stimulation protocol. The majority (nearly 80%) of the SI-units were excited during the stimulus presentation (e.g. figure 4.11b), despite the fact that, by definition, none of the SI-units was an input unit. It is interesting to notice that the remaining SI-units were strongly inhibited during the stimulus presentations (e.g. figure 4.11a), despite the fact that the stimulus was delivered only to excitatory cells. This is explained by the network of inhibitory units that receive the projections from the input units. The inhibitory response observed in the pool of the SI-units is due to the balanced network reacting to the increased activity by more inhibition.

We compared the SI-units circuits that emerged for stimulus intensities 30 and 60 mV. We selected four SI-units that were present in both emerged circuits and that received no common input from other SI-units (surrounded by bold ellipses in figure 4.8). We recorded the activity of these units in the fixed stimulation protocol conditions, and compared the time series

[1]`http://www.graphviz.org/`

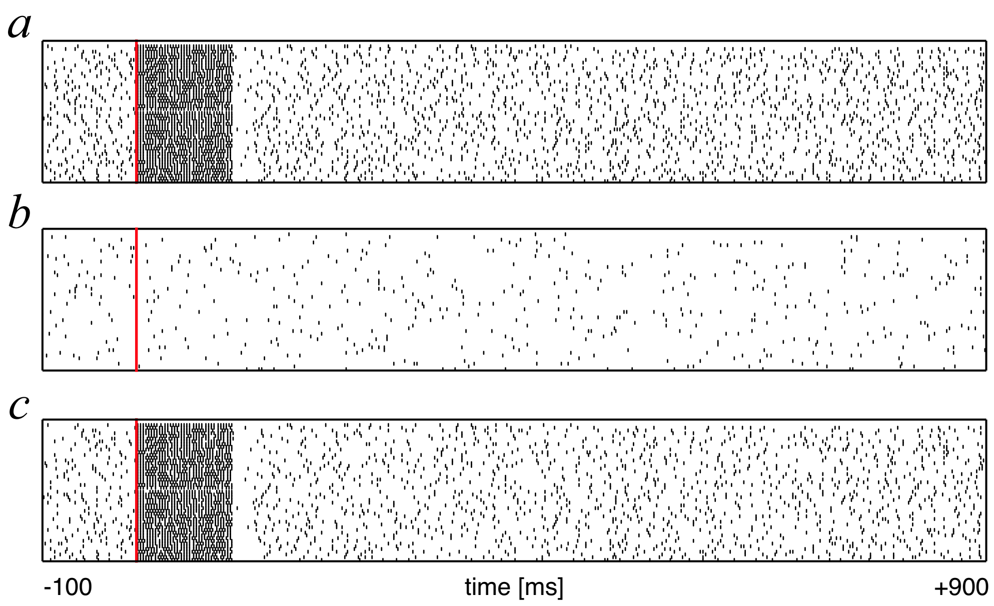


Figure 4.12: Construction of a differential raster. A raster plot is a cumulated representation of the neuron spiking activity aligned on a specific event. *Ticks* mark the occurrence of a spike. Time flows from left to right. Each line of the raster is stacked on top of the previous so that the stimulus onsets are aligned on the *vertical bar* at time $t = 0$. ***(a):*** original time serie recorded from the spiking activity of a neuron for the 50 last presentations of the stimulus; ***(b):*** spikes generated by the neuron background activity (see section 3.2.2, p.19); ***(c):*** the subtraction of the *(b)* time serie from *(a)* time serie. Network size: 100×100; background activity: 5 sp/s; stimulus : simple; stimulus duration: 100 ms; stimulus intensity: 60 mV; input units: 10%.

with mean firing rates in the range of 30 to 40 spikes/s. We computed the differential raster plots by subtracting the known random background activity from the recorded time series (see figure 4.12). Using this technique, the part of the activity driven by the interactions with the other units of the network can be extracted from the background activity. We observed that the stimulus offset is followed by a period of lower activity, explained by the balanced network reaction discussed before. The duration of this poststimulus inhibition period was a function of the stimulus intensity (compare 30 and 60 mV, figure 4.13a and figure 4.14a respectively).

Using the differential time series, it is possible to compute the crosscorrelation between each pair of units. Some of the resulting crosscorrelograms were significant, suggesting some kind of temporal relation in the firing patterns of the investigated *SI*-units. Comparing figure 4.13b with figure 4.14b for 30 and 60 mV respectively, we observed that the correlation between units #1234 and #7794 was amplified with the stronger stimulus. At the same time, the correlation between

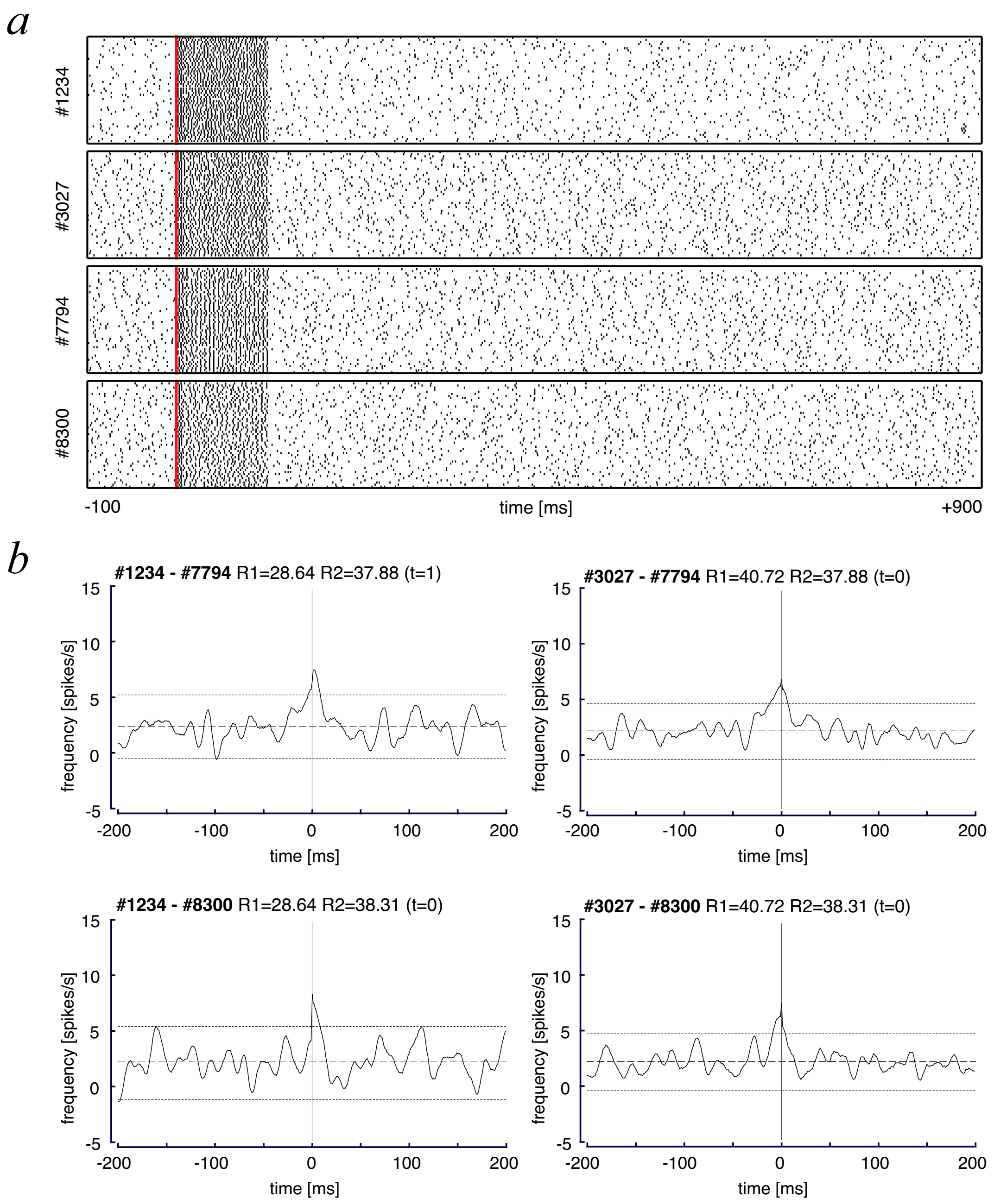


Figure 4.13: Effect of the 30 mV stimulus intensity. 4 sample units were selected from the emerged circuit presented in figure 4.8. ***(a):*** raw raster plots for the 50 last presentations of the stimulus; ***(b):*** crosscorrelograms based on the differential time series, like figure 4.12c, interrupted line: mean activity, dotted lines: confidence limits according to Abeles (1982b), Gaussian smoothing bin: $5ms$. Compare with the 60 mV stimulus, figure 4.14. Network size: 100 × 100; background activity: 5 sp/s; stimulus: simple; stimulus intensity: 30 mV; stimulus duration: 100 ms; input units: 10%.

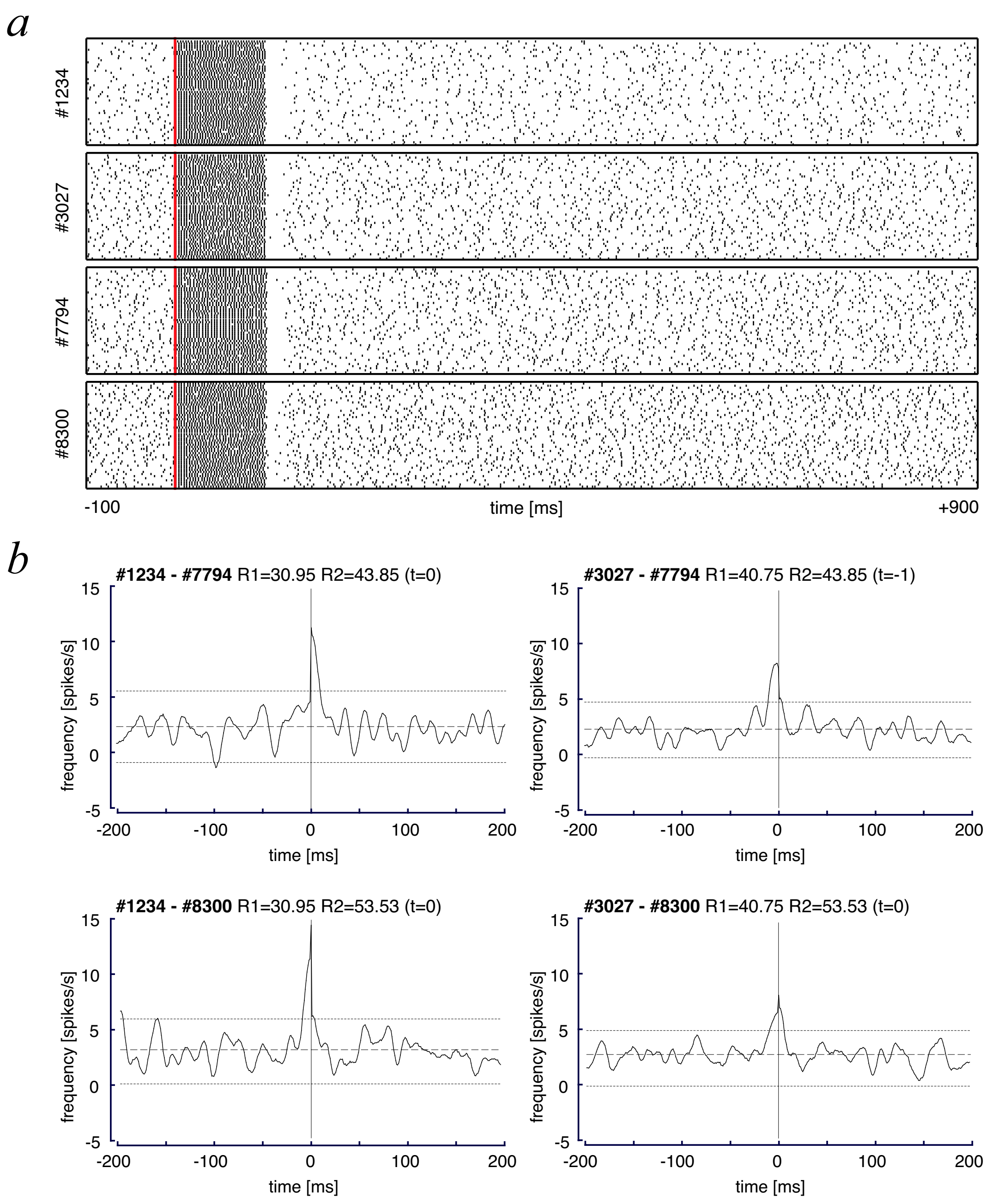


Figure 4.14: Effect of the 60 mV stimulus intensity. ***(a):*** raw raster plots for the 50 last presentations of the stimulus; ***(b):*** crosscorrelograms based on the differential time series, like figure 4.12c, interrupted line: mean activity, dotted lines: confidence limits according to Abeles (1982b), Gaussian smoothing bin: $5ms$. Compare with the 30 mV stimulus, figure 4.13. Note the stimulus offset inhibition. Network size: 100×100; background activity: 5 sp/s; stimulus: simple; stimulus intensity: 30 mV; stimulus duration: 100 ms; input units: 10%.

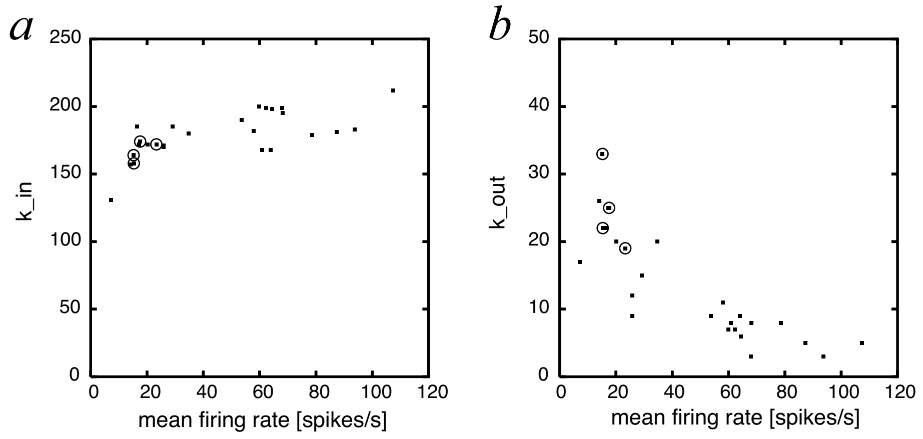


Figure 4.15: Connectivity indexes for 26 *SI*-units at $t = 500$ seconds vs. mean firing rates ρ between $t = 450$ and $t = 500$ seconds. ***(a):*** k_{in} vs. ρ; ***(b):*** k_{out} vs. ρ. $\cdot$: values for one *SI*-unit. The four units surrounded by *circles* are part of the spatiotemporal pattern of activity shown in figure 4.16. Network size: 100×100; background activity: 5 spikes/s; stimulus: complex; stimulus intensity: 60 mV; stimulus duration: 100 ms; stimulus protocol: AB|BA; input units: 10%.

units #1234 and #8300 was reversed with the stronger stimulus compared to the weaker.

4.2.4 Spatiotemporal pattern of activity

parameters summary

network size	100×100
background	5 sp/s
input units ratio	10%
stimulus pattern	complex
stimulus duration	100 ms
stimulus intensity	60 mV
stimulus protocol	none, AA, BB, AB, BA, AB|BA

The last batch of simulations was performed with the complex spatiotemporal stimulus protocols defined in section 3.4.2, p.26. All protocols were run a first time with the same initial conditions and random generator seed. The networks that emerged from the initial random connections through pruning were analysed at time $t = 500$ seconds. From the excitatory units that were not directly stimulated, we extracted 26 units that had $k_{in} = k_{out} = 0$ in absence of stimulus (none stimulus protocol), but maintained strong input and output *exc–exc* connections under every other protocol (AA, BB, AB, BA, AB|BA). These 26 units were recorded for the complete duration of a second run of simulations. We restarted the simulations for each protocol with the same initial conditions and random generator seed as previously, but from time $t = 500$ seconds on, the simulations were continued in two directions. In the first case, the STDP-driven pruning was maintained as before (continuous pruning), and in the second case, synaptic pruning was discontinued, i.e. the pruning stopped after the first 500 seconds (interrupted pruning).

The recordings were chopped into 50 seconds chunks, and analysed separately. We observed

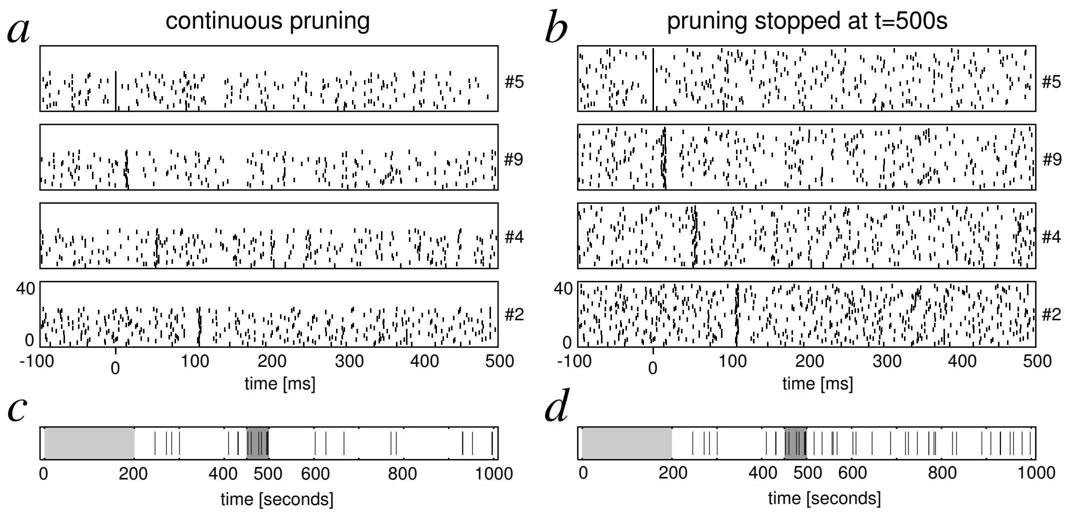


Figure 4.16: Sample spatiotemporal pattern of activity $< 5, 9, 4, 2; 13, 53, 109 >$. ***(a, b):*** raster plots of the spatiotemporal pattern $< 5, 9, 4, 2; 13, 53, 109 >$ aligned on the first spike of the pattern. Full abscissa scale is 600 ms. ***(c, d):*** strip representing 1,000 seconds of simulated time, with ticks marking the timing of the first spike of the pattern. The pattern ($n = 6$ occurrences) was significantly detected in the interval $450 - 500$ seconds (dark shaded area) and then searched throughout the simulation run starting from $t = 200$ seconds. 8 occurrences were observed in the interval $200 - 450$ seconds. We observed 10 additional occurrences during continuous pruning ***(a, c)***; 26 additional occurrences if pruning stopped after 500 seconds ***(b, d)***. Network size: 100×100; background activity: 5 spikes/s; stimulus: complex; stimulus intensity: 60 mV; stimulus duration: 100 ms; stimulus protocol: AB|BA; input units: 10%.

a negative correlation between the firing rate ρ and the number of outgoing connections k_{out} between $t = 450$ and $t = 500$ seconds (see figure 4.15b). Units with large k_{out} tended to have weaker firing rates than units with comparable k_{in} but lower k_{out}. Some units displayed unrealistic mean firing rates up to $\rho = 110$ spikes/s.

From the group of 26 units, we arbitrarily chose the 11 units with the smaller mean firing rates ($\rho \leq 40$ spikes/s), and their time series were searched for spatiotemporal patterns of activity between $t = 450$ and $t = 500$ seconds (Tetko and Villa, 2001c). We analysed the different stimulation protocols separately, and no significant difference could be found. For this reason, table 4.2, p.49 summarizes all the patterns that were found in all the recordings with a maximal duration of 200 ms, a jitter of 3 ms around each pattern spike, and a level significance of 0.1.

Most patterns (12 out of 14) were composed by the activity of a single unit (different units = 1), like the triplet $< 5, 5, 5; 92, 156 >$. On 21 occasions between $t = 450$ and $t = 500$ seconds during the AB|BA protocol, unit #5 spiked a first time followed 92 ± 3 ms later by a second spike, and a third spike 156 ± 3 ms after the first. Two quadruplets were composed by spikes

	number of different units N	number of different patterns N	number of pattern repetitions $\hat{N} \pm SEM$	pattern duration $[ms]$ $\hat{t} \pm SEM$	shortest interval $[ms]$ $\hat{t} \pm SEM$	longest interval $[ms]$ $\hat{t} \pm SEM$
triplet	1	4	34.5 ± 9.6	128.8 ± 22.2	47.5 ± 8.9	82.0 ± 15.4
quadruplet	1	8	31.3 ± 6.6	149.5 ± 11.3	22.6 ± 2.2	84.0 ± 8.8
quadruplet	4	2	10.5 ± 3.5	119.5 ± 10.5	21.0 ± 8.0	58.5 ± 2.5

Table 4.2: Summary of all detected spatiotemporal patterns of activity for 11 selected units across all protocols and conditions between $t = 450$ and $t = 500$ (see text). SEM stands for "standard error of the mean". Confidence level: 0.1; jitter: 3 ms; maximum pattern duration: 200 ms.

produced by four different units, like the pattern $< 5, 9, 4, 2; 13, 53, 109 >$. On seven occasions between $t = 450$ and $t = 500$ seconds during the AB|BA protocol, unit #5 spiked 13 ± 3 ms before unit #9, 53 ± 3 ms before unit #4, and 109 ± 3 ms before unit #2. The four units composing this pattern are encircled in figure 4.15. Between $t = 450$ and $t = 500$ seconds, 3.86% of all spikes of the 11 units were detected as being part of a spatiotemporal pattern.

The detected patterns were then searched in both continuous and interrupted pruning conditions between $t = 200$ and $t = 1,000$ seconds. The results of the quadruplet pattern described previously are shown in figure 4.16. After the pruning was stopped at $t = 500$ seconds, 26 additional occurrences appeared between $t = 500$ and $t = 1,000$ seconds (figure 4.16b,d), while only 10 could be found when pruning continued after $t = 500$ seconds (figure 4.16a,c). It is possible that after $t = 500$ seconds, the pruning process removed by chance one or more connections that were necessary for the production of the spatiotemporal pattern.

What I find helps me
to understand what I seek for.

– Pablo Picasso

Chapter 5

Discussion

Résumé Les résulats présentés au Chapitre précédent sont discutés ici à la lumière des travaux réalisés par d'autres groupes. Tout d'abord, le point est fait sur les problèmes liés à la plasticité synaptique et les synapses à états finis. Les propriétés émergentes dans les réseaux de neurones sont ensuite abordées, avant de revenir sur la structure des circuits mis en évidence par le pruning. Après quelques considérations autour de la simulation de l'élagage synaptique pendant le développement, les problèmes rencontrés avec les interactions entre les taux de décharge élevés de certaines unités et la plasticité synaptique à modulation temporelle relative (STDP) sont discutés, avant de terminer sur les "synfire chains".

Massive synaptic pruning following over-growth is a general feature of mammalian brain maturation (Rakic et al., 1986). Pruning starts near time of birth and is completed by the time of sexual maturation. Trigger signals able to induce synaptic pruning could be related to dynamic functions that depend on the timing of action potentials. Spike-timing-dependent synaptic plasticity (STDP) is a change in the synaptic strength based on the ordering of pre– and postsynaptic spikes. The relation between synaptic efficacy and synaptic pruning (Chechik et al., 1999; Mimura et al., 2003) suggests that the weak synapses may be modified and removed through competitive "learning" rules. This plasticity rule might produce the strengthening of the connections among neurons that belong to cell assemblies characterized by recurrent patterns of firing. Conversely, the connections that are not recurrently activated might decrease in efficiency and eventually be eliminated.

The main goal of our study is to determine whether or not, and under which conditions, such cell assemblies may emerge out of a locally connected random network of integrate-and-fire units distributed on a 2D lattice receiving background noise and content-related input organized in both temporal and spatial dimensions. The originality of our study stands on the relatively large size of the network, 10,000 units, the duration of the experiment, 10^6 time units – one time unit corresponding to the duration of a spike – and the application of an original bio-inspired STDP modification rule compatible with hardware implementation (Eriksson et al., 2003).

5.1 Synaptic plasticity

We assumed a number of simplified hypotheses about the presence of only two types of units, their leaky integrate-and-fire dynamics, their distribution and the dynamics of the transfer functions of the synapses that connect them. With all these assumptions, we observed that the network reached a steady state when the synaptic weights were either incremented to the maximum value or decremented to the lowest value (see figure 4.4). Our result is in agreement with the bimodal distribution of synaptic strengths observed with different STDP-based models (Chechik et al., 1999; Abbott and Nelson, 2000; Dayan and Abbott, 2001). This effect is interpreted as the effect of the STDP rule leading presynaptic neurons to compete for the capacity to drive the postsynaptic unit to produce an action potential. This competition has been shown to maintain the average neuronal input to a postsynaptic neuron (Abbott and Nelson, 2000; Song and Abbott, 2001).

There is increasing evidence for a biologically plausible function to state synapses (see Montgomery and Madison (2004) for a review). The synapse model we use here was not based on these experimental results. Our discrete activation level states simplification was required for the hardware implementation discussed later (section 7.2.1, p.109). They could be interpreted as a combination of two factors: the number of synaptic *boutons* between the pre– and postsynaptic units and the changes in synaptic conductance. Nevertheless, the experimental and theoretical work undertaken around the state synapses will certainly help us to refine our simplification and maintain a degree of biological plausibility in the future. Among other questions, if NMDA receptor regulation proves to be state-dependent, this would provide a metaplastic aspect (Abraham and Bear, 1996) to state-dependent synaptic plasticity.

Metaplasticity is defined as the plasticity of synaptic plasticity. A cascade model has been recently proposed (Fusi et al., 2005) as a theoretical mechanism for the storage of everyday

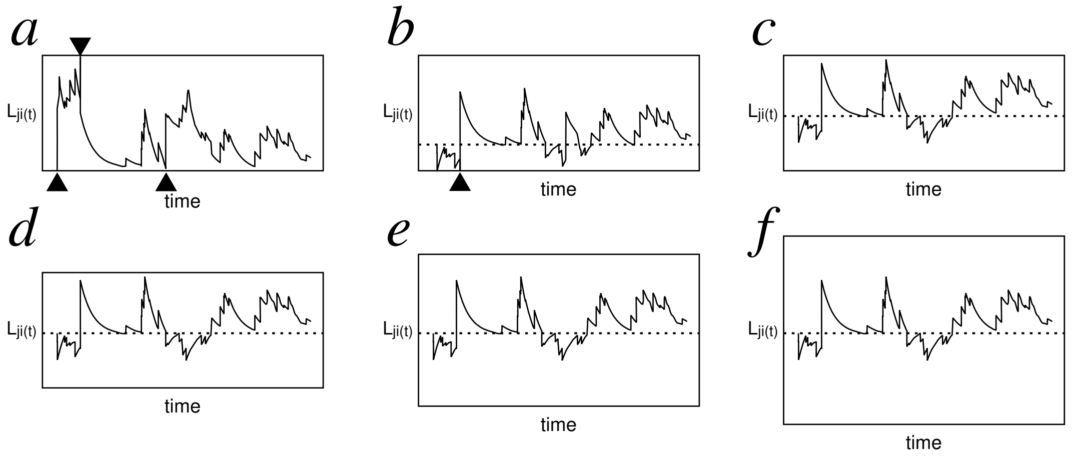


Figure 5.1: Extending the model towards simulating continuous childhood to adolescence to adulthood transitions. A sample evolution of $L_{ji}(t)$ as a function of time in a representation similar to figure 3.5, p.23. Each time $L_{ji}(t)$ outcomes the upper bound (*down-pointing triangles*), or the lower bound (*up-pointing triangles*), $A_{ji}(t)$ state changes accordingly. ***(a,b,c):*** continuous transition from pruning to learning; ***(d,e,f):*** introduction of a metaplastic aspect to the model. ***(a):*** original model presented here, $L_{ji}(t)$ decreases towards the lower bound L_{k-1}; ***(b):*** sample transitory state between (a) and (c); ***(c):*** $L_{ji}(t)$ is attracted by the center of the class, inducing a more stable acquired activation level; ***(d):*** same as (c) to serve as the starting point for the introduction of metaplasticity; ***(e):*** sample transitory state between (d) and (f); ***(f):*** a larger ΔL_k makes it gradually more difficult for $L_{ji}(t)$ to outcome the upper or the lower bound. Note that in this example, metaplasticity is introduced following the transition from pruning to learning, while these transitions could be performed separately, synchronously or in the reversed order. See text for a discussion.

experience memories. Acquiring such memories, like single-trial learning, requires a high degree of plasticity but retaining these memories requires protection against changes induced by further activity and experience. The authors defined two synaptic strength levels ("weak" and "strong") characterized by different degrees of plasticity. Each level is associated with a cascade of n states that introduce a range of transition probabilities between the two synaptic levels. Moving from one state to the next does not modify the synaptic strength, but lowers the probability to switch synaptic level, hence the metaplasticity.

Note that metaplasticity is inherently related to learning. In the present work, we simulated activity-driven synaptic pruning during a development experiment. Nonetheless, a slight change to our model could provide a gradual transition from pruning to learning, keeping the same synaptic principle, allowing to simulate childhood to adolescence to adulthood transitions (see figure 5.1). In section 3.3.1, p.20, we have seen that a continuous internal variable $L_{ji}(t)$ was used to control the discrete activation level $A_{ji}(t)$ using STDP. User-defined boundaries $L_0 < L_1 < \cdots < L_{N-1} < L_N$ were used to determine the transitions between activation levels states, with a slow decaying factor driving $L_{ji}(t)$ towards the lower boundary leading to synaptic pruning (figure 3.3c). This rule could be changed continuously, as in figure 5.1a-c, with a transition towards a relation between $L_{ji}(t)$ and $A_{ji}(t)$ where the decaying factor drives $L_{ji}(t)$ to $\frac{L_k - L_{k-1}}{2}$ (figure 5.1c) instead of driving it to L_{k-1} (figure 5.1a) (Eriksson et al., 2003). In this situation, synapses tend to stay stable at their acquired activation level, and learning becomes possible. Such a rule has been shown to be capable of maintaining memories in the presence of noise (Fusi, 2001). Here, we arbitrarily decided that $\Delta L_k = L_k - L_{k-1} = 20$ for any attractor state $[A_k]$. In addition to the transition proposed above, modifying ΔL_k as a function of the time spent in a particular state $[A_k]$, like in figure 5.1d-f for example, would introduce a metaplastic aspect to our model. Indeed, the larger the ΔL_k, the more stable will be the acquired activation level.

It was reported recently that both in hippocampal cultures and acute hippocampal slices, the synaptic strength of GABAergic synapses could be persistently modified through repetitive postsynaptic spiking within 20 ms before and after the activation of the synapse (Woodin et al., 2003). The detection and modification of these synapses by coincident pre– and postsynaptic spiking allows the level of inhibition to be modulated according to a spike timing rule. Such a modification of inhibitory-excitatory and inhibitory-inhibitory connections has not been considered in our simplified model. In the beginning of the simulations, depending on the conditions, we observed periods of saturated network activity while the network is transiently composed of a large number of synapses in the strongest activation level (see figure 4.4a around first stimulus presentation). This suggests the need for stronger inhibition during that period to avoid unrealistic periods of high activity. As the simulation runs, and the excitatory-excitatory connections are pruned from the network, the strength of inhibitory connections should probably be adapted to maintain a dynamic balance between excitation and inhibition.

5.2 Network size effect

The choice of a 2D lattice topology allowed us to study the effect of incrementing the network size from 10×10 to 100×100 units. It was interesting to observe that the final number of synapses that remained active was below 10% of the number of initially active synapses. This number varied only slightly with changes in network size. The effect of different random number generator seeds was also limited. We observed that the number of active synapses characterized by the maximum strength could transiently reach a proportion as high as 50% of all initial synapses.

The existence of a scaling factor to adjust the activity of the network as a function of the network size could indicate that a very large network may not be necessary for recurrent networks to emerge. This result is in agreement with several findings of the theory of dynamic similarity (MacGregor et al., 1995). Interconnected sizable "modules" up to 50×50 or 60×60 units embedded in larger networks may offer a more efficient way to recruit active synapses that compete for generating a postsynaptic spike. The question was raised whether the ability of large collections of neurons to perform "computational" tasks may in part be a spontaneous collective consequence of having a large number of interacting simple neurons (Hopfield, 1982). The observation that most of the cerebral cortex is composed of local circuits, the cortical columns (Douglas and Martin, 1991) and hypercolumns (Sur et al., 1980), with well-defined functions suggests that the bridge between simple circuits and the complex computational properties of the brain may be the spontaneous emergence of new computational capabilities from the collective behaviour of large numbers of simple processing elements.

5.3 Circuit emergence

A bias in the geometrical orientation of the synapses might produce important effects on the global dynamics as it could introduce singularities in the network topology. These singularities could sustain attractors with unbalanced excitatory/inhibitory inputs if they were the consequence of content-related inputs. In the presence of only background activity, these attractors would be spurious and could mask input-related features. We observed that the reinforcement of few synapses occurred without geometric distortion either in direction and or in source-to-target distance over the 2D lattice. Synaptic pruning proceeded in a homogeneous and isotropic way across the network. This result suggests that, in the presence of mere random background activity, the implementation of the current STDP rule is equivalent to random pruning and does not introduce spurious biases.

We observed that a recurrent spatiotemporal stimulation pattern, in the presence of background activity, could induce the emergence of oriented cell assemblies when associated with STDP-driven pruning. We observed that the unsupervised pruning process, associated with short and stable stimulation patterns, tended to organize units in strongly interconnected feed-forward assemblies on top of the input units. However, the emergence of the diverging projections was much more difficult to observe than the convergence.

The detection of complex spatiotemporal patterns in the simulated firing activity suggests that layered topologies may appear during our pruning process. Synfire chains (Abeles, 1991) are diverging / converging chains of neurons discharging synchronously to sustain the propagation of the information through a feed-forward neural network. This theoretical model has proven to be very efficient for the transmission of precisely timed information through the cerebral cortex, but the mechanisms that may underlie its appearance in the mature brain have never been deeply investigated. Some works have used the Hebbian learning rule (Bienenstock, 1995; Hertz and Prugel-Bennett, 1996a,b), or STDP (Levy et al., 2001; Kitano et al., 2002) to let synfire-like structures emerge out of relatively small randomly generated networks. The resulting assemblies were composed of a few (3-4) groups of neurons firing synchronously in loops, but the topological distribution of the connections was not discussed.

An alternative approach to the emergence of selected circuits has been proposed by Quenet et al. (2005). They formally described networks of multistate neurons connected by multidimensional synapses which allow the emergence of selected spatiotemporal patterns of activity strongly associated with precise stimuli. Each neuron state corresponds to an activity state, like inactive, spiking, bursting, etc. A synapse between two neurons is not defined by its single efficacy but by a weight matrix. This allows us to take into account the facts that neurons can be in different states, that different types of synapses can be used, e.g. involving different neurotransmitters, and that multiple contacts might exist between the neurons in different locations on the dendritic tree.

The self-organization of spiking neurons into neuronal groups was recently described in a study featuring large simulated networks connected through STDP-driven projections (Izhikevich et al., 2004). 80,000 excitatory and 20,000 inhibitory spiking neuromimes (to compare with network size and distribution of our 100×100 2D lattice) were wired on a spherical surface by 8.5 millions connections (in our 100×100 2D lattice, the total number of connections in the beginning was about 3 millions) using a random local connectivity rule, plus one small long-distance connectivity spot for excitatory units. They studied the spatiotemporal structure of emerging firing patterns, finding that if axonal conduction delays and STDP were incorporated in the model and in the presence of noise, neurons in the network spontaneously self-organized into neuronal groups, even in the absence of correlated input. They also observed recurrent firing patterns appearing spontaneously among units that were located close to one another. Their approach is similar to ours, despite the fact that their simulations were not aimed at studying the synaptic pruning, but the emergence of neural groups. Time locked presynaptic spikes were able to fire postsynaptic units with precise timing due to the modeled axonal conduction delays. After the detection of spatiotemporal patterns of activity, they could determine the set of units producing this activity, and reconstruct the circuits. During the simulations, they could assist in the emergence, maintenance and disappearance of such circuits. On average, there were less than 30 units per circuits, and less than 7.5% of all the units were part of any such circuit.

The study by Izhikevich et al. (2004) emphasizes the importance of axonal conduction delays that we did not initially consider in our model. If spikes arrive at the postsynaptic unit at different times, then the simultaneous firing of presynaptic units is not the proper way to fire a postsynaptic unit (Bienenstock, 1995). Presynaptic units with matching conduction delays and firing in a certain temporal pattern can drive a postsynaptic unit to spike by time-locking the arrival of their spikes. Through a different firing pattern, the same presynaptic units could drive another postsynaptic unit. We are currently considering the addition of conduction delays to our model.

There is a trend towards the use of graph theory methods (see Albert and Barabasi (2002) for a review) for the analysis of neuroscientific data (Sporns et al., 2004). For example, these methods have been used for the analysis of neural connectivity patterns (Sporns, 2002). In particular, graph theory has been successfully applied to functional magnetic resonance imaging (fMRI) data for a finger tapping task (Chialvo, 2004), suggesting that functional networks are scale free, where the highly connected nodes are, on average, connected with highly connected nodes, a feature that is inverse to a hierarchical organization. However, the current status of graph theory has little to offer to problems involving dynamic oriented weighted graphs, like ours. We have been looking for a proper index to estimate the quality of circuit reconstructions like the one in figure 4.8. None of the standard indexes, like the clustering coefficient (the fraction of connections between the neighbours of a node with respect to the maximum possible) or the average path length (the minimum number of links necessary to connect two nodes), suited to the measure of a converging / diverging network. We are still trying to find an appropriate index for such a measure.

5.4 Synaptic pruning

Massive synaptic pruning during childhood after synaptic over-growth is an intriguing feature of the mammalian brain development. A few theoretical works were undertaken to investigate the computational advantage of such a seemingly wasteful developmental strategy. In a first theoretical work, Chechik et al. (1998) suggested that the memory performance of a network is optimally maximized if, under limited metabolic energy resources restricting their number and strength, synapses are first overgrown and then pruned. Two "adult organisms" with the same synaptic resources can store different numbers of memories depending on the way they acquired their adult synaptic densities. An organism with an excessive number of synapses that are selectively pruned can store more memories than another adult with fixed synaptic density during infancy. The simulations were performed with an Hebbian synaptic modification rule, and a minimal value deletion function to prune the weakest connections from the network. Such an algorithm was shown to maintain the network performance.

Neuronal regulation is an experimentally identified mechanism regulating the input synaptic strength to maintain the homeostasis of the neuron's membrane potential (Turrigiano et al., 1998). Horn et al. (1998) suggested that, theoretically, neuronal regulation might maintain the memory performance of networks undergoing synaptic degradation. In Chechik et al. (1999), the authors discussed the role of the neuronal regulation in maintaining the average postsynaptic input field, removing the weaker synapses and modifying the remaining synapses (see Mimura et al. (2003) for an analytical discussion of these results). STDP has also been shown to maintain the postsynaptic input field (Abbott and Nelson, 2000; Song and Abbott, 2001). For this reason, STDP might be a proper choice as a synapse modification rule for synaptic pruning simulation.

Neural Darwinism – or the theory of neuronal group selection (TNGS) – is a population theory of the nervous system aimed at understanding the significance of variation and selection in the brain development and function (see Edelman (1993) for a review). According to this theory, the world becomes perceptually categorized to an organism as a consequence of two interactive processes of selection upon variation. The first process occurs during embryonic and postnatal development, when adjacent neurons tend to be strongly interconnected in "assemblies" of variable size and structure called "neuronal groups". The second process consists of alterations in synaptic strengths following individual's activity, with a selection of the adaptive neuronal groups on the basis of their correlated responses. In Neural Edelmanism (see Crick (1989) for a criticism of the theory), the emphasis is put on the "competition" between the neuronal groups without referring to the mechanisms of synaptic pruning. We must recognize that the Neural Edelmanism theoretical framework produces many results that challenge our approach.

5.5 Effect of firing rate

The results presented here suggest that layered topologies compatible with synfire chains may appear during unsupervised pruning processes. However, it was not possible to determine the exact requirements to let the synaptic pruning mechanisms drive the emergence of balanced converging / diverging chains, i.e. with comparable k_{in} and k_{out}.

The strongly interconnected units (*SI*-units, see section 4.2, p.34) displayed mean firing rates in the range of 30 to 40 spikes/s. Besides the value being too high to be biologically plausible for a model of the cerebral cortex, we observed a negative correlation between the mean firing rate and the k_{out}. This is inherent in a standard STDP rule. As explained in figure 5.2a,b, a unit spiking faster than the others will tend to maintain all its inputs, and loose all its outputs. As a consequence, we may assume that due to excessively high firing rates, STDP tended to be too much affected by firing rate differences.

With this observation concerning the dynamics of STDP, a scenario can be drawn to explain the evolution of the connectivity indexes k_{in} and k_{out} as presented in figure 4.6. It is possible that during the first few time steps of the simulation, the excitatory units that were randomly connected from a large set of presynaptic excitatory units start spiking with a large rate, because

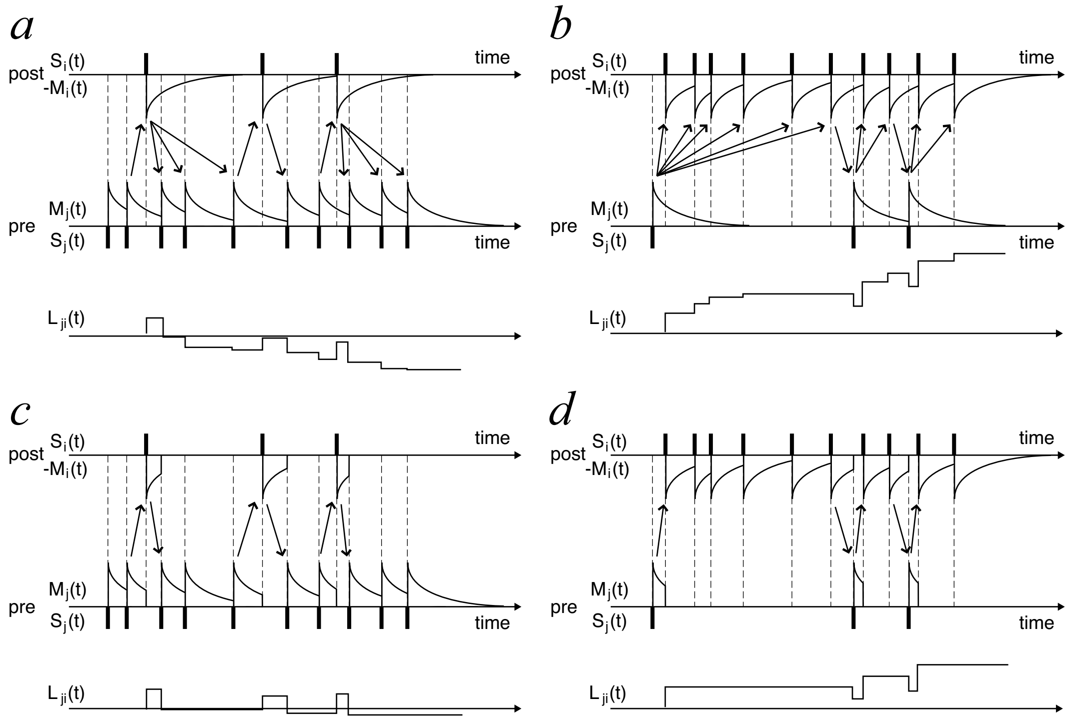


Figure 5.2: STDP and units spiking at different rates. Compare with figure 3.4. A fast spiking unit will tend to keep all the input connections, and loose all the output connections. ***(a):*** presynaptic unit spikes faster than postsynaptic unit. Multiple presynaptic spikes will integrate the decay of $M_i(t)$ resulting in the pruning of the connection; ***(b):*** postsynaptic unit spikes faster than presynaptic unit. Multiple postsynaptic spikes will integrate the decay of $M_j(t)$ resulting in the reinforcement of the connection; ***(c):*** considering only the first pair of spikes, the effect of a fast spiking presynaptic unit can be reduced; ***(d):*** considering only the first pair of spikes, the effect of a fast spiking postsynaptic unit can be reduced; Note that for the sake of clarity, the evolution of $L_{ji}(t)$ functions ignore the slow decaying term introduced in section 3.3.1, p.20, as $k_{act} >> k_{learn}$.

of the numerous inputs. Figure 5.2b suggests that these units will reinforce their inputs, thus inducing even more postsynaptic activity. At the same time, figure 5.2a suggests that the fast spiking units will start loosing all their outputs. Such a dynamic results in the unit running down the k_{out} axis in figure 4.6, corresponding to the pruning dynamics observed for almost all *SI*-units. This scenario provides an explanation for the excess of convergence over divergence in the emerged networks.

It has been observed in rat visual cortical slices preparation (Froemke and Dan, 2002), that the contribution of each pre–/postsynaptic spike pair to synaptic modification depends not only on the interval between the pair, but also on the timing of preceding spikes. The efficacy of each spike in synaptic modification was suppressed by the preceding spike in the same neuron, occurring within several tens of milliseconds. The authors suggest that the timing of the first spike in each burst is dominant in synaptic modification, with the additional spikes having only a marginal contribution. Based on this observation, an integrate-and-fire neuromime with an STDP-inspired learning rule for its first spike only was proposed in a recent theoretical study (Guyonneau et al., 2005). The spiking unit receiving a wave of input from 1,000 presynaptic units learned to react faster to a repeated input spike pattern. The postsynaptic spike latency tended to stabilize at a minimal value while the first synapses became fully potentiated and later ones fully depressed (the bimodal distribution of efficacies), implying that the STDP rule was able to focus on the spikes whose timing is reproducible and early, while discarding the rest.

Figure 5.2c,d shows how considering only the first pairs of spikes could modify the convergence/divergence proportion in a balanced way. With such a modified STDP rule, the impact of the fast firing units on the loss of output connections could be limited. Besides, avoiding a unit entering into the vicious circle of firing more, inducing stronger connections, inducing even more firing, could remove the problem of large firing rates altogether. If we plan to simulate larger networks with more connections, then introducing this modified STDP rule seems necessary.

5.6 Synfire chains

A recent addition to the original synfire chain description, as depicted in figure 2.3, is the role of the inhibitory neurons in stabilizing a network with embedded synfire chains. In a theoretical work, Aviel et al. (2004) emphasized the necessity for inhibition in a balanced network – a network where each unit receives equal excitation and inhibition – to avoid spontaneous ignition of embedded synfire chains, and to avoid that the rest of the network activity becomes saturated by the synchronous discharge of the synfire chain layers. The necessary balanced inhibition is obtained by wiring to each layer a *shadow pool* of inhibitory units that receive convergent inputs from the previous layer as if they were excitatory units. In addition, these inhibitory units do not project diverging connections to the following layer, but project locally within the network like ordinary interneurons. The necessity for such inhibition mechanism for the propagation of a wave of activity through a synfire chain opens the question of the role of inhibitory units in our simulations. We have observed balanced activity in our networks, even the appearance of an intensity-dependent stimulus offset inhibition (compare figures 4.13 and 4.14). This suggests that we might consider inhibitory units when looking for emerging circuits.

The analysis of the correlograms is often considered as a valuable tool for the deduction of functional connectivity (Abeles, 1982a). Figure 4.14b shows an asymmetrical peak near time zero which could be interpreted as a temporal correlation between units #1234 and #7794, that suggests a direct projection from unit #1234 to unit #7794. However, there is not such a projection in the actual topology (see figure 4.8b). The timing of that correlation could be explained by a systematic lag in the firing of #7794 with respect with #1234. Such a variability is nonetheless consistent with the synfire activity (Gewaltig et al., 2001). In any case, further investigation is required to determine if self-sustained synfire activity (Tetzlaff et al., 2004) may appear in the observed emerging circuits embedded in the complete network. The network state characterization proposed by Brunel (2000) could be of great help in this task, but the firing rate of the excitatory units should not exceed the levels incompatible with STDP.

Part II

(Neuro)Informatics

Programming today is a race between software engineers
striving to build bigger and better idiot-proof programs, and
the Universe trying to produce bigger and better idiots.

So far, the Universe is winning.

– Rich Cook

Chapter 6

Software

Résumé Ce Chapitre commente brièvement les nombreux logiciels qui ont été développés dans le cadre de ce travail, en mettant l'accent sur leur indépendance par rapport à la plateforme et leur interactivité avec l'utilisateur. La figure 6.1 donne un aperçu de l'organisation des deux chaînes distinctes de traitement de l'information : la première pour les circuits émergés par élaguage au cours des simulations (moitié supérieure de la figure) ; la seconde pour les séries temporelles résultant de l'activité des neurones (moitié inférieure). Les logiciels sont décrits par familles, selon qu'ils sont utilisés pour simuler le modèle (section 6.1, p.69), manipuler les données produites (section 6.2, p.78), analyser ces données (section 6.3, p.85), ou visualiser les résultats (section 6.4, p.88).
Les séries temporelles multivariées qui ont fait l'objet de ce travail ont été produites par des simulations, mais le traitement peut également être appliqué à tout enregistrement électrophysiologique *in vivo*, voire à tout processus qui peut être représenté par une série temporelle.
Les programmes présentés sont construits sur des librairies logicielles libres pré-existantes validées par leurs communautés d'usagers respectives. Ils sont tous disponibles sous la GNU Public License (GPL). Toutes les informations nécessaires à leur téléchargement, leur compilation et leur exécution se trouvent regroupées dans l'Appendix A, p.115.

Simulating large spiking neural networks implies a large computational power and several pieces of software. This Chapter is dedicated to the description of the different programs that were necessary to simulate, analyse, and visualize our model. The next Chapter, starting at page 103, is dedicated to the hardware sustaining our computational needs.

It appeared clearly since the beginning of the work that, in order to perform the simulations, we would need to rely on several computers, and that the whole software setup would need to scale up with our modeling needs. In order to exploit at best the available hardware substrate, the software had to be portable. It had to be able to run on GNU/Linux, Apple MacOS X, as well as on Microsoft Windows computers. No programming language or platform suits all needs. We used three approaches to produce portable code: Sun's Java programming language, ANSI C code with GNU platform libraries, as well as interpreted scripting languages like Perl, Python, bash or PHP.

The Java programming language relies on the existence of a virtual machine for the underlying operating system (OS), removing the need to compile for a specific OS by compiling into the virtual machine language. We used this strategy mainly for interactive programs, for the convenience of the graphical user interfaces and the interactive features of the graphical tools.

ANSI C code was written for the GNU platform, providing portable libraries and tools (like compilers and linkers) for all the targeted platforms. Note that the Windows platform, lacking some of the most interesting concepts of Unix, required the use of a GNU environment like Cygwin[1] or mingw[2]. We used this strategy for programs that were generating, analysing or manipulating the data.

The glue between the different pieces of software was implemented using the different scripting languages enumerated before, resulting sometimes in short-span *ad hoc* solutions.

The software had to run on different Operating Systems, but it also had to interoperate and exchange information between the different platforms, operating systems, and programming languages. For this reason, text-based file formats were preferred to binary ones. XML, one of the most versatile by-products of the web, was used for structured information.

The reuse of components available from identified sources has been emphasized, as well as producing reusable components. A large amount of work was saved by collecting the fruit of the work of many programmers all over the web into a set of modular applications. All programs and applications were written in a modular way, opting for all kinds of dynamical linking to modules instead of monolithic architectures. This approach required more thought, planning and programming at the beginning of the project. Relational and object databases were used in multiple places of the complete system, as a persistent storage, but also as a volatile, programmable data structure.

Beside the existence of portable environments like the Java platform or the GNU project, this work was made possible by the availability of an amazing amount of software written by clever and dedicated programmers and distributed under one of the many different Open Source licenses. We are indebted to them collectively as they made this work possible. As a

[1]`http://www.cygwin.com/`
[2]`http://mingw.sourceforge.net/`

demonstration of our gratitude, and knowing that the Open Source scheme is the only possible way for scientific programming, the software produced during this work is itself available on the web, under the GPL licensing scheme. Feel free to use, correct and expand the code base. Details concerning how to obtain, compile and run the software presented in the following pages have been gathered into Appendix A, p.115.

Figure 6.1 presents a structured view of the programs and applications further described in this Chapter. All software pieces are organized in four parts: simulation, data manipulation, data analysis, and visualization of the results. The Chapter is organized around these four parts, one section for each part. An additional section (section 6.5, p.97) is devoted to file formats, with emphasis on those defined along the software developments, and on established file formats that have been used.

When simulating models, odds are that none of the existing software precisely suits your needs. The downstream manipulation and analysis of the simulated results will require either a lot of time-consuming human hand-work or a set of home-made tools. Our philosophy was to hand-craft programs, relying as much as possible on existing frameworks and tools, and to design them to be as flexible and portable as possible to minimize development effort and, possibly, ease reuse of the software in other groups or even other fields. OAN (see section 6.3.2, p.87) and `XY-Viewer` (see section 6.4.2, p.89) are two showcases of this philosophy.

In this Chapter, we will explore the different pieces of software, with the objective to stress some of the unusual or innovative uses of existing programs and concepts, and how they can contribute to acquire insight into spiking neuron modeling even outside of the present research. Source code and programming interfaces (API) are volatile, they do not belong on a permanent media. Nevertheless, should the reader be interested in those elements, they are accessible from the web, as explained in Appendix A, p.115.

6.1 Simulation environment

Our approach required the simulation of a large number of interconnected integrate-and-fire neuron models. The model we described in section 3.2, p.18 is compatible with a hardware implementation that was not available at the time we started our research. To circumvent the

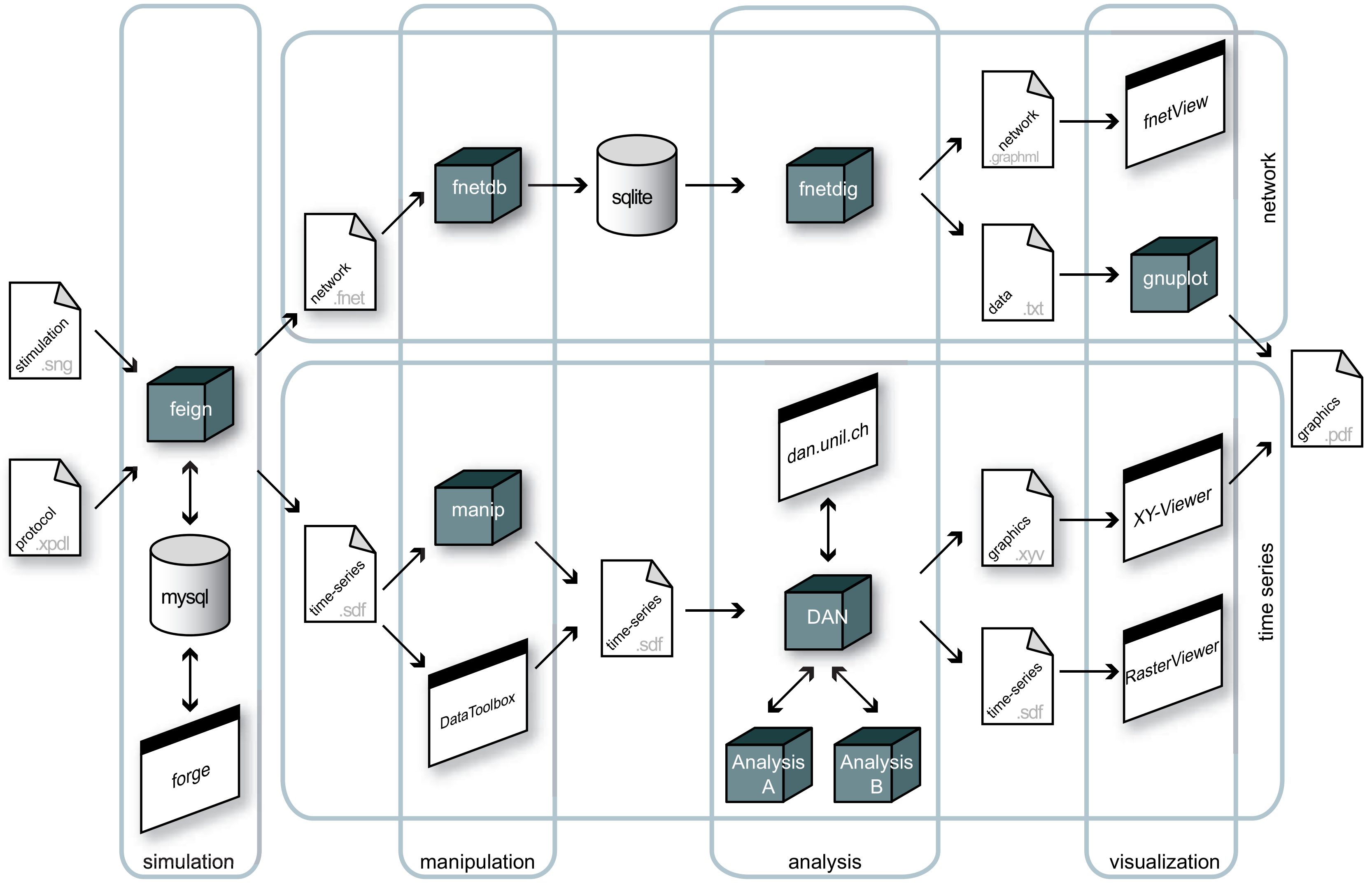


Figure 6.1: Overview of all software pieces. Programs are shown as cubes (black boxes), applications as stylized “windows”, and databases as cylinders. Software is logically organized in four parts: simulation (section 6.1, p.69), data manipulation (section 6.2, p.78), data analysis (section 6.3, p.85), and results visualization (section 6.4, p.88). Read dedicated sections for details on each part. Data formats are discussed in section 6.5, p.97. Gnuplot, MySQL and SQLite are existing projects that were reused.

computing power problem, we turned to the cheapest architecture available: a Beowulf-class cluster of PCs (see section 7.1, p.104). The next problem to solve was to decide which simulator to run.

The two most commonly used simulators in the field of computational neuroscience are Neuron[3], a project lead by Michael Hines; and GENESIS[4], lead by Dave Beeman. The main difference between the two is the model programming language (GENESIS being object-oriented). Still both were specifically designed to solve the equations that describe nerve cells based on their ionic currents. XNBC[5], a project lead by Jean-François Vibert, offers a rich set of neuron models, some of them phenomenological. It is slightly different from the previous two by its physiological/biophysical approach, providing extended possibility to interactively apply drugs for example. These simulators, and a few others, come with a large number of interactive tools to specify and control the simulations. For our needs, they were consuming computer resources focusing on details of the model we were not interested in.

The NEST Initiative[6], coordinated by Marc-Oliver Gewaltig, is an evolution of the (now outdated) *synod* simulator showing great promise. The approach is closer to ours, centered on large numbers (up to 10^6) integrate-and-fire model neurons. The simulation environment has been efficiently parallelized for multiprocessor computers as well as distributed architectures (Morrison et al., 2005). At the time we started our research, this software was not available, either. Having checked those options, and others, we decided to develop our own simulator, adhering closely to the evolution of our needs in relation to our distributed setup.

6.1.1 `feign`: a spiking neuron simulator

`feign` is a modular simulation framework aimed at providing the basic functionalities that are common to different neural network simulations, with hooks to implement the model-specific parts of the simulation. It does not provide a language for the models description. `feign` is implemented in ANSI C. All details on getting, and compiling the simulator and the documentation are available in Appendix A, p.115.

As sketched in figure 6.2, the main core of `feign` gathers modules and services together.

[3]`http://www.neuron.yale.edu/`
[4]`http://www.genesis-sim.org/`
[5]`http://www.b3e.jussieu.fr/xnbc/`
[6]`http://www.nest-initiative.org/`

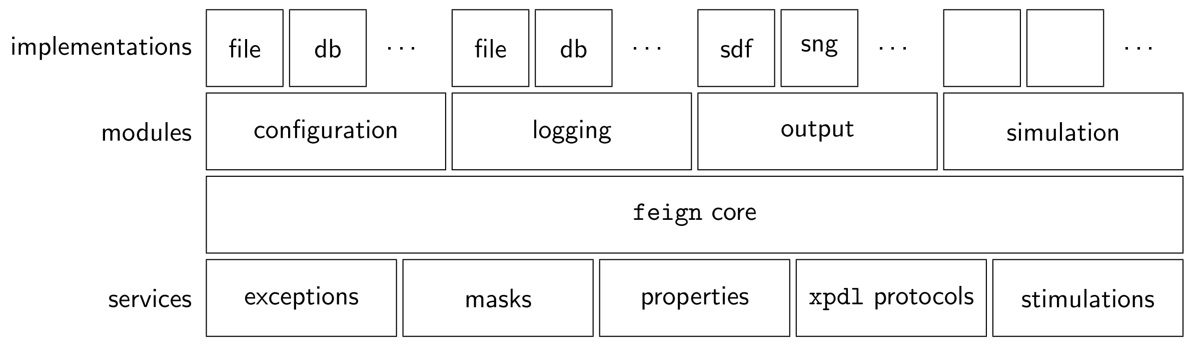


Figure 6.2: Layer-cake diagram sketching `feign` logical split into a core providing services to modules implementations. Modules and services are defined by their interfaces. Only one implementation of every module can be compiled in the final executable. Read text for details on modules and services.

Modules code for the behaviour of the simulations. Services are a set of generic functionalities available for the modules implementations at runtime.

The details of a simulation are scattered in several configuration files: model constants, network connectivity statistics, experimental protocol, stimulation files, ... are separated entities that can be mixed and reused in different ways to produce slightly different simulations.

6.1.1.1 Modules

Modules are portions of the program that can be implemented in different ways. The user chooses one implementation for each module that will be compiled into the program and used during the simulation. Modules are not linked dynamically at runtime to avoid the measurable overhead cost. Each of the four modules has a short interface (set of functions) that is defined in appropriate header files. The interfaces were designed to minimize the dependences between the modules. There can be many different implementations for each module, but only one of them is statically compiled into the executable.

configuration module: loads a configuration from some source into the properties service described later. It was expected that each simulation run would have a slightly different configuration. Currently, two implementations were written for this module. The first uses a text-based file format. The second collects the necessary information from a relational database. See section 6.1.2, p.75 for more information on database usage.

logging module: keeps trace of what happens and at which speed. Two types of information

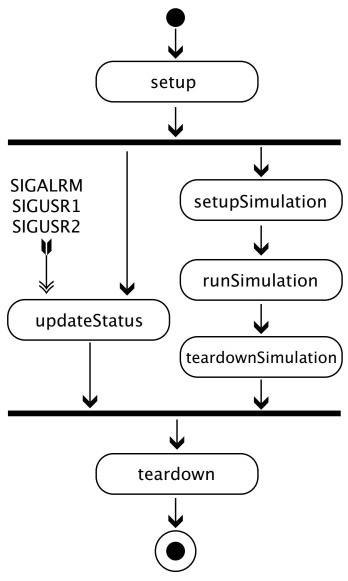


Figure 6.3: UML activity diagram. Program execution runs from top to bottom. Left path: relying on ANSI C alarms and signaling features, the status of the simulation is periodically written to the log; Right path: the sequence of simulation module specific functions corresponds to the most time consuming code of the execution.

are logged: the status of the simulation and the messages. Status is stored periodically by `feign` as shown in figure 6.3, while messages can be logged at any time from any portion of the code. Saved logs contain information concerning simulations that might be used later for debugging, forensics (in case of crash), or as reference. Currently, three implementations were written for this module. The first does not store logs at all, throwing away every information it receives. The second uses a text-based file format. The third writes log information into a relational database. See section 6.1.2, p.75 for more informations on database usage.

output module: saves the multivariate time series for the simulated neuron spiking times. Storing the activity of some or all spiking units of the simulated network leads to the analysis of their time series. Currently, three implementations were written for this module. The first does not store time series at all, throwing away all information it receives. The second saves events into the spike data format (see section 6.5.4.2, p.101) used for time series analysis as discussed later in this Chapter. The third saves events into one `sng` image per time step (see section 6.5.4.3, p.102) for visual inspection.

simulation module: contains the neuron model and network description to be simulated. The purpose of all the other modules and services is to help this module to perform its task.

Currently, two implementations were written for this module. The first does not simulate anything and is mainly used for testing. The second is the implementation of the model discussed in Part 1 of this document.

6.1.1.2 Services

Some general purpose features were required by several modules. They are labeled *services*, isolated into their own interfaces and exposed to the modules by the core. Services are unconditionally compiled in `feign` and available for all module implementations.

exceptions service: allows any portion of the code to gracefully stop the execution of the program by throwing a logged exception. This functionality is consistent with the philosophy of the simulator: any problem should be reported to the user and no simulation should be run in an improper state. This service is implemented using the *long jump* functionality of ANSI C.

masks service: defines vectors of *yes* and *no* that answer to runtime questions concerning neuromime units and time steps like "shall unit #7438 be recorded at time step 22,343?" or "is unit #445 to receive stimuli at time step 12,703?" This service is also used to determine the duration and composition of the spiking activity recordings.

properties service: stores configuration settings at runtime. All the settings for the simulator and the modules are collectively known as properties, i.e. key-value pairs. They are used by the different modules to determine their configuration and behaviour. There is only one properties object per `feign` process, such that properties name clashes might appear. There are three different ways to set these properties:

- at compilation time. Default property values can be hard-coded in the source files.
- at runtime. Multiple command line arguments *-Pkey=value* can be used to set properties at start-up time. This method is mainly used to set the properties required by the configuration module implementation.
- through a configuration module implementation. Many ways can be conceived to define properties (from flat files, databases, environment variables, . . .) The config-

uration module interface has been designed to let developers write their own implementations.

`xpdl` **experimental protocols service:** defines (stochastic) sequences of steps to follow during the simulation run. `xpdl` stands for eXtensible experimental Protocol Description Language. It is an XML grammar that describes the protocols to follow as a discrete state machine. It is extensible in the way that the language describes the transitions between the machine states as sequences, loops, and branches, but has no limitations on the complexity of the states that are left to the programmer's choice. Details concerning the `xpdl` data format can be found in section 6.5.2, p.98. Several protocols can be loaded for the same `feign` process, i.e. the implementation is *reentrant.* Protocols are used to determine simulations sequences and timing. They can also be used to temporarily switch on or off the STDP synaptic plasticity.

stimulator service: applies an arbitrarily complex stimulus on the simulated network. The stimulus details are described in a sequence of `sng` images (see section 6.5.4.3, p.102). The use of an image format to describe stimulus enables simple visual inspection, like the ones available from Appendix A, p.115.

6.1.2 `forge`: a simulation organizer

Organizing and keeping track of the thousands of simulations that were required for the present work would have been difficult without the help of an assistant. It appeared early in the work that editing, checking and comparing the simulation configurations were difficult tasks. They usually only differed from a couple of lines and could be easily interchanged. The distributed nature of the simulation environment required a centralized persistent location where configurations and logs could be stored for each simulation for later reference. A MySQL database server accessible from the network was a suitable answer to this problem. `feign` configuration and logging modules were implemented to optionally access that database instead of flat text files. Thanks to those implementations, the simulation status is periodically sent to log (see figure 6.3) along with the percentage of completion. Beside the persistent storage, this information can be used to dynamically monitor the ongoing simulation effort in a distributed way.

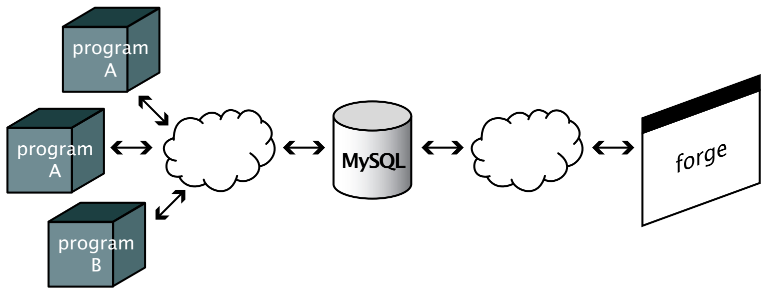


Figure 6.4: A sketch drawing showing how `forge` outsources the persistence and distributed concurrent access handling to a MySQL database. This clean design simplified the development process and removed any direct dependency between programs and `forge`.

`forge` is a light-weight, HTML based user interface on top of the database contents. It can be used to edit, copy and compare simulation configurations, and to monitor running simulations for progress from all over the world. `forge` is an unanticipated development that turned out to become a cornerstone of the system.

Using a MySQL database server as the persistent repository is like giving away the most complex problems to be solved. The database management system is in charge of handling the problems related to the concurrent access to the information that is inherent to such a decoupled design. It is also responsible for the consistency of the data.

The database design is fairly simple. It aims at expressing the concept of a `Program`, a piece of software written to communicate with the database. A `Program` has many `Executions` to be monitored and/or configured. These `Executions` can be optionally grouped into `Experiments` to help the end user keep track and organize her work. `Experiments` themselves can be added to other `Experiments`. How these classes are related one to another, and the role of some other minor classes are shown in the UML class diagram of figure 6.5.

`forge` was such an unanticipated development that it grew out of multiple, short and unsupervised coding sessions. If using a networked relational database server was the fastest way to get such a tool up and running, the use of HTML for graphical user interface made it really reactive and flexible for change.

A first version of `forge` was implemented using Apache[7] and PHP[8]. It turned out to be a quick and dirty version, which is currently being reimplemented with Java technologies, using

[7]`http://www.apache.org/`
[8]`http://www.php.net/`

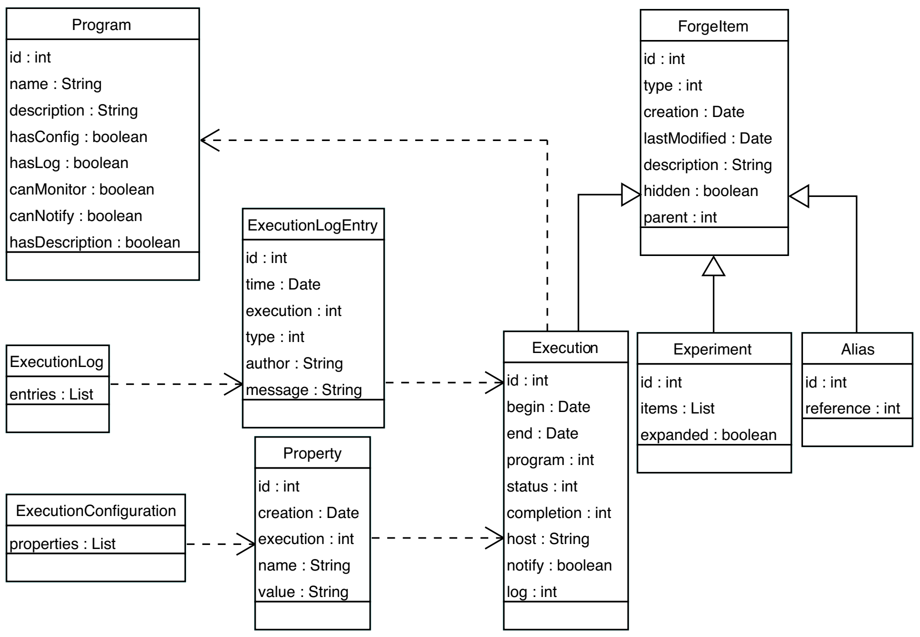


Figure 6.5: UML class diagram for the `forge` data model. The central class is `Execution`. It represents one run of a `Program`. It comes with a configuration (a list of `Property` grouped into an `ExecutionConfiguration` instance) and a log (a list of `ExecutionLogEntry` grouped into an `ExecutionLog` instance). `Experiments` are ordered lists of `Excecutions` and other `Experiments` grouped together according to some user rules. `Aliases` are place holders used for including an `Execution` at several positions in a list.

Tomcat[9] and the Spring Framework[10]. This reimplementation will help cleaning the code, give new opportunities for enhancement, and simplify the installation procedure of `forge`.

As well as `feign` can be used to run any kind of simulation that produces multivariate time series, `forge` can be used to setup, monitor and archive any kind of process. Nothing in the design of `forge` is intimately bound to `feign` or to spiking neural networks. Thus, `forge` is available from the web as a separate project (see Appendix A, p.115). Note that `forge` is not a distributed scheduling system. It cannot be used to start or stop executions through the network or interact in any way with remote processes, except with the database server.

6.2 Data manipulation

The output produced by the simulations has to be prepared before being analysed. The time series must be processed to extract the interesting parts from the complete recordings. The circuits that emerged from the initial random network as a result of the pruning must be transformed in a representation suitable for investigation. This is the role of the data manipulation programs presented in this section. These tools are also the entry point in the system for data that might have been collected using different techniques than the ones presented here (other simulators, *in vivo* recordings, ...)

From this point on, time series and networks will follow parallel ways in the information flow described in figure 6.1, Note that the "fnet" prefix is used for the programs – `fnetdb`, `fnetdig`, and `fnetview` – and file formats related to the networks produced by the simulation runs. It is the contraction of "`feign`-networks".

6.2.1 `fnetdb`: handling graphs through relational databases

Results presented in Part 1 show the emergence of subnetworks in larger networks associated with spike-timing-dependent synaptic plasticity (STDP). A flexible representation of the graphs was required to investigate the relations between spiking neuromimes interconnected in a network. We chose to represent these relations in a relational database. Beside considerably easing the development effort by relying on a fully debugged data structure, the possibility to search the

[9] `http://java.apache.org/tomcat/`
[10] `http://www.springframework.org/`

network using a relational language like SQL was an incredibly versatile solution.

`fnetdb` is the shell script developed to load and query `.fnet` network states (see section 6.5.1, p.98) into SQLite3[11] file databases. These databases are binary files containing the relational table structures and indexes that can be queried using the SQLite command line tool, or through the libsqlite programming interface. `fnetdb` is an interface on top of the SQLite command tool, that translates graph queries into SQL queries. The relational queries were complex enough not to be typed directly to `fnetdb`, but by using `fnetdig`, an helper script described in section 6.3.1, p.86.

6.2.2 `DataToolbox`: an interactive tool

Most of the programs and applications we have developed in the last few years are aimed at interactively manipulating point process time series, both simulated *in silico* and recorded *in vivo*. Working with simulated time series data looks like working with recorded time series data. It only requires far less effort to record thousands of simulated units for hours than properly separate and record tens of neurons out of implanted electrodes. The resulting simulated files are usually larger and contain more, different event types than the recorded ones, ensuing many information management issues. The latest recording developments using optical methods and micro-electrodes arrays suggest that the same problems are becoming visible to the experimentalists working on both behaving animals and brain preparations. `YaTiSeWoBe` (see section 6.4.4, p.96) is a contribution to the management of multivariate time series by a team of collaborating scientists over a computer network.

The simplest tool that emerged from this experience is the `DataToolbox`, a modular Java application aimed at manipulating time series data through filter tools that are dynamically discovered at application start-up. Sitting on top of the data manipulation libraries developed along `YaTiSeWoBe`, `DataToolbox` shows most of the advanced features present in `YaTiSeWoBe`, but at a smaller scale. Two aspects are shared by both applications: the time series data model and the filtering data manipulation model. The implementation of these two models is available from the web (nhrg-java-apps), as explained in Appendix A, p.115. We will now describe these two models.

[11] `http://www.sqlite.org/`

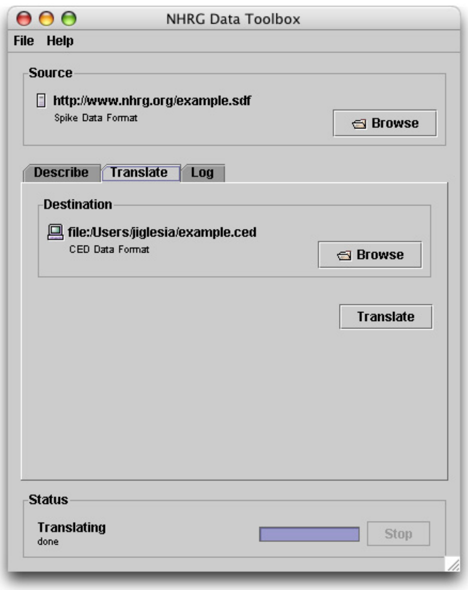


Figure 6.6: `DataToolbox` screenshot. The application is visually organized into 3 distinct parts. Top part defines the origin of the data to be manipulated (Source). Files and URLs can be dragged on the component. Alternatively, the "Browse" button can be hit to pop a file chooser up. Middle part is composed of tabs, one tab for each tool detected at start-up, plus one for the common "Log" tab. Each tool is responsible for laying down its front-end controls, help and output with unbound complexity. In the case of the "Translate" tool, it offers the possibility to choose a file (Destination) where data should be written to, along with the desired format, and a button to launch translation. The destination format is usually different from the source format. Bottom part shows a common "Status" bar providing information on what is going on in real time.

6.2.2.1 Time series data model

A clear model of the objects at hand is an important tool to have when manipulating data. It took us a few years working with time series to develop the data model described in the class diagram figure 6.7. This model was not the objective of our work, but a tool to describe how data is consistently understood and handled in the different programs.

The left part of the figure 6.7 models single events of the time series. This part is directly influenced by the Spike Data Format `sdf` (see section 6.5.4.2, p.101), with the use of a `RecordType` composed of a code and a qualifier. Codes and qualifiers are used to create a hierarchical classification of the event types. All event types of the same "kind" (e.g. recorded cell, stimulus onset, ...) have the same code and are distinguished by the qualifier (e.g. recorded cell #1, recorded cell #2, ...). Using the *bitwise and* operator, event types can be grouped into classes of events, like with `RecordTypeImpl`'s mask attribute. The stream filters described in the next section typically handle `Record` objects. The center part of the figure 6.7 models sequences of contiguous data like digitalized signals and point processes. `DigitalSignal`s focus on the intensity of a variable along time, and are typically a continuous sample of a physical variable. `PointProcesse`s focus on the timing of the data, and are a resource-friendly simplification of digital signals in the case of a two state variable ('0-1', 'yes-no', ...), with one of the states having a small probability of occurrence, like the spiking activity recorded during our simula-

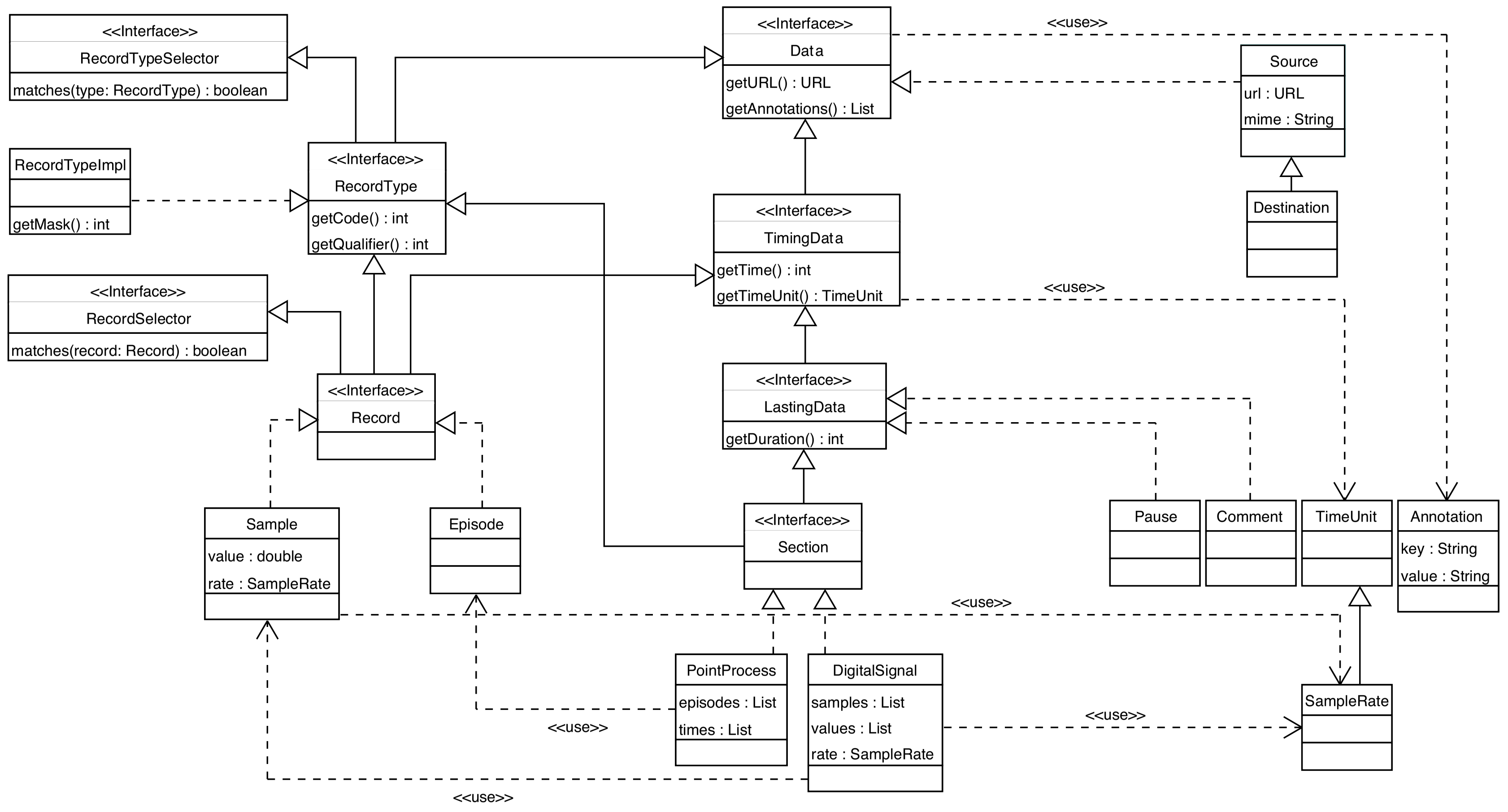


Figure 6.7: UML class diagram for the time series data model, both point processes and digitalized signals. The left part is aimed at modeling single records; the center part models continuous recordings; and the right part models meta-information that can be gathered around time series through the `Annotation`s at every level of the class hierarchy.

tions. Some data manipulation algorithms are more efficient on a `Section` representation than using separated `Records`. Finally, the right part of the figure models the different pieces of meta-information that can be attached to all the objects of the hierarchy, and specifically the `Annotations`, a dictionary-like list of key-value pairs.

One unusual aspect of the class diagram figure 6.7 requires some explanation. There are two circular dependencies running from the `Record` interface to the `Data` interface, one by extension of interface `RecordType` and the second by extension of interface `TimingData`. In a parallel manner, the `Section` interface extends the `RecordType` and the `LastingData` interfaces, that both extend `Data`. Those cycles result from the need for both a top `Data` interface and a mean to manipulate `Sections` and individual `Records` in similar ways. It is not straightforward until one starts programming with the data model, but the existence of those cycles in the object hierarchy greatly simplifies the data manipulation model discussed later.

Despite the historical link, this data model is not bound to the SDF data format. It has been successfully used to map data stored in different file formats like Holland Sleep Research's binary European Data Format (EDF and EDF+)[12], and Cambridge Electronic Design's CED[13].

6.2.2.2 Data manipulation model

The proposed data model separates the data representation, like data formats, from data manipulation. According to figure 6.8, `Sources` of data are entirely defined by the location where they are stored (expressed as a URL) and the data format (expressed as a MIME type). The MIME type provides enough information to determine which parser to use, provided as one of the implementations of the `FormatedDataReader` interface. The role of the `FormatedDataReader` is to load data into a `DataBridge` implementation, through such methods as `addComment()` and `addEpisode()`. Linking a simple string of characters (the MIME type) with a parser can be determined at runtime and allows multiple file formats support. At this point, the data is free from any formatting and enters the realm of the proposed data model described in figure 6.7. It is only linked back to its original source by its URL. Figure 6.8 shows the relations between the different objects associated with data manipulation.

The `DataBridge` object encapsulates many different processes. For now, two of them have

[12]`http://www.hsr.nl/edf/`
[13]`http://www.ced.co.uk/`

been described: a `DataFilter` subclass modifies and/or discards data somehow before passing it further to another `DataBridge` subclass; a `FormatedDataWriter` represents data back into a specific format, and stores it into a `Destination`, potentially in a different format than the original one.

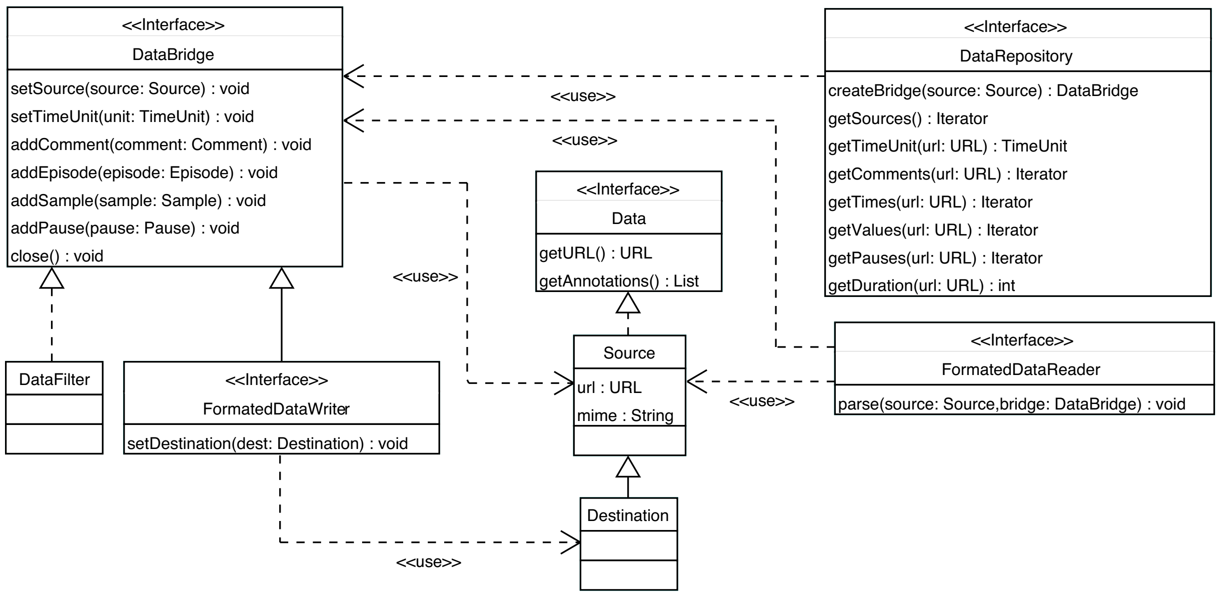
`DataFilter`s manipulate streams of data, as time series flow through them. Despite the thousands of ways to articulate chains of filters to produce the desired results, some manipulations, like merging two `Source`s into one `Destination` for example, are out of reach with this construction. For those cases, a `DataRepository` is a place that can optionally store one or more sources of data in a representation with a finer-grain interface suitable for more complex algorithms. When ready, the data can be dumped back into a `DataBridge` for further processing and/or formatting to a `Destination`.

With this simple construction, it is possible to read data from a URL, parse the format, manipulate information through successive independent filters and write it back in some different format. It is greatly influenced by the piping facilities available on Unix operating systems, where the output of one program can be chained into the input of another. The conceptual benefits are valid – independent programs communicating through a simple and well defined interface scale better with the complexity of the tasks – but time series semantics are enforced.

`DataToolbox` is a generic graphical front-end for `DataFilter`s. Each implementation detected while starting the application gets its own tab in the middle part of the window, as described in figure 6.6. `FormatedDataWriter` implementations are listed in the open (save) file dialog list of available formats.

6.2.3 `manip`: a versatile filter and editor

After acquiring time series, either by simulation or by *in vivo* recording, and before analysing them, there was a need for a tool to extract the interesting parts from the raw series. `manip` was developed with the objective of simplifying the cutting of the data along the time axis, selecting portions of the recording duration (e.g. around the stimulation times) as well as along the event axis, selecting the events we are interested in (e.g. getting rid of episodes recorded from cells #2 and #4.) This program was written in C with the help of several Open Source projects. It relies on the SQLite3 in-memory relational database library to store the configuration and



<<Interface>>
DataBridge
setSource(source: Source) : void
setTimeUnit(unit: TimeUnit) : void
addComment(comment: Comment) : void
addEpisode(episode: Episode) : void
addSample(sample: Sample) : void
addPause(pause: Pause) : void
close() : void
<<Interface>>
DataRepository
createBridge(source: Source) : DataBridge
getSources() : Iterator
getTimeUnit(url: URL) : TimeUnit
getComments(url: URL) : Iterator
getTimes(url: URL) : Iterator
getValues(url: URL) : Iterator
getPauses(url: URL) : Iterator
getDuration(url: URL) : int
<<use>>
<<use>>
<<Interface>>
Data
getURL() : URL
getAnnotations() : List
<<use>>
DataFilter
<<Interface>>
FormatedDataWriter
setDestination(dest: Destination) : void
Source
url : URL
mime : String
<<Interface>>
FormatedDataReader
parse(source: Source,bridge: DataBridge) : void
<<use>>
Destination
<<use>>

Figure 6.8: UML class diagram for the time series data manipulation model. Formated data is represented by `Source` and `Destination` classes. `DataFilter` subclasses can be used to manipulate time series data in any appropriate way before passing information to the next `DataBridge`, that can manipulate data further, format it back to some persistent location through a `FormatedDataWriter`, or place it into a `DataRepository`. Only part of the rich `DataRepository` interface has been represented for the sake of readability. This figure is related to figure 6.7 through `Data`, `Source`, and `Destination` classes.

the data at runtime. This unusual solution was preferred over others that would have required more coding, debugging, and maintenance. The processing required from `manip` is expressed as a set of SQL queries efficiently interpreted by the library with the help of the automatically maintained indexes. The program comes with its own imperative configuration language. The use of the GNU flex[14] fast lexical analyser generator and the GNU bison[15] general purpose parser generator eased the writing of the program, its maintenance, and its evolution.

`manip` is used like most Unix command line programs, with the possibility to introduce it in a pipe of programs reading the input from `stdin` and writing their output to `stdout`. Working this way allows maximum flexibility and performs particularly well in collaboration with many other existing programs. A special mode turns `manip` into a general-purpose pre-processing stage for other analyses. Another mode provides a detailed description of a file contents for human inspection. Using a third mode, the spiking activity of the cells can be efficiently searched for patterns of activity with the help of a template. `manip` can list all the occurrences of cell

[14]`http://www.gnu.org/software/flex/`
[15]`http://www.gnu.org/software/bison/`

#10 firing 243ms before cell #1, and 341ms before cell #5, for example, optionally allowing a jitter around each inter-spike interval. The search algorithm was efficiently implemented within minutes, thanks to the SQL relational query language and the indexes maintained by the database. Some functionalities of `manip` were general enough to be placed in the `libNHRG` library.

6.2.3.1 `libNHRG`: a data manipulation and analysis library

This C library was developed to encapsulate common tasks for programs and applications. Among them are APIs for time series data reading and formatting and `XY-Viewer` graphics production. The common time series data format used between all programs is the Spike Data Format (`sdf`), described in section 6.5.4.2, p.101. The library allows different data formats to be manipulated though, thanks to the time series data model presented in section 6.2.2.1, p.80.

`XY-Viewer` is a general-purpose 2D plot rendering application described in section 6.4.2, p.89. Formating data analysis results in a suitable form for that application, described in section 6.5.3, p.100, would be cumbersome without a proper programming interface like the one provided by `libNHRG`.

6.3 Data analysis

A large amount of data came out of every single simulation we performed. Part of the data is related to the circuits that emerged from the randomly connected network through the synaptic pruning. The rest is composed of the multivariate time series recorded from the activity of the neurons. The circuits were analysed with the tools provided by the graph theory (see Albert and Barabasi (2002) for a review). Some well known indices of the graphs were used to describe the emerged circuits, like the input/output degrees (k_{in} and k_{out}) which represent the number of incoming and outgoing projections of one neuron, respectively. The mathematical field of statistics provides the methods for the description and investigation of the time series, like first order statistics (mean and variance) and second order statistics (auto– and crosscorrelations). In addition to those established methods, we also used the search for spatiotemporal patterns of activity as a mean to detect the presence of recurrent "functional" paths in the emerged circuits.

No time was spent describing new methods or implementing existing ones. We did spend some time and energy trying to put together a comprehensible and distributed environment to simplify the usage and dissemination of established and novel methods in the field of neuroscience. The current state of the project is described in section 6.3.2, p.87.

6.3.1 `fnetdig`: searching graphs

The status of the networks at specific times during the simulations were dumped as graphs. In order to investigate them, they were loaded into relational databases with the help of `fnetdb` (see section 6.2.1, p.78). `fnetdig` is a shell script around a GNU *make* script that uses `fnetdb` to query the database. It constructs textual lists of units or projections that show a particular aspect, e.g. units that have more than 7 strong inputs, 10 strong outputs among which one goes to unit #132. These textual lists are then further merged, compared for union or exclusion with the GNU coreutils[16] (sed, grep, cut, sort, uniq, tr, . . .)

GNU *make*[17] is a tool which controls the generation of some files from other files. *Make* figures out automatically the proper order to produce files, in case one file depends on another. Consider the following scenario: a user wants to compute the list D of all units with $k_{in} > 7$, $k_{out} > 10$ and a connection to unit #132. *Make* has been instructed that in order to compute D, it requires list A of units with $k_{in} > 7$, list B of units with $k_{out} > 10$, and list C of units projecting to unit #132. It then computes the intersection of these 3 lists. If any of A, B or C is missing, it will first be constructed before performing the intersection. Recursively, if any of A, B or C relies on the existence of other lists, those will be computed first.

The way `fnetdig` uses the *make* tool is rather unusual, but it helped saving a lot of time. First by simplifying the development. Second because some of the lists are quite expensive to obtain, computationally speaking. *Make* saved a considerable amount of time by not recomputing lists that were already available, and by computing only the strict amount of information required to get the job done.

The use of SQLite3 as a data structure, *make* for dependency solving, and coreutils for list processing proved to be solid and versatile. Looking for new aspects of the graphs is just a matter of defining how pieces of existing information can be merged together to extract the new

[16]`http://www.gnu.org/software/coreutils/`
[17]`http://www.gnu.org/software/make/`

aspect, or writing the SQL query to get the information out of the database. The amount of work required to get new functionalities out of the program is small. The next best thing would be to develop a complete programming language to manipulate the graphs, although requiring a larger investment. One important drawback is that it is left to be checked if the scripts are fully portable.

6.3.2 OAN: a distributed analysis framework

A critical feature of brain theories is whether neurons convey a noisy rate code or a precise temporal code. One of the most valuable ways to test these theories consists of collecting the electrophysiological activity of cell assemblies under several experimental conditions. The sequences of cell discharges, – the spike trains – form a time series whose dynamics are strongly related to the information processing carried out in the brain areas under study.

The targeted users of OAN are graduates and PhD students as well as senior scientists working in electrophysiology. We expect these people to work in a University, a High School or other public or private educational departments. The number of users cannot be anticipated because OAN is only a partial set of our Virtual laboratory that expanded with several modules since the beginning of the project. The original programs designed for electrophysiological analyses have been generalized in order to accept data from several types of discrete time series. A current field of interest is represented by the simulation of neural networks.

The overall objective of this project is to create an integrated multi-platform software allowing the neuroscience research scientist, starting at the level of graduate students, to perform a comprehensive series of electrophysiological data analyses without the need of a long training with a specific software. This goal can be achieved by using the Internet browser as a multipurpose graphic terminal. The possibility to use remote computers for data analysis represents an important feature of the framework OAN and is based on the powerful network features provided by the Java language. Our purpose is to provide a user-friendly computational framework that is compatible with a more general concept of 'Virtual laboratory', i.e. a laboratory where data collection, computational power for analysis and display of results can be distributed over a computer network, like Internet.

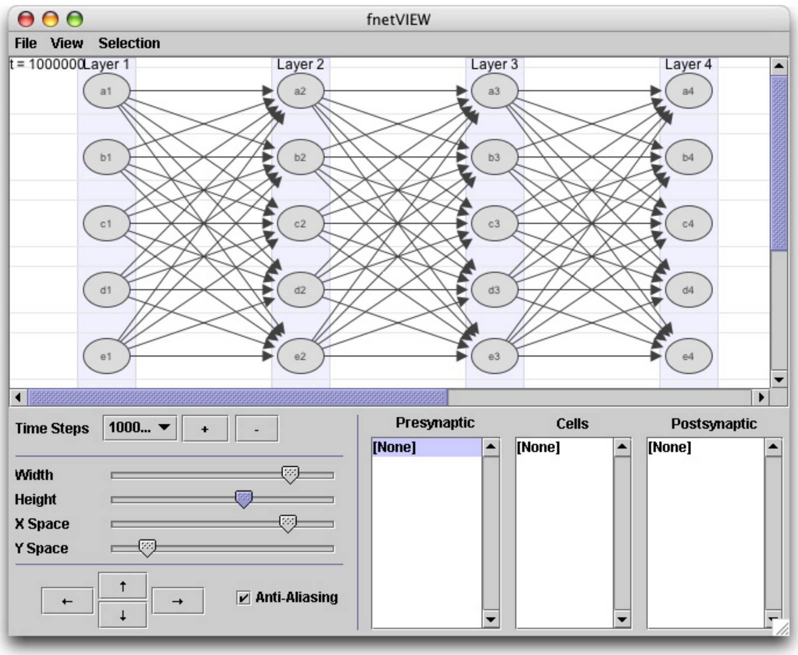


Figure 6.9: `fnetview` screenshot. The top part of the window shows an interactive representation of a 4 layer synfire-chain. *Ovals* represent neurons, *arrows* the directed connections, and their thickness the strength of the projection. Neurons can be reordered by clicking and dragging them around. The layers are "sticky" and neurons are always kept in line. Neurons and projections are dynamically color coded to help users finding the best position of a neuron in a feed-forward structure.

6.4 Visualization

The applications discussed in this section help users to visualize, interpret, and save for later reference the large amounts of information implied by large scale simulations. Without them, the task of plotting and screening the activity of tens of recorded units activity, and the subsequent correlations would have been out of reach. The outputs of these applications are vector graphics files in PDF, EPS, or SVG format. General-purpose vector graphics editors can be used to turn these high quality outputs into scientific journal figures. `XY-Viewer` is of special interest in this area, as it can be reused immediately outside the work presented here, and even outside the field of computational neuroscience. Most figures and plots of this document were generated using `XY-Viewer`.

6.4.1 `fnetview`: interactive reconstruction of feed-forward networks

Graph visualization[18] is a research field of its own. There are complex issues associated with rendering large graphs with lots of connections. `fnetview` is a first attempt to represent and edit layered feed-forward networks in a user-friendly and interactive way.

[18]`http://www.graphdrawing.org/`

fnetview relies on the drawing capabilities offered by the Java Universal Network/Graph Framework JUNG[19] Open Source project. It reads graphml files and displays the contents on a sticky grid. The nodes can be moved around the grid in an attempt to reorganize the network. The resulting network can be exported to SVG vector images for further processing with a vector graphics editor. The same network can be loaded at several time steps of the simulation. The rendering can be switched from one time step to the next in order to display the evolution of the connection weights through time. Many aspects of this application could be developed further. In order to represent synfire like structures, the 2D coordinate system should be replaced by some kind of cylindrical coordinates.

6.4.2 XY-Viewer: a generic plot viewer

XY-Viewer is a Java application that displays x-y plots (see figure 6.10) described in a dedicated XML based format discussed in section 6.5.3, p.100. This application was originally developed during summer 2002 by Olexiy Gospodarchuk at the PNN software company under my supervision as a side product of the OAN project (section 6.3.2, p.87). The original implementation had several flaws that led to its complete rewriting at the end of year 2004.

Within the OAN project, the need appeared to have a portable and user-friendly tool that could provide a fast glimpse on the output of different unrelated scientific analyses. We needed a generic front-end for scientific plotting. None of the tools available at that time had the possibility to display a wide range of graphics obtained through the web, run on multiple platforms without needing complicated installation procedures, and produce scientific journal-quality graphics output. For free.

The work on XY-Viewer was largely influenced by the web technologies like LaTeX and CSS, and the daily use of Gnuplot[20], a portable command-line driven interactive data and function plotting utility available for many platforms. The desire for clear separation of the data, the format in which the data is stored, the details of the graphics rendering, and finally the rendering driver (on screen or virtual paper) drove the development of this application by mimetism. Gnuplot is designed as a command-line tool and a rendering engine. The handling of a large

[19]http://jung.sourceforge.net/
[20]http://www.gnuplot.info/

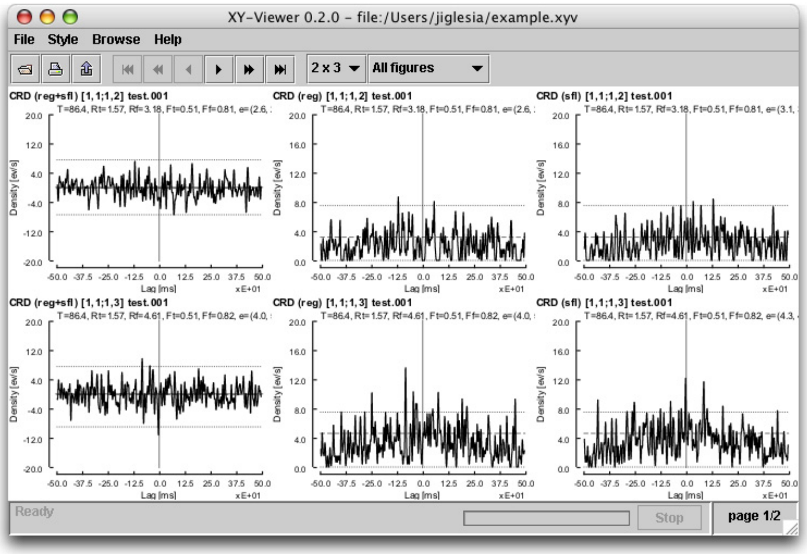

XY-Viewer 0.2.0 - file:/Users/jiglesia/example.xyv
File
Style
Browse
Help
2 x 3
All figures
CRD (reg+sfl) [1,1;1,2] test.001
CRD (reg) [1,1;1,2] test.001
CRD (sfl) [1,1;1,2] test.001
CRD (reg+sfl) [1,1;1,3] test.001
CRD (reg) [1,1;1,3] test.001
CRD (sfl) [1,1;1,3] test.001
Density [ev/s]
Lag [ms]
xE+01
Ready
Stop
page 1/2

Figure 6.10: `XY-Viewer` screenshot. The application is designed to render on screen and in print (PDF) a wide variety of scientific data described as sets of X-Y values. Many convenient shortcuts help users to browse large numbers of plots in an efficient way. It contains a high quality, and style-based rendering library (see section 6.4.2.1, p.91), driven by style sheets that the user "teaches" to the application whenever it encounters a new class of graphics. User can dynamically interact with plotted data. The modular nature of the application leaves multiple hooks for future enhancements and customizations, from data file format (see section 6.5.3, p.100 for default format), to rendering details, through statistical analysis.

number of graphics, multiple graphics per page, and point-and-click interaction with the graphics are available features, but not really user-friendly.

Sun Microsystem's Java language has many features that suit perfectly with the requirements of the application. Beside the availability of the Java virtual machine for most common operating systems[21], full-featured applications can be deployed over the network with a single click, using Java Webstart[22] technology. It even ensures that the latest version is always run, downloading updated components before launching the application. Since the appearance of Java 2, the graphics environment allows one to draw at the best quality of the different drivers, on screen and on print. Multiple projects have exploited this feature. We used the iText[23] library to generate PDF files on the fly. The heart of the system, the rendering library, was developed from scratch. The original ideas date back to 1999 and were first sketched out for `RasterViewer` (see section 6.4.3, p.95), at a time when the quality of Java graphics implementation was less obvious, like the lack of sub-pixel precision.

6.4.2.1 Rendering library

`XY-Viewer` and `YaTiSeWoBe` share the same underlying graphical rendering engine that stresses the separation of the content from the representation, and the model from the view. That library evolved on top of the Java 2 standard graphics capabilities to circumvent some of its limitations. The principal concepts of the library are: the *spot*, encapsulating the details of how a mathematical point should be displayed; the *path*, encapsulating the details of how the connection between points should be displayed; the *label*, encapsulating the details of how a piece of text should be displayed; and finally the *stylesheet*, a dictionary associating style information to keys. The hierarchy of classes in the library is reproduced in figure 6.11.

Each class is usually extended with multiple implementations. Some of the `SpotDrawingStyle` implementations are: the cross ($\times$), the dot ($\cdot$), the oval ($\circ$), and the empty spot ( ). Instances can be resized, both foreground and background colors can be assigned, leading to a finite but large number of combinations. Current `StrokeStyle` implementations are: dotted ($\cdot\cdot\cdot$), interrupted (- - -), plain (—), and empty stroke (   ). Two implementations of

[21]`http://www.java.com/`
[22]`http://java.sun.com/products/javawebstart/`
[23]`http://www.lowagie.com/iText/`

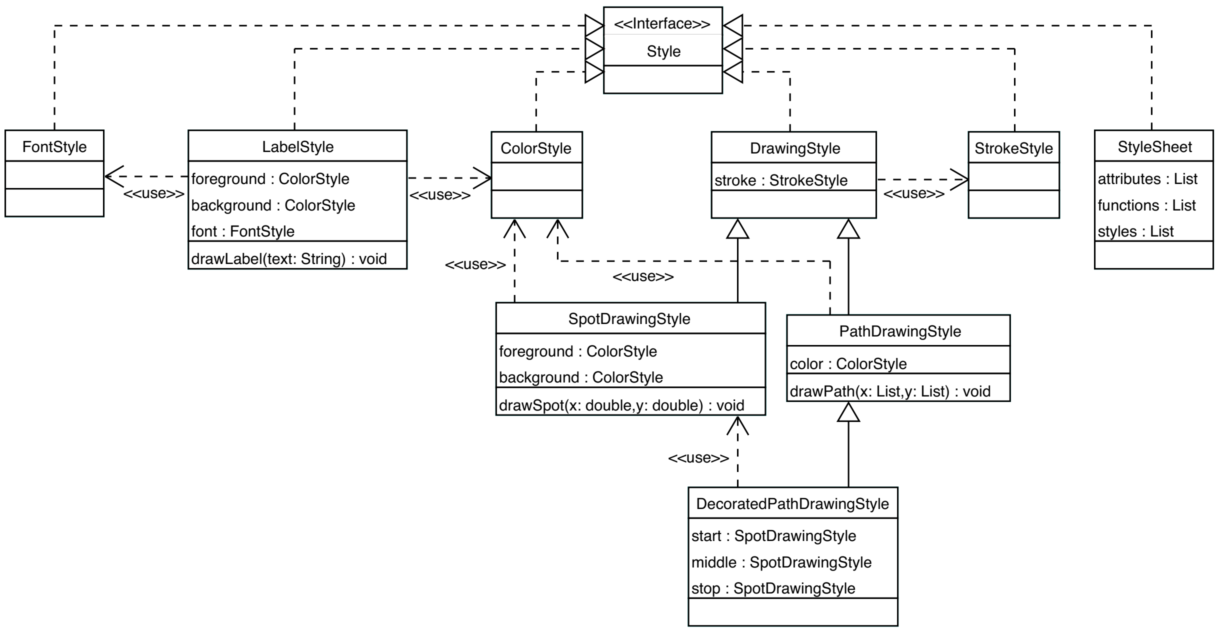


Figure 6.11: UML class diagram for the rendering library. At the top of the hierarchy is the `Style` interface. The interface is empty and is used as tag marking all classes in the library. `StyleColor`, `StyleFont`, and `StrokeStyle` are wrapper classes tagging as `Style`s the Java standard classes `java.awt.Color`, `java.awt.Font`, and `java.awt.BasicStroke` respectively. `SpotDrawingStyle`s, `PathDrawingStyle`s, and `LabelStyle`s encapsulate the details of rendering mathematical points, and connections between points and texts respectively. `StyleSheet`s are further detailed in figure 6.12.

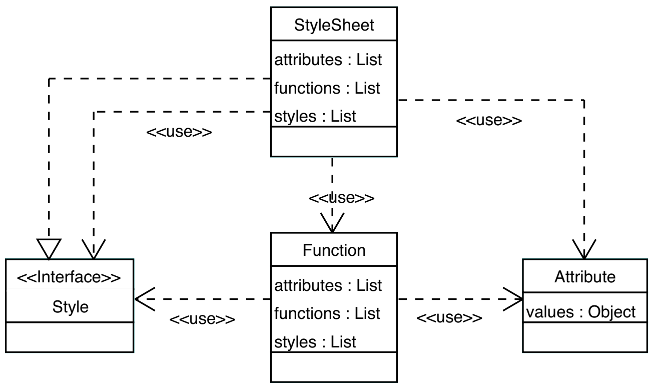


Figure 6.12: UML class diagram for the style sheet collaborations. `StyleSheet`s are the central objects in the rendering library hierarchy, containing `Style`s, `Function`s and `Attribute`s. They offer two recursion possibilities, first by being able to contain other `StyleSheet`s, and second by containing `Function`s, that can themselves contain other `StyleSheet`s.

`PathDrawingStyle` are missing from figure 6.11. One connects each point in a path with a line; the other is usually used for histograms, connecting successive points with a step. Using the `DecoratedPathDrawingStyle` implementation of `PathDrawingStyle`, one can combine spot styles with stroke styles to produce a large number of different path styles, like a line starting with an oval spot, ending with a rectangle spot, and connected by an interrupted stroke (○- - -□).

The cornerstone of the library is the `StyleSheet` object, that helps organizing `Style`s, along with two other types of Objects: `Attribute`s and `Function`s. `Attribute`s are Integer, String, Boolean or Double values associated with keys. `Attribute`s offer a solution to get some simple object values that are not `Style`-tagged into the rendering engine configuration. `Function`s are pieces of interchangeable implementations for a specific action. There are two extensions of `Function` defined in the library: `IntervalMappingFunction`s, used to map some piece of a 2D space on screen; and `DataSmoothingFunction`s, used to smooth data before rendering it. Current implementations for `IntervalMappingFunction` are `LinearIntervalMapping` and `LogarithmicIntervalMapping`, but polar or hyperbolic coordinate systems could be implemented. Current implementations for `DataSmoothingFunction` are `NoSmoothing`, that does not perform smoothing on data at all, `GaussianSmoothing` that performs a Gaussian bell convolution on the data, and `CERNSmoothing` that implements an unusual smoothing function based on running medians (Friedman, 1974). Finally, two more classes, `StyleSheetEncoder` and `StyleSheetDecoder`, can be used to recursively dump to and load from XML files the complete style sheet tree. The above lists of implementations are not constrained. When the library is loaded during application start-up, the complete list of available implementations is built according to what is found on the system at that time, through the Java Service Provider Interface (SPI). To hide the details of this runtime catalog construction from the library users, a factory class `GraphFactory` is provided that will return all the available implementations for each `Style` extensions.

6.4.2.2 Configuring styles

Providing the end-user with the graphical user interface widgets she expects from modern applications to configure all the possible combinations of spots, path and functions is a daunting task. First, writing the end-user graphical interface (buttons, menus, point-and-click handling, ...) is one of the most time consuming tasks for a programmer. Second, style sheets provide a recursive and fine-grain description of the rendering details. Third, the exact number of available implementations of each style, and its nature is only known at the time the library is loaded to memory. This is precisely the reason why the rendering library comes with a sister library aimed at generating the graphical user interfaces automatically, based on the contents of the style sheets.

Consider the following example of the style edition complexity: as of version 0.2.0 of `XY-Viewer`, each plot axis has three `LabelStyle`s (ticks, exponent and label), and one `PathDrawingStyle` (axis line and ticks rendering). Each of them has multiple styles and attributes, plus two `DoubleAttribute`s (range minimum and maximum), and one `IntegerAttribute` (number of ticks). Each of these objects requires a different dialog window for edition, some of them with the ability to edit sub-styles further, like `PathDrawingStyle`s that are optionally associated to `SpotDrawingStyle`s. Figure 6.13 is a screenshot of the dialog window automatically generated by the library for the edition of a `PathDrawingStyle`.

The technique used to generate the graphical user interface from the contents of the style sheet is loosely related to the Mozilla's framework cross-platform toolkit[24] and the XUL movement[25]. These efforts aim at describing a graphical user interface through the use of portable XML files instead of using one specific toolkit. At runtime, an engine is responsible for translating

[24] `http://www.mozilla.org/xpfe/`
[25] `http://xul.sourceforge.net/`

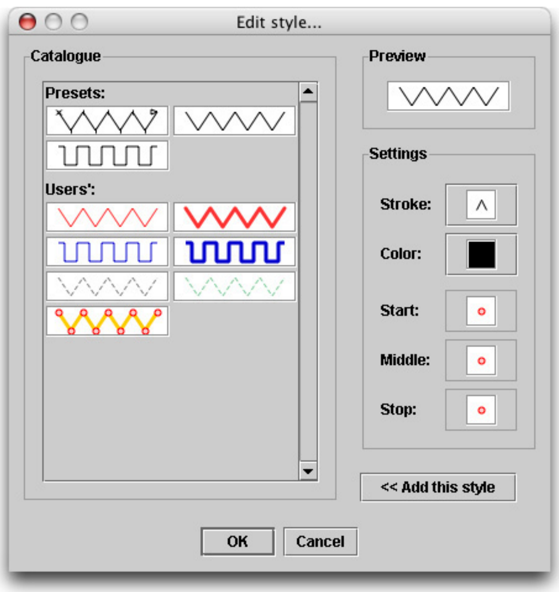


Figure 6.13: Screenshot of a dialog window automatically generated by the rendering engine configuration library for the edition of the details of a `PathDrawingStyle`. The left part of the dialog displays a catalogue of the available styles: "Presets" lists the available implementations with their default settings; "Users'" lists the styles defined by the current user and stored for later use. On the right part, below the "Preview" of the currently selected style, are the editors of the 5 sub-styles for this `PathDrawingStyle`. Hiting the different buttons opens a different dialog window, tailored for the edition of the details of the specific sub-style.

the description file into calls to the toolkit available on the system.

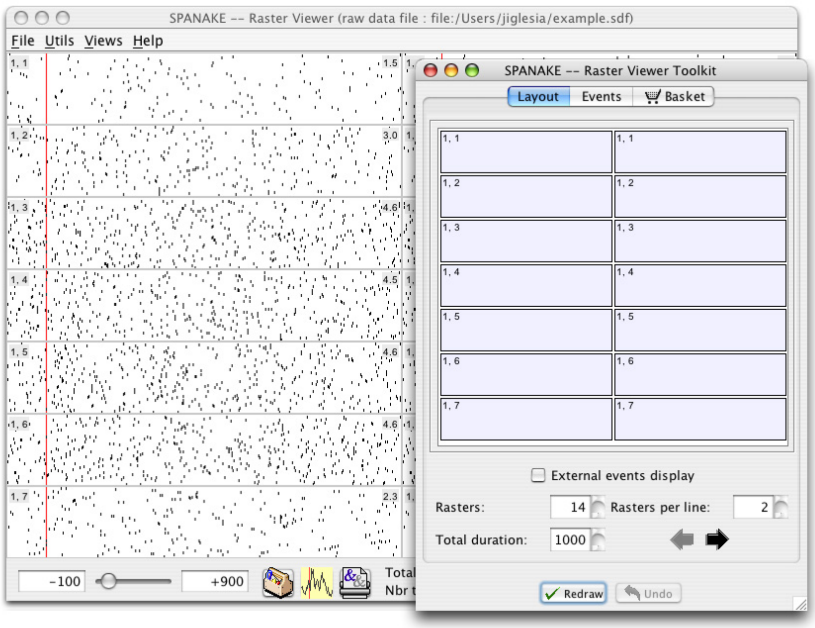
The current implementation is transitory. XForms[26] appears to be a cleaner solution to the problem, as one describes what one wants (e.g. "an integer value between 5 and 12"), rather than *how* one asks for it ("a text box at 12 pixels horizontally, 24 vertically"), like with XUL solutions. XForms is a rich and complex recommendation of the World Wide Web Consortium (W3C) that has not yet reached a sufficient momentum. The absence of a small and compliant library for XForms rendering with Java portable toolkit, along with the missing time for this secondary task, has momentarily stopped development in that direction. Another interesting aspect of XForms is its potential to solve the problems of automatic form generation for OAN analyses.

6.4.3 `RasterViewer`: a raster-plot viewer

`RasterViewer` is the oldest of all the tools presented here. It was written in 1999 with the objective to display the spiking activity of cells recorded *in vivo*. It reads time series in the `.sdf` format and displays them as raster plots.

Discrete time series of point processes (e.g. sequences of neuronal discharges – the spike trains) can be displayed as dot rasters. Each display line corresponds to a time interval (e.g. a single trial or a stimulus repetition) that is repeated one line on top of the other. Each dot

[26] `http://www.w3.org/MarkUp/Forms/`


SPANAKE -- Raster Viewer (raw data file : file:/Users/jiglesia/example.sdf)
File Utils Views Help
SPANAKE -- Raster Viewer Toolkit
Layout
Events
Basket
External events display
Redraw
Undo

Rasters: 14 Rasters per line: 2
Total duration: 1000
-100 +900

Figure 6.14: `RasterViewer` screenshot. The purpose of the application is to help the user to screen spiking activity. It is designed with a semi-direct graphical user interface. A remote controller (front window) is used to determine the details of what should be displayed on the main application window (in the back) after pressing the "Redraw" button.

represents the occurrence of an event (e.g. a cell discharge). The rasters are aligned by a trigger event (e.g. the beginning of the trial or the start of the subject's response).

This software allows one to analyse rasters of points corresponding to time series of recorded point processes. The user can customize the layout for the display of multiple rasters (one raster is referred to one type of event) by selecting the number of rasters on screen, as well as the number of rasters per line of display. The duration of the time interval can be selected and the rasters are dynamically controlled with a slider. The user has the possibility to select the event that will be the trigger of the raster display.

Sooner or later, `RasterViewer` is expected to be replaced by `YaTiSeWoBe`.

6.4.4 `YaTiSeWoBe`: an interactive workbench

`YaTiSeWoBe` stands for *Yet Another Time Series Workbench*. It is a pluggable framework aimed at providing groups of collaborating researchers a mean to exchange time series data, tools to visualize them, and results over a computer network. `YaTiSeWoBe` is designed to gather a maximum amount of information and knowhow for team members to access and manipulate them. This application represents the outcome of an ambitious project to merge some of the aspects of OAN and `XY-Viewer` into a user-friendly desktop application. So far, the work has been limited to the modeling and implementation of the underlying libraries for time series data

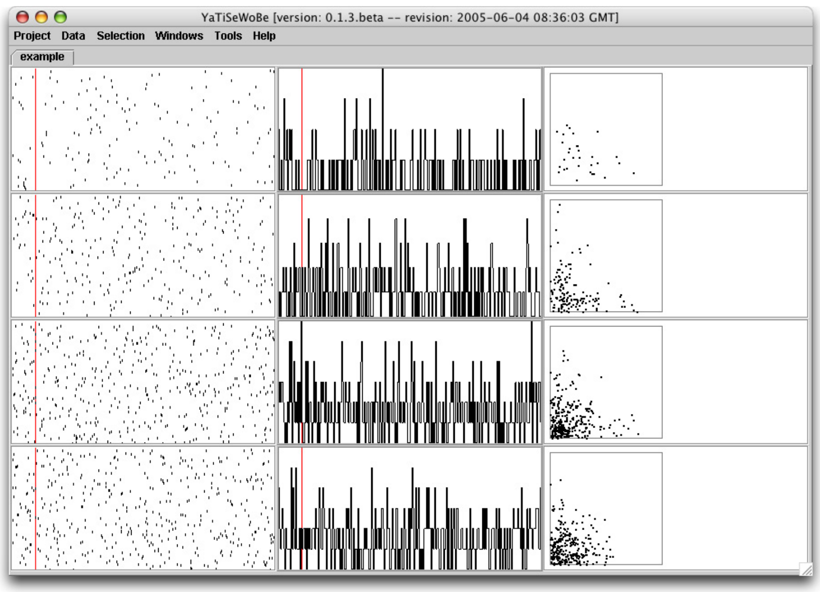

YaTiSeWoBe [version: 0.1.3.beta -- revision: 2005-06-04 08:36:03 GMT]
Project Data Selection Windows Tools Help
example

Figure 6.15: `YaTiSeWoBe` screenshot. The activity of 4 cells, one per line, is plotted around the stimulus onset (red bar). First column: raster plots akin to the first four lines of figure 6.14; Second column: raw (un-convoluted) peri-event time histograms; Third column: inter-spike interval return map.

processing and graphics rendering on which `DataToolbox` and `XY-Viewer` are also based. A powerful file format has also been designed and implemented to support the rich features of the application. A lot of work remains to be done on the graphical user interface and the scientific quality of the analysis.

Figure 6.15 shows the current status of the workbench. As an evolution of `RasterViewer`, `YaTiSeWoBe` can plot rasters of activity. Other types of representations can be plotted simultaneously, with all the power of the rendering library discussed previously. Unlike `XY-Viewer`, `YaTiSeWoBe` produces the graphics internally and does not read them from an external source. Users can interactively modify their contents along with their rendering details.

6.5 File formats

Following our software philosophy, text based formats were preferred to any binary file format for portability and interoperability reasons. Existing file formats were used when appropriate (see section 6.5.4, p.100) while *ad hoc* formats are XML grammars. Extensible Markup Language (XML) is a World Wide Web consortium (W3C) recommendation. It is defined as a simple and very flexible text format derived from SGML (ISO-8879). It is out of the scope of this discussion to provide details on XML, as many resources and tutorials are available.

Using XML to exchange information between programs has many advantages: parsers, validators and formatters are generic. Many of them are available as Open Source projects. They have been written, optimized and debugged for all modern languages. XML tags tend to auto-document the format for humans that would open the files using a simple text editor of their choice. Many technologies around the XML data files simplify information manipulation, like XML Schemas, XPath and XSLT, to cite a few. Relying on existing generic information manipulation tools has been of great help in the course of the present work.

The main disadvantage of XML files is the usual verbosity of the format that, along with the use of text encoding, leads to larger disk usage than binary formats. By experience, the use of a generic compression algorithm such as GZip allows far smaller files that still maintain most of the advantages listed above.

Sample files for each of the following formats are available in Appendix B, p.117 and from Appendix A, p.115.

6.5.1 `.fnet`: `feign` networks

A dedicated file format was required at some point to store the complete status of the simulated network at one time step. This format contains information concerning the connections between units, as well as the model details for the units and the connections. This way, the status of the network could be saved and analysed, or reloaded into the simulator and further simulated. The `.fnet` data files are used for that purpose. The information structure is described in an XML grammar. Such files are produced by the simulator, `feign` (see section 6.1.1, p.71), that is also able to read them back to continue a simulation from a previously saved state. They are loaded into relational databases by `fnetdb` (see section 6.2.1, p.78) in order to be analysed by `fnetdig` (see section 6.3.1, p.86).

6.5.2 `.xpdl`: `feign` protocols

XPDL stands for eXtensible experimental Protocol Description Language. Each protocol is a finite state machine mapped on a DOM XML tree. The "states" are the leaves of the tree. They are not specified by the `xpdl` grammar. The non terminals define the possible transitions (as of version 1.1 of the format). A sample protocol can be found in Appendix A, p.115.

protocol: The root of the state machine from where it starts and where it ends. A protocol contains either one of sequence, loop, branch or terminal.

sequence: An ordered sequence of states or transitions that a state machine will go through. A sequence contains any list of sequences, loops, branches and terminals.

loop: A repetition of its content, either a transition or a terminal. The repetition count can be fixed *a priori* (e.g. 4 times), randomly bound (e.g. between 10 and 20 times), or unbound (until computer stops). A loop contains either one of sequence, loop, branch or terminal.

branch: A point in the protocol where only one of the optional paths will be randomly chosen, according to their declared probabilities. Another possibility is to declare the branch equiprobable: one of the options will be randomly selected, with equal chance. A branch contains at least two options in which cumulated probabilities should be equal to 1.

option: A path of the protocol that can be followed with some explicit probability. If part of an equiprobable branch, the probability is ignored. An option contains one of a sequence, loop, branch or terminal.

The flexibility of XML name spaces allows the addition of undetermined terminals to this set of transitions. The principle of using the `xpdl` protocols is that once the finite state machine is loaded from the protocol, the machine is programmatically put to motion by successive calls to function `protocol_getNextState()`. The returned state is a terminal that the program using the state machine is in charge of analysing with the help of function `protocol_getAttributeValue()` to determine the actions associated with that state of the machine. The following call to function `protocol_getNextState()` will run the finite state machine through the different transitions of the protocol: branches, loops, etc – until the next state is reached. That state is returned to the calling program by the function.

There are many advantages in using this solution. At the programmer level, it is possible to use a standard DOM XML parser to read and validate the grammar before starting the experiment; the DOM tree representation can be used to travel through the protocol, jumping from leaf to leaf using simple XPath queries. Generating `xpdl` files can be done by hand, or tools could be developed to interactively design protocols using the basic transitions. Because of missing time, and a lack of interest from our undergraduate students, no such tool has been written so far.

6.5.3 `.xyv`: `XY-Viewer` native data format

`XY-Viewer` (see section 6.4.2, p.89) is a generic plotting application aimed at displaying scientific data into journal quality figures. We needed a data format able to act as the glue between OAN analyses (see section 6.3.2, p.87) and the viewer, encapsulating multiple sets of points with an arbitrary number of dimensions, along with meta-information related to the origin of the data and the treatment performed on it. The format had to be simple enough to be generated by analyses potentially developed by unknown contributors. In the field of cross-platform and cross-language data representation, XML has become the typical answer, and `.xyv` data format is no exception.

The `.xyv` data format is an XML grammar providing a default implementation for `XY-Viewer`'s representation model for multi-dimensional plots proposed in figure 6.16. Both reader and writer feature an optional and transparent non-destructive compression, using the GZip algorithm that typically reduces the size of the original file by a factor of 6 to 10 times. The compressed variant of the format is typically labeled `.xyvz`. The distinction is suggested but not imposed. This implementation was selected as the native data format for the application, but is by no means the only possible or desirable implementation. At start-up time, `XY-Viewer` looks for all the implementations available on the system and proposes to the end-user to choose between them during the session.

6.5.4 Existing formats

The obligation to define and maintain dedicated file formats has a price that should be avoided as much as possible. We tried to use existing file formats, whenever such formats existed and were suitable to our needs. Only a few of them are mentioned here because they deserve special attention.

6.5.4.1 `.graphml`: Graph Markup Language

Many file formats are established in the field of graph theory. A few of them are XML grammars. XML is really adapted for this purpose because of its hierarchical structure. Among the different

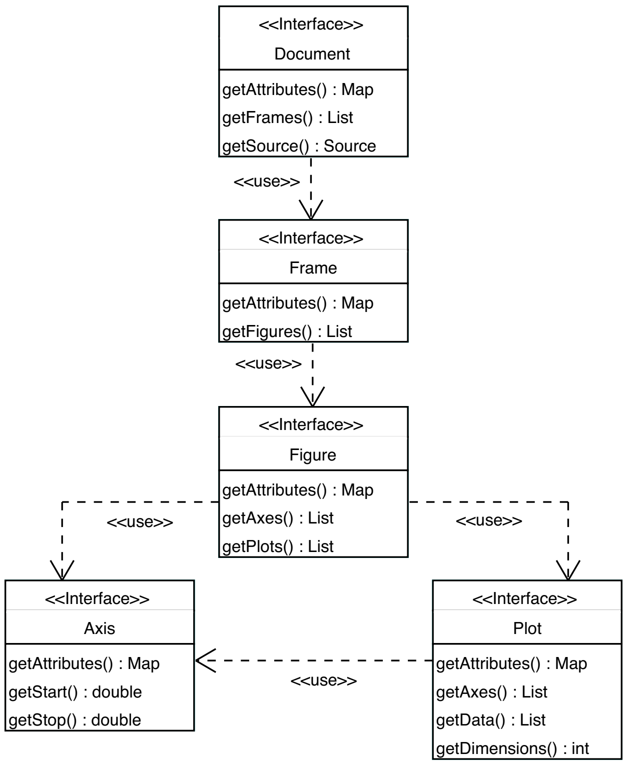


Figure 6.16: UML class diagram for `XY-Viewer` representation model for multi-dimensional plots. A `Document` is composed of one or more `Frames`, each of them describing an ordered set of logically related `Figures`. Each `Figure` is defined in $\mathbb{R}^n$ with one `Axis` instance representing each of the n dimensions. One `Figure` can contain multiple `Plots`, each of them encapsulating an unbound ordered set of points defined in $\mathbb{R}^m \subseteq \mathbb{R}^n$ and associated with the instances of `Axis` representing the m dimensions of the sub-space ($m \leq n$). Each object of the model can be documented with annotations, an unbound dictionary-like structure that associates one key with one value. The `.xyv` data format discussed in this document (see section 6.5.3, p.100) is one of the possible implementations of this model.

available formats, `graphml`[27] was preferred because of its simple and extensible nature, as well as the existence of many software pieces able to read and understand that format. Any time we needed general algorithms for graph manipulation, we used the Java Universal Network/Graph JUNG[28] framework.

6.5.4.2 `.sdf`: Spike Data Format

Persistent storage of multi-variate time series is achieved in the Spike Data Format SDF proposed by Moshe Abeles in 1991. `sdf` is a text-based format structured with a semi-formated header and a body containing the episode types and times, along with optional free-text comments. At the time this file format was developed, data used to be transfered on untrusty serial lines. The presence of checksums in the format body ensures that the information is not modified during file transfer.

The main drawback of this data format is the absence of structured meta-information like acquisition time, a feature present in other formats like EDF or EDF+[29]. The structure as well

[27]`http://graphml.graphdrawing.org/`
[28]`http://jung.sourceforge.net/`
[29]`http://www.hsr.nl/edf/`

as the format of this information are left to user interpretation.

The simplicity, efficiency and portability of this simple text-based format are un-matched for now. Nonetheless, an update of the meta-information capabilities of the format will certainly by required in the upcoming years in order to cope with the challenge of large-scale simulations and new electro-physiological recording techniques.

6.5.4.3 .sng: Scriptable Network Graphics

The Portable Network Graphics file format (PNG) is a Recommendation of the World Wide Web Consortium for lossless, portable, and well-compressed storage of raster images[30]. The Scriptable Network Graphics file format (SNG) is a minilanguage designed specifically to represent the contents of a PNG file in an editable form.

`sng` files are used in two places of `feign`: to define stimuli patterns to be applied on the network during simulation, and as a way to store spiking activity with an optional implementation of `feign`'s output module. Stimulation patterns are decomposed in one separated `.sng` file for each time steps. A tool could be written to interactively construct the stimuli, but a general-purpose text editor is good enough. Using an Open Source program[31], the ASCII representation can be converted into an animated raster image for visual inspection. Animated sequences for the two usages can be found in the Appendix A, p.115.

[30]`http://www.w3.org/Graphics/PNG/`
[31]`http://sng.sourceforge.net/`

Communications without intelligence is noise;
Intelligence without communications is irrelevant.

– Gen Alfred. M. Gray

Chapter 7

Hardware

Résumé Les ressources de calculs nécessaires pour simuler en grand nombre des réseaux de grandes tailles ont été puisées dans une grappe de 18 PCs constituée pour l'occasion. Les détails de la mise en place de ces machines multiprocesseurs dédiées au calcul scientifique sont donnés dans ce Chapitre. En outre, le modèle de réseaux neuronaux décrit précédemment a été développé pour une implémentation matérielle réalisée dans le cadre d'un projet annexe. Quelques informations pertinentes sur ce projet et sur l'implémentation matérielle sont fournies ici.

During the early phase of the research presented in this document, the question of the large computing power required to simulate large-scale neural networks for long durations was raised. The model we chose is suitable for an hardware implementation, but that hardware was not expected to be available for the next few years.

In 1994, a team of NASA researchers developed a technique to get supercomputer performances without the budget to buy one. They hooked up off-the-shelf personal computers with open source software (GNU/Linux) to create a system that could scale up to deliver supercomputer performances. A *Beowulf* system[1] is built out of truly commodity systems, unlike other clusters commercially available for years from large computer companies. Clusters of computers are typically used for High Availability (HA) for greater reliability, or High Performance Computing (HPC) to provide greater computational power than a single computer can. Most of the time, they use message passing to achieve parallel computations.

The idea of clusters of commodity PCs as an inexpensive alternative to expensive supercomputers turned somehow into an even larger scale idea. In 1999, another team of NASA

[1]`http://www.beowulf.org/`

researchers launched the SETI@home project[2] to search for extraterrestrial intelligence through signals collected by the Arecibo Radio Telescope with the help of thousands of volunteers that gave their free CPU cycles (e.g. during night) to the SETI project. Currently, about 40 gigabytes of data is pulled down daily by the telescope and sent to computers all over the world to be analysed. The results are sent back through the Internet, and the program then collects a new segment of radio signals for the PC to work on. The Grid Computing[3] was born. Our cluster was used like a computing farm, closer to the Grid computing philosophy than to the High Performance computing philosophy, with the advantage of having a complete control over the execution of the simulations. That control simplified enormously the development of the simulator as well as the management of the simulations.

We never intended to parallelize our simulator (see section 6.1.1, p.71) to run on multiple CPUs at the same time. The task is daunting, as there are many technical problems to solve. The NEST Initiative[4] is a promising project in this field. The developers already have several years of experience with large-scale neuronal network simulations on multi-processor supercomputers and Beowulf clusters.

In this Chapter, we will not discuss the theory of distributed computing, nor the details of the clusters of computers. We will extract from our experience a pragmatic approach to the usage of clusters of commodity computers for scientific research.

7.1 Beowulf-class cluster

7.1.1 Overview

We built an 18 + 2 Beowulf class cluster. There are 18 bi-processor nodes for calculation, 1 bi-processor computer to act as the head of the nodes (`master`), and 1 bi-processor computer with a safe storage facility to store the valuable results and access them trough the network (`store`). Figure 7.1 shows how these 20 computers were hooked together on a dedicated network switch, and how they are accessible from the rest of the University local area network.

The Institute for Computer Science of the Faculty of Sciences funded the cluster in 2001 for

[2] `http://setiathome.ssl.berkeley.edu/`
[3] `http://www.grid.org/`
[4] `http://www.nest-initiative.org/`

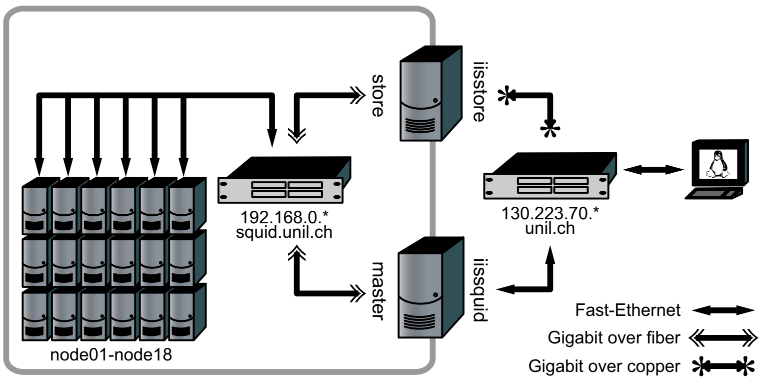


Figure 7.1: Overview of the Beowulf-class cluster we built for the purpose of the present research. Calculation nodes have a dedicated network used to communicate with the head of the cluster, the `master`. The `store` machine is used as a general-purpose network storage facility.

a total amount of about 50,000 €. The initial budget plan was to recycle the computers of the cluster into the students' computer room every 2-3 years. There, they could still be used for scientific research through a Grid computing software for another 2-3 years. With this plan, the hardware could have been optimally used for 5-6 years, with a new dedicated cluster hardware setup refreshed every 2-3 years. This original idea explains why some solutions were preferred despite their disadvantages, like the use of standard "tower" *vs.* "thin" rackable boxes that could not be used in a classroom.

7.1.2 Hardware configuration

In order to simplify the administration of the cluster, and as we had the opportunity to buy all the hardware at the same time, we chose one machine type and multiplied it 20 times. The two special computers (`master` and `store`) were extended from the basic setup with addition of some special hardware tuned for their specific tasks. To save time for the order and money on the bill, all machines were hand-crafted in-house from separated pieces bought at a computer pieces reseller at the end of the year 2001.

The computers were built around the first bi-processor motherboard available for AMD athlon CPUs. Bi-processors were chosen to save space and money, as only one power unit, one network interface, one floppy disk, and one case are necessary for two CPUs. The choice of conventional ATX cases instead of rackable ones was dictated by the price difference between them, the availability of a large room, and the original idea to recycle the cluster in the classroom.

The details of the 18 calculation nodes is provided here:

board	:	1× Dual AMD S462, 64bit PCI, ATX (Tyan Tiger MP S2460)
CPU	:	2× athlon 1600+ (AMD AX1600MT3C AGKGA0135SPFW)
RAM	:	4× 512MB 266MHz DDR ECC (Kingston ValueRAM KVR266X72RC25/512)
HDD	:	1× 20.5Gb, RPM 7200 (IBM DeskStar IC35L020AVER07-0)
network	:	1× Fast Ethernet (Netgear Bay Networks FA310TX Rev-D2)

This basic setup is aimed at providing computing power. The `master` is derived from this configuration by the addition of larger storage and broader networking capabilities. At that time (end of 2001), Gigabit Ethernet over copper had just hit the market, and no compatible hardware was available. We used the extension bays of the network switch (see below) to connect Gigabit Ethernet over fiber modules, despite the higher cost of that technology. The following hardware was added to the master:

HDD	:	3× 123.5Gb, RPM 7200 (IBM DeskStar IC35L120AVVA07-0)
RAID	:	1× RAID0 “striping” (Adaptec ATA RAID 2400A)
network	:	1× Gigabit Ethernet over fiber (Netgear Networks GA621)

The storage facility computer, `store`, was also derived from the basic configuration by addition of larger and safer storage as well as broader networking capabilities. The motherboard died after a few months, and had to be changed. By that time, the original motherboard was not available anymore and we had to change for another brand. About a year later, we could finally have access to Gigabit over copper links in our building, and an interface was installed on that machine.

board	:	1× Dual Socket A, AMD 760MPX (Asus A7M266-D/PA-UAY)
HDD	:	3× 123.5Gb, RPM 7200 (IBM DeskStar IC35L120AVVA07-0)
RAID	:	1× RAID5 “striping” and checksum (Adaptec ATA RAID 2400A)
network	:	1× Gigabit Ethernet over fiber (Netgear Networks GA621)
		1× Gigabit Ethernet over copper (3COM 3C996B-T)

Finally, three standard garage-like metal shelves were installed in a air-conditioned server room to accommodate the twenty 70 centimeters high “tower” boxes, the network switches, the two keyboard/video/mouse (KVM) switches and an uninterruptible power supply (UPS) for the `master` and the `store` computers. This miscellaneous hardware is listed here:

network switch	:	1× bandwidth 17.5 Gbps (Extreme Networks Summit 48)
KVM switch	:	2× 16 port PS/2, VGA (Aten MasterView Pro CS-1016)
UPS	:	1× 700VA, 450W, RS-232 (APC Smart-UPS 700)
metal shelves	:	3× H200 × W100 × D50 cm

7.1.3 Cluster management

Hooking together a bunch of computers into a Beowulf-class cluster is becoming easier every day. Dedicated GNU/Linux distributions like Scyld[5] can be used as an all-in-one solution for installation and management of large clusters of computers.

We tried this kind of solutions, but they did not fit well with our hardware and local setup. Bearing in mind that the cluster computers would someday be used as student workstations, we decided to go for a general-purpose GNU/Linux distribution, SuSE[6]. It had the drivers for all our hardware and proposed many software packages for High Performance Computing out-of-the-box.

The most time consuming task for a Beowulf-class cluster management is the installation process. Installing each computer by hand is impractical. Using SystemImager[7], the 18 nodes of our cluster can be reinstalled in a matter of 15 minutes, mainly devoted to booting the systems on floppy disks before the complete system image can be downloaded automatically from the `master` through the network.

Our cluster will soon be 4 years old. During this time, some nodes have been running for months, sometimes on a 100% load, without a single crash, besides the three local area power breakdowns of 2002, 2004 and 2005. The first few months were characterized by multiple, apparently uncorrelated problems, which resulted from the `master` fast-ethernet network interface not being fast enough to handle the connections from the 18 nodes. Since the introduction of the Gigabit over fiber interface, these erratic problems have disappeared. Two computers broke, apparently due to motherboard failures.

The most problematic piece of hardware appeared to be the air-conditioner. One was completely broken due to a mishandling of the outgoing water pipe. The current air-conditioner

[5]`http://www.scyld.com/`
[6]`http://www.suse.com/`
[7]`http://www.systemimager.org/`

was undersized by the technical people, forcing us to shut down some of the computers to avoid overheating... It also had water leakages due to problems of sealing with the water tank. A roof had to be built on top of the computers.

7.2 POEtic tissue

The neuromimetic model described in section 3.2, p.18 was originally designed (Eriksson et al., 2003) for an hardware implementation using a novel electronic device (Tyrrell et al., 2003) that includes features derived from some living beings' properties. The POEtic tissue is a matrix of reconfigurable POEtic chips, physically interconnected by a bidirectional bus, featuring self-repairing mechanisms, automatic routing features, and rich input/output connectivity to external devices. At the logical level, the tissue was designed to provide a flexible substrate for three organizing principles – the POE axes (Mange and Tomassini, 1998) – driving living beings as we know them on Earth:

Phylogeny: also called *evolution*. It includes all the mechanisms that allow: to encode a possible solution to a problem (*organism*) into a mutable description (*genome*); to measure the performance of the organism (*fitness*); and to manipulate the genome (*mutation operators*) to give rise to a new organism derived from one or more well-performing ancestors, mimicking the natural selection by means of environmental pressure. With the POEtic tissue, a user can encode the configuration details in a genome. There are no hard wired mechanisms for fitness evaluation and genome mutation, though. The evolution of a population can be performed off-line, and organisms can be uploaded on the tissue to be simulated sequentially.

Ontogeny: also called *development*. It describes the unfolding of events involved in the development of an individual organism changing gradually from a simple to a more complex level. The POEtic chips feature the possibility to reprogram themselves partially or completely at runtime. They provide the mean to colonize the available electronic substrate, starting from a small set of totipotent cells that multiply and differentiate into the components of a complex organism. This process is driven by the information contained in the genome.

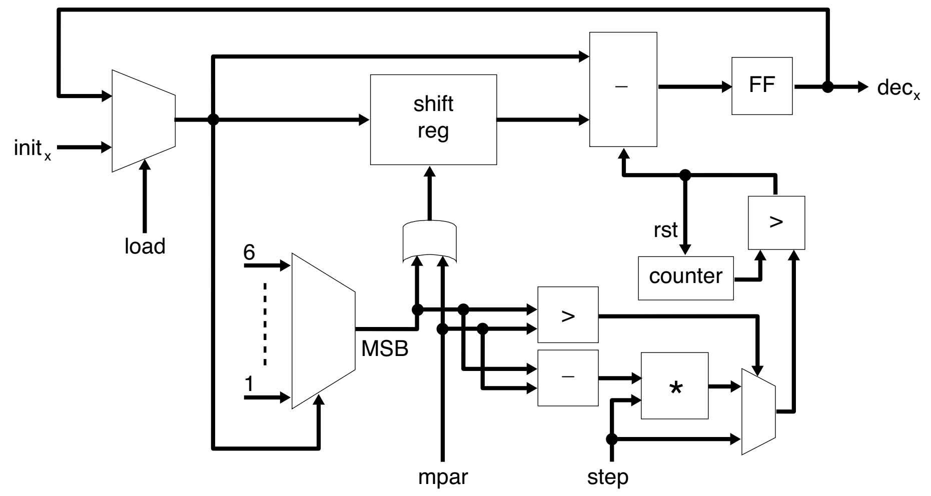


Figure 7.2: Block diagram for the decay block. The value x is obtained and input into the shift register, which is controlled by x most significant bit and the external parameter $mpar$. The output is subtracted from x. This operation is performed when the time control indicates it. The time control is achieved by the value of a counter that is compared to the result of choosing between the external value *step* and the $(step \cdot (MSB - mpar))$. The decay time constant τ depends on the input parameters $mpar$ and *step*.

Epigeny: also called *adaptation* or *plasticity*. It includes the processes that allow an organism to modify its internal structure or behaviour to adapt to its environment. The POEtic chips can be programmed to perform neuromimetic simulations showing such properties. A serial implementation of our spiking neuron model (Torres et al., 2004) has been proposed and implemented to overcome the limited quantity of available chips.

7.2.1 Neuron implementation

A neuron model, very close to ours (see section 3.2, p.18), has been implemented (Eriksson et al., 2003) and optimized (Torres et al., 2004) for the POEtic platform. We will stress the relation between our model and the hardware implementation, without entering in the details of the novel elementary programmable elements underpinning it. According to Moreno (personal communication), the membrane potential V has a resolution of 12 bits, with a range $[-2048, 2047]$. The threshold is kept fixed at $+640$. For efficiency reasons, the refractory period is set to $t_{refract} = 1$ for both excitatory and inhibitory neurons. The membrane time constant was set to $\tau_{mem} = 20$.

A decay block (see figure 7.2) has been defined to approximate the logarithmic decay of the input (x) in an efficient way (Eriksson et al., 2003). This decay block is used for the membrane potential computation, as well as for the learning and the synapse blocks.

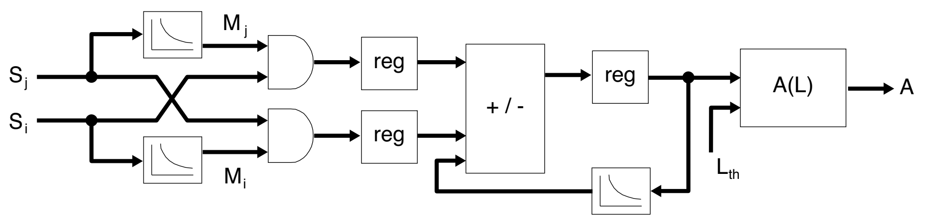


Figure 7.3: Block diagram for the STDP learning block. When a spike is produced $S_n = 1$, the corresponding memory variable M_n is reset to its maximal value and starts decaying. The ordering of pre– S_j and postsynaptic S_i spikes determines if the decaying learning variable L_{ji} is incremented or decremented. The activation level A_{ji} is updated whenever L_{ji} overcomes the threshold value L_{th}, unless it has already reached its minimal or maximal value.

The learning block implements the spike-timing-dependent synaptic plasticity (STDP) described in section 3.3.1, p.20. Depending on the timing of the presynaptic neuron $S_j(t)$ and the postsynaptic neuron $S_i(t)$, as well as on the type of neurons (excitatory and/or inhibitory), the activation level of the synapse will be updated as depicted in figure 7.3. The memory variable M, the learning variable L and the learning variable threshold L_{th} have resolutions of 6, 8 and 8 bits respectively. Four activation levels are defined for A. The synaptic plasticity time constant is set to $\tau_{syn} = 20$, whereas the slow synaptic pruning time constant is set to $\tau_{act} = 4,000$.

Note that the L_{ji} / A_{ji} relation is different from the one presented in section 3.3.1, p.20. In our model, A_{ji} state is switched when L_{ji} leaves the range $]0.0001, L_{th}[$ with a decaying function driving L_{ji} to 0. The connections tend to switch A_{ji} states down, leading to the synaptic pruning. In the present variant, A_{ji} state is switched when L_{ji} leaves the range $]-L_{th}, L_{th}[$ with a decaying function driving L_{ji} to 0, the middle of the range. The connections tend to keep the same A_{ji} state, a learning rule proposed to address the problem of memory loss in noisy environments (Fusi, 2001).

The synapse block shown in figure 7.4 computes the contribution w_{ji} to the postsynaptic membrane potential by a synapse from presynaptic neuron j of type q_j, and postsynaptic neuron i of type q_i. This block has been carefully designed and tailored to spare hardware resources.

The complex neuron model was optimized for the POEtic platform. The resulting model simplification for a serial implementation is presented in figure 7.5. The same hard wired neuron can simulate multiple units sequentially, storing variable values in a local memory. Using this time multiplexing technique, 10,000 units networks can be simulated in real time.

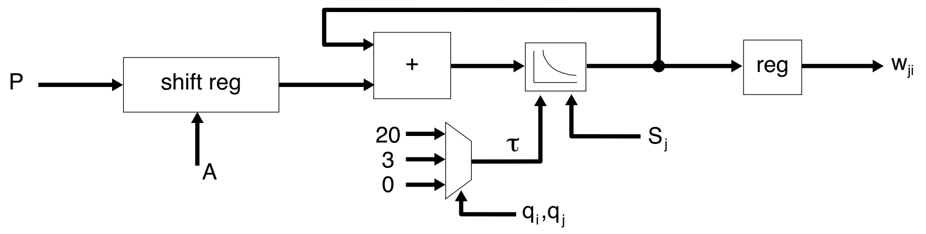


Figure 7.4: Block diagram for the synapse block. The quantum postsynaptic potential $P_{[q_j,q_i]}$ is multiplied by the value of the projection activation level $A_{ji} = 0, 1, 2, 4$ thanks to a shift register. The contribution is then added to the decaying value, which time constant is a function of the types of neurons q_j and q_i, excitatory and/or inhibitory sorted out by a multiplexer. w_{ji} has a resolution of 8 bits, but, internally, the synapse block has a 10 bits resolution.

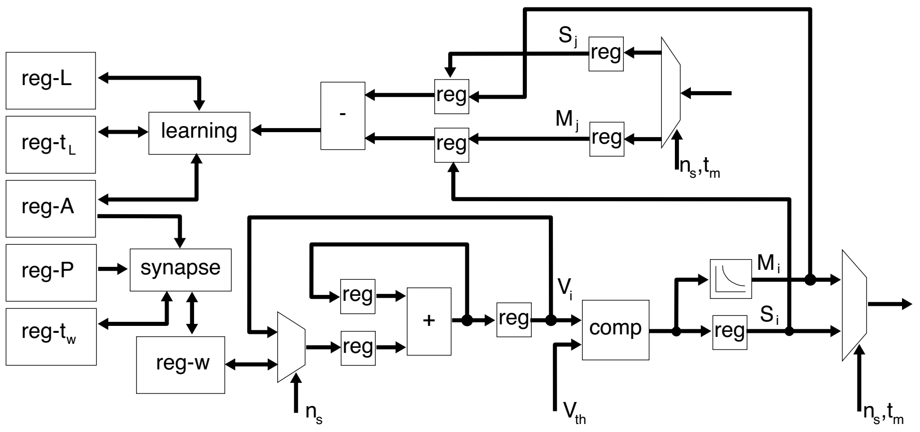


Figure 7.5: Block diagram for the neuron model serial implementation. It results from the optimization of the neuron model for the POEtic platform.

The first 90% of a project takes 90% of the time,
the last 10% takes the other 90% of the time.

– Anonymous

Chapter 8

Conclusion

Résumé En guise de conclusion, quelques remarques sur le déroulement du travail sont fournies ici. Elles mettent l'accent sur les difficultés liées à la simulation de réseaux de grande taille et sur les contributions de ce travail pour aider à y répondre.

Simulating large-scale networks is a time consuming task. The number of parameters of the simulated system, and the complexity of their non-linear interactions required an initial parameter tuning phase that lasted for several months. During this period, we had to determine the right measures to assess that the network activity corresponded to our needs, and test a wide range of parameter values before we could start any real experiment. In a first approximation, one might think that the duration of the simulations is the limiting factor. In fact, our experience taught us that, given enough computational power, the human part in the analysis of the data collected during the simulations was the real bottleneck.

One contribution of this work is the large set of tools that has been developed for the semi-automatic manipulation and analysis of the data. These tools were made user-friendly because they represent a large investment that will take years to return; they were given portable and modular designs to be able to adapt to the changes in users' needs in the forthcoming years; they were made Open Source to be accessible to anyone interested in using them, and maybe in contributing to their feature lists.

The timing of an endeavour is also an important factor for its success. On the one hand, the availability of "cheap" multi-core 64bits CPUs, inexpensive Gigabit over-copper networks, and the parallelized open source NEST simulator with a growing community of users; On the other hand, the POEtic tissue has been physically available in spring 2005 for fast hardware implementation of our model. Starting our research now would have been a perfect timing! We look forward having the opportunity to continue the investigation of our model on that novel hardware.

The world of software is about change.
Putting crap on a permanent media doesn't help it.

– Hazah

Appendix A

On the web

Some information did not fit into the permanent format of this document. Dynamic content like animations and sample data are impossible to reproduce or of little use on a printed media. A web page was set up on my personal web site to gather the electronic copy of this document, the virtual appendices like multimedia files, links to the downloadable software, and some bookmarks. You will find this page at:

`http://jiglesia.alawa.ch/phd/`

I will maintain that page along with the rest of the information on my web site for some time... probably as long as I will be involved in some kind of professional activity. I cannot swear there will be anything at all by the time you will try to access the URL.

Give a person a fish and you feed them for a day;
teach that person to use the Internet
and they won't bother you for weeks.

– Anonymous

Appendix B

Sample files

This Appendix collects sample files, one for each file format discussed in section 6.5, p.97. Check Appendix A, p.115 to find how to download those files in a digital format.

B.1 `.fnet`: `feign` network

The following file describes the state, at some simulated time step, of a 4×3 network (l.5) spiking units (11 excitatory and 1 inhibitory) wired in a complete synfire chain of length $l = 4$ and width $w = 3$. The neuromimetic model parameters are defined in the `cellTypes` header section of the document (l.7–23). Two types of units are defined in sequence: excitatory units with *cell type*=0 (l.8–14), and inhibitory units with *cell type*=1 (l.16–23). There is no limitation to the number of cell types in the file format. 4 connection types are then defined in the `connectionTypes` header section (l.25–44), one for each combination of pre– and postsynaptic unit types ($C_2^2 = 4$):

presynaptic cell type	postsynaptic cell type		connection type
0	0	$\Rightarrow$	0
0	1	$\Rightarrow$	1
1	0	$\Rightarrow$	2
1	1	$\Rightarrow$	3

In our simulations, only excitatory–excitatory connections (*type* = 0, l.26–31) were subject to spike-timing-dependent synaptic plasticity, with $A_{ji}(t) \in \{0, 1, 2, 4\}$ (l.27–30). The other types of connections were assigned $A_{ji}(t) = 1$ (l.34, l.38, and l.42).

Between l.46 and l.110, the 12 units of the network are defined in sequence along with their incoming projections. To limit the XML syntax overhead as much as possible, tag and attribute names were limited to 1-2 characters in this section. In a file containing thousands of units and millions of connections, this trick saves megabytes of text, at the expense of a less human-readable format. There is one `<c>` tag for each "cell" in the network, with the following attribute values (see section 3.2, p.18 for details):

attribute	description	value
i	unique index in network	0 to network size
t	type of unit (index of `cellType`)	0 to `cellTypes` count
m	$V_i(t)$ membrane potential	
s	$S_i(t)$ state of the cell	0 or 1
wr	$M_i(t)$ memory of previous spike when presynaptic	0 to w_max_pre
wo	$M_i(t)$ memory of previous spike when postsynaptic	0 to w_max_post
c	time steps (*clock*) since last spike	

For each cell, there is one `<p>` tag for each incoming "projection", with the following attribute values:

attribute	description	value
t	type of projection (index of `connectionType`)	0 to `connectionTypes` count
f	unit index *from* which the projection is coming	0 to network size
a	$A_{ji}(t)$ activation level of the projection	one of the levels for this type
l	$L_{ji}(t)$ learning variable	l_min to l_max

This sample file can be loaded by `feign` (section 6.1.1, p.71) as a starting point for a simulation, or transformed by `fnetdb` (section 6.2.1, p.78) into a relational database for further investigation with `fnetdig` (section 6.3.1, p.86). See section 6.5.1, p.98 for a description of the format.

```
<?xml version="1.0"?>
<state
    xmlns="http://feign.nhrg.org/schema/snn/network"
    version="1.1"
    width="4" height="3">

    <cellTypes count="2">
        <cellType
            index="0"              label="excitatory"
            k_membrane="0.954842"  refractory="3"
            reset="-78.0"          threshold="-40.0"
            k_learn_pre="0.982821" k_learn_post="0.982821"
            m_max_pre="2.0"        m_max_post="2.0"
            psp="0.84" />

        <cellType
            index="1"              label="inhibitory"
            k_membrane="0.954842"  refractory="2"
            reset="-78.0"          threshold="-40.0"
            k_learn_pre="0.982821" k_learn_post="0.982821"
            m_max_pre="2.0"        m_max_post="2.0"
            psp="-1.4" />
    </cellTypes>

    <connectionTypes count="4">
        <connectionType index="0" count="4" l_min="0.00001" l_max="20.0" k_act="0.999937">
            <activation value="0"/>
            <activation value="1"/>
            <activation value="2"/>
            <activation value="4"/>
        </connectionType>

        <connectionType index="1" count="1" l_min="0.0" l_max="0.0" k_act="0.0">
            <activation value="1"/>
        </connectionType>

        <connectionType index="2" count="1" l_min="0.0" l_max="0.0" k_act="0.0">
            <activation value="1"/>
        </connectionType>

        <connectionType index="3" count="1" l_min="0.0" l_max="0.0" k_act="0.0">
            <activation value="1"/>
        </connectionType>
    </connectionTypes>

    <cells count="12">

        <!-- layer #1 receives no projections -->
        <c i="0" t="0" m="-94.461342" s="0" wr="0.002323" wo="0.002323" c="390" />
        <c i="1" t="0" m="-91.170998" s="0" wr="0.292193" wo="0.292193" c="111" />
        <c i="2" t="0" m="-91.442390" s="0" wr="0.002715" wo="0.002715" c="381" />

        <!-- layer #2 receives projections from layer #1 -->
        <c i="3" t="0" m="-96.317589" s="0" wr="0.0" wo="0.0" c="1105">
            <p t="0" f="0" a="3" l="11.044183"/>
            <p t="0" f="1" a="3" l="10.502504"/>
            <p t="0" f="2" a="3" l="15.415247"/>
        </c>
        <c i="4" t="1" m="-68.236671" s="0" wr="0.032352" wo="0.032352" c="238">
            <p t="1" f="0" a="0" l="0.0"/>
            <p t="1" f="1" a="0" l="0.0"/>
            <p t="1" f="2" a="0" l="0.0"/>
        </c>
        <c i="5" t="0" m="-96.602310" s="0" wr="0.933032" wo="0.933032" c="44">
            <p t="0" f="0" a="3" l="8.652210"/>
            <p t="0" f="1" a="3" l="17.744703"/>
            <p t="0" f="2" a="3" l="13.020258"/>
        </c>

        <!-- layer #3 receives projections from layer #2 -->
        <c i="6" t="0" m="-89.785965" s="0" wr="0.138696" wo="0.138696" c="154">
            <p t="0" f="3" a="3" l="16.295609"/>
            <p t="2" f="4" a="0" l="0.000000"/>
            <p t="0" f="5" a="3" l="12.725195"/>
        </c>
        <c i="7" t="0" m="-46.665009" s="0" wr="0.151249" wo="0.151249" c="149">
            <p t="0" f="3" a="3" l="18.882530"/>
            <p t="2" f="4" a="0" l="0.0"/>
            <p t="0" f="5" a="3" l="11.053406"/>
        </c>
        <c i="8" t="0" m="-95.146950" s="0" wr="0.0" wo="0.0" c="949">
            <p t="0" f="3" a="3" l="11.093911"/>
            <p t="2" f="4" a="0" l="0.000000"/>
            <p t="0" f="5" a="3" l="17.009546"/>
        </c>

        <!-- layer #4 receives projections from layer #3 -->
        <c i="9" t="0" m="-97.246460" s="0" wr="0.615571" wo="0.615571" c="68">
            <p t="0" f="6" a="3" l="16.732300"/>
            <p t="0" f="7" a="3" l="9.454819"/>
            <p t="0" f="8" a="3" l="9.624744"/>
        </c>
        <c i="10" t="0" m="-50.104996" s="0" wr="0.366020" wo="0.366020" c="98">
            <p t="0" f="6" a="3" l="10.889402"/>
            <p t="0" f="7" a="3" l="11.312034"/>
            <p t="0" f="8" a="3" l="12.388318"/>
        </c>
        <c i="11" t="0" m="-80.327469" s="0" wr="1.866066" wo="1.866066" c="4">
            <p t="0" f="6" a="3" l="16.865042"/>
            <p t="0" f="7" a="3" l="12.081224"/>
            <p t="0" f="8" a="3" l="10.807630"/>
        </c>

    </cells>
</state>
```

B.2 .xpdl: experimental protocol

The following protocol describes a simulation experiment that starts with an initialization of the network (l.7) and a recovery pause (l.8), followed by a learning phase (l.10–17) during which a specific 100 time steps stimulation is presented ten times in a row (l.12–14), every 2,000 time steps (l.15). After the learning stage, the spike-timing-dependent synaptic plasticity process is switched off (l.19). For the rest of the simulation duration (l.21), 5% of the stimulus presentations (l.24) will be constituted of 3 variants of the learned stimulus in equivalent proportions (l.25–41). The other 95% of the times (l.43), the learned stimulus will be presented (compare l.13 and l.45, versus l.27, l.33 and l.38).

Tags in the `xpdl:` name space (l.3) are the transitions – protocol, sequence, loop, branch and option – defined by the language described in section 6.5.2, p.98, while the tags in the (fake) `snn:` name space (l.4) are the terminals – initialization, pause and stimulation – of the finite state machine used to represent the protocol and are interpreted by the software loading the protocol. In the case of this file, it represents a `feign` experimental protocol for the model discussed previously. See section 6.5.2, p.98 for a description of the format.

```
1   <?xml version="1.0"?>
2   <xpdl:protocol
3       xmlns:xpdl="http://feign.nhrg.org/schema/xpdl"
4       xmlns:snn="http://jiglesia.alawa.ch/schema/protocol/snn/stimulation"
5       version="1.1">
6       <xpdl:sequence>
7           <snn:initialization duration="10" />
8           <snn:pause duration="990" />
9
10          <xpdl:loop current="0" to="10" label="learning">
11              <xpdl:sequence>
12                  <snn:stimulation
13                      file="training.stim"
14                      code="52" qualifier="1" />
15                  <snn:pause duration="1900" />
16              </xpdl:sequence>
17          </xpdl:loop>
18
19          <snn:stdp swith="off" />
20
21          <xpdl:loop infinite label="experiment">
22              <xpdl:sequence>
23                  <xpdl:branch>
24                      <xpdl:option probability="0.05" label="variant">
25                          <xpdl:branch equiprobable="true">
26                              <xpdl:option>
27                                  <snn:stimulation
28                                      file="variant-001.stim"
29                                      code="52" qualifier="2" />
30                              </xpdl:option>
31                              <xpdl:option>
32                                  <snn:stimulation
33                                      file="variant-002.stim"
34                                      code="52" qualifier="3" />
35                              </xpdl:option>
36                              <xpdl:option>
37                                  <snn:stimulation
38                                      file="variant-003.stim"
```

```
                                    code="52" qualifier="4" />
                            </xpdl:option>
                        </branch>
                    </xpdl:option>
                    <xpdl:option probability="0.95" label="invariant">
                        <snn:stimulation
                            file="training.stim"
                            code="52" qualifier="1" />
                    </xpdl:option>
                </xpdl:branch>
            </xpdl:sequence>
        </xpdl:loop>
    <xpdl:sequence>
</xpdl:protocol>
```

B.3 .xyv: XY-Viewer native file format

XY-Viewer (section 6.4.2, p.89) can load .xyv files describing the data to represent according to the data model presented in figure 6.16. Each section of the document can optionally contain an unbound number of meta-informations (attributes), associating a *key* with a *value* (e.g. l.12–16). Some of these keys are interpreted by XY-Viewer, the others are ignored and can be used by other programs manipulating the file. The frame (l.6–75) contains two figures, each of them defining two axes (e.g. l.17–30), and one plot (e.g. l.32–39), encapsulating a string of points in the space defined by the axes (e.g. l.38). See section 6.5.3, p.100 for a description of the format.

```
1   <?xml version="1.0" standalone="yes"?>
2   <gallery
3       xmlns="http://xyviewer.nhrg.org/schema/xyv"
4       version="1.0">
5     <frames>
6       <frame>
7         <attributes>
8           <attribute name="title">ISI</attribute>
9         </attributes>
10        <figures>
11          <figure>
12            <attributes>
13              <attribute name="title">ISI (reg) [1,2] ../2476/2476-raw.sdf</attribute>
14              <attribute name="description">T=50.0, R=23.34, E=1164.84, F=0.46 </attribute>
15              <attribute name="class">nhrg:ISI</attribute>
16            </attributes>
17            <axes>
18              <axis name="1">
19                <attributes>
20                  <attribute name="title">Lag</attribute>
21                  <attribute name="unit">ms</attribute>
22                </attributes>
23              </axis>
24              <axis name="2">
25                <attributes>
26                  <attribute name="title">Density</attribute>
27                  <attribute name="unit">ev/s</attribute>
28                </attributes>
29              </axis>
30            </axes>
31            <plots>
32              <plot>
33                <attributes>
34                  <attribute name="title">renewal density</attribute>
35                  <attribute name="mean-firing-rate">23.339999</attribute>
36                  <attribute name="total-time-measurement">0.001000</attribute>
37                </attributes>
38                <data tuple="2" axes="1 2 "> 4.0 23999.998 5.0 7999.999  [...] </data>
39              </plot>
40            </plots>
41          </figure>
42
43          <figure>
44            <attributes>
45              <attribute name="title">ISI (reg) [1,2] ../2476/2476-raw.sdf</attribute>
46              <attribute name="description">T=50.0, R=23.34, E=1129.02, F=0.46 </attribute>
```

```
            <attribute name="class">nhrg:ISI</attribute>
          </attributes>
          <axes>
            <axis name="1">
              <attributes>
                <attribute name="title">Lag</attribute>
                <attribute name="unit">ms</attribute>
              </attributes>
            </axis>
            <axis name="2">
              <attributes>
                <attribute name="title">Density</attribute>
                <attribute name="unit">ev/s</attribute>
              </attributes>
            </axis>
          </axes>
          <plots>
            <plot>
              <attributes>
                <attribute name="title">renewal density</attribute>
                <attribute name="mean-firing-rate">23.339999</attribute>
                <attribute name="total-time-measurement">0.001000</attribute>
              </attributes>
              <data tuple="2" axes="1 2 ">10.0 8640.899740 11.0  [...] </data>
            </plot>
          </plots>
        </figure>
      </figures>
    </frame>
  </frames>
</gallery>
```

B.4 .graphml: graphs

This sample file contains a representation of the network described for the `.fnet` example. This graph being a simulated neural network is only visible by two extensions we made to the original file format: the `time` attribute (l.5) and the `activation` attribute (e.g. l.22). The `node`s of the graph are defined (l.7–18) first. Then, the `edge`s are listed as connections between a `source` node and a `target` node (l.20–48). This file can be loaded by `fnetview` (section 6.4.1, p.88). See section 6.5.4.1, p.100 for a description of the format.

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
 3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4          xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml">
 5      <graph id="synfire-chain" time="0" edgedefault="directed">
 6          <!-- nodes by pools -->
 7          <node id="0"  label="1.1" />     <node id="3"  label="2.1" />
 8          <node id="1"  label="1.2" />     <node id="4"  label="2.2" />
 9          <node id="2"  label="1.3" />     <node id="5"  label="2.3" />
10
11          <node id="6"  label="3.1" />     <node id="9"  label="4.1" />
12          <node id="7"  label="3.2" />     <node id="10" label="4.2" />
13          <node id="8"  label="3.3" />     <node id="11" label="4.3" />
14
15          <!-- pool #1 projects on pool #2 -->
16          <edge source="0" target="3"  activation="3" />
17          <edge source="0" target="4"  activation="3" />
18          <edge source="0" target="5"  activation="1" />
19          <edge source="1" target="3"  activation="3" />
20          <edge source="1" target="4"  activation="3" />
21          <edge source="1" target="5"  activation="3" />
22          <edge source="2" target="3"  activation="3" />
23          <edge source="2" target="4"  activation="3" />
24          <edge source="2" target="5"  activation="3" />
25          <!-- pool #2 projects on pool #3 -->
26          <edge source="3" target="6"  activation="3" />
27          <edge source="3" target="7"  activation="3" />
28          <edge source="3" target="8"  activation="3" />
29          <edge source="4" target="6"  activation="2" />
30          <edge source="4" target="7"  activation="3" />
31          <edge source="4" target="8"  activation="3" />
32          <edge source="5" target="6"  activation="3" />
33          <edge source="5" target="7"  activation="3" />
34          <edge source="5" target="8"  activation="3" />
35          <!-- pool #3 projects on pool #4 -->
36          <edge source="6" target="9"  activation="3" />
37          <edge source="6" target="10" activation="3" />
38          <edge source="6" target="11" activation="3" />
39          <edge source="7" target="9"  activation="3" />
40          <edge source="7" target="10" activation="2" />
41          <edge source="7" target="11" activation="3" />
42          <edge source="8" target="9"  activation="3" />
43          <edge source="8" target="10" activation="1" />
44          <edge source="8" target="11" activation="3" />
45      </graph>
46  </graphml>
```

B.5 `.sdf`: time series

A `.sdf` file is composed of a semi-structured header part (l.1–3), where *keys* and *values* are associated, and a body part (l.4–43) containing the time series enclosed between a `0,1,0` triplet and a pair of `0,2,0` and `0,FFFF,0` triplets. Only a few keys are defined by the format specification, and the others are left to the user to interpret. The data is coded by triplets of numbers: the first two numbers describe the event and the third, a time value. The first number describes the *type* of the event. The second is used to provide additional information associated to the event and is referred to as the *qualifier*. The third describes the *time* of occurrence, expressed as an interval since the previous event. Such `.sdf` files can be read by `DataToolbox` (section 6.2.2, p.79), `manip` (section 6.2.3, p.83), `RasterViewer` (section 6.4.3, p.95), and `YaTiSeWoBe` (section 6.4.4, p.96). See section 6.5.4.2, p.101 for a description of the format.

```
1    "VERSION=0"
2    "TIME_UNITS=0.001"
3    "TITLE( 0)='0:SPONT <15:52:30>'"
4    0,1,0
5    51,1,0
6    1,4,54
7    1,5,10
8    1,3,76
9    1,5,2
10   1,4,1
11   1,6,61
12   1,6,34
13   1,5,1
14   1,3,47
15   1,3,43
16   1,5,5
17   1,3,80
18   1,3,72
19   1,2,31
20   1,5,72
21   1,7,142
22   1,6,63
23   1,4,102
24   1,3,74
25   51,1,29
26   1,5,46
27   1,3,109
28   1,5,56
29   "CHKSM=1C52"
30   1,3,20
31   1,5,41
32   1,7,27
33  [...]
34   1,5,35
35   1,4,23
36   1,3,48
37   1,6,83
38   1,5,63
39   0,2,11
40   "CHKSM=17C2"
41   0,FFFF,0
```

B.6 .sng: Scriptable Network Graphics

This is an example of the subset of `.sng` files that `feign` stimulation service is able to load and apply on the network. A complex stimulus can be built on several of those `.sng` files, applying one file per time step, much like a cartoon. The first part (l.1–12) gives some meta information concerning the file, like the dimensions of the network (l.3–4). From l.13 to l.35, one can read the hexadecimal ASCII representation of the stimulus. Values are coded on 8 bits (0–255) as a pair of characters in {0, 1, 2, . . . , 9, a, b, . . . , f}, with `00` standing for value 0, `80` for value 128, and `ff` for value 255. One can distinguish an interrogation mark, but using the *sng*[1] program, this script can be compiled into a PNG image for visual inspection. This file format has also been used to record the activity of the network during the simulation to produce some of the animated sequences accessible from Appendix A, p.115. See section 6.5.4.3, p.102 for a description of the format.

```
1   #SNG: This is a sample stimulus
2   IHDR: {
3       width: 20;
4       height: 20;
5       bitdepth: 8;
6       using grayscale;
7   }
8   bKGD {gray: 0;}
9   tEXt: {
10      keyword: "Title";
11      text: "Sample SNG stimulus";
12  }
13  IMAGE: {
14      pixels hex
15  0000000000000000000000000000000000000000
16  0000000000000000000000000000000000000000
17  0000000000000000000000000000000000000000
18  0000000000000000ffffffa08000000000000000
19  00000000000000ffffffffffa080000000000000
20  000000000000ffff800000ffffa0800000000000
21  0000000000ffff8000000000ffffa08000000000
22  0000000000ffff8000000000ffffa08000000000
23  000000000000ffff000000ffffa0800000000000
24  00000000000000000000ffffa080000000000000
25  000000000000000000ffffa08000000000000000
26  0000000000000000ffffa0800000000000000000
27  0000000000000000ffffa0800000000000000000
28  0000000000000000000000000000000000000000
29  000000000000000000ffa0000000000000000000
30  0000000000000000ffffffa00000000000000000
31  0000000000000000ffffffa00000000000000000
32  000000000000000000ffa0000000000000000000
33  0000000000000000000000000000000000000000
34  0000000000000000000000000000000000000000
35  }
```

[1]http://sng.sourceforge.net/

The question of whether computers can think
is like the question of whether submarines can swim.

– Edsger Dijkstra

Appendix C

Procedure

In this section, we will go through a complete session of simulation, analysis and visualization using the software developped during the course of the present work. Look at Appendix A, p.115, to find a package containing the configuration and partial result files – in the formats described in Appendix B, p.117 – as well as the source code of the different programs and applications we are going to run.

C.1 Setup

For sake of simplicity, we will only consider here a computer system able to run a *bash* shell interpreter, that is, among others: GNU/Linux, Apple MacOS X, and Microsoft Windows with Cygwin installed. Note that we are going to compile C code into executables, which requires a compiler (GCC) and dependency libraries (libsqlite3, libxml2, libxslt and the GNU Scientific Library).

You will need to download, compile and install `feign` and `manip` according to the documentation included in each package. The compilation and installation procedure can be boiled down to:

```
1   $> tar xvzf <program>-<release>.tar.gz
2   [...]
3   $> cd <program>-<release>/
4   $> ./configure
5   [...]
6   $> make
7   [...]
8   $> make install
9   [...]
10  $> cd ..
```

After decompressing the downloaded package (l.1) and entering the newly created directory containing the source code for the program (l.3), all the dependencies will be checked (l.4). This command will fail loudly in case of a missing library on your system. Conform to the provided explanations and fix all problems before running line 4 again. Once the previous step is completed successfully, the software will be compiled (l.6) into an executable suitable for your computer, that will finally be copied to */usr/local/bin* (l.8) according to the default behaviour. Specific details are available from the package documentation, like, for instance, the indication to run *./configure --enable-simul=snn* for the `feign` package to compile the appropriate simulation module into `feign`.

After compiling and installing `feign` and `manip`, you should be able to run the following commands, bearing in mind that some details, like the release numbers, might differ from the printed information:

```
$> feign --help
NHRG Feign Simulator 0.10.0
Copyright (c) 2003-2005, INFORGE-NHRG, javier iglesias <javier.iglesias@alawa.ch>
Submit bug reports to http://sourceinforge.unil.ch/bugs/?group=nhrg-feign

Specialized neural network simulator that extracts most of it’s
functionalities from compiled modules:

  config     : FILE config        ($Id: config-file.cc 14 2005-04-06 16:38:09Z jiglesia $)
  logger     : FILE logger        ($Id: logging-file.cc 14 2005-04-06 16:38:09Z jiglesia $)
  output     : SDF output         ($Id: output-sdf.cc 14 2005-04-06 16:38:09Z jiglesia $)
  protocol   : XSPDL              ($Id: protocol.cc 28 2005-04-08 17:42:15Z jiglesia $)
  simulation : SNN 2D simulation  ($Id: simul-snn.cc 62 2005-06-01 08:35:58Z jiglesia $)

[...]

$> manip --help
NHRG Data Manipulation Tool, Javier Iglesias <javier.iglesias@alawa.ch>
Copyright (c) 2002-2005, NHRG-INFORGE
release:  NHRG Analyses 0.5.0
revision: $Id: manip.cc,v 1.11 2005/05/27 07:35:02 jiglesia Exp $

Type ’manip -G’ or ’manip --generate’
to produce a documented example of a configuration file.

[...]
```

Now that the software is ready, it is time to get the configuration package from the web site (see Appendix A, p.115) and decompress it in an appropriate location on your computer. The *cheating* directory contains partial result files, should you wish to jump over a step.

```
$> tar xvzf procedure.tar.gz
[...]
$> cd procedure/
$> ls
cheating
simulation
manipulation
visualization
$> ls cheating/
simulation.sdf
data.sdf
correlations.xyv
```

C.2 Simulation

```
$> cd simulation/
$> ls
simul.ini
no-input.xpdl
```

```
 5  recorded-times.mask
 6  recorded-units.mask
 7  $> feign
 8  [...]
 9  $> ls
10  simul.log
11  simulation.sdf
12  dump-00000000.xml
13  dump-00010000.xml
14  input.log
15  input-protocol.log
16  sneak.activations
17  [...]
18  $> cp simulation.sdf ../manipulation/
19  $> cd ../
```

We will now run a simulation using the `feign` simulator compiled in the previous section. *simul.ini* (l.3) is read at start-up to determine the details of the simulation. Among other things, the network size, the simulation duration, the parameters of the simulated units and connections, the stimulation protocol *no-input.xpdl* (l.4), the recorded times *recorded-times.mask* (l.5) and range of units *recorded-units.mask* (l.6) are defined.

After running for a few seconds (l.8), you can check that the simulator produced a few new files in the same directory: *simul.log* (l.10) containing the runtime log; *simulation.sdf* (l.11) containing the time series of the formated recorded units; two network files *dump-00000000.xml* and *dump-00010000.xml* (l.11-12) containing respectively the complete network status at $t = 0$ and $t = 10^4$ time steps; and a set of *sneak.** files providing information on the network activity evolution. All these files can be read and modified using an ASCII text editor.

The file *simulation.sdf* is considered in this procedure as the result of the simulation and will be further analysed. This is the reason why we copy it to the *manipulation* directory (l.18). Note that the network files could be analysed further to extract the circuits or displayed. This will not be discussed here. It is also possible to instruct `feign` to continue a simulation from one of these network status files, or another one that could have been wired by hand. You are invited to edit the configuration using an ASCII text editor and run simulations with larger networks, or longer durations to experience the ease of use.

C.3 Manipulation

```
 1  $> cd manipulation/
 2  $> ls
 3  manip.cfg
 4  simulation.sdf
 5  $> manip -B simulation.sdf
 6  Source      : simulation.sdf
 7  Time unit   : 0.001000 [s]
 8  Duration    : 10.00 [s]
 9  Sections    : 1
10  Annotations:
11    TITLE(0)  : generated by SDF output $Id: output-sdf.cc 14 2005-04-06 16:38:09Z jiglesia $
12  Events:
13    1,0       :      61 episodes =>  6.10 [ep/s]
14    1,1       :      93 episodes =>  9.30 [ep/s]
15  [...]
```

```
16    1,18E     :      90 episodes =>  9.00 [ep/s]
17    1,18F     :      97 episodes =>  9.70 [ep/s]
18                ----------------
19                  31136 episodes
20
21  $> manip -f manip.cfg -o data.sdf simulation.sdf
22  $> manip -B data.sdf
23  Source      : data.sdf
24  Time unit   : 0.001000 [s]
25  Duration    : 10.00 [s]
26  Sections    : 1
27  Annotations:
28    TITLE(0)  : generated by SDF output $Id: output-sdf.cc 14 2005-04-06 16:38:09Z jiglesia $
29    SOURCE    : simulation.sdf
30    MANIP     : # Tue Sep 27 12:47:25 2005
31    MANIP     : # $Id: manip.cc,v 1.12 2005/05/27 14:03:58 jiglesia Exp $
32    MANIP     : INCLUDE   1,FA&FFFF ;
33    MANIP     : INCLUDE   1,12&FFFF ;
34    MANIP     : INCLUDE   1,A&FFFF ;
35    TITLE(99) : formatted by $Id: SDFFormatter.cc,v 1.5 2005/02/03 16:41:42 jiglesia Exp $
36  Events:
37    1,A       :      60 episodes =>  6.00 [ep/s]
38    1,12      :      62 episodes =>  6.20 [ep/s]
39    1,FA      :      76 episodes =>  7.60 [ep/s]
40                ----------------
41                    198 episodes
42
43  $> cd ../
```

We are not going to analyse the complete set of time series produced by the simulation and stored in file *simulation.sdf*. To extract only the time series we are interested in from all the available ones (l.13-17), we will use the manip tool. The manipulation details are configured in the *manip.cfg* script. Like all other files in this procedure, this configuration can be edited using an ASCII text editor. There, you can read that manip is instructed to extract three time series from the original data, those corresponding to the units #A, #12 and #FA in hexadecimal notation, and to units #10, #18 and #250 in decimal notation.

Issuing *manip -B <file>* (l.5, 22) prints a human readable description of the contents of the *<file>* to the terminal. Note that manip leaves traces of all the actions it undertakes (l.30-34) in the output file to help recovering the memory of the different processings a data set could have gone through. Reader is invited to consult the configuration file *manip.cfg* to discover the numerous functionalities offered by manip. Such a fresh, up-to-date and empty commented configuration file can be produced at any time issuing *manip -G -o manip.cfg.*

C.4 Analysis

The data analysis will be performed online, thanks to the OAN framework:

http://openadap.net/

Please read the online documentation for the details of how to run the program. This part is too volatile to be described on a paper medium. Note that, at some point, you will be asked to pre-process your data. At that point, you should explicitly confirm that you want all the *1.** event types to be included for analysis.

The analysis we will perform here is named *correlograms.* After the analysis is successfully completed (this should take a few seconds), you will be proposed to *plot graphics* directly from the analysis results page. This is equivalent to applying the visualization procedure explained in the next section. Would you wish to save the analysis results for future inspection, you should download and save the output of the analysis to your hard disk. You are invited to save this data to the *visualization* directory of the *package* tree, under the file name *correlations.xyv.*

C.5 Visualization

To visualize large number of graphics and produce reports in a semi-automatic way on multiple platforms, `XY-Viewer` is an efficient solution. It can be installed, updated and launched from the web, thanks to the Java Web Start Technology, by browsing to the page:

`http://jiglesia.alawa.ch/phd/`

You will find the *Launch Me* link to `XY-Viewer` in the *Software* section. From the main application window, it is possible to load the data files through the usual *File > Open* menu item, or the appropriate button in the toolbar.

Bibliography

Abbott, L. F., Nelson, S. B., November 2000. Synaptic plasticity: taming the beast. Nature Neuroscience 3, 1178–83.

Abeles, M., 1982a. Local cortical circuits. Springer-Verlag, Berlin.

Abeles, M., 1982b. Quantification, smoothing, and confidence limits for single-units' histograms. Journal of Neuroscience Methods 5, 317–325.

Abeles, M., 1991. Corticonics: Neural Circuits of the Cerebral Cortex, 1st Edition. Cambridge University Press.

Abeles, M., Gat, I., 2001. Detecting precise firing sequences in experimental data. Journal of Neuroscience Methods 107, 141–154.

Abeles, M., Gerstein, G. L., 1988. Detecting spatiotempral firing patterns among simultaneously recorded single neurons. Journal of Neurophysiology 60 (3), 910–924.

Abeles, M., Hayon, G., Lehmann, D., 2004. Modeling compositionality by dynamic binding of synfire chains. Journal of Computational Neuroscience 17, 179–201.

Abraham, W. C., Bear, M. F., April 1996. Metaplasticity: the plasticity of synaptic plasticity. Trends in Neurosciences 19 (4), 126–30.

Adams, R., teBoekhorst, R., Rust, A. G., Kaye, P., Schilstra, M., 2004. Design of spatially extended neural. IJCNN-2004.

Aghajanian, G. K., Bloom, F. E., December 1967. The formation of synaptic junctions in developing rat brain: A quantitative electron microscopic study. Brain Research 6 (4), 716–27.

Aihara, K., Tokuda, I., 2002. Possible neural coding with interevent intervals of synchronous firing. Physical Review E 66.

Albert, R., Barabasi, A. L., 2002. Statistical mechanics of complex networks. Rev. Mod. Phys. 74, 47–97.

Albo, Z., Prisco, G. V., Chen, Y., Kangarajan, G., Trucculo, W., 2004. Is partial coherence a viable technique for identifying generators of neural oscillations? Biol. Cybern. 90, 318–26.

Amin, H. H., Fujii, R. H., 2004. Spike train decoding scheme for a spiking neural. IJCNN-2004.

Amir, Y., Harel, M., Malach, R., 1993. Cortical hierarchy reflected in the organization of intrinsic connections in macaque monkey visual cortex. Journal of Comparative Neurology 334 (1), 19–46.

Amit, D. J., Mongillo, G., 2003. Spike-driven synaptic dynamics generating working memory states. Neural Computation 15, 565–96.

Arnoldi, H. R., Englmeier, K., Brauer, W., 1999. Translation-invariant pattern recognition based on synfire chains. Biol. Cybern. 80, 433–47.

Aviel, Y., Horn, D., Abeles, M., March 2004. Synfire waves in small balanced networks. Neurocomputing 58-60, 123–7.

Aviel, Y., Horn, D., Abeles, M., 2005. Memory capacity of balanced networks. Neural Computation 17, 691–713.

Aviel, Y., Mehring, C., Abeles, M., Horn, D., June 2003. On embedding synfire chains in a balanced network. Neural Computation 15 (6), 1321–40.

Awiszus, F., 1997. Spike train analysis. Journal of Neuroscience Methods 74, 155–166.

Baudry, M., Davis, J. L., Thompson, R. F., 2000. Advances in Synaptic Plasticity, 1st Edition. Vol. 1. MIT Press.

Bear, M. F., Connors, B. W., Paradiso, M. A., 1996. Neuroscience: Exploring the Brain. Williams and Wilkins.

Bell, C. C., Han, V. Z., Sugawara, Y., Grant, K., May 1997. Synaptic plasticity in a cerebellum-like structure depends on temporal order. Nature 387 (6630), 278–281.

Ben-Shaul, Y., Bergman, H., Ritov, Y., Abeles, M., 2001. Trial to trial variability in either stimulus or action causes apparent correlation and synchrony in neuronal activity. Journal of Neuroscience Methods 111, 99–110.

Bi, G., 2002. Spatiotemporal specificity of synaptic plasticity: cellular rules and mechanisms. Biol. Cybern. 87, 319–32.

Bi, G., Poo, M., 1998. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. Journal of Neuroscience 18 (24), 10464–10472.

Bi, G., Wang, H., 2002. Temporal asymmetry in spike timing-dependent synaptic plasticity. Physiology and behaviour 77 (2002), 551–555.

Bienenstock, E., May 1995. A model of neocortex. Network: Comput. Neural Syst. 6 (2), 179–224.

Boergers, C., Kopell, N., 2005. Effects of noisy drive on rhythms in networks of excitatory and inhibitory neurons. Neural Computation 17, 557–608.

Bourgeois, J., Rakic, P., 1993. Changes of synaptic density in the primary visual cortex of the macaque monkey from fetal to adult stage. Journal of Neuroscience 13, 2801–20.

Braitenberg, V., Schuez, A., 1998. Cortex: statistics and geometry of neuronal connectivity, 2nd Edition. Springer, Berlin.

Brodmann, K., 1909. Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues. Barth, Leipzig.

Brunel, N., May 2000. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. Journal of Computational Neuroscience 8 (3), 183–208.

Burkitt, A. N., Meffin, H., Grayden, D. B., 2004. Spike-timing-dependent plasticity: The relationship to rate-based learning for models with weight dynamics determined by a stable fixed point. Neural Computation 16, 885–940.

Caldarelli, G., Castellano, C., Petri, A., 1999. Criticality in models for fracture in disordered media. Physica A 270, 15–20.

Cateau, H., Fukai, T., 2003. A stochastic method to predict the consequence of arbitrary forms of spike-timing-dependent plasticity. Neural Computation 15, 597–620.

Changizi, M. A., February 2001. Principles underlying mammalian neocortical scaling. Biological Cybernetics 84 (3), 207–15.

Chechik, G., 2003. Spike timing dependent plasticity and relevant information maximization. Neural Computation 15 (7), 1481–1510.

Chechik, G., Meilijson, I., Ruppin, E., 1997. Synaptic pruning : A novel account in neural terms. Computational Neuroscience Meeting Proceedings.

Chechik, G., Meilijson, I., Ruppin, E., October 1998. Synaptic pruning in development: A computational account. Neural Computation 10 (7), 1759–77.

Chechik, G., Meilijson, I., Ruppin, E., 1999. Neuronal regulation: A mechanism for synaptic pruning during brain maturation. Neural Computation 11, 2061–2080.

Chialvo, D. R., 2004. Critical brain networks. Physica A 340, 756–765.

Chialvo, D. R., Bak, P., June 1999. Learning from mistakes. Neuroscience 90 (4), 1137–48.

Christen, M., Kern, A., Nikitchenko, A., Steeb, W., Stoop, R., 2004a. Fast spike pattern detection using the correlation integral. Phys. Rev. E 70, 0119011–7.

Christen, M., Kern, A., Stoop, R., 2003. A correlation integral based method for pattern recognition in series of interspike intervals. Proceedings of the 11th Workshop on the Nonlinear Dynamics of Electronic Systems, 49–52.

Christen, M., Kern, A., van der Vyver, J., Stoop, R., 2004b. Pattern detection in noisy signals. Proc. of ISCAS 4, 669–72.

Cisse, Y., Crochet, S., Timofeev, I., Steriade, M., 2004. Synaptic responsiveness of neocortical neurons to callosal volleys during paroxysmal depolarizing shifts. Neuroscience 124, 231–239.

Cossart, R., Aronov, D., Yuste, R., 5 2003. Attractor dynamics of network up states in the neocortex. Nature 423, 283–288.

Cox, D. R., Isham, V., 1980. Point Processes, 1st Edition. Monographs on Applied Probability and Statistics. Chapman and Hall.

Crick, F., 1989. Neural edelmanism. Trends in Neurosciences 12 (7), 240–8.

Dayan, P., Abbott, L. F., 2001. Theoretical Neuroscience, 1st Edition. Vol. 1. MIT Press.

De Bellis, M. D., Keshavan, M. S., Beers, S. R., Hall, J., Frustaci, K., June 2001. Sex differences in brain maturation during childhood and adolescence. Cerebral Cortex 11 (6), 552–557.

De Los Rios, P., 2001. Power law size distribution of supercritical random trees. Europhys. Lett. 56 (6), 898–903.

DeFelipe, J., Alonso-Nanclares, L., Arellano, J. I., March 2002. Microstructure of the neocortex: Comparative aspects. Journal of Neurocytology 31 (3), 299–316.

Delorme, A., Thorpe, S. J., November 2003a. Spikenet: an event-driven simulation package for modelling large networks of spiking neurons. Network: Comput. Neural Syst. 14 (4), 613–627.

Delorme, A., Thorpe, S. J., 2003b. Spikenet: an event-driven simulation package for modelling large networks of spiking neurons. Network: Comput. Neural Syst. 14, 613–27.

Derakhshani, R., Schuckers, S. A., 2004. Continuous time delay neural networks for detection of temporal patterns in signals. IJCNN-2004.

Diesmann, M., Gewaltig, M., Aersten, A., 1999. Stable propagation of synchronous spiking in cortical neural networks. Nature 402 (6761), 529–533.

Diesmann, M., Gewaltig, M., Rotter, S., Aertsen, A., 2001. State space analysis of synchronous spiking in cortical neural networks. Neurocomputing 38-40, 565–71.

Doboli, S., Minai, A. A., 2004. Using latent attractors to discern temporal order. IJCNN-2004.

Douglas, R. J., Martin, K. A., 1991. A functional microcircuit for cat visual cortex. Journal of Physiology 440, 735–69.

Edelman, G. M., February 1993. Neural darwinism: Selection and reentrant signaling in higher brain function. Neuron 10 (2), 115–25.

Elston, G. N., 2003. Cortex, cognition and the cell: New insights into the pyramidal neuron and prefrontal function. Cerebral Cortex 13 Supplement 1001 (11), 1124–1138.

Emsley, J. G., Mitchell, B. D., Kempermann, G., Mackli, J. D., April 2005. Adult neurogenesis and repair of the adult cns with neural progenitors, precursors, and stem cells. Progress in Neurobiology 75 (5), 321–41.

Engel, D., Pahner, I., Schulze, K., Frahm, C., Jarry, H., 2001. Plasticity of rat central inhibtiory synapses through gaba metabolism. Journal of Physiology 535 (2), 473–482.

Eriksson, J., Torres, O., Mitchell, A., Tucker, G., Lindsay, K., 2003. Spiking neural network for reconfigurable poetic tissue. Lecture Notes in Computer Science 2606, 165–173.

Eriksson, J., Villa, A. E., January-March 2005. Event-related potentials in an auditory oddball situation in the rat. Biosystems 79 (1), 207–212.

Foldy1, C., Dyhrfjeld-Johnsen, J., Soltesz, I., November 2004. Structure of cortical microcircuit theory. Journal of Physiology 562 (1), 47–54.

Foss, J., Milton, J., August 2000. Multistability in recurrent neural loops arising from delay. J. Neurophysiol. 84 (2), 975–85.

Frazor, R. A., Albrecht, D. G., Geisler, W. S., Crane, A. M., 2004. Visual cortex neurons of monkeys and cats: Temporal dynamics of the spatial frequency response function. J. Neurophysiol 91, 2607–27.

Friedman, J. H., 1974. Data analysis techniques for high energy particle physics. Proc. of the 1974 CERN School of Computing, 271–366.

Froemke, R. C., Dan, Y., 2002. Spike-timing-dependent synaptic modification induced by natural spike trains. Nature 416 (6879), 433–438.

Fusi, S., June 2001. Long term memory: Encoding and storing strategies of the brain. Neurocomputing 38-40, 1223–8.

Fusi, S., 2002. Hebbian spike-driven synaptic plasticity for learning patterns of mean firing rates. Biol. Cybern. 87, 459–70.

Fusi, S., Annunziato, M., Badoni, D., Salamon, A., Amit, D. J., 2000. Spike-driven synaptic plasticity: Theory, simulation, vlsi implementation. Neural Computation 12 (10), 2227–2258.

Fusi, S., Drew, P. J., Abbott, L. F., 2005. Cascade models of synaptically stored memories. Neuron 45, 599–611.

Gardner, E. P., Palmer, C. I., Hamalinen, H., Warren, S., 1992. Simulation of motion on the skin. v. effect of stimulus temporal frequency on the representation of moving bar patterns in primary somatosensory cortex of monkeys. Journal of Neurophysiology 67 (1), 37–63.

Gat, I., October 2000. Ladies and gentleman: The ultimate cutengine.

Gewaltig, M., Diesmann, M., Aersten, A., March 2001. Propagation of cortical synfire activity: survival probability in single trials and stability in the mean. Neural Networks 14, 657–73.

Gomez, L., Budelli, R., Saa, R., Stiber, M., Segundo, J. P., February 2005. Pooled spike trains of correlated presynaptic inputs as realizations of cluster point processes. Biological Cybernetics 92 (2), 110–27.

Gutig, R., Aertsen, A., Rotter, S., 2003. Analysis of higher-order neuronal interactions based on conditional inference. Biol. Cybern. 88, 352–359.

Guyonneau, R., Van Rullen, R., Thorpe, S. J., 2005. Neurons tune to the earliest spikes through stdp. Neural Computation 17, 859–79.

Hamaguchi, K., Aihara, K., 2004. Quantitative information transfer through layers of spiking neurons connected by mexican-hat-type connectivity. Neurocomputing 58-60, 85–90.

Harris, K. D., May 2005. Neural signatures of cell assembly organization. Nature Reviews Neuroscience 6, 399–407.

Hayon, G., Abeles, M., Lehmann, D., 2005. A model for representing the dynamics of a system of synfire chains. Journal of Computational Neuroscience 18, 41–53.

Hebb, D., 1949. The organization of behavior. John Wiley, New York.

Hellwig, B., January 2000. A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex. Biological Cybernetics 82 (2), 111–121.

Herrmann, M., Hertz, J. A., Prügel-Bennett, A., February 1995. Analysis of synfire chains. Network: Computation in Neural Systems 13 (1), 115–129.

Hertz, J. A., Prugel-Bennett, A., 1996a. Learning short synfire chains by self-organization. Network: Computation in Neural Systems 7, 357–363.

Hertz, J. A., Prugel-Bennett, A., September 1996b. Learning synfire chains: turning noise into signal. International Journal of Neural Systems 7 (4), 445–50.

Hess, G., Donoghue, J. P., June 1994. Long-term potentiation of horizontal connections provides a mechansim to reorganize cortical motor maps. Journal of Neurophysiology 71 (6), 2543–7.

Hilgetag, C., Burns, G. A., O'Neill, M. A., Scannell, J. W., 2000. Anatomical connectivity defines the organization of clusters of cortical areas in the macaque and the cat. Philosophical Transactions: Biological Sciences 355 (1393), 91–110.

Hill, S., Villa, A. E., 1997. Dynamic transitions in global network activity influenced by the balance of excitation and inhibtion. Network: computational neural networks 8, 165–184.

Hopfield, J. J., 1982. Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci USA 79 (8), 2554–8.

Hopfield, J. J., Brody, C. D., 2000. What is a moment? "cortical" sensory integration over a brief interval. Proc Natl Acad Sci U S A 97 (25), 13919–13924.

Hopfield, J. J., Brody, C. D., 2004. Learning rules and network repair in spike-timing-based computation networks. Proc Natl Acad Sci U S A 101 (1), 337–342.

Horn, D., Levy, N., Ruppin, E., January 1998. Memory maintenance via neuronal regulation. Neural Computation 10 (1), 1–18.

Hosaka, R., Ikeguchi, T., Nakamura, H., Araki, O., 2004. Information transformation from a spatiotemporal. IJCNN-2004.

Hubel, D. H., Wiesel, T. N., LeVay, S., April 1977. Plasticity of ocular dominance columns in monkey striate cortex. Philos Trans R Soc Lond B Biol Sci 278 (961), 377–409.

Huttenlocher, P. R., March 1979. Synaptic density in human frontal cortex – developmental changes and effects of aging. Brain Research 163 (2), 195–205.

Huttenlocher, P. R., de Courten, C., Garey, L. J., Van der Loos, H., December 1982. Synaptogenesis in human visual cortex – evidence for synapse elimination during normal development. Neuroscience Letters 33 (3), 247–252.

Iglesias, J., Eriksson, J., Grize, F., Tomassini, M., Villa, A. E., 2005a. Dynamics of pruning in simulated large-scale spiking neural networks. BioSystems 79 (1), 11–20.

Iglesias, J., Eriksson, J., Pardo, B., Tomassini, M., Villa, A. E., 2005b. Emergence of oriented cell assemblies associated with spike-timing-dependent plasticity. Lecture Notes in Computer Science.

Innocenti, G. M., September 1995. Exuberant development of connections, and its possible permissive role in cortical evolution. Trends in Neurosciences 18 (9), 397–402.

Izhikevich, E. M., 2004. Which model to use for cortical spiking neurons? IEEE Transactions on Neural Networks 15 (5), 1063–70.

Izhikevich, E. M., Gally, J. A., Edelman, G. M., August 2004. Spike-timing dynamics of neuronal groups. Cerebral Cortex 14, 933–44.

Jahnke, A., Schonauer, T., Roth, U., Mohraz, K., Klar, H., October 1997. Simulation of spiking neural networks on different hardware platforms. Lecture Notes in Computer Science 1327, 1187–92.

Jin, D. Z., 2004. Spiking neural network for recognizing spatiotemporal sequences of spikes. Physical Review E 69, 021905.

Kanamaru, T., Sekine, M., 2005. Synchronized firings in the networks of class 1 excitable neurons with excitatory and inhibitory connections and their dependences on the forms of interactions. Neural Computation 17, 1315–38.

Kantor, D. B., Kolodkin, A. L., June 2003. Curbing the excesses of youth: Molecular insights into axonal pruning. Neuron 38, 849–52.

Karmarkar, U. R., Buonomano, D. V., 2002. A model of spike-timing dependent plasticity: one or two coincidence detectors? J Neurophysiol 88 (1), 507–513.

Karmarkar, U. R., Najarian, M. T., Buonomano, D. V., 2002. Mechanisms and significance of spike-timing dependent plasticity. Biol. Cybern. 87, 373–82.

Kelso, S. R., Ganong, A. H., Brown, T. H., 1986. Hebbian synapses in hippocampus. Proc Natl Acad Sci U S A 83 (14), 5326–5330.

Kemp, B., Olivan, J., Sept. 2003. European data format plus (edf+), an edf alike standard format for the exchange of physiological data. Clin Neurophysiol 114 (9), 1755–61.

Kepecs, A., van Rossum, M. C., Song, S., Tegner, J., 2002. Spike-timing-dependent plasticity: common themes and divergent vistas. Biological Cybernetics 87, 446–458.

Kim, B., October 2004. Geographical coarse graining of complex networks. Physical Review Letters 93 (16), 168701.

Kitano, K., Cateau, H., Fukai, T., May 2002. Self-organization of memory activity through spike-timing-dependent plasticity. Neuroreport 13 (6), 795–8.

Kitano, K., Fukai, T., 2002. A multiple synfire-chain model for the predictive synchrony in the motor-related cortical areas. NIP 4, 1634–38.

Knoblauch, A., 2003. Synchronization and pattern separation in spiking associative memories and visual cortical areas. Ph.D. thesis, Universität Ulm, http://www.informatik.uni-ulm.de/ni/publ/AKnoblauch/Knoblauch2003B.pdf.

Knoblauch, A., Palm, G., 2004. What is signal and what is noise in the brain? BioSystems.

Kopka, H., Daly, P. W., 1999. A Guide to LaTeX, 3rd Edition. Addison Wesley Longman Ltd.

Kuhn, A., Rotter, S., Aertsen, A., June 2002. Correlated input spike trains and their effects on the response of the leaky integrate-and-fire neuron. Neurocomputing 44-46, 121–6.

Lestienne, R., Tuckwell, H. C., 1998. The significance of precisely replicating patterns in mammalian cns spike trains. Neuroscience 82, 315–336.

Levy, N., Horn, D., Meilijson, I., Ruppin, E., July 2001. Distributed synchrony in a cell assembly of spiking neurons. Neural Networks 14 (6), 815–24.

Lichtman, J. W., Colman, H., February 2000. Synapse elimination and indelible memory. Neuron 25 (2), 269–78.

Litvak, V., Sompolinsky, H., Segev, I., Abeles, M., 2003. On the transmission of rate code in long feedforward networks with excitatory inhibitory balance. The Journal of Neuroscience 23 (7), 3006–3015.

Lorente de No, R., 1949. Cerebral cortex: architecture, intracortical connections, motor projections. Oxford University Press.

Lovelace, J. J., Cios, K. J., 2004. Dual threshold based neural modeling to study. IJCNN-2004.

Lumer, E. D., Edelman, G. M., Tononi, G., apr/may 1997. Neural dynamics in a model of the thalamocortical system. i. layers, loops and the emergence of fast synchronous rhythms. Cerebral Cortex 7, 207–227.

Lytton, W. W., Hines, M. L., 2005. Independent variable time-step integration of individual neurons for network simulations. Neural Computation 17, 903–21.

Maass, W., Natschlaeger, T., Markram, H., 2002. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation 14 (11), 2531–60.

MacGregor, R. J., 1991. Sequential configuration model for firing patterns in local neural networks. Biological Cybernetics 65, 339–349.

MacGregor, R. J., 1993. Composite cortical networks as systems of multimodal oscillators. Biological Cybernetics 69, 243–255.

MacGregor, R. J., Ascarrunz, F. G., Kisley, M. A., 1995. Characterization, scaling, and partial representation of neural junctions and coordinated firing patterns by dynamic similarity. Biological Cybernetics 73, 155–166.

MacGregor, R. J., Gerstein, G. L., 1991. Cross-talk theory of memory capacity in neural networks. Biological Cybernetics 65, 351–355.

MacGregor, R. J., Tajchman, G., August 1988. Theory of dynamic similarity in neuronal systems. Journal of Neurophysiology 60 (2), 751–68.

Malinow, R., Malenka, . C., March 2002. Ampa receptor trafficking and synaptic plasticity. Annual Review of Neuroscience 25, 103–26.

Mange, D., Tomassini, M., 1998. Bio-Inspired Computing Machines: Towards novel Computational Architectures. Presses Polytechniques et Universitaires Romandes.

Markram, H., Lubke, J., Frotscher, M., Sakmann, B., 1997. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. Science 275 (5297), 213–215.

Masuda, N., Aihara, K., 2004. Self-organizing dual coding based on spike-time-dependent plasticity. Neural Computation 16, 627–63.

Matsumoto, N., Okada, M., December 2002. Self-regulation mechanism of temporally asymmetric hebbian plasticity. Neural Computation 14 (12), 2883–902.

Matsumoto, N., Okada, M., September 2004. Impact of deviation from precise balance of spike-timing-dependent plasticity. Neural Networks 17 (7), 917–24.

Mattia, M., Del Giudice, P., 2000. Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. Neural Computation 12, 2305–29.

Mehring, C., Hehl, U., Kubo, M., Diesmann, M., Aersten, A., 2003. Activity dynamics and propagation of synchronous spiking in locally connected random networks. Biological Cybernetics 88, 395–408.

Mehta, M. R., Lee, A. K., Wilson, A., June 2002. Role of experience and oscillations in transforming a rate code into a temporal code. Nature 417, 741–6.

Mimura, K., Kimoto, T., Okada, M., 2003. Synapse efficiency diverge due to synaptic pruning following over-growth. Phys Rev E Stat Nonlin Soft Matter Phys 68, 031910.

Montgomery, J. M., Madison, D. V., February 2002. State-dependent heterogeneity in synaptic depression between pyramidal cell pairs. Neuron 33 (5), 765–777.

Montgomery, J. M., Madison, D. V., December 2004. Discrete synaptic states define a major mechanism of synapse plasticity. Trends in Neurosciences 27 (12), 744–750.

Morrison, A., Mehring, C., Geisel, T., Aersten, A., Diesmann, M., 2005. Advancing the boundaries of high connectivity network simulation with distributed computing. Neural Computation.

Mountcastle, V. B., July 1957. Modality and topographic properties of single neurons in cat's somatic sensory cortex. Journal of Neurophysiology 20, 408–34.

Nakamura, H., O'Leary, D. D., November 1989. Inaccuracies in initial growth and arborization of chick retinotectal axons followed by course corrections and axon remodeling to develop topographic order. Journal of Neuroscience 9 (11), 3776–95.

Newman, M. E., Watts, D. J., Strogatz, S. H., February 2002. Random graph models of social networks. PNAS 99, 2566–72.

Nowotny, T., Huerta, R., 2003. Explaining synchrony in feed-forward networks: Are McCulloch-Pitts neurons good enough? Biol. Cybern. 89, 237–41.

Nowotny, T., Rabinovich, M. I., Abarbanel, H., 2003a. Spatial representation of temporal information through spike-timing-dependent plasticity. Physical Review E 68, 0119081–0119089.

Nowotny, T., Zhigulin, V. P., Selverston, A. I., Abarbanel, H. D., Rabinovich, M. I., October 2003b. Enhancement of synchronization in a hybrid neural circuit by spike-timing dependent plasticity. Journal of Neuroscience 23 (30), 9776–85.

Okatan, M., Grossberg, S., September 2000. Frequency-dependent synaptic potentiation, depression and spike timing induced by hebbian pairing in cortical pyramidal neurons. Neural Networks 13 (7), 699–708.

Oram, M. W., Wiener, M. C., Wiener, M. C., Richmond, B. J., 1999. Stochastic nature of precisely timed spike patterns in visual system neuronal responses. J. Neurophysiol 81, 3021–3033.

Palotai, Z., Szirtes, G., Lorincz, A., 2004. Emerging evolutionary features in noise driven. IJCNN-2004.

Pantic, L., Torres, J. J., Kappen, H. J., Gielen, S. C., December 2002. Associative memory with dynamic synapses. Neural Computation 14 (12), 2903–23.

Peterson, K. B., september 2004. The matrix cookbook.
URL `http://www.imm.dtu.dk/pubdb/p.php?3274`

Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T., 1992. Numerical recipes in C: The art of scientific computing, 2nd Edition. Cambridge University Press, http://www.nr.com/.

Prut, Y., Vaadia, E., Bergman, H., Slovin, H., Abeles, M., 1998. Spatiotemporal structure of cortical activity: Properties and behavioral relevance. J. Neurophysiol 79, 2857–2874.

Purves, D., LaMantia, A. S., August 1990. Numbers of ”blobs” in the primary visual cortex of neonatal and adult monkeys. Proc Natl Acad Sci USA 87 (15), 5764–7.

Quenet, B., Horcholle-Bossavit, G., Wohrer, A., Dreyfus, G., January-March 2005. Formal modeling with multistate neurones and multidimensional synapses. Biosystems 79 (1), 21–32.

Quenet, B., Horn, D., February 2003. The dynamic neural filter: A binary model of spatiotemporal coding. Neural Computation 15 (2), 309–29.

Rakic, P., February 1974. Neurons in rhesus monkey visual cortex: Systematic relation between time of origin and eventual disposition. Science 183 (4123), 425–427.

Rakic, P., November 1981. Development of visual centers in the primate brain depends on binocular competition before birth. Science 214 (4523), 928–931.

Rakic, P., Bourgeois, J., Eckenhoff, M. F., Zecevic, N., Goldman-Rakic, P. S., 1986. Concurrent overproduction of synapses in diverse regions of the primate cerebral cortex. Science 232 (4747), 232–235.

Ramakers, G. J., 2005. Neuronal network formation in human cerebral cortex. Progress in Brain Research 147, 1–14.

Rao, R. P., Sejnowski, T. J., 2001. Spike-timing-dependent hebbian plasticity as temporal difference learning. Neural Computation 13, 2221–37.

Reyes, A. D., 2003. Synchrony-dependent propagation of firing rate in iteratively constructed networks in vitro. Nature Neuroscience 6 (6), 593–9.

Rice, F. L., Van Der Loos, H., 1977. Development of the barrels and barrel field in the somatosensory cortex of the mouse. The Journal of Comparative Neurology 171 (4), 545–60.

Roberts, P. D., Bell, C. C., 2002. Spike timing dependent synaptic plasticity in biological systems. Biol. Cybern. 87, 392–403.

Rubin, J. E., Terman, D., 2004. High frequency stimulation of the subthalamic nucleus eliminates pathological thalamic rhythmicity in a computational model. Journal of Computational Neuroscience 16, 211–235.

Rumsey, C. C., Abbott, L. F., 2004. Equalization of synaptic efficacy by activity- and timing-dependent synaptic plasticity. J. Neurophysiology 91, 2273–80.

Sakata, S., Komatsu, Y., Yamamori, T., March 2005. Local design principles of mammalian cortical networks. Neuroscience Research 51 (3), 309–15.

Schauer, C., Gross, H., 2004. Design and optimization of amari neural fields. IJCNN-2004.

Schemmel, J., Meier, K., Mueller, E., 2004. A new vlsi model of neural microcircuits. IJCNN-2004.

Scott, A., 2002. Neuroscience: A Mathematical Primer. Springer.

Segala, M., Korkotiana, E., Murphy, D. D., February 2000. Dendritic spine formation and pruning:next term common cellular mechanisms? Trends in Neurosciences 23 (2), 53–57.

Segundo, J. P., Stiber, M., Vibert, J., Hanneton, S., 1995a. Periodically modulated inhibition and its postsynaptic consequences-ii. influence of modulation slope, depth, range, noise and of postsynaptic natural discharges. Neuroscience 68 (3), 693–719.

Segundo, J. P., Vibert, J., Stiber, M., Hanneton, S., 1995b. Periodically modulated inhibition and its postsynaptic consequences-i. general features. influence of modulation frequency. Neuroscience 68 (3), 657–692.

Senn, W., 2002. Beyond spike timing: the role of nonlinear plasticity and unreliable synapses. Biol. Cybern. 87, 344–55.

Senn, W., Markram, H., Tsodyks, M., 2000. An algorithm for modifying neurotransmitter release probability based on pre- and postsynaptic spike timing. Neural Computation 13, 35–67.

Seth, A. K., Baars, B. J., March 2005. Neural darwinism and consciousness. Consciousness and Cognition 14 (1), 140–168.

Shen, Y., Gao, H., Yao, H., 2005. Spike timing-dependent synaptic plasticity in visual cortex: A modeling study. Journal of Computational Neuroscience 18, 25–39.

Shepherd, G. M., 1994. Neurobiology, 3rd Edition. Oxford University Press.

Singer, W., March 1993. Synchronization of cortical activity and its putative role in information processing and learning. Annual Review of Physiology 55, 349–374.

Sjostrom, P. J., Turrigiano, G. G., Nelson, S. B., December 2001. Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. neuron 32, 1149–64.

Sjostrom, P. J., Turrigiano, G. G., Nelson, S. B., 2003. Neocortical LTD via coincident activation of presynaptic NMDA and cannabinoid receptors. Neuron 39, 641–654.

Song, S., Abbott, L. F., 2001. Cortical development and remapping through spike timing-dependent plasticity. Neuron 32 (2), 339–350.

Song, S., Miller, K. D., Abbott, L. F., 2000. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. Nature Neuroscience 3, 919–26.

Sporns, O., 2002. Graph theory methods for the analysis of neural connectivity patterns. in Neuroscience Databases. A Practical Guide., 169–83.

Sporns, O., Chialvo, D. R., Kaiser, M., Hilgetag, C. C., September 2004. Organization, development and function of complex brain networks. Trends in Cognitive Sciences 8 (9), 418–25.

Stiber, M., 2005. Spike timing precision and neural error correction: Local behavior. Neural Computation 17, 1577–1601.

Stiber, M., Pottorf, M., 2004. Response space construction for neural error. IJCNN-2004.

Stoop, R., Bank, D. A., van der Vyver, J., Kern, A., 2001a. Synchronization-based computation, chaos and spike patterns in neocortical neural networks. Circuit Paradigm in the 21st Century, ECCTD'01 1, 221–4.

Stoop, R., van der Vyver, J., Kern, A., 2001b. Detection of noisy and pattern responses in complex systems. NDES IEEE Conference on Nonlinear Dynamics of Electronic Systems, 113–6.

Stuart, G. J., Hausser, M., 2001. Dendritic coincidence detection of epsps and action potentials. Nature Neuroscience 4 (1), 63–71.

Sur, M., Merzenich, M. M., Kaas, J. H., August 1980. Magnification, receptive-field area, and "hypercolumn" size in areas 3b and 1 of somatosensory cortex in owl monkeys. J Neurophysiol 44 (2), 295–311.

Suri, R. E., Sejnowski, T. J., 2002. Spike propagation synchronized by temporally asymmetric Hebbian learning. Biol. Cybern. 87, 440–5.

Takacs, J., Hamori, J., 1994. Developmental dynamics of purkinje cells and dendritic spines in rat cerebellar cortex. Journal of Neuroscience Research 38 (5), 515–530.

Tetko, I., Villa, A. E., 2001a. Pattern grouping algorithm and de-convolution filtering of non-stationary correlated poisson processes. Neurocomputing, 1709–1714.

Tetko, I. V., Villa, A. E., 1997a. A comparative study of pattern detection algorithm and dynamical system approach using simulated spike trains. Lecture notes in computer science 1327, 37–42.

Tetko, I. V., Villa, A. E., 1997b. Fast combinatorial methods to estimate the probability of complex temporal patterns of spikes. Biol. Cybern. 76, 397–407.

Tetko, I. V., Villa, A. E., 2001b. A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains. 1. detection of repeated patterns. Journal of Neuroscience Methods 105, 1–14.

Tetko, I. V., Villa, A. E., 2001c. A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains. 2. application to simultaneous single unit recordings. Journal of Neuroscience Methods 105, 15–24.

Tetzlaff, T., Geisel, T., Diesmann, M., 2002. The groundstate of cortical feed-forward networks. Neurocomputing 44-46, 673–8.

Tetzlaff, T., Morrison, A., Geisel, T., Diesmann, M., 2004. Consequences of realistic network size on the stability of embedded synfire chains. Neurocomputing 58-60, 117–21.

Tonnelier, A., 2005. Catgorization of neural excitability using threshold models. Neural Computation 17, 1447–55.

Torres, O., Eriksson, J., Moreno, J. M., Villa, A. E., August-October 2004. Hardware optimization and serial implementation of a novel spiking neuron model for the poetic tissue. BioSystems 76 (1), 201–208.

Troyer, T. W., Miller, K. D., 1997. Physiological gain leads to high isi variability in a simple model of a cortical regular spiking cell. Neural Computation 9, 971–983.

Turova, T. S., 2003. Long paths and cycles in dynamical graphs. Journal of Statistical Physics 110 (1), 385–417.

Turrigiano, G. G., Leslie, K. R., Desai, N. S., Rutherford, L. C., Nelson, S. B., February 1998. Activity-dependent scaling of quantal amplitude in neocortical neurons. Nature 391 (6670), 892–6.

Tyrrell, A. M., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., 2003. Poetic: An integrated architecture for bio-inspired hardware. Lecture Notes in Computer Science 2606, 129–140.

Uzzell, V. J., Chichilnisky, E. J., 2004. Precision of spike trains in primate retinal ganglion cells. J. Neurophysiol 92, 780–9.

van Ooyen, A., 2003. Modeling Neural Development, 1st Edition. Vol. 1. MIT Press.

Villa, A. E., 1992. Temporal aspects of information processing in the central nervous system. Annales CARNAC 5, 15–42.

Villa, A. E., 2000. Time and the Brain. Vol. 2. Harwood Academic Publishers.

Villa, A. E., Tetko, I., Hyland, B., Najem, A., 1999. Spatiotemporal activity patterns of rat cortical neurons predict responses in a conditioned task. Proc. Natl. Acad. Sci. USA 96, 1106–1111.

Villa, A. E., Tetko, I. V., Iglesias, J., June 2001. Computer assisted neurophysiological analysis of cell assemblies activity. Neurocomputing 38-40, 1025–1030.

Villa, A. E., Tetko, I. V., Iglesias, J., Filipov, D., 2000. Transdisciplinary approach to scientific data analysis through Internet. Haffmans Sachbuch Verlag, Zürich.

von Economo, C., 1929. The Cytoarchitectonics of the Human Cerebral Cortex. Oxford University Press, London.

Watts, D. J., 1999. Small Worlds: The Dynamics of Networks between Order and Randomness. Princeton University Press.

Wiener, M. C., Richmond, B. J., 2003. Decoding spike trains instant by instant using order statistics and the mixture-of-poissons model. The Journal of Neuroscience 23, 2394–2406.

Wigstrom, H., Gustafsson, B., 1986. Postsynaptic control of hippocampal long-term potentiation. Journal of Physiology 81 (4), 228–236.

Woodin, M. A., Ganguly, K., Poo, M., 2003. Coincident pre- and postsynaptic activity modifies GABAergic synapses by postsynaptic changes in Cl- transporter activity. Neuron 39, 807–820.

Yabuta, N. H., Callaway, E. M., 1998. Cytochrome-oxidase blobs and intrinsic horizontal connections of layer 2/3 pyramidal neurons in primate v1. Visual Neuroscience 15 (6), 1007–27.

Yazdanbakhsh, A., Babadi, B., Rouhani, S., Arabzadeh, E., Abbassian, A., 2002. New attractor states for synchronous activity in synfire chains with excitatory and inhibitory coupling. Biol. Cybern. 86, 367–78.

Yoshida, M., Hayashi, H., 2004. Organization of cell assemblies that code. IJCNN-2004.

Yoshioka, M., 2001. Spike-timing-dependent learning rule to encode spatiotemporal patterns in a network of spiking neurons. Physical Review E. 65, 011903.

Zecevic, N., Rakic, P., 1991. Synaptogenesis in monkey somatosensory cortex. Cerebral Cortex 1 (6), 510–523.

Zilles, K., 1990. in: The Human Nervous System, 757th Edition. Academic Press (G. Paxino, Ed.), San Diego.