

Détection de patterns d'activité bioélectrique simulée et modélisation de réseaux neuraux bioinspirés par l'expression génique

Vladyslav Shaposhnyk

▶ **To cite this version:**

Vladyslav Shaposhnyk. Détection de patterns d'activité bioélectrique simulée et modélisation de réseaux neuraux bioinspirés par l'expression génique. Médecine humaine et pathologie. Université de Grenoble; Université catholique d'Ukraine, 2011. Français. NNT : 2011GRENS017 . tel-00685211

HAL Id: tel-00685211

https://theses.hal.science/tel-00685211

Submitted on 4 Apr 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Modèles, méthodes et algorithmes pour la Biologie, la Santé et l'Environnement**

Arrêté ministériel : 7 août 2006

Présentée par

Vladyslav SHAPOSHNYK

Thèse dirigée par **M. Alessandro E.P. VILLA** et
codirigée par **Mme. Tetiana AKSENOVA**

préparée au sein du **Institut des Neurosciences de Grenoble, l'Equipe 7** dans **l'École Doctorale « Ingénierie pour la Santé , la Cognition et l'Environnement »**

Détection de patterns d'activité bioélectrique simulée et modélisation de réseaux neuraux bioinspirés par l'expression génique

Thèse soutenue publiquement le **« »** _____,
devant le jury composé de :

Prof. François BERGER
Président
Prof. Gilles SASSATELLI
Rapporteur
Prof. Roman BORISYUK
Rapporteur
Prof. Nataliia KUSSUL
Membre
Dr. Tetiana AKSENOVA
Membre
Prof. Alessandro E.P. VILLA
Membre

UNIVERSITE JOSEPH FOURIER GRENOBLE 1
ECOLE DOCTORALE INGENIERIE POUR LA SANTE, LA COGNITION ET L'ENVIRONNEMENT (EDISCE)

et / and

NATIONAL TECHNICAL UNIVERSITY OF UKRAINE "KYIVSKIY POLITECHNIY INSTITUT"
INSTITUTE OF APPLIED SYSTEM ANALYSIS

Thèse de doctorat
Doctoral Thesis

présenté par
presented by

Vladyslav Shaposhnyk

DETECTION DE PATTERNS D'ACTIVITE BIOELECTRIQUE SIMULEE ET MODELISATION DE RESEAUX NEURAUX BIOINSPIRES PAR L'EXPRESSION GENIQUE

DETECTION OF PATTERNS OF SIMULATED BIOELECTRIC ACTIVITY AND MODELING OF BIOINSPIRED NEURAL NETWORKS WITH GENETIC EXPRESSION

Thèse dirigée par
directed by

Prof. Alessandro E.P. Villa et Dr. Tetiana Aksenova

Jury :

Prof. François Berger, Président
Prof. Gilles Sassatelli, Rapporteur
Prof. Roman Borisyuk, Rapporteur
Prof. Nataliia Kussul, Membre
Dr. Tetiana Aksenova, Membre
Prof. Alessandro E.P. Villa, Membre

Soutenue le …….. à ……..

GRENOBLE, FRANCE ET KIEV, UKRAINE

Members of the Jury

Supervisors

Dr. **Tetiana Aksenova**
National Technical University of Ukraine
Institute of Applied System Analysis
Department of Nonlinear System Analysis
03056 Kiev, Ukraine
tanya_ aksfr @ yahoo.com

Prof. **Alessandro E.P. Villa**
University Joseph Fourier Grenoble 1
Institute of Neuroscience of Grenoble
Equipe 7 - Nanomedicine and Brains
38042 Grenoble, France
alessandro.villa @ ujf-grenoble.fr

Internal Experts

Prof. **François Berger**
University Joseph Fourier Grenoble 1
Institute of Neuroscience of Grenoble
Equipe 7 - Nanomedicine and Brains
38042 Grenoble, France
fberger @ ujf-grenoble.fr

External Experts

Prof. **Gilles Sassatelli**
University of Montpellier 2
Laboratory of Informatics, Robots
and Microelectronics, CNRS
34392 Montpellier, France
sassatelli @ lirmm.fr

Prof. **Nataliia Kussul**
Space Research Institute of National
Academy of Sciences and National
Space Agency of Ukraine
03680 Kiev, Ukraine
inform @ ikd.kiev.ua

Prof. **Roman Borisyuk**
University of Plymouth
Centre for Neural and Adaptive Systems,
Plymouth, PL4 8AA, United Kingdom
r.borisyuk @ plymouth.ac.uk

Abstract

Modular architecture is a hallmark of many brain circuits. Particularly, in the cerebral cortex it has been observed that reciprocal connections are often present between functionally interconnected areas that are hierarchically organized. Evolutionary development is another distinctive characteristic of living species, even the simplest viruses are capable to adapt to better fit new environmental conditions.

Having hierarchical architectures and evolutionary features in mind, we build unique and novel simulation framework, which allows us to model and to study evolving hierarchically organized circuits of modules of spiking neural networks. Each module is characterized by embedded neural development and expression of spike timing dependent plasticity. Cell death, synaptic plasticity and projection pruning, embedded in the neural model, drive the build-up of auto-associative links within each module, which generate an areal activity that reflect the changes in the corresponding functional connectivity within and between neuronal modules. Bioelectric activity of each module is recorded by means of virtual electrodes and these signals, called electrochipograms (EChG), are analyzed by time and frequency domain methods in order to find general patterns of emerging behavior. Beside time and frequency domain analysis methods, a novel robust non-linear structural regression approach is proposed to provide researchers with more powerful tools specially adapted to the data typically used in the domain. We tested the effect of an external stimulus at fixed frequency fed to a sensory module, which projecting its activity to two hierarchically organized parallel pathways. We found that modeled circuits manifest behavior similar in certain aspects to that of real brains. We show evidence that all networks of modules are able to maintain long patterns of activity associated with the stimulus offset.

These findings bring new insights to the understanding of EEG-like signals, both real and virtual. The findings prove that the approach is successful and could be extended to model cognitive and behavioral processes in the brains.

Keywords: hierarchical neural networks, evolutionary networks, electroencephalograms (EEG), group method of data handling (GMDH), robust regression.

Synopsis

L'architecture modulaire est une caractéristique distinctive des circuits cérébraux. En particulier, il a été observé l'existence de connexions réciproques entre des zones fonctionnellement interconnectées dans le cortex, et qui par ailleurs sont hiérarchiquement organisées. De plus, le développement évolutif est une autre caractéristique distinctive des espèces vivantes ; même les virus sont capables d'adaptation pour mieux répondre à de nouvelles conditions environnementales.

En tenant compte de ces deux importants aspects, nous avons construit un nouvel et unique outil de simulation permettant de modéliser et d'étudier l'évolution des circuits multi-modulaires hiérarchiques. Dans ce modèle, chaque module est représenté par des réseaux de neurones impulsionels et caractérisé à la fois par des changements d'activités neurales imbriquées et par la plasticité synaptique. La morte cellulaire, la plasticité synaptique et l'apoptose intégrés dans le modèle créent des liens auto-associatifs au sein des modules. Ces liens peuvent générer une activité zonale qui reflète l'évolution de la connectivité fonctionnelle à l'intérieur comme à l'extérieur des modules, et donc entre les plusieurs modules neuronaux. L'activité bioélectrique de chaque module est enregistrée au moyen des électrodes virtuelles. Les signaux, electrochipogrammes (EChG), sont analysés par les méthodes fréquentiels et les méthodes de potentiels évoqués afin de trouver des généralités dans le comportement émergeant. En plus de ces méthodes conventionnelles, nous proposons une nouvelle approche de régression non-linéaire structurelle afin de fournir des outils plus puissants et mieux adaptés aux données habituellement analysées dans ce domaine. Nous avons donc testé l'effet d'un stimulus externe sur le développement de liens fonctionnels d'un réseau neuronaux. Le circuit est structuré hiérarchiquement avec un unique module sensoriel et d'autres modules constitués de deux voies parallèles organisées aussi de façon hiérarchique. Nos résultats montrent que les circuits modélisés manifestent un comportement similaire que les circuits biologiques réels. En particulier, tous les éléments du circuit peuvent traiter et maintenir des patterns d'activité liés à la disparition du stimulus.

Les résultats obtenus dans nos expériences apportent un éclairage sur les processus émergents et coordonnés de l'activité électrique enregistrée par des EEG de circuits inter-corticaux hiérarchiques et évolutifs qui sont artificiels ou réels. Plus généralement, notre approche concernant les signaux EEG pourrait être étendue à la modélisation d'une vaste variété des processus cognitifs et comportementaux.

Mots-clés : reseaux hierarchiques de neurones, reseaux neuronaux evolutionaire, encephalogrammes (EEG), group method of data handling (GMDH), regression robuste.

Анотація

Модульна архітектура – одна з характерних властивостей організації мозку. Давно відомо, що в корі головного мозку ділянки, відповідальні за обробку певних видів інформації, з'єднані між собою в ієрархічно організовану мережу. Інша характерна риса живих істот – притаманна їм еволюція, спричинена кращою адаптацією до умов середовища кожного наступного покоління.

Орієнтуючись на ці дві відмінні властивості, ми побудували унікальне пристосування для симуляції нейронних мереж, яке дозволяє моделювати та вивчати еволюційні ієрархічно організовані нейронні кола з модулів імпульсних нейронних мереж. Для нейро-модулів цих мереж характерні функції розвитку нейронів та синаптичної пластичності залежної від часу надходження вхідних імпульсів. Видалення неефективних нейронів та синапсів разом з синаптичною пластичністю призводять до побудови в кожному модулі ауто-асоціативних зв'язків між групами нейронів. Зональна активність, що генерується цими групами, відповідає встановленим функціональним зв'язкам як між групами нейронів різних модулів, так і між різними групами всередині модуля. Біоелектрична активність кожного нейро-модуля реєструється за допомогою віртуальних електродів. Ці сигнали, які ми назвали електрочіпограми (EChG), аналізувалися за допомогою частотних методів та методів ключових-подій для виявлення загальних закономірностей, притаманних процесам обробки інформації в нейронних мережах. Ми запропонували новаторський метод нелінійної робастної структурної та параметричної регресії, спеціально адаптований до характерних для нейрофізіології даних. Ми дослідили ефекти подачі стимулу фіксованої частоти на двошарові ієрархічні нейронні мережі з виділеним сенсорним нейро-модулем. Ці дослідження показують, що наші моделі нейронних кіл здатні демонструвати певні властивості, що притаманні нейро-системам живих істот. При цьому, всі мережі здатні обробляти вхідну інформацію, зберігаючи характерну активність, асоційовану з кінцем подачі стимулу.

Наші дослідженні проливають світло на природу формування EEG-подібних сигналів в мозку та в модельованих великих ієрархічних еволюцінуючих нейро-системах. Виконані нами експерименти доводять можлівість застосування розробленого симулятору для подальшого моделювання складних когнітивних процесів.

Ключові слова: ієрархічні нейронні мережі, еволюційні нейронні мережі, енцефалографія (EEG), метод групового урахвання аргументів (МГУА), регресія стійка до викидів.

List of publications

Vladyslav Shaposhnyk, Pierre Dutoit, Victor Contreras-Lámus, Stephen Perrig, and Alessandro E.P. Villa.

"A framework for simulation and analysis of dynamically organized distributed neural networks" in *Artificial Neural Networks – ICANN 2009*, volume 5768 of *Lecture Notes in Computer Science*, pages 277–286. Springer Berlin / Heidelberg, 2009.

Vladyslav Shaposhnyk, Alessandro E.P. Villa, and Tetyana Aksenova.

"Advances in structural modeling robust to outliers in explanatory and response variables" *Proceedings of the 2010 IEEE World Congress on Computational Intelligence*, pages 628–635, IEEE, 2010.

Vladyslav Shaposhnyk, Pierre Dutoit, Stephen Perrig, and Alessandro Villa.

"Functional connectivity driven by external stimuli in a network of hierarchically organized neural modules" in *Artificial Neural Networks – ICANN 2010*, volume 6352 of *Lecture Notes in Computer Science*, pages 135–144. Springer Berlin / Heidelberg, 2010.

Olga Chibirova, Javier Iglesias, Vladyslav Shaposhnyk, and Alessandro Villa.

"Dynamics of firing patterns in evolvable hierarchically organized neural networks" in *Evolvable Systems: From Biology to Hardware*, volume 5216 of *Lecture Notes in Computer Science*, pages 296–307. Springer Berlin / Heidelberg, 2008.

Stephen Perrig, Pierre Dutoit, Katerina Espa-Cervena, Vladislav Shaposhnyk, Laurent Pelletier, François Berger, and Alessandro E. P. Villa.

"Changes in quadratic phase coupling of EEG signals during wake and sleep in two chronic insomnia patients, before and after cognitive behavioral therapy" in *Proceedings of the 2009 conference on Neural Nets WIRN09*, pages 217–228, Amsterdam, The Netherlands, IOS Press, 2009.

List of presentations

Neural Networks in Classification, Regression and Data-mining
Porto, Portugal, 2008

The 9th International Conference on Evolvable Systems: From Biology to Hardware
Prague, Check Republic, 2008

The middle-term PERPLEXUS project progress presentation
Brussels, Belgium, 2008

The 19th International Conference on Artificial Neural Networks
Limassol, Cyprus, 2009

The final PERPLEXUS project results presentation
Lausanne, Switzerland, 2010

International Joint Conference on Neural Networks 2010
Barcelona, Spain, 2010

The 20th International Conference on Artificial Neural Networks
Thessaloniki, Greece, 2010

to my Family
for their love and support

Preface

It is hard to find a black cat
in a dark room, especially,
if it is not there

S. Belloc

This history started during one pleasant evening filled with shiny red light of sunset somewhere at the end of august, when hot summer weather already had gone, but cold and rainy autumn weather had not arrived jet. Two month ago, I have finished my master's studies and received an appropriate diploma, even with honors. I have talked about possible doctorate studies with my colleagues and, as well, as with several graduate students of previous years and after all I had built up a picture which was representing the subject, thought it was not in the shiny colors at least here in *alma mater*, so I decided to put my efforts into commercial area. That evening, I was on the work, it was quite late, so most co-workers are already went home. I could calmly concentrate on an urgent stuff I had to do and benefit of quiet office without treat of being disturbed by other urgent things or by colleagues.

My phone rang, and to my great surprise it was not a customer with an urgent need, nor a big boss with just another urgent matter, but Alexander Michaylovich Reznik, my Master Thesis' director. The talk was rather short, in a straightforward and clear manner he told me that it happens such that there is an open doctorate position in France, he told me that work will be based on neural networks, just like the master thesis was, but he did not have more information. He asked, if I want to try opportunity and if I mind if he would pass my contacts to people interested to fill the position. Obviously, I had answered that it sounds really exiting, thought it was totally unexpected. Sure, I did not mind take a contact and to discuss a bit those matters. I had talks on the subject with my friends and my family and was pretty sure that I will take the opportunity to follow academic career.

Few days later, it was Monday, the 4^{th} of September 2006, I had a call from Tatyana Ivanovna Aksenova, my future supervisor of the thesis. She described briefly on what was she working in Grenoble, we have discussed several more points about the position and the expected outcome of the study, I was quite surprised to hear about studies with obvious medico-biological focus, but nevertheless, driven by curiosity, I signed to start the doctoral study in a frame of joint thesis supervision agreement between Kiev and Grenoble. That was a start of a long way to this great day of final defense of the Thesis. In that moment it was

left only 4 days before deadline of the submission of the documents for a doctoral selection concurs in Kyïv and it was almost 5 years to the day of the defense.

Acknowledgements

First of all I want to thank Tatyana Aksenova and Alessandro Villa, without whom this Thesis would ever be started. Then I would like to thank my parents for their faith in me and for the support I had always have from their side.

I would like to thank all my friends in the France, in the Switzerland and in the Ukraine for keeping me motivated in this work and pushing me forward. In particular, I thank Jérémie and Bertrand, for widening my views for their domains and, especially, for teaching me a proper French. Same credits go to Pascal and Pierre, with whom I met during the later phases of the Thesis's venue. I thank Dima especially for for keeping me focused on academic carrier, scientific life-style and life-values, also I thank him also for a huge work done on commenting chapters' organization and on grammatical check of the Thesis. I thank to Pascal, who helped me a lot with understanding of our results from neurophysiological point of view, for giving me basic concepts of real electroencephalography (EEG) acquisition and data analysis and also for helping me with french parts of the Thesis. I thank Christian and Enea especially for their help, support and first glitches on that what is called an assistantship. I thank Sanya and Venya for still keeping in touch even despite all geographical distance between us. I thank Fabio, Pierre, Julien, Andriï and all other people I have been connected with in my university life. Special thanks go to Olya and Kyrill, who gave me an indispensable "coup de main" in my first steps of live in Grenoble and with whom I had a lot of pleasant hours in our journeys all around Grenoble. I thank all the members of Equipe 7 of GIN, especially Sandra and Laurent, Jean-Paul and Isabelle, and Chantale for warm receipt and for support from the very first my days in Grenoble. I thank all the members of PERPLEXUS project, and especially Pierre, Olivier and Thierry for the invaluable help in sorting things in distributed model development and data analysis. I thank all the people were around me during more these four years of work. That is with your help it is finished now.

Lausanne, the 3^{rd} of December 2010

Résumé (en français)

L'architecture modulaire est une caractéristique distinctive des circuits cérébraux. En particulier, il a été observé l'existence de connexions réciproques entre des zones fonctionnellement interconnectées dans le cortex, et qui par ailleurs sont hiérarchiquement organisées. De plus, le développement évolutif est une caractéristique distinctive des espèces vivantes ; même les virus sont capables d'adaptation pour mieux répondre à de nouvelles conditions environnementales.

En tenant compte ces deux importants aspects, nous avons construit un nouvel et unique outil de simulation permettant de modéliser et d'étudier l'évolution des circuits multi-modulaires hiérarchiques de neurones à « décharges » qui comportent des particularités ontogénétiques et épigénétiques de développement (voir Chapitre 3).

Nous avons donc testé l'effet d'un stimulus externe sur le développement de liens fonctionnels de réseau neuronaux. Deux conditions expérimentales ont été analysées. Dans la première, nous avons étudié le changement d'activité émergeant par rapport aux phases développementales. Dans la seconde, nous nous sommes focalisé sur l'évolution de la connectivité fonctionnelle à l'intérieur comme à l'extérieur des modules ; donc entre les plusieurs modules neuronaux selon les types des projections inter-modulaires présents dans la topologie du circuit. Cette synchronisation des oscillations neurales est associée avec processus du développement des liens entre les régions du cerveau qui a lieu pendant le traitement d'information [127, 148]. Les études de simulation de l'électroencéphalographie (EEG), réalisées dans les expériences avec les modèles oscillatoires de populations de cellules [89, 48], ont souligné comment la modulation de la puissance des décharges synaptiques affecte l'évolution du signal dans le temps et la forme de l'onde évoquée (i.e. ERPs).

Le circuit modélisé est structuré hiérarchiquement avec un unique module sensoriel et d'autres modules constitués de deux voies parallèles organisées également de façon hiérarchique. La plasticité synaptique, la mort cellulaire et l'apoptose sont intégrées au modèle et créent des liens auto-associatifs au sein des modules. Ces liens peuvent générer une activité zonale qui reflète l'évolution de la connectivité fonctionnelle à l'intérieur comme à l'extérieur des modules, et donc entre les plusieurs modules neuronaux. L'activité bioélectrique de chaque module est enregistrée au moyen des électrodes virtuelles. Les signaux, electrochipogrammes (EChG), sont analysés par les méthodes fréquentielles et les méthodes de potentiels évoqués afin de trouver des généralités dans le comportement émergeant.

Par ailleurs, les outils génétiques du cadre de simulation ont été utilisés pour créer des générations de circuits neuronaux. En particulier, les paramètres du modèle des réseaux neuronaux sont codés dans le génome des circuits. Le modèle de simulation intègre la mutation du génome, ce nous a permis de récupérer les

propriétés générales des circuits neuronaux.

Réseaux de neurones à décharges

Dans ce modèle, le circuit neuronal est constitué de plusieurs modules dans lequel chacun entre eux est un réseau d'unités neuronales distribuées sur une grille bidimensionnelle. Les unités neuronales peuvent être de deux types : excitatrices et inhibitrices. Ces unités sont distribuées au hasard sur la surface de la grille en proportion 80 : 20 (respectivement, excitatrices et inhibitrices). Ces deux types de neurones suivent la même dynamique « integrate-and-fire ». La plasticité synaptique se trouve modulée par l'activité de ces neurones. La plasticité synaptique à modulation temporelle relative (Spike-Timing Dependent Plasticity, STDP) est un changement de la force des synapses basé sur l'ordre des décharges pré- et post-synaptique.

Les processus dynamiques (plasticité synaptique, mort cellulaire et apoptose) intégrés dans le modèle provoquent l'apprentissage compétitif grâce à l'élimination des synapses les moins efficaces. De même, ces processus créent des liens auto-associatifs au sein des modules qui ouvrent les relations fonctionnelles entre des groupes de neurones. Pour éviter la morte cellulaire excessive, le modèle intègre l'activité de fond. Ces sont des décharges neuronales non corrélées générées selon la distribution de Poisson à un taux moyen de 300 décharges/s et une amplitude de 1,9 mV (ou 900 décharges/s et 1,0 mV pour les petits réseaux modélisés ; voir l'Article A).

Deux ensembles de cellules ont été choisis au hasard parmi les neurones excitateurs pour chaque module neuronal. Ces ensembles correspondent aux couches efférentes et afférentes. En plus de l'activité interne ces neurones sont connectés aux autres modules. Pendant les stades précoces de développement, les connexions inter-modulaires sont établies de façon autoréflexive. C'est-à-dire que le nombre de projections extérieures d'un neurone est proportionnel au nombre de projections internes du même neurone. Selon la configuration expérimentale chacune de ces couches pourraient contenir de 10% ou 20% de toutes les cellules excitatrices du module. La présence de cellules efférentes et afférentes dans les modules nous guide logiquement vers des ensembles hiérarques des modules, i.e. les circuits neuronaux.

Réseaux hiérarchiques

Quatre topologies de circuit ont était utilisées dans nos études (voir la figure 7.5). Chacune était créée avec une combinaison différente de liens réciproques inter-modulaires. Dans tous les cas, les topologies étaient composées de 6 modules neuronales pouvant jouer 3 rôles distincts : sensoriel, traitement et moteur. Le module sensoriel neuronal était le seul module recevant un stimulus artificiel de l'extérieur. En plus, ce module présentait une activité de fond plus puissante que les autres modules. Cette activité de fond reflétait les signaux plus d'activité spontanée (« bruit ») du cortex périphérique.

Les quatre modules traitement ont des connexions avec le module sensoriel, et des connexions entre-eux réciproques. Ces modules ont un rôle de traitement de l'information. La première paire d'entre eux recevait des inputs directement à

partir du module sensoriel, formant la première couche de traitement du circuit. Tandis que la deuxième paire était reliée avec les autres modules de traitement uniquement, formant ainsi la deuxième couche de traitement. Les principales caractéristiques des topologiques du circuit était la présence de projections réciproques à l'intérieur d'une couche de traitement ou entre les deux couches de traitement.

La combinaison de ces caractéristiques donne quatre types de circuits possibles :

- une topologie exclusivement « feed-backward » (FB), où il y a projections réciproques entre les couches de traitement, mais sans projections intra-couche ;
- une topologie exclusivement « feed-forward » (FF) qui était similaire à la première, mais sans projections réciproques entre les couches ;
- une topologie « feed-backward » avec projections intra-couche (appelée ci-dessous « liens horizontaux ») et abrégé en FBH,
- et une topologie « feed-forward » avec des liens horizontaux (FFH) qui n'ont pas projections réciproques entre couches, mais avec projections réciproques intra-couche.

Dans les articles A « Dynamically organized neural networks » et B « Stimuli-driven functional connectivity », les expériences incluent la topologie FBH uniquement. Nous nous sommes focalisés sur les changements du comportement émergent causés par la présentation d'une stimulation externe. Tandis que dans la dernière expérience (voir Chapitre 7) nous nous sommes focalisés sur l'effet des projections réciproques sur la connectivité fonctionnelle entre modules.

La stimulation externe était appliquée dans les cellules afférentes du module sensoriel au moyen d'un stimulus spatio-temporel. Chaque stimulation était suivie par une période de silence (inter-stimulus intervalle, ISI) de 1000 ms. Ensemble, ils formaient un essai d'apprentissage du réseau. Plusieurs répétitions de stimulation forçaient l'apprentissage non supervisé du circuit grâce aux processus de mort cellulaire et plasticité synaptique. Le motif de stimulus dans chaque essai était légèrement modifié par rapport au motif initial par l'introduction d'une variabilité de 10% ; i.e. une variation de 1 ms du temps d'activation (10% des cellules afférentes sélectionnées au hasard).

Signal bioélectrique

L'activité bioélectrique de chaque module était enregistrée au moyen des électrodes virtuelles. Ces signaux, appelé electrochipogrammes (EChG), se caractérisent par des propriétés proches de l'encéphalographie (EEG), l'electrocorticographie (ECoG) et des potentiels des champs locaux (LFP). L'électrode virtuelle comporte deux paramètres importants : sa zone de couverture et sa fonction de sensibilité. La zone de sensibilité de l'électrode était limitée par un « cercle ». La sensibilité de l'électrode virtuelle diminuait linéairement du centre (100%) au bord (0%) de l'électrode. Chaque module neuronal était équipé d'au moins une électrode virtuelle pour recueillir des signaux. Ces signaux ont été analysés par des méthodes fréquentielles et les méthodes de potentiels évoqués afin de trouver les caractéristiques générales dans les patterns émergents (voir Chapitre 6).

Traits génétiques et évolution

La partie évolutive du simulateur nous permet de créer un ensemble des circuits neuronaux lesquels ont été analysés pour déterminer les comportements communs à l'ensemble. Quatre génomes différents correspondant à chacune des quatre topologies décrites ci-dessus ont été utilisés comme ceux de base et chacun d'eux a été modifié plusieurs fois par mutation aléatoire d'un gène. La technique « Multiple flip-bit » [92] a été appliquée au gène du circuit responsable de la connectivité synaptique dans les modules. Une faible variation de ce gène était même suffisante pour produire une carte de connexion interne du module radicalement différente, mais cette carte préserve toujours la distribution des connexions. Les espèces des futures générations de circuits ont été réalisés selon le résultat d'une fonction d'évaluation du circuit actuel. Donc production d'un circuit « fils » était réalisé au moment de la mort du circuit « père ». Ceci permettait d'obtenir des espèces différentes pour chaque type de topologie et de moyenner les signaux EChG afin de lisser le bruit d'activité spontanée des réseaux.

Analyses bispectrales

Les electrochipogrammes sont analysés par les méthodes fréquentiellss et les méthodes de potentiels évoqués afin de trouver des généralités dans le comportement émergeant des réseaux. Le paradigme expérimental présenté dans l'Article « Stimuli-driven functional connectivity » décrivait les étapes précédentes, postérieures, au commencement et à la fin de présentation d'un stimulus spatio-temporel répété plusieurs fois (étapes nommées PRE-lerning, EARLY-learning, LATE-learning et POST-learning). Les résultats de la modélisation sont discutés à la lumière d'une expérience appliquée à une étude chez l'homme dans un protocole expérimental et de méthodes de traitement similaires au signal traité ici.

L'étape de « PRE-learning » pourrait représenter une situation de contrôle conduite exclusivement par l'activité de fond du cerveau du sujet. Le sujet était naïf, de sorte qu'un processus d'apprentissage peut se produire. Au cours de la phase « EARLY-learning », une répétition des stimuli à intervalles réguliers était réalisée. Elle pouvait ainsi engager un processus de reconnaissance non supervisée qui forme les ensembles des cellules mis en évidence par la connectivité fonctionnelle inter-modulaire. L'étape de « LATE-learning » était caractérisée par la maturité du réseau dans le quel la plasticité synaptique et la mort cellulaire sont stabilisées. Le niveau d'activité spontanée était le plus bas comparé aux autres étapes. Ceci offre les meilleurs conditions d'analyse du signal. Finalement, le « POST-learning » est conduit par échos internes du stimulus et montre comment le réseau réagit à la disparition du stimulus.

Nous avons donc testé l'effet d'un stimulus externe sur le développement d'un réseau de neurones. L'analyse spectrale à hauts dégrées de signaux EEG et EChG nous permit de déterminer la gamme de fréquences de couplage de phase (fréquences de résonance) entre les zone corticales et entre les modules neuronaux [140, 141]. Selon l'interprétation habituelle basée sur la théorie des ondes stationnaires, les hautes fréquences de résonance signifient que le traitement des informations est transmit à courte distance. Ceci signifie que la distance entre une paire

de zones corticales qui traitent le signal est courte alors qu'un couplage qui se produit à des fréquences basses peut être interprétée comme un signe d'interaction inter-corticale.

Un résultat remarquable est la constatation que dans les modélisations de circuits, l'étape de « LATE-learning » est caractérisée par l'index de fréquences résonantes (IRF) de 14 par rapport à l'IRF entre 43 et 62 pour les étapes de « PRE- et de EARLY-learning » (voir les tableaux B.1 et B.2). Chez l'homme, nous avons observé que les contrôles et les patients insomniaques après traitement sont eux-aussi caractérisés par des valeurs d'IRF plus basses avant traitement durant toutes les phases du sommeil.

Il faut aussi signaler que la seule condition sommeil REM laisse apparaître une différence de fréquences de résonance dans l'intervalle 14-33 Hz au cours du sommeil lent quel que soit le traitement. Ce dernier résultat suggère que malgré un changement global de fréquences de résonance les interactions corticales ont tendance à persister chez les patients lors des périodes de sommeil lent. À la fois une stimulation appropriée des circuits modélisés et la thérapie cognitive semblent modifier le rapport des fréquences de résonance et provoquent ainsi un déplacement de l'index vers les basses fréquences à tous les états du cerveau.

Analyses fonctionnelles

Le Chapitre 7 décrit une des premières études concernant la simulation de signaux EEG générés par un grand échantillon de réseau de neurones hiérarchiques de type « intgrate-and-fire ». La plupart des études importantes du domaine s'appuient sur la dynamique des populations cellulaires et des masses de neurones [56, 65, 39, 11, 59]. Quant aux études impliquant l'analyse de signaux EEG, elles étaient généralement destinées à déterminer la stabilité de l'état dynamique du réseau, l'effet du bruit sur le réseau, ou l'émergence d'une activité synchrone. Notre approche expérimentale, quant à elle, offre la possibilité d'étudier l'effet important que la connectivité inter-corticale (inter-modulaire) pourrait avoir sur l'activité de circuits neuronaux ainsi que sur le développement de liens fonctionnels.

Nous avons donc testé l'effet de projections réciproques sur développement des liens fonctionnels dans un circuit neuronal lors d'une stimulation externe artificielle. Le circuit est structuré hiérarchiquement avec un unique module sensoriel tandis que les autres modules sont constitués de deux voies parallèles organisées et hiérarchiques.

Nos résultats montrent que les circuits modélisés manifestent un comportement similaire aux circuits biologiques réels. En particulier, tous les éléments du circuit maintiennent des patterns d'activité caractéristiques liés à la disparition du stimulus. En accord avec l'étude précédente focalisée sur simulation d'un seul module neuronal, ce résultat montre que des patterns caractéristiques de décharges cellulaires n'étaient pas déclenchées seulement par la seule présentation du stimulus, mais aussi en raison de la présence de motifs spatio-temporels intégrés dans le stimulus. Les associations transitoires entre les réseaux neuronaux étaient évoquées par le stimulus, mais pas nécessairement calées à son apparition. Ceux-ci pouvaient être aussi associés avec un phénomène de propagation de potentiels de cohérence enregistrés dans les cultures organotypiques du cortex [136].

Les résultats d'analyse temps-fréquentielles montrent que la disparition du stimulus est un événement très important parce qu'il déclenche une activité cohérente dans les basses fréquences sur l'ensemble des circuits neuronaux étudiés ici. Ce résultat s'accorde avec d'autres obtenus chez l'homme montrant des patterns spécifiques dans les fréquences basses associés au type de traitement d'information (simultané ou successive) uniquement, et indépendant des autres conditions expérimentales [108]. Il est important de noter que les densités spectrales de puissance (PSD) des enregistrements EChG montrent plus d'énergie dans la bande gamma pour les modules de la 2-ème couche de traitement du circuit FBH que dans tous les autres circuits (la même couche). Le circuit FBH, qui se caractérise par des projections réciproques et horizontales, est également caractérisé par une cohérence élevée entre les couches s'étendant au cours de l'ISI dans la bande de fréquence gamma.

Egalement, il a été modélisé un réseau caractérisé par de multiples modules partiellement synchronisés et fortement excité par un stimulus. Ce réseau comportait un large éventail de comportements flexibles, adaptables et complexes. Il a été modélisé comme la variance de l'augmentation de gain de connexion, les connexions inhibitrices deviennant plus susceptibles de synchronisation et mondial est représenté à diminuer [61]. L'effet de l'introduction de liens entre les modules de la même couche (« horizontaux ») évoque également de plus puissantes excitations associées au début du stimulus et une inhibition associée à la fin de sa présentation (voir la figure 7.7) dans les modules neuronaux de la 1-ère couche de traitement. Ces effets sont indépendamment des projections réciproques de type « feed-back ». Dans la deuxième couche, l'effet sur les ERPs était plus subtil. Cet effet a été plus facilement mis en évidence dans l'étude des cohérences entre les couches. Les durées transitoires évoquées était sensibles à l'augmentation en fonction de la profondeur hiérarchique de traitement [40].

Toutefois, dans les topologies FF et FB, nous avons trouvé des composantes synchronisées apparaissant plusieurs centaines millisecondes après la fin du stimulus. Ceci soulève la possibilité d'autres hypothèses que la simple dépendance des connexions réciproques de « feed-back » afin de refléter une rentrée des patterns dynamiques de comportement dans les niveaux inférieurs de la hiérarchie. L'écart entre nos résultats et les résultats expérimentaux [40] peut être expliqué par le modèle neuronal choisi. Les grandes populations neuronales qui se trouvent dans le cerveau d'où émergent les patterns de décharges sont beaucoup plus complexes que ceux qui peuvent être modélisés.

Optimisation structurelle et paramétrique de données multivariées contaminées par des artéfacts

En plus des méthodes conventionnelles d'analyse mentionnées ci-dessus (i.e. analyses de potentiels évoqués ainsi que méthodes fréquentielles), nous proposons une nouvelle approche de régression non-linéaire structurelle afin de fournir des outils plus puissants et mieux adaptés aux données habituellement analysées dans ce domaine.

Plus précisément, le Chapitre 8 décrit en premier lieu les avancées algorithmiques concernant l'optimisation structurelle non-linaire robuste. Ces méthodes

d'optimisation s'appuient sur des modèles polynômiaux qui concernent des données multivariées. Dans notre travail, les méthodes d'optimisation considérées sont toujours basées sur l'approche itérative du « Group Method of Data Handling », ou plus particulièrement sur l'algorithme « Polynomial Neural Network » (PNN). Toutes les versions des algorithmes PNN permettent une modélisation universelle de la structure des données, et ce grâce à la synthèse adaptative et évolutive des modèles effectuée par le biais d'un réseau de neurones artificiel. Ces méthodes d'optimisation permettent alors de déduire la structure d'un modèle non-linaire à partir des données possiblement polluées, ainsi que d'optimiser les paramètres du modèle en question.

Dans l'étude précédente [5], l'algorithme PNN considéré traitait des données dont les artéfacts ne se manifestaient que dans les variables de réponse (Y). La présente contribution, quant à elle, propose une approche plus générale où les artéfacts peuvent se manifester aussi bien dans les variables de réponse (Y) que dans les variables d'explication (X). Nous avons amélioré l'algorithme PNN par un système d'estimation généralisée du maximum de vraisemblance (GM-estimation) qui permet d'évaluer la probabilité qu'une donnée soit polluée par des artéfacts provenant d'erreurs de mesure ou de complications expérimentales. Cette méthode d'estimation possède l'avantage d'être robuste aux artefacts qui se manifestent à la fois dans les variables de réponse et d'explication. La nouvelle version de l'algorithme PNN est désignée par « Enhanced Robust Polynomial Neural Network » (ERPNN).

L'implémentation de l'algorithme a été testée sur des données artificielles créées à partir de polynômes aléatoires du premier, deuxième ou troisième degré, puis contaminées par l'introduction d'artéfacts ainsi que de bruit blanc. Nos tests montrent que la nouvelle version s'avère plus précise que la précédente. En outre, elle permet de reconstruire automatiquement la forme explicite des modelés non-linéaires sous-jacents, ainsi que d'estimer leurs paramètres, et ce malgré la présence d'artéfacts importants.

Conclusions

Les résultats obtenus dans nos expériences apportent un éclairage sur les processus émergents et coordonnés de l'activité électrique enregistrée par des EEG de circuits inter-corticaux hiérarchiques et évolutifs qui sont artificiels ou réels. Plus généralement, notre approche concernant les signaux EEG pourrait être étendue à la modélisation d'une vaste variété des processus cognitifs et comportementaux.

Contents

Members of the Jury **v**

Abstracts **vii**
Abstract (in english) vii
Synopsis (en français) ix
Анотація (українською) xi

List of publications **xiii**

List of presentations **xv**

Preface **xix**

Résumé (en français) **xxi**

Contents **xxix**

1 Introduction **1**

2 Thesis composition **7**

I Modeling Biologically Plausible Networks **9**

3 Spiking neural networks **11**
3.1 Neural network stem 12
3.2 Neuron model 12
3.3 Neuronal connectivity 13
3.4 Synapse model 15
3.5 Spike-timing-dependent plasticity 16
3.6 Cell death 17
3.7 Spontaneous spiking activity 18
3.8 Efferent and afferent cells 18
3.9 Model implementation 19

Article A Dynamically organized neural networks **23**
A.1 Introduction 24
A.2 General concepts 24
A.2.1 The bio-inspired neural network simulator. 25
A.2.2 The distributed hierarchical framework. 25

A.2.3 The neural state recording facilities. . . . 26
A.3 Model implementation . . . 26
A.3.1 The neural network simulator . . . 26
A.3.2 Modules distribution . . . 27
A.3.3 Recording of module activity . . . 29
A.4 Discussion . . . 30

4 Hierarchical neural systems 33
4.1 Input/Output activity mapping . . . 34
4.2 Neural topologies . . . 35
4.2.1 Dynamic topologies . . . 36
4.2.2 Static topologies . . . 40
4.3 Artificial stimulus . . . 43
4.3.1 Stimulus structure . . . 43
4.3.2 Stimulus-driven development stages . . . 45
4.4 Bio-electrical signal model . . . 46
4.5 Hierarchical simulator architecture . . . 47

5 Evolutionary and genetic networks 49
5.1 Evolutionary algorithms . . . 50
5.1.1 Traits variation . . . 51
5.1.2 Evolutionary mechanisms . . . 51
5.2 Genetic algorithms . . . 52
5.3 Evolving hierarchical model . . . 53
5.3.1 Genome coding scheme . . . 54
5.3.2 Mutation procedure . . . 55
5.3.3 Fitness function . . . 56
5.3.4 Evolutionary framework conclusions . . . 56

II Bioelectrical Activity Analysis 57

6 Bioelectrical data analysis 61
6.1 Spatiotemporal patterns of activity . . . 61
6.2 Event-related potentials . . . 63
6.3 Spectral analysis . . . 65
6.3.1 Second order spectral analysis . . . 65
6.3.2 Third order spectral analysis . . . 67
6.3.3 Case study of non-linear signal analysis . . . 68
6.4 Spectral analysis results . . . 71

Article B Stimuli-driven functional connectivity 73
B.1 Introduction . . . 74
B.2 Hybrid system implementation . . . 75
B.3 Electrochipograms . . . 76
B.4 Power Spectrum Analysis . . . 77
B.5 Quadratic Phase Coupling . . . 79
B.6 Discussion . . . 80

7 Stimulus-evoked activity **83**
7.1 Spontaneous activity adaptation . . . 84
7.1.1 Spontaneous activity adaptation methods . . . 84
7.1.2 Activity adaptation results and discussion . . . 87
7.2 Reciprocal projections role . . . 91
7.2.1 Modules' characteristics . . . 91
7.2.2 Hierarchical setup . . . 92
7.2.3 Spatiotemporal stimulus setup . . . 93
7.2.4 Experimental virtual subjects . . . 93
7.3 Dynamics evoked by reciprocal projections . . . 94
7.3.1 Event-related Potentials . . . 96
7.3.2 Frequency Domain Analyses . . . 97
7.4 Discussion . . . 102

8 Approaches for non-linear data analysis **105**

Article C Structural modeling robust to outliers **107**
C.1 Introduction . . . 109
C.2 Methods . . . 110
C.2.1 GMDH Approach . . . 110
C.2.2 Robust Estimation Criteria . . . 111
C.2.3 Enhanced RPNN . . . 113
C.3 Results . . . 115
C.4 Discussion . . . 119

III Conclusions **123**

9 Final chapter **125**
9.1 Conclusions . . . 125
9.2 Future work . . . 128

Appendices **133**
List of acronyms . . . 133
List of figures . . . 135
List of tables . . . 137
Class diagrams of the framework . . . 139
Simulator's User Guide . . . 143

Bibliography **159**

Chapter 1

Introduction

> Good evening, ladies and gentlemen. We are tonight's entertainment! I only have one question. Where is Harvey Dent?..
>
> The Joker, The Dark Knight

Résumé :

Nous avons construit un nouvel et unique outil de simulation permettant de modéliser et d'étudier l'évolution des circuits multi-modulaires hiérarchiques de neurones à « décharges » qui comportent des particularités ontogénétiques et épigénétiques de développement. Nous avons mené des études de comportement émergeant de circuits structurés hiérarchiquement d'un module sensoriel et de quatre autres modules constitués de deux voies parallèles organisées également de façon hiérarchique. Notre approche permet de récupérer les propriétés générales des circuits neuronaux modélisés et d'établir des schémas de liaison fonctionnelle entre modules de circuit neuronal.

Growing experimental evidence that spike timing may be important to explain neural computations has motivated many neuro-scientists to use spiking neuron models, rather than the traditional rate-based models [26]. This created favorable conditions for development of a large variety of spiking neural network simulators. Such tools offer to the user the possibility to obtain precise simulations of a given computational neuro-paradigm and enrich their understanding of the processes beneath. However, the range of computational problems related to spiking neurons is very wide, as well as the number of non-resolved neuro-physiological problems they try to address.

The primary goal of this Thesis is to create a novel simulator based on biological principles of the mammalian cortex. From the very beginning we supposed it should support large-scale neural assemblies modeling, inter-cortical hierarchical connectivity modeling, and evolutionary and genetic features for large time-scale neural development tracking. Here, a reasonable question may arise: why do we need another neural network simulator?

The literature review gave us four spiking neural simulators which have strong support from scientific communities and are distributed on a charge-free, open source basis: NEURON, GENESIS, NEST, and NCS. There are many other smaller ones, but usually they are not so well developed, maintained and documented, so that evokes many difficulties of a practical kind. One of the oldest neural simulators is NEURON. Developed in early 90s, NEURON is a simulation environment for creating and using empirical models of biological neurons and neural circuits [102]. Initially it earned a reputation for being well-suited for models of cells with complex branched anatomy, including extracellular potential near the membrane, and biophysical properties such as multiple channel types, inhomogeneous channel distribution, ionic accumulation and diffusion, and second messengers. Although complex neuron models are supported, the NEURON simulator does not have, at least at the time the Thesis was started, a built-in mechanism for Spike-timing-dependent plasticity (STDP), which we wanted to be present in our model. Being written in C/C++, it features fast and efficient implementations of these neuronal models, at the same time, that implies neither easy simulator's expandability, nor adoptability to new or non-standard hardware, nor flexibility of the synapse model. Also, its primary targeting to huge supercomputers with thousands of processors was far beyond the scope of the hardware that was expected to be available to us.

Next simulator – GENESIS is also of that kind. GENESIS (the General Neural Simulation System) was given its name because it was designed to be an extensible general simulation system for the realistic modeling of neural and biological systems [22]. Typical simulations that have been performed with GENESIS range from sub-cellular components and biochemical reactions [20] to complex models of single neurons [124]. Here, "realistic models" are defined as those models that are based on the known anatomical and physiological organization of neurons, circuits and networks [21]. For example, realistic cell models typically include dendritic morphology and a large variety of ionic conductances, whereas realistic network models attempt to duplicate known axonal projection patterns. This level of biological plausibility was beyond our needs, to illustrate it we can say only that GENESIS is not normally provided with Integrate-and-Fire (IF) model neurons. While users have added, for example, the Izhikevich simplified spiking neuron model [82] and they could also add IF-based forms of abstract neuron models, these forms of neurons are not realistic enough for most GENESIS modelers [26] and were never considered as a part of the simulator.

The NEST initiative was founded as a long term collaborative project to support the development of technology for neural systems simulations [43]. NEST simulator was expected to address a problem of modeling neuronal networks of biologically realistic size and complexity. A lot of emphasis is placed on the efficient representation and update of synapses. In many applications the network construction has the same computational costs as the integration of the dynamics. Consequently, NEST parallelizes both. One of the distinct features of NEST is that it was designed to guarantee strict reproducibility: the same network is required to generate the same results independently of external factors, like the number of machines participating in the simulation. After all, we did not use this simulator, but it was serving as an example for us.

The NeoCortical Simulator (NCS), as its name suggests, is optimized to model

the horizontally dispersed, vertically layered distribution of neurons characteristic of the mammalian neocortex. NCS development began in 1997, a time when fascinating details of synaptic plasticity and connectivity were being discovered [100] yet available simulators such as GENESIS and NEURON did not offer parallel architectures nor the degree of neuronal compartmental simplification required for reasonable performance times. NCS uses clock-based IF neurons whose compartments contain conductance-based (COBA) synaptic dynamics and Hodgkin–Huxley formulations of ionic channel gating particles [68]. The compartments are allocated in 3D space, and are connected by forward and reverse conductances without detailed cable equations. Synapses are COBA, with phenomenological modeling of depression, facilitation, augmentation, and STDP. No nonlinear simplifications, such as the Izhikevich formulation, are supported. Implementation of inter-cortical interactions in our simulator was inspired to some extent by NCS.

In brief, the first two simulators were oriented towards very deep and detailed neuronal morphology and geometry, they exceeded our needs in terms of bioplausibility. We were oriented towards IF-based neuronal models, because they are relatively fast to simulate and are better adapted for large-scale parallel modeling [26]. When modeling, precision is lost on the level of individual neuron, but then that is compensated by mass-effect emerging from larger cell assemblies simulated. The NeoCortical Simulator used interesting approaches to multi-layered dispersed network modeling, but, except this part, it does not fit very well to our initial concept. We saw in 2008 that its development was in stagnation phase (now we know that it was completely abandoned in 2009), so that was not an option. NEST could be a good candidate to work with, but it was not designed for hierarchical network simulation, which was important for us. Anyway, neither of four simulators supported evolutionary features and there was no clear way to add it. Finally, given that the simulator was supposed to be compatible with PERPLEXUS project's computation's acceleration hardware (oriented towards Java-code translation), we had a solid set of reasons to start an independent simulator development. In this frame we started a novel simulator, which should surpass existing ones with its evolutionary-oriented features and hierarchical circuits support, which would shed light on inter-cortical processes having place in hierarchical evolvable neural circuits, such as brain.

The PERPLEXUS was an international project of the European Commission. The aim of the project was to develop a scalable hardware platform made of custom reconfigurable devices endowed with bio-inspired capabilities that will enable the simulation of large-scale complex systems and the study of emergent complex behaviors in a virtually unbounded network of computing modules. In other words, it was targeted to power a swarm of intelligent robots with capabilities to solve given tasks. Created infrastructure was tested to prove its usefulness as a powerful and innovative simulation tool in the following applications: neural networks modeling, culture dissemination modeling, and cooperative collective robotics modeling. All of these applications are computationally heavy and implementation of such complex models requires high performance of the simulation that can be achieved thanks to combination of a powerful hardware platform, bio-inspired capabilities, dynamic features, flexibility of artificial neuronal model and data processing parallelization techniques.

Most of the Thesis' development has been done during the project, when our Grenoble team was working on the neural network modeling application. We expected that the software and hardware frameworks developed in the project will greatly speed up the simulation and will aid in creation and analysis of large biologically plausible networks of spiking neurons. After the first year of the project venue, when the initial versions of the software and the hardware were developed by the respective teams, it became clear that our expectations about the simulations will be fulfilled only partly. Unexpected difficulties and complications with design and production of the custom computation module met by our colleagues were leading to certain delays and ambiguities in development. It turned out that, while we can simulate biologically plausible neural networks within the framework, in the absence of the final hardware either simulation speed or size of the network will be different from the expected ones. At the moment, we decided to make accents not on study of emerging behaviors of large-scale stand-alone Spiking Neural Network (SNN), but on simulation of distributed hierarchical neural systems, where a set of smaller biologically plausible neural tissues inter-connected by relatively sparse projections (in comparison to the density of internal connections inside each tissue) were simulated. In this way we could efficiently apply the framework to simulate large cell assemblies, while using smaller neural networks as building blocks in the large hierarchical circuits. This approach was inspired by real brains' organization, where specialized areas of neurons are connected to other ones, forming complex signal processing chains of the brain. Fortunately, the project's architecture was favorable to a small army of independent, thus parallel, computational agents connected by the means of the Internet Protocol (IP) networks, which agrees with the concept of the project. Another consequence of deviation from the initial development plan was utilization for testing purposes of a less powerful universal processor (Intel XScale), instead of a custom-build bio-inspired processor and in our turn we were forced to drastically reduce the size of the stand-alone SNN. Despite it was known to have noticeable impact on pattern generation abilities of the network.

At the same time, we were looking for a bio-inspired neural network state acquisition technique, which should be fast and efficient in computation and which could provide us with a possibility to apply analysis methods well-known from neurophysiological studies. We knew that there are lots of studies on EEG modeling and closer inspection of the topic revealed that there are two main approaches used: detailed and macroscopic modeling of EEG signals. The macroscopic models such as the neural mass model (NMM) and the mean-field model were originally developed to simulate activity in the olfactory cortex [55] and emergence of the spontaneous alpha rhythms [94]. These models represent the mean activity of a large neuronal populations and could be used to study generation of the epileptic activities [145], to analyze the connectivity and coherence on EEG rhythms [151], etc. In other words, because of relatively small number of parameters these models could be tuned to empirical data and then used like predictors [11]. In contrast, detailed models due to the very large number of parameters involved and the strong dependencies between them, cannot be fit to empirical data, but they can be used for simulations [41, 70]. That fit well to our concept, so we introduced a way to record neuro-mimetic signals from the simulated network. We called that approach – electrochipography (EChG). These signals were captured by the means

of virtual electrodes injected in the neural network model and are characterized by dynamics and features similar to those recorded in living brain structures, such as EEG, electrocorticography (ECoG) and Local Field Potentials (LFP). According to the model used, EChG could be considered as similar to the LFP, which is a particular class of electrophysiological signals, dominated by the electrical current flowing from all neighboring synaptic activity of a cell tissue. At mesoscopic level, the recording of brain activity by means of EEG, ECoG and LFP collects the signals generated by neurophysiological processes of cell assemblies. Thus, the underlying non-linear dynamic activity of the system can be extracted from the signal [27] by the means of second and third order poly-spectral analysis methods [106] and by non-linear regression methods with dependency structure discovery.

Many brain analysis studies are targeted on discovery of the areas' specialization and information processing chains of the brains, conducted on data recorded by the means of EEG. Among the approaches most frequently used for such analysis are multivariate regression methods. Classical linear regression methods are widely used in quantitative-structure relationship studies because of fast and mature implementations and ease of results interpretation. They reveal channels which participate more than others in the output and give a quantitative measure of such involvement. Another automatic structure modeling approach working with non-linear dependencies in the data is a set of methods united under the title of Group Method of Data Handling (GMDH), which make use of Polynomial Neural Networks to achieve the result. Initially proposed by Ivakhnenko in the late sixties of the XX-th century [79, 150] they were used for identification, pattern recognition and short-term forecasting. They are featuring automatic structural dependency discovery with a search though non-linear model space. Although wide-spread usage of traditional linear, and sometimes non-linear, analysis methods for EEG analysis, such data often contaminated by very strong artifacts or outliers created by muscular activity or by external events, which have no correlation with experiment. That is why, the *robust* analysis methods with automatic structure disclosure should be used to obtain meaningful undistorted results. We proposed a method named Enhanced Robust Polynomial Neural Network (ERPNN) based on GMDH paradigm and enhanced with a robustness to presence of noise and outliers in explanatory and response variables of the data. Thus the power of structure analysis of non-linear dynamic activity of the brain is increased once by the absence of oversimplification introduced by linear models of activity [27], i.e. by application of non-linear model identification offered by GMDH, and then increased in a second time by a robust model parameter estimation. Although, the proposed algorithm was successfully tested on artificially generated data and has shown good results, the data obtained from the simulations was analyzed with other approaches developed initially to work with real EEG signals, these were evoked potential analysis, power-spectrum density analysis [47], and third-order spectral density analysis [93, 105]. Application of the ERPNN algorithm to real or simulated bio-electrical data is a matter of future work.

The results of the PERPLEXUS project were successfully presented to the experts of European commission in the spring of 2010 in Lausanne. At the moment a working implementation of the distributed neural simulator with bio-electrical signal recording and with dynamic topologies features was done. Starting from here, we were not limited any more by the frame of the project's goals and by the

constrains imposed by the custom-made processor, so we have started an adaptation of the simulator to run it on the powerful Mac OS X based cluster. Which was done in a short time, thanks to initial cross-platform and distributed-platform targeting of PERPLEXUS's framework. Having the powerful hardware in our disposition, we scaled the neural network model to a large enough size and set up a final goal to implement an expansion module to the simulator which will add support for evolvability of the neural systems modeled on a large time scale.

The evolvability means that one or multiple neural systems will be created according to a particular "chain of model parameters' values" – circuits' genome. The framework will allow the neural systems to replicate them-selves under certain circumstances. Replicas will not be perfect copies of the parent system, but will be affected by random changes in the genome – the mutations, which will have an effect on the emerging behavior of the systems. The framework will allow studing ongoing development of the neural systems during the generation series.

Finally, without claiming completeness in a huge problem of understanding the brain activity patterns, which is one of the major tasks of humankind, here, we will try to put a light on one particular part of the issue. We decided to create a novel simulation framework, which allows to model evolvable hierarchical neural networks undergoing neural developmental phases simulated by the means of biologically plausible neural models and to capture virtual bioelectric signals. Modeled neural systems will allow conducting experiments with higher level of precision and control than achievable in vivo and to decrease an amount of external unexplainable noise in the data. We will demonstrate that our simulation framework under certain conditions could produce results similar to observed in brains.

Chapter 2

Thesis composition

Résumé :

Cette thèse est composée en trois parties : la première partie décrit le modèle de réseau de neurones ; la seconde est consacrée aux méthodes d'enregistrement et d'analyse des signaux bioélectriques émergeant du réseau et à la discussion des résultats obtenus. Les conclusions sont présentées dans la troisième partie de ce travail.

This Thesis is written in the form of compilation of articles, according to the french higher education rules. Three articles published during the years of work on the Thesis are assembled together, joined by intermediate parts and elucidated in a way, which assures seamless and fluid comprehension of the work done in the frame of the Thesis. According to the general academic paper presentation layout the Thesis is divided into three parts: the Part I: "Modeling Biologically Plausible Networks" – explaining the neural models used in the simulations, the Part II: "Bioelectrical Activity Analysis" – explaining experiments organization and data analysis done, and finally the Part III: "Conclusions" – summarizing the discussion of the results obtained and making the final conclusions on the work done. The articles included to the appropriate parts of the thesis so that they either summarize the work described in a part or a chapter or they give a self-contained description of the advancements done. We will avoid repetition of the topics covered by the articles, whenever possible, without sacrificing the comprehensibility of the Thesis.

The Part I: "Modeling Biologically Plausible Networks" is sub-divided into three chapters: the first chapter shows the model of stand-alone SNN, then Article A: "Dynamically organized neural networks" summarizes the model and introduces hierarchical neural networks. Second chapter continues with description of the approach used to organize SNNs neuronal modules in a hierarchical way creating neural circuits of complex topologies, and the third chapter focuses on genetical, genomic and evolutionary features (on large time scale) of the simulation framework.

In the Part II: "Bioelectrical Activity Analysis" we explain in to details analysis approaches used, which includes event-related and time-frequency technics. These

technics were applied in a study described in Article B: "Stimuli-driven functional connectivity" to make a comparison between EEG-like signals obtained from circuits modeled with the simulator and real patients' EEG data. In the following chapter an experiment on functional connectivity between neural modules is explained and the results obtained are discussed. This part is closed by a conceptual application of a robust non-linear structural and parametric regression approach for bioelectrical data-mining, explained in Article C: "Structural modeling robust to outliers".

Finally, conclusions and directions for future work are given in the Part III: "Conclusions".

Part I

Modeling Biologically Plausible Networks

Chapter 3

Spiking neural networks

> The number of possible "on-off" patterns of neuronal firing is immense; the fact that it is often organized and functional is quite an accomplishment!
>
> Daniel J. Siegel

Résumé :

Ce chapitre introduit les structures biologiques et les concepts théoriques que nous avons cherché à modéliser. Certains aspects développementaux et fonctionnels du système nerveux central sont abordés. Les notions de réseaux, de neurones, de synapses, de plasticité synaptique à modulation temporelle relative (STDP), du processus d'apoptose et de l'activité neuronale spontanée sont présentées ici, de même que les aspect pratiques d'implémentation du modèle.

A huge part of the Thesis' work was done in the PERPLEXUS project, which was aimed to develop a scalable and distributed platform for simulation of large-scale auto-organisative phenomena, and to the observation of emerging complex behavior. The project's framework creates a virtually unbounded network (Ubinet) of modules (Ubidules), each featured a bio-inspired reconfigurable custom processor, which provides the platform with development, learning, and evolution capabilities. Those features are particularly interesting for an application in the realm of the neuro-mimetic neural networks.

From the very beginning we have decided to use a neural network model, which is based on the large number of Leaky Integrate-and-Fire (LIF) spiking neurons. The Integrate-and-Fire (IF) models and its derivatives have been used in a wide variety of studies ranging from investigations of synaptic integration by single neurons to simulations of networks containing hundreds of thousands of neurons [32]. The IF models has proved to be particularly useful in elucidating the properties of large neural networks and the implications of large numbers of synaptic connections in such networks. Integrate-and-fire model have an important role in the debates about the origin and nature of response variability in cortical neurons

[125]. The LIF model was selected as the one providing good compromise between bio-plausibility and requirements to the computational power. More complex models based, for example, on the Izhikevich neuron [82], were not used, because of limited computation power at our disposal.

Our model includes such important bio-inspired features as 2 types of neurons (excitatory and inhibitory), Spike-timing-dependent plasticity (STDP) process, apoptosis process during early development phase, synaptic pruning and cellular death processes.

The model suggests the cells are arranged in a 2 dimensional square lattice. Cells are differentiated into excitatory (80%) and inhibitory (20%) neurons during early-development stem phase [25]. Initial connections between cells are created according to the 2D Gaussian distribution. Synapse distribution parameters differed depending on the type of the neuron it is connected to. Excitatory cell projections were around three times shorter than the ones of inhibitory cell. This measure proactively eliminates creation of a compact self-contained cell clusters. In order to avoid the negative edge-effects which can occur on a real square lattice, our lattice was considered to be wrapped around a torus surface, so that cells on the opposite edges are adjacent. A brief description of standard LIF model and STDP model necessary for comprehension of the further results are given in the following sections. We emphasize on the features added to the model, while very detailed description of standard models could be found elsewhere [18, 72, 74, 32, 142].

3.1 Neural network stem

In the mature brain, the cerebral cortex appears as a layered structure (layers I-VI, [16]) characterized by the changes in the density of cells and neuropil morphology. The human cerebral cortex is a highly folded sheet of neurons at a density of *circa* 10^5 neurons per mm^2. One half of these cells are pyramidal cells which are characterized by the distal connection of their axon and are the primary excitation units of the mammalian prefrontal cortex [2]. The ability of pyramidal neurons to integrate information depends on the number and distribution of the synaptic inputs they receive.

Our model considers that initial cell stem is a 2-dimensional ($N \times N$) square lattice filled with neurons (N^2 neurons in a total). Once the stem cells are located on their final positions on the surface, they differentiate into 2 cellular types with different probabilities. Type I neurons make excitatory projections, while Type II neurons make inhibitory connections. This probability reflects the ratio between the different cellular types as determined by the genome (see Figure 3.1). The physiological ratio is about four excitatory neurons per one inhibitory neuron [25], leading to approximately to 80% excitatory and 20% inhibitory neurons.

3.2 Neuron model

According to the chosen LIF neuromime model, at each time step the value of the membrane potential of the i^{th} neuron, $V_i(t)$, is calculated according to next

iterative equation

$$\begin{aligned} V_i(t+1) &= V_{\text{reset}} + B_i(t) \\ &\quad +(1-S_i(t))((V_i(t)-V_{\text{rest}})k_{\text{mem}}) \\ &\quad +\sum_{\forall j} w_{ji}(t) \end{aligned} \tag{3.1}$$

where V_{reset} corresponds to the value of resting potential of the neuron; $B_i(t)$ is the background activity arriving to the neuron (see Section 3.7 for details); $S_i(t)$ is a state of the neuron for given time t, which can be either 0 (non-spiking) or 1 (spiking); $k_{\text{mem}} = \exp(-1/\tau_{\text{mem}})$ is the time constant associated with the leakage of the neuron, and $w_{ji}(t)$ is the post-synaptic potential of the j^{th} neuron projecting to the i^{th} neuron (see Section 3.4 for details).

The state of a neuron $S_i(t)$ is a function of the membrane potential $V_i(t)$ and a threshold potential $\theta_{[q]}$ of the neuron of type q, such that

$$S_i(t) = \mathcal{H}(V_i(t) - \theta_{[q]}). \tag{3.2}$$

where $\mathcal{H}$ is the Heaviside function

$$\mathcal{H}(x) = \begin{cases} 0 & : \quad x < 0 \\ 1 & : \quad x \geq 0 \end{cases} \tag{3.3}$$

In addition, the state of a neuron depends on the refractory period $t_{\text{refract}[q]}$, which forbids continuous spike bursts, such that

$$S_i(t+\Delta t) = \frac{(t_{\text{refract}[q]} - \Delta t)}{t_{\text{refract}[q]}} \cdot S_i(t) \tag{3.4}$$

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 neuron generates a spike exactly for $S_i(t) = 1$.

You can see an example of membrane potential $V_i(t)$ dynamics during 160 ms of life of a neuron on the Figure 3.2. The neuron has $V_{\text{reset}} = -78\ mV$, $\theta_{[q]} = -40\ mV$ and $t_{\text{refract}[q]} = 3ms$ a spike train of random activity is coming ranging from 0 mV to 8 mV. The current value of membrane potential is plotted in the bottom part of the Figure, while on the upper part a sum of random incoming activity is shown (displayed as bars). One can see a positive discharge of 1 mV happens as a membrane potential is reaching the threshold. At time $t = 130\ ms$ input activity is stopped and one can see an exponentially decaying potential.

3.3 Neuronal connectivity

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 [78]. The concentration of the neuro-mediator determines the possibility to establish a connection with another cell and it is a function of the two pre-/post-synaptic cell types (see Figure 3.1). This random connection pattern results in a larger number of connections in the vicinity (short projections) compared to long-distance

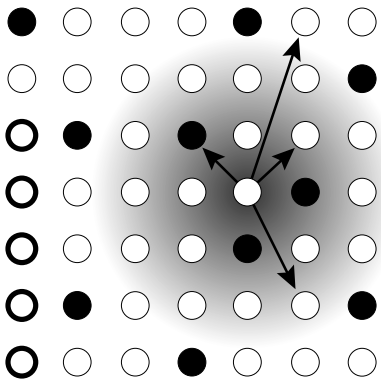


Figure 3.1: A schematic connectivity map, showing the gradient-based synaptogenesis for one excitatory neuron. Excitatory neurons are displayed as white circles and inhibitory as the black.

projections. Despite the fact that the genome codes for the diffusion parameters on a per-neuron-type basis, the use of two gradients introduces variability in the connection patterns, resulting in the production of diverse phenotypes trough the same ontological process. We were developing our neural network model anticipating that each neuron will receive around 3% of all possible synapses from potentially any other neurons. Synaptic projections are established according to the 2D Gaussian distribution. Projections originating from the inhibitory cells in the average are 3 times longer than those originating from the excitatory cells. No generation of new projections is allowed, although specific rules could be defined to this purpose.

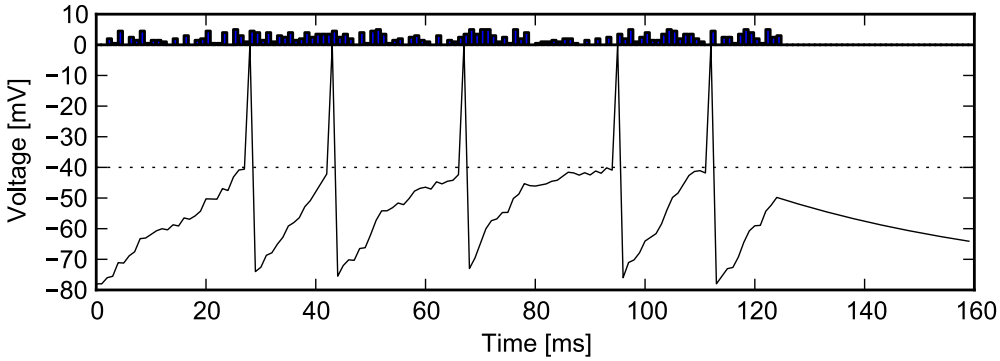


Figure 3.2: Sample dynamics of LIF neuron's membrane potential. Current value of membrane potential is plotted by line. A sum of random incoming activity is shown by bars on the upper part of the figure. Discharge threshold $\theta_{[q]} = -40\ mV$ is shown by dotted horizontal line. At time $t = 130\ ms$ input stimulation is stopped. and The membrane potential returning back to the V_{reset}.

To avoid negative edge effects it is assumed that cells at edge of the lattice are located topologically near the cells from the opposite edge, such that the whole network is virtually located on a torus surface.

3.4 Synapse model

Synapses can change their strength in response to the activity of both pre- and post-synaptic cells (see Section 3.5). 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 (e-PSPs), whereas LTD is measured as a persistent decrease in the amplitude of the e-PSPs.

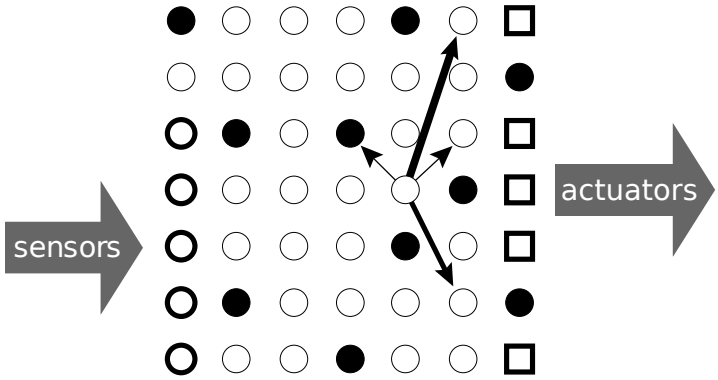


Figure 3.3: Arrow thickness: representation of the synapse state. Strong synapses are represented by thicker arrows than week synapses. Excitatory neurons selected for external input are shown as circles with a thick stroke and neurons selected to project activity are shown as squares with a thick stroke.

Physiological studies suggest that the strength of the synapses may vary between discrete mechanistic states, rather than by adjusting their efficacy along a continuum (see [104]). We recapitulate here the five synaptic states that have been suggested [103] without entering into the molecular details:

active state – normal state of synapse;

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. Though, they can be potentiated in the same way as active synapses;

recently silent state – silent synapses undergoing LTP enter this state. It differs from the active state in that synapses cannot undergo LTD.

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 $exc \to exc$ and $exc \to inh$ synapses are *integer-valued levels* $A_{ji}(t)$, with $A_{ji}(t) \in \{[A_1] = 0, [A_2] = 1, [A_3] = 2, [A_4] = 4\}$ (here and after *exc* is used as a shortcut for "excitatory cell", and *inh*, as a shortcut for "inhibitory cell"). Index j is referred

to as the presynaptic neuron and index i as the post-synaptic neuron. These discrete levels could be interpreted as a combination of two factors: the number of synaptic boutons between the pre- and post-synaptic neurons and the changes in synaptic conductance. We attributed a fixed activation level (that means no synaptic modification) $A_{ji}(t) = 1$, to $inh \rightarrow exc$ and $inh \rightarrow inh$ synapses. The system's genome determines the $A_{ji}(t = 0)$ value to use for a newly differentiated synapses. Schematical illustration can be seen on Figure 3.3.

3.5 Spike-timing-dependent plasticity

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

It has been proposed to explain the origin of LTP, i.e. a mechanism for reinforcement of synapses repeatedly activated shortly before the occurrence of a post-synaptic spike [87]. 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 post-synaptic spike [83]. This rule is used in the model [67, 37].

The important consequences of synaptic strength change is possible production of stronger information transmission. The post-synaptic potential w_{ji} is a function of the state of the presynaptic neuron S_j, of the "type" of the synapse P_{ji}, and of the activation level of the synapse A_{ji}. It is expressed by the following equation

$$w_{ji}(t) = S_j(t-1) \cdot A_{ji}(t-1) \cdot P_{ji}. \tag{3.5}$$

A real-valued variable $L_{ji}(t-1)$ is used to implement the STDP rule for $A_{ji}(t-1)$, with integration of the timing of the pre- and post-synaptic activities. The STDP defines how the value of L_{ji} at time t is changed by the arrival of presynaptic spikes, by the generation of post-synaptic spikes and by the correlation existing between these two events. On the generation of a post-synaptic 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 post-synaptic spike. The 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} neurons 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, according to the equation:

$$M_i(t+1) = S_i(t) \cdot M_{\max[q]} + (1 - S_i(t)) \cdot M_i(t) \cdot k_{syn} \tag{3.6}$$

where $M_{\max[q]}$ a resting decay value and k_{syn} is a time constant associated with a memory properties – synaptic plasticity – of a neuron.

3.6 Cell death

The adult pattern of neuronal connectivity in the cerebral cortex is determined by the expression of some genetic information and by epigenetic processes associated to plasticity and learning. During the early stages of development, excessive branches and synapses are initially formed and distributed somewhat diffusely [78]. This over-growth phase is generally followed by massive synaptic pruning [114] partially associated with genetically or pathologically programmed cell death.

Cell death may be provoked by two mechanisms: an excessive firing rate and the loss of all excitatory inputs. An excessive firing rate is assumed to correspond to the biological effect known as glutamate neurotoxicity, that induce the expression of genes provoking cell death [38]. On the contrary, inactive synapses that reach the lowest activation level disappear due to the absence of activity that affects the function of the mitochondries maintaining the synapse through the provision of energy.

Model suggests that during an initial "early developmental phase" for each time step an average firing rate of the neuron is computed. It is computed over a running window corresponding to 50 ms. For each type of neuron a maximum firing rate (FRM) was determined following a parameter search procedure. In this study we used $FRM_{exc} = 245$ $spikes/s$ and $FRM_{inh} = 250$ $spikes/s$. If average spiking rate FR_{50} exceeds FRM for the corresponding neuron type the cell had a probability to die according to the following function

$$P_{\text{death}}(t) = \frac{0.5 \cdot t^2 - 4.5 \cdot 10^{-6} \cdot t^3}{44 \cdot (2.5 \cdot 10^6 + 6 \cdot 10^{-3} \cdot t^2)}. \tag{3.7}$$

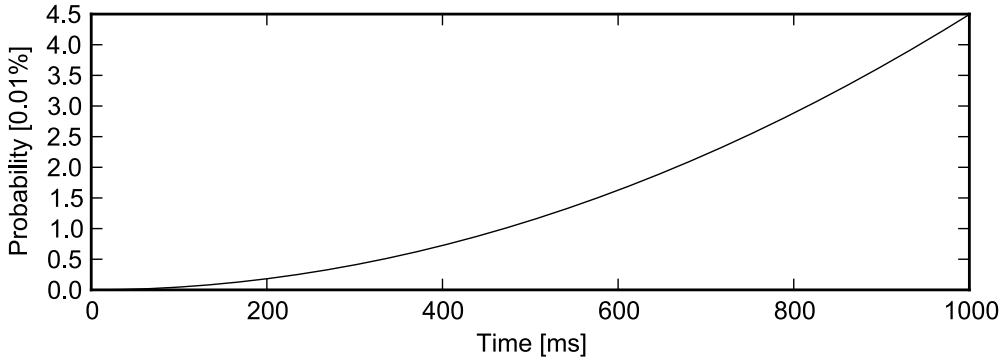


Figure 3.4: Cell death probability (in 0.01% units) as a function of time.

The early developmental phase, characterized by cell death provoked by excessive firing rate, begins at time $t = 0$ and lasts until $t = 750$ ms. At the time step next to the end of apoptosis process the STDP process is activated. The addition of this feature greatly improved the stability of the network while maintaining its ability to produce spatiotemporal firing patterns. We assume that developmental and learning processes are likely to potentiate (or weaken) certain pathways through the network and provoke an appearance of cell assemblies characterized by recurrent firing patterns.

3.7 Spontaneous spiking activity

The central nervous system is a huge neural network whose activity varies continuously without interruptions. Even at what is considered to be a low pace of brain activity (like in sleep, in relaxed sensory deprivation states or anesthesia) neurons receive inputs from many functional areas of the brain without any apparent relation. This activity, referred as spontaneous activity or background activity, is often considered to be associated to a kind of global controller able to increase or decrease the reactivity of the system to a an information flow according to its global pertinence. This mechanism could explain the differences in our reaction times according to our state of vigilance (fast in alertness, slow in drowsiness and even absent in deep sleep or coma).

With the advent of the network activity, several processes take place under the control of the genetic information. In the model the background activity $B_i(t)$, as defined in Equation 3.2, is used to simulate the input of afferents to the i^{th} neuron that are not explicitly simulated within the neural network. We assume that each neuron receives n_{ext} external afferents. We simplify by a setting that all neurons receive the same number of external projections of same strength, thus not affected by the regular STDP, and that all of them are excitatory. Spike train of the external input is distributed according to an independent random Poisson process with mean rate λ.

One should note, that for smaller networks the rate of external background activity is a critical parameter, though for larger networks it plays lesser role. In the absence of the background activity neural networks are slowly dying, as all projections are going to the silent state thus producing no spikes.

3.8 Efferent and afferent cells

A sub-set of excitatory neurons (about 10% of all excitatory neurons) differentiate into the "sensory neurons" of the network. They are randomly distributed over the surface of the network and their dynamics are the same as all the other excitatory neurons, except that they can receive input from outside of the neural network. We assume that the input stimulus might be an external stimulus from a source like a microphone, an infrared photodiode, a camera or an artificial stimulus specially developed for an experiment, or also in can be a spiking activity originating from the output layer of another neural network in a hierarchical neural system. Do not be confused by this incoming activity and the spontaneous network activity (mentioned in the Section 3.7), which each neuron can receive with certain probability regardless of its type and regardless of presence external stimulus.

Another group of the excitatory neurons, approximately of the same quantity as "sensory" ones, differentiate into actuator cells. Their dynamics is also the same as all the other excitatory neurons, except that their spiking activity can be send to other neural networks in hierarchical systems. A schematic representation of the principle is shown on the Figure 3.3. Please note, although on the Figure for the sake of simplicity "input" and "output" layers are grouped, in the modeled network, they are are scattered randomly over the network surface.

3.9 Model implementation

According to the PERPLEXUS project requirements the model is implemented in Java programming language. In this way the simulator code can be split into two parts by the translator software developed by our colleagues [29]: a custom-processor accelerated part, which is converted into special assembly instructions and a general part to be executed on a general purpose processor. The simulator is written in modular way providing rich possibilities of future extension. Java-based it can be easily run on virtually any hardware platform from a tiny mobile phone up to a large powerful cluster.

The simulation dynamics followed a discrete time scale with time units here-in-after referred to as *time-steps*. The extensive use of Fast Fourier Transform (FFT) in our signal analysis imposed, for improved efficiency, usage of data acquisition frequencies which are powers of two. In practice, the size of time-steps of the simulator was adjusted to facilitate the analysis' computations – 1024 *time-steps* correspond to 1000 *milliseconds*.

Introduction to hierarchical neural systems

Inspired by biological systems, where it is common to have hierarchical information processing formed by information processing chains of specialized brain areas where every area is responsible for its particular type of processing and driven by PERPLEXUS project unexpected hardware limitations we have decided to shift our focus from the simulation of large stand-alone neural networks to the simulation of large distributed systems of smaller stand-alone neural networks.

Next Chapter will be focused on the models used to build hierarchical neural circuits. But before that we present Article A: "Dynamically organized neural networks", where stand-alone neural network model is summarized and a novel framework, aimed to drive studies of emerging behavior of hierarchical neural networks, is described along with short explanation of virtual electrode approach that allows to record electrochipographys (EChGs) neuro-mimetic signal. The framework described in the article is based on evolving spiking neural network model, which is certainly an oversimplification of the reality, but it paves the way to models which will embed increasingly higher biologically inspired parameters. A particular feature of the approach used is a possibility to enable modeling of highly dynamic environment characterized by evolvable topologies with modules can join or leave the simulation at any time. The drawback is that fast changing topologies might introduce delays of information processing and will inevitably slow down whole simulation. Nevertheless, the framework fits well to the requirements of sophisticated control circuits for robotic applications in collective behavioral studies where each robot is driven by one or more neural network [45]. That is achieved by utilization of the Java Agent Development Environment (JADE) library – an agent-based programming framework characterized by reduced footprint and compatibility with mobile robot-oriented Java environments [17].

Then in further sections, while trying to avoid repetition between the article and the Thesis' body, we will continue with a mapping scheme used to interexchange spiking activity between the neural networks (Section 4.1), with a description of the neural circuit' topologies (Section 4.2), the principal structure of artificial stimulus used in the article and in the following experiments (Section 4.3), a very brief explanation of the bio-electrical signal model (Section 4.4), as it is described in the Article A: "Dynamically organized neural networks" and in the Article B: "Stimuli-driven functional connectivity". We will conclude the chapter by covering an architecture and implementation details of the developed simulator of distributed neural systems.

Article A

A framework for simulation and analysis of dynamically organized distributed neural networks

Authors:	Vladyslav Shaposhnyk, Pierre Dutoit, Victor Contreras-Lámus, Stephen Perrig and Alessandro E. P. Villa
Published in:	Lecture Notes in Computer Science Artificial Neural Networks – ICANN 2009
Publisher:	Springer-Verlag Berlin Heidelberg
Volume:	5768
Pages:	277-283
Year:	2009

Résumé :

Dans ce travail, nous présentons un outil de modélisation et d'analyse des propriétés émergentes de réseaux neuraux multimodales.
Chaque module est représenté par un réseau neuronal déchargeant. Notre modèle est un ensemble de neurones du type "leaky integrate-and-fire" bidimensionnel qui comporte des traits génétiques, ontogénétiques et épigénétiques.
Une libraire de logiciel (JADE : Java Agent Development Environment) nous a permis d'implémenter une machine artificielle (un automate) qui gère de façon efficace des système quasi illimités d'agents communiquant par messages. Cette approche nous permet de gérer les systèmes dynamiques des modules neuraux interconnectés. A n'importe quel moment, chaque module a la possibilité d'entrer ou de sortir du système de simulation tandis que le système s'adapte automatiquement à sa topologie.
Conjointement, chaque module comporte une électrode virtuelle qui capte un signal généré par des décharges neuronales. Ce signal, electrochipogram (EChG), a des propriétés proches des signaux réels enregistrés par les moyennes des potentiels de champs locaux (LFP) ou electroencephalogrames (EEG). Puis une analyse spectrale et en potentiels liés à l'événement peut être réalisée sur ces signaux pour trouver des patterns du comportement des réseaux de neurones.

Abstract

We present a framework for modelling and analyzing emerging neural activity from multiple interconnected modules, where each module is formed by a neural network. The neural network simulator operates on a 2D lattice tissue of leaky integrate-and-fire neurons with genetic, ontogenetic and epigenetic features. The Java Agent DEvelopment (JADE) environment allows the implementation of an efficient automata-like virtually unbound and platform-independent system of agents exchanging hierarchically organized messages. This framework allowed us to develop linker agents capable to handle dynamic configurations characterized by the entrance and exit of additional modules at any time following simple rewiring rules. The development of a virtual electrode allows the recording of a "neural" generated signal, called electrochipogram (EChG), characterized by dynamics close to biological local field potentials and electroencephalograms (EEG). These signals can be used to compute Evoked Potentials by complex sensory inputs and comparisons with neurophysiological signals of similar kind.

Keywords: spiking neural networks, hierarchical neural networks, distributed computing, computational neuroscience, bio-informatics

A.1 Introduction

The brain represents by far the most complex organ of the human body and its simulation will certainly remain out of reach for a long time. However the principle that govern its development and processing represent a source of inspiration for the design of artifacts [63]. In principle the design would consist to create programs that reproduce cognitive processes directly at higher representational level or to create *in silico* artificial neural network systems. The project PERPLEXUS is aimed at developing an ubiquitous, scalable and distributed platform dedicated to the simulation of large-scale self-organising networks and to the observation of potentially emerging behaviours [123, 138]. This platform is composed of custom reconfigurable devices endowed with computing, behaving and communicating modules called *Ubidules*. They are based on a custom designed processor called *Ubichip* and are characterized by custom designed bio-inspired features such as growth, learning, and evolution.

This paper advocates that a network of Ubidules may offer an interesting platform to implement a network of dynamically interacting modules characterized by integrate-and-fire neuromimes. In particular we present the JUbiNet simulator of distributed neural networks developed in the frame of the PERPLEXUS project and some examples of its output in the form of brain-like recorded signals.

A.2 General concepts

JUbiNet is a highly expandable and flexible framework aimed at simulating hierarchical neural systems. The framework is based upon three major levels, which are: phylogenetic, ontogenetic, and epigenetic. All components are organized in a modular way such to enable inter-operability, compatibility, and expandability of the system and its parts on all levels [29].

At the Phylogenetic level several simulated neural system features (neural network parameters, topology rules of distributed network, etc.) are encoded and stored in a

genome, distributed to lower levels of the application. Selection of alternate values of the parameters (i.e., the *alleles*) is performed at this level associated to a computational neurogenetic modeling [18].

The Ontogenetic level describes the origin and the development of the system during its early stages of development. Genome decoding, neural network initialization, and inter-network connection establishment rules are performed within this conceptual level. The Epigenetic level refers to learning features, which are limited to an individual lifetime. The neural network simulator itself naturally fits in this layer.

The current version of JUbiNet carries a full implementation of the epi- and ontogenetic levels and partially the phylogenetic level. JUbiNet is provided with flexible configuration facilities, a collection of data processing objects and network handling that allows the simulation of customized spiking neural networks organized in topologies of interest.

A.2.1 The bio-inspired neural network simulator.

The simulator is designed to efficiently emulate neural network models with emphasis on facilities for model reconfiguration and adjustment and on functionally rich possibilities for detailed network state acquisition. The neural simulation consists in a set of processes run over a set of neurons.

The processes in the neural network fully determine the functional model that includes processes such as synaptogenesis, activity transmission, state recorder, learning, etc. The simulator defines a set of interfaces to general neural concepts and property access routines, like: neuron, synapse, network, signal-processing routines, input/output routines. With predefined implementations of standard objects it is possible to assemble common neural network models. The interfaces provided by the simulator are designed to extend or replace all default objects by user defined ones.

A.2.2 The distributed hierarchical framework.

The simulation begins when a network is composed with agents (of software or hardware nature) running distributed networking modules. Those agents are waiting for genomic information, which is prepared and transmitted by simulation of planning or phylogenetic modules. Genome decoding triggers network initialization and the simulation starts when all systems are initialized.

The distributed network simulator is divided into four main parts: the network discovery system, the link manager system, the input/output mapping and conversion processes, and the neural network simulator itself. The network discovery system maintains and updates the list of available agents and their network role, which identifies the inter-modules connectivity pattern. The link management system is instantiated as soon as minimal information about the actual state of the network is gathered by the network discovery system. The link manager establishes the characteristics of data-processing connections between agents. The number, type, size, and direction of data-flows could differ in accordance to the information in the agent's genome.

The simulation starts as soon as mandatory data-processing links are established. Input/output mapping and conversion routines are executed in order to handle data translation from the internal simulator format to the format suited for data transmission on physical supports (i.e., Bluetooth, WiFi, etc.). It is possible to emulate distributed neural networks with dynamic or static topologies, with different triggering events associated with topological changes, with different synchronisation routines or different behavior patterns within given interfaces and protocols.

A.2.3 The neural state recording facilities.

The electroencephalogram is the most commonly used signal to detect and analyze brain activity. The biophysical model of EEG generation relies on the assumption that the current flows generated by clusters of simultaneously active synapses produce an elementary signal [51]. By means of virtual electrodes we aim at implementing the recordings of local field potentials of densely interconnected mesh-work of simulated neurons. The virtual electrode recorder is implemented by the simulator process and easily integrated in the neural network simulator.

A.3 Model implementation

The main goal of our simulation is to emulate the biologically plausible behavior of hierarchically organized inter-connected brain areas receiving external inputs from sensory modules and ultimately projecting to actuator modules. In addition to single unit spike trains an EEG-like signal can be recorded from each brain area. Each area is implemented by means of a neural simulator agent. Each agent is aimed to include 10,000 spiking neurons, with each neuron receiving an average of 300 synaptic-inputs. Neurons and synapses may exhibit complex dynamics characterized by first order kinetics and may use combinations of arithmetic and logic functions. The design of the model is initially planned at implementing up to 64 agents in the global network of agents.

A.3.1 The neural network simulator

The model is described elsewhere in more details [74, 72]. Briefly, the neural network of each module is laid on a 2D lattice of neurons. At early developmental stages neural cells are differentiated into two types, excitatory (exc) and inhibitory (inh) neurons. Further differentiation mechanisms lead to the identification of input and output projecting neurons in each module. For the sake of simplicity we can consider input and output neurons forming an efferent and the afferent layers. Each such layer consists of approximately 10% of the total number of excitatory neurons. Initial connections between the populations of cells are driven by synaptogenesis process and are randomly generated according to a 2D Gaussian density function. In order to match the current Ubichip design [138] the actual modules are based on a 20×20 network which corresponds, at a mature stage of development, to 4299 ± 37 *exc-exc* (average±SEM), 1070 ± 15 *exc-inh*, 3918 ± 52 *inh-exc* and 961 ± 33 *inh-inh* connections.

Both types of neurons in the network are simulated by leaky integrate-and-fire neuromimes, with different tuning parameters. At each time step, the value of the membrane potential of the i-th cell $V(t)$ is calculated such that $V(t+1) = V_{\text{rest}} + \sum_j w_j(t) + Bi(t) + (1 - Si(t))((Vi(t) - V_{\text{rest}})k_{\text{mem}})$, where $w_j(t)$ is the synaptic weight from j-th to i-th neuron, V_{rest} corresponds to the value of the resting potential for the neuron, $B(t)$ is the background activity arriving to the i-th neuron, $S(t)$ is a binary state function of the i-th neuron, and k_{mem} is a membrane kinetic constant. The post-synaptic potential is implemented as a function of the relative timing of pre- and post-synaptic spikes [118, 81], that is usually referred as spike-timing dependent plasticity (STDP). In case of synaptic depression we consider the possibility of decaying synaptic strength to zero, thus triggering synaptic pruning and ultimately cell pruning processes [73]. Fig. A.1 illustrates the characteristic stages of early phases of each module history.

The simulator is written using the Java programming language. In addition to the proper description of the neural network it includes special routines aimed to transmit data from/to other spiking neural networks. The simulator is configured through the

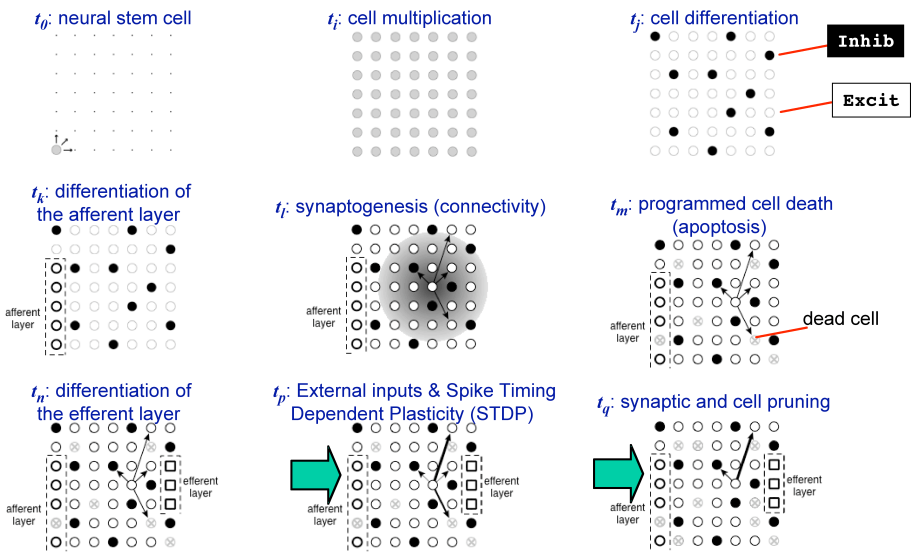


Figure A.1: Early developmental phases in the life of a neural network module. Time flows from left to right and from upper to lower panels. Grey dots represent neural stem cells, white dots excitatory neurons, black dots inhibitory neurons, crossed dots dead cells, and squares correspond to the excitatory cells that differentiated into output projecting neurons of the efferent layer.

"genome", which is read from the genome distributor agent in the network or from the configuration file in case of a stand-alone simulation.

A.3.2 Modules distribution

We used an IP-adressing scheme to broaden the range of supported hardware platforms, reduce development time and increase overall package performance. We use the Java Agent DEvelopment [17] framework in order to work with an high-level abstract environment while developing the distributed multi-module system. The JADE platform simplifies implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications [36] and through a set of tools that support debugging and deployment phases. Thanks to its design the agent platform can be distributed across machines, which not even need to share the same OS. Like the simulator, the JADE library is fully implemented in Java, providing cross-platforms integration and allows us to focus on model development rather than on low-level system programming.

All package's modules, including the neural network simulator, are packaged in JADE network agents. Then, the JADE framework is used by the hierarchical neural network stimulator routines to build up the topology of simulator agents. All inter-modules communication and data-processing are considered as message exchanges ruled by several protocols. JUbiNet reads all incoming messages from JADE message queue in a sequential manner and processes them by appropriate handlers. The processing handler sends back a new status message to the input queue as a result of the change. We use separated execution flows, in terms of processing threads, for the neural network simulator itself and for the hierarchically network logic. This allows to process network communications and simulation in parallel.

Inside the packages the execution flow is implemented as a sequential automata. Thread switching operations are reduced in order to increase computationally efficiency. Software testing and development is simplified by sequential data-processing routines

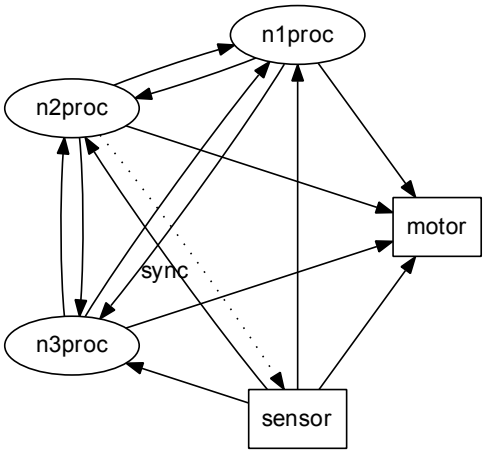


Figure A.2: Sample network topology with 1 sensory module (sensor), 3 processing modules (n1proc, n2proc, n3proc) and 1 actuator module (motor). Data-flows between agents and their directions are depicted by arrows. The dotted arrow (sync) refers to the synchronization link of the sensory module.

that provide higher system stability and predictability. In accordance to this concept the network monitoring, the link manager and the input/output mapping routines are implemented as subset automates. This implementation gives us the possibility to consider a dynamic network of neural modules where some modules may enter or leave the simulation at any time. The dynamic rearrangement of the topology is handled without need of restarting all modules thanks to the the network monitoring system. This system sends notification messages to the link monitor handler about agents joined or left network and about their respective role (i.e., either sensory, actuator or processing).

Inter-agent data-processing

Sensory modules are implemented by *input agents* characterized by afferences originating from sources other than other neural network simulator agents. External stimuli could be either predefined artificial stimuli or input data generated by external sensors like camera, radar, microphones, etc. In the current study we used input agents with predefined artificial spatio-temporal stimuli in order to simplify the test cycle of neural modeling and software development. Input agents have no restrictions concerning their target agents. Agents that project their activity to recipients other than the neural network simulator agents (e.g., external actuators) are called *output agents*. Output agents have no explicit limitations to their input agents. The modules that can be connected to any other neural network simulator agent are implemented by *processing agents*. An all-to-all link manager is used to establish as many inter-modules connections as allowed by each module's role. An other link manager was implemented to deal with pre-defined inter-modules topologies of special interest.

Inter-modules synchronization

The input/output data-processing routines used in the simulator are synchronized on a same clock cycle. The simulation actually begins only after all links are established. The execution of each next time step of the simulation is performed only after data reception is acknowledged from all known links. This method ensures simulation to run always in synchronization, but each topology modification provokes a pause in the simulation. In a static topology or in an environment with a slow rate of changes (as it

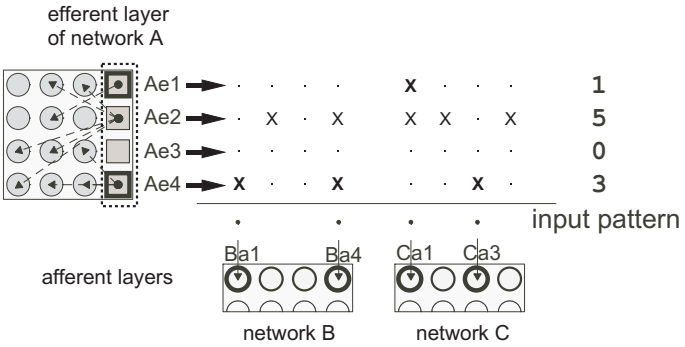


Figure A.3: Activity mapping scheme for network A connected to networks B and C. The crosses indicate the checkerboard of the active connections between network A and networks B and C. In case cells $Ae1$ and $Ae4$ are spiking, the resulting output pattern of activity is determined by the combination of the appropriate connectivity maps and provokes the excitation of cells $Ba1$, $Ba4$, $Ca1$ and $Ca3$.

happens most of the time) this implementation provides efficiency and consistency, but in a highly dynamic environment it slows the overall execution due to the large time spent in waiting status.

Input/output activity mapping

The projecting pattern of an efferent neuron towards the other agents is a copy of its intra-module projecting pattern. Let us consider the example illustrated by Fig. A.3. Four efferent neurons of network A, labelled $Ae1$, $Ae2$, $Ae3$ and $Ae4$, project to 1, 5, 0 and 3 cells within network A itself. This means neuron $Ae1$ will also project to 1 neuron among all possible afferent neurons of the target modules, $Ae2$ to 5 neurons among all possible afferent neurons of the target modules, and so on for all the other afferent neurons. A connectivity pattern is established based on a probabilistic basis defined by the number of potential target neurons. In this example the count of potential target neurons (i.e., the neurons belonging to the afferent layers of the target modules) is equal to 8.
This means that in case of neuron $Ae1$, each target neuron has an equiprobable chance to be connected equal to $\frac{1}{8}$. In this example the target of $Ae1$ is actually neuron $Ca1$. And so on for all other neurons. In case of a discharge pattern corresponding to cells $Ae1$ and $Ae4$ simultaneously activated the afferent neurons $Ba1$, $Ba4$, $Ca1$ and $Ca3$ would receive a postsynaptic potential. Notice that the connectivity pattern is reshuffled when topology is changed and probabilities of connection are modified.

A.3.3 Recording of module activity

Besides the available routines to extract spike train activity into multivariate time series, we have developed a new package designed to record the activity of a set of neurons as a function of their membrane potentials and the distance to the electrode tip. by means of virtual macro electrodes. The two main parameters of the electrode are its coordinate position $C = (x, y)$ specified by the coordinates on the neural network lattice and its *sensibility function*.

The model assumes that the electrode could be placed exactly over one neuron or exactly in-between four neighbour neurons. The sensibility function calculates the mag-

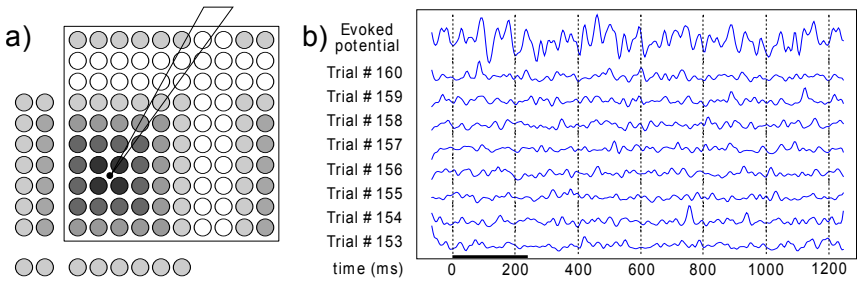


Figure A.4: a) Schematic electrode's sensitivity area on the square neural network lattice in wrapped and unwrapped representation. Each circle represents a neuron. Intensity of black color corresponds to the intensity of the electric field recorded by the electrode. b) Local field potentials (arbitrary scale) evoked by a stimulus lasting 250 ms starting on time zero, recorded from a processing ubidule. The top trace shows the averaged signal over 8 consecutive trials. The lower traces show the individual trials.

nitude of the "electric field" generated by the neural cells at a certain distance from the electrode tip. This function is meant to combine the electrical characteristics of the electrode (mainly its impedence) and the volume conduction properties of the underlying tissue. In the simplest case it is a linear decay function, but other user-defined functions can be selected. We assume an isotropic neuropile such that for any sensibility function all neurons which are located at the same distance from the electrode tip form an equipotential layer L, thus contributing equally to the recorded signal (Fig. A.4a). The electrode radius R is the total number of equipotential layers generating a recordable signal.

The equation $E_r(k) = \sum_{t=\tau\times k}^{\tau\times(k+1)-1} \sum_{r=1}^{R} \varphi(r) \sum_{\forall i \in L(r)} \Psi_i(t)$, calculates the electrode signal $E_r(k)$ where τ is a down-sampling parameter depending on the sampling frequency of the recording, $\varphi(r)$ is the sensibility function of the electrode, $L(r)$ is the set of all contributing neurons lying at distance r from the electrode tip located at C and forming the equipotential layer L, $\Psi_i(t)$ is an electric field function (*e.g.*, $\Psi_i(t) = |B_i(t)| + \sum_j |w_{ji}(t)|$ or $\Psi_i(t) = V_i(t)$) depending on the model to be selected, $B_i(t)$ is the background activity afferent to the i^{th} neuron, and $w_{ji}(t)$ are the post-synaptic potentials of the j^{th} neurons projecting to the i^{th} neuron. Notice that the raw signal recorded by such virtual macro-electrode is called *electro-chipogram* (EChG) and is monopolar. Fig. A.4b shows an example of such recordings, during a stimulus-driven task, with an Evoked Potential obtained by averaging a few consecutive trials. The current paper is not aimed to discuss the results of such recordings, which are now analyzed and will be extensively reported in future papers. A common reference signal generated by the spiking activity of all neurons of all modules is also recorded such to allow the generation of bipolar signals, akin of biologically recorded signals, for further analysis in a standard data format used for EEG recordings [88].

A.4 Discussion

We have presented a novel framework that allows the study of the activity of distributed neural networks organized in distributed interacting modules by means of virtual electrodes that record electrochipograms in each module. This framework offers a tools to study neural network interactions, complex signal processing and to compare EChG with real local field potentials and EEG recorded in experimental conditions. The current

implementation of an evolving spiking neuronal model is certainly an utmost oversimplification of the reality, but the current framework opens the way to models that will embed increasingly higher biologically inspired parameters. In a separate paper we will report the first analyses of evoked EChG in a network of ubidules undergoing classical paradigms such as the odd-ball or stimulus-compatibilty tasks.

A particular feature of our approach is the possibility to enable a highly dynamic environment characterized by evolvable topologies with modules that can enter or exit the simulation at any time. The overall design is based on a highly organized system of agents messaging operated by automata, thus allowing dynamic topology "rewiring" following simple rules. The drawback is that fast changing topologies might introduce delays of information processing. This problem could be managed by implementing a separate signal recorder agent, which would receive EChG data from all simulation agents via the network. Then, the recordings could be synchronized. This solution would in turn require an increased network communication bandwidth and the appearance of a kind of centralized authority, but offer reduced agent's file-system loads and an improvement of agent's performance.

This new framework fits well the requirements of sophisticated control circuits for robotic implementations in collective behavioral studies where each robot is driven by one or more neural networks (*e.g.* [45]). Mobile version of JADE compatible framework [17] could be used in order to obtain a platform with reduced footprint and compatibility with mobile Java environments.

Acknowledgments

The authors ackowledge the support by the European Union FP6 grant #034632 (PERPLEXUS) and the contributions of J. Iglesias for the simulator core and of O. Brousse, Th. Gil, G. Sassatelli and F. Grize for the JADE integration.

Chapter 4

Hierarchical neural systems

> The inevitable result of improved and enlarged communications between different levels in a hierarchy is a vastly increased area of misunderstanding.
>
> Laws of communication

Résumé :

Ce chapitre ainsi que l'article qui le précède décrivent le modèle de réseau de neurones hiérarchique inspiré par les voies de traitement de l'information dans cerveau. Dans ce travail, nous avons retenus quatre topologies principales de circuits neuronaux. Pour toutes les analyses, les topologies des circuits étaient composées de six modules neuronaux pouvant représenter trois traitements distinctifs : sensoriel, cognitif et moteur. Seul le module sensoriel neuronal recevait un stimulus artificiel de l'extérieur. Les quatre modules traitement cognitif ont des connexions qui ont des connexions réciproques et des connexions avec le module sensoriel. La première paire d'entre eux recevait des inputs du module sensoriel, formant la première couche de traitement du circuit. Tandis que la deuxième paire était reliée uniquement avec les autres modules de traitement cognitif, formant ainsi la deuxième couche de traitement. La différence principale entre les topologiques était dans la présence de projections réciproques à *l'intérieur* d'une couche de traitement ou *entre* les deux couches de traitement.

In real brains stimulus is transduced by sensory receptors into a sequence of electric impulses (spikes) which are sent to a group of specialized cells having, depending of the sensory modality, either the structure of a nucleus or a sheet of interconnected excitatory and inhibitory neurons for further processing. Like in real brains, our model of the distributed neural system is built on the three principal specialization of underlying neural networks. In the simulated systems the *Sensory* module starts processing of sensory information received from external (hardware) source or from artificial data-set and transmits the resulting activity by a specific group of efferent cells (see Section 3.8), to higher level *Processing* neural

networks. The neural network having a *Sensory* role receives only an external information and then it may send its efferent activity to any other node of the system. The final outcome of signal processing is *Motoric* output, which may actually correspond to a real action towards the external world or to an internal action (*e.g.*, memorization). *Motoric* nodes acts on actuators, *i.e.*, they process informations from nodes other than sensory, but do not project back to the system.

Processing of sensory information is performed along chains of parallel and interconnected areas where stimulus features are extracted and associated into relevant representations following auto-associative and reinforcement learning rules. In an oversimplified view, we can compare processing modules to cortical association areas. An illustration of these principles could be seen on Figure 4.1. Here, modeled neural modules of a given topology put into correspondence with the real cortical areas. The *Sensory* module of the system corresponds to the occipital lobe, which is the visual processing center of the mammalian brain containing most of the anatomical region of the visual cortex. The first *processing* layer is set to be in the correspondence with the parietal lobe, which integrates sensory information from different modalities to make possible spatial sense and navigation. We assume that second *processing* layer corresponds to the frontal lobe, which is associated with reward, attention, long-term memory, planning, and drive. And finally, the *Motoric* module corresponds to the Cerebellum, which plays an important role in motor control [57].

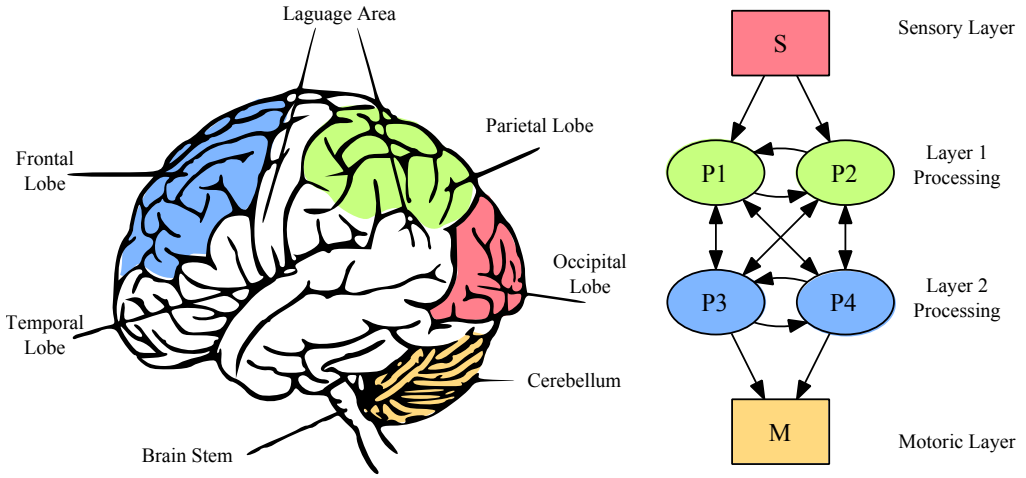


Figure 4.1: A sample brain map with hierarchical neural system graph projected on it. Virtual and real neural sub-systems with similar functions are shown in the same color. The *Sensory* module of the system with the Occipital Lobe are shown in red. The *Processing* layers 1 and 2 with Parietal and Frontal Lobes are shown in green and blue accordingly. And the *Motoric* module with Cerebellum are shown in yellow.

4.1 Input/Output activity mapping

According to the model a set of stand-alone neural networks could be combined into inter-connected neural circuit. During the early developmental stage affer-

ent and efferent layers of cells are created, as it was described in the Section 3.8. Then inter-network connections (input/output inter-network activity mapping) are established in a way that the number of external projections from a particular efferent neuron is proportional to the number of internal projections of the same neuron at the early developmental stage. In the case, when a pair of neural networks is considered, we will call an *upstream* network that, which is projecting its activity with regard to another one, and we will call a *downstream* network that, which is receiving an activity from another one. In the same way we can define *upstream* and *downstream* networks in multi-modular circuits, with the only difference that here certain networks could play both roles at the same time depending on the pair we are looking at.

The scheme is described in details in the Article A: "Dynamically organized neural networks", Section A.3.2, it supposes an input-output mapping rule with a 1-to-1 efferent-to-afferent proportion, which reflects ratio between amount of projections in efferent layer of upstream networks and afferent layer of downstream networks. This rule fits well to small networks (*i.e.* 20×20) we are started from. However, in larger networks (*i.e.* 75×75) used in latest experiments, the 1-to-1 rule produces constant stimulation of *all* afferent cells, which is equivalent to the absence of the stimulus. Thus we used a 5-to-1 proportion, when for each 5 internal projections 1 external projection is created. The number of external projections remain fixed during whole simulation flow, even despite that the number of internal projections will inevitably decrease due to projection or cell death processes.

In our simulator processing of spike train activity always done in a synchronous way, which means that at a time-step t of a simulation's time, all neural networks receive an input from their appropriate *upstream* network (which is the output of the *upstream* networks at same time-step t), then the time-step t is modeled and the output is sent to the appropriate downstream networks of next processing layer if it is present. To make this principle works, we suppose that a) every neural module of every simulation starts with absolutely no input activity at time $t_o = 0$ and b) *Motoric* module project a fictive synchronization link to *Sensory* module that creates a loop in the topology, which prevents asynchronous information processing.

4.2 Neural topologies

A layout pattern of inter-connections of circuit's neural modules is called topology, it plays very important role in information processing. From connectivity point of view, researchers usually distinguish three main categories of topologies: *regular*, *irregular* and *random* structures. Regular topologies are characterized by constant connectivity for all nodes, by a high clustering or by a long characteristic path of the appropriate graph. The most common regular topologies are: n-dimensional lattice, ring, and torus. Another example, which is neither of mentioned above two types, but still a regular topology, could be seen on the Figure 4.1. Because of their regular structure, systems of that type can be easily described by mathematical models and are easier to model and to study than other types. Random network structures, as ensued from their name, have random or stochastic connectivity between the nodes. A characteristic property of such networks is small average

path lengths between two random nodes of the network, which leads to fast, but not assured information transmission.

Neuro-scientific studies have been shown that gray matter networks in healthy volunteers have small-world topology with relatively low wiring costs. That is consistent with prior evidence that nervous systems are organized to nearly minimize wiring costs [15].

Irregular network structures (as on a Figure 4.2) are most common in the real life, they are characterized by features present in both regular structures and random structures, *i.e.* high clustering and small average path lengths accordingly [143]. Those are observed in large quantities in biology, sociology, economy, and as well, as among human created networks, like telecommunication or transport networks. Small average path with high clustering make possible fast information processing/transmission, so this is a good option if by whatever reason all-to-all type of connectivity is not possible or is not desirable in the system.

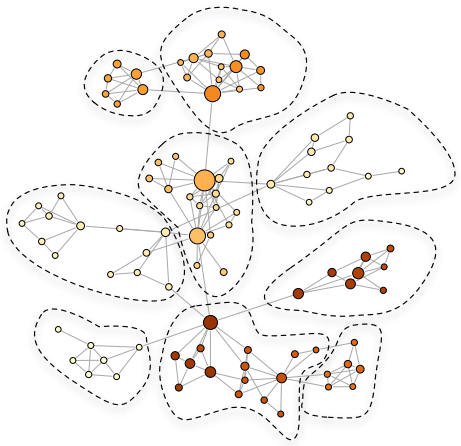


Figure 4.2: An example of the irregular topology

From another point of view, topologies also can be classified as either dynamic ones, if they are supposed to change during their "life-time" or as static ones, if they stay fixed all the way long. Our hierarchical neural network simulation framework supports both topological types. The first one was inspired by robotic application of the PERPLEXUS project, where robots should navigate in a room in order to solve goods transportation task, while avoiding collisions with other robots and with obstacles present in the room. Although, this scenario was really complex to analyze from the neuro-physiological point of view, we will give here a brief description of the dynamic features of the simulator. Then we will focus on static topologies (Section 4.2.2) inspired by the biological systems, with which we worked a lot during the development of the Thesis.

4.2.1 Dynamic topologies

When a "community" or a number of "communities" of robots is working on a set of tasks, then in order to complete these tasks in an efficient way, robots should cooperate with each other and transmit to the other members of a community

useful information, which they obtained through their own experience. It is natural to assume that robots inevitably will leave the community due to energy refill needs, tasks of higher priority, other system tasks, technical problems or whatever. By the same reason from time to time new robots will join the community working on a task, so all of them must be able to handle events of arrival of a new member or quit of another member. A sequence of actions required to solve the given task could be, at least hypothetically, produced by an Artificial Neural Network (ANN), but the mentioned above events should be handled by a higher level meta-system.

Hierarchical neural network support in the framework is done on a completely different level from the one of the neural simulation. The task may seem simple from the first view, but it is not actually the case, because of asynchronous nature and unpredictable relative timings of events. To make things work, we used event serialization approach, which consider all events as messages and put them in a processing queue. This helps to avoid arrival of multiple events at the same time and makes system's state change an atomic-event, which prohibits unexpected system's behavior. Without deepen into the low-level technical details, we will coarsely describe here an algorithm of the developed hierarchical sub-system of the simulator, which is schematically shown on a Figure 4.3. Then we will illustrate it with a dynamical topology example used at the final demonstration of the PERPLEXUS project's results.

In a few words a neural network agent's behavior could be described as follows: request all others networks for a connection, collect positive responses until topology's constraints on connections' type and number are fulfilled, notify others that your constraints are fulfilled (in that way processing could be started quasi-simultaneously), collect the same notification from all connected network agents and when that is done – start simulation processing.

In greater details the scheme is as follows. Let we have a system S of neural modules assigned to a task. They can be already working on it (processing it) or they can be still waiting for more Neural Network Agents (NAs) to start working. At this point a new agent A join the system. At a particular time neural network agent A could be found only in three different states: *unconnected* – it does not have all mandatory inter-agent connections, *connected* – it has all mandatory connections, but it waits for all other agents to switch into the same state, and finally in a *processing* state, when the simulation is started. While in the case of simulation task *processing* will mean spiking neural network (Chapter 3) modeling, in the case of robotic application it could be something else. From the very beginning A is in *unconnected* state, it has a list of all NAs in S such that it can and will send requests for link establishment to all known agents of the S. Some of them will send a positive response to the A including a notification of its-own current state, while other may not. A, in its turn, will accept all positive responses until a constraint on the connections number is fulfilled. It will remember the state of the agent the response is received from. Then it will switch to the *connected* state emitting a notification of state change to all networks it is connected to. If the constraint will not be fulfilled within a certain time period, the agent will resend link establishment request to all known agents of the S, this part is not presented on the Figure 4.3 to void its over-bloating.

When A is in the connected state it will start to collect state change notifications from network agents it is connected to. And when all agents, A is linked

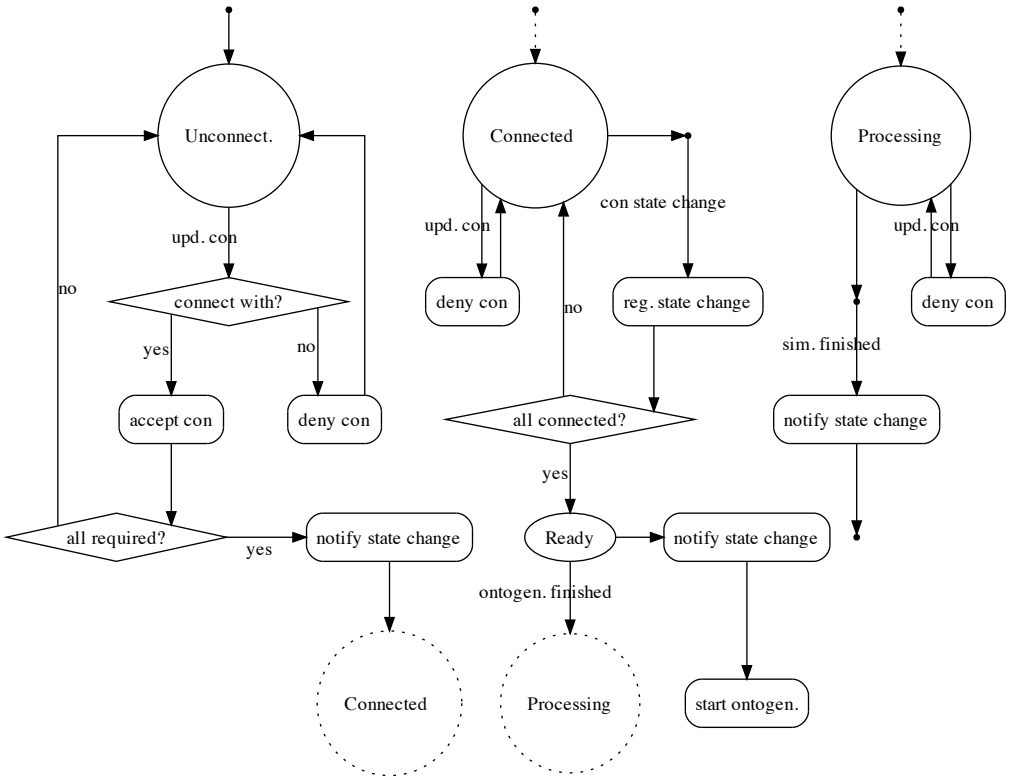


Figure 4.3: A simplified algorithm flow used by a neural network, which is a part of a dynamic hierarchical neural system. Module's states are represented by circles, actions – by rectangles, conditions – by diamonds, events – by labeled arrows, and unconditional actions are depicted by unlabeled arrows. Dotted circles are copies of particular states. Their only purpose to make figure comprehensible and fit well to the page.

with, confirm their connected state, A will switch to a *ready* sub-state, which will start ontogenesis of the neural network modeled. Having ontogenesis finished (it is not a long process in comparison with the simulation time, but it can be long enough in comparison with the connection establishment time, so it is advisable to start it earlier) it will switch to the *processing* state and will notify others about new state change. Normally in the processing state only data transmission activity is maintained up to the moment, when whole duration of the neural system's life time, given by the system's genome, is simulated. Then the agent A will proceed with a shutdown procedure, to quit S in a correct way.

In the some cases, an agent would like to break a connection or connections, like in the event of shutdown, to do this it will send a link-break notification to all network agents concerned. Agent receiving such notification, no matter in which state it is, will remove the link and will switch back to the unconnected state, sending a notification of state change to all agents concerned and pausing its Spiking Neural Network (SNN) simulation flow until a constraint on connections will not be fulfilled again (this part of algorithm is not presented on the figure). Then the system will continue the simulation from the ready sub-state, as in the normal case.

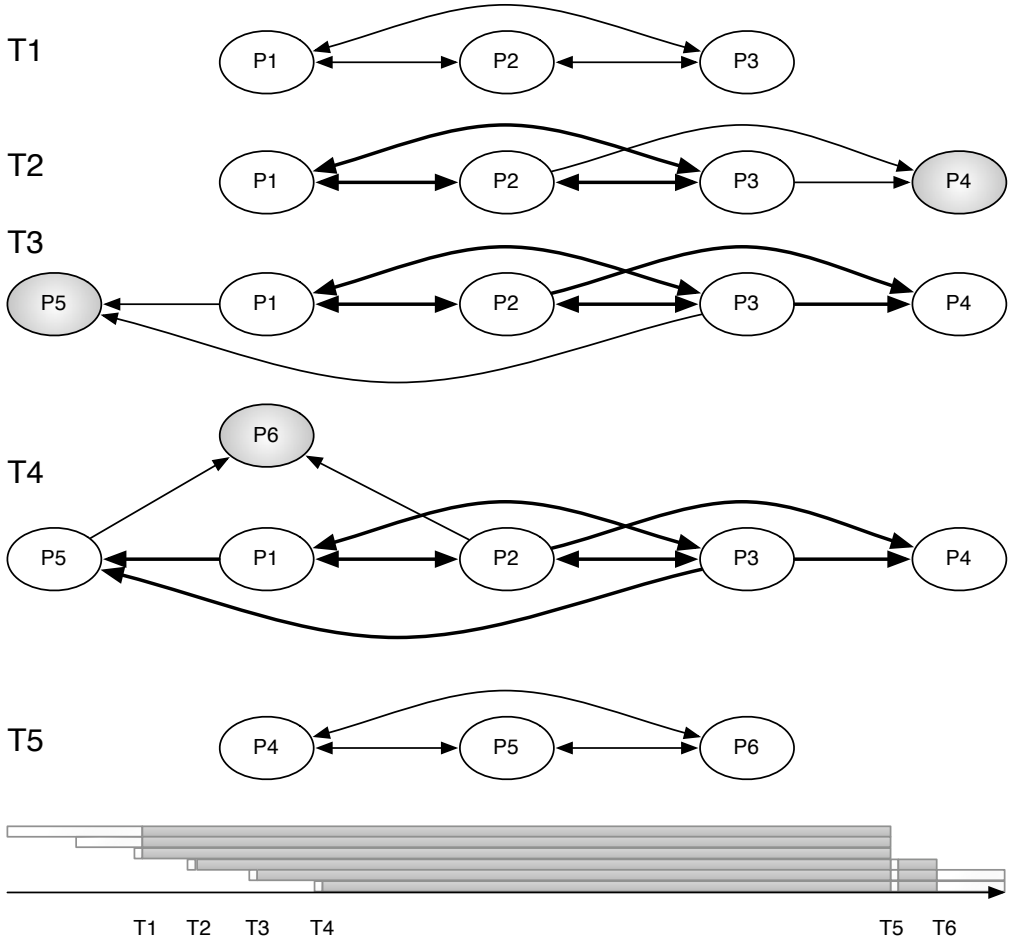


Figure 4.4: Topology evolution in a dynamic system. Evolution of the topology having 6 agents P1, P2, ... P6. At 6 distinct time-moments T_1, T_2, ... T_6 each one associated with the time moment when agent joins or leaves the system. Upper panel depicts topology changes for each time-moment. And lower panel depicts the simulation flow. One bar corresponds to one agent P1, P2 ... P6 accordingly, from top to bottom. White parts of the bars correspond to the time spend in non-processing states and gray parts of the bars correspond to the time spend in the *processing* state.

Brief demonstration of the algorithm is shown on the following example and it is also depicted on a Figure 4.4. The simulated neural system consists of 6 NAs (named P1, P2, ... P6), they all are of processing type and thus every NA can generate output spiking activity and can accept incoming spiking activity from others. Every NA has a mandatory requirement to have exactly 2 upstream networks connected, which means that each of them will have connections from exactly 2 other agents. Every agent can project its activity to any number of other agents, so there are no limitations on that side. All six NAs are added to the system one by one with a delay ΔT in between. At the very beginning, when the first (P1) and the second (P2) neural modules are added to the system, they cannot start processing up to the moment when P3 is added, lets call that time-moment – T_1. At that time every agent fulfilled its requirement to have 2

connections, so they switch to the connected state and then to the processing state, starting the simulation. As time goes by, next agents (P4, P5, P6) are added to the simulation (time moments T_2, T_3, and T_4 respectively) and they are randomly connected to already present and processing NAs P1, P2, and P3. Each NA has limited simulation time, so at the moment T_5 the agents P1, P2 and P3, which all started processing together, will leave the simulation. In consequence, the agents P4, P5, and P6 lost their connections from P1, P2, and P3 and are forced to re-establish new ones. Fortunately, there is one possible connection-map which fulfill all requirements. After a small delay from T_5, needed for connectivity reconfiguration, they create a new neural system with a new topology. Later, P4 also finishes its simulation and quits the system (time moment T_6), breaking the topology and leaving nodes P5 and P6 to wait, in the unconnected state, for new arrivals (at this time there will not be any) to continue their unfinished simulation.

Although our simulator supports dynamic features, studies in the area of the complex dynamic topologies should be based on very precise knowledge of behavior of the SNN agents of the network. In order to proceed with that task one should first study an emerging behavior of a neural system in stable conditions. We decided to leave SNN application to the robotics for further study, given limited knowledge of the processes ongoing in simulated neural circuits and a huge work should be done to achieve a goal of creating of a learning neural circuits capable to solve these tasks.

4.2.2 Static topologies

Neural systems with topological structure fixed through entire simulation are much easier to organize and study, than their dynamic counterparts. As it was mentioned before, an assembly of neural networks organized in a static hierarchical topology could be considered from neuro-physiological point of view as cortical association areas (see Figure 4.1). The developed framework supports a number of static topologies of interest which were used in the different our studies during the Thesis' development. Among of them there are 2 principal hierarchical topologies used most often in the studies (each of them having an optional small, but important, variation) and a couple of other interesting topologies, which were not extensively studied because of their complexity or on the contrary their simplicity, but rather were taken for few simulations to check selected features of the framework. Here, we will start from a detailed description of the principal topological types and then we will give a brief description of the second – auxiliary ones.

Two principle topologies were inspired by information processing chains in the brain. Minimal circuit of that topology contains 4 basic layers: a sensory layer, 2 layers of processing elements (so we are able to study signal propagation), and a motoric layer (see Figure 4.5). A *Sensory* module corresponds to a visual processing center of the brain, is connected to first two *processing* modules forming the first processing layer, which is set to be in the correspondence with the parietal lobe. Then in its turn, the first processing layer connects to the next modules correspond to the frontal lobe (associated with long-term memory, planning, and drive). And finally, all processing modules are connected to the *Motoric* module (for the sake of simplification of topology representation on the figures, which are not covering motoric module behavior, are drawn with connection only from the second process-

ing layer even despite that connections from the first processing layer are present), which correspond to the motoric control center of the brains. Depending of the presence of reciprocal connections between first and second processing layers, we will recognize pure feed-forward (FF) topology in the case of the absence of such connections and in the case of the presence – pure feed-backward (FB) topology. Two important variations of these topologies differ by the presence of intra-layer reciprocal connections (horizontal connections), they are named feed-forward with horizonal reciprocal connections (FFH) and feed-backward with horizonal reciprocal connections (FBH) for FF and FB types accordingly. All four topologies are depicted on the Figure 4.5, the topologies without horizontal connections are on the left of the figure and those with horizontal reciprocal connections are on the right. These four topologies in conjunction can give an insight on importance of reciprocal connections and better understanding of information processing in brains (as described in the Section 7.2).

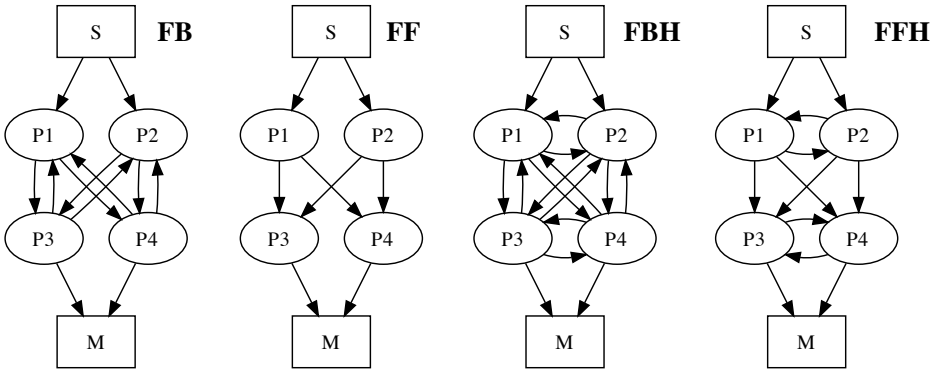


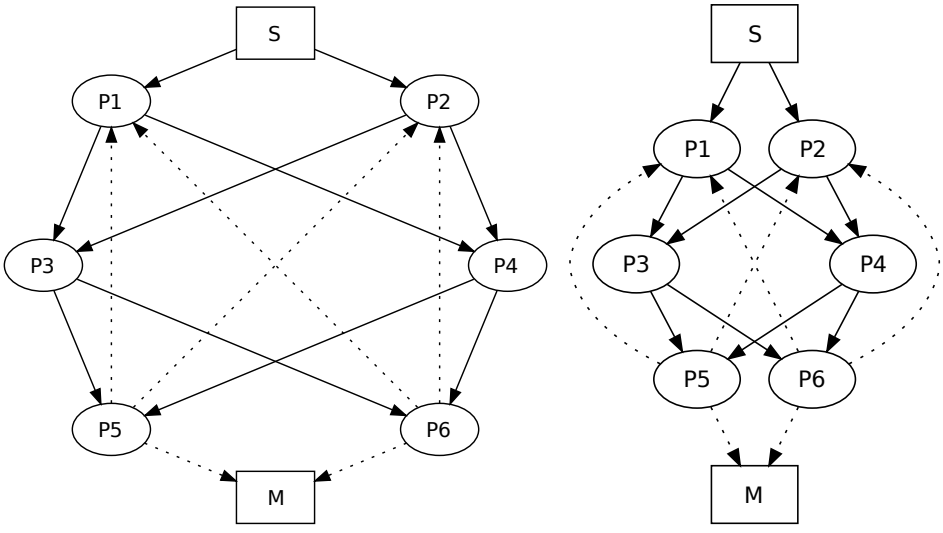
Figure 4.5: Four principal static topologies. From left to right: pure feed-backward topology, pure feed-forward topology, feed-backward topology with horizontal reciprocal links, and feed-forward topology with horizontal links.

Available auxiliary topologies are "coupled networks", an "all-to-all", a "fixed-connection number", and a set of "circular" topologies. Coupled networks is a simple chain topology – simplest neural system where an output of upstream network is connected to (and only to) an input of the downstream network, this is useful to study an effect of a single connection between the networks, as it was done in the [74]. An "all-to-all" topology is created when each module of a system is connected to all other modules, this is useful for verification of behavior of distributed circuits. The topology following this simple rule could be used to model static systems as well, as the dynamic systems, which are growing or shrinking every time new modules join or leave the system. Pure "all-to-all" systems has a limit to be constructed only from modules of processing type, as each module should be able to project and receive spiking activity and thus there is no place for a sensory module or a motoric module in the topology. Circuits of this topological type were used extensively during the PERPLEXUS project's venue.

A "fixed-connections number" topology is the one on which our principal topolo-

gies, mentioned earlier, are based. Here one can control a number of connections each neural module has. Variation of this number from one module to another one will allow to create complex topologies. Number of connections of each module of a system is controlled through its genome. The FBH neural system in an example of what can be created using this approach. Being improved by flexible technic of connection "black-listing", *i.e.* by creation of lists of agents forbidden to connect with, it allows to derive the circuits with topologies like the FFH, FF, and FB have.

Finally, circular topologies, which are also supported by the model, are natural extension of the principle 4 layered hierarchical topology described above to the class of multi-layered hierarchical regular topologies. Those ones have N processing layers, of M modules each, all modules of the layer i connected to all modules of the layer $i+1$, sensory modules is connected to the first processing layer. Last processing layer N is connected either to a motoric module or to the first processing layer. Reciprocal connection within a single layer of processing modules can be allowed or disabled, depending on the needs. On a Figure 4.6 same topology is shown in a form of circular graph and in the form of hierarchical graph. One can see, that this class of topologies is a feed-forward signal propagation one and is similar to the FF and FFH topologies, depending on that if horizontal connections are allowed or not. This class of topologies can aid in studies of signal propagation in neural systems, as it gives an opportunity to extract differences in emerging behavior between multiple processing layers.



(a) Circular representation (b) Hierarchical representation

Figure 4.6: Hierarchical and circular representation of the same topology. The topology features 3 processing layers of 2 neural modules each, connected in a feed-forward way.

4.3 Artificial stimulus

A vast majority of neural systems' topologies studied during the years of work on the Thesis were composed not only from processing modules, but also were having a sensory neural network in the structure. Which is not surprising, as we were interested not only in dynamics of the SNN model it-self, but also in the emergent behavior of the system in the different "environmental" conditions. The sensory module is the one and only one neural network in the our neural circuit supposed to receive an external (in relation to the system) stimulus.

The model supposes an application of a spatiotemporal spike pattern, which is here and after referred simply as a stimulus, to the "afferent neurons" of the sensory network. Spatiotemporal patterns for the stimulus could be passed to the simulator's framework via a set of files in Scriptable Network Graphics (SNG) format, described in details in [76]. We decided to apply a relatively simple spatiotemporal stimulus repeated at regular intervals, so neural system can produce an emergent behavior by learning the stimulus step-by-step, repetition-by-repetition. The structure and basic parameters of the artificial stimulus is described in details in the this Section.

4.3.1 Stimulus structure

The core of the stimulus is a spatiotemporal spiking pattern, which covers all afferent neurons of sensory module and during certain time period of simulation, this pattern is called Base Stimulus (BS). Temporal length of the BS is quite short: tens of simulator's time-steps depending on the simulation (most commonly used were one of 10, 16 and 32 time-steps). The BS used in the simulations was designed to stimulate each afferent neuron of the sensory network exactly one time per BS's temporal length. This was an unique constraint on the motif used, as both temporal and spatial (neurons) distributions of stimulation moments were random (see sample on the Figure 4.7). As a period of tens of milliseconds is a really short to evoke a noticeable effect in a neural network we repeat the BS several times in raw in order to expand overall length of the stimulation period to hundreds of milliseconds.

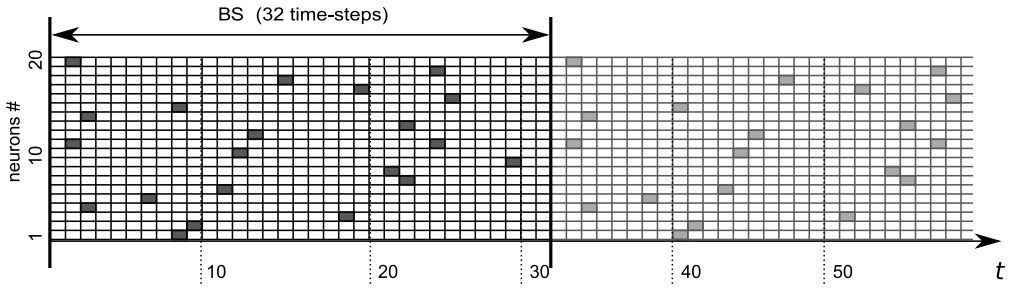


Figure 4.7: Spatiotemporal structure of the Base Stimulus (BS). Each row represents one of the 20 afferent neurons of the sensory module. Each dark block represents a stimulation impulse to a neuron. Each column represents a simulation time-step.

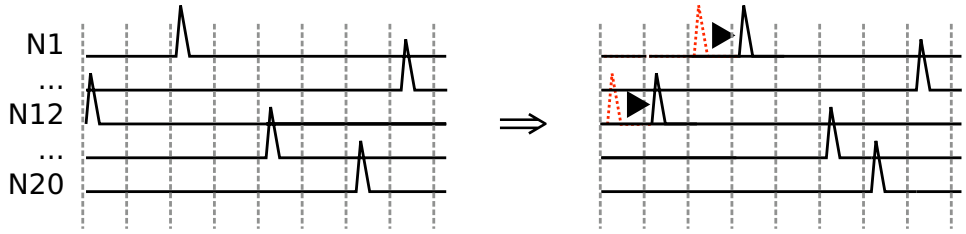


Figure 4.8: Stimulus variability procedure. In this example the impulses to the randomly selected neuron *N1* and neuron *N12* of the stimulation *motif* were shifted forward by 1 time step. Dotted spikes mark the initial position.

As a perfect repetition of events is improbable in the reality, we introduced a variability of the BS. Each repetition of the BS has a 10% variability in a neuron stimulation pattern. That means stimulation times can have a jitter of ± 1 time-step, *i.e.* a particular stimulation time of a neuron could happen 1 time-step earlier or later than it was in the original BS. A 10% variability means that at each step a number of spikes jittered was equal to the 10% of all neurons present in the afferent layer. Those 10% spikes were selected randomly and in the independent way (see Figure 4.8 for an example). Please note that described procedure can also produce an activation of a particular neuron shifted more than only for 1 time-step from its initial position in the BS. This can happen when a particular neuron will be selected more than once for an introduction of the variability and an activation time will be shifted multiple times in the same direction, thought a probability of such event is much more lower than a probability of altering different neurons each times and thus it does not have noticeable influence on the output. In the same way, with a small probability a repetition could became the initial BS, if activations of the same neurons would be occasionally shifted forward and backward same number of times. This probability also goes down very fast, as the number of neurons in the afferent layer increases.



Figure 4.9: Stimulation flow and stimulus structure. A number of repetition of the Base Stimulus followed by an Inter-Stimulus Interval constitutes a elementary stimulus "epoch".

Obviously, it is still nearly impossible to extract and generalize a stimulus-induced reaction of a neural system from its dynamics while having only one stimulus presentation even if it was of 100-500 ms. According to the general neurophysiological approach in the event-related evoked reaction studies, stimulus application was extended to the following procedure: the repetitions of the BS were followed by a silent period of certain length (according to the experiment protocol), when no external stimulus is not applied. This was followed by a new repetition set of the BS and the silent period, see Figure 4.9 for an illustration. Now stimulus was

presented to the system multiple times depending on the simulation's length and thanks to statistical analyses we have an opportunity to study stimulus-induced activity of the circuit.

Mentioned above period of absence of external artificial input is called Inter-Stimulus Interval (ISI). And the stimulation sequence formed by BS repetitions and ISI is called an elementary stimulus epoch (or epoch for short). The repetition of epochs are meant not only to build a base for statistical analysis, but also to produce self-organizing changes in the neural system corresponding to its associative memory abilities.

4.3.2 Stimulus-driven development stages

On the global time scale (whole simulation) an application of the stimulus allows to distinguish 4 major stages in the system's behavior driven by the stimulus. Let's consider that the simulation is starting at $T_0 = 0$ and is lasted up to T_{sim} marking the end of the system's life time. At the time moment T_S the first stimulus application happens and after M stimulation's epochs it ends (time moment T_E).

The 4 developmental stages are:

pre-learning stage $t < T_S;\ t \in [T_{preS}; T_{preE}]$: starts slightly after T_0 at the moment T_{preS}, when an initial transitional state of the system stabilizes to an acceptable level. It ends with a start of the stimulation – at time T_S. During this state no stimulation is performed, the system's behavior mostly conducted by the initial conditions and by stochastic process of background activity.

early-learning stage $T_s \leq t < T_E;\ t \in [T_{earlyS}; T_{earlyE}]$: starts with the first stimulus application T_S and ends somewhere after at the time moment T_{earlyE}. A stimulus is applied to the system, forcing it to enter into transient state of hyper-activity.

late-learning stage $T_S \leq t < T_E;\ T_{earlyE} < T_{lateS};\ t \in [T_{lateS}; T_{lateE}]$: starts when a system driven by the stimulus enters in to a state, which can be considered as steady, and ends when last epoch of the stimulus application periods ends – time T_E. This state is characterized by a maturing of the system, where cell and synaptic pruning reached a steady state. This is the most promising state for studies, because of the absence of huge transient components in the emergent behavior of the system.

post-learning stage $T_E \leq t < T_{sim};\ t \in [T_{postS}; T_{postE}]$: starts after the end of the stimulation T_E and lasts almost until to the end of simulation – time moment T_{postE}. No external stimulation is present, and the system is driven only by its own interal "echoes" of the stimulus.

The above mentioned scheme of system development stages is shown on the Figure 4.10.

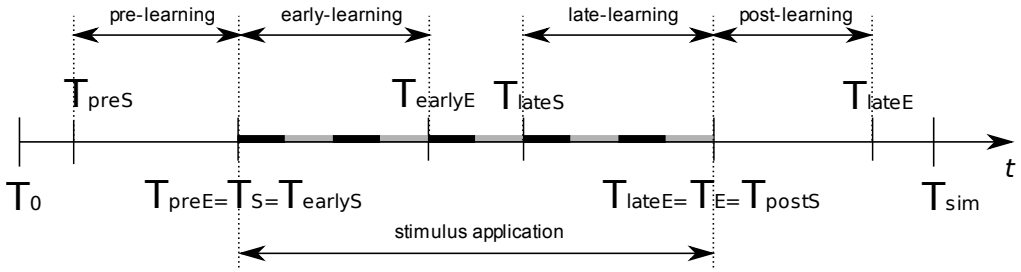


Figure 4.10: Neural system development stages. Stimulation periods are schematically represented with black (BS repetitions) and grey (ISI periods) line segments.

4.4 Bio-electrical signal model

Discovered in 1929 by Berger, electroencephalography was at first the display over time of the tiny rhythmic electrical currents captured by electrodes on the surface of the scalp. Very soon the origin of these currents has been associated with brains activity, but their precise mechanisms of generation were not completely determined. In brief, the electroencephalography (EEG) waves were interpreted as the result of summation of the more or less synchronized action potentials running in the brain [3]. Interpreting the recordings of the Local Field Potentials (LFP) of such a dense interconnected meshwork of neurons which is brain is still a difficult task.

One of the most intriguing features of brain electrical activity is the fact that the recorded signal is apparently similar at every scale from a few dozen of microns (LFP) to a dozen of centimeters (EEG). This started to be solved first with the discovery by Mandelbrot of the fractals, that is structures which are similar at every scale. The fractal nature of the brain electric activity cannot be explained by simple dipoles. However the increased knowledge in brain circuits [23, 24, 25, 132, 130, 131, 1] proved that the dipole theory was correct if the nature of the dipoles was modified and the role of the thalamo-cortical loops minimized [107].

The EEG signal is the superimposition of all overlapping dipoles underneath a given electrode. There is a clear difference in the frequency content of the signal between rest state, active state, some sleep phases and some pathologic states (epilepsy, senescence) of the brain, which can be discovered with EEG analysis.

Now when the computers are powerful enough to simulated large cell assemblies, we are trying to record EEG-like or LFP-like signals from the modeled neural circuits by the means of virtual electrodes. We call this approach an electrochipography (EChG). The virtual electrode model it-self is described in the Section A.3.2 of the Article A: "Dynamically organized neural networks" and is briefly recapitulated in the Section B.3 of the Article B: "Stimuli-driven functional connectivity". Here we will only add, that we consider the output of the electrode as a sum of a product of cells' spiking activity voltages over the electrode's coverage area and linear decaying function, which reflex currents loss due to distance traveled from a particular cell to the electrode's tip.

A big advantage of this approach is that signal analysis techniques equivalent to those applied to *real* EEG and LFP recordings in neuro-physiology can be also applied to the artificial neural system model by the means of virtual electrode's

signal.

4.5 Hierarchical simulator architecture

The simulator package is a highly expandable and flexible software framework aimed to a simulation of the distributed hierarchical biologically plausible neural systems. All components of the framework are organized in a modular way that allows inter-operability, flexibility, and expandability of the simulator in accordance with particular needs. Formed by the biologically plausible neural network simulator and the distributed hierarchical neural network framework it fully covers epigenetic and onto-genetic levels of a generic evolutionary concept. On the ontogenetic level are the routines, which are in charge of the neural system's development during early stages, in particular there are: genome decoding, neural network initialization, and inter-network connection establishment rules. On epigenetic level there is a neural network simulator with features limited to an individual neural network lifetime.

All framework's modules are implemented in Java programming language and are using the standard Internet Protocol (IP) networks, as communication layer. This allows to support a wide variety of the hardware and software platforms. Distributed hierarchical neural systems are assembled from a number of basically independent neural networks modules. Almost all their properties are configured through a genome, at the creation time, the particular gens will control how and when a set of the independent neural networks will assembly into a neural circuit and which would be its topology.

As it was mentioned before in the article, the distributed neural network framework is an event-driven system, it uses a Java Agent Development Environment (JADE) library to organize inter-neural network communications of the low-level. All events in the network are serialized at processed one by one, which facilitate event handling and improves system's predictability. All inter-NA communications and spike train activity transmissions are messages ruled by number of protocols.

The distributed hierarchical neural network framework consists from a number of software-modules, most important of them are a network discovery module, a connection establishment module, input/output activity mapping modules, and genome decoding modules. An overview of the framework's architecture could be seen on Figure 4.11. The network discovery module is responsible for the processing of system's IP network events in the order to discovery Neural Network Agent (NA) present in the physical IP network. The connection establishment module is in charge, non-surprisingly, of connection establishment (according to the algorithm described in the Section 4.2.1) and the transmission of spiking activity between NAs of a circuit. The input/output mapping modules implements spike activity transmission rules described in the Section 4.1. The genome decoding modules are designed to instantiate the neural networks according to the configuration parameters in the genome strings taken from simulation's pre-configured genomes or provided by the evolutionary level of the framework.

This evolutionary layer of the model along with network's genetics are discussed in the next Chapter.

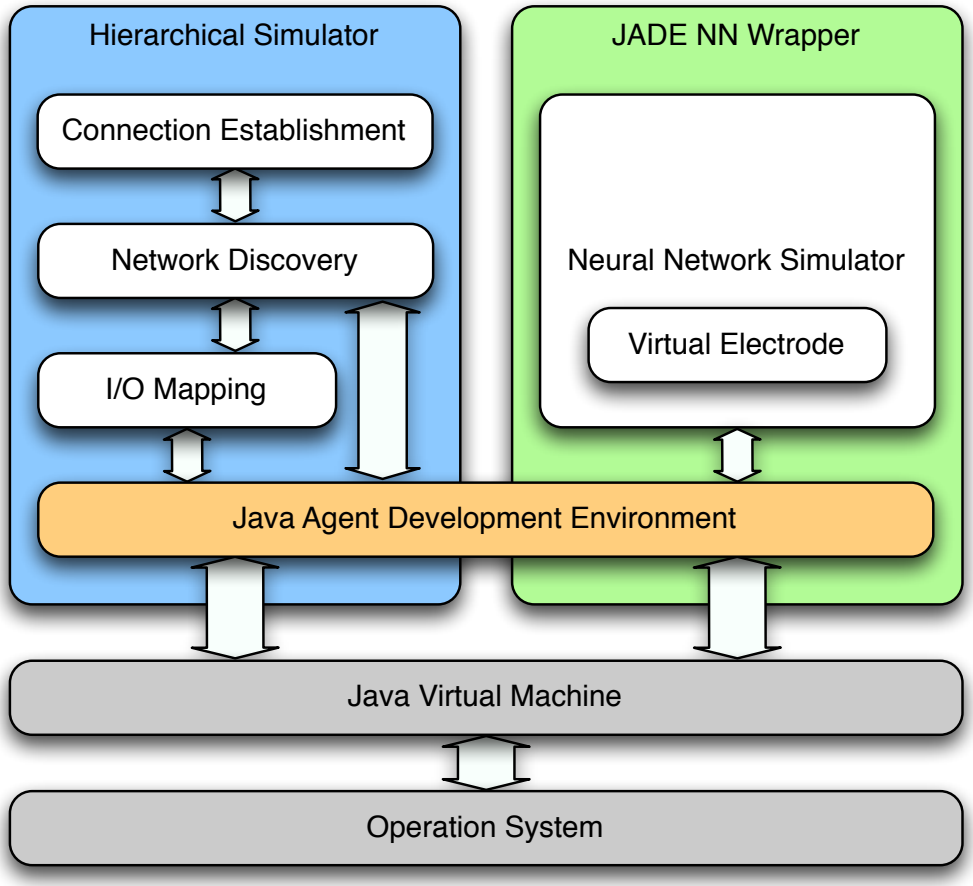


Figure 4.11: Hierarchical Simulator Architecture.

Chapter 5

Evolutionary and genetic networks

> The evolution of the brain not only overshot the needs of prehistoric man, it is the only example of evolution providing a species with an organ which it does not know how to use
>
> Arthur Koestler

Résumé :

Ce chapitre traite de la partie évolutive du simulateur. Cette partie évolutive nous permet de générer des ensembles de circuits neuronaux pour déterminer les comportements communs de l'ensemble. Quatre génomes différents correspondant chacune aux quatre topologies principales décrites ci-dessus ont été utilisés comme génomes de base et chacun d'eux a été modifié plusieurs fois par mutation aléatoire. La réplication est réalisée selon le résultat d'une fonction qui évalue le résultat de l'activité des circuits. Cette approche permet d'obtenir des ensembles de circuits différents pour chaque type de topologie. Les signaux EChG enregistrés sont moyennés afin de lisser le bruit d'activité spontanée dans le réseau.

From the pragmatic computational science point of view, despite developed so well and so far connectionist approaches [84], which include all kinds of the Artificial Neural Networks (ANNs), there are still no efficient connectionist methods capable to model complex brain functions, like adaptive learning of a large number of objects in a multi-dimensional space or dynamic learning of multiple models in a multidimensional and changing environment [10, 144] or complex problems in a dynamically changing environment, where information from different abstraction levels are properly used. Using, for example, evolutionary algorithms to train an ANN and adjust the connection weights according to a fixed data set given in advance, does not reflect the nature of learning in the brain [54]. Existing connectionist models still suffer from generic problems of computational intelligence:

the curse of dimensionality – in a large dimensional space existing feature se-

lection algorithms related to ANN models fail to select an optimal set of input variables and to modify this set optimally when new data arrives;

the curse of the local optimum – the models usually reach a local, rather than global optimum solutions;

the curse of multiple modality and multiple task learning – it remains difficult to dynamically integrate multiple models in order to discover common patterns/relationships, features, to make the models share knowledge in order to improve the learning processes.

From modeling point of view single neural network also cannot reflex all the complexity of the brain. The principle of evolvability states that a system evolves its structure and functionality through incremental learning from active interaction with the environment, thus continuously improving its performance. The evolving process is based on incremental forming of local clusters of data and developing local functions. This principle is fundamental for any brain at all functional levels.

As we shown in previous Chapters, Spiking Neural Networks (SNNs) can be used to build biologically plausible models of brains. Here, we will briefly cover an evolution of an SNN based hierarchical system from the biological and the computational points of view, then we will describe a model of the Evolving Hierarchical Neural Network (EHNN), which should able to develop its functionality by interaction with the environment in an incremental way, thanks to "natural" selection approach.

5.1 Evolutionary algorithms

In modeling evolutionary approaches mean exploitation and modeling of biological evolution for the sake of creation of intelligent systems. Evolution is considered as a process of development of biological species in order to better adapt to environment, which is accompanied by genetic change of population and by generation replacement [146]. In particular, this is the product of two opposing processes: one that constantly introduce variation in traits of species and other that make particular species become more common or rare. On one side main cause of changes in species' rarity are natural selection and disrupt changes of environment. On another side main cause of species variation are mutations and recombinations, which changes the sequence of a gene, which is responsible for these or those features.

A gene is a unit of heredity in living organisms. According to a modern working definition of a gene, it is a locatable region of genomic sequence, corresponding to a unit of inheritance, which is associated with regulatory regions, transcribed regions, and or other functional sequence regions [110]. In biology it is normally a stretch of deoxyribonucleic acid (DNA) that codes for a type of protein or for an chain ribonucleic acid (RNA) that has certain functions in the organism. Genes hold information, which allows to build and maintain an organism's cells and pass genetic traits to its offspring. The entirety of an organism's hereditary information is called genome and is described by an assembly of genes of organisms, as well, as non-coding sequences of the DNA [117].

The growth, development, and reproduction of organisms relies on cell division, or the process by which a single cell divides into several (usually two) identical daughter cells. This requires first to make a duplicate copy of every gene in the genome in a process called replication and then to make a species from the genome in a processes called transcription.

5.1.1 Traits variation

Mutations in genetic material are caused by errors occurred during transcription phase and by reshuffling of genes in sexual reproduction (recombination). These two factors are main sources of trait variations and they are tightly linked to the genome variation and to the successful development of species in the natural conditions. Though, even relatively small changes in genotype can lead to dramatic changes in phenotype, in most cases they did not produce visible effects in phenotype, partially because of robustness of gene transcription procedure [7].

Usually researchers consider two main sources of genome variation:

Mutation – mutations are random changes in the DNA sequence of a cell's genome and are caused by radiation, viruses and mutagenic chemicals, as well, as errors that occur during replication and transcription. These changes can either have no effect on specie, because mutations error are suppressed by the robustness mechanisms of genome transmission or non-favorable environment conditions or alter the product of a gene, later includes a gene non-functioning situation [146]. Although most mutations, which change protein sequences are neutral or harmful, some mutations under certain circumstances can have a positive effect on the organism. Such mutation may enable the mutant organism to withstand particular environmental stresses better than other species. In these cases a mutation will tend to become more common in a population through mechanisms of natural selection.

Recombination (or crossover) – refers to recombination between the paired chromosomes inherited from each of the parents. While in this formation, homologous sites on two chromatids can mesh with one another and may exchange genetic information. Because recombination can occur with small probability at any location along chromosome, the frequency of recombination between two locations depends on their distance. Therefore, for genes sufficiently distant on the same chromosome the amount of crossover is high enough to destroy the correlation between alleles.

These two variation sources in conjunction with natural selection and genetic drift, will assure viability of adapted species and extinction of the least adapted ones.

5.1.2 Evolutionary mechanisms

The two main mechanisms, which *produce* evolution in the terms of iterative adaptation to current environmental conditions, are natural selection and genetic drift. Natural selection is the process which "favors" species with genes that aid survival and reproduction. Genetic drift is a random change in the frequency of alleles,

caused by the random sampling of a generation's genes during reproduction. The relative importance of natural selection and genetic drift in a population varies depending on the environment conditions and the effective population size, which is the number of individuals capable of breeding [128]. Natural selection usually predominates in large populations, whereas genetic drift dominates in small populations. The dominance of genetic drift in small populations can even lead to the fixation of slightly deleterious mutations. As a result, population size can dramatically influence the course of evolution.

5.2 Genetic algorithms

In the computer science, Genetic Algorithms (GA) is an evolutionary algorithm-based methodology greatly inspired by biological evolution to find or improve models (individual solutions) that perform a user-defined task. The GA began with the evolutionary algorithms first utilized by Nils Aall Barricelli applied to evolutionary simulations [14].

Usually to perform a task lots of individual solutions are generated. From generation to generation they compete among them-selves and only the best solutions pass to the next generation. Because of this typical population contain several hundreds or *thousands* or even *millions* of possible solutions, depending on the nature of a problem. And after that for each next generation GA could require to produce a sequence of populations each them usually of the same or larger size than the first one. Initially GA was mainly used to solve relatively simple problems because it is hight computational intensity. Nowadays, with grow of accessible computational power, GA is used to solve for more complex and complex tasks.

In the genetic approach a population of genomes encoding individual solutions of an optimization problem is generated. Every individual solution is evaluated according to a fitness function, which in the biology is survivability of a specie and in the computational science it is usually a minimization criterion. Best individuals are selected from the current population (stochastically, based on their fitness evaluation result), their genome is modified (randomly mutated or recombined with other genomes) and a new population spawns evolving towards even better solutions. The new population is created on the each next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced or a satisfactory fitness level has been reached for the population. Though, a termination of the algorithm does not explicitly mean that optimal solution is reached, usually GA gives a good sub-optimal solutions in a reasonable time.

The GA is virtually applicable to any kind of problems. To be able to apply GA to a problem three mandatory components should be defined: a genetic representation of the solution space, a fitness function to evaluate individual solutions and evolutionary operators to produce the offspring. In the next section we will show how that fits to the biologically plausible hierarchical neural networks.

5.3 Evolving hierarchical model

The concept of evolvable hierarchical neural network's model is based upon three major evolutionary levels usually considered by researchers, they are: phylogenetic, ontogenetic, and epigenetic [122].

epigenetic level refers to learning capacities, which are limited to an specie's individual lifetime. The Spiking Neural Network itself naturally fits into this layer, because of synapse and cell evolution under Spike-timing-dependent plasticity (STDP) and pruning processes.

ontogenetic level describes origin and development of the system during its early stages of development. Genome decoding sequences, neural system initialization based on decoded genome and handling neural network's pre-development external inter-connections establishment rules are performed within this conceptual level. It is the level of the hierarchical meta-system.

phylogenetic level is responsible for system's development on the large time-scale, *i.e.* for feature encoding and storing in a genome, as well, as driving the system though evolutionary and genetic development processes. Selection of alternate values of the parameters (i.e., the alleles) is performed at this level associated to a computational neuro-genetic modeling. The evolvable hierarchical neural network, described in this Chapter, fits well into this level.

Final model of hierarchical evolving spiking neural network covers all these levels of the evolutionary scheme from the epigenetic level and up to the phylogenetic. Almost every feature of neural network is represented by genome, which incapsulates all relevant genes responsible for neural network shaping in a proper way: cell types and distributions, post-synaptic potentials (PSPs) levels and refraction periods, connectivity patterns, network topologies, and so forth.

From the evolutionary point of view modeling is a process, which consists from the next stages: formation of an initial population, network early development, system modeling and reproduction of the next generation of the population. Given an initial genome the neural system is populated with the neural network modules. Each parameter of the neural network is set by the genome including, but not limited by, connectivity rules applied to the circuit, as it is described in the Chapter 4. When all inter- and intra-module connections are established, system starts simulation of SNN behavior, according to the model given in the Chapter 3. A fitness function evaluates (as described later in this Section) every complete simulation (simulated circuit) of the population and basing on the result a decision whether to reproduce or not is taken for each particular circuit. If the decision to reproduce is taken, one or more replicas of this system is created. Genome of the replica is based on the one of its parent, but with random mutations. The new population is formed by a set of all replicas produced and they are used on the next iteration of the evolutionary procedure. The modeling terminates when a given number of generations has been produced.

One can see that mandatory components of the GA, mentioned in the previous section, are fulfilled in a natural way: through a genome we set up multiple parameters of a neural network model, then we evaluate this specie by modeling a neural

network and by application of a fitness function to the produced output. Having this done, we will easily select best neural system and re-spawn a new population for further improvement. However, to complete the model we should explicitly define a genome coding scheme, a mutation procedure and a fitness function.

5.3.1 Genome coding scheme

While in the biology gene is a DNA sequence, in the computational biology it is a sequence of bits, describing particular specie. This approach is used in the our model. In the our model in the absence of a level which directly models amino-acids we should also chose the level of desired abstraction of genes and genome coding scheme. The level of abstraction should determine if a gene will correspond to high or low level model abstracts. For example, it can define which features should be coded will it describe an inhibitory postsynaptic potential (i-PSP) level of a projection or a complete topology of a neural circuit. As soon, as the model does not have explicit DNA level, the coding scheme should determine, which algorithm which will be used to describe, encode and decode genes of the system. The selection of those two components will greatly affect a result of the evolutional operators, i.e. mutation and recombination.

We used an approach, which reflects the architecture of the developed neural network simulation framework. It allows to code almost any existing parameters of the model, but also that means for each particular simulation we should explicitly specify which genes will be mutated and which will not be, according to the task of particular experiment.

The coding scheme is as follows: an ordered set of modeled genes of the neural system is selected before simulation, it stays fixed through the whole simulation, each numerical model parameter to encode in the genome is converted to an appropriate bit-stream, then all bit-streams are merged together thus forming the final system's genome representation. As a set of genes of the genome is finite and ordered and length of each gene is known *a priori*, the genome's bit-chain will definitely describe circuit's parameters, so they could be encoded and decoded without a loss of information. A number of descriptors is associated with each numeric parameter coded in a genome, they are: a maximal possible parameter's value, a minimal possible change Δ, and a number of bits, which could be changed by an evolutionary operator. First two parameters define how many bits are used to describe a gene and the third defines how many of them could be changed by mutation procedure.

Let's consider an example: we want to code by a gene a i-PSP potential of a projection, which is equal to 2.125 mV for the first population, the model at whole is very sensitive to this parameter, so we do not want it to be higher than 16 mV and change Δ should be even to a $1/256$ mV, that gives us 4 bits ($16 = 2^4$) for the integer part of the value and 8 bits ($1/256 = 2^{-8}$) for the fractional part, so the final gene's bit-chain is be as follows: "0010.0010.0000". This example as well, as a few others, is summarized in a Table 5.1.

The final system's genome consists from a number of bit-chains, as the one mentioned above, each corresponding to a modeled parameter. In that way, one can encode and decode almost any set of parameters and perform experiments of interest.

Table 5.1: Genome coding and mutation example

Coded parameters:	refractory period	efferents number	i-PSP
Units:	time-steps	cells	mV
Delta:	1	4	1/256
Max:	8	1024	16
Bits total:	3 (3+0)	8 (10-2)	12 (4+8)
Value (sample):	3	400	2.125
Bitstream:	011	0010.0100	0010.0010.0000
Mutated genome:			
Mutated bits:	3	6	8
Mutated stream:	01**0**	0010.**1**100	0010.00**01**.0000
Mutated value:	2	432	2.0625

5.3.2 Mutation procedure

The mutation operator used in the model is an extension of the bit-flip operator [92]. Having a bit-stream corresponding to the genome, we can establish an α variability level of the genome. According to which each particular bit of the genome may inverse its value (changing form 0 to 1 and from 1 to 0, depending on its initial value) with a probability α/b, where b is number of bits in the bit-stream representing the genome. It is clear, that depending on the parameters coded such mutation procedure could product a lot of non-vital species. That would be a costly processes in terms of computational power, given the complexity and the time expensiveness of the simulations. In order to decrease number of non-vital species, we introduced a rule, already mentioned above, when one should select gene bits which could be affected by the mutation. That will allow to select appropriate magnitude of the genome's variability and to diminish number of lethal mutations. In the biology genes are usually not so badly affected by the mutation. This problem is addressed by the robustness of the gene's structure and the transcription mechanisms. In the our model this aspect also could be addressed by adding the robust coding schemes, thought foresee the complexity of the task it is left for the future work.

Considering the example from the Section 5.3.1: a gene corresponding to the i-PSP with an initial value of 2.125 mV, represented by a bit-chain "0010.0010.0000", have a 10 mutation-allowed bits with an $\alpha = 20\%$. Application of the mutation procedure could occasionally change 2 bits the numbers 5 and 6 (from the right), thus the gene will be mutated to a bit-chain "0010.0001.0000", which corresponds to a new value of 2.0625 mV. In the example shown in the Table 5.1 a genome "011:00100100:001000100000" representing the values of "3:400:2.125" in the same way will be mutated to the "01**0**:0010**1**100:001000**10**0000" string and the corresponding values of "2:432:2.0625".

5.3.3 Fitness function

The replication is made at the end of system' simulation and is governed by the fitness function, which should play a role of the natural selection, *i.e.* which will "favor" the neural system species with genes that aid to solve a particular task. Started from a modeling application, here we have not a strictly fixed function, because there are very vide range of possible tasks and applications, but a framework which allows to implement user defined functions. Current framework implements only two possible fitness functions: the first one – the random one "selects" a specie for a reproduction with a given probability and the second one is a threshold function of a number of active cells in the system, like that:

$$\mathcal{H}(n_{active}) = \begin{cases} 0 & : \quad n_{active} < N_{Tr} \\ 1 & : \quad n_{active} \geq N_{Tr} \end{cases} \tag{5.1}$$

Where n_{active} is a number of active cells, which is defined as the number of cells spiking in a low-frequency band, for example: $1 - 50$ Hz, and N_{Tr} is a threshold value. In such way we consider overexcited and under-excited networks as a dead ones.

5.3.4 Evolutionary framework conclusions

To conclude, an unique simulation framework is created, it allows to model and to study biologically plausible hierarchical neural circuits defined by genome and which are evolving from generation to generation with aid of mutations and a user defined fitness function. A sample application of this framework will be presented in the Section 7.2.

Now we have covered all modeling aspects of the biologically plausible SNN used in our experiments, so we will proceed to the data analysis and results part of the Thesis.

Part II

Bioelectrical Activity Analysis

Introduction to the Bioelectrical Activity Analysis part

In the previous part of the Thesis the model of the evolving hierarchical spiking neural network was described. This part is dedicated to bioelectrical data analysis approaches and to the results obtained in the experiments observing an emerging behavior of the neural model created. This part is divided into three chapters, each reflecting an important period of the development of the Thesis: the first chapter describes an application of the higher order spectral analysis to the biologically plausible electric signals recorded by the simulated circuits, the second one describes findings on an effect inter- and intra-layer reciprocal projections have on the hierarchical neural circuit's emerging activity and the last chapter describes a concept of application of the robust non-linear regression analysis methods.

Signal analysis approaches, originated from the neurophysiology, as Evoked Potential, Power Spectral Density and bispectral methods, applied to our modeled electrochipography (EChG) signals are described in the Chapter 6. Then the results obtained by these methods application to real and to simulated data are given in the Article B: "Stimuli-driven functional connectivity". We will touch briefly a neural network scaling questions and, in particular, spontaneous activity level adaptation techniques in the Section 7.1. The behavior of the larger networks ($6 \times 75 \times 75$ cells) of four topologies featuring different combinations of the reciprocal inter-module projections are summarized in the Section 7.2. Finally, a concept of application of robust non-linear regression analysis is given in the Article C: "Structural modeling robust to outliers". The article is dedicated to the novel non-linear regression algorithm developed during the flow of the Thesis and featuring automatic model search in non-linear model space and model's parameter estimation robust to the outliers in both explanatory and response variables.

Chapter 6

Bioelectrical data analysis

Résumé :

L'activité bioélectrique de chaque module a été enregistrée au moyen des électrodes virtuelles. Ces signaux ou electrochipogrammes (EChG) se caractérisent par des propriétés proches de celles obtenues avec l'encéphalographie (EEG), l'electrocorticographie (ECoG) et des potentiels des champs locaux (LFP). Les méthodes d'analyses fréquentielles et de potentiels évoqués appliquées ces signaux sont décrits dans ce chapitre.

In many ways, neuroscience is a reverse-engineering of the principles of the biology. The unknown machines are the biological systems. We dissect them to figure out how they work. Artificial model allows to dissect the systems, without doing this in the reality. As on the bottom level of the model we have neurons and projections, it is natural to start a study from that level.

This chapter describes main techniques which are applied for the analysis of simulated neural circuit's behavior. These approaches are spatiotemporal pattern analysis, event-related analysis, and spectral analyses. The first one is targeted at knowledge discovery based on exact spike trains gathered from neurons or group of neurons during a simulation or an experiment. Second one is used to generalize activity levels of emerging behavior registered by the means of EEG-like recordings and thus to deduct higher level patterns of behavior. The latest is used to discover signals' time-frequency characteristics. Let's start from the spatiotemporal pattern analysis.

6.1 Spatiotemporal patterns of activity

The rationale behind spike train analysis is to deduce principles of operation of a neural network, as a black-box, by interpolation of state of neurons based on raw spike sequences recorded from the neurons. That is, given a set of spike train signals representing the input/output or intermediate signal of a network, we deduce what the network is doing and how it is developing. Spike trains are bio-electrical signals recorded from individual neurons in the brain/model. They are essentially action potentials generated by the neurons. They are generated

by neurons in order to communicate an information to another neurons. The information is represented by series of "spike-coded" signals.

In the model two principal events could force a neuron to produce a spike. On one hand, spontaneous activity process (see Section 3.7) can provoke a neuron to fire whenever its excitation level is close to the activation threshold (see Section 3.2). On the other hand spikes also can be produced by convergence of synchronous activity (*i.e.*, temporal summation of excitatory post-synaptic potentials) of a group of neurons within the network. The information processed by a neuron is embedded into a time-series of spike train. Since all spiking potentials are essentially identical to one another (*i.e.*, spikes have the same amplitude), they represent an information used by neurons by its time-of-arrival and not by its amplitude. It is the time-of-occurrence of the spike and the frequency-of-occurrence of the spike, that encodes content/information carried by the signal.

Mathematically, spike trains belong to a class of stochastic processes called a *point process*. A point process is a natural process that is characterized by the occurrence of a point-event. A point-event is an event that could be represented by a point in time or by a point in space. A point does not occupy any finite time or space, rather it signifies the onset of an event in time or the limit of an event in space. In other words, a point is infinitely small. Usually points are used to signify the onset of events. Although spiking potentials do occupy a finite time, the time of occurrence is considered as a point because it is negligibly short. Thus, in the analysis we could treat signal content as a point process, that allows us to simplify the complex problem into elegant mathematics.

The spike trains of activity analyzed for a presence of recurrent patterns of spikes, which could be recurrent in time or in space (in between several neurons or groups of neurons). Usually this is done by complex statistical approaches as, for example, Pattern Grouping Algorithm, described in greater detail in the [135, 133, 134]. Despite that, some patterns could be easily spotted by a visual inspection of a raster plot of spike occurrences. A raster plot is a figure where each spike is depicted by a point, every line corresponds to activity over certain period of time, usually an epoch, and each next line represents next period in the life of a neuron or neurons.

For example on the Figure 6.1 on the top, on the middle, and on the bottom panels one can see three raster plots corresponding to signals recorded from three neurons in the networks. Each panel described by 250 lines of 1536 points each, each point corresponds to a 1/1024 of a second, thus every line is 1500 ms long and these raster-plots depict 375 seconds in live of three neurons. On the bottom panel one can see a neuron falling into the depressive state after approximately half of the simulation time, which is seen by lower spiking activity and in consequence fewer points on the upper part of the raster plot. On the top and middle panels one can see neurons with higher spiking activity during first 500 ms – having much more points on the left part of the rasters – this is produced by incoming external stimulus applied to the system. And also one can notice slightly higher spiking activity at around offset of 1300 ms driven by internal synchronous activity of neurons of the network.

Main advantage of the spike train recordings is spatial and temporal precision, in the same time it is somehow the weak point of these recordings, because it is difficult to find generalities in emerging patterns of behavior described

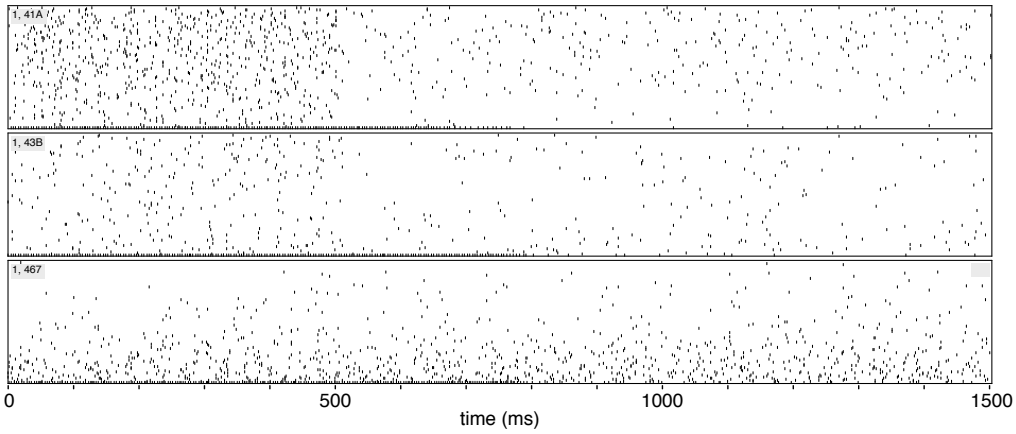


Figure 6.1: Sample spike trains captured from three neurons are shown on top, middle and bottom panels. 375 seconds in the life of three neurons are described by a raster of 250 lines of 1536 points (1500 ms) each. Each black dot corresponds to a spike produced by the neuron.

by spike trains. More general behavior trends could be obtained while working with assemblies of neurons. This is where analytical approaches based on the electroencephalography (EEG), the Local Field Potentials (LFP), and the simulated EChG are handy (see the Section 4.4 for the model).

6.2 Event-related potentials

Both EEG and EChG are complex combinations of signals related to a mixture of transient, oscillatory or relatively permanent activities of cell assemblies, some being related together in synchronous and/or asynchronous modes. The method chosen to sort out specific activity in the signal is the Event-related analysis, based on event aligned Evoked Potential (EP) of the signal. Momentary (temporal) changes in brain activity, as reflected in EEG, are rarely exploited due to lack of analytical tools and methodology [53], that is why we have used the averaging event-related techniques. We use a repetitive triggering event to average the signal's epochs such that those components that are time-locked to the trigger-event are summed up, while event non-related activity is averaged out, weakly summed up or substracted when in opposite phase. In any case wanted effect is to eliminate the irrelevant "on-going" brain or spontaneous neural network activity referred as "background" activity and to amplify event-related part of the signal, if it is possible.

Among the discarded "irrelevant" components there might also be signals related to the event processing, if they are different after each presentation. An example is oscillatory activity, which might have a different phase in each single measurement and therefore it would cancel out with signal averaging. This type of activity is invisible on the EPs, but it is likely to be revealed by higher order spectral analysis approaches like bispectrum and bicoherence (see Section 6.3.2 for details), which preserve the phase information by shifting the signal from time space to frequency space through the Fourier transform of each event-related signal before averaging. Unlike to bispectral methods, Event-related Potential (ERP)

will aid to discover only most general linear patterns of activity.

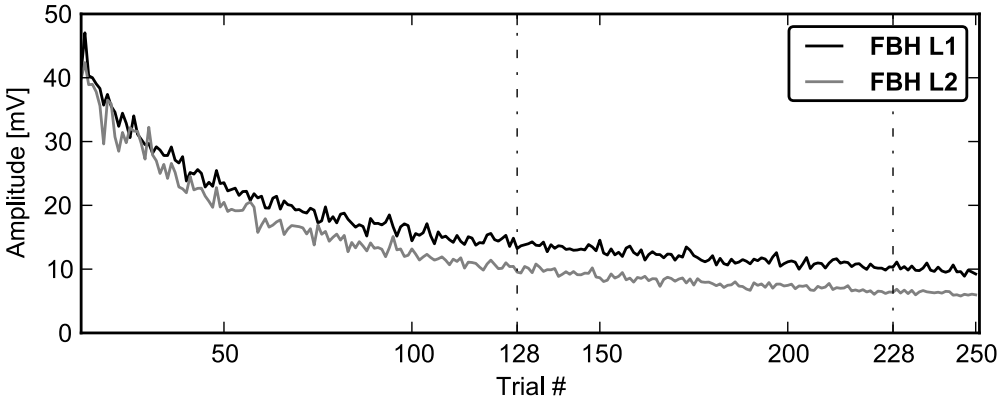


Figure 6.2: Average amplitude of the oscillations during development of a neural system of FBH topology, in the first (solid line) and second (dotted line) processing layers.

The easiest way to apply the ERP technique is to use an offset of the stimulus as triggering event then to average activity during the periods equal to the stimulus' presentation time with the Inter-Stimulus Interval (ISI) – the epoch. In this case epochs (see the Section 4.3.1) correspond to *trials*, which is a more convenient term when applied to ERP technique. The trials of the signal with stimulus-related part discarded are called trimmed trials, they are useful in spectral analysis, as will be discussed later. The number of trials to average depends on the signal variability and the desired accuracy and other limiting constraints of experimental conditions. In our neural system simulations, ERP were computed for a minimum of 25 trials, and for most simulations it was equal to either 50 or 125 trials depending of the experimental setup.

The last limitation is due to the stabilization time of the network at start of the experiment due to the high level of pruning that occurs during the early developmental phase, which is illustrated on the Figure 6.2. On the figure an average amplitude of the EChG signal during appropriate trial (on the x-axis) is depicted on the y-axis in mV. The early seconds of life are characterized by chaotic transient activity, while later network converge to a stable state, so we observe an exponential decay of activity's amplitude during network's development. This signal is taken from the experiment described in a Section 7.2 and is an average of the EChG signals from 21 neural systems simulated each having the feed-backward with horizonal reciprocal connections (FBH) topology.

As it was mentioned before, the advantage of EPs is that the irregular part of the signal is summed out, and the regular one is amplified. This is illustrated by the Figure 6.3, where 25 individual trials aligned by the stimulation event (from the trial #201 and up to the #225 from the first processing layer of the FBH circuit) are displayed on the bottom part of the figure and corresponding EP is shown on the upper part. While the individual trials seem chaotic, their sum reveals certain regularities. First, higher average level of activity during 500 ms after the stimulus onset – duration of the stimulation – short silent period just after the end of the stimulus, characteristic for that kind of cells and occasional high synchronous neuronal activity at around 1290 ms onset driven by internal

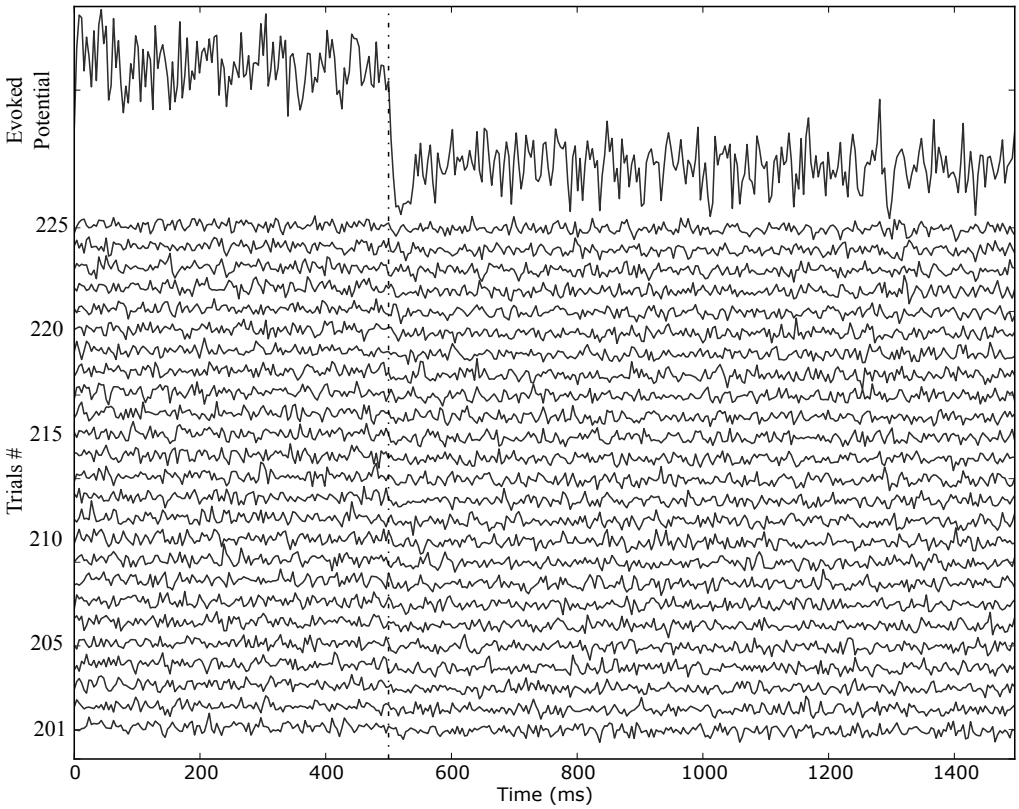


Figure 6.3: Individual EChG traces per trials #201-225 aligned by stimulation event are displayed on the bottom part of the figure. Corresponding Evoked Potential (EP) is displayed in the upper part of the figure.

synchronization of the neurons (please notice high at that time-moment).

While EPs are useful for activity levels analysis, it is also important to perform time-frequency analysis of the signal in order to find characteristic changes in the spiking frequencies and in the frequencies of synchronous activity.

6.3 Spectral analysis

The spectral analysis provides valuable information about distribution of energies transferred by the activity over selected frequency band. In the current study we discuss only the two conventional methods of spectral analysis, which rely on the stationarity of random signals, they are Power Spectral Density analysis and bispectral analysis. A random process is strictly stationary if time shifts do not affect its probability characteristics (mean, variance, etc). A process is Wide Sense Stationary (WSS) if its expected power is finite, and its mean is constant and its autocorrelation depends only on the time difference of the samples. However, stationary processes are WSS, but not vice versa.

6.3.1 Second order spectral analysis

The Power Spectral Density (PSD) or Energy Spectral Density (ESD) is a positive real function of a frequency variable associated with a stationary stochastic

process, or a deterministic function of time, which has dimensions of power per frequency. In signal analysis is useful to examine the energy of a signal by decomposing it into a series of elementary sinusoidal components of constant amplitudes and frequencies. The best-known method to obtain such result is the Fourier transform to decompose the signal into series of sinusoidal and cosinusoidal waves. The estimates of the energy of each elementary wave will give us energy distribution in the frequency domain. In this case the energy is represented by its distribution along the frequency dimension. The result of this operation is the Power Spectral Density (PSD).

The frequency spectrum is the base conventional signal analysis. With digitized signal, the power spectral density can be estimated with the Discrete Fourier Transform (DFT) or its fast computational variant the Fast Fourier Transform (FFT). There are many ways to extract the power information [129]. The two conventional methods (direct and indirect periodograms) are parametric and easily implemented.

The direct estimation of the PSD is a periodogram analysis, it extracts directly the energy from the signal. Let $x(n)$ be a WSS random process. $x(n)$ has an average power E given in Watts. The average of the total of energy is distributed over some range of frequencies. The distribution over frequency ω is described by the average PSD $S_x(\omega)$ as follows

$$S_x(\omega) = \frac{1}{N}|\sum_{n=0}^{N-1} x^*(n)e^{-j\omega n}|^2. \tag{6.1}$$

The power spectra of random, zero-mean, Wide Sense Stationary signals are obtained from the Fourier transform of these signals. In an analogous fashion the cross-spectrum of two signals $x(n)$ and $y(n)$ is defined to be:

$$S_{xy}(\omega) = \frac{1}{N}(\sum_{n=0}^{N-1} x(n)e^{-j\omega n})(\sum_{n=0}^{N-1} y(n)e^{-j\omega n})^*. \tag{6.2}$$

The indirect method of estimation of the PSD is based on the autocorrelation sequence $\varphi_{XX}(\tau)$ of the signal defined as

$$\varphi_{XX}(\tau) = \sum_t x(t)x(t+\tau). \tag{6.3}$$

With this method the signal is multiplied by a shifted version of itself, the estimate of the PSD $\Phi_{XX}(\omega)$ being obtained by the following Fourier transform of the autocorrelation sequence:

$$\Phi_{XX}(\omega) = \sum_\tau \varphi_{XX}(\tau)e^{-j\omega\tau}. \tag{6.4}$$

The cross correlation $\varphi_{XY}(\tau)$ of two signals is obtained by multiplying the first one by a shifted version of the second, i.e.

$$\varphi_{XY}(\tau) = \sum_t x(t)y(t+\tau). \tag{6.5}$$

Accordingly, the cross PSD $\Phi_{XY}(\omega)$ is the Fourier transform of the cross correlation sequence, i.e.

$$\Phi_{XY}(\omega) = \sum_{\tau} \varphi_{XX}(\tau) e^{-j\omega\tau}. \tag{6.6}$$

The periodogram are unbiased estimators but they are not consistent. This is due to the fact that the Fourier transform shows fluctuations at all frequencies even if the duration of the signal tends to infinity. Thus the variance of the estimate does not decrease to zero with increasing the duration of the time window. Usage of long duration observation interval has another drawback with EEG because in this case the signal tends to lose its stationarity over the entire recording interval.

A few methods can be used to reduce the variance of the PSD estimate and make it more consistent:

- smoothing (filtering) in the frequency domain;
- multiplying the autocorrelation sequence by a lag window function;
- multiplying the time-domain data by a window function;
- averaging several periodogram estimates.

In our studies we used only the filtering technique to reduce high-frequency noise.

The two last techniques are the most used modifications of the periodogram method in electrophysiology. First, one data interval is divided into small segments, which are multiplied by a symmetrical windowing function which tapers the extremities of the segment minimizing the border effects on the fourier transform. The resulting PSDs are then averaged to yield the final spectral estimate. Notice that a white noise input should give a flat PSD, the would have a neural network with pure gaussian activity. This fact will be used for adaptation of the spontaneous activity level described in the Section 7.1. The PSD will provide a measure of the level of activity of the neural modules in the simulated hierarchical neural circuit.

In clinical EEG, this parameter is commonly used intuitively by simple visual inspection to estimate the importance of the evolution of some pathological states. Power Spectral Density (PSD) is commonly used in several analyses when the fundamental question is a detection of little variations or changes inside one specific window of signal like in the detection of changes in insomnia under NREM sleep period and sex differences [35], in the identification of properties in spreading depression in conscious rabbits [90], in the characterization of signals from sleep EEG in twins discordant of chronic fatigue syndrome [9], in the quantification of pharmacological effects of anesthetics on the brain and the level of sedation [46], in the detection of annual variations of EEG in patients with chronic epilepsy [9] among others.

6.3.2 Third order spectral analysis

This section presents the bispectral analysis in a way to emphasize its relevance to the study of functional connectivity between interacting neuronal networks. Table 6.1 summarizes the main functions describing second and third order cumulant

statistics. For sake of simplicity the presentation of the bispectral analysis that follows is based on random signal description.

Any signal (like an EEG trace) can be modeled as a polynomial function such as

$$\begin{aligned} x(t) &= \sum_n (a_n^k (\sum_n \cos(2\pi f_n t + \varphi_n))) \\ &= \sum_n a_n n^v = a_0 + a_1 v^1 + a_2 v^2 + a_3 v^3 + \cdots \end{aligned} \tag{6.7}$$

where $x(t)$ is the model signal formally made of a series of cosine waves, each having their own amplitude a_n, frequency f_n and phase φ_n which are linearly combined to form a compound wave V.

With $n = 0, 1$, the resulting signal is $a_0 + a_1 V$, a simple combination (addition) of independent cosine waves which is the paradigmatic model used in EEG until the 1990s. With $n = 0 \ldots 2$, one adds to the previous series $(a_0 + a_1 v)$, a new series $(a_2 v^2)$ which is no more made of linear independent wave components, but is a non linear (quadratic) combination of components. With $n = 0 \ldots 3$, one adds to the previous combination of linear and quadratic series $(a_0 + a_1 v + a_2 v^2)$, a new series $(a_3 v^3)$ which is made of new non-linear (cubic) combination of wave components. By continuing this way, the model signal can be built to generate an infinitely complex and precise representation of real signals.

6.3.3 Case study of non-linear signal analysis

To help with the understanding of what quadratic non linearities means in term of signals one can simulate the case of two electrodes recording a mixture of linear and non-linear waves.

Let's consider an example: Channel #1 receive M successive signal which is a mixture of linear and non linear generators G_1 ($f_1 = 4$ Hz) and G_2 ($f_2 = 7$ Hz), such that

$$\begin{aligned} x_i^{C1}(t) &= \sum_i (g_1 + g_2 + g_1 g_2) \\ &= \sum_i (\cos(2\pi f_1 t + \varphi_1) + \cos(2\pi f_2 t + \varphi_2)) \\ &+ \sum_i (\cos(2\pi f_1 t + \varphi_1) \cos(2\ pi f_2 t + \varphi_2)) \end{aligned}$$

Channel #2 receive M successive signal which is a mixture of a linear combination of two generators G_3 ($f_1 = 4$ Hz) and G_4 ($f_2 = 7$ Hz) and a non linear interaction of generators G_3 and G_2 ($f = 7$ Hz), such that

$$\begin{aligned} x_i^{C2}(t) &= \sum_i (g_3 + g_4 + g_3 g_2) \\ &= \sum_i (\cos(2\pi f_3 t + \varphi_3) + \cos(2\pi f_4 t + \varphi_4)) \\ &+ \sum_i (\cos(2\pi f_3 t + \varphi_3) \cos(2\pi f_2 t + \varphi_2)). \end{aligned}$$

Table 6.1: Comparison of spectral and bispectral functions

Second order cumulant statistics	**Third order cumulant statistics**
Model Signal	
Linear Components	Quadratic Components
$A_1V^1 = a_1(k\sum\cos(2j\pi ft))^1$	$A_2V^2 = a_2(k\sum\cos(2j\pi ft))^2$
f_1, f_2 (no phase relations)	f_1, f_2, f_1+f_2, f_1-f_2 (same for phases)
Time Domain	
Auto-Correlation	Auto-Bicorrelation
$\varphi_{xx}(\tau) = \sum_t x(t)x(t+\tau)$	$\varphi_{xxx}(\tau_1,\tau_2) = \sum_\tau x(t)x(t+\tau_1)x(t+\tau_2)$
Cross Correlation	Cross Bicorrelation
$\varphi_{xy}(\tau) = \sum_t x(t)y(t+\tau)$	$\varphi_{xxy}(\tau_1,\tau_2) = \sum_\tau x(t)x(t+\tau_1)y(t+\tau_2)$
Frequency Domain	
Power Spectrum	Auto-Bispectrum
$\Phi_{xx}(f) = \sum_\tau \varphi_{xx}(\tau)e^{-2j\pi f\tau}$	$\Phi_{xxx}(f_1,f_2) = \sum_\tau \varphi_{xxx}(\tau_1,\tau_2)e^{-2j\pi(f_1\tau_1+f_2\tau_2)}$
$= X(f)X(f) = \lvert X(f)\rvert^2$	$= \sum_t X(f_1)X(f_2) * X(f_1+f_2)$
Cross Power Spectrum	Cross Bispectrum
$\Phi_{xy}(f) = \sum_\tau \varphi_{xy}(\tau)e^{-2j\pi f\tau}$	$\Phi_{xxx}(f_1,f_2) = \sum_\tau \varphi_{xxy}(\tau_1,\tau_2)e^{-2j\pi(f1\tau_1+f2\tau_2)}$
$= \sum_t X(f_1)Y(f_2)$	$= \sum_t X(f_1)Y(f_2)X^*(f_1+f_2)$
Phase Coupling Domain	
Auto-Coherence	Auto-Bicoherence (normalized bispectrum)
$\gamma_{xx}(f) = \frac{\Phi_{xx}(f)}{\sqrt{\Phi_{xx}(f)\Phi^*_{xx}(f)}} \leq 1$	$\gamma_{xxx}(f_1,f_2) = \frac{\Phi_{xxx}(f_1,f_2)}{\sqrt{\Phi_{xx}(f_1)\Phi_{xx}(f_2)\Phi_{xx}(f_1+f_2)}} \leq 1$
	Cross Bicoherence (normalized bispectrum)
Cross Coherence	$\gamma_{xxy}(f_1,f_2) = \frac{\Phi_{xxy}(f_1,f_2)}{\sqrt{\Phi_{xx}(f_1)\Phi_{xy}(f_2)\Phi_{xx}(f_1+f_2)}} \leq 1$
$\gamma_{xy}(f) = \frac{\Phi_{xy}(f)}{\sqrt{\Phi_{xx}(f)\Phi^*_{yy}(f)}} \leq 1$	
Loose phase information. Non-linearities considered as linearities. Any distribution considered as gaussian.	Retain quadratic non-linear phase informations. Detect non gaussian (skewed) distribution. Detect quadratic non-linearities. Zero value if a gaussian signal.

In this particular setting, Channel #1 contains non linearities which are local to the channel #1 electrode. Conversely, Channel #2 contains non linearities which are due to a generator common to both channels but which is non independent with respect to the channel #2 electrode.

The Table 6.1 shows that spectra are Fourier transforms of correlations and coherences compare spectra by their coherent versions (i.e., all phases set to 0). By applying conventional spectral analysis (that is computing PSD) one obtains identical power spectra for both channels #1 and #2 with 4 frequency peaks (Figure 6.4).

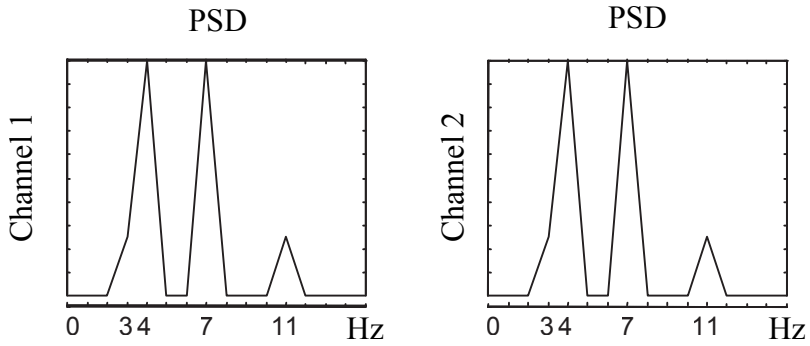


Figure 6.4: The power spectral densities of both sample channels

The Figure 6.4 shows that most energy lies at 4 and 7 Hz, as predicted by linear theory. In addition two other peaks centered at 3 and 11 Hertz are visible, although not predicted by linear theory. These frequency components are the sum and the difference of the two frequencies of generators G_1 ($f_1 = 4$ Hz) and G_2 ($f_2 = 7$ Hz) in Channel #1, and of G_3 ($f_3 = 4$ Hz) and G_2 ($f_2 = 7$ Hz) in Channel #2. The PSD on the Figure 6.4 reveal the presence of a mixture of linear oscillators giving peaks at f_1 and f_2 and non linear oscillators giving peaks at f_1, f_2, $(f_1 + f_2)$ and $(f_1 - f_2)$.

Bispectral analysis can reveal non-linear interactions because there is a difference in the phase content of each channel. Like conventional spectral analysis, bispectra can be applied in the single channel case (auto-bispectrum, aBIS) and between two channels (cross-bispectrum, xBIS) Representations of bispectra are quite difficult to understand as they are in fact 3-dimensional projections in euclidian parts of a globally non euclidian "cumulant space" manyfold.

The bispectral matrix is symmetrical and has a lot of redundancy, so it is possible to display only a triangular part in the first quadrant. For each channel the first quadrant of the bispectral matrix is displayed in 3 dimensions with a more precise 2 dimensional representation underneath which shows lines identifying the relevant triangular region of interest. The Figure 6.5 shows the bispectra for both channels. Notice that the $x = y$ line is a symmetry axis for the quadrant.

Figure 6.5 shows four high peaks at coordinates (4,3), (7,4), (3,4) and (4,7) in the auto-bispectrum of Channel #1. Only the first two coordinate pairs are needed as the other two ones are their symmetric counterpart. In Channel #2 the magnitude of the peaks at coordinates (4,3), (7,4), (3,4) and (4,7) is much lower. The significance of the peaks can be assessed by computing the bicoherence, that is comparing bispectrum with its maximally coupled version (all phases are set to zero), as shown on the Figure 6.6.

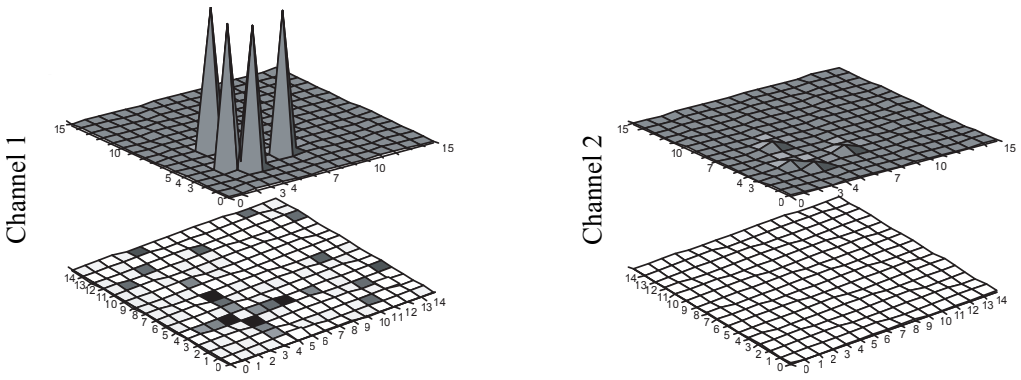


Figure 6.5: The bispectrum of both sample channels

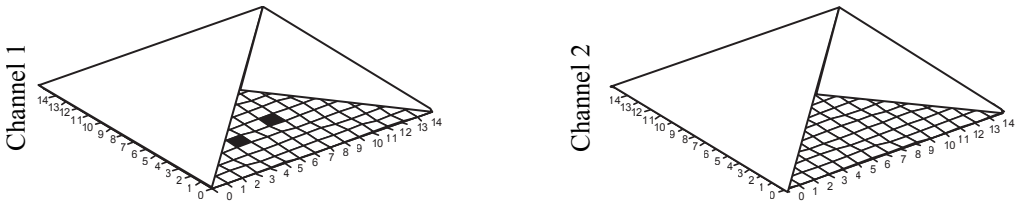


Figure 6.6: Final coupling information obtained from both bispectrum and bicoherence for both sample channels. Black dots marks coupled frequencies.

To summarize, Channel #1 contains 3 and 11 Hz non-linear local frequency components, while Channel #2 has no local non-linear frequency components. Channel #1 X(t) is non-gaussian and non-linear, while Channel#2 X(t) is non-gaussian, but a linear combination of components, even if one of its components is non-linear. This non-linear component is not phase linked to the other linear component and thus is independent. To detect its phase linked companion one has to do a cross-bispectral analysis and to be lucky enough to place the electrode at the right place.

6.4 Spectral analysis results

The Article B: "Stimuli-driven functional connectivity" describes a study on a development of information processing in hierarchically organized neural circuits. We have explored one simple neural network circuit characterized by a sensory network processing the external input and projecting its activity to two processing areas which eventually project on a motor network. The experimental approach to the system's activity by recording the EChG was aimed to assess the effect of a repeated stimulation on the functional connectivity established between the neural network agents in the topology of interest. The third order spectral analysis of EChG and EEG signals, which were the main tools of the study, allows to determine the frequency range of quadratic phase coupling (resonant frequency) across cortical areas [140, 141] and thus describe functional connectivity of the brain areas.

The results are such that late-learning development stages of the system were characterized by lower high-frequency activity in comparison with early-learning stages, which is somehow similar to the behavior observed in the clinical EEG

recordings taken from insomniac patients after appropriate treatment. In other words, both the custom stimulation of the modeled neural network and the cognitive brain therapy appear to modify the ratio of resonant frequencies provoking a shift of the indexes towards low frequencies at all brain states.

Article B

Functional connectivity driven by external stimuli in a network of hierarchically organized neural modules

Authors:	Vladyslav Shaposhnyk, Pierre Dutoit, Stephen Perrig and Alessandro E. P. Villa
Published in:	Lecture Notes in Computer Science Artificial Neural Networks – ICANN 2010
Publisher:	Springer-Verlag Berlin Heidelberg 2010
Volume:	6352
Pages:	135-144
Year:	2010

Résumé :

Les complexes de réseaux neuraux multimodales qui comporte des fonctions de développement synaptique et cellulaire sont connectés afin qu'il se forme un circuit hiérarchique récurent. Chaque module comporte une électrode virtuelle qui capte un signal, l'electrochipogram (EChG), généré par des décharges neuronales. L'analyse de l'EChG est réalisé par des méthodes d'extraction fréquentielles non linéaires. Ces méthodes nous renseignent sur les modifications de connectivité fonctionnelle par changement de fréquences d'activité couplées.
Le paradigme expérimental permet de décrire le développement des réseaux activés par le stimulus. L'analyse est réalisée sur les signaux qui (1) précédent le stimulus, (2) suivent son apparition, et qui (3) précédent et (4) suivent la disparition du stimulus. Ces résultats sont comparés à d'autres résultats obtenus chez des patients dans conditions expérimentales similaires. Nous montrons que notre modèle comporte des caractéristiques semblables aux signaux EEG.

Abstract

Complex neural modules with embedded neural development and synaptic plasticity features have been connected to form a hierarchical recurrent circuit. Virtual electrodes have been used to record a "neural" generated signal, called electrochipogram EChG, from each module. The EChG are processed by frequency domain methods to determine the modifications in functional connectivity by assessing quadratic phase coupling. The experimental paradigm is aimed to describe what happened prior to, at the beginning, towards the end, and after repeating an external input at fixed frequency. The results are discussed by comparing with the same signal processing methods applied to a human study.

Keywords: spiking neural networks, hierarchical neural networks, distributed computing, computational neuroscience, EEG.

B.1 Introduction

At mesoscopic level, the recording of brain activity by means of electroencephalography (EEG), electrocorticography (ECoG) and local field potentials (LFP) collects the signals generated by multiple cell assemblies. The neurophysiological processes underlying those signals are determined by highly non-linear dynamical systems [106]. Because of these nonlinearities the functional interactions between brain areas that are simultaneously sampled by electrophysiological techniques generate signals that can be better analyzed by third order polyspectral methods that retain phase relationships [27]. This analysis was applied to EEG by pioneers as early as the 1970s [49]. Phase coupling frequencies can be interpreted as frequencies of resonance of standing waves whose wavelength is associated to the average distance between interacting cell assemblies [140, 141].

In the present study we simulate the activity of interconnected neural networks undergoing neural developmental phases. The implementation of such complex models requires high performance of the simulation that can be achieved thanks to a powerful hardware platform, its bio-inspired capabilities, its dynamical topology, and generic flexibility of artificial neuronal models presented elsewhere [74, 77]. The outcome is the implementation of each neural network into a *Ubidule* and a network of *Ubidules* as a *Ubinet*. Within each *Ubidule* the emergence of functional connectivity driven by neural development, cell and synaptic pruning, and selective external stimuli was assessed by recording Electrochipograms (EChG) which are analog signals similar to EEG generated by virtual electrodes located into each *Ubidule* [126].

The experimental paradigm is aimed to describe what happened prior to, at the beginning, towards the end, and after repeating an external input at fixed frequency. The rationale is that the spike timing dependent plasticity (STDP) embedded in the neural network models would drive the build-up of auto-associative network links, within each *Ubidule*, such to generate an areal activity, detected by EChG, that would reflect the changes in the corresponding functional connectivity within and between *Ubidules*. This experiment is compared to a small set of recordings performed in patients suffering of primary insomnia whose EEG recordings were analyzed during several sleep phases, before and after a clinical treatment.

B.2 Hybrid system implementation

The *Ubidule* is a custom reconfigurable electronic device allowing an implementation of several bio-inspired mechanisms such as growth, learning, and neural processing [123]. The common *Ubidule* platform is an hybrid system with an XScale-class processor that manages the software components of the system, such as ontogenetic processes, communications with other *Ubidules*, monitoring and recording of the activity. This processor is equipped with an open hardware subsystem which allows connecting any sort of USB device (sensors, actuators, Wifi / Bluetooth dongles, mass storage, etc.). The processor runs an embedded Linux operating system which facilitates *Ubidule* programming and management while ensuring portability at the same time.

Both hardware and software platforms are based upon modular architecture that offers interoperability among the hardware and the software parts of the system and simplifies the usage of bio-inspired features of the hardware. The neural system simulator consists of multiple computational modules, each one corresponding to a neural network, exchanging their neural activity and/or receiving input data from hardware sensors (camera, photodiode, radars, etc.) and/or providing output to hardware actuators (motor, diode array, etc.). The characteristics of the implementation naturally geared the modeling framework towards agent oriented programming. An evaluation of the available platforms of this kind led us to select JADE [17] for the development and runtime execution of peer-to-peer applications which are based on the agent oriented paradigm [28]. It is a JAVA-based multi-agent development system that fulfils the FIPA specifications [111].

In this study each network is a 2D lattice of 20 x 20 units that includes 80% of excitatory units and 20% of inhibitory units. Our framework implements several features of brain maturation, including apoptosis active during the very initial 700 time units and STDP active from the end of apoptosis until the end of simulation. This framework was extensively described elsewhere [73, 74, 77]. Synaptic pruning occured when the activation level of a synapse reached a value of zero, so that besides cell death and axonal pruning of dead cells provoked by apoptosis, the units whose all synaptic connections were characterized by a zero level of activation were definitely eliminated from the network. All units were simulated by leaky integrate-and-fire neuromimes with background activity used to simulate the effect of afferences that were not explicitly simulated within a network. The background activity to each neuron was set to 900 $spikes/s$ with a low amplitude (1 mV) generated by uncorrelated Poisson distributed inputs. In each *Ubidule* two sets of 20 excitatory units were randomly selected among the excitatory units corresponding to the "input" and "output" layers of the *Ubidule*. The neurons of these layers send and receive connections from the other units of both types (excitatory and inhibitory) within the network in addition to the connections with other *Ubidules*.

Our circuit topology remained fixed during all simulations and the *Ubidules* were characterized by their role in the network, *i.e.*, *sensory*, *processing*, or *motor* (Fig. B.1). In our network, the *u1Sensory Ubidule* has a pure sensory role. *Ubidules* labeled *u3Process, u4Process, u5Process, u6Process* have a pure information processing role and are characterized by having neither external inputs nor afferences from the motor *Ubidule*. They are all reciprocally interconnected and send

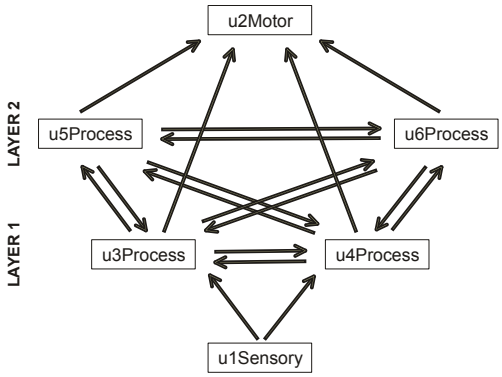


Figure B.1: The *Ubinet* hierarchical circuit used in all simulations. Solid arrows depict connections and directions of information flow between the *Ubidules*.

efferent projections to *u2Motor*.

B.3 Electrochipograms

Our design of the bio-inspired artificial neural networks allowed us to implement realistic virtual electrodes to record neuro-mimetic signals, called *Electrochipograms* (EChG), characterized by dynamics and features similar to those recorded in living brain structures. In our implementation the virtual electrode measures the potentials over a certain 'area' of the 2D lattice neuronal network according to an appropriate weighted sum [126]. The main parameters of the electode are its position over the neural network and its sensibility function. The tip of the virtual electrode was located in the middle of the 2D lattice of each *Ubidule* neural network. The sensibility function depends only on the distance between a given point of the lattice and the centre of the electrode field. According to this model, all neurons located at the same radial distance from the center of the electrode field make an equivalent contribution to the final electrode output and thus form an equi-potential layer [126]. In this study, the sensibility radius was set equal to 9 with a linear decaying function.

The EChG was recorded with a 6 channels virtual electrode system with one channel per *Ubidule* during 350 trials. Each trial had a fixed duration and included two intervals: a stimulation interval followed by an inter-stimulus interval. The stimulation was generated by spatio-temporal external stimuli applied only to the input layer of *u1Sensory* lasting 128 (*Type A*) and 512 (*Type B*) time steps. The group of simulations with higher stimulation frequency (0.89 Hz) was called "Simulations A" and the group with lower stimulation frequency (0.67 Hz) was called "Simulations B". The extensive use of Fast Fourier Transform in our signal analysis imposed, for improved efficiency, sampling frequencies which are powers of two. In practice the time-steps of the simulator were selected for convenient time units, *i.e.*, 1024 *time steps* corresponding to 1000 ms. The inter-stimulus interval was always equal to 1000 ms. The recording time was divided into four periods defined following the amount of time the *Ubinet* was exposed to the stimulation: *(i) PRE-learning* beginning at time zero and lasting 27 trials characterized by

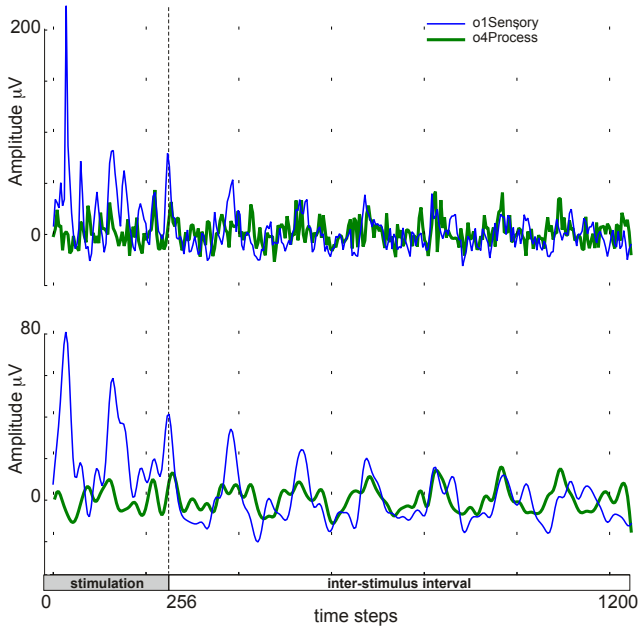


Figure B.2: Evoked potentials averaged over 50 trials obtained from *u1Sensory* (blue solid trace) and from *u4Process* (green dotted trace) *Ubidules* during the *EARLY-learning* stage. The stimulus was applied during 256 time steps. The upper panel displays the raw evoked potentials and the lower panel shows the signals smoothed by a Blackmann smoothing window in order to emphasize the low frequency components.

the absence of any external stimulation (*i.e.*, only the background activity was present during the stimulation interval); *(ii) EARLY-learning* lasting 50 trials, between trials #28 and #77; *(iii) LATE-learning* lasting 50 trials, between trials #228 and #277; and *(iv) POST-learning* lasting 50 trials, between trials #278 and #327 again characterized by the absence of any external stimulation.

The signals recorded *during the stimulation interval* were averaged across several trials in order to compute evoked potentials (*e.g.*, Fig. B.2). The signals recorded *during the inter-stimulus interval* were used for frequency domain analyses that included power spectrum, bispectrum and bicoherence analyses.

B.4 Power Spectrum Analysis

Figure B.3 shows the averaged evoked potentials for the "first" (*u3P,u4P*) and the "second" (*u5P,u6P*) processing layers and their corresponding Power Spectrum Densities (PSD). In the PSD several peaks could be observed around 10 Hz, 15 Hz and 25 Hz. The results obtained during the *EARLY-learning* stage were not significantly different from the *PRE* recording condition. This suggests that PSD is little affected by the stimulus structure and by the subsequent functional connectivity at the begin of the stimulation. This is probably due to the fact that stimulus-driven selective cell and synaptic pruning were not yet producing any effect. During the *LATE* period the PSDs were characterized by a generalized decrease in the power and the preservation of the peak near 10 Hz with a noticeable

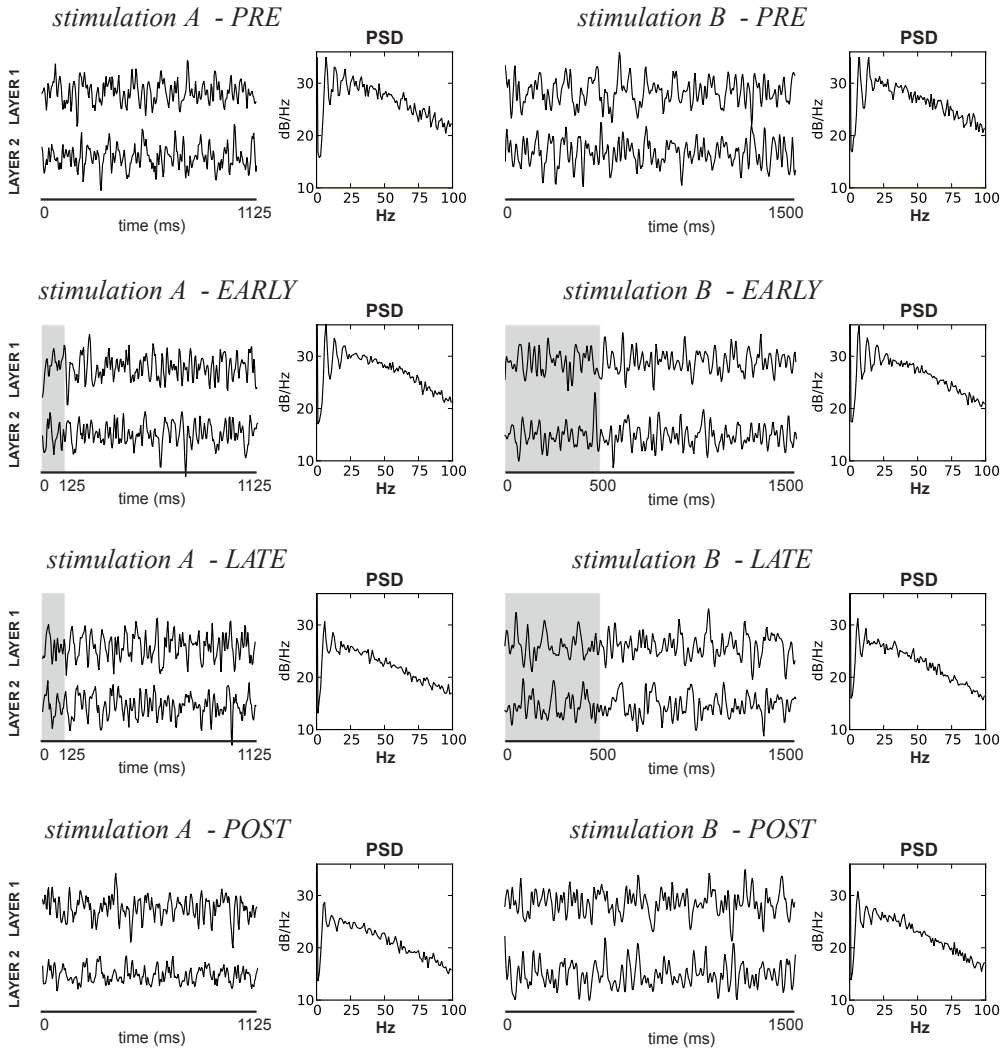


Figure B.3: Evoked Potentials and Power Spectrum Densities for the averaged recordings of the pair of Ubidules in Layer 1 and in Layer 2. The left panels correspond to stimulus Type A and the right panels to stimulus Type B. The gray stripes correspond to the periods of stimulation. From top to bottom the results referred to the *PRE-learning*, *EARLY-learning*, *LATE-learning* and *POST-learning* periods.

decrease of the other peaks. It is interesting to notice that in the *POST-learning* stage the multiple peaks tended to appear again, thus suggesting that they are mainly driven by the combined effect of background activity and internal features of the model. Another general observation is that in mature networks, *i.e.* during the *LATE-* and *POST-learning* phases in comparison with *EARLY-* and *PRE-learning* phases, PSD is getting lower, which means the total amount of energy transferred by the neural networks is decreasing. The *POST-learning* phase was characterized by 3.5 dB/Hz lower values of power than appropriate values during PRE-learning phases. This decrease is likely to be associated to the pruning of synpatic links and cell death.

B.5 Quadratic Phase Coupling

The bispectral analysis was performed for all channels separatedly and the values of phase-coupled frequencies (*i.e.*, the frequencies of resonance f_3) were determined. Let us consider the distribution of all phase-coupled frequencies f_3 observed in single-channel and cross-channel analyses. Let us consider the frequency band $]1-24]\ Hz$ for EChG and LF the relative number of f_3 falling into this low frequency range. Let us consider the frequency band $]60-84]\ Hz$ and HF the relative number of f_3 falling into this high frequency range. The *index of resonant frequencies* IRF is defined in the range 0–100 as follows: $IRF = \frac{1}{2} \times \left(100 + \left(\frac{HF-LF}{HF+LF} \times 100\right)\right)$. A value of IRF close to 100 corresponds to a shift of f_3 towards higher frequencies and value of IRF close to 0 corresponds to a shift of f_3 towards lower frequencies. IRF values close to 50 indicates the phase-coupling was equally distributed in low- and high-frequency bands. The raw frequency ratio is simply defined by $RFR = \frac{LF}{HF}$. This means a large value of RFR corresponds to a shift of phase-coupling towards higher frequencies and a low value of RFR corresponds to a shift towards lower frequencies.

Figure B.4 shows the distribution of f_3 in the range 1 to 100 Hz during all recording periods and for the two types of stimulus used in the *Ubinet* simulation. These histograms show a shift towards an increase in low-frequencies resonances during the *LATE-learning* phase, especially when compared with the distribution during the *POST-learning*, when the input stimulus was absent. The quantitative assessment of this analysis presented in Table B.1 emphasizes the change in the value of IRF between *EARLY-* and *LATE-learning* phases. $IRF \approx 60$ decreased to $IRF \approx 14$ followed by an increase to the range 26–29 during the *POST-learning* phase suggests that the shift towards low frequencies of phase-coupling was provoked by the learning protocol and not only due to the maturation of the network. The analysis of IRF and RFR shows also that in the *POST-learning* stage the resonant features remained affected by the functional connectivity that developed during the trials with external stimulation and the values were intermediate between *PRE/EARLY-learning* and *LATE-learning* phase.

Table B.2 shows the relative count of phase-coupling in the frequency bands of interest and the values of indexes IRF and RFR for all recording periods in controls and patients suffering primary insomnia before and after treatment [112]. The frequency ranges of the bands refer to those generally used for human studies and are different from those used for studying the *Ubinet* activity. However, there is a linear correspondence between the two sets of frequency bands. The general pattern was a high level of high frequency coupling in the group of patients before treatment. The main effect of the treatment was to reduce high-frequency coupling and shift phase-coupling towards low frequencies, somehow with a significant increase of low frequency coupling compared to the controls. The treatment significantly increased the phase-coupling in the low frequency band during all other intervals, either re-establishing a level close to the controls or even beyond that level, as observed during the REM sleep phases.

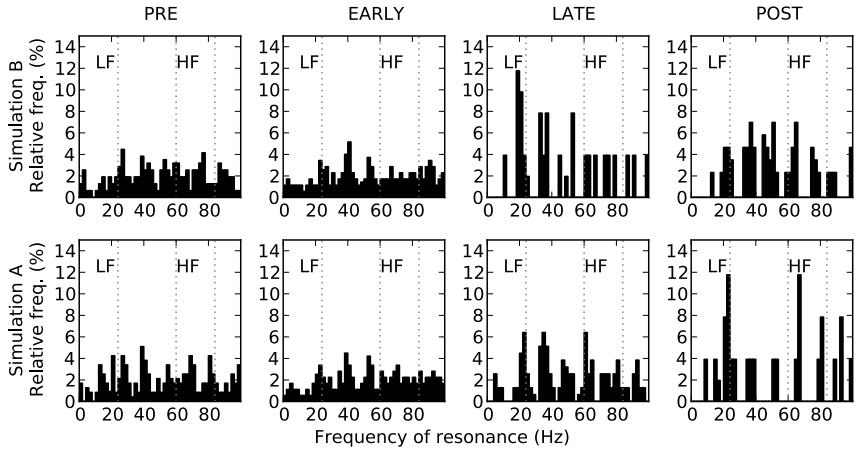


Figure B.4: Relative distribution of the frequencies of resonance for each period for Simulations A and B. Bin size corresponds to 2 Hz intervals. The dotted lines delineate the limits of LF and HF bands.

B.6 Discussion

This paper described the implementation of a neuronal system simulator on a hybrid scalable multi-agent hardware platform based on the *Ubidules* framework [123] and its application to the study of information processing in hierarchically organized neural networks circuits. We have explored one simple *Ubinet* network circuit characterized by a sensory network processing the external input that projects to a hierarchically organized multilayered (in our case formed by only two layers) recurrent network of processing areas which eventually project on a motor network that generates an activity keen to be encoded into actuators. The experimental approach to the *Ubinet* activity by recording the EChG was

Table B.1: Percentage of phase-coupled frequencies in each frequency bands of interest for the stimulus Type A and B within neural network development stages. *IRF*: index of resonant frequencies. *RFR*: raw frequency ratio.

Learning	Percentage of phase-coupled frequencies			Indexes	
Phase	LF: ⌋ 1-24⌊Hz	⌋24-60⌊Hz	HF: ⌋60-84⌊Hz	*IRF*	*RFR*
Stimulus Type A					
PRE	27	53	20	43	1.34
EARLY	20	50	30	60	0.66
LATE	38	56	6	13	6.67
POST	38	49	13	26	2.83
Stimulus Type B					
PRE	20	48	32	62	0.62
EARLY	21	47	31	60	0.68
LATE	49	43	8	14	6.00
POST	44	38	18	29	2.43

aimed to assess the effect of a repeated stimulation on the functional connectivity established between the *Ubidules*. Our *PRE-learning* stage could represent a control situation driven exclusively by the background activity of the subject's brain. The subject is naive to the coming stimulus so that a learning process can occur. During the *EARLY-learning* stage the repetition of the stimuli at regular intervals might initiate an unsupervised recognition process that eventually shaped the functional connectivity of feature detecting cell assemblies after selective synaptic and cell pruning.

The third order spectral analysis of EChG and EEG allows to determine the frequency range of quadratic phase coupling (resonant frequency) across cortical areas [140, 141]. According to the usual interpretation based on standing waves theory, high resonant frequencies mean that information processing is transmitted at short distance (*i.e.*, the distance between two nodes of the wave). A coupling that occurs at high frequencies may be interpreted as a sign of focal cortical interactions. Conversely, a coupling at low frequencies suggests an increased cross-areal involvement in neural processing.

A remarkable result is the finding that in the *Ubinet* simulations the *LATE-learning* stages were characterized by $IRF \approx 14$ compared with *PRE-* and *EARLY-learning* stages ($IRF \approx 43 - 62$). In the study with human Subjects we observed that controls and patients after treatment were characterized, during all sleep phases by values of IRF lower than insomniac patients before treatment. It is also worth reporting that the only condition that let appear a difference of resonant frequencies in the range]13-33] Hz was during NREM sleep irrespective of the treatment. This last result suggests that despite an overall shift of resonant frequencies towards recovery, focal cortical interactions tended to persist in patients during NREM sleep periods. Both an appropriate stimulation of the *Ubinet*

Table B.2: Percentage of phase-coupled frequencies in each frequency bands of interest for the the control group and for the group of patients before and after treatment. *REM*: rapid eye movement sleep. *NREM*: rapid eye movement sleep.

Subject Group	Percentage of phase-coupled frequencies			Indexes	
	LF:] 1-13]Hz	]13-33]Hz	HF:]33-48]Hz	*IRF*	*RFR*
Eyes Closed					
Control	12	74	14	54	1.17
Patient before	2	77	21	91	10.50
after treatment	8	88	4	33	0.50
NREM					
Control	57	30	13	19	0.23
Patient before	27	60	13	33	0.48
after treatment	42	57	1	2	0.02
REM					
Control	4	90	5	56	1.25
Patient before	4	85	12	75	3.00
after treatment	19	79	2	10	0.11

and the cognitive brain therapy appear to modify the ratio of resonant frequencies provoking a shift of the indexes towards low frequencies at all brain states. Our findings suggest that new tools provided by modular and scalable neural network simulators offer new opportunities to neurophysiologists and clinicians to test hypotheses based on the analysis of neural signals at mesoscopic levels.

Chapter 7

Stimulus-evoked activity

> If the human brain were so simple That we could understand it, We would be so simple That we couldn't.
>
> Emerson M. Pugh

Résumé :

Cette partie inclus une étude pionnière dans la simulation de signaux EEG générés par un grand échantillon de réseaux hiérarchiques de neurones de type « intgrate-and-fire ». En effet, les études rapportées dans la littérature analysent la stabilité de l'état dynamique du réseau, l'effet du bruit sur le réseau, ou l'émergence d'une activité synchrone. En revanche, notre approche expérimentale offre la possibilité d'étudier l'effet que pourrait avoir de la connectivité inter-corticale (inter-modulaire) sur l'activité de circuits neuronaux et sur le développement de liens fonctionnels.
Dans ce chapitre nous avons adapté les paramètres du modèle pour qu'il reflète au mieux le comportement des réseaux biologiques en tenant compte de la dimension du réseau élargi par rapport à nos expériences précédentes. Ensuite, nous avons testé l'effet de projections réciproques sur développement des liens fonctionnels dans les quatre circuits neuronaux principaux lors d'une stimulation externe artificielle. Les résultats de la modélisation montrent que les circuits modélisés manifestent un comportement similaire aux celui-ci des circuits biologiques réels.

The study done in the Article B: "Stimuli-driven functional connectivity" examines the electroencephalography (EEG) and electrochipography (EChG) signals by time- and frequency-domain methods (described in the previous Section) in order to determine underlying modifications in the functional connectivity of circuits. The experimental paradigm was aimed to describe what happened prior to, at the beginning, towards the end, and after the repetition of the external input. The results obtained from the simulated EChG signal are somehow similar to those one can observe in clinical EEGs recorded from real persons. The EChG signals were recorded by the virtual electrodes from multiple modules of the complex hierar-

chical recurrent circuit with embedded neural development and synaptic plasticity features. An implementation of the circuit (neural system) was done for a hybrid scalable multi-agent hardware platform of the PERPLEXUS project [123].

Because of the framework's constraints, in earlier studies our neural networks were modeled by 20×20 cell lattices (400 neurons per module) giving 2'400 neurons per circuit in total. These networks could be considered as medium sized ones. It is known from theoretical studies that these circuits may not exhibit certain types of biologically plausible features, which are in contrast observed in the larger cell assemblies of the same neuromime [31, 30, 113]. In particular, smaller networks may have a longer period of the initial transitional state and a shorter total life, *i.e.* period from the start of simulation and up to the moment, when the majority of neurons or projections are dead. That cuts number of the trials which could be effectively used in the analysis.

With the end of the PERPLEXUS project we moved from custom hardware modules to a more powerful universal computational cluster, so we had increased network size of every module up to 5'625 cells (by using 75×75 lattice), which gives us a total of 33'750 cells per neural circuit (compare that with 2'400 cells of smaller circuit) of a typical topology consisting of 6 modules (see the Section 4.2.2 and the Article B for details). Next two sections are dedicated accordingly to the enlarged network's parameters adaptation, in particular to the adaptation of spontaneous activity levels, and to the activity patterns observed in these larger circuits of topologies with different characteristic reciprocal projections.

7.1 Spontaneous activity adaptation

For the later studies of the Thesis module's neural lattice size was scaled up to the 75×75 cells. We started from the model's configuration parameters values being the same as in our earlier experiments. They are summarized in the Table 7.1, column "*Initial net*". Then we modified them according to a network scaling study done earlier [73]. Although many parameters are covered by this paper, it does not cover an effect of spontaneous activity level on the network behavior. Thus this was done separately and the results spontaneous activity adaptation are presented here. Resulting model's parameters are presented in the Table 7.1, column "*Final net*".

Activity's level adaptation served to achieve two principal goals:

- to exclude a state of permanent very low spiking activity of the network, which in turn causes massive amount of synapses falling in depressed state followed by neural death and network's overall inactivity
- to exclude a state of permanent very high spiking activity of the network, leading to over-excitation of cells, making them spiking every time refractory-lock period (2-3 time-steps) is lifted, and producing biologically non-plausible outcome.

7.1.1 Spontaneous activity adaptation methods

The simulations were done with single Sensory neural module of 75×75 cells and for each combination of frequency and amplitude modeling was repeated 3 times

Table 7.1: Small and Large models' parameters summary

Parameter	Unit	Initial net	Final net
Network			
Lattice dimentions	HxW	20×20	75×75
Neurons		400	5'625
Projections		$\approx$ 8'000	$\approx$ 850'000
Efferent neurons		20	450
Afferent neurons*		20	900
Stimulus amplitude	mV	1.60	1.90
Apoptosis started	ms	0	0
Apoptosis ended	ms	750	750
STDP started	ms	750	750
Spontaneous activity amplitude*	mV	1	1.9
Spontaneous activity frequency*	Hz	900	300
Excitatory cell parameters			
probability	%	80	80
refractory period	ts	3	3
PSP (base)	mV	1.6	0.92
reset threshold	mV	-78	-78
spiking threshold	mV	-40	-40
Excitatory cell distribution			
probability	%	20	20
Uniform distribution weight		0.02	0.02
2D Gaussian distribution weight		0.6	0.6
2D Gaussian diameter		4	15
2D Gaussian sigma		0.5	1.875
leakage constant		0.98	0.98
Inhibitory cell parameters			
probability	%	20	20
refractory period	ts	2	2
PSP (base)	mV	-1.96	-1.64
reset threshold	mV	-78	-78
spiking threshold	mV	-40	-40
Inhibitory cell distribution			
Uniform distribution weight		0.02	0.02
2D Gaussian distribution weight		0.2	0.2
2D Gaussian diameter		12	45
2D Gaussian sigma		2	7.5
leakage constant		0.974	0.974
Electrode parameters			
center (first electrode)	x,y	9.5, 9.5	20.5, 20.5
center (second electrode)	x,y	N/A	55.5, 55.5
diameter	cells	18	32
area covered	cells	322×1	900×2
acquisition frequency	Hz	256	256
decay function		linear	linear

* parameters were varying according to experimental protocol

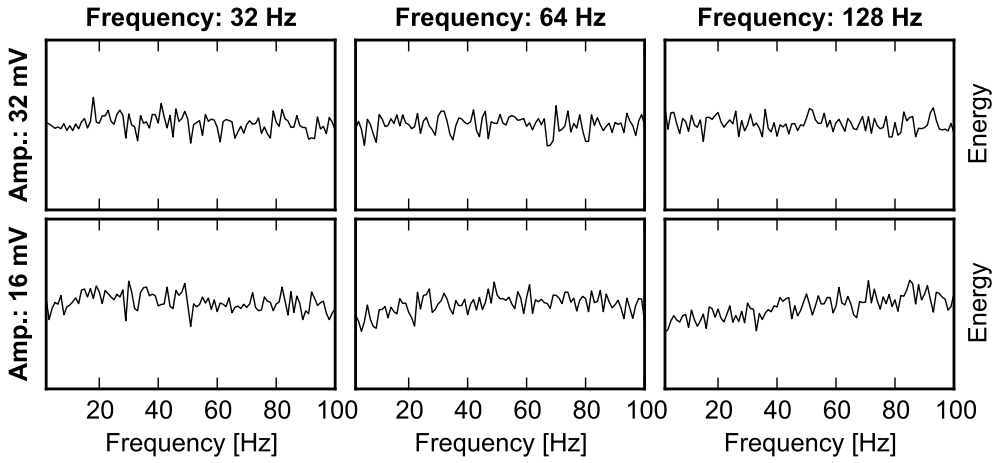


Figure 7.1: PSDs of the simulations with low frequency (16-32 Hz) of the spontaneous activity.

to obtain convergent data.

We established a "grid" of possible pairs of spontaneous activity's amplitude/frequency combinations. The upper boundary of stimulation amplitude was limited by cell's spiking threshold, *i.e.* by 38 mV and the lower, obviously by a zero. Spiking frequency was limited by simulator's time-resolution – up to 1024 spikes per second, *i.e.* 1024 Hz. From our previous studies we knew that high energies transmitted by spontaneous activity will provoke continuos discharges of the cells, which is not biologically plausible. According to that two grids of amplitude/frequency combinations were established, such that we avoid the combinations of amplitude and frequency *a priory* leading to the network's hyper activity. These grids were as follows:

- a lower stimulation frequency grid with amplitudes of 16, 24, 32, and 40 mV and frequencies of 32, 64, 128 Hz;
- a higher stimulation frequency grid with amplitudes of 2, 4, 8, and 16 mV and frequencies of 250, 450, 900 Hz;
- having obtained the results from the first two sets, we added one more grid with amplitudes ranging from 1.6 to 2.1 mV with a step of 0.1 mV and the frequency fixed at 300 Hz.

The averaged EChG signal obtained from the simulations was used to calculate Power Spectral Densitys (PSDs) to ensure bio-plausibility of the registered signal. Having spatiotemporal stimulus applied to the neural module, we expect to register a non-gaussian signal, which should result in "non-flat" power-spectrum in contrast to the gaussian one, as it is explained in the Section 6.3. Average cell death rates were monitored to ensure that networks stay alive and are not hyper active. In real brains during early development phase around one third of all cells are dying [137] and we want to stay close to this threshold.

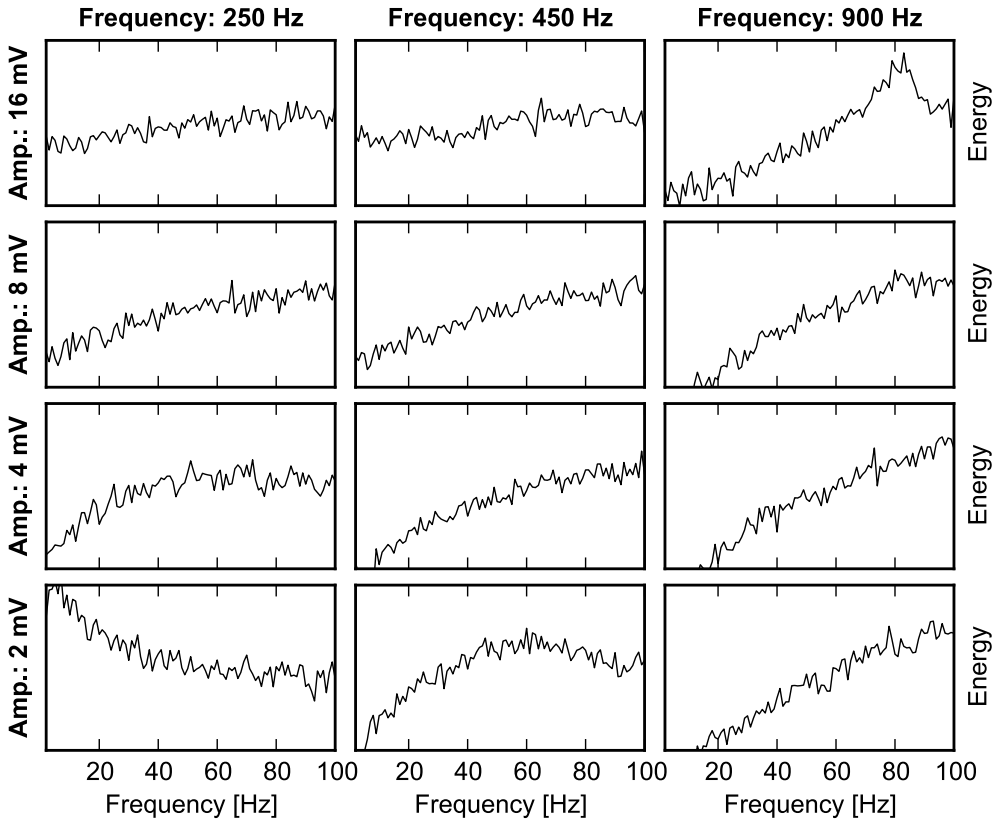


Figure 7.2: PSDs of the simulations with high frequency (250-900 Hz) of the spontaneous activity.

7.1.2 Activity adaptation results and discussion

The most representative PSDs from the simulations are shown on Figure 7.1 (lower-frequency set), Figure 7.2 (higher-frequency set), and Figure 7.3 (300 Hz frequency set). Cell death rates caused by apoptosis process (which is accounted for approximately 96.7% of all dead cells) are summarized in the Table 7.2 and are visualized on Figure 7.4.

Power-spectrum of the lower-frequency set is quite flat, as one can see from the Figure 7.1. Slight shift in the activity is visible on plots corresponding to the simulations with 128 Hz of stimulation frequency, but it is the shift to the 70-100 Hz frequency band, which is mostly produced by random activity of cells. That assumption is supported by the results of the high frequency set of simulations Figure 7.2 and, in particular, by three simulations with very high spontaneous activity levels with frequency of 900 Hz with amplitude of 16, 32, and 40 mV, which are producing PSD with strong peaks (5-20 times higher than any other peaks) at $\approx$ 80 Hz mark. One can observe an appearance of a such peak on the 16:900 panel of the Figure 7.2, obtained from the simulation with the amplitude of 16 mV and the stimulation's frequency of 900 Hz. Visual inspection of spike-trains in these simulations revealed constant spiking as soon as neuron's refractory period is lifted. This observation led us to believe that peaks at 80 Hz are residuals of artifacts of the neuronal activity at frequencies of 512 Hz and 341 Hz (refractory periods of 2 and 3 time-steps used in the our model). An activity occurring at these

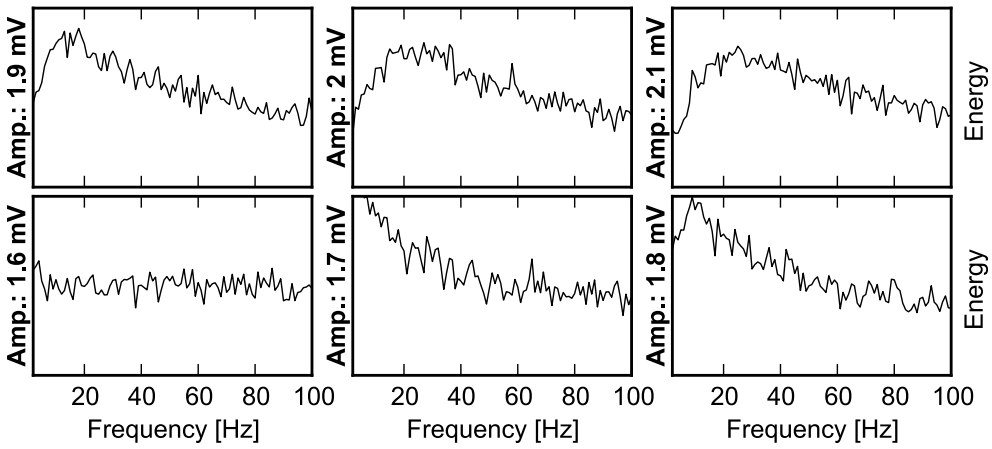


Figure 7.3: PSD of the simulations with frequency of 300 Hz.

rates cannot be captured by Fast Fourier Transform (FFT) estimations capped by the frequency two times lower than signal's acquisition frequency, which in the case of EChG was equal to 256 Hz.

The results obtained with the lowest amplitude and the frequencies of 250 and 450 Hz were promising, because of a shift of the activity to the lower-frequency band, which is much more biologically plausible than a shift towards the 80 Hz. Cell death monitoring (summarized on the Figure 7.4) revealed a local-minimum at 300 Hz, which corresponds to the 29% cell death. This allowed us to specify an area of the interest at around the point of 2 mV and 300 Hz. The third set of frequency/amplitude combinations was simulated (see Figure 7.3), as mentioned above.

Although the very similar results obtained in this set of simulations a slighter preference was given to the 1.9 mV stimulation, where a sharper low-frequency peak was discovered. Visual inspection of recorded spike-trains was done to verify observed results. It is confirmed an acceptable activity level in the whole simulations' set and, in particular, in the simulation with spontaneous activity process characterized by a frequency of 300 Hz and an amplitude of 1.9 mV.

Now, when the parameters tuning is done, we will proceed to the study of an effect of presence of reciprocal projections on the functional connectivity between neural areas.

Table 7.2: Apoptosis driven cell death

Simulation parameters		**Observed cell death rate**		
Frequency	**Amplitude**	**Exitatory**	**Inhibitoty**	**Total**
Hz	mV		%	
	Low frequencies			
016	16.0	25.0%	3.9%	28.9%
016	24.0	25.5%	4.0%	29.5%
016	32.0	26.2%	4.9%	31.1%
016	40.0	28.3%	5.8%	34.1%
032	16.0	25.3%	4.0%	29.4%
032	24.0	26.5%	4.7%	31.2%
032	32.0	29.2%	6.1%	35.3%
032	40.0	28.1%	6.1%	34.2%
064	16.0	27.0%	4.6%	31.6%
064	24.0	28.4%	5.6%	33.9%
064	32.0	29.9%	7.3%	37.2%
064	40.0	29.5%	7.1%	36.6%
128	16.0	28.9%	5.5%	34.4%
128	24.0	29.2%	6.5%	35.6%
128	32.0	30.8%	7.8%	38.6%
128	40.0	30.3%	7.6%	37.9%
	High frequencies			
250	1.0	25.2%	3.9%	29.1%
250	2.0	26.0%	3.9%	29.9%
250	4.0	26.4%	3.9%	30.3%
250	8.0	29.2%	5.1%	34.4%
250	16.0	31.5%	7.1%	38.7%
450	1.0	24.7%	3.5%	28.2%
450	2.0	26.0%	3.7%	29.7%
450	4.0	28.5%	4.4%	32.9%
450	8.0	31.9%	5.4%	37.3%
450	16.0	35.4%	8.1%	43.5%
900	1.0	26.8%	3.8%	30.7%
900	2.0	29.2%	4.1%	33.3%
900	4.0	33.7%	5.2%	39.0%
900	8.0	37.6%	6.3%	43.9%
900	32.0	39.5%	7.7%	47.2%
	Mid-frequencies			
280	2.0	25.2%	3.3%	28.5%
300	1.8	25.2%	3.5%	28.7%
300	1.9	25.8%	3.5%	29.3%
300	2.0	25.9%	3.6%	29.5%
320	2.0	25.7%	3.6%	29.3%

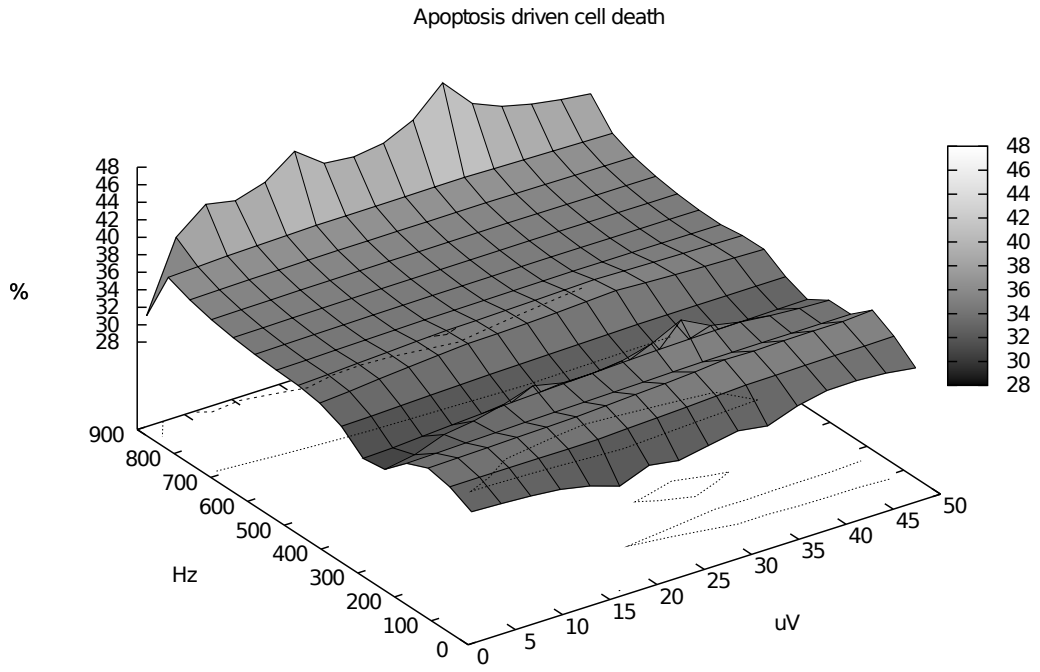


Figure 7.4: Apoptosis driven cell death landscape depending on the spontaneous activity level. Activity's amplitude in mV on the X-axis, frequency in Hz on the Y-axis, and cell death in the percentage over the total number of cells in the network in on Z-axis.

7.2 Reciprocal projections role

Synchronization of neural oscillations is likely to be associated with the key process of binding of information processed in distributed brain regions [127, 148]. The simulation studies of event-related EEG performed with oscillatory cell population models [89, 48], have emphasized how modulation of the strengths of positive and negative feedback between brain modules may affect the wave shape and the time course of Evoked Potentials (EPs) [116] and in particular the emergence of damped oscillations in presence of backward connections [40].

In the current Section we present the simulation results of activity of interconnected spiking neural modules undergoing ontogenetic and epigenetic developmental phases (see the Chapter 3). We recorded neuro-mimetic signals EChG, by the means of realistic virtual electrodes. The neural circuit was characterized by two layers of information processing networks and we study emergent properties of stimulus-locked response depending on the presence in the circuit of inter- and intra-layer projections. The rationale is that the Spike-timing-dependent plasticity (STDP) embedded in the model would drive the build-up of auto-associative network links, within each neural module, which generate an areal activity, detected by EChG, that would reflect the changes in the corresponding functional connectivity within and in-between neuronal modules. We used genetic features of the simulation framework to code model parameters in the neuronal genome. The drift of genes through generations of neural circuits allowed us to observe *general* results that are shared by all simulated networks.

7.2.1 Modules' characteristics

The experiments were performed using the model parameters obtained after the tuning described in the Section 7.1. Final model parameters are shortly summarized below.

Every neural module was simulated by a 2D lattice of 75×75 cells, that includes 80% of excitatory neurons and 20% of inhibitory neurons. The background activity was generated by uncorrelated Poisson-distributed inputs at an average rate of 300 $spikes/s$ with an excitatory postsynaptic potential (e-PSP) of 1.9 mV. An exception to this rule was the Sensory module, that receives a stimulation of 3.8 mV, aimed to reflect stronger noise at the peripheral cortex. Synaptic pruning occurred when the activation level of a synapse reached a zero, the cells having all synaptic connections at zero level of activation were definitely eliminated from the network. Pruning process co-exists in the model with cell death provoked by apoptosis (see Section 3.6). Evolutionary part of the framework was used to apply a mutation operator to the genome of neural circuits modeled. Multiple bit-flip mutation operator was applied to genome, which is an extension of single bit-flip operator [92]. Species of next generation of circuits were produced with a probability specified by its fit function at the event associated with the "death" (the end of the simulation time) of the parent. In this study, we used multiple bit-flip mutation operator in conjunction with a dummy selection function, which gives 100% probability of replication (see Section 5.3.3).

In the each neuronal module two sets of cells were randomly selected among excitatory neurons of the module. These sets correspond to the efferent and the af-

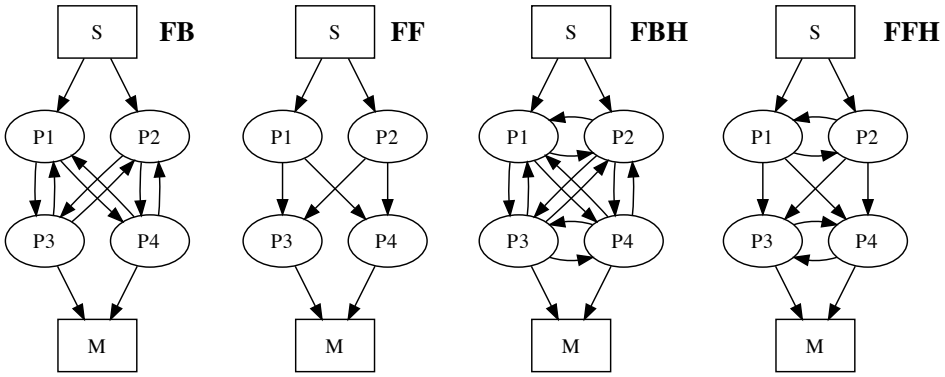


Figure 7.5: Four hierarchical circuits used in the experiment. The arrows depict projections and directions of information flow between the neural network modules. FB: feed-backward topology without reciprocal projections within processing layer (horizontal connections); FF: same as FB, but without feed-back projections between successive layers; FBH: feed-backward topology *with* horizontal connections; FFH: same as FBH, but without feed-back projections

ferent layers. During early developmental stages inter-network connections (input-output inter-network activity mapping) are established in a self-reflective manner so that the number of external projections from a particular efferent neuron is proportional to the number of internal projections of the same neuron at the developmental stage. For all modules but Sensory, there were 900 afferent cells and 450 efferent cells (approximately 20% and 10% of all excitatory cells of a module). The Sensory module had both layers consisting of 450 cells. Efferent cells were projecting 5 times less synapses to external network than they had in the internal network. This was done to escape overexcitement (persistent stimulation) of the afferent layer, which was observed in the case of one-to-one reflection scheme. Large number of internal projections each neuron has, caused all afferent neurons to be stimulated every time-step of the simulation.

7.2.2 Hierarchical setup

Four circuit topologies featuring different combinations of reciprocal inter-module links were used in the study. All topologies were composed from 6 neural modules. In all cases, neural modules of 3 roles *Sensory*, *Processing*, and *Motor* were assembled in to hierarchical circuits of interest (see Figure 7.5).

In our networks, the *Sensory* neuronal module, labeled *S*, always had a pure sensory role, this was the only network receiving external artificial sensory stimulus and two times stronger background activity, reflecting noisier signals of peripheral cortex. Four modules labeled *P1, P2, P3, P4* had pure information processing role and were characterized by having neither external artificial inputs nor afferences from the *Motor* module. First pair of them was receiving input directly from the *Sensory* module, thus forming first processing layer of the network (*Layer 1*),

and second pair was connected only with other *Processing* and, eventually, *Motor* modules, forming second processing layer of the network (*Layer 2*). Two principal topological features of the circuit were reciprocal projections *inside* a processing layer and reciprocal projections *in-between* processing layers. Combination of these features gives us 4 possible types of topology:

- the one with reciprocal projections between the processing layers, but without intra-layer projections is a feed-backward topology, abbreviated FB, depicted on the left panel of the Figure 7.5;
- another one without any reciprocal projection is a feed-forward topology (FF – second on the left panel on the figure);
- a topology similar to the first, but with intra-layer reciprocal projections (further called "horizontal" links) is called feed-backward with horizontal projections and abbreviated as FBH, second on the right panel of the figure;
- the latest is feed-forward with horizontal links – FFH, depicted on the right panel of the Figure 7.5.

As it was mentioned above, we focused on effect the feed-forward, feed-backward, and horizontal inter-module projections have on the functional connectivity of the circuit in the presence of external spatiotemporal stimulus.

7.2.3 Spatiotemporal stimulus setup

The external stimulation was applied by means of a spatiotemporal stimulus fed to the afferent cells of the sensory module. The stimulus lasted 500 ms and activated each afferent cell once per 10 ms on average. Each stimulus was followed by a silent period of 1000 ms, called Inter-Stimulus Interval (ISI). The duration corresponding to the stimulus application and ISI is an elementary "trial". The stimulus' pattern in each trial was slightly modified by introduction of a 10% variability, which means a jitter of ± 1 ms introduced in the activation time of 10% randomly selected afferent cells. This straight-forward procedure was repeated irrespective of the selected cells, so that a cell could be selected more than once by chance and the final jitter would became greater than ± 1 ms. This occurrence introduced even greater variability, but it happened only rarely to deviate simulation results.

7.2.4 Experimental virtual subjects

In this study the electrode's sensitivity was limited by a circular area with a radius of 19 cells. Sensitivity function was decaying linearly from the center (100%) to its edge (0%). Two electrodes, one located in the top-left "extremity" of the 2D lattice and the second one located in the bottom-right "extremity", were placed in every module. The dual recording is performed only to gather additional signals for data analysis and reduce the effect of the noise embedded in the signal. Indeed, no difference between the electrode locations is expected because of the wrapped toroidal model of the network's lattice and the random distribution of the efferent and the afferent cells across each module.

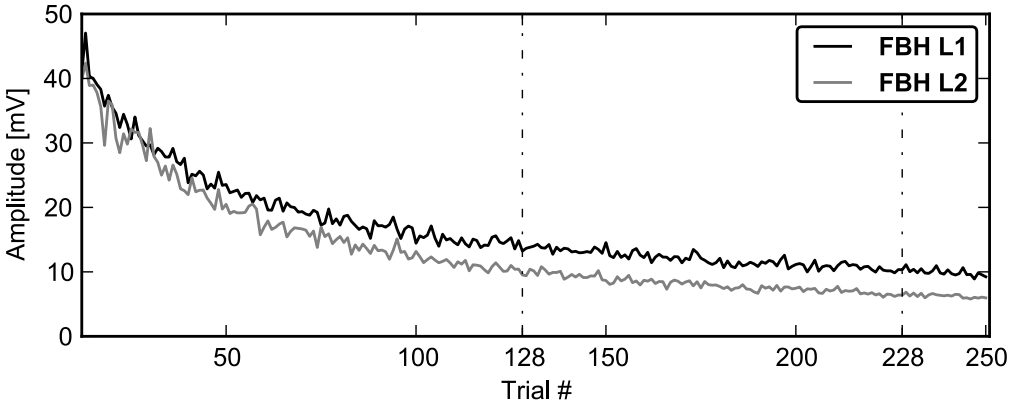


Figure 7.6: Time-course of the peak-to-peak amplitude of the ERPs for each layer of circuit FBH measured during the ISI as a function of the trial number. Each point is the average of 84 recordings. The dashed lines indicate the period from trial #128 to #228, which has been selected for further averaging for event-related analysis.

The most common method of Signal-to-Noise Ratio (SNR) enhancement in EEG-like signals is the stimulus synchronized averaging [71]. It suggests to average signal over multiple subjects and over multiple trials aligned by a key-event – usually start of the stimulus presentation. This is where introduced evolutionary features of the simulation framework came in handy. The evolutionary simulator was used to create different genomes for each subsequent neural circuit modeled. Four different genomes corresponding to each of the 4 topologies described above were used as basic ones and each of them was changed 21 times by random mutation of a gene responsible for actual connectivity pattern within neural module. Even a small variation of this gene was enough to produce drastically different internal connection maps, while preserving given distribution of the connections. Like that, we got 21 different species representing each type of the circuit, which is enough to average out the signal's noise and spontaneous activity. While in this experiment we did not use the evolutionary features of the framework to the full extent, it is an important step to verify that everything works as intended and it could be used in the future in more complex experiments with evolutionary features.

7.3 Dynamics evoked by reciprocal projections

The total duration of a single simulation run was 375 seconds (250 trials of 1500 ms each). In order to improve the signal-to-noise ratio of the EChG in the time-domain we have calculated ERPs by averaging several recordings triggered by the same stimulus onset. The average was extended across all 21 different Virtual Subjects obtained by genome mutation. Moreover, we grouped together the four recordings performed from within the same layer of modules (2 recordings $\times$ 2 modules per layer). Which means that for each circuit and for each layer of the circuit we analyzed the signals averaged across 84 recordings ($= 21 \times 2 \times 2$).

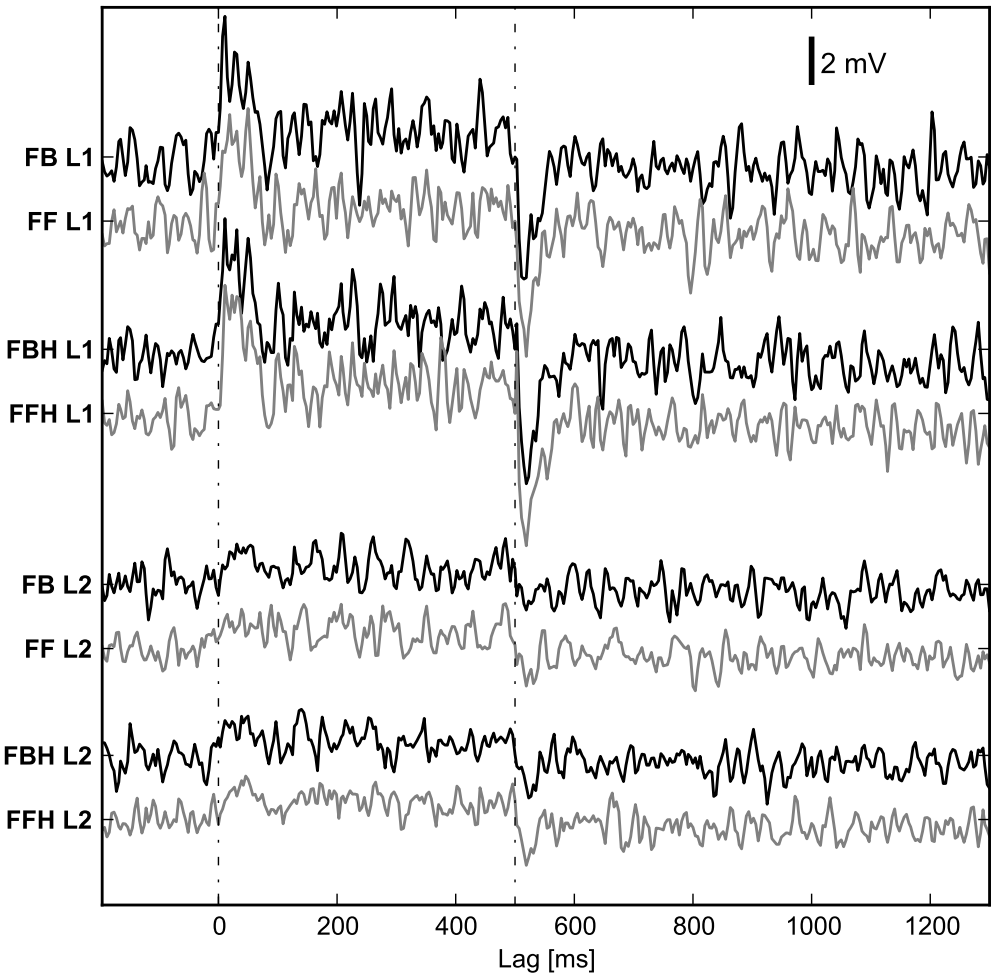


Figure 7.7: ERPs averaged across 21 Subjects, 8400 trials overall, triggered by stimulus onset. The dotted lines at 0 and 500 m correspond to the onset and offset of the stimulus. A Blackman-Tukey curve smoothing with a window of 20 ms was applied to eliminate high frequency components. The labels refer to the circuit and to the order of the layer. Black curves refer to ERPs recorded in FB and FBH circuits. Gray curves refer to ERPs recorded in FF and FFH circuits. Notice that the amplitude scales for *Layer 1* and *Layer 2* are slightly different.

The effect of network maturation due to plasticity, synaptic pruning and cell death processes is illustrated by the time-course of the peak-to-peak amplitudes of the ERPs measured after the end of the stimulus presentation during the ISI of successive trials (Figure 7.6). The curves show that the maximum amplitude of *Layer 2* activity tended to decrease more than in *Layer 1* until approximately trial #100. After this time both layers showed a tendency to decrease the level of activity, but in much more steady way than before and having their relative difference unchanged.

7.3.1 Event-related Potentials

For ERP analysis we decided to select an arbitrary range of 100 trials between trials #128 and #228 for further averaging. This means that ERPs were analyzed across a grand average of 8400 trials ($= 21 \times 2 \times 2 \times 100$).

Let us define MAD_{ISI} as the median absolute deviation amplitude during ISI as

$$MAD_{ISI} = \underset{\forall x \in X_{ISI}}{median}(|\underset{\forall t \in X_{ISI}}{median}(t) - x|) \tag{7.1}$$

where X_{ISI} stands for a signal recorded during the ISI periods. We can express the strength of the response (S_R) as the ratio between the median amplitude of ERP during the stimulus presentation and the corresponding MAD_{ISI}. For *Layer 1* modules S_R was equal to 7.58, 7.18, 7.68 and 7.65 for the FF, FB, FFH and FBH circuits, respectively. The values of S_R for each circuit and for each layer pair were calculated with 95% confidence intervals (Table 7.3). These data show that in *Layer 1* a moderate increase in S_R by 3-7% observed in the presence of horizontal projections was not significant. For *Layer 2* modules S_R was much lower than the values found in *Layer 1*, as it can be immediately observed on the ERPs plots (Figure 7.7). This is a trivial observation given the direct afferences from the sensory module to *Layer 1*. For *Layer 2* modules S_R was equal to 2.66, 2.01, 2.05 and 1.84 for the FF, FB, FFH and FBH circuits, respectively. The presence of horizontal links decreased S_R in *Layer 2* by 10-20%. The presence of feedback projections, irrespective of the horizontal links, also reduced S_R in Layer 2 response by a similar proportion. In *Layer 2* these effects were cumulative and $S_{R_{FBH}}$ was reduced by 30% compared to $S_{R_{FF}}$.

The Figure 7.7 shows the ERPs for each topology and each layer of the circuit. At the onset of the stimulus the amplitude of ERP increased for *Layer 1* modules, irrespective of the circuit. Interestingly, the presence of feedback projections from *Layer 2*, modified the response in *Layer 1* of FB and FBH *vs.* FF and FFH. In the *Layer 1* of feed-backward topologies a burst of γ-oscillations appeared immediately after the stimulus onset in presence of feedback projections from *Layer 2*. The horizontal projections emphasized the inhibitory offset response: in *Layer 1* the

Table 7.3: S_R confidence interval calculated according to the smoothed percentile bootstrap methodology [50].

Layer	**Circuit**	S_R	**95% confidence**	
			lower	**higher**
L1	FB	7.180	6.020	8.938
	FBH	7.645	6.532	8.904
	FF	7.580	6.626	9.060
	FFH	7.678	6.516	9.111
L2	FB	2.011	1.510	2.581
	FBH	1.837	1.381	2.187
	FF	2.661	1.972	3.278
	FFH	2.054	1.592	2.504

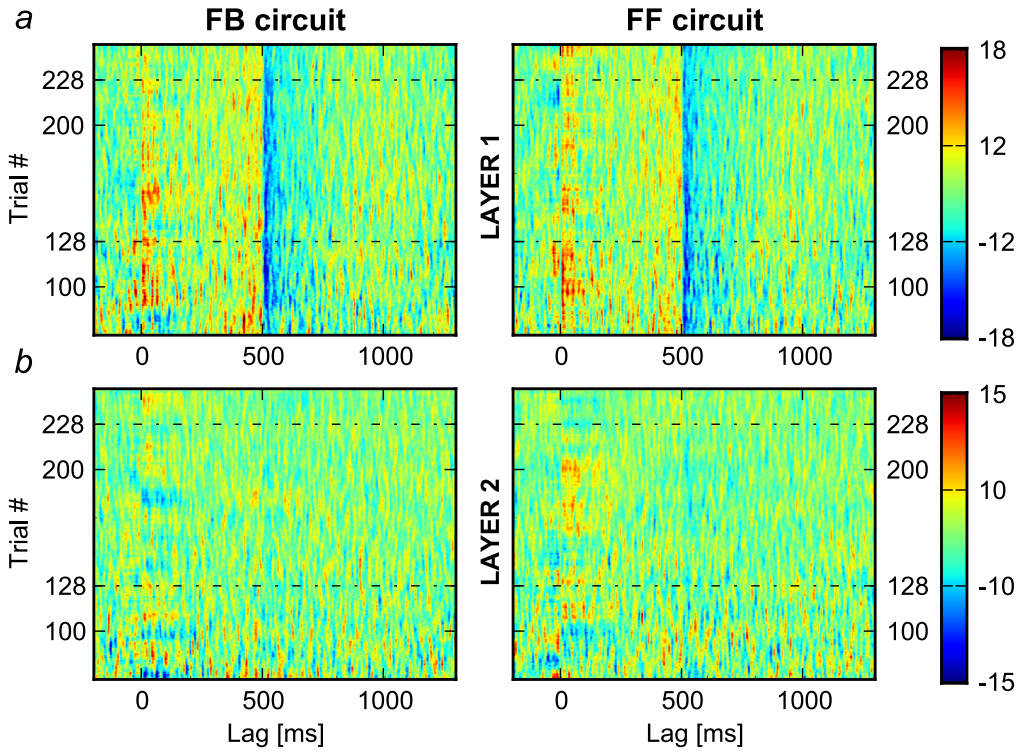


Figure 7.8: Trial-by-trial dynamics of the ERPs in *Layer 1* and *Layer 2* of circuits *without* horizontal projections. The amplitude of the ERPs is scaled in mV and color-coded following the colors on the right scale. The dotted lines at trials #128 and #228 indicate the range that was used to calculate the grand averaging.

duration of the offset inhibition was prolonged by approximately 40 ms; in *Layer 2* the offset inhibition was sharper.

The dynamics of the ERPs can be observed on a trial-by-trial average on Figure 7.8 and Figure 7.9. It is interesting to notice that before trial #128 inhibitory onset responses appeared transiently in *Layer 2* with any kind of connectivity among the modules. This pattern occurred briefly again near trial #180 only in the FB circuit. For *Layer 1*, the comparison of the figures shows that the presence of horizontal connections is not only making the onset excitation and the offset inhibition sharper, but is also reducing inter-trial variability.

7.3.2 Frequency Domain Analyses

Trials #128 and #228 were used for the computation of the PSD and we averaged the trial-by-trial PSDs. The PSD analysis shows that in any circuit *Layer 1* was characterized by a higher powers than in *Layer 2* for all frequencies. This difference is due to the direct input from the Sensory module to *Layer 1*. We have assessed the effect of introducing feedback projections in the circuits by computing the difference between the PSDs with and without feedback projections (FB and FBH vs. FF and FFH, respectively). For each bin of the power difference curves we calculated the 95% confidence interval based on the distribution of the trial-by-trial difference in PSDs. We consider here the frequency bands α, β and γ in the

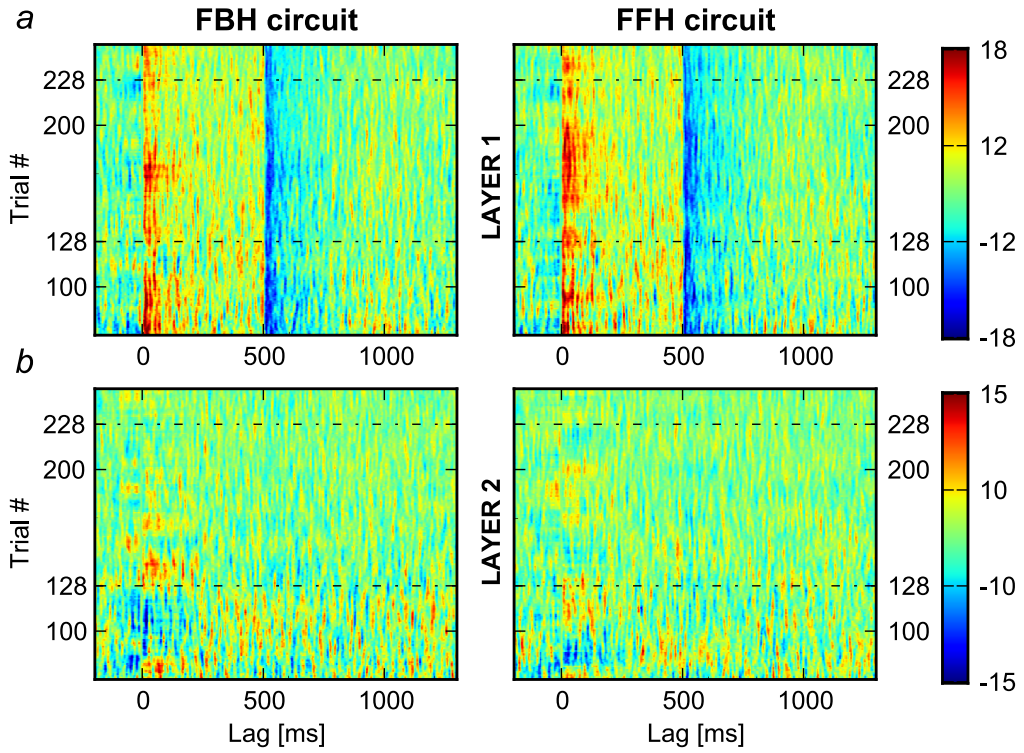


Figure 7.9: Trial-by-trial dynamics of the ERPs in *Layer 1* and *Layer 2* of circuits *with* horizontal projections. The amplitude of the ERPs is scaled in mV and color-coded following the colors on the right scale. The dotted lines at trials #128 and #228 indicate the range that was used to calculate the grand averaging.

ranges [5-20[, [20-40[, and [40-100] Hz, respectively. The shift in the range limits of the frequency bands towards higher frequencies is determined by the smallness of the network size of a module with respect to a realistic brain area. The power of $PSD_{FB.L1}$ and $PSD_{FBH.L1}$ was larger due to the presence of the feedback, as shown by the curves of the power differences that tended to stay above the zero line (Figure 7.10b,e). In particular we observed two significant peaks in the differential curves of Layer 1 of either circuit. In the presence of additional horizontal links both significant peaks were in the γ-range tended to stay above the zero line (Figure 7.10e). Notice that a burst of γ-oscillations appeared immediately after the stimulus onset in the presence of feedback projections (Figure 7.7).

It is noticeable that in *Layer 2* the horizontal links increased even further the overall power of PSD and particularly the γ-oscillations (Figure 7.10f). On the opposite, in the absence of horizontal links, $PSD_{FB.L2}$ was characterized by a power that tended to be smaller than $PSD_{FF.L2}$, in particular in the γ-range (Figure 7.10c). Then, PSD analysis showed that *Layer 2* activity during the stimulation was very much affected by the circuit connectivity.

During the ISI, *Layer 1* was characterized by a power larger than *Layer 2* at all frequencies (Figure 7.11a,d), in a way similar to what was observed during the stimulation. In *Layer 1* the presence of feedback projections in the circuit tended to increase the number of significant peaks in the γ-range, (Figure 7.11b,e) especially in the absence of horizontal links (Figure 7.11b). It is interesting to

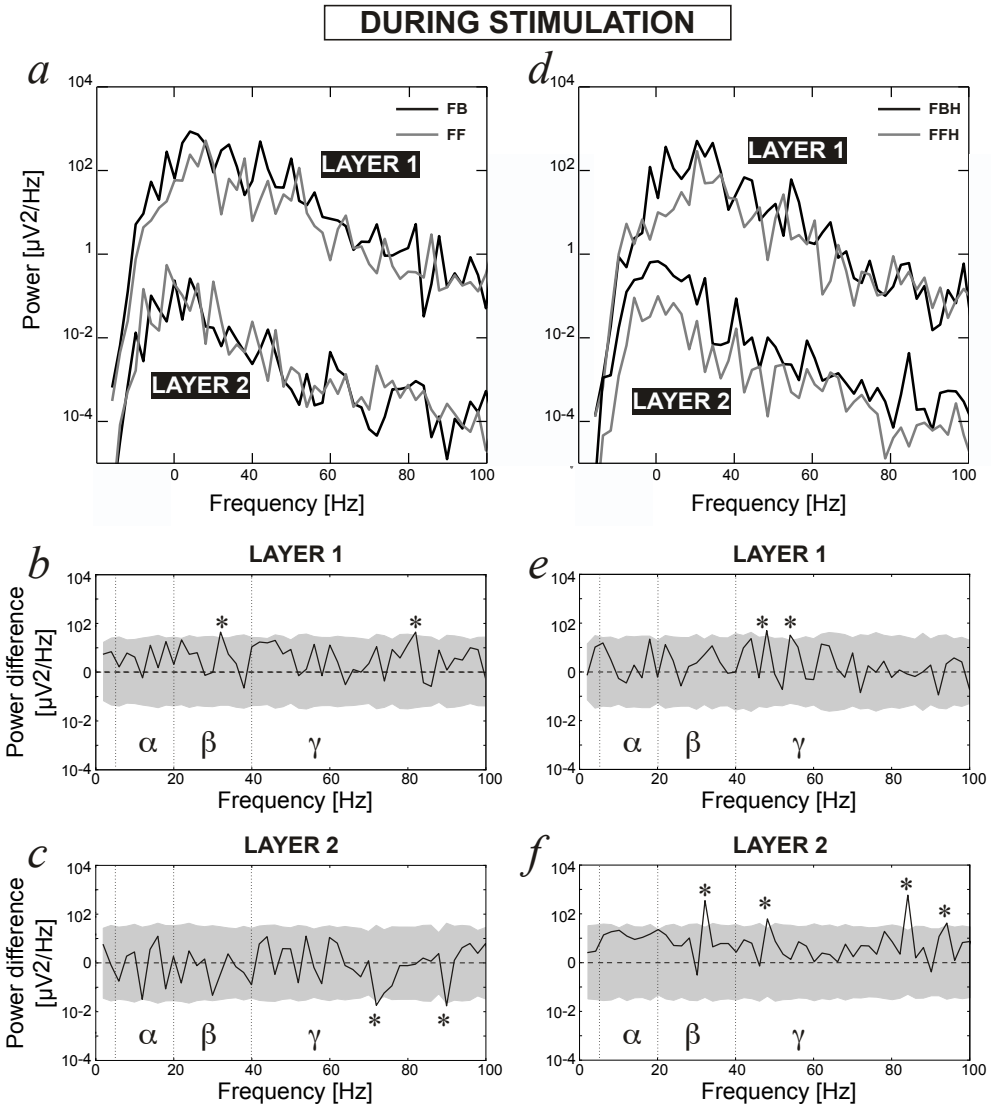


Figure 7.10: Averaged Power Spectrum Densities during the stimulus presentation in Layer 1 and Layer 2 of FB, FF circuits (panel a) and circuits with horizontal links, FBH and FFH (panel d). Black curves refer to circuits with feedback projections and gray lines to feed-forward circuits. The difference $PSD_{FB} - PSD_{FF}$ with the 95% two-tailed confidence intervals (limits of the shaded area) for Layer 1 and Layer 2 is plotted in panels b,c, respectively. The difference $PSD_{FBH} - PSD_{FFH}$ for Layer 1 and Layer 2 is plotted in panels e,f, respectively. The analysis is performed with a resolution of 2 Hz. The asterisks are used to label the significant peaks of the differential curves. We consider here the frequency bands α, β and γ in the ranges [5-20[, [20-40[, and [40-100] Hz, respectively.

notice that the PSD in FB.L2 and FF.L2 were very similar (Figure 7.11a,c). On the opposite, the presence of both horizontal and feedback connections increased the power of FBH.L2 *vs.* FFH.L2 throughout the frequency range (Figure 7.11d), in particular in the γ-range (Figure 7.11f).

The assessment of the correlation between EChG signals from *Layer 1* and

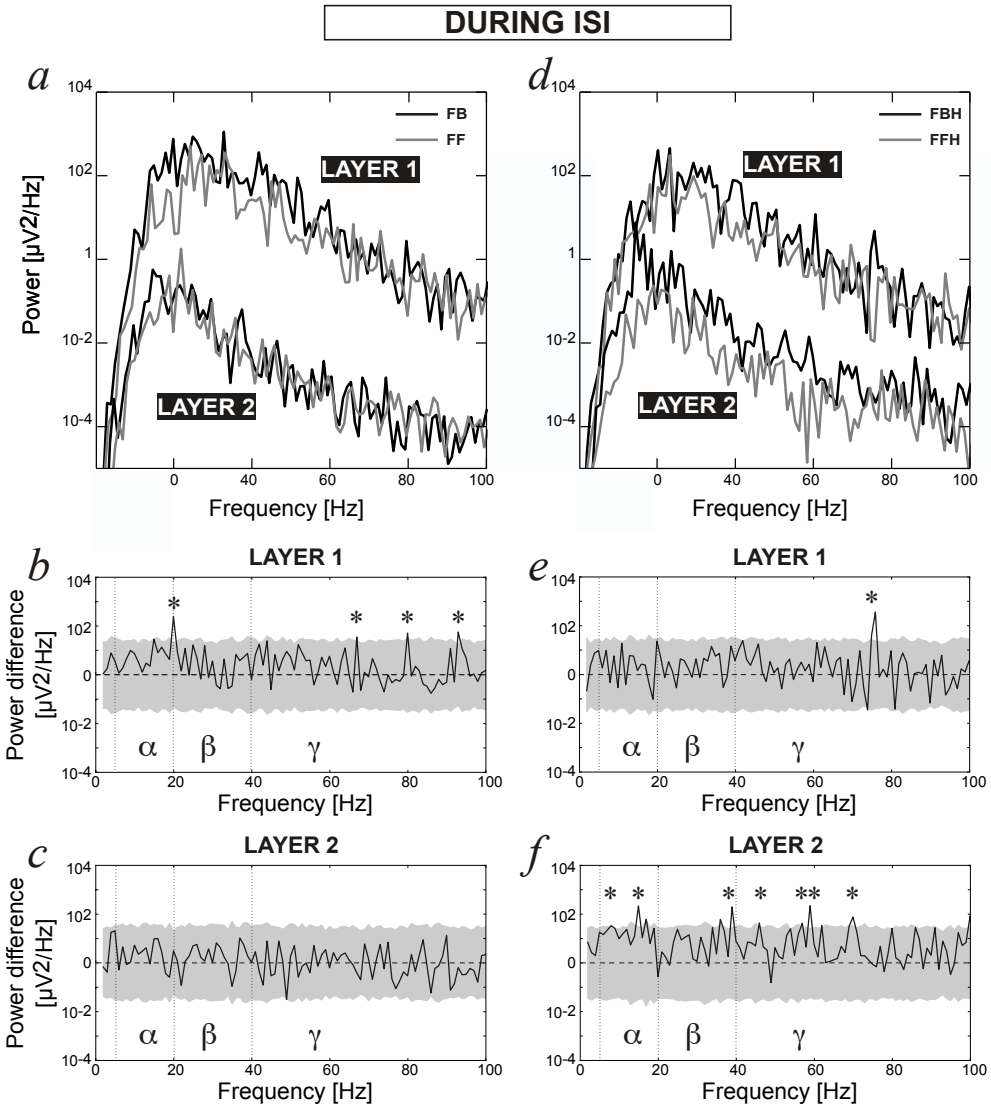


Figure 7.11: Averaged Power Spectrum Densities during ISI in Layer 1 and Layer 2 of FB, FF circuits (left panels) and circuits with horizontal links, FBH and FFH (right panels). The analysis is performed with a resolution of 1 Hz. We consider here the frequency bands α, β and γ in the ranges [5-20[, [20-40[, and [40-100] Hz, respectively. The labels are the same as in Figure 7.10.

Layer 2 in the frequency and in the time domains on a trial-by-trial basis was performed by the cross-coherence analysis between trials #128 and #228. The cross-coherence $XCOH^{A,B}(f,t)$ between two channels, A and B, at the given frequency f in the time window centered on t is calculated according to the equation

$$XCOH^{A,B}(f,t) = \frac{1}{n}\sum_{k=1}^{n}\frac{F_k^A(f,t)F_k^B(f,t)^*}{|F_k^A(f,t)F_k^B(f,t)|} \ ,$$

where $F_k^A(f,t)$ and $F_k^B(f,t)$ are short-time Discrete Fourier Transform (DFT) of signals A and B, and $F_k^B(f,t)^*$ is the complex conjugate of $F_k^B(f,t)$ [42]. The

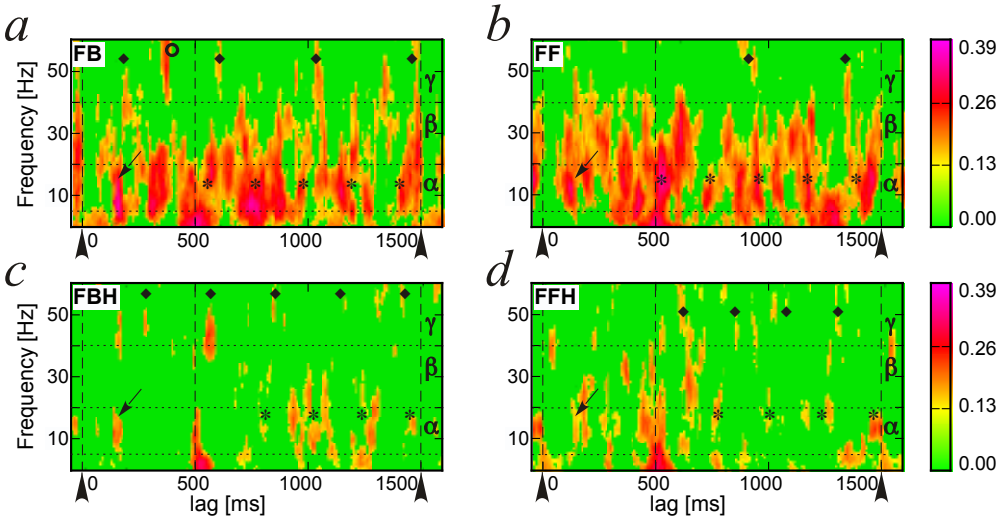


Figure 7.12: Cross-coherence maps between the *Layer 1* and the *Layer 2* for all circuits calculated between trials #128 and #228. The value of cross-coherence is color-coded according to the scale on the right, ranging between 0 and 0.39. Non-significant values are zeroed. The map is calculated using DFT with a resolution of $1/16\ Hz$ in frequency and 6 ts in time. The horizontal dotted lines correspond to frequency band boundaries. The big arrows indicate the stimulus onset and the vertical dashed lines at 500 and 1500 ms indicate the Inter-Stimulus Interval. A digital infinite impulse response filter for $[1-55\ Hz]$ bandpass was applied before signal processing in order to reduce high frequency noise. Notice that during ISI peaks of cross-coherence tended to appear at regular intervals in the α-band, marked by the asterisks, and in the γ-band, marked by black diamonds. The small arrows indicate cross-layer coherence at $\sim$150 ms in all circuits.

value of the cross-coherence varies between 0 meaning a complete absence of synchronization and 1 meaning perfect synchronization.

The cross-layer coherence for each circuit type is illustrated by Figure 7.12. The significance of the coherence values was assessed by bootstrap statistics [42] such that non-significant values ($2p < 5\%$) were zeroed for the sake of the colored drawing of the map of Figure 7.12. We use the same α, β, and γ frequency bands defined above. The presence of the horizontal links decreased the vast majority of the cross-layer coherence throughout the frequency spectrum (Figure 7.12a,b *vs.* Figure 7.12c,d). Notice the strong cross-layer coherence for all circuits at very low frequencies right at the stimulus offset (lag = 500 ms) due to the offset inhibition visible also in the ERPs (Figure 7.7).

During the stimulus presentation the cross-layer coherence was strong in the α-band at $\sim$150 ms after the stimulus onset in all circuits (Figure 7.12, small arrows). The presence of the feedback projections decreased the cross-layer coherence in the β-band (Figure 7.12a,c *vs.* Figure 7.12b,d). A very strong cross-layer coherence appeared $\sim$400 ms after the stimulus onset in the γ-band of FB circuits (Figure 7.12a, empty circle).

During ISI we observed several interesting significant values of cross-layer coherence that emphasizes the effect of inter-modules connectivity on the pattern of

activity of the entire circuit. The strongest cross-layer coherence in the β-band was observed for the pure feed-forward circuit (Figure 7.12b). Both circuits without horizontal links were characterized by strong cross-layer coherence in the α-band. More interestingly the significant peaks tended to appear at regular intervals in the α-band (marked by the asterisks on Figure 7.12) of $\sim$220 ms in FB, and $\sim$220 ms in FF. Despite a much lesser degree of cross-layer coherence produced by the presence of the horizontal links this rhythmic pattern was also observed in FBH and FFH with intervals of $\sim$220 ms and $\sim$230 ms, respectively. This suggests the presence of a slow rhythm ($\sim$4.4 Hz) across the whole circuit independent of backward and horizontal projections.

During ISI the activity of the two layers was coherently correlated at regular intervals also in the γ-band (see the black diamond symbols in Figure 7.12). In the absence of the horizontal links (Figure 7.12a,b) the rhythm was 2.3 Hz for both FB and FF circuits. On the opposite, in the presence of the horizontal links the rhythm of the peaks in the γ-band for FBH and FFH was a bit faster, 3.5 and 4.3 Hz, respectively. We observe that in the presence of feedback projections (FB and FBH) the first peak of these rhythms tended to appear before the stimulus offset (Figure 7.12a,c).

7.4 Discussion

The encoding of connectivity properties in the "genome" of the circuit allowed us to produce many different circuit species and study the common features of information processing shared by the whole sample of individuals. We have analyzed the activity of four basic circuits characterized by a sensory module receiving an external input carrying spatiotemporal information that projects to two hierarchically organized multilayered streams characterized by optional recurrent (feedback) projections from the downstream to the upstream modules and optional intra-layer projections. Each network module undergoes a maturation process followed by an active unsupervised learning process determined by spike-timing-dependent plasticity rules meant to maintain active learning dynamics. These processes are simulated at the cellular level and the network activity is recorded by virtual electrodes located in each module. The recorded EChG signals are analyzed by ERPs techniques triggered by the stimulus onset and by power density and cross-coherence analyses.

This is the one of the first works that reports simulated EEG-like signals generated by large sample of evolvable networks of leaky integrate-and-fire neurons. Previous simulation studies of EEG were based on population dynamics and neural masses [56, 65, 39, 11, 59]. They were generally aimed at determining the stability of network dynamics, the effect of noise and the emergence of synchronous activity in relation to epileptogenesis, etc. Our goal was limited to a computational study that partially reproduces the signals observed in biological experimental conditions. Though it represents an oversimplified approach to the complexity of real brain networks it offers the possibility to address a key issue like the effect of inter-areal connectivity on the network activity.

We showed an evidence that all circuits are able to maintain patterns of activity of hundreds of milliseconds triggered by the stimulus offset. This finding

is in agreement with the occurrence of preferred sequences of spikes, which are dependent on the stimulus presentation but not triggered by it, recorded in the single module simulations [75]. We have shown that the offset of the stimulus is also the most significant event that triggers coherent activity in the low frequency range throughout the network of any circuit studied here. It could be interpreted according to standing waves theory. Low frequencies suggest that information processing is made by areas located far away from each other, thus involving large neural networks in processing stimulus related activity. This is also in agreement with recent experimental findings in human experiments that revealed specific low frequency coherence patterns associated with processing type (simultaneous or successive) regardless of other experimental conditions (contents and modality) [108].

It is interesting to notice that the Power Spectral Density of the EChG recordings showed more energy in the γ-band for *Layer 2* of the FBH circuit than in the same Layer of the other circuits. The FBH circuit, which is characterized by feedback and by horizontal projections, was also characterized by cross-layer coherence extending during ISI in the γ-frequency range. These results suggest that in the circuits with feedback projections the bursts of cross-layer coherent γ-activity are likely to be triggered by some process that started during the stimulus presentation and that is not affected by stimulus offset. A network exhibiting multiple partially synchronized modes strongly excited by a stimulus, with a wide range of flexible, adaptable, and complex behavior, has been modeled as the variance of the connection gain increases, inhibitory connections become more likely and global synchronization is shown to decrease [61]. This activity might be associated with a maintenance and control task integrated in the stimulus memorization process, as a form of working memory [147, 136].

The effect of introducing connections between modules of the same layer provoked also an enhancement of the stimulus-locked onset excitation and offset inhibition in the ERPs of *Layer 1*, the layer receiving the input from the sensory module, irrespective of the feedback links. In *Layer 2* the effect was more subtle and we could observe it better by the cross-coherence. The duration of evoked transients is likely to increase with the hierarchical depth of processing [40]. However, we found late components after stimulus offset in both FF and FB circuits, which raises the possibility of alternative hypotheses that the simple dependency on backward connections to reflect a reentry of dynamics to hierarchically lower processing areas [40]. The discrepancy with those results may be due to the differences with their modeling because neural masses are unlikely to realistically account for the diversity of activity patterns that can emerge within the networks of spiking neurons that belong to a neural module. It is important to remember that the coherence value indicates a linear statistical association between time-series in a given frequency band [33]. The absence of linear statistical association between two processes does not mean the absence of any interaction. Higher-order frequency domain statistics like bicoherence and cross-bispectral analyses might be well suited to reveal interesting nonlinear interactions as suggested in a FBH-like network study [112] and in the Article B: "Stimuli-driven functional connectivity". The search for inter-module transient functional connectivity and its comparison with linear methods [52] is still limited by the understanding of the impact of different methodological choices on the outcome of the analysis [15].

Chapter 8

Approaches for non-linear data analysis

All brontosauruses are thin at one end, much much thicker in the middle, and then thin again at the far end. That is the theory...

The Dinosaur Sketch
Monty Python

Résumé :

Dans ce chapitre, nous présentons les objectifs pour mener les analyses d'optimisation structurelle et paramétrique de données multivariées contaminées par des artéfacts. Ensuite, il est rapporté la description des avancées algorithmiques concernant l'optimisation structurelle non-linaire robuste (cf. Article C « Structural modeling robust to outliers »).

There are many methods that can be used to extract knowledge from experimental data and to determine its mathematical structure [6]. Linear regression analysis is widely used in quantitative structure-relationship studies because of simplicity of the approach itself and because of ease of results interpretation. These studies represent an important part of the drug design process, where they are used to reveal relationships between chemical structure of compounds and their biological activities, or of the neuro-physiology, where they are used to uncover brain areas more than other influenced by an input stimulus in respect to electroencephalography (EEG) recorded, or of the physics – to discover new laws from experimental data. The power of linear regression analysis can be significantly increased, if it is combined with evolutionary approaches. They provide powerful techniques to analyze large multivariate data-sets with highly collinear variables [91, 115]. To overcome the limitation of linear models the feed-forward Artificial Neural Networks (ANNs) and the support vector machiness (SVMs) can be used to model complex nonlinear relationships and thus they are useful methods in such studies. However, a serious disadvantage of these methods is that the dependencies detected between parameters and response variables are hidden within

inner structure of weight matrices and therefore an interpretation of calculated results is next to impossible.

Approaches of Group Method of Data Handling (GMDH) represent sorting-out methods that can be used for analysis of complex objects having no definite theory [97, 150]. The choice of the appropriate GMDH algorithm depends on the specifics of the problem to be solved. While the classical GMDH approaches are well suited to solve such problems in general, the mean least squares (MLS) method used in the core of iterative GMDH approaches, is sensitive to outliers. In the presence of outliers the model becomes unstable which often leads to distorted interpretation of the data. To overcome these limitations, the Robust Polynomial Neural Network (RPNN) was proposed [5]. It is an iterative GMDH type algorithm that provides robust linear and nonlinear modeling in the presence of outliers in response variables and correlated or irrelevant variables. It is robust to certain outliers and allows to control models' complexity – number and the maximal power of terms in the models. The algorithm converges to stable results that can be easily interpreted. But, while the RPNN is robust to the outliers in the *response* variables, it is not robust to the outliers in the *explanatory* variables. Unfortunately, this type of outliers is often seen in the neuro-physiological data, where EEGs data to analyze could be contaminated by muscular activity, eye movement artifacts or other types of experiment non-related artifacts.

The method developed in the frame of the Thesis is named Enhanced Robust Polynomial Neural Network (ERPNN). It is still an iterative GMDH-type algorithm, like its predecessor (RPNN algorithm), so it inherits all its advantages, but at the same time it is improved by the Generalized Maximum likelihood estimator (GM-estimator) based core, which adds robustness to the outliers in the *both* explanatory variable and response variables of the data set. Detailed description of the proposed algorithm is given in the Article C: "Structural modeling robust to outliers". The algorithm could be used for the task of EEG signal (real or modeled) classification as a function of subject's state or mental activity type (like in the studies [8, 12, 66]) or in EEG channels localization, to provide for a particular task meaningful set of linear or non-linear combination of EEG channels for further interpretation and analysis [99, 139]. Despite very promising preliminary results obtained on the artificial data revealed in the article, an application of the algorithm to the real data is left for the future work.

Article C

Advances in structural modeling robust to outliers in explanatory and response variables

Authors:	Vladyslav Shaposhnyk, Alessandro E.P. Villa and Tetyana Aksenova
Published in:	Proceedings of the 2010 World Congress on Computational Intelligence
Pages:	628-635
Year:	2010

Résumé :

La régression robuste est une approche statistique d'analyse de données de distribution non gaussienne. Plus spécifiquement, la régression non-linéaire robuste et la modélisation structurelle encore se trouvent dans une phase du développement active. Cet article décrit les avancées algorithmiques de résolution de taches de l'optimisation structurelle. L'optimisation mathématique s'appuie sur des modèles polynomiales de données multivariées polluées par des artefacts. Dans ces données, les variables des réponses (Y) et les variables d'explication (X) peuvent contenir des artefacts. Dans l'étude précédente, l'ancienne méthode d'optimisation structurelle (Polynomial Neural Network ; PNN) traitait les données avec artefacts inclus dans des variables des réponses (Y) uniquement. La nouvelle version de l'algorithme décrite dans ce travail est toujours basée sur les approches PNN du type GMDH. Grâce aux réseaux neuronaux artificiels de synthèse adaptative et évolutive, cette méthode propose une modélisation universelle de la structure du modèle. Plus particulièrement, cet l'algorithme a été amélioré par un système de l'estimation générale du maximum de vraisemblance (GM-estimation). Ce système permet à faire l'estimation des paramètres du modèle. Cette estimation confère l'avantage d'être robuste à la fois aux artefacts dans les variables de la réponse, et aussi aux variables d'explication.
La nouvelle version du réseau de neurones polynomial robuste (ERPNN). L'implémentation a été testée sur les données générées artificiellement. Les données ont été crées a partir des polynômes aléatoires de deuxième et troisième dégrée. Ensuite, un bruit blanc et des artefacts ont été introduis dans les données. Cette nouvelle version s'avère plus précise que la précédente. En outre, elle permet de reconstruire automatiquement les modelés non linéaires de et d'estimer les paramètres des modèles malgré la présence importante d'artefacts.

Abstract

The robust regression analysis works on data affected by deviations from a general assumption of normality. Currently the field of robust linear regression analysis is well developed and there are number of stable and verified by time methods. In contrast the robust structural modeling and high-order model parameter estimation are still under active development.

This paper describes advances in the algorithm development designed to solve a task of optimal polynomial model selection on multivariate data sets in presence of outliers in both explanatory and response variables. Previous version of our robust Polynomial Neural Network (PNN) was addressed to the modeling of the data with outliers in response variable only. On one side novel algorithm is still based on GMDH-type PNN, which gives an universal model structure identification thanks to the evolving adaptively synthesized bounded network. And on the other side the algorithm is enhanced with GM-like estimator used for parameter estimation,

which allows to achieve robustness to outliers in both explanatory and response data-sets.

Enhanced RPNN was developed and tested on the artificial data-sets generated from polynomials of up to third degree. The Gaussian noise as well as outliers was added to the data. Enhanced RPNN demonstrated robustness to outliers in both explanatory and response variables (with 25% of outliers) and good accuracy of the automatic structure syntheses as well as of the parameters estimation.

Keywords: polynomial neural network, robust regression, non-linear regression, gm-estimators, structure selection

C.1 Introduction

The most important legacy of the scientific method is the necessity to build a model aimed to explain the physical world and to test one's hypotheses and ideas against that model. In an ideal case an experimenter manipulates some variables and measures the results of this manipulation. This paradigmatic case is characterized by the selection of the variables to be manipulated – called *independent variables* – and the observed variables – called *dependent variables*. The goal of a *regression* is to find a model fitting the experimental observations given the independent variables. Let us consider a non-linear regression model:

$$y = f(\mathbf{x}; \beta_{\mathbf{o}}) + \xi \tag{C.1}$$

where $f(\cdot)$ is a non-linear model function, $\mathbf{x} = \{x_1, \ldots, x_m\} \in \mathbb{R}^m$ is a vector of independent variables, $\beta_{\mathbf{o}} \in \mathbb{R}^p$ is a vector of model parameters, $\mathbf{y}$ is a dependent variable, and ξ is an error term. Measurements of dependent variables are generally assumed to be characterized by a distribution identical to ξ has. However, in most cases the distribution of ξ is not known and it is necessary to assume the presence of *outliers* – observations which are very different from others in a data-set – among the dependent variables. The outliers are characterized by measurement errors leading to a numerical difference between the value generated by the causal model – the *real* value – and the *observed* value. Created by an unexpected error distribution or by a mistaken observation, outliers may lead to selection of a wrong model $f(\cdot)$ and to a wrong estimation of model parameters $\beta_{\mathbf{o}}$. In order to overcome the effect of the outliers in the case of linear models several methods (e.g., Least Median Squares [120], S-estimators [95, 121], MM-estimators [149]), were developed in order to provide *robust* estimators with high breakdown point [64, 44]. In the non-linear case, a wide set of Group Method of Data Handling (GMDH) algorithms [79] were developed for structure and model selection.

In most studies there are no possibilities to manipulate the variables and the experimenter is actually collecting measurable information and observe how variables are related to each other. In those cases the definition of independent variable holds with the implicit belief of the experimenter about causal relations between variables. That means an independent variable is defined with respect to how it is related to or influences a dependent variable. Conversely, a dependent variable is defined with respect to how it is related to or influenced by the independent variables. In other terms, it is often assumed that the independent variables predict or explain the dependent variables and viceversa the dependent variables are

explained or predicted by the independent variables. For this reason, an independent variable may be called an *explanatory* variable and a dependent variable may be called a *response* variable. In this paper, we address for the first time the problem raised by the presence of outliers in *both* explanatory and response variables in non-linear models. We present an extension of the Robust Polynomial Neural Network (RPNN) [5] inspired by GMDH algorithm, which allows a universal model structure and parameters identification with robust statistics [69, 44].

C.2 Methods

C.2.1 GMDH Approach

The basic principles of iterative GMDH are described as follows (see [150, 96] for more details). Let us call input variables the explanatory variables and output variables the response variables. Let $\mathbf{x} = \{x_1, \ldots, x_m\}$ be the vector of input variables and let y be the output variable that is a function of a subset of input variables $y = u_i(x_{i1}, x_{i2}, \ldots, x_{ip})$. Let us assume that there are n observations of the output variable, such that $\mathbf{X} = \{x_{ij}\}$ is a $m \times n$ matrix of the input variables and $\mathbf{Y} = (y_1, \ldots, y_n)$ the vector of observations of the output variable. A random error ξ affects each observation y_j and we assume that all errors are uncorrelated and identically distributed with finite variance $\mathbf{Y} = E(\mathbf{Y}|\mathbf{x}) + \xi$.

The GMDH considers the class of models $\mathbf{G}$ that are characterized by the following three properties: *(i)* class $\mathbf{G}$ contains *structures* (the term structure is referred to any model with unidentified parameters) that are linear according to the parameters; *(ii)* it exists a known transform $g(\cdot)$ such that $g(f_i, f_j) \in \mathbf{G}$, if $f_i, f_j \in \mathbf{G}$; *(iii)* any element of class $\mathbf{G}$ is either constant, or one of the initial input variables, or it is calculated using the transform function $g(\cdot)$ applied to the other elements of the class. This method is aimed at finding a subset of variables $\{x_{i1}, x_{i2}, \ldots, x_{ik}\}$ and a model belonging to class $\mathbf{G}$ that minimizes some criterion.

In the simplest case $g(\cdot)$ can be defined as $g(f_i, f_j) = \mathrm{a}f_i + \mathrm{b}f_j$ and the class $\mathbf{G}$ consists of linear functions only. The input-output relationship of the analyzed system can be implemented using an artificial neural network (ANN) having a multilayered perceptron-type network structure (see Figure C.1). Each element in the network implements a polynomial function of its inputs. The neurons can be characterized by a transfer function $g(\cdot)$ that is a short-term polynomial of number of variables. For example: in linear case $g(\cdot)$ can be $g(f_i, f_j) = \mathrm{a}f_i + \mathrm{b}f_j$ or in non-linear case one of the possible generators can be $g(f_i, f_j) = \mathrm{a}f_i + \mathrm{b}f_j + \mathrm{c}f_if_j + \mathrm{d}f_i^2 + \mathrm{e}f_j^2$. The composition of quadratic polynomials of latest $g(\cdot)$ forms a high-order regression polynomial known as the Ivakhnenko polynomial. Notice that the degree of the polynomial can double at each next layer of the ANN, when models created by generator function will be used by next-step's generator function. Because of that, control over model complexity is required and usually done by at least rejecting high-degree and/or long-term models.

The GMDH training algorithm is based on an evolutionary principle. The data set is subdivided into training and test sets. At the first layer of the ANN all possible combinations of two inputs generate the first population of neurons according to the transfer function $g(\cdot)$. The size of the population at the first layer is equal to C_m^2. The coefficients of the polynomials of $g(\cdot)$ are estimated by *least*

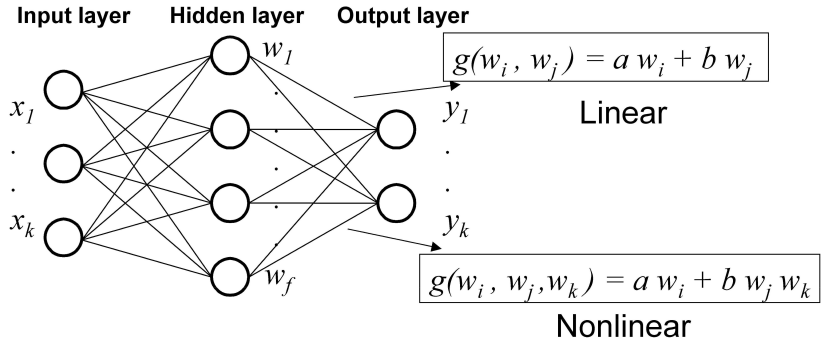


Figure C.1: GMDH Principle. Given a set of input variables $\mathbf{x} = \{x_i, \ldots, x_m\}$ and a set of output variables $\mathbf{y} = \{y_i, \ldots, y_m\}$ GMDH is aimed at finding the models f in the form of polynomial functions such that $y_i = g_1(g_2(\ldots g_k(\mathbf{x})))$.

square fitting using the training set. The best neurons are selected by evaluating the performance on the test set according to a criterion value. The outputs of selected neurons of the first layer are treated as the inputs to the neurons of the second layer, and so on for the next layers, if more than two layers are used. The size of the population of the successive layers become equal to C_f^2. The external iterative procedure controls the complexity of the models, i.e. the number of the terms and the power of the polynomials in the intermediate models. The best models form the initial set for the next iterative procedure. The outcome of such internal iterative procedure is a search for optimal models given the fixed complexity by discarding those models that are out of the specified range. The final models found by GMDH can be obtained in their explicit form by tracing back the path of the polynomials generated during the procedure.

C.2.2 Robust Estimation Criteria

This section explains the application of robust statistics to the criterion value used in GMDH training instead of the originally proposed criterion of the square of residuals. We focus here on robust *linear* regression methods, because GMDH algorithms use strictly linear solver in the core, while non-linear features are introduced by the form of the generator function. The solution of the Equation C.1 or the linear model case is based on the minimization of the following criterion which maximizes the likelihood function over vector of the model parameters β. The classical generalized maximum likelihood-type estimator (M-estimator) for linear regression was proposed by Huber [69] as

$$min_\beta \sum_{i=1}^{n} \rho(r_i) \tag{C.2}$$

where residuals r_i are defined by $r_i = y_i - f(\mathbf{x}, \beta)$, and $\rho(\cdot)$ is a symmetric residuals weight function with single minimum at zero. One can consider ρ as a penalty function for those points which are expected to be outliers in the response variable.

If function $\rho(\cdot)$ is differentiable Equation C.2 can be transformed into

$$\begin{aligned} \sum_{i=1}^{n} \rho'(r_i(\mathbf{x};\beta))r_i(\mathbf{x};\beta) &= \\ \sum_{i=1}^{n} \psi(r_i(\mathbf{x};\beta))r_i(\mathbf{x};\beta) &= 0 \end{aligned} \tag{C.3}$$

where $\psi(r_i(\mathbf{x};\beta))$ is a derivative of the penalty function with respect to the variable β, so $\psi(r_i(\mathbf{x};\beta)) = \rho(r_i(\mathbf{x};\beta))/d\beta$. Equation C.3 can be solved by using the robust Iteratively Reweighted Least Squares method (IRLS) [109]. The iterative estimation of vector β in matrix form is expressed by

$$\hat{\beta}^{i+1} = (\mathbf{X^T}\mathbf{w}^{-1}(\hat{\beta}^i)\mathbf{X})^{-1}\ \mathbf{X^T}\mathbf{w}^{-1}(\hat{\beta}^i)\mathbf{y} \tag{C.4}$$

$$\mathbf{w}^{-1} = min\{\mathbf{1}, 1/|\psi(\mathbf{r})|\} \tag{C.5}$$

where $\mathbf{X}$ is a matrix of n observations of the output variable vector $\mathbf{x}$, $\mathbf{w}^{-1}$ are weights calculated on the base of function $\psi(\cdot)$ and residuals. Iterations are repeated until convergence of estimations of parameters vector $\hat{\beta}^i$. Initial $\hat{\beta}^0$ is found by the Mean Least Squares algorithm. The constraints put on function $\rho(\cdot)$ make the method robust to outliers in the response variable $\mathbf{y}$, but it is still sensitive to outliers in the explanatory variable $\mathbf{x}$.

Yohai [149] introduced MM-estimators with high efficiency and high breakdown point. The method proceeds by finding a highly robust estimate that minimizes an M-estimate of the scale of the residuals (the first M in the method's name) and then the estimated scale is held constant whilst a close-by M-estimate of the parameters is found (the second M) by

$$min_\beta \sum_{i=1}^{n} \rho(r_i/\hat{\sigma}_r). \tag{C.6}$$

An approach used in MM-estimators and S-estimators to obtain highly robust estimates [95] suggested us that in case of outliers in the explanatory variables we could introduce weights function, which take into account also the estimation of real location and scale of the observed explanatory variables $\mathbf{X}$. In this case we should minimize the criterion

$$min_\beta \sum_{i=1}^{n} \psi(r_i(\mathbf{x};\hat{\mu};\hat{\sigma}_x))r_i(\mathbf{x};\hat{\mu};\hat{\sigma}_x)). \tag{C.7}$$

Here $\psi(\mathbf{r}) = \rho(\mathbf{r})/d\beta$ can be considered as a penalty function for those points which are expected to be outliers in explanatory or in response variables or in both at the same time. We used Huber's ψ function, which is

$$\psi(r) = \begin{cases} r & \text{if } abs(r) < c \\ c \cdot sign(r) & \text{if } abs(r) \geq c \end{cases} \tag{C.8}$$

where c is a tuning constant [69].

We assumed that all errors are uncorrelated, so it is natural to define a function $\psi(\cdot)$ as a multiplication of two functions, weighting the outliers independently in

the explanatory and in the response variables. The weight vector for the IRLS methods is constructed in the same way: $\mathbf{w} = \mathbf{w_r}\mathbf{w_x}$ where weights $\mathbf{w_r}$ are associated with response variable and defined as in Equation C.5. The weights $\mathbf{w_x}$ are associated with the data-set formed by the observed explanatory variables and are defined by

$$w_i = min\{1, \sqrt{C_\tau / d_m^2(\mathbf{x})}\} \quad \forall i = \{1, \ldots, n\}, \tag{C.9}$$

where C_τ is a constant tuned for a selected threshold τ of the expected number of outlier points and $d_m^2(\mathbf{x})$ is a Mahalanobis distance [98]. We used of C_τ equal to the value of the inverse of a cumulative density function for a χ^2 distribution with $m-1$ degrees of freedom for point τ [62, 101]. The Mahalanobis distance is defined by

$$d_m^2(\mathbf{x}) = (\mathbf{x} - \hat{\mu}_x)^T \hat{\mathbf{S}}^{-\mathbf{1}} (\mathbf{x} - \hat{\mu}_x) \tag{C.10}$$

where $\hat{\mu}_x$ and $\hat{S}$ are robust estimations of location and covariance of the observed explanatory variables $\mathbf{X}$. In a general case $\mathbf{w_x}$ should be recalculated at each iteration, when robust location $\hat{\mu}_x$ and covariance $\hat{\mathbf{S}}^{-\mathbf{1}}$ are re-estimated. The robust location estimator was computed as the *median* and the robust scale estimator was computed using the pair-wise algorithm based on the Orthogonalized Gnanadesikan and Kettenring algorithm [101, 58].

In the case of outliers in the response variables, in both robust IRLS and original RPNN, the model fit function is the weighted sum of squares of residuals

$$RSS = \frac{1}{n-1} \sum_{i=1}^{n} \rho(r_i / \hat{\sigma}) \tag{C.11}$$

In our new enhanced RPNN (ERPNN) algorithm we take into account possible outliers in the explanatory variables $\mathbf{X}$ and the weighted sum of squares of residuals is expressed by

$$RSS_w = \frac{1}{n-1} \sum_{i=1}^{n} \rho(r_i \cdot w_i^2 / \hat{\sigma}) \tag{C.12}$$

where $\mathbf{w_x} = \{w_i\}, i = \{1, \ldots, n\}$ as in Equation C.9.

Notice that at the very begin of the GMDH algorithm, instead of the regular Akaike criteria used for model selection, we use a modified robust version of the criteria with second-order bias correction [4, 34, 119] with additional leverage point resistant term

$$\begin{aligned} AICr &= \frac{1}{n-k} \sum_{i=1}^{n} \rho(r_i \cdot w_i^2 / \hat{\sigma}) + \frac{n+k}{n-k-2} \\ &= \frac{n-1}{n-k} RSS_w + \frac{n+k}{n-k-2}. \end{aligned} \tag{C.13}$$

where k is a number of terms in a model evaluated.

C.2.3 Enhanced RPNN robust to outliers in explanatory and response variables

The enhanced version of the algorithm is based on GMDH PNN for model and parameters selection, described elsewhere [5], with robust techniques extension

inspired by MM-estimators and S-estimators approach [149]. Let us assume n observations of the explanatory variable, such that $\mathbf{X}' = \{\hat{x}_{ij}\}$ is a $m \times n$ matrix of the observed measurements of the explanatory variables $\mathbf{X}$, $\mathbf{x} \in \mathbb{R}^m$. We assume one response variable $\mathbf{y}$, such that the $1 \times n$ vector of the observed response variable is denoted $\mathbf{y}'$.

The goal is to find a model $y = f(\mathbf{x}; \beta_\mathbf{o}) + \xi$ where $f(\cdot)$ is multinomial non-linear function whose order is known *a priori* to be not higher than p_{max} and that cannot consist of more than t_{max} terms. Taking into account the presence of outliers in both explanatory and response variables we can write $\mathbf{X}' = \mathbf{X} + \xi_x$ and $\mathbf{y}' = \mathbf{y} + \xi$, whre ξ_x and ξ are not correlated and have a distribution that follows a sum of Gaussian distributions with "heavy tails".

The general procedure of the enhanced RPNN algorithm aimed at optimal model selection is the following:

(1) Compute the *robust estimation* of location μ and covariance S of the explanatory data-set $\mathbf{X}'$;
(2) Initialize the vector of weights $\mathbf{w_x}$ based on the estimates $\hat{\mu}$ and $\hat{S}$ for later use by the iterative estimator inside the robust enhanced PNN-core algorithm, as shown in Equation C.9;
(3) Initialize the best models set $M'_{best} = \emptyset$;
(4) For each pair of constraints (t, p) to the *terms* and *power* of the multinomial non-linear function, *i.e.*, $\forall(t, p)$ where $t \in \{1, \ldots, t_{max}\}$ and $p = p_{max}$ do
    (4.1) Run the robust enhanced PNN-core algorithm beginning with models set to M_{start} equal to M'_{best} and with model constraints set to t and p of M'_{best};
    (4.2) Get M_{best} model-set from enhanced RPNN-core algorithm;
    (4.3) Evaluate the models from M_{best} in accordance with the robust fit criteria AICr (given in Equation C.13);
    (4.4) Update M'_{best} with those models found in $M \in M_{best}$, which have lower values of AICr criteria and are different by structure from the current models in M'_{best};
(5) Choose the model $M \in M'_{best}$ that minimizes the AICr criteria, *i.e.*, the best fit model (or use the full set M'_{best} with appropriate AICr values to obtain an ordered set of best fit models, if one wants to search for more than one model).
(6) Obtain *explicit* form of the selected best fit models for future analysis from information associated with model by core of the RPNN algorithm.

Notice that the search for the raw model and parameters selection are performed at the above step (4.1). The major difference of the enhanced algorithm is that an advanced estimator robust to outliers in both explanatory and response variables is used instead of the least square estimator (as described in Section C.2.1) for the linear regression between the terms of model. The generator function $g(f_i, f_j, f_k) = \mathrm{a}f_i + \mathrm{b}f_j f_k$ was used in order to achieve faster computations, without sacrificing the accuracy of the algorithm on quadratic models. Notice that such change of the core linear solver into a fully robust one provoked changes in the fit function and top-level model selection criterion.

The enhanced RPNN-core algorithm searches for best fit models for a given

pair (t, p) of constraints on model terms and order, the correction weight vector $\mathbf{w_x}$, and a list of starting models M_{start} according to the following procedure:

(1) Initialize the working sets:
- set best models $M_{best} = M_{start}$;
- estimation of the best models set $\mathbf{X_{best}}$ in accordance to M_{best};
- expand the working data-set $\mathbf{X_{all}} = [\mathbf{X}; \mathbf{X_{best}}]$;

(2) $\forall\{i, j, k\}$ where $i, j, k \in \{1, \ldots, |\mathbf{X_{all}}|\}$ do

(2.1) Build a model $M_{ijk} = \alpha_1 x^i + \alpha_2 x^j x^k$ where the generator function $G(i, j, k) = \mathbf{x^i} + \mathbf{x^j x^k}$ and $\mathbf{x^p}$ denotes the vector correspoding to the p-th column of matrix $\mathbf{X}$;

(2.2) Reject the model M_{ijk} if it does not fit to terms and order constraints (t, p);

(2.3) Evaluate the linear regression coefficients α_1 and α_2 with IRLS method using the weight vector $\mathbf{w_x}$, such that the vector $\alpha = \{\alpha_1; \alpha_2\}$ satisfies the robust linear regression model

$$y \approx [\mathbf{x^i}; \mathbf{x^j} \cdot \mathbf{x^k}]^T \cdot \alpha;$$

(2.4) Calculate the robust model fit function for model M_{ijk}:

$$\begin{aligned} RSS_w(r) &= \sum_{t=1}^{n} \rho(r_t(M_{ijk}) \cdot w_{xt}^2/\hat{\sigma})) \\ &= \rho(\mathbf{w_x}([\mathbf{x^i}; \mathbf{x^j} \cdot \mathbf{x^k}]^T \cdot \alpha - \mathbf{y})/\hat{\sigma}); \end{aligned}$$

(2.5) Reject the model M_{ijk} if

$$RSSw(r(M_{ijk})) \geq RSSw(r(M_i)), \forall M_i \in M_{best};$$

(2.6) Reject the model M_{ijk} if
$$\begin{cases} \exists M_i \in M_{best} : M_{ijk} \text{ has same structure as } \mathrm{M_i} \\ RSSw(r(M_{ijk})) > RSSw(r(M_i)); \end{cases}$$

(2.7) Include the model M_{ijk} into the M_{best} set and model estimation

$$\mathbf{x_{ijk}} = \alpha_\mathbf{1} \mathbf{x^i} + \alpha_\mathbf{2} \mathbf{x^j x^k} = [\mathbf{x^i}; \mathbf{x^j} \cdot \mathbf{x^k}]^T \cdot \alpha$$

into $\mathbf{X_{best}}$, replacing a model with the same structure if it exists;

(3) Reduce the M_{best} set and $\mathbf{X_{best}}$, given the models with the best RSS_w criteria;

(4) Update the model estimation set $\mathbf{X_{all}} = [\mathbf{X}; \mathbf{X_{best}}]$;

(5) Repeat steps (2) to (4) until convergence of the models.

C.3 Results

The descibed algorithm was tested on artificial data-sets. Models of specified degree and terms number were generated, as well as, appropriate data-sets.

An artificial data-set $\mathbf{X}$ was created in accordance with Gaussian distribution $\eta(0, \sigma_x^2)$, in all tests done $\sigma_x^2 = 10$. An initial model M_{init} satisfying to constraints on model degree and number of terms was generated and evaluated on data-set $\mathbf{X}$, in order to obtain realizations of dependent variable $\mathbf{y} = M_{init}(\mathbf{X})$. An input data-set consist from $\mathbf{X_{fit}}$ and $\mathbf{y_{fit}}$ was forged as follows: explanatory variable realizations were a superposition of "real" data and outliers

$$\mathbf{X_{fit}} = \mathbf{X} + \eta(0, 7\sigma_x^2) = \eta(0, \sigma_x^2) + \eta(0, 7\sigma_x^2) \tag{C.14}$$

And dependent variable realizations were a superposition of "real" output, systematic error ξ, and outliers

$$\mathbf{y_{fit}} = \mathbf{y} + \eta(0, 1) + \eta(0, 3\sigma_y^2) \tag{C.15}$$

When M_{fit} was build by algorithm it was tested on "clear" from outliers data-set with zero mean and with 3 times higher standard deviation than initial data-set has $\mathbf{X_{test}} \sim \eta(0, 3\sigma_x^2)$. In this case we are be able to fight against model over-fitting if it was present. Average of square of residuals (RS) was recorded as performance measure of the fit model

$$RS = \frac{1}{|\mathbf{X_{test}}|} \cdot \sum_{\forall \mathbf{x_i} \in \mathbf{X_{test}}} (M_{fit}(\mathbf{x_i}) - M_{init}(\mathbf{x_i}))^2 \tag{C.16}$$

In all experiments initial data-set $\mathbf{X_{init}}$ was made from 100 points and test data-set was made from other 100 points.

We run three groups of tests on those data. We test general performance and prediction accuracy in comparison with the initial RPNN algorithm robust to outliers in response variable only and with the IRLS algorithm based on fully robust $\rho(\cdot)$ function as described in Section C.2.2. We test model selection quality having two different model fit functions at top-level algorithm. And at the end, we did tests with varying number of outlier points present in the data in order to verify robustness of the algorithm.

General performance of algorithm was estimated by comparison with original RPNN [5] and with IRLS algorithm.

For all three algorithms we did tests with linear, quadratic, and weak-cubic models (latest ones were composed from one term of third degree and other terms from up to second degree). In the case of IRLS algorithm, which is linear by its nature, and models of second and third degree the initial data-set $\mathbf{X_{init}}$ was transformed to $\mathbf{X'_{init}}$ having all possible combinations of initial variables, which give terms of up to second degree and up to third degree appropriately. In all tests initial models were of second order, except general performance linear and cubic tests, and consist from 5 terms (4 terms are made up from 4 available variables plus a constant term). Please note, because of randomness of model generation, it could happen, that some models were composed from not full set of variables, for example: a model $f(\mathbf{x}) = x_3^2 + x_4^2 + x_3 x_4 + 1$ uses only 2 variables out of all, while having 4 terms. The PNN algorithms were limited to search within the models of appropriate degree and consisting from up to 6 terms. In all experiments there were a total of 25 outliers (15 in explanatory $\mathbf{X}$ and 10 in response $\mathbf{y}$). For each condition the input data and the models were generated 200 times and then all three algorithms were run of them. The results are summarized in

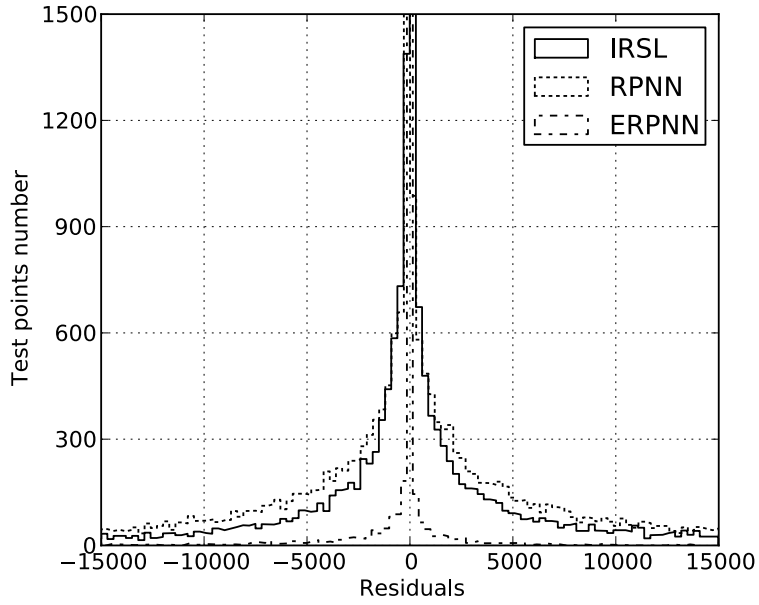


Figure C.2: Distributions of residuals obtained with IRLS (solid line), RPNN (dotted line), and ERPNN (dash-dot line) algorithms for the case of quadratic models. Bin size is 300. Bins corresponding to $[-300; 300)$ interval are truncated at 1500, actual values are: 7749, 4831, and 18637 points for IRLS, RPNN, and ERPNN appropriately.

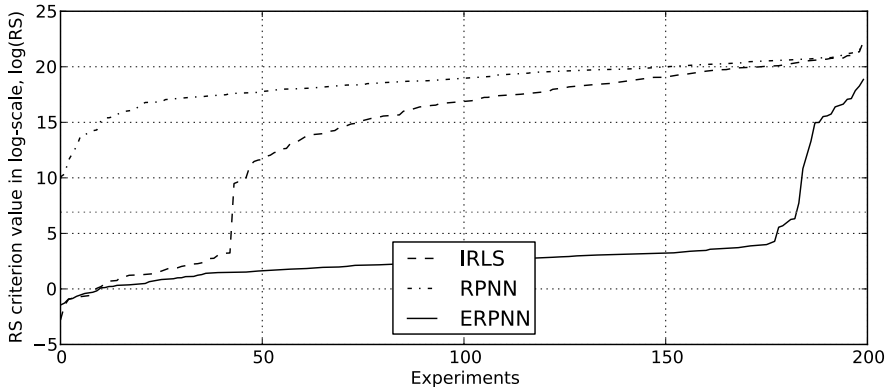


Figure C.3: Comparison between IRLS (dashed line), RPNN (dash-dot line), and Enhanced PRNN (solid line) algorithms. Experiments sorted by values of RS criterion are placed on x-axis and values of RS criterion them-selves are on the y-axis.

Table C.1: General prediction accuracy with 10 outliers in dependent variable and 15 outliers in explanatory variables

Method Used	**RS, Total**	**Fit Models**
Linear models	$mean \pm std$	w. $RS > 10^3$
RPNN	9336.239 ± 44850.23	11.0%
IRLS	424.638 ± 5278.51	1.0%
ERPNN	0.471 ± 0.43	0.0%
Quadratic models	$mean/10^3 \pm std/10^3$	w. $RS > 10^3$
RPNN	$346\,164 \pm 451\,348$	99.5%
IRLS	$198\,490 \pm 453\,752$	78.5%
ERPNN	$2\,106 \pm 13\,404$	8.5%
Cubic models	$mean/10^3 \pm std/10^3$	w. $RS > 10^3$
RPNN	$266\,120\,481 \pm 878\,222\,086$	99.5%
IRLS	$273\,272\,658 \pm 945\,446\,327$	85.5%
ERPNN	$231\,003\,849 \pm 876\,882\,759$	38.5%

Table C.1. For quadratic models case distributions of residuals obtained thorough all experiments for each algorithm are shown on Figure C.2 (residuals with absolute values larger than 15000 are not shown on the figure; number of such points was 2111 (10.56%), 3821 (19.11%), and 34 (0.17%) for IRLS, PRNN, and ERPNN algorithms appropriately) and means of square of residuals (RS criterion) for each algorithm in experiments are shown on Figure C.3. On the histogram plot bins corresponding to $[-300; 300)$ interval holding: 7749 (38.75%), 4831 (24.16%), and 18637 (93.18%) points for IRLS, RPNN, and ERPNN.

Please note, that for quadratic and cubic cases because of initial data-set (with dimensions of 100×5) expansion, it became a data-set $\mathbf{X}'_{\mathbf{fit}}$ with dimensions 100×15 and 100×35 appropriatly.

To measure model selection features of the algorithm we expanded model search space allowing to look up though models consist from up to 12 terms (instead of default 6 and thus 7 terms more than actual models have). Two versions of the algorithm were used: first as described above with AICr as model fit function and second with plain RSS_w in this role. Please note, improvement of the value of the fit criteria was achieved with longer models allowed, which gave 2 models more (1% more) with low values of RS criteria. The results are summarized in Tables C.2 and C.3.

Sensibility to the outlier quantity was tested by running algorithm on models with $N_O = \{5, 10, 20\}$ outliers in the response variable and $N_L = \{0, 5, 10, 15, 20, 25\}$ outliers in the explanatory variables. Most illustrative results are summarized in Table C.4. Number of models with high values of RS criterion as function of N_O and N_L is summarized on Figure C.4. Change of the RS criterion through experiments for selected cased are depicted on Figure C.5. Please note, that algorithm always remained tuned for 25% of outliers (i.e. the value of the constant C_τ was the same across all experiments).

Table C.2: Enhanced PRNN model selection performance with AICr and RSS_w model fit criterions and maximal model length of 6 or 12 terms.

Fit Criteria (max terms allowed)	**Fit Models** w. $RS > 10^3$	**Average model length** (terms in model)
AIC_r (T6)	8.5%	5.11
AIC_r (T12)	7.5%	5.74
RSS_w (T12)	7.5%	10.56

Table C.3: Enhanced PRNN fit model quality with AICr and RSS_w model fit criterions and maximal model length of 6 or 12 terms

Fit Criteria (max terms allowed)	**RS, best 80%** $mean \pm std$	**RS, worst 20%** $mean/10^3 \pm std/10^3$
AIC_r (T6)	10.80 ± 8.22	$10\,532 \pm 28\,453\,546$
AIC_r (T12)	12.31 ± 8.89	$6\,536 \pm 16\,842\,335$
RSS_w (T12)	15.75 ± 9.95	$6\,536 \pm 16\,842\,333$

C.4 Discussion

From the Table C.1 it is seen that RPNN can not cope with such kind and amount of outliers in the case where models are not linear. For the quadratic case only 1 model out of 200 was found correctly by RPNN, while by Enhanced RPNN – 17 models were *not* found correctly. Comparison with robust IRLS reveals that it is not as good as ERPNN on linear tasks. Thought both are very good, they were able to find good approximations for more than 99% of models. Although, ERPNN is noticeably better than IRLS in handling higher-degree models if one apply initial matrix expansion approach for IRLS. Obviously this approach leads to bad-conditioned initial data matrices and much higher impact of outliers, especially, when for high-degree models. Enhanced RPNN, because of its GMDH origins, is less affected by bad conditioned data matrices, as it takes a combination of only three variables at each step and model fit function can coupe with that in more effective way, than IRLS.

As one can see from the Figure C.3 and the Figure C.5 fit models build by algorithm are quite accurate until certain point, after which there is a great "jump" of RS criterion to the values of 10^6 and higher. This happen when the algorithm is failed to find correctly one or more terms of actual model. One absent term, when multiplied by the order of model and, usually, replaced with another inappropriate high-order term, will cause such values of RS criterion. In other cases fit model can have inaccurate parameters estimations and/or excessive terms with coefficients close to zero, but according to our experience this will cause relatively small deviations in RS criterion up to around 10^3.

Please note, that the results of the linear and the cubic cases are given for the purpose of raw comparison with the quadratic case. One should take in to account that RS criterion is rising far much slower in the linear case and far much faster in the cubic case.

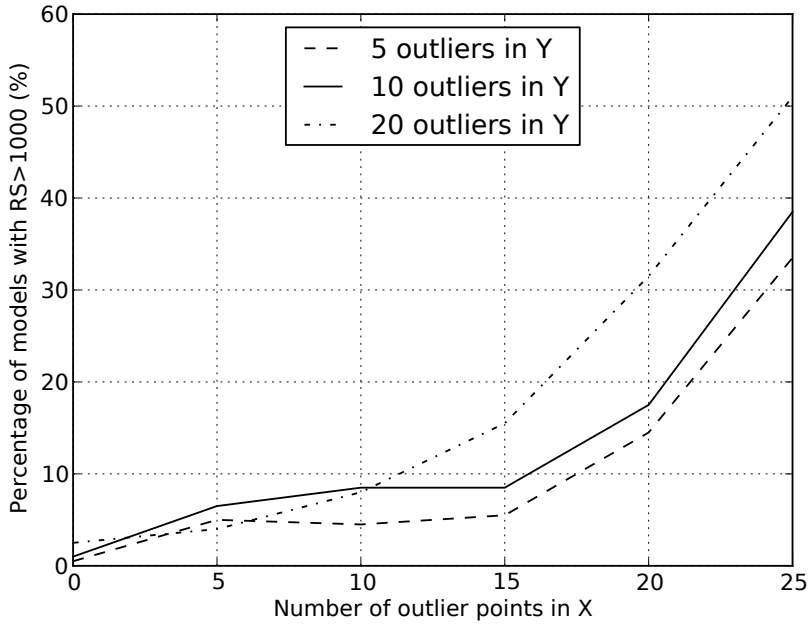


Figure C.4: Percentage of models with high values of RS criterion ($RS > 10^3$) found with ERPNN algorithm as a function of number of outlier points in explanatory variables (x-axis) varying from 0 to 25 and in response variable: 5 (dashed line), 10 (solid line), and 20 (dash-dot line).

From the Figure C.4 we can see a dependency between number of the outliers and number of models with high RS criterion ($RS > 10^3$). Because of algorithm tuning to 25% of outliers present in the initial data, occasionally, it can behave better on data contain more outliers, thought it can depend on the ratio between outliers in the explanatory variables and in the response variable.

The difference between all three methods is best seen on the Figure C.2, which shows residuals distribution. From the figure one can see that majority of points are in the region close to zero for all algorithms, thought the distribution of residuals of ERPNN algorithm has much more points close to 0 (93% points) in comparison to any other method (39% for IRLS and 24% for RPNN). In consequence, distributions of residuals of models found by another algorithms have much heavier "tails", which means more wrongly selected models were used or more purely estimated coefficients were found.

ERPNN preserves one important feature of the original PNN algorithm – an explicit form of fit models are automatically found by the algorithm. This feature of the algorithm is not really discussed in this paper, except for the length of models found with different fit criterions. As it is seen from the Table C.2 with two times longer models allowed to search through we can decrease number of models not found correctly by 11% in comparison with standard case of 6 term models allowed. Also it is clearly seen that Akaike based model fit criterion can achieve at least the same results in terms of RS fit criterion values, while having almost two times shorted models – 5.74 terms in average instead to 10.56 terms in avarage for RSS_w case. That will reduce possibility of over-fittting given by RSS_w criterion. We can see in the Table C.3, that models found with RSS_w have higher value of RS criterion on the 80% best models, than those found with AICr criterion, which

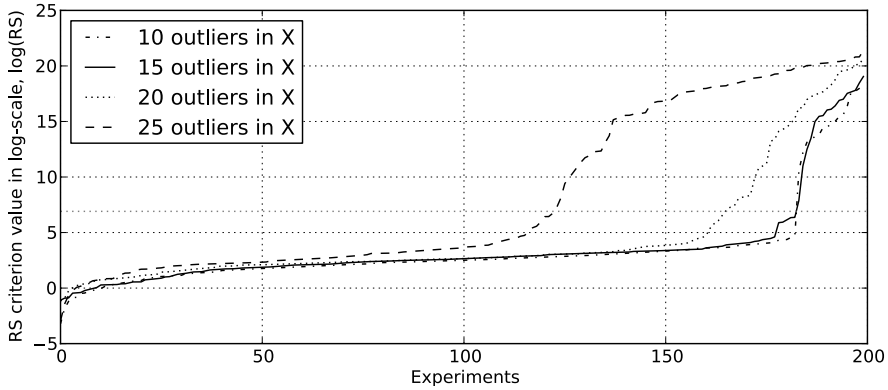


Figure C.5: Robustness of ERPNN to the ratio of outlier points present in the regression data-set. Experiments, sorted by value of RS criterion, are placed on x-axis. Values of RS criterion are on the y-axis in the logarithmic-scale. 10 outliers were always present in the response variable and 10 (dash-dot line), 15 (solid line), 20 (dotted line), and 25 (dashed line) outliers were present in the explanatory variables.

Table C.4: Sensibility to the ratio of the outlier points present in the regression data-set.

Outliers number in		**RS, total**	**Fit Models**
in X	in Y	$mean/10^3 \pm std/10^3$	w. $RS > 10^3$
0	10	1.92 ± 19.26	1.0%
5	10	193.1 ± 931.9	6.5%
10	10	$1\,128 \pm 7\,503$	8.5%
15	5	$1\,227 \pm 8\,922$	5.5%
15	10	$2\,106 \pm 13\,404$	8.5%
15	20	$7\,795 \pm 33\,049$	15.5%
20	10	$15\,051 \pm 75\,973$	17.5%
25	10	$67\,459 \pm 177\,511$	38.5%

proves that longer models found by the algorithm with RSS_w criterion are worth than those found with $AICr$ criterion. Please note, that IRLS method does not do model selection at all it always produce models with all possible terms made from a combination of all initial variables.

Last, but not least, we discuss some issues associated to the practical implementation of the algorithm. As mentioned above the correction weights $\mathbf{w_x}$ should be recalculated at each iteration, when robust location $\hat{\mu}_x$ and covariance $\hat{S}$ are re-estimated. In our implementation they are calculated only once and are used in all runs of the IRLS algorithm for linear regression in the core of the PNN algorithm. In our implementation we sacrifice somehow the algorithm's precision in favor of calculation speed. We observed that gain of speed was worth of precision lost, but we are investigating further to determine in which cases this could pose a problem.

Several ways are still open to improve performance of the algorithm. First of

all quality of the outlier detection can be greatly improved if pair-wise algorithm for robust covariance estimation will be replaced by more complex one which will take into account all variables at whole, and not two of them at once, as it is done in pair-wise algorithm, like OGK. From the other side such algorithms are much more computational heavy than pair-wise ones, so if computational speed is really important it cannot be advised. Our research was quite limited to the class of the quadratic models, so more deep analysis should be done on algorithm's performance on models of higher-order. The generator function $g(f_i, f_j, f_k) = \mathrm{a}f_i + \mathrm{b}f_j f_k$ used is very good for quadratic models' class, but it is also known to have some limitations when applied to higher-order polynomes. More additional studies should be done on selection the generator function in terms of trade off between computation speed and prediction accuracy for other higher-degree polynoms. By the design original GMDH-algorithm has problems in the case of the high-order models when higher-degree terms will completely outperform low-degree terms of a model and thus later ones always will be lost during search. Possible outliers make the solution in the case even harder and more work on adoption of fit criterion should be done when terms with greatly different degree appear during model estimation by the core of the PNN algorithm. Proposed enhanced RPNN preserves good accuracy of the automatic structure synthesis of its predecessor and offers robustness to outliers in both explanatory and dependent variables.

Part III

Conclusions

Chapter 9

Final chapter

I may not have gone where I intended to go, but I think I have ended up where I needed to be.

Douglas Adams

Résumé :

Notre conclusion montre traite de tous les résultats obtenus et des chemins possibles pour de futurs travaux. Quelques remarques sur le déroulement du travail sont présentées. Elles mettent l'accent sur les contributions pratiques et théorétiques de la simulation de réseaux neuronaux de grande taille, ainsi que sur les difficultés liées à cette simulation.

This work has been conducted through a number of years. During that long period some decisions about the contents of the Thesis were taken well in advance and were well planned, while others were taken on the fly and developed in artistic way. In this Chapter we will conclude all the work done in the frame of the Thesis and then we will point out directions for possible future work.

9.1 Conclusions

In the frame of this Thesis:

- a novel hierarchical evolutionary SNN simulator was created;
- an unique agent-oriented framework targeted on mobile platforms was developed;
- we showed the evidence that hierarchical neural circuits we model exhibit behavior similar to that observed in real humans;
- a role of inter-layer and intra-layer reciprocal inter-connections in hierarchical neural circuits was studied and the results bring the light on characteristic activity patterns associated with particular type of reciprocal connection;

- a multi-variative regression algorithm, featuring automatic model selection and robust estimation of model parameters in the presence of outliers in *both* explanatory and response variables, was developed and tested on artificial data.

Novel hierarchical evolutionary SNN simulator supports a rich set of biologically plausible features including: excessive cell death during early development phases (apoptosis process), evolvable synapses via Spike-timing-dependent plasticity (STDP), cell death and synaptic pruning, as a results of synapse depression, and neurons generated bio-electrical field recordings (electrochipography) similar by its properties to the real electroencephalography (EEG) or Local Field Potentials (LFP). Its distinct among other simulators by a unique combination of cutting edge simulation features united in the one piece of software. These most important features are hierarchical neural circuit modeling, which allow to reflect information procession areal-chains of real brains, evolutionary features of circuits, including circuit genomic representation, natural selection over circuit generations and genetic mutations of genome, and, finally, an ability to do bio-electrical signal recordings from virtual electrodes installed on the modeled circuit's surface (electrochipography (EChG)). Despite the fact that current models are rather simple ones we had shown that our simulator is capable to produce biologically plausible output and it could be used for analysis of a large number of experiments originating from real neuro-physiology.

An unique agent-oriented framework can run a swarm of "small" computing modules (like PERPLEXUS's Ubichip-accelerated platform or java-enabled smartphones) reconfiguring dynamically network's topology depending on the agents available. Every agent of robotic system bear a Spiking Neural Network (SNN), which can be used for environment perception and action control. Although a possibility as such to run the simulator on a large number of computational modules in dynamic topologies, an overload produced by currently used synchronous data transmission approach will be noticeable in large systems and more efficient solutions should be studied. Notably, we believe that asynchronous data-transmission approaches will improve the situation, but this kind of research is left for future work.

In the Article B: "Stimuli-driven functional connectivity" **we showed the evidence that our hierarchically organized neural networks exhibit behavior similar to that observed in real humans.** The third order spectral analysis of both electrochipography and electroencephalography allowed us to determine the frequency range of quadratic phase coupling (resonant frequency) across cortical areas [140, 141]. A remarkable result is that modeled circuits during later developmental stages, just like the patients after treatment during sleep phases, were characterized by lower values of Index of Resonant Frequencies (IRF) than during earlier stages and before treatment accordingly. Both an appropriate stimulation of the circuit and the cognitive brain therapy appear to modify the IRF provoking a shift of the indexes towards low frequencies at all brain states. Which means that, according to the usual interpretation based on standing waves theory, information processing is transmitted at long distances such increasing cross-areal involvement in neural processing.

In the Section 7.2 **we depicted a study of a role of reciprocal connections in hierarchical circuits on an example of four topologies characterized**

by different combination of reciprocal connection types present and we have found activity patterns characteristic for each type of connection. In particular, intra-layer projections suppress cross-areal synchronization of a brain, forcing the circuit to exhibit stronger stimulus evoked activity, while absence of such projections will result in weaker reaction to the stimulus and amplification of inner-state evoked activity. Although, strength and time of exposure of the behavior is topology dependent, these findings prove presence of the non-superwised learning in the circuits and a potential possibility to achieve supervised-like learning by introduction of the topological changes.

A number of data-analysis techniques were developed for application to the bio-electrial signal recorded from the modeled neural networks. They were ranging from simple Power Spectral Density (PSD) analysis to the bispectral analysis and the non-linear robust regression analysis approaches. **We created a powerfull multi-variative regression algorithm featuring automatic model selection and following robust parameters estimation, in the presence of outliers in *both* explanatory and response variables**. In all artificial test runs the developed Enhanced Robust Polynomial Neural Network (ERPNN) approach has shown a really good performance and a robustness to outliers in either explanatory or response variables of the data. Although, the proposed method was really performant on artificial data tests, there are still many options to improve its performance. But this is a subject for a next Section.

9.2 Future work

As it has been told, there are several ways to increase the performance of the ERPNN approach: first, outlier detection can be greatly improved if *pair-wise* algorithm for robust covariance estimation will be replaced by a more sophisticated one, which will analyze also information present in the variables as whole, and not just in the pairs of them, as it is done in our current implementation. Other possibilities lie in the adaptation of the generator function in terms of trade off between computation speed and prediction accuracy for higher-degree models. The final note about ERPNN is: although it has good results on the artificial data, the real work will start when it will be tested on a real-world data.

Now let's back to the simulator framework perspectives. From the SNN modeling point of view through the years Leaky Integrate-and-Fire (LIF) neuro-mimetic model was pressed by the Izhikevich's and Hodgkin-Huxley's models [80], first of all from the point of bio-plausibility of the observed emerging activity. The architecture of the simulator allows relativly easy integration of the new models of neurons. It would be really interesting to compare results of experiments based on different neural model, which will give more insights on how, when and which model could be and should be used.

Larger and more complex topologies could be studied with the current state of the simulator, but a lot of research work should be done first on the scaling of such areal neural networks. We saw in our experiments, that even with few modules hierarchical network behavior is very different from the one expected from the stand-along neural network of the same number of cells in total. But given good model parameters setup, the simulations where billions of cells organized in the biologically plausible information processing chains will undoubtfully give very precise information on the processes taking place in the real neural systems and thus probably could be used to predict effect of new drugs or treatments on human's neural system.

At the moment of writing this section we finished development of large multi-layered hierarchical simulation-starter. Now the topologies with arbitrary number of agents per layer and arbitrary number of layers can be started easily. Horizontal and feed-backward reciprocal projections could be switched on and off and motoric layer influence could be studied by adding feed-back projections from that layer to selected processing or sensory layers.

More complex input stimulus application protocol should be implemented in order to be able to reproduce and then to compare results of real experiments. The first steps are already done in this direction, by introducing Go/NoGo-like stimulus filter, which is in conjunction with motoric-layers' feed-back will allow to setup really interesting experiments.

As our reciprocal projection study in Section 7.2 testify, more sophisticated analysis methods should be applied to recordings to extract useful knowledge from the simulation results. Simple PSD and Evoked Potential (EP) are not sufficient to obtain proofs of the behavior differences in the current state of the model, when it is plagued by high frequency noise. Non-linear methods, like third order spectral analysis (Section 6.3) or non-linear regression methods, as Robust Polynomial Neural Network (RPNN), could be usefull and are very promising to that kind of studies. Another possible solution could be to tune the model

to produce really biologically plausible output, i.e. shifted to the low-frequency domain in comparison to the currently observed activity.

The SNN could be used to perform recognition or prediction tanks, which was demonstrated by number of works [85, 86]. Many learning strategies are based on the evolutionary principles. Currently our simulation framework was used only to obtain biologically plausible data from modeling, but it is not only possible application of the framework. Mutation and specie selection evolutionary features of the framework in conjunction with complex stimulation protocols could be used to create predictors and recognizers, and, what is even more important, it will be possible to study how exactly the learning is going on the cellular level in these biologically plausible networks. That applications have huge potential for comprehension of human's memorization processes and thus could be a first step for development of the real-world learning catalysts.

From the framework development point of view current implementation already offers a lot of features. While every single feature is not unique to our framework, they are also present in simulators available [60, 19, 13], the combination of the features proposed is absolutely unique, there is no any other simulation framework, which features distributed simulations, hierarchically organized modular network simulations, evolutionary simulations with mutation, replication and selection support, and finally – a hallmark of the framework – bio-electric EChG recordings. The another one really interesting and promising development direction is implementation of a version of the simulator well optimized for parallel computational architectures, such as graphic-chip arrays, which could give a huge rise of 10-100 times of modeling speed while preserving current network dimensions, which will fasten and ease an access to new even more sophisticated data.

Finally a powerful and feature-rich framework is created to stimulate spiking neural networks organized into hierarchical neural circuits with evolutionary features and to study emerging behaviors of these circuits. It is waiting to be applied to even more complex cognitive and behavioral problems in the domain of neuroscience.

Appendices

List of acronyms

aBIS	auto-bispectrum
AI	Artificial Intelligence
ANN	Artificial Neural Network
BS	Base Stimulus
COBA	conductance-based
DFT	Discrete Fourier Transform
DNA	deoxyribonucleic acid
DSS	decision support systems
EChG	electrochipography
ECoG	electrocorticography
EEG	electroencephalography
EHNN	Evolving Hierarchical Neural Network
e-PSP	excitatory postsynaptic potential
EP	Evoked Potential
ERPNN	Enhanced Robust Polynomial Neural Network
ERP	Event-related Potential
ESD	Energy Spectral Density
FBH	feed-backward with horizonal reciprocal connections
FB	pure feed-backward
FFH	feed-forward with horizonal reciprocal connections
FFT	Fast Fourier Transform
FF	pure feed-forward
FRM	maximum firing rate

GA Genetic Algorithms

GMDH Group Method of Data Handling

IF Integrate-and-Fire

IIR Infinite Impulse Response

i-PSP inhibitory postsynaptic potential

IP Internet Protocol

IRF Index of Resonant Frequencies

ISI Inter-Stimulus Interval

JADE Java Agent Development Environment

LFP Local Field Potentials

LIF Leaky Integrate-and-Fire

LTD long-term depression

LTP long-term potentiation

MLS mean least squares

NA Neural Network Agent

NMM neural mass model

PSD Power Spectral Density

PSP post-synaptic potential

RNA ribonucleic acid

RPNN Robust Polynomial Neural Network

SNG Scriptable Network Graphics

SNN Spiking Neural Network

SNR Signal-to-Noise Ratio

STDP Spike-timing-dependent plasticity

SVM support vector machines

WSS Wide Sense Stationary

xBIS cross-bispectrum

xCoh cross-coherence

List of Figures

3.1 Schematic neuron connectivity map 14
3.2 Membrane potential dynamics 14
3.3 Network's synaptogenesis 15
3.4 Cell death probability during the apoptosis 17

A.1 Early developmental phases of a neural network 27
A.2 Sample network topology with one sensory module 28
A.3 Activity mapping scheme 29
A.4 Electrode's sensitivity area and its sample output 30

4.1 Hierarchical network projected on a brain map 34
4.2 Irregular topology example 36
4.3 Connection establishment algorithm 38
4.4 Topology evolution in a dynamic system 39
4.5 Principal static topologies 41
4.6 Three processing layers topology 42
4.7 Spatiotemporal structure of a base stimulus 43
4.8 Stimulus variability procedure 44
4.9 Stimulation flow and stimulus structure 44
4.10 Neural system development stages 46
4.11 Hierarchical Simulator Architecture 48

6.1 Spike-train raster plot 63
6.2 Decaying amplitude of the oscillations 64
6.3 Plain EChG trials and corresponding Evoked Potential 65
6.4 The power spectral densities of sample channels 70
6.5 The bispectrum of both sample channels 71
6.6 Coupled frequencies for both channels 71

B.1 Ubinet hierarchical circuit 76
B.2 Averaged evoked potentials 77
B.3 Evoked Potentials and Power Spectrum Densities 78
B.4 Relative distribution of the frequencies of resonance 80

7.1 PSDs for LF spontaneous activity stimulations 86
7.2 PSDs for HF spontaneous activity stimulations 87
7.3 PSDs for 300Hz spontaneous activity stimulations 88
7.4 Apoptosis driven cell death landscape 90
7.5 Principal topologies 92
7.6 Average oscillation amplitude 94

7.7 EP plots for 4 circuits 95
7.8 ERP images for FB and FF circuits 97
7.9 ERP images for FBH and FFH circuits 98
7.10 PSD of activity within 4 topologies 99
7.11 PSD of activity within 4 topologies 100
7.12 Cross-coherence of activity within 4 topologies 101

C.1 GMDH Principle 111
C.2 Residuals distribution for quadratic models 117
C.3 IRLS, RPNN and EPRNN performance evaluation 117
C.4 Percentage of models with high RS criterion 120
C.5 Robustness of ERPNN to outliers 121

List of Tables

5.1 Genome coding and mutation example 55

6.1 Comparison of spectral and bispectral functions 69

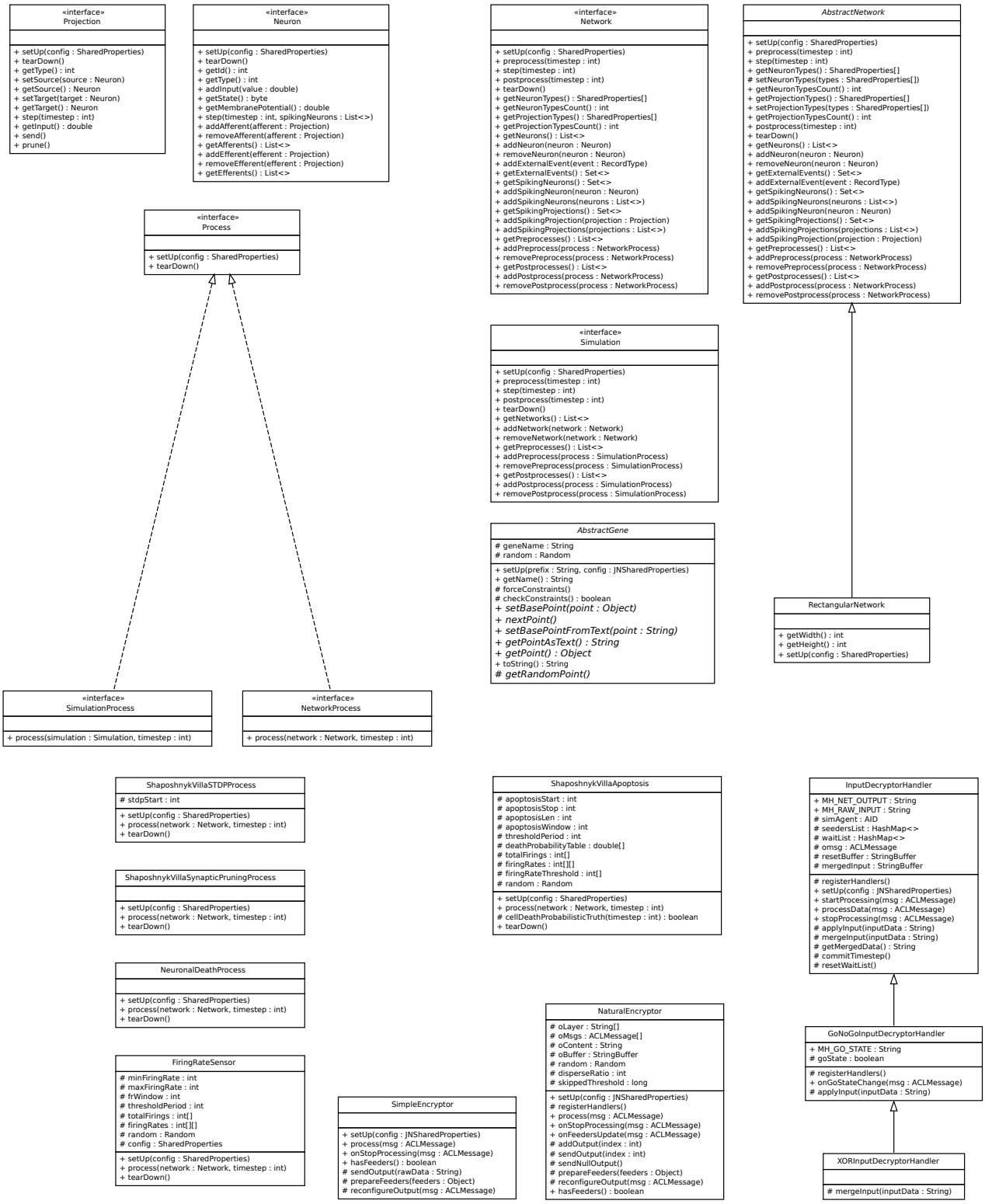
B.1 Phase-coupled frequencies per band for simulated networks 80
B.2 Phase-coupled frequencies per band for patients 81

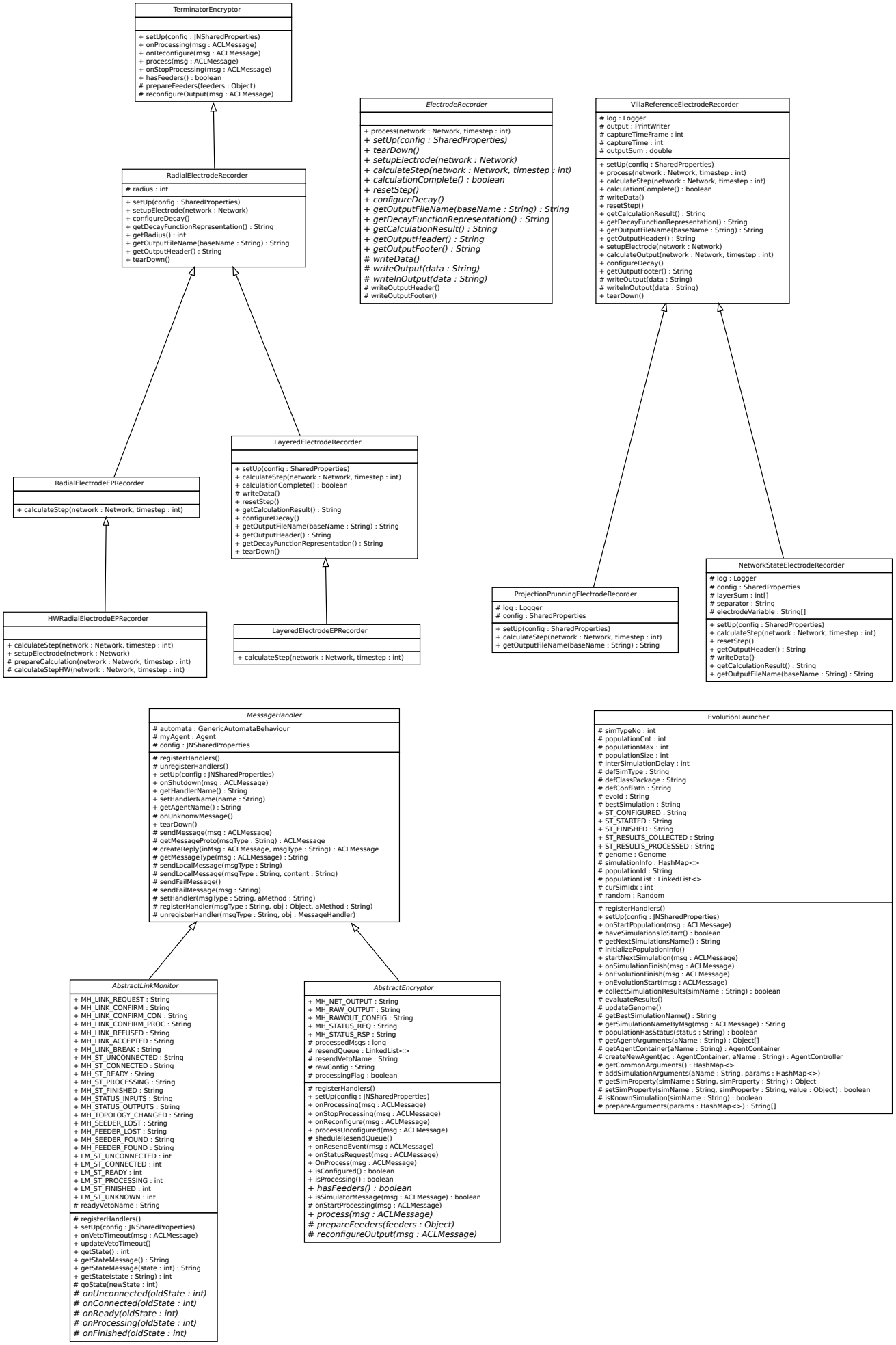
7.1 Model parameters summary 85
7.2 Apoptosis driven cell death 89
7.3 SR confidence intervals 96

C.1 Prediction accuracy comparison 118
C.2 Enhanced PRNN model selection performance 119
C.3 Enhanced PRNN fit model quality 119
C.4 Enhanced RPNN sensibility to the outliers 121

Class Diagrams of the Framework

The whole set of classes of the developed framework contains 157 Java-classes grouped into 23 packages, not counting those from important libraries, *i.e.* JADE, and stand-alone analysis tools developed during the Thesis's venue. That amount of logical structures and relationships cannot be easily depicted on the paper. Thus, the diagrams of this Appendix are neither complete, nor self-sufficient, they are given here with only purpose to briefly depict *very general frame* of the developed software.


«interface»
Projection
+ setUp(config : SharedProperties)
+ tearDown()
+ getType() : int
+ setSource(source : Neuron)
+ getSource() : Neuron
+ setTarget(target : Neuron)
+ getTarget() : Neuron
+ step(timestep : int)
+ getInput() : double
+ send()
+ prune()
«interface»
Neuron
+ setUp(config : SharedProperties)
+ tearDown()
+ getId() : int
+ getType() : int
+ addInput(value : double)
+ getState() : byte
+ getMembranePotential() : double
+ step(timestep : int, spikingNeurons : List<>)
+ addAfferent(afferent : Projection)
+ removeAfferent(afferent : Projection)
+ getAfferents() : List<>
+ addEfferent(efferent : Projection)
+ removeEfferent(efferent : Projection)
+ getEfferents() : List<>
«interface»
Network
+ setUp(config : SharedProperties)
+ preprocess(timestep : int)
+ step(timestep : int)
+ postprocess(timestep : int)
+ tearDown()
+ getNeuronTypes() : SharedProperties[]
+ getNeuronTypesCount() : int
+ getProjectionTypes() : SharedProperties[]
+ getProjectionTypesCount() : int
+ getNeurons() : List<>
+ addNeuron(neuron : Neuron)
+ removeNeuron(neuron : Neuron)
+ addExternalEvent(event : RecordType)
+ getExternalEvents() : Set<>
+ getSpikingNeurons() : Set<>
+ addSpikingNeuron(neuron : Neuron)
+ addSpikingNeurons(neurons : List<>)
+ getSpikingProjections() : Set<>
+ addSpikingProjection(projection : Projection)
+ addSpikingProjections(projections : List<>)
+ getPreprocesses() : List<>
+ addPreprocess(process : NetworkProcess)
+ removePreprocess(process : NetworkProcess)
+ getPostprocesses() : List<>
+ addPostprocess(process : NetworkProcess)
+ removePostprocess(process : NetworkProcess)
AbstractNetwork
+ setUp(config : SharedProperties)
+ preprocess(timestep : int)
+ step(timestep : int)
+ getNeuronTypes() : SharedProperties[]
# setNeuronTypes(types : SharedProperties[])
+ getNeuronTypesCount() : int
+ getProjectionTypes() : SharedProperties[]
+ setProjectionTypes(types : SharedProperties[])
+ getProjectionTypesCount() : int
+ postprocess(timestep : int)
+ tearDown()
+ getNeurons() : List<>
+ addNeuron(neuron : Neuron)
+ removeNeuron(neuron : Neuron)
+ getExternalEvents() : Set<>
+ addExternalEvent(event : RecordType)
+ getSpikingNeurons() : Set<>
+ addSpikingNeurons(neurons : List<>)
+ addSpikingNeuron(neuron : Neuron)
+ getSpikingProjections() : Set<>
+ addSpikingProjections(projections : List<>)
+ addSpikingProjection(projection : Projection)
+ getPreprocesses() : List<>
+ addPreprocess(process : NetworkProcess)
+ removePreprocess(process : NetworkProcess)
+ getPostprocesses() : List<>
+ addPostprocess(process : NetworkProcess)
+ removePostprocess(process : NetworkProcess)
«interface»
Process
+ setUp(config : SharedProperties)
+ tearDown()
«interface»
Simulation
+ setUp(config : SharedProperties)
+ preprocess(timestep : int)
+ step(timestep : int)
+ postprocess(timestep : int)
+ tearDown()
+ getNetworks() : List<>
+ addNetwork(network : Network)
+ removeNetwork(network : Network)
+ getPreprocesses() : List<>
+ addPreprocess(process : SimulationProcess)
+ removePreprocess(process : SimulationProcess)
+ getPostprocesses() : List<>
+ addPostprocess(process : SimulationProcess)
+ removePostprocess(process : SimulationProcess)
AbstractGene
geneName : String
random : Random
+ setUp(prefix : String, config : JNSharedProperties)
+ getName() : String
# forceConstraints()
# checkConstraints() : boolean
+ setBasePoint(point : Object)
+ nextPoint()
+ setBasePointFromText(point : String)
+ getPointAsText() : String
+ getPoint() : Object
+ toString() : String
# getRandomPoint()
RectangularNetwork
+ getWidth() : int
+ getHeight() : int
+ setUp(config : SharedProperties)
«interface»
SimulationProcess
+ process(simulation : Simulation, timestep : int)
«interface»
NetworkProcess
+ process(network : Network, timestep : int)
ShaposhnykVillaSTDPProcess
stdpStart : int
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
+ tearDown()
ShaposhnykVillaSynapticPruningProcess
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
+ tearDown()
NeuronalDeathProcess
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
+ tearDown()
FiringRateSensor
minFiringRate : int
maxFiringRate : int
frWindow : int
thresholdPeriod : int
totalFirings : int[]
firingRates : int[][]
random : Random
config : SharedProperties
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
+ tearDown()
ShaposhnykVillaApoptosis
apoptosisStart : int
apoptosisStop : int
apoptosisLen : int
apoptosisWindow : int
thresholdPeriod : int
deathProbabilityTable : double[]
totalFirings : int[]
firingRates : int[][]
firingRateThreshold : int[]
random : Random
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
# cellDeathProbabilisticTruth(timestep : int) : boolean
+ tearDown()
InputDecryptorHandler
+ MH_NET_OUTPUT : String
+ MH_RAW_INPUT : String
# simAgent : AID
# seedersList : HashMap<>
# waitList : HashMap<>
# omsg : ACLMessage
# resetBuffer : StringBuffer
# mergedInput : StringBuffer
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ startProcessing(msg : ACLMessage)
+ processData(msg : ACLMessage)
+ stopProcessing(msg : ACLMessage)
# applyInput(inputData : String)
# mergeInput(inputData : String)
# getMergedData() : String
# commitTimestep()
# resetWaitList()
NaturalEncryptor
oLayer : String[]
oMsgs : ACLMessage[]
oContent : String
oBuffer : StringBuffer
random : Random
disperseRatio : int
skippedThreshold : long
+ setUp(config : JNSharedProperties)
# registerHandlers()
+ process(msg : ACLMessage)
+ onStopProcessing(msg : ACLMessage)
+ onFeedersUpdate(msg : ACLMessage)
# addOutput(index : int)
# sendOutput(index : int)
# sendNullOutput()
# prepareFeeders(feeders : Object)
# reconfigureOutput(msg : ACLMessage)
+ hasFeeders() : boolean
GoNoGoInputDecryptorHandler
+ MH_GO_STATE : String
# goState : boolean
# registerHandlers()
+ onGoStateChange(msg : ACLMessage)
# applyInput(inputData : String)
SimpleEncryptor
+ setUp(config : JNSharedProperties)
+ process(msg : ACLMessage)
+ onStopProcessing(msg : ACLMessage)
+ hasFeeders() : boolean
# sendOutput(rawData : String)
# prepareFeeders(feeders : Object)
# reconfigureOutput(msg : ACLMessage)
XORInputDecryptorHandler
mergeInput(inputData : String)


TerminatorEncryptor
+ setUp(config : JNSharedProperties)
+ onProcessing(msg : ACLMessage)
+ onReconfigure(msg : ACLMessage)
+ process(msg : ACLMessage)
+ onStopProcessing(msg : ACLMessage)
+ hasFeeders() : boolean
# prepareFeeders(feeders : Object)
# reconfigureOutput(msg : ACLMessage)
ElectrodeRecorder
+ process(network : Network, timestep : int)
+ setUp(config : SharedProperties)
+ tearDown()
+ setupElectrode(network : Network)
+ calculateStep(network : Network, timestep : int)
+ calculationComplete() : boolean
+ resetStep()
+ configureDecay()
+ getOutputFileName(baseName : String) : String
+ getDecayFunctionRepresentation() : String
+ getCalculationResult() : String
+ getOutputHeader() : String
+ getOutputFooter() : String
# writeData()
# writeOutput(data : String)
# writeInOutput(data : String)
# writeOutputHeader()
# writeOutputFooter()
VillaReferenceElectrodeRecorder
log : Logger
output : PrintWriter
captureTimeFrame : int
captureTime : int
outputSum : double
+ setUp(config : SharedProperties)
+ process(network : Network, timestep : int)
+ calculateStep(network : Network, timestep : int)
+ calculationComplete() : boolean
# writeData()
+ resetStep()
+ getCalculationResult() : String
+ getDecayFunctionRepresentation() : String
+ getOutputFileName(baseName : String) : String
+ getOutputHeader() : String
+ setupElectrode(network : Network)
+ calculateOutput(network : Network, timestep : int)
+ configureDecay()
+ getOutputFooter() : String
# writeOutput(data : String)
# writeInOutput(data : String)
+ tearDown()
RadialElectrodeRecorder
radius : int
+ setUp(config : SharedProperties)
+ setupElectrode(network : Network)
+ configureDecay()
+ getDecayFunctionRepresentation() : String
+ getRadius() : int
+ getOutputFileName(baseName : String) : String
+ getOutputHeader() : String
+ tearDown()
LayeredElectrodeRecorder
+ setUp(config : SharedProperties)
+ calculateStep(network : Network, timestep : int)
+ calculationComplete() : boolean
# writeData()
+ resetStep()
+ getCalculationResult() : String
+ configureDecay()
+ getOutputFileName(baseName : String) : String
+ getOutputHeader() : String
+ getDecayFunctionRepresentation() : String
+ tearDown()
RadialElectrodeEPRecorder
+ calculateStep(network : Network, timestep : int)
ProjectionPrunningElectrodeRecorder
log : Logger
config : SharedProperties
+ setUp(config : SharedProperties)
+ calculateStep(network : Network, timestep : int)
+ getOutputFileName(baseName : String) : String
NetworkStateElectrodeRecorder
log : Logger
config : SharedProperties
layerSum : int[]
separator : String
electrodeVariable : String[]
+ setUp(config : SharedProperties)
+ calculateStep(network : Network, timestep : int)
+ resetStep()
+ getOutputHeader() : String
# writeData()
+ getCalculationResult() : String
+ getOutputFileName(baseName : String) : String
HWRadialElectrodeEPRecorder
+ calculateStep(network : Network, timestep : int)
+ setupElectrode(network : Network)
# prepareCalculation(network : Network, timestep : int)
# calculateStepHW(network : Network, timestep : int)
LayeredElectrodeEPRecorder
+ calculateStep(network : Network, timestep : int)
MessageHandler
automata : GenericAutomataBehaviour
myAgent : Agent
config : JNSharedProperties
registerHandlers()
unregisterHandlers()
+ setUp(config : JNSharedProperties)
+ onShutdown(msg : ACLMessage)
+ getHandlerName() : String
+ setHandlerName(name : String)
+ getAgentName() : String
# onUnknonwMessage()
+ tearDown()
# sendMessage(msg : ACLMessage)
# getMessageProto(msgType : String) : ACLMessage
# createReply(inMsg : ACLMessage, msgType : String) : ACLMessage
# getMessageType(msg : ACLMessage) : String
# sendLocalMessage(msgType : String)
# sendLocalMessage(msgType : String, content : String)
# sendFailMessage()
# sendFailMessage(msg : String)
# setHandler(msgType : String, aMethod : String)
# registerHandler(msgType : String, obj : Object, aMethod : String)
# unregisterHandler(msgType : String, obj : MessageHandler)
EvolutionLauncher
simTypeNo : int
populationCnt : int
populationMax : int
populationSize : int
interSimulationDelay : int
defSimType : String
defClassPackage : String
defConfPath : String
evoId : String
bestSimulation : String
+ ST_CONFIGURED : String
+ ST_STARTED : String
+ ST_FINISHED : String
+ ST_RESULTS_COLLECTED : String
+ ST_RESULTS_PROCESSED : String
# genome : Genome
# simulationInfo : HashMap<>
# populationId : String
# populationList : LinkedList<>
# curSimIdx : int
# random : Random
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ onStartPopulation(msg : ACLMessage)
# haveSimulationsToStart() : boolean
# getNextSimulationsName() : String
# initializePopulationInfo()
+ startNextSimulation(msg : ACLMessage)
+ onSimulationFinish(msg : ACLMessage)
+ onEvolutionFinish(msg : ACLMessage)
+ onEvolutionStart(msg : ACLMessage)
# collectSimulationResults(simName : String) : boolean
# evaluateResults()
# updateGenome()
# getBestSimulationName() : String
# getSimulationNameByMsg(msg : ACLMessage) : String
# populationHasStatus(status : String) : boolean
# getAgentArguments(aName : String) : Object[]
# getAgentContainer(aName : String) : AgentContainer
# createNewAgent(ac : AgentContainer, aName : String) : AgentController
# getCommonArguments() : HashMap<>
# addSimulationArguments(aName : String, params : HashMap<>)
# getSimProperty(simName : String, simProperty : String) : Object
# setSimProperty(simName : String, simProperty : String, value : Object) : boolean
# isKnownSimulation(simName : String) : boolean
# prepareArguments(params : HashMap<>) : String[]
AbstractLinkMonitor
+ MH_LINK_REQUEST : String
+ MH_LINK_CONFIRM : String
+ MH_LINK_CONFIRM_CON : String
+ MH_LINK_CONFIRM_PROC : String
+ MH_LINK_REFUSED : String
+ MH_LINK_ACCEPTED : String
+ MH_LINK_BREAK : String
+ MH_ST_UNCONNECTED : String
+ MH_ST_CONNECTED : String
+ MH_ST_READY : String
+ MH_ST_PROCESSING : String
+ MH_ST_FINISHED : String
+ MH_STATUS_INPUTS : String
+ MH_STATUS_OUTPUTS : String
+ MH_TOPOLOGY_CHANGED : String
+ MH_SEEDER_LOST : String
+ MH_FEEDER_LOST : String
+ MH_SEEDER_FOUND : String
+ MH_FEEDER_FOUND : String
+ LM_ST_UNCONNECTED : int
+ LM_ST_CONNECTED : int
+ LM_ST_READY : int
+ LM_ST_PROCESSING : int
+ LM_ST_FINISHED : int
+ LM_ST_UNKNOWN : int
# readyVetoName : String
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ onVetoTimeout(msg : ACLMessage)
+ updateVetoTimeout()
+ getState() : int
+ getStateMessage() : String
+ getStateMessage(state : int) : String
+ getState(state : String) : int
# goState(newState : int)
# onUnconnected(oldState : int)
# onConnected(oldState : int)
# onReady(oldState : int)
# onProcessing(oldState : int)
# onFinished(oldState : int)
AbstractEncryptor
+ MH_NET_OUTPUT : String
+ MH_RAW_OUTPUT : String
+ MH_RAWOUT_CONFIG : String
+ MH_STATUS_REQ : String
+ MH_STATUS_RSP : String
# processedMsgs : long
# resendQueue : LinkedList<>
# resendVetoName : String
# rawConfig : String
# processingFlag : boolean
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ onProcessing(msg : ACLMessage)
+ onStopProcessing(msg : ACLMessage)
+ onReconfigure(msg : ACLMessage)
+ processUncofigured(msg : ACLMessage)
# sheduleResendQueue()
+ onResendEvent(msg : ACLMessage)
+ onStatusRequest(msg : ACLMessage)
+ OnProcess(msg : ACLMessage)
+ isConfigured() : boolean
+ isProcessing() : boolean
+ hasFeeders() : boolean
+ isSimulatorMessage(msg : ACLMessage) : boolean
# onStartProcessing(msg : ACLMessage)
+ process(msg : ACLMessage)
# prepareFeeders(feeders : Object)
# reconfigureOutput(msg : ACLMessage)


SimulationLauncher
agentInfo : HashMap<>[]
netId : String
nameFormat : String
defClassPackage : String
dataPath : String
confPath : String
inputPath : String
random : Random
interAgentDelay : long
shutdownMaxDelay : long
shutdownVetoName : String
launchVetoName : String
simStartIdx : int
+ ST_INIT : String
+ ST_UNCONNECTED : String
+ ST_PROCESSING : String
+ ST_FINISHED : String
# config : JNSharedProperties
# lastContainerIdx : int
# containerDivisor : int
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ startSimulation(msg : ACLMessage)
# getAgentContainerId(aID : int) : ContainerID
+ onStartAgent(msg : ACLMessage)
+ onSimulationFinish(msg : ACLMessage)
+ onPauseTimeout(msg : ACLMessage)
+ onAgentFinish(msg : ACLMessage)
+ onAgentProcessing(msg : ACLMessage)
+ onAgentPaused(msg : ACLMessage)
# setAgentStatus(agentName : String, status : String) : boolean
# agentFromNet(agentName : String) : boolean
# allAgentsHaveStatus(status : String) : boolean
# getAgentArguments(i : int) : Object[]
# getAgentContainer(i : int) : AgentContainer
# createNewAgent(ac : AgentContainer, i : int) : AgentController
# createNewAgent(ac : AgentContainer, agentName : String, classPackage : String, args : Object[]) : AgentController
# initializeInfo(aNum : int)
# getRoleByType(type : String) : String
# getCommonArguments() : HashMap<>
# addTypeSpecificArguments(params : HashMap<>, type : String)
# addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)
# prepareArguments(params : HashMap<>) : String[]
DistributedSimulationLauncher
lastContainerIdx : int
getAgentContainerId(aID : int) : ContainerID
+ onStartAgent(msg : ACLMessage)
CircleSimulationLauncher
linksNum : int
+ setUp(config : JNSharedProperties)
# initializeInfo(aNum : int)
# addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)
All2AllSimulationLauncher
addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)
+ onAgentFinish(msg : ACLMessage)
S2LMSimulationLauncher
forbidOnLevelLinks : boolean
sconfName : String
mconfName : String
pconfName : String
+ setUp(config : JNSharedProperties)
# initializeInfo(aNum : int)
# addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)
GenomeCircleLauncher
+ replicateVetoName : String
# genome : Genome
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ replicate(msg : ACLMessage)
+ onPauseTimeout(msg : ACLMessage)
+ onSimulationFinish(msg : ACLMessage)
# prepareReplicationArguments() : String[]
S2LMFFSimulationLauncher
+ setUp(config : JNSharedProperties)
# addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)
GenomeS2LMLauncher
+ replicateVetoName : String
# genome : Genome
# stopReplicate : boolean
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ replicate(msg : ACLMessage)
+ onPauseTimeout(msg : ACLMessage)
+ onSimulationFinish(msg : ACLMessage)
# prepareReplicationArguments() : String[]
GenomeS2LMFFLauncher
+ replicateVetoName : String
# genome : Genome
# stopReplicate : boolean
# registerHandlers()
+ setUp(config : JNSharedProperties)
+ replicate(msg : ACLMessage)
+ onPauseTimeout(msg : ACLMessage)
+ onSimulationFinish(msg : ACLMessage)
# prepareReplicationArguments() : String[]
SNLMSimulationLauncher
layersNum : int
layerSize : int
ffTopology : boolean
pLoopFrom : int
pLoopTo : int
+ setUp(config : JNSharedProperties)
# addAgentSpecificArguments(params : HashMap<>, type : String, aId : int)

Simulator's User Guide

Table of Contents:

Introduction 3

General Overview 3

Installation Guide 5

Software dependencies *5*

Software packaging *6*

Configuration Guide 7

The Simulator Package Configuration (simul.ini) *7*

Stand-alone neural network configuration *7*

Virtual Electrode Configuration *8*

The JNET Simulator Package *9*

Defining network behavior automata (descriptor.ini) *9*

Defining hierarchical circuits (net.ini) *10*

Defining evolutionary properties (net.ini) *10*

Running distributed simulation *12*

Simulators Output *12*

Getting the simulation status *13*

Preconfigured Simulations 14

run-Single.sh *14*

run-GSNLPM-2L-FB.sh *14*

run-GSNLPM-6L-[MES].sh *14*

Developer's Documentation 16

Introduction

This project was aimed at developing a scalable and distributed software platform dedicated to the simulation of large-scale multi-scale auto-organisative phenomena, and to the observation of potentially emerging behaviors. The platform is composed of many computing, behaving and communicating modules called Ubidules with developmental, learning, and evolution capabilities. Ubidules can be associated to sensory, actuators and communicators elements.

An unique simulation framework was developed to model large-scale biologically plausible neural networks and study the emergent complex behaviors in a virtually unbounded network of simulator modules. The biologically plausible neural network simulator application is offered as a tool to study information processing in brain-like systems. Concept of the distributed neural network is in auto-organizing three layered (sensory, processing, and motor) artificial neural network, each Ubidule of the network allows a study of an endless number of configurations aimed to suit researcher's specific interest. The simulation framework also can be used to process information in a real environment thanks to the interface with sensors and actuators. The real environment circumvent the need to simulate a virtual environment and ease the occurrence of unexpected emergent phenomena.

This Guide is organized in the following chapters. Chapter 1 describes a general concepts of the simulator. Chapter 2 describes an installation steps should be done to obtain runnable instance of the simulator. Chapter 3 describes simulator's organization, Chapter 4 describes configuration facilities of the simulator software in order to create custom simulation and finally Chapter 5 describes typical simulator's output.

General Overview

The simulator software package is a highly expandable and flexible framework aimed for hierarchical biologically plausible neural systems simulation. All components of the framework are organized in a modular way and allow inter-operability, compatibility, and expandability of the simulator system and its parts in accordance to the current needs. It is divided onto biologically plausible neural network simulator and distributed hierarchical evolutionary neural network simulator packages, which are together fully cover epi-, and onto-genetic and phylo-genetic level of evolutionary concept. On ontogenetic level simulator implements routines, which are in charge of origination and the development of the neural system based on decoded genome during its early stages. Genome decoding, neural network initialization, and inter-network connection establishment rules are also on this conceptual level of the simulator. On epigenetic level there is neural network simulator package learning features, which are limited to an individual lifetime. The neural network simulator itself naturally fits in this layer.

The simulator software consists from four major sub-packages oriented for specific simulation needs, they are:

nhrg-Jeign - the core simulator package. It allows simulation of neural network of leaky integrate-and-fire spiking (LIF) neurons arranged onto 2D square lattice featuring biologically plausible processes in the network (i.e. STDP, synaptic prunning, cell death, and apoptosis).

nhrg-JJeign - an utility package designed to make core simulator package JADE compatible and thus distributable over remote computational hosts.

nhrg-JElectrode - the core package implementing full set of virtual electrode functionality, which is run over simulated neural network.

nhrg-JNet - the core package targeted for distributed and evolutionary hierarchical neural network simulation. It allows to plan, create and maintain network activity connection-links between neural simulator agents.

Installation Guide

The simulator is written in Java language and run on any hardware with Java Virtual Machine (JVM) implementation confirming to the JVM ver.1.5 specifications or later. In particular it was tested to run on linux and MacOSX platforms, it runs also on win32, but to do so batch files should be rewritten in a win32 way, which is different from unix batch. All instructions here are given in assumption that simulator is running on unix-like platform. Basic understanding of command-line commands is necessary to run the simulator.

In order to install the simulator follow the next basic steps:

1. Download the simulator software binary package, file usually named in accordance to the following scheme: *nhrg-jnet-<version>-bin.zip*, where <version> is a three digit package version
2. Unzip the package (`unzip nhrg-jnet-<version>-bin.zip`)
3. It will create a folder name *nhrg-jnet*
4. Verify that you have Java Virtual Machine installed, not earlier than version 1.5 (can been done with `java -version`)
5. Open terminal and navigate to *nhrg-jnet* folder
6. In the terminal start the basic simulation by invoking the a command ./run.sh or sh run.sh from the folder where simulator was unpacked
7. Wait until simulation finish

By default simulator's configuration is such that 5 simulator agents (one sensory neural network, one motoric neural network and three processing neural network agents) will be started in the same JADE container, if no special simulation parameters are specified, they all will be situated on the same hardware node.
If such basic test-run of the simulation is failed, most probably, not all required libraries are in the Java's CLASSPATH. Please check dependency packages listed below and JVM's documentation.

Software dependencies

Basic software: Java Virtual Machine, version 1.5 or higher.

All following libraries are mandatory for the simulator package and normally are included in the simulator package.

NHRG libraries:

- **nhrg-jeign.jar** - Jeign package - stand-alone biologically plausible neural network simulator;
- **nhrg-jelectrode.jar** - JElectrode package - the virtual electrode simulator;
- **nhrg-common.jar** - common NHRG utilities library (including configuration parser);
- **nhrg-jjeign.jar** - JJeign package - JADE agent wrapper for Jeign simulator package;
- **nhrg-data.jar** - basic data manipulation library;
- **jade-3.7.0.jar** - JADE Library, including graphical user interface (GUI) for basic agent management tasks.
- **jadeLeap-3.7.0.jar** - portable version of JADE Library (the one should be used instead of jadeTools-3.7.0.jar - JADE tools library.
- **http-3.7.0.jar** - supplementary JADE library for inter-agent communication.

- **iiop-3.7.0.jar** - supplementary JADE library for inter-agent communication.

NB! Current version of the Simulator is tested for compatibility only with JadeLeap version 3.7.0. It is known that Simulator will not work with JadeLeap version 3.5.0 due to inner Jade's implementation bug with agent invocation (despite that the Simulator will work with generic Jade library higher than v.3.5.0, but v.3.7.0. has noticeably higher performance).

Other misc libraries:
- activation-1.1.jar
- jms-1.1.jar
- jmxri-1.2.1.jar
- jmxtools-1.2.1.jar
- log4j-1.2.15.jar
- mail-1.4.jar

Software packaging

The simulator is implemented in Java programming language, it is designed to efficiently emulate hierarchical neural network with especial emphasis on dynamic topology reconfiguration, rich possibilities for network state acquisition and evolutionary features of hierarchical networks.

jnet-bin.zip is a binary package based on JADE implementation. It is suitable for MacOSX, Linux, and win32 platforms with J2SE VM version 1.5 or higher. It consists of JNet package, as well as all other mandatory packages, like JElectrode, and Jeign. A number of default simulation with symmetric S3PM topology are pre-configured, so it will work "out of box", start-up scripts for MacOSX/Linux are supplied.

jnet-src.zip is a java source-code package. It will be required to extend functionality of the simulator beyond its current limits.

Packages are available for download via these links:
http://neuroheuristic.org/~vshaposh/jnet-bin.zip
http://neuroheuristic.org/~vshaposh/jnet-src.zip

Configuration Guide

In this section simulator configuration will be described. An approach to simulation's configuration is following simulator's hierarchical architecture (jeign - jjeign - jnet), so one should configure each stand-alone neural module, than hierarchical organization of the modules, than evolutionary features of the circuit. The configuration is propagated through configuration files or through genomic parametrization on the evolutionary level of the network.

Configuration of the simulator is non-trivial task, however software framework provides number of facilities to ease it. By default the simulator package (unpacked) is organized in the following way:

lib/ folder - contains collection of all mandatory simulator's libraries;

data/ folder - will contain number of output files created by simulator, those ones could be EChG files, network topology description files, etc;

conf/ folder - contains sub-folders with set of configuration files for each agent role, usually they are: def for processing agents, input-sng for sensory agent with arti- ficial stimulus, and output for motoric agents. This is the place were most changes will be done in order to get custom simulation.

conf/P-def/net.ini file - neural circuit specific configuration parameters, this covers all parameters which are agent-network related, but not related nor to the epi-genetic agents (Jeign), nor to the general automate configuration of onto-genetic agents;

conf/X-starter-*/descriptor.ini file - onto-genetic agent message handling automate configuration file, this specified main functionality of the onto-genetic agents, and from that point of view is analogous to the Jeign simulator pre-, post-processes configuration lists;

input/ folder - contain several folders with predefined artificial stimulus SNG file-sets for application on sensory module; One could apply different stimulus to the sensory agents by redefining those SNG file-sets.

run-*.sh file - a set of default start-up scripts. Each specify particular pre-configured simulation. This is most convenient configuration place if one could change one or several parameters for all modules of the network, without altering all `simul.ini` files.

The Simulator Package Configuration (simul.ini)

Stand-alone neural network configuration

Implemented in Java programming language the simulator is designed to efficiently emulate neural network models with especial emphasis on facilities for model recon- figuration and adjustment and on functionally rich possibilities for detailed network state acquisition. Model used is described in the main body of the Vladyslav Shaposhnyk's Doctoral Thesis.

The simulator defines a set of interfaces to general neural concepts and property access routines, like: neuron, synapse, network, signal-processing routines, input/output

routines. Within the range of implemented objects it is possible to assembly network of neurons following the integrate-and-fire spiking neuron model.
The simulator is an independent software package and thus it could be executed without all other modules provided by other packages, but it this way it is limited to the simulation of stand-alone rectangular SNN. In assumption that all mandatory libraries are in the Java's CLASSPATH.

The SNN simulator could be started by executing following command:

```
#java org.nhrg.apps.jeign.Jeign <parameters>
```

It is configured either with parameters read from configuration file or passed with genomic variables by higher-level packages of the simulator.

The configuration file by default is called `simul.ini` and is searched in the current working directory. Path to configuration file could be changed manually by specifying a parameter `-Pconf.path=<path/to/simul.ini>` in the command line.

According to the architecture each simulated timestep is pre- or post- processed by various processors. These processors and the order they executed fully define the simulation flow. Processors could be attached to Simulation object or to Network object.

Processor implementation classes and the order they executed are specified by the parameters in the file:

```
net.preprocess.<index> = <ClassURI>
```

and

```
net.postprocess.<index> = <ClassURI>.
```

Indexes of the processes start from 0 and continue with increments of one, gaps in the indexation are not allowed.

Sample process configuration section of the simulator is show below:

```
# define network processing routines
net.preprocess.0  = jeign.process.BackgroundActivityProcess
net.preprocess.1  = jeign.process.InputStimulusProcess
net.postprocess.0 = jeign.process.STDPProcess
net.postprocess.1 = jeign.process.PruningProcess
net.postprocess.2 = jeign.process.SynapticActivityTransmission
```

This sample configuration defines 2 pre-processors and 3 post-processors, which totals in five active processes: application of background activity, stimulation input activity, STDP synapse modification, cell pruning, and transmission of synaptic activity. Comments and blank lines are allowed in order to improve visual performance. Comments start with '#' sign and will be ignored by the software;.
NB! Please, pay special attention to mandatory parameters and parameters by default, as they could dramatically change the simulation results.

Virtual Electrode Configuration

The virtual electrode package implements EChG recorder processes compatible with Jeign simulator. In order to install virtual electrode an appropriate process should be added to processor's list in simulation's configuration. The electrode measures local field potentials over simulated neural network and produces Electro-EncephaloGraphy-like output (EChG signal).

Sample configuration of composite electrode is show below.

```
#electrode recorder process
net.postprocess.3=jelectrode.impl.process.CompositeElectrodeRecorder
#sub-electrodes' implementations
electrode.count=2
electrode.0.impl=jelectrode.impl.process.ReferenceElectrodeRecorder
electrode.1.impl=jelectrode.impl.process.RadialElectrodeEPRecorder
# default radial electrode parameters: X,Y, R, decay type
electrode.def.pos.x=11.5
electrode.def.pos.y=9.5
electrode.def.radius=9
electrode.def.decay.type=lin
```

This configuration will create a composite recorder (`CompositeElectrodeRecorder`) made of two virtual electrodes: a reference electrode () and a regular radial electrode (`RadialElectrodeEPRecorder`), which will be placed in between cells rows 9 and 10 on Y-axis and in between columns 11 and 12 on X-axis of 2D cell lattice of the SNN.

For a complete list of configuration parameters, please, consult Developers Guide section.

The JNET Simulator Package

Hierarchical evolutionary neural networks simulator is designed to run on distributed hardware platform, i.e. composed from a number of remote computational nodes. The JNET simulator could be started with following command:

```
#java -Xms64M -Xmx16G jade.Boot \
 -name mySim -agent agentName:org.nhrg.apps.jnet.JNet"( \
 -Pinet.msg.descriptor.conf.path=agent/conf/path/descriptor.ini \
 -Pinet.conf.path=agent/conf/path/net.ini)"
```

It is configured with parameters read from configuration file or from command line. Configuration parameters passed through command line will overwrite those from configuration file. It is a must to specify agent's (or agents') configuration file paths explicitly.

NB! If modeling neural circuits composed from SNN of 75x75 (or above) it is strongly recommended to use systems, which have **at least 1GB of RAM per pair** of SNN modules.

Defining network behavior automata (descriptor.ini)

All package modules, including the neural network simulator, are packaged in JADE network agents and then JADE framework is used by JNET hierarchical neural network framework to build up network of neural simulator modules. On each module JNet takes incoming messages from JADE message queue and processes them by appropriate handlers defined in the network behavior automata. Inter-agent communication and data-processing is considered as message exchange ruled by a number of internal protocols. Message processing handlers implement communication protocols and affect every aspect of message exchange.

Highest level handlers are:

- **AMSNetworkMonitor** - responsible for agents discovery in the network
- **InputLinkMonitor** - accept or refuse connections from upstream networks

- **OutputLinkMonitor** - requests and establishes connections from downstream networks
- **InputActivityMappingCodec** - decode spike train arriving from other neural modules from a form suited for transmission to a form suited to afferent layer stimulation
- **OutputActivityMappingCodec** - encode efferent layer activity into a form which is suited for transmission over the physical IP network

The simulator come with preconfigured automata for Processing, Sensory and Motoric modules of circuit. Module's automate configurations are very general and under normal circumstances should not be altered, but if there is such need, please, consult Developers Guide section of this Manual.

The execution flow is separated in terms of processing threads for the SNN simulator itself and hierarchical network logic, allowing to process network communications and to run simulations in the independent way.

Defining hierarchical circuits (net.ini)

Although, automates are standard their behavior could be and should be customized in order to create different circuit topologies. Usually it is *LinkMonitor*'s configuration parameters which play major role in the formation of topology, they are set up in *net.ini* file.

Regular *net.ini* configuration is explained here:

```
# AMSNetworkMonitor options
network.role=P

# LinkMonitor option
links.monitor.inputs=2
links.monitor.inputs.strategy=exact
links.monitor.connect.from.blacklist=o-P1:o-P2

# Agent Launcher class
sim.agent.impl=org.nhrg.apps.jjeign.JeignAgent
```

This configuration is made for a processing module (`network.role=P`). Every neural network module started using this configuration will have exactly 2 incoming connections (`links.monitor.inputs=2`) and will be modeled using `JeignAgent` class from *nhrg-jjeign* package for SNN lattice simulation, which is JADE wrapper for `Jeign` class from *nhrg-jeign* package. Depending on desired topology topology one could blacklist connections from another agents (`links.monitor.connect.from.blacklist`). It could happen that for complex topologies multiple *net.ini* files one for each agent/layer will be required.

Defining evolutionary properties (net.ini)

In order to introduce evolutionary features to the simulator and to simplify startup of complex hierarchical neural circuits special agent role (*Starter*) was introduced. Its only task is to start replication of its-own neural circuit on a user-specified event. Also starting that agent with an immediate replication option and number of generations equal to one will allow to model complex hierarchical topologies without creation of a number of configuration files for every module of the circuit, Starter agent do this. Starters agents still configured via net.ini files, but with additional features specific to this role. A number of default Starters are included in the **conf/** folder of the simulator binary package.

Features of this configuration will be explained on the following example:

```
#genetic network on circle topologies
inet.agent.impl=org.nhrg.apps.jnet.JNet
```

```
#config, output, and input consolidate path parameters
inet.def.conf.path=./conf
inet.def.data.path=./data
inet.def.input.path=./input

# topology specific options
inet.agent.num=6
inet.id.prefix=FS

# first generation autostart options
inet.agents.delay=1000
inet.autostart=true
```

This parameters will vary depending on type of topology starter agent is dedicated. Usually it is possible to set up how many agents are in the circuit (`inet.agent.num`), are links between layers allowed or not, etc. It is also possible to setup specific network prefix (`inet.id.prefix`) and if first generation should be auto-started (`inet.autostart`) or it will be started manually by the researcher. Input and output folders could be set up with `inet.def.*.path` parameters.

```
# Genetic options
genome.population.no.cur=0        # generation to start with
genome.population.no.max=6        # generation to stop
genome.size=3
genome.prefix=gene

# Genes 0 and 1 are system ones
gene.0.impl=org.nhrg.apps.jnet.impl.genes.FixedTextGene
gene.0.name=inet.msg.descriptor.conf.path
gene.0.init=devel/conf/X-starter-GS2P2PM/descriptor.ini

# Genes 0 and 1 are system ones
gene.1.impl=org.nhrg.apps.jnet.impl.genes.FixedTextGene
gene.1.name=inet.conf.path
gene.1.init=devel/conf/X-starter-GS2P2PM/net.ini

# Connectivity seed will be mutated every time
gene.2.impl=org.nhrg.apps.jnet.impl.genes.NumericGene
gene.2.name=inet.rng.seed         # corresponding config option name
gene.2.min=0                      # min value
gene.2.max=99999                  # max value
```

```
gene.2.init=11111                  # initial value
gene.2.sigma=10000                 # variability range
gene.2.granularity=1               # minimal delta of changes
gene.2.mutation.prob=1             # mutation probability
```

```
# Simulation options
XCsim.len=384000
XCinet.input.neurons=450
XSinet.input.neurons=900
XSsng.writer.file.mask=input/s450-l10-v45/s450-l10-v45
XSinet.input.amplitude=19
```

It is possible to add special parameters which will overwrite default configuration parameters given in particular *simul.ini* and *net.ini* files. If a parameter is prepended by *XC* it will be transmitted to all modules of the circuit (useful to ease the control of common parameters like simulation length). If a parameter is prepended by *XS*, *XP*, or *XM* that parameter will be transmitted only to a module of Sensory, Processing or Motoric role respectively (useful to setup specific parameters like external activity stimulation amplitude for Sensory module).

Running distributed simulation

A distributed simulation from configuration point of view is absolutely the same, as evolutionary hierarchical one, but it will differ by the way JADE containers are started. It is advised to launch a distributed simulation in a following way:

1. Start empty JADE's MainContainer (without agents) with a command:

```
#java -Xms64M -Xmx16G jade.Boot -name mySim
```

2. On every remote host, but one start empty JADE's secondary containers specifying an IP address of a host where MasterContainer resides:

```
#java -Xms64M -Xmx16G jade.Boot \
 -name mySim -container -host $MASTERNODE
```

3. On the last remote host start JADE's secondary container (as before), but with a starter agent:

```
#java -Xms64M -Xmx16G jade.Boot \
 -name mySim -container -host $MASTERNODE \
 -agent starterName:org.nhrg.apps.jnet.JNet"( \
 -Pinet.msg.descriptor.conf.path=agent/conf/path/descriptor.ini \
 -Pinet.conf.path=agent/conf/path/net.ini)"
```

After that simulation should be treated as a regular one. Simulation's output files will be created on each remote node. It is recommended to put data folder outside of NFS system. There is no GUI in distributed simulations.

Simulators Output

It is (if enabled in appropriate configurations):

- EChG signals (*composite.ecg files) - virtual electrode output. EEGLab compatible plain text file EEG file; one row per channel;
- network dump (*xml files) - full network dump for each timestep of simulation, for details, please, consult nhrg-Feign simulator documentation;
- Spike-trains (*sdf files) - SDF formatted spike trains from selected neurons, for details, please, consult nhrg-Feign simulator documentation.

Getting the simulation status

In order to check status of the simulation a management agent with basic functionality exists. It provides facilities to track progress of the simulation and to synchronize semi-finished simulation, which happens when several agents were started before other agents and their connection constraints were self-sufficient. Agent's actions are triggered by JADE messages. Full list of available actions is available in developer's documentation or could be obtained by sending d_help message to the management agent.

To get help message through JADE GUI:
- select management agent, usually named sMgr;
- right click, select "Send Message";
- in the popup dialog fill in language: "jnet-sl" and ontology "d_help";
- press "OK" button.

To get help message though remote command line connection:
- open telnet connection to the simulator on port 2222, p.ex:

```
telnet node1 2222
```

- A command line prompt will appear;
- type `d_help` and then enter.

An output will be written to the terminal where simulation was started.

In the same way, it is possible to check current simulation progress on all known agents, send "d_current_progress" to the management agent. It will request onto-genetic agents about their progress and the results will be printed on the terminal.

Preconfigured Simulations

There are 3 pre-configured simulation included in the *jnet-bin.zip* package. Batch files which start simulations are: run-Single.sh, run-GSNLPM-2L-FB.sh and

run-Single.sh

A stand-alone SNN simulation. Its corresponding configurations files are in the *conf/Z-Single* folder.

Usage is following:

```
################################################
####
####     ./run-Single.sh NETID NA NF
####NETID    - connectivity RNG seed
####NA   - noise (BGA) amplitude
####NF   - noise (BGA) frequency
####
################################################
```

run-GSNLPM-2L-FB.sh

A simulation of a hierarchical neural circuit of 6 modules (S, 4xP, M) connected into a topology with feed-backward connections, but without horizontal connections (FB). It takes configurations from *conf/X-starter-GSNLPM*, as well as *S-input-sng*, *P-def* and *M-output* folders;

Usage is following:

```
################################################
####     ./run-GSNLPM-2L-FB.sh NETID
####NETID    - connectivity RNG seed
####NA   - noise (BGA) amplitude - fixed to 1.9
####         (Sensory will have 2x that)
####NF   - noise (BGA) frequency - fixed to 300
####
################################################
####NB! At least 4GB of RAM is recommended
#### 6-8GB is highly recommended
################################################
```

run-GSNLPM-6L-[MES].sh

A set is specially adopted for distributed hierarchical evolutionary simulations of FBH-like topologies with multiple processing layers (topology could be specified in parameters between FF, FB, FFH, FBH; number of processing layers by default is 6, i.e. it will start a

total of 14 SNN modules 6*2+2), with mutation of a gene corresponding to the connectivity seed parameter.

- run-GSNLPM-6L-M.sh is an empty JADE Master Container as explained in the Section “Running distributed simulation”
- run-GSNLPM-6L-E.sh is an empty JADE Slave Container as explained in the Section “Running distributed simulation”
- and run-GSNLPM-6L-S.sh is an JADE Slave Container with a starter agent, which will start a simulation over all nodes connected via JADE framework.

Master and Empty will start empty JADE Containers (please refer JADE documentation for more information), when Starter agent is launched the simulation will be started over the whole set of platforms hosting JADE framework.

Master MUST be started first and its usage is following:

```
################################################
####     ./run-GSNLPM-2L-M.sh NETID ISFF HAVEH PLOOP
####NETID    - connectivity RNG seed
####ISFF- if true will be a Feed-forward topology
####HAVEH    - if true will have Horizontals
####PLOOP    - if not -1 will have a loop between layers
####
################################################
```

Container should be started after and the last one will be Starter, their usage is following:

```
################################################
####     ./run-GSNLPM-2L-[ES].sh NETID ISFF HAVEH PLOOP
####NETID    - connectivity RNG seed
####ISFF- if true will be a Feed-forward topology
####HAVEH    - if true will have Horizontals
####PLOOP    - if not -1 will have a loop between layers
####
################################################
```

```
################################################
#### NB! A cluster system with 16GB of RAM is
#### recommended for the simulation (6Lx2P+S+M)
#### 20GB of RAM is highly recommended
################################################
```

Developer's Documentation

Please download javadoc documentation pages from following links:

http://neuroheuristic.org/~vshaposh/nhrg-jeign.pdf
http://neuroheuristic.org/~vshaposh/nhrg-jelectrode.pdf
http://neuroheuristic.org/~vshaposh/nhrg-jjeign.pdf
http://neuroheuristic.org/~vshaposh/nhrg-jnet.pdf

Bibliography

[1] M. Abeles. *Local cortical circuits*. Springer-Verlag, Berlin, 1982.

[2] M. Abeles. *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, 1 edition, 1991.

[3] E.D. Adrian and B.H.C. Matthews. The berger rhythm: potential changes from the occipital lobes in man. *Brain*, 57:355–385, 1934.

[4] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[5] T. I. Aksenova, V. Volkovich, and A.E.P. Villa. Robust structural modeling and outlier detection with GMDH-type polynomial neural networks. *LNCS*, pages 881–886, 2005.

[6] Tetyana I. Aksyonova, Vladimir V. Volkovich, and Igor V. Tetko. Robust polynomial neural networks in quantative-structure activity relationship studies. *Syst. Anal. Model. Simul.*, 43:1331–1339, October 2003.

[7] W. Amos and J. Harwood. Factors affecting levels of genetic diversity in natural populations. *Phil. Trans. R. Soc.*, 353:177–86+, 1998.

[8] Charles W. Anderson and Zlatko Sijerčić. Classification of eeg signals from four subjects during five mental tasks. In *Proceedings of the Conference on Engineering Applications in Neural Networks (EANN'96*, pages 407–414, 1996.

[9] R. Armitage, C. Landis, R. Hoffmann, M. Lentz, N. Watson, J. Goldberg, and D. Buchwald. Power spectral analysis of sleep eeg in twins discordant for chronic fatigue syndrome. *J Psychosom Res.*, 66(1):51–7, Jan 2009.

[10] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, 25(1):25–29, May 2000.

[11] A Babajani-Feremi and H Soltanian-Zadeh. Multi-area neural mass modeling of eeg and meg signals. *Neuroimage*, 52(3):793–811, Sep 2010.

[12] Alexandre Barachant, Stéphane Bonnet, Marco Congedo, and Christian Jutten. Riemannian geometry applied to bci classification. In *Proceedings of the 9th international conference on Latent variable analysis and signal separation*, LVA/ICA'10, pages 629–636, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] Paolo Emilio Barbano, Marina Spivak, Jiawu Feng, Marco Antoniotti, and Bud Mishra. A coherent framework for multiresolution analysis of biological networks with ,äúmemory,äù: Ras pathway, cell cycle, and immune system. *Proceedings of the National Academy of Sciences of the United States of America*, 102(18):6245–6250, 2005.

[14] Nils Barricelli. Numerical testing of evolution theories. *Acta Biotheoretica*, 16:69–98, 1962. 10.1007/BF01556771.

[15] D S Bassett and E T Bullmore. Human brain networks in health and disease. *Curr Opin Neurol*, 22(4):340–347, Aug 2009.

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

[17] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems With Jade*. Wiley, Wiltshire, Great Britain, 2007.

[18] Lubica Benoskova and Nicola Kasabov. *Computational Neurogenetic Modeling*. Springer, New York, NY, USA, 2007.

[19] Christoph Bergmeir and José M. Benítez. *Neural Networks in R using the Stuttgart Neural Network Simulator: RSNNS*, 2010. R package version 0.3.

[20] Upinder S Bhalla. Signaling in small subcellular volumes. i. stochastic and diffusion effects on individual pathways. *Biophys. J.*, 87(2):733–744, August 2004.

[21] James M. Bower. *Reverse engineering the nervous system: an anatomical, physiological, and computer based approach*, pages 3–24. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[22] James M. Bower and David Beeman. *The book of GENESIS - exploring realistic neural models with the GEneral NEural SImulation System (2. ed.)*. Springer, 1998.

[23] V. Braitenberg. *On the Texture of Brains: An introduction to neuroanatomy for the cybernetically minded*. Heidelberg Science Library. Springer, New York, NY, USA, 1977. ISBN: 0-387-08391-X.

[24] V. Braitenberg. Cortical architectonics : General and areal. In M.A.B. Brazier and H. Petsche, editors, *In: Architectonics of the Cerebral Cortex*, pages 443–465. Raven Press, New York, 1978.

[25] V. Braitenberg and A. Schuez. *Cortex: statistics and geometry of neuronal connectivity*. Springer, Berlin, second edition, 1998.

[26] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris Jr., Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Viéville, Eilif Mueller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007.

[27] D. R. Brillinger. An introduction to polyspectra. *Ann. Math. Stat.*, 36:1351–1374, 1965.

[28] Olivier Brousse, Jeremie Guillot, Gilles Sassatelli, Thierry Gil, Michel Robert, Juan Manuel Moreno, Alessandro Villa, and Eduardo Sanchez. A bio-inspired agent framework for hardware accelerated distributed pervasive applications. In *2009 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 415–422, Washington, DC, USA, 2009. IEEE Computer Society.

[29] Olivier Brousse, Gilles Sassatelli, Thierry Gil, Michel Robert, François Grize, Eduardo Sanchez, Andrés Upegui, and Yann Thoma. The perplexus programming framework: Combining bio-inspiration and agent-oriented programming for the simulation of large scale complex systems. *Lect Notes Comput Sci*, 5216:402–407, 2008.

[30] Nicolas Brunel. Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *Journal of Physiology-Paris*, 94(5-6):445 – 463, 2000.

[31] Nicolas Brunel. Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *Journal of Computational Neuroscience*, 8(3):183–208, May 2000.

[32] Guido Bugmann, Chris Christodoulou, and John Taylor. Role of temporal integration and fluctuation detection in the highly irregular firing of a leaky integrator neuron model with partial reset, 1995.

[33] T.H. Bullock, M. McClune, J. Achimowicz, V. J. Iragui-Madoz, R. B. Duckrow, and S. S. Spencer. EEG coherence has structure in the millimeter domain, 1995.

[34] K. P. Burnham and D. R. Anderson. Multimodel inference: Understanding aic and bic in model selection. *Sociological Methods Research*, 33(2):261–304, November 2004.

[35] D.J. Buysse, A. Germain, ML. Hall, D.E. Moul, E.A. Nofzinger, A. Begley, C.L. Ehlers, W. Thompson, and D.J. Kupfer. Eeg spectral analysis in primary insomnia: Nrem period effects and sex differences. *Sleep*, 31(12):1673–82, Dec 2008.

[36] FIPA TC C. Fipa communicative act library specification. Technical report, IEEE Foundation for Intelligent Physical Agents, 2001.

[37] Natalia Caporale and Yang Dan. Spike Timing–Dependent Plasticity: A Hebbian Learning Rule. *Annual Review of Neuroscience*, 31(1):25–46, February 2008.

[38] D. W. Choi. Glutamate neurotoxicity and diseases of the nervous system. *Neuron*, 1(8):623–634, October 1988.

[39] D Cosandier-Rimélé, I Merlet, F Bartolomei, J M Badier, and F Wendling. Computational modeling of epileptic activity: from cortical sources to EEG signals. *J Clin Neurophysiol*, 27(6):465–470, Dec 2010.

[40] O David, L Harrison, and K J Friston. Modelling event-related responses in the brain. *Neuroimage*, 25(3):756–770, Apr 2005.

[41] Gustavo Deco, Edmund T. Rolls, and Barry Horwitz. "what" and "where" in visual working memory: A computational neurodynamical perspective for integrating fmri and single-neuron data. *J. Cognitive Neuroscience*, 16:683–701, May 2004.

[42] Arnaud Delorme and Scott Makeig. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *J Neurosci Meth*, 134(1):9–21, March 2004.

[43] Markus Diesmann, Max planck Inst Für Strömungsforschung, and Marc oliver Gewaltig. Nest: An environment for neural systems simulations. In *In T. Plesser and V. Macho (Eds.), Forschung und wisschenschaftliches Rechnen, Beitrage zum Heinz-Billing-Preis 2001, Volume 58 of GWDG-Bericht*, pages 43–70, 2002.

[44] D. L. Donoho and P. J. Huber. The notion of breakdown point. *Festschr. for Erich L. Lehmann*, pages 157–184, 1983.

[45] Marco Dorigo, Vito Trianni, Erol Sahin, Roderich Gros, Thomas Halva Labella, Gianluca Baldassarre, Stefano Nolfi, F. Mondada, Jean-Louis Deneubourg, Dario Floreano, and Luca Maria Gambardella. Evolving Self-Organizing Behaviors for a Swarm-bot. *Autonomous Robots, special Issue on Swarm Robotics*, 17(2-3):223–245, 2004. September - November 2004 Sponsor: swarm-bots, OFES 01-0012-1.

[46] O. Dressler, G. Schneider, G. Stockmanns, and Kochs E. Awareness and the eeg power spectrum: Analysis of frequencies. *British Journal of Anaesthesia*, 93(6):806–9, 2004.

[47] O. Dressler, G. Schneider, G. Stockmanns, and E. Kochs. Awareness and the eeg power spectrum: Analysis of frequencies. *British Journal of Anaesthesia*, 93(6):806–9, 2004.

[48] J D Drover, N D Schiff, and J D Victor. Dynamics of coupled thalamocortical modules. *J Comput Neurosci*, 28(3):605–616, Jun 2010.

[49] G. Dumermuth, P. J. Huber, B. Kleiner, and T. Gasser. Analysis of the interrelations between frequency bands of the EEG by means of the bispectrum. a preliminary study. *Electroencephalogr. Clin. Neurophysiol.*, 31:137–148, 1971.

[50] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):pp. 1–26, 1979.

[51] R. Elul. The genesis of the eeg. *Int Rev Neurobiol*, 15, 1972.

[52] A A Fingelkurts and A A Fingelkurts. Short-term EEG spectral pattern as a single event in EEG phenomenology. *Open Neuroimag J*, 4:130–156, 2010.

[53] A A Fingelkurts, A A Fingelkurts, and S Kähkönen. Functional connectivity in the brain–is it an elusive concept? *Neurosci Biobehav Rev*, 28(8):827–836, Jan 2005.

[54] D. B. Fogel, L. J. Fogel, and V. W. Porto. Evolving neural networks. *Biol. Cybern.*, 63:487–493, September 1990.

[55] W J Freeman. Simulation of chaotic eeg patterns with a dynamic model of the olfactory system. *Biol. Cybern.*, 56:139–150, May 1987.

[56] W J Freeman. A field-theoretic approach to understanding scale-free neocortical dynamics. *Biol Cybern*, 92(6):350–359, Jun 2005.

[57] M. S. Gazzaniga. Organization of the human brain. *Science*, 245:947–952, 1989.

[58] R. Gnanadesikan and J. R. Kettenring. Robust estimates, residuals, and outlier detection with multiresponse data. *Biometrics*, 28(1):81–124, 1972.

[59] M Goodfellow, K Schindler, and G Baier. Intermittent spike-wave dynamics in a heterogeneous, spatially extended neural mass model. *Neuroimage*, 55(3):920–932, Apr 2011.

[60] Dan F M Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, 2(0), 2008.

[61] R T Gray and P A Robinson. Stability and structural constraints of random brain networks with excitatory and inhibitory neural populations. *J Comput Neurosci*, 27(1):81–101, Aug 2009.

[62] A. S. Hadi, A. H. M. Rahmatullah Imon, and M. Werner. Detection of outliers. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):57–70, 2009.

[63] Pentti O. Haikonen. *Robot Brains: Circuits and Systems for Conscious Machines*. WileyBlackwell, 2007.

[64] F. R. Hampel. A general qualitative definition of robustness. *Ann. Math. Stat.*, 42:1887–1896, 1971.

[65] L M Harrison, O David, and K J Friston. Stochastic models of neuronal dynamics. *Philos Trans R Soc Lond B Biol Sci*, 360(1457):1075–1091, May 2005.

[66] Neep Hazarika, Jean Zhu Chen, Ah Chung Tsoi, and Alex Sergejew. Classification of eeg signals using the wavelet transform. *Signal Processing*, 59(1):61–72, May 1997.

[67] D. Hebb. *The organization of behavior*. John Wiley, New York, 1949.

[68] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, August 1952.

[69] P. J. Huber. *Robust Statistics*. Wiley, New York, 1981.

[70] F. T. Husain, M. A. Tagamets, S. J. Fromm, A. R. Braun, and B. Horwitz. Relating neuronal dynamics for auditory object processing to neuroimaging activity: a computational modeling and an fmri study. *NeuroImage*, 21(4):1701 – 1720, 2004.

[71] P Husar, S Berkes, A Götze, G Henning, and K U Plagwitz. Improving snr (signal to noise ratio) in multichannel eeg recording. *Biomedizinische Technik Biomedical engineering*, 47 Suppl 1 Pt 2:566–569, 2002.

[72] J. Iglesias, O.K. Chibirova, and A.E.P. Villa. Nonlinear dynamics emerging in large scale neural networks with ontogenetic and epigenetic processes. *Lect Notes Comput Sci*, 4668:579–588, 2007.

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

[74] J. Iglesias and A. E. P. Villa. Effect of stimulus-driven pruning on the detection of spatiotemporal patterns of activity in large neural networks. *BioSystems*, 89:287–293, 2007.

[75] J Iglesias and A E P Villa. Recurrent spatiotemporal firing patterns in large spiking neural networks with ontogenetic and epigenetic processes. *J Physiol Paris*, 104(3-4):137–146, May-Sep 2010.

[76] Javier Iglesias. *Emergence of Oriented Circuits driven by Synaptic Pruning associated with Spike-Timing-Dependent Plasticity*. PhD thesis, University of Lausanne, University Joseph Fourier Grenoble 1, 2005.

[77] Javier Iglesias and Alessandro E. P. Villa. Emergence of preferred firing sequences in large spiking neural networks during simulated neuronal development. *Int J Neural Syst*, 18(4):267–277, 2008.

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

[79] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(4):364–378, 1971.

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

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

[82] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Trans. Neural Networks*, pages 1569–1572, 2003.

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

[84] Nikola Kasabov. *Evolving Connectionist Systems: The Knowledge Engineering Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[85] Nikola Kasabov. Integrative connectionist learning systems inspired by nature: current models, future trends and challenges. *Natural Computing*, 8:199–218, June 2009.

[86] Nikola Kasabov. Integrative probabilistic evolving spiking neural networks utilising quantum inspired evolutionary algorithm: a computational framework. In *Proceedings of the 15th international conference on Advances in neuro-information processing - Volume Part I*, ICONIP'08, pages 3–13, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[88] Bob Kemp and Jesus Olivan. European data format [']plus' (edf+), an edf alike standard format for the exchange of physiological data. *Clinical Neurophysiology*, 114(9):1755 – 1761, 2003.

[89] B W Knight. Dynamics of encoding in neuron populations: some general mathematical features. *Neural Comput*, 12(3):473–518, Mar 2000.

[90] V.I. Koroleva, V.I. Davydov, and G.Y. Roshchina. Properties of spreading depression identified by eeg spectral analysis in conscious rabbits. *Neurosci Behav Physiol*, 39(1):87–97, 2009.

[91] Hugo Kubinyi. Evolutionary variable selection in regression and pls analyses. *Journal of Chemometrics*, 10(2):119–133, 1996.

[92] William Langdon. 2-bit flip mutation elementary fitness landscapes. In Anne Auger, Jonathan L. Shapiro, L. Darrell Whitley, and Carsten Witt, editors, *Theory of Evolutionary Algorithms*, number 10361 in Dagstuhl Seminar Proceedings, pages 1–19, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[93] K. S. Lii and K. N. Helland. Cross-bispectrum computation and variance estimation. *ACM Trans. Math. Softw.*, 7(3):284–294, 1981.

[94] F. H. Lopes da Silva, A. Hoeks, H. Smits, and L. H. Zetterberg. Model of brain rhythmic activity. *Biological Cybernetics*, 15:27–37, 1974. 10.1007/BF00270757.

[95] H. P. Lopuhaa. Asymptotics of reweighted estimators of multivariate location and scatter. *The Annals of Statistics*, 27(5):1638–1665, 1999.

[96] H. R. Madala and A. G. Ivakhnenko. *Inductive Learning Algorithms for Complex Systems Modeling.* CRC Press Inc., Boca Raton, FL, USA, 1994.

[97] Hema R. Madala and Aleksei Grigorievich Ivakhnenko. *Inductive Learning Algorithms for Complex Systems Modeling.* CRC Press, Inc., Boca Raton, FL, USA, 1994.

[98] P. C. Mahalanobis. On the generalized distance in statistics. *Natl. Inst. Science*, 12:49–55, 1936.

[99] S. Makeig, A.J. Bell, T.P. Jung, and T.J. Sejnowski. Independent component analysis of electroencephalographic data. *Advances in Neural Information Processing Systems*, 8:145–151, 1996.

[100] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275:213–215, 1997.

[101] R. A. Maronna and R. H. Zamar. Robust estimates of location and dispersion for high-dimensional datasets. *Technometrics*, 44(4):307–317, 2002.

[102] M.L.Hines and N.T Carnevale. The neuron simulation environment. *Neur Comp*, 9:1179–1209, 1997.

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

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

[105] C.L. Nikias and M.R. Raghuveer. Bispectrum estimation: a digital signal processing framework. *Proc. IEEE*, 75:869–891, 1987.

[106] Paul L. Nunez and Ramesh Srinivasan. *Electric Fields of the Brain.* Oxford University Press, New York, NY, USA, 2006.

[107] P.L. Nunez. *Neocortical Dynamics and Human EEG Rhythms.* Oxford University Press, New York, NY., 1995.

[108] S T Okuhata, S Okazaki, and H Maekawa. EEG coherence pattern during simultaneous and successive processing tasks. *Int J Psychophysiol*, 72(2):89–96, May 2009.

[109] D. P. O'Leary. Robust regression computation computation using iteratively reweighted least squares. *SIAM J. Matrix Anal. Appl.*, 11(3):466–480, 1990.

[110] H. Pearson. Genetics: What is a gene? *Nature*, 441(7092):398–401, 05 2006.

[111] Taraka D. Peddireddy and José M. Vidal. Multiagent network security system using FIPA-OS. In *Proceedings of the IEEE SoutheastCon*, 2002.

[112] Stephen Perrig, Javier Iglesias, Vladislav Shaposhnyk, Olga Chibirova, Pierre Dutoit, Jeremie Cabessa, Katerina Espa-Cervena, Laurent Pelletier, Francois Berger, and Alessandro E. P. Villa. Functional interactions in hierarchically organized neural networks studied with spatiotemporal firing patterns and phase-coupling frequencies. *Chin J Physiol*, 53(6):382–395, 2010.

[113] Antonio Politi and Stefano Luccioli. Dynamics of networks of leaky-integrate-and-fire neurons. In Ernesto Estrada, Maria Fox, Desmond J. Higham, and Gian-Luca Oppo, editors, *Network Science*, pages 217–242. Springer London, 2010.

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

[115] Stefan Rännar, Paul Geladi, Fredrik Lindgren, and Svante Wold. A PLS kernel algorithm for data sets with many variables and fewer objects. Part 1: Theory and algorithm. *Journal of Chemometrics*, 8(2):111–125, 1994.

[116] C J Rennie, P A Robinson, and J J Wright. Unified neurophysical model of EEG spectra and evoked potentials. *Biol Cybern*, 86(6):457–471, Jun 2002.

[117] M. Ridley. *Genome*. ISBN 0-06-019497-9. NY: Harper Perennial, New York, 2006.

[118] P. D. Roberts and C. C. Bell. Spike timing dependent synaptic plasticity in biological systems. *Biol. Cybern.*, 87:392–403, 2002.

[119] E. Ronchetti. Robustness aspects of model choice. *Statistica Sinica*, 7:327–338, 1997.

[120] P. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79:871–880, 1984.

[121] P. Rousseeuw and V. Yohai. Robust regression by means of S-estimators. *Robust and nonlinear time series analysis*, SMC-1(26):256–272, 1983.

[122] Eduardo Sanchez, Daniel Mange, Moshe Sipper, Marco Tomassini, Andres Perez-Uribe, and André Stauffer. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. In Tetsuya Higuchi, Masaya Iwata, and Weixin Liu, editors, *Evolvable Systems: From Biology to Hardware*, volume 1259 of *Lecture Notes in Computer Science*, pages 33–54. Springer Berlin / Heidelberg, 1997.

[123] Eduardo Sanchez, Andres Perez-Uribe, Andres Upegui, Yann Thoma, Juan Manuel Moreno, Alessandro Villa, Henri Volken, Andrzej Napieralski, Gilles Sassatelli, and Erwan Lavarec. Perplexus: Pervasive computing framework for modeling complex virtually-unbounded systems. In *AHS '07: Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems*, pages 587–591, Washington, DC, USA, 2007. IEEE Computer Society.

[124] Erik Schutter and James M. Bower. An active membrane model of the cerebellar Purkinje cell. II. *Journal of Neurophysiology*, 71:401–419, 1994.

[125] Michael N. Shadlen and William T. Newsome. The variable discharge of cortical neurons: Implications for connectivity, computation, and information coding. *J. Neurosci*, 18:3870–3896, 1998.

[126] Vladyslav Shaposhnyk, Pierre Dutoit, Victor Contreras-Lámus, Stephen Perrig, and Alessandro Villa. A framework for simulation and analysis of dynamically organized distributed neural networks. *Lect Notes Comput Sci*, 5768:277–286, 2009.

[127] W. Singer. Synchronization of cortical activity and its putative role in information processing and learning. *Ann Rev Physiol*, 55:349–374, March 1993.

[128] Montgomery Slatkin. Fixation probabilities and fixation times in a subdivided population. *Evolution*, 35(3):pp. 477–488, 1981.

[129] A. Swami, J.M. Mendel, and C.L. Nikias. *Higher-Order Spectral Analysis Toolbox*, volume 1. The Mathworks Inc, 2nd edition, 1998.

[130] J. Szentagothai. The "module-concept" in cerebral cortexarchitecture. *Brain Res*, 95:475–496, 1975.

[131] J. Szentagothai. The neural network of the cerebral cortex: A functional interpretation. *Proc. Roy. Soc. Lon.*, 201:219–248, 1978.

[132] J. Szentagothai and M.A. Arbib. Conceptual models of neural organization. *Neurosci. Res. Bull.*, 12:307–510, 1974.

[133] I. V. Tetko and A. E. Villa. A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains. 1. detection of repeated patterns. *J Neurosci Meth*, 105:1–14, 2001.

[134] I. V. Tetko and A. E. Villa. A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains. 2. application to simultaneous single unit recordings. *J Neurosci Meth*, 105:15–24, 2001.

[135] I. V. Tetko and A. E. P. Villa. Pattern grouping algorithm and deconvolution filtering of non-stationary correlated poisson processes. *Neurocomputing*, pages 1709–1714, 2001.

[136] T C Thiagarajan, M A Lebedev, M A Nicolelis, and D Plenz. Coherence potentials: loss-less, all-or-none network events in the cortex. *PLoS Biol*, 8(1), Jan 2010.

[137] CB Thompson. Apoptosis in the pathogenesis and treatment of disease. *Science*, 267(5203):1456–1462, 1995.

[138] Andres Upegui, Yann Thoma, Eduardo Sanchez, Andres Perez-Uribe, Juan Manuel Moreno, Jordi Madrenas, and Gilles Sassatelli. The perplexus bio-inspired hardware platform: A flexible and modular approach. *Int. J. Know.-Based Intell. Eng. Syst.*, 12(3):201–212, 2008.

[139] Errikos M. Ventouras, Periklis Y. Ktonas, Hara Tsekou, Thomas Paparrigopoulos, Ioannis Kalatzis, and Constantin R. Soldatos. Independent component analysis for source localization of eeg sleep spindle components. *Comp. Int. and Neurosc.*, 2010, 2010.

[140] A. E. P. Villa, I. V. Tetko, P. Dutoit, Y. De Ribaupierre, and F. De Ribaupierre. Corticofugal modulation of functional connectivity within the auditory thalamus of rat. *J. Neurosci. Meth.*, 86:161–178, 1999.

[141] A. E. P. Villa, I. V. Tetko, P. Dutoit, and G. Vantini. Non-linear cortico-cortical interactions modulated by cholinergic afferences from the rat basal forebrain. *BioSystems*, 58:219–228, 2000.

[142] A. E. P. Villa, I. V. Tetko, and J. Iglesias. Computer assisted neurophysiological analysis of cell assemblies activity. *Neurocomputing*, 38-40:1025–1030, June 2001.

[143] D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393:440–442, 1998.

[144] Michael Watts and Nik Kasabov. Evolutionary optimisation of evolving connectionist systems. In *In CEC'2002*, pages 606–610. IEEE Press, 2002.

[145] F. Wendling, J. J. Bellanger, F. Bartolomei, and P. Chauvel. Relevance of nonlinear lumped-parameter models in the analysis of depth-eeg epileptic signals. *Biological Cybernetics*, 83:367–378, 2000. 10.1007/s004220000160.

[146] Wikipedia. Evolution — Wikipedia, the free encyclopedia, 2010. [Online; accessed 22-November-2010].

[147] G Wolters and A Raffone. Coherence and recurrency: maintenance, control and integration in working memory. *Cogn Process*, 9(1):1–17, Mar 2008.

[148] T Womelsdorf, J M Schoffelen, R Oostenveld, W Singer, R Desimone, A K Engel, and P Fries. Modulation of neuronal interactions through neuronal synchronization. *Science*, 316(5831):1609–1612, Jun 2007.

[149] V. Yohai. High breakdown-point and high efficiency robust estimates for regression. *Ann. Stat.*, 15:642–656, 1987.

[150] Y. P. Yurachkovsky. Restoration of polynomial dependencies using self-organization. *Soviet Automatic Control*, 14:17–22, 1981.

[151] Melissa Zavaglia, Laura Astolfi, Fabio Babiloni, and Mauro Ursino. The effect of connectivity on eeg rhythms, power spectral density and coherence among coupled neural populations: Analysis with a neural mass model. *IEEE Transactions on Biomedical Engineering*, 55:69–77, 2008.