UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

Ph.D. DEGREE IN
PHYSICS

Cycle XXXVI

TITLE OF THE Ph.D. THESIS

Large-scale neuronal networks: from simulation technology to the study of plasticity-driven cognitive processes

Scientific Disciplinary Sector(s)

FIS/07

Ph.D. Student:	Gianmarco Tiddia
Supervisor	Bruno Golosio
Co-Supervisor	Pier Stanislao Paolucci

Final exam. Academic Year 2022/2023
Thesis defence: January 2024 Session

UNIVERSITÀ DEGLI STUDI DI CAGLIARI

Faculty of Science

Department of Physics

PhD school of Physics

Cycle: XXXVI

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Large-scale neuronal networks: from simulation technology to the study of plasticity-driven cognitive processes

Gianmarco Tiddia

matr: 200/1053/65055

Academic Advisors

Prof. Bruno Golosio
Dott. Pier Stanislao Paolucci

Submitted 2022/2023

Index

List of Publications 5

Abstract 7

I Introduction 9

1 Modeling neurons and network dynamics 11
- 1.1 Biological neurons and how to model them 11
  - 1.1.1 Modeling using equivalent circuits 12
- 1.2 Integrate-and-fire neuron models 14
  - 1.2.1 Leaky integrate-and-fire (LIF) 14
- 1.3 Modeling synaptic response 16
- 1.4 Additional calculation for the LIF model 18
- 1.5 Firing rate models 19

2 Synaptic plasticity and cognitive processes 23
- 2.1 Short-term synaptic plasticity (STP) 23
  - 2.1.1 STP and Working Memory 26
- 2.2 Spike timing-dependent synaptic plasticity (STDP) 28
- 2.3 Structural plasticity and learning 29

II GPU-based simulations of spiking networks 33

3 Simulation technology: CPU, GPU-based and beyond 35
- 3.1 From CPUs to GPUs and neuromorphic computing: a brief introduction 35
  - 3.1.1 GPU-based simulation 37
- 3.2 NEST GPU 38
  - 3.2.1 Spike delivery algorithm 39
  - 3.2.2 Network construction algorithm 40

4 Spiking network models and validation techniques 43
4.1 Balanced network 43
4.2 Cortical microcircuit 44
4.3 Multi-area model 46
4.4 Validation techniques and benchmarking 47
4.4.1 Getting the distributions 47
4.4.2 Getting the metric to compare the distributions 50

5 NEST GPU validation 53
5.1 Single neuron simulation 53
5.2 Simulations of the cortical microcircuit model 55
5.3 Simulations of the multi-area model 57

6 NEST GPU performance evaluation 65
6.1 Phases of the simulation 65
6.2 Simulations of the cortical microcircuit model 66
6.2.1 Network construction time performance 66
6.2.2 Simulation performance 70
6.3 Simulations of the multi-area model 74
6.4 Simulations of the balanced network model: NEST GPU scaling performance 78

III Synaptic mechanisms and cognitive processes: from the synaptic theory of Working Memory to a framework for learning through Structural Plasticity 81

7 An STP-driven Working Memory spiking model 83
7.1 Synaptic theory of Working Memory 83
7.2 Spiking model and reproduction of the results 85
7.2.1 Model description 85
7.2.2 Results of the simulation 94
7.3 Discussion of the results 103

8 Structural plasticity and learning: theory 107
8.1 Model introduction 107
8.2 Model overview 108
8.3 Discrete rate model 112
8.4 Continuous model 118
8.4.1 Poisson distribution of incoming connections per neuron . . 121
8.4.2 Connection Rewiring 122
8.4.3 Introduction of noise into input patterns 123
8.4.4 Summary of theoretical model equations 124
8.5 Recurrent neuron model 125

9 Structural plasticity and learning: simulations 133
9.1 Two neuron population simulations 134
9.1.1 Comparison between continuous and discrete firing rate . . 134
9.1.2 Comparison between theoretical predictions and simulation results 136
9.1.3 Impact of synaptic rewiring 139

Discussion 143

Discussion 145
GPU-based simulations using NEST GPU 145
Short-term plasticity and working memory 148
Structural plasticity and learning 149
Outlook 151

Bibliography 153

A Balanced network model 171

B Validation of the multi-area model 175

C Implementation of STP model 189

D Working memory spiking model 193
D.1 Changes of simulation time step and connectivity 193
D.2 Stochastic integration methods for neuronal dynamics integration: a results' comparison 194
D.3 Applicability of this model on a sequence learning network 197
D.3.1 Sequential stimulation 199
D.3.2 Selective populations down-scaling 202

E Lognormal distribution of the firing rate 205

F Estimation of the variance of k 209

G Noise addition during test phase 213
G.1 Calculation of the mean value of k'_t over the rewiring steps 214

List of Publications

Publications discussed in this thesis:

- [1] Golosio B, **Tiddia G**, De Luca C, Pastorelli E, Simula F and Paolucci PS (2021) Fast Simulations of Highly-Connected Spiking Cortical Models Using GPUs. Front. Comput. Neurosci. 15:627620. doi: `https://doi.org/10.3389/fncom.2021.627620`

- [2] **Tiddia G**, Golosio B, Albers J, Senk J, Simula F, Pronold J, Fanti V, Pastorelli E, Paolucci PS and van Albada SJ (2022) Fast Simulation of a Multi-Area Spiking Network Model of Macaque Cortex on an MPI-GPU Cluster. Front. Neuroinform. 16:883333. doi: `https://doi.org/10.3389/fninf.2022.883333`

- [3] **Tiddia G**, Golosio B, Fanti V and Paolucci PS (2022) Simulations of working memory spiking networks driven by short-term plasticity. Front. Integr. Neurosci. 16:972055. doi: `https://doi.org/10.3389/fnint.2022.972055`

- [4] Golosio B, Villamar J, **Tiddia G**, Pastorelli E, Stapmanns J, Fanti V, Paolucci PS, Morrison A and Senk J (2023) Runtime Construction of Large-Scale Spiking Neuronal Network Models on GPU Devices. Appl. Sci. 13(17):9598. doi: `https://doi.org/10.3390/app13179598`

- [5] **Tiddia G**, Sergi L and Golosio B (2023) A theoretical framework for learning through structural plasticity. arXiv:2307.11735 [q-bio.NC]. doi: `https://doi.org/10.48550/ARXIV.2307.11735`

Abstract

The brain is one of the most intricate systems and the quest for simulating the neuronal dynamics has led to the development of several approaches and simulation tools able to represent the behavior of portions of the brain with different levels of detail. Among the different techniques that we can adopt to shed light on neuronal dynamics, mean-field models and spiking neural networks are two of the most relevant, and are introduced in Part I of this doctoral thesis. In particular, spiking neural network models have become an effective tool to study brain functions as they capture several aspects of natural neural networks, with every neuron being characterized by a membrane potential and a mechanism to emit electrical pulses for communication with other neurons. While mean-field approaches are suitable for the simulations of models of the entire brain with population resolution, spiking neural networks are able to simulate portions of the brain at cellular level. However, recent computing technologies are paving the way for large-scale simulations through the usage of cutting-edge supercomputer clusters, and it is of fundamental importance for computational neuroscientists to have tools able to take advantage of these technologies. In recent years, Graphical Processing Units (GPUs) established themselves as promising hardware to be employed for such simulations, thanks to their high degree of parallelism, and several GPU-based simulation codes have been developed. In Part II of this thesis, we describe the GPU code for spiking neural network simulations NEST GPU, which is able to efficiently exploit GPU hardware spanning from consumer GPUs to data-center cards employed in MPI-GPU clusters. The thesis is devoted both to evaluate the performance of such a simulator in the simulation of neuroscientifically relevant models, and, most importantly, to validate the results of the neuronal dynamics with respect to established spiking network simulators such as NEST.

To better understand the link between brain functioning and high-level cognitive processes with low-level neuronal activity, there is the need to provide realistic models both for the neurons and the synapses. Indeed, there is broad consensus in the neuroscientific community that synaptic mechanisms, such as short-term synaptic plasticity and structural synaptic plasticity underlie cognitive processes like working memory and learning. Part III of this thesis is devoted to developing simulation and theoretical frameworks that shed light on the possible relation between these synaptic mechanisms and the previously mentioned cognitive processes. In particular, Chapter 7 focuses on the simulation of a working memory

spiking network driven by short-term plasticity (STP), which is believed to be responsible for the activity-silent mechanisms that characterize working memory networks, while Chapter 8 presents a theoretical framework able to describe a learning process mediated by structural synaptic plasticity, evaluating the memory capacity of the network as a function of the simulation parameters.
This thesis aims to start facing the challenge of the study of high-level cognitive processes through simulations of large-scale neuronal networks. In a framework in which computing technologies are opening to the realm of large-scale simulations through the usage of GPU clusters, there is a need for simulators capable of exploiting this fast-growing hardware being efficient and, more importantly, reliable. Additionally, modeling neuron and synaptic scale mechanisms can shed light on their impact on high-level cognitive processes such as learning and memory and, together with large-scale simulations at neuron resolution, it would be possible to estimate the relation of these mechanisms and the dynamics of neuronal networks representing a significant portion of the brain. These works are oriented toward the development of more detailed network models, which will pave the way for the usage of these tools in medicine as support for novel therapies.

Part I

Introduction

Chapter 1

Modeling neurons and network dynamics

Summary

This chapter introduces the topic of neuron modeling and provides the equations of neuronal dynamics used in the thesis to perform spiking network simulations. The chapter, starting with a brief description of the biological neuron and the first neuron modeling approaches, introduces one of the most simple spiking neuron models, i.e., the leaky integrate-and-fire (LIF). Moreover, it will describe how we can model synapses, distinguishing them from current and conductance-based approaches. Finally, firing rate models will be briefly introduced.

1.1 Biological neurons and how to model them

In order to model neuron dynamics, we have to be aware of how a neuron works and what mechanisms it exploits to communicate with other neurons. Neurons are cells that, together with glial cells, make up the neural tissue, which is the main component of the nervous system. They are the elementary units of such a system and are of large number indeed: a human brain contains around 86×10^9 neurons [6]. Each neuron is provided with thousands of synapses to communicate with other neurons[1], forming intricate networks. Neurons communicate using a combination of electrical and chemical signals which are delivered to all the neurons they are connected with. This peculiar communication method requires a likewise peculiar cell structure. Indeed, neurons are composed of three basic structures such as the *soma*, which is the cell body, and extensions such as the *axon* and the *dendrites*, which have the role of transmitting or receiving signals, respectively. Axons and dendrites are not simple extensions of the body cell, since some cellular components are only located in the latter, and also the cellular membrane itself has different characteristics.

[1]It is estimated that neurons of the cerebral neocortex have, on average, 7000 synapses each [7].

The effect of the electric pulses exchanged between neurons results in a change in the neuronal membrane electric potential. Indeed, a resting neuron shows a membrane electric potential because of a difference in concentration of certain ions between the inside and the outside of the membrane, which leads to a difference in the membrane electric potential according to the Goldman equation. The main ions involved in this process are sodium (Na^+) and potassium (K^+), with a resting membrane potential of around -70 mV. A neuron emits a short and intense signal called *action potential* (or *spike*) when the membrane is sufficiently depolarized. Indeed, a depolarization of the membrane opens voltage-gated channels leading to a flux of sodium ions towards the outside of the neuron. A sufficiently depolarized membrane (around -55 mV) leads to a positive feedback mechanism and to the emission of an action potential that propagates toward the axon to be transmitted to other neurons. After that, the membrane potential is restored to the resting value through specific ion pumps.
In order to provide detailed neuron models able to explain the dynamics of a neuron membrane potential we can adopt a relatively simple yet powerful approach, based on the construction of a so-called *equivalent circuit*, which has the aim of capturing the main aspects of the mechanisms briefly discussed above that describe the neuronal dynamics. Next, it will be shown a typical example of a neuron model based on the design of an equivalent circuit, i.e., the Hodgkin-Huxley model. This model has not been employed for the aims of this thesis, but it is of fundamental importance since it represents the first neuron model with a high predictive power.

1.1.1 Modeling using equivalent circuits

The idea of describing the neuronal membrane and its dynamics using equivalent circuits comes from a phenomenological approximation for the description of membrane and ion channels. First of all, the membrane in such models is described as a capacitor, since its impermeability prevents ions from passing through it and the charge is represented by the ion concentrations in proximity to the inside and outside of the membrane. The only elements that enable an ion flux through the membrane are the ion channels, which can be modeled using variable resistors and voltage generators to describe the ion flux based on the membrane potential.
The following is a typical example of a neuron model described by an equivalent circuit. Developed in 1952, the Hodgkin-Huxley (HH) model [8] is a semi-empirical model that takes into account the dynamics of sodium and potassium ions, together with a non-voltage dependent leakage channel, which considers minor contributions driven by the rest of ion species. Figure 1.1 shows the circuit scheme of the model. Supposing that an input current I arrives from the outside of the membrane, we can use the conservation of charge to derive a differential equation on the membrane potential V_m. The equation will have the form $C_\mathrm{m}\dot{V}_\mathrm{m} = I - I_\mathrm{K} - I_\mathrm{Na} - I_\mathrm{L}$, and the ion currents can be modeled using channel conductance.

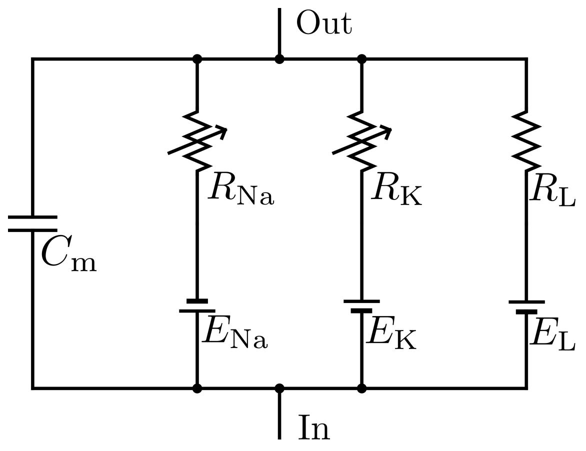


Figure 1.1: Equivalent circuit for the Hodgkin-Huxley model. Out and In represent, respectively, the outside and the inside of the neuron. Generator and resistive elements represent sodium (Na), potassium (K), and leakage (L) channels.

The set of equations describing the HH model is

$$\begin{aligned}
C_{\mathrm{m}}\dot{V}_{\mathrm{m}} &= I - g_{\mathrm{K}}(V_{\mathrm{m}} - E_{\mathrm{K}}) - g_{\mathrm{Na}}(V_{\mathrm{Na}} - E_{\mathrm{Na}}) - g_{\mathrm{L}}(V_{\mathrm{m}} - E_{\mathrm{L}}) \\
&= I - g_{\mathrm{K}\infty}n^4(V_{\mathrm{m}} - E_{\mathrm{K}}) - g_{\mathrm{Na}\infty}m^3h(V_{\mathrm{Na}} - E_{\mathrm{Na}}) - g_{\mathrm{L}}(V_{\mathrm{m}} - E_{\mathrm{L}}) \\
\dot{n} &= \alpha_n(V_{\mathrm{m}})(1-n) - \beta_n(V_{\mathrm{m}})n \\
\dot{m} &= \alpha_m(V_{\mathrm{m}})(1-m) - \beta_m(V_{\mathrm{m}})m \\
\dot{h} &= \alpha_h(V_{\mathrm{m}})(1-h) - \beta_h(V_{\mathrm{m}})h
\end{aligned} \tag{1.1}$$

where the α and β functions represent the nonlinear voltage dependence and are obtained through experimental observations. The interesting part is the fact that using the variables n, m, h we are able to provide the description of sodium and potassium channel dynamics: n and m are defined as *potassium activation gate* and *sodium activation gate* since their value changes as a function of the membrane potential. The variable h is called *sodium inactivation gate* and regulates the dynamics of the channel as soon as the action potential is emitted. This variable leads to a short time period, called *refractory period*, during which the neuron is not able to depolarize the membrane.
Such a model is able to describe in detail the dynamics of the membrane potential of a neuron and the mechanism of spike emission by modeling the dynamics of the most relevant ion channels. However, we can guess that simulating large-scale networks of HH neurons can be arduous since, to compute the dynamics of each neuron, a system of four differential equations has to be solved numerically. Thus, simpler neuron models are needed in order to pave the way for large-scale simulation within a reasonable computing time, still capturing the main aspects of the dynamics of a neural network. These neuron models are described in the next section.

1.2 Integrate-and-fire neuron models

Although the Hodgkin-Huxley model provides a detailed description of the dynamics of a neuron membrane potential, and several models taking account of diverse ionic currents have been developed during the last decade, the nonlinearity of their differential equations can result in computationally expensive simulations when a large number of neurons is employed. For this reason, large-scale spiking neural network simulations rarely employ this model. Indeed, the need for building large-scale networks of neurons within a reasonable time-to-solution time requires a compromise on single-neuron modeling. In this regard, *integrate-and-fire* models were introduced to provide a simplified yet computationally efficient description of the dynamics of single neurons. Starting from the work of Lapicque in 1907 [9] and developed around the 1960s (see [10] for a nice report), integrate-and-fire models describe the relationship between current inputs received by the neuron and its membrane potential using differential equations (or systems of differential equations), with membrane potential changes that can be modeled in different ways depending of a *voltage threshold*, a parameter that is not present in models such as the HH. When the membrane potential is below the threshold, if the neuron is not injected by a current, the membrane behaves passively acting like a discharging capacitor tending towards a resting membrane potential value. When the threshold is reached (ergo when the neuron has received enough current input), the model *fires* a spike and suddenly resets the membrane potential to a baseline level. Thus, there is no modeling of the ion channel dynamics that lead to the emission of the action potential, nor to the dynamics that take place after the emission during which ion concentration is restored before the neuron is able to fire again. The latter is simply modeled defining a *refractory time*, during which the model does not integrate any input.
Thus, integrate-and-fire models are able to provide us only the time at which the spike is fired. Indeed, these models assume that, since the shape of the action potential is approximately the same among all the spikes fired by neurons, the information to be transmitted to other neurons is not contained in the shape itself, but rather by the time at which spikes are emitted. Ergo, the description of the action potential can be discarded and only the spike time can be considered as a relevant quantity able to make neurons communicate with each other.
The next sections describe one of the most used integrate-and-fire models, i.e., the leaky integrate-and-fire, mainly following the text Neuronal Dynamics [11].

1.2.1 Leaky integrate-and-fire (LIF)

The leaky integrate-and-fire model (LIF) is the most simple model of this kind of neurons and is described by the most simple equivalent circuit for the neuronal membrane, i.e., an RC circuit. The circuit is also provided with a voltage source to have a defined resting membrane potential. Figure 1.2 provides a scheme of the model.

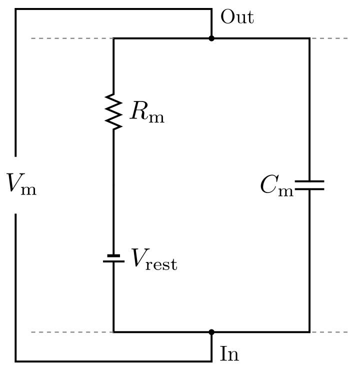


Figure 1.2: Leaky integrate-and-fire neuron model equivalent circuit. The circuit comprises a capacity (C_m), a resistance (R_m) and a generator with voltage V_rest. The membrane potential V_m measured between the inside and the outside of the membrane. The gray lines indicate that this circuit represents a portion of the whole neuronal membrane.

To derive the equations describing the dynamic of the LIF neuron model we have to consider an input current $I(t)$ which, using the law of current conservation, can be split as follows:

$$I(t) = I_{R_\mathrm{m}} + I_{C_\mathrm{m}} = \frac{V_\mathrm{m} - V_\mathrm{rest}}{R_\mathrm{m}} + C_\mathrm{m}\dot{V}_\mathrm{m} \tag{1.2}$$

where $\dot{V}_\mathrm{m}$ is the derivative of the membrane potential with respect to the time t. We can write Equation (1.2) in a more usual way by calling $\tau_\mathrm{m} = R_\mathrm{m}C_\mathrm{m}$ and putting the derivative of the membrane potential to the left side, so that

$$\tau_\mathrm{m}\dot{V}_\mathrm{m} = -(V_\mathrm{m}(t) - V_\mathrm{rest}) + R_\mathrm{m}I(t) \tag{1.3}$$

From the equation above we derive the description of the behavior of the membrane below the voltage threshold V_th

$$V_\mathrm{m}(t) = V_\mathrm{rest} + (V_\mathrm{m}(t_0) - V_\mathrm{rest})\exp\left(\frac{t - t_0}{\tau_\mathrm{m}}\right) \tag{1.4}$$

when $t > t_0$. Thus, according to Equation (1.4), if no input is provided after a certain time t_0, the membrane potential of the neuron decays exponentially with a time constant τ_m reaching the resting value V_rest.
If the input signal received by the neuron is sufficient for it to reach the potential threshold V_th, as discussed before, the membrane potential drops to a reset value V_reset suddenly after the time of spike emission t_f

$$\lim_{\epsilon \to 0^+} V(t_f + \epsilon) = V_\mathrm{reset} \tag{1.5}$$

and remains clamped at this value after the spike emission for a time t_ref, which is the refractory time.
This model has been employed in large-scale models such as the cortical microcircuit of [12], in particular for its simplicity and the fact that it can be integrated

using efficient integration methods such as [13]. Moreover, it is possible to obtain closed form expressions to derive, for instance, the external current needed to elicit a defined change in the membrane potential. We see some additional calculations on this model later in this chapter, to support the description of the spiking network models presented in this thesis.
However, such an elementary model has some limits. For example, the linearity of the model makes the response much simpler than it is in reality, discarding the nonlinear behavior of the membrane dynamics that show up in particular when the membrane potential is near the threshold. Moreover, in the case of spike emission, after the neuron membrane potential is reset, all the information related to the previous activity is lost. Indeed, it has been shown that neurons have some mechanisms related to the possibility of keeping a "memory" of their previous synaptic activity since repetitive stimulus leads to a lower response over time. This mechanism is called *adaptation*, and can be added into integrate-and-fire neuron models with nonlinearity. Thus, instead of having a neuron model described by a linear differential equation, we can have something like

$$\tau_{\mathrm{m}}\dot{V}_{\mathrm{m}} = f(V_{\mathrm{m}}) + R_{\mathrm{m}}I(t) \tag{1.6}$$

with $f(V_{\mathrm{m}})$ being a nonlinear term. An example of an integrate-and-fire model able to describe this mechanism is called Adaptive Exponential integrate-and-fire (AdEx) [14]. Such a model is able, after appropriate parameter tuning, to show a voltage trace compatible with the HH model when the same input current is given, meaning that an integrate-and-fire model, despite the simplicity related to the threshold mechanism and the need for a compromise between computational cost and realistic behavior, can be able to reproduce analogous results of a more detailed semi-empirical model.
For the purpose of this thesis, we employ leaky integrate-and-fire neurons, in particular, because of the possibility of finding closed form solutions for a proper choice of the inputs to be given to the neurons. In this regard, some examples will be shown later in this chapter. The next section shows how to model synaptic response, i.e. how we can model the shape of the postsynaptic potential (PSP) when a neuron receives a spike.

1.3 Modeling synaptic response

Here we present a brief introduction to synaptic response, functional to the neuron models employed in this work. For an exhaustive discussion please refer to Modeling Synapses [15], a chapter of [16], from which the following discussion is drawn.
As we already discussed in the first section, the presynaptic spike depolarizes the synaptic terminal, causing the influx of Calcium ions that trigger the neurotransmitter release. Then, neurotransmitters bind to receptors in the postsynaptic terminal, leading to an ionic current across the postsynaptic neurons and polarizing or depolarizing its membrane. After that, neurotransmitters return to the presynaptic terminal. This mechanism has been modeled under the name of short-term

synaptic plasticity (STP) and will be discussed in detail in the next chapter.
To model the postsynaptic response we can apply some simple phenomenological models. The most simple way to model synaptic transmission would be to increase the membrane potential of the postsynaptic neuron by a determined amount. However, to offer a more realistic model we can link the current transmitted by a presynaptic neuron j to a change in the synaptic conductance g_{syn} of the postsynaptic neuron i, so that

$$I_i(t) = \sum_f g_{\text{syn},i}(t - t_f^{(j)})(V_{\text{m},i}(t) - E_{i,j}^{\text{syn}}) \tag{1.7}$$

where the index $t_f^{(j)}$ is the time at which a spike is fired from the presynaptic neuron j and $E_{i,j}^{\text{syn}}$ is the reversal potential, i.e., the membrane potential at which the ion current on the membrane reverses. $g_{\text{syn}}(t)$ can be modeled using different functions.
A first detail that can be added in this regard can be an exponential decay using a separate time constant suddenly after a step-like increase of the conductance. Thus

$$g_{\text{syn}}(t) = \begin{cases} \bar{g}_{\text{syn}} e^{-(t-t_f)/\tau_{\text{syn}}} & \text{for } t > t_f \\ 0 & \text{for } t < t_f \end{cases} \tag{1.8}$$

The conductance thus raises from 0 to $\bar{g}_{\text{syn}}$ at t_f, and it decreases exponentially with a time constant τ_{syn}. This model can describe a very quick neurotransmitter release and binding to the postsynaptic ion channel, followed by a slower closure of the ion channels, and thus a reduction of the conductance. It is a good approximation for certain types of neurotransmitters but can be ineffective in describing a slow-rising postsynaptic potential.
To simulate a finite rise time, we can use an *alpha function*, as described in the following equation

$$g_{\text{syn}}(t) = \bar{g}_{\text{syn}} \frac{t - t_f}{\tau_{\text{syn}}} e^{1-(t-t_f)/\tau_{\text{syn}}} \tag{1.9}$$

However, in this case, we do not have control of the rising and decreasing phase separately, since we are using the same synaptic time constant τ_{syn}. To add this detail, we have to define a time constant for rising and decay and sum the two exponential contributions as follows

$$g_{\text{syn}}(t) = A(e^{-(t-t_f)/\tau_{\text{decay}}} - e^{-(t-t_f)/\tau_{\text{rise}}})\bar{g}_{\text{syn}} \tag{1.10}$$

where A is a normalization factor so that the amplitude of the postsynaptic potential is equal to $\bar{g}_{\text{syn}}$. The time at which the postsynaptic potential reaches the peak (t_{peak}) can be derived by setting the derivative of Equation (1.10) to zero, obtaining

$$t_{\text{peak}} = t_f + \frac{\tau_{\text{rise}}\tau_{\text{decay}}}{\tau_{\text{decay}} - \tau_{\text{rise}}} \ln\left(\frac{\tau_{\text{decay}}}{\tau_{\text{rise}}}\right) \tag{1.11}$$

and by imposing that $g_{\text{syn}}(t_{\text{peak}}) = \bar{g}_{\text{syn}}$ we have that the normalization factor is

$$A = \frac{1}{e^{-(t_{\text{peak}}-t_f)/\tau_{\text{decay}}} - e^{-(t_{\text{peak}}-t_f)/\tau_{\text{rise}}}} \tag{1.12}$$

This approach is one of the most flexible since it enables finding a close form for the time at which the postsynaptic potential has a peak and is able to describe efficiently the postsynaptic response.
There is a further approximation that can be done in terms of synaptic transmission, which models directly the current injected into the postsynaptic neuron instead of the conductance. These models are known as *current-based* as opposed to the *conductance-based* approach discussed so far. In this approximation, we can refer to Equation (1.7) and suppose that there is no voltage-dependent effect, which can be a good approximation in case of small depolarization. In the context of this thesis, the vast majority of neurons employed will have a current-based synaptic response, mainly modeled with exponential decay similar to what is described in Equation (1.8), so that:

$$I_{\text{syn}}(t) = \begin{cases} \bar{I}_{\text{syn}} e^{-(t-t_f)/\tau_{\text{syn}}} & \text{for } t > t_f \\ 0 & \text{for } t < t_f \end{cases} \tag{1.13}$$

where the current raises from 0 to $\bar{I}_{\text{syn}}$ at time t_f, and it decreases exponentially with a time constant τ_{syn}.

1.4 Additional calculation for the LIF model

As mentioned in Section 1.2.1, with the LIF model we are able to find useful analytic expressions. Here we provide a relatively simple example, that is employed in several network models that will be described in this thesis.
This expression is related to the tuning of synaptic weights, and in particular to the amount of current needed to elicit a given postsynaptic potential.
To find this expression we can start considering a LIF neuron model with exponential current-based postsynaptic response. According to the equation describing the sub-threshold dynamics of the LIF neuron model we have

$$\tau_{\text{m}} \frac{dV}{dt} = -(V - V_{\text{rest}}) + RI(t) \tag{1.14}$$

where $R = \tau_{\text{m}}/C_{\text{m}}$ and $I(t) = I_0 e^{-t/\tau_{\text{s}}}$, i.e., the neuron receives an input current of amplitude I_0 at $t = 0$. Let us also consider $V_{\text{rest}} = 0$ for simplicity. τ_{m} and τ_{s} represent the membrane time constant and the synaptic current time constant respectively. The general solution of Equation (1.14), considering the exponential behavior of I_0 is

$$V(t) = V_0 e^{-t/\tau_{\text{m}}} + \frac{\tau_{\text{m}}}{C_{\text{m}}} \frac{\tau_{\text{s}}}{\tau_{\text{s}} - \tau_{\text{m}}} I_0 e^{-t/\tau_{\text{s}}} \tag{1.15}$$

and considering $V(0) = 0$

$$V(t) = -\frac{\tau_{\text{m}}}{C_{\text{m}}} \frac{\tau_{\text{s}}}{\tau_{\text{s}} - \tau_{\text{m}}} I_0 \left[e^{-t/\tau_{\text{m}}} - e^{-t/\tau_{\text{s}}} \right] \tag{1.16}$$

The Equation (1.16) represents the postsynaptic response to a current stimulus of amplitude I_0 reached at $t = 0$. Imposing the derivative of Equation (1.16) to zero enables us to find the time t_{max} at which the membrane reaches the maximum depolarization because of the stimulus applied. So we have

$$t_{\mathrm{max}} = \ln\left[\left(\frac{\tau_{\mathrm{m}}}{\tau_{\mathrm{s}}}\right)^{\tau_{\mathrm{m}}\tau_{\mathrm{s}}/(\tau_{\mathrm{m}}-\tau_{\mathrm{s}})}\right] \tag{1.17}$$

and

$$V(t_{\mathrm{max}}) = \frac{\tau_{\mathrm{m}}\tau_{\mathrm{s}}I_0}{C_{\mathrm{m}}(\tau_{\mathrm{s}}-\tau_{\mathrm{m}})}\left[\left(\frac{\tau_{\mathrm{m}}}{\tau_{\mathrm{s}}}\right)^{-\tau_{\mathrm{m}}/(\tau_{\mathrm{m}}-\tau_{\mathrm{s}})} - \left(\frac{\tau_{\mathrm{m}}}{\tau_{\mathrm{s}}}\right)^{-\tau_{\mathrm{s}}/(\tau_{\mathrm{m}}-\tau_{\mathrm{s}})}\right] \tag{1.18}$$

The equation above represents the amplitude of the postsynaptic potential elicited by a postsynaptic current of amplitude I_0. Thus, the value of I_0 needed to obtain a postsynaptic potential amplitude of $V(t_{\mathrm{max}})$ is

$$\alpha = \frac{I_0}{V(t_{\mathrm{max}})} = \left\{\frac{\tau_{\mathrm{m}}\tau_{\mathrm{s}}}{C_{\mathrm{m}}(\tau_{\mathrm{s}}-\tau_{\mathrm{m}})}\left[\left(\frac{\tau_{\mathrm{m}}}{\tau_{\mathrm{s}}}\right)^{-\tau_{\mathrm{m}}/(\tau_{\mathrm{m}}-\tau_{\mathrm{s}})} - \left(\frac{\tau_{\mathrm{m}}}{\tau_{\mathrm{s}}}\right)^{-\tau_{\mathrm{s}}/(\tau_{\mathrm{m}}-\tau_{\mathrm{s}})}\right]\right\}^{-1} \tag{1.19}$$

which is the analytic expression used in this model to get the value of the synaptic efficacy needed to elicit appropriate postsynaptic potentials. In fact, the term α represents the variation of input current needed to have a unit of variation of the postsynaptic potential. Thus, since usually simulators express the synaptic weights in current units, we can use the expression above to tune the synaptic weights to have the wanted response in terms of membrane potential.

1.5 Firing rate models

We discussed how a biological neuron can be modeled by using equivalent circuits and integrate-and-fire neuron models, which are able to provide the times at which spikes are emitted. We can further increase the level of abstraction by characterizing a neuron using a different state variable, i.e. the firing rate. Thus, instead of representing the neuron behavior through its spike sequence, we can describe it using the rate of spike emission, identified as the ratio between the number of spikes emitted and the elapsed time.
The development of firing rate models is justified by the large number of inputs a neuron can receive (a neuron can have around seven thousand incoming connections); if a neuron receives uncorrelated presynaptic input from a large number of incoming synapses, the overall input of the neuron will grow approximately linearly with the number of synapses, with the standard deviation of the input growing as the square root of this number [17]. Thus a firing-rate-based description, in such a case, should be compatible with a spike-based description.
To model the input incoming to a neuron we need the firing rate of the neurons that target the one under consideration and the synaptic efficacy (or weight) of the respective synapses. These quantities can be described as vectors: we call $\mathbf{u}$ the output rate of the presynaptic neurons and $\mathbf{w}$ the synaptic weights of their synapses. Thus, the total input can be simply described by the scalar product $\mathbf{w}\cdot\mathbf{u}$. Figure 1.3 shows a scheme of inputs from presynaptic neurons to a postsynaptic neuron.

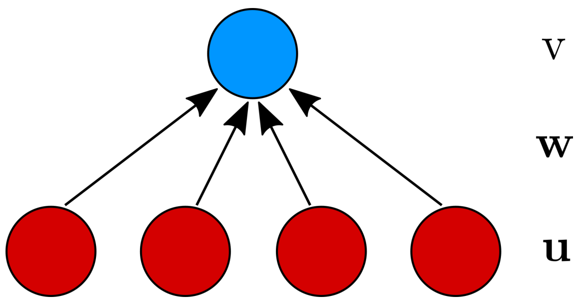


Figure 1.3: Scheme representing a postsynaptic neuron (blue dot) being targeted by presynaptic neurons (red dots) through synaptic connections (black arrows). The synaptic weights of the connections are indicated with the vector $\mathbf{w}$, whereas the output rate of the presynaptic neurons is indicated with $\mathbf{u}$. The output of the postsynaptic neuron is indicated with v.

If we consider an exponential synaptic response, similar to Equation (1.8) but describing the synaptic current, we can write

$$\tau_{\text{syn}} \frac{dI_{\text{syn}}}{dt} = -I_{\text{syn}} + \mathbf{w} \cdot \mathbf{u} \tag{1.20}$$

Once the total input is evaluated, we have to estimate the response of the neuron through a *activation function* $\mathcal{F}(I_{\text{syn}})$. Among the most common activation functions are the Heaviside step function $\mathcal{H}(x)$ and the threshold-linear (or ReLU)

$$\mathcal{F}(x) = \alpha \max\{0, x\} \tag{1.21}$$

where α is a multiplicative coefficient.
Generally, in such models, neurons do not respond instantaneously to the synaptic input, and thus the firing rate follows the linear differential equation

$$\tau_{\text{v}} \frac{d\text{v}}{dt} = -\text{v} + \mathcal{F}(I_{\text{syn}}) \tag{1.22}$$

The time constant τ_{v} expresses the time needed for the neuron to reach a steady state firing rate when a constant input is given. Usually, in firing rate models, the time constant of the input current τ_{syn} is considered negligible with respect to τ_{v}, and thus, for $\tau_{\text{syn}} \ll \tau_{\text{v}}$ Equation (1.22) reduces to

$$\tau_{\text{v}} \frac{d\text{v}}{dt} = -\text{v} + \mathcal{F}(\mathbf{w} \cdot \mathbf{u}) \tag{1.23}$$

Indeed, a network behavior can be described by these equations, and steady-state solutions can shed light on the long-term behavior of a network performing determined tasks, as we will see further in this thesis.
In this regard, we will present both feed-forward and recurrent networks, thus a brief introduction of how such networks can be described is provided following [17].
Figure 1.3 shows an example of a feed-forward network, since the flow of the signals has only one direction, i.e., from the population below to the neuron above. Considering a population of postsynaptic neurons instead of having a single cell

and calling $\mathbf{u}$ the rate of the population below (ergo the presynaptic one) and $\mathbf{v}$ the rate of the population above, we can write

$$\tau_v \frac{d\mathbf{v}}{dt} = -\mathbf{v} + \mathcal{F}(\mathbf{W} \cdot \mathbf{u}) \tag{1.24}$$

where $\mathbf{W} \cdot \mathbf{u} = \sum_{b=1}^{N_u} W_{ab} u_b$ for $a = 1, \ldots, N_v$. Indeed, in the case of a feed-forward network, the steady-state solution is similar to the one shown before, however, this is not true for recurrent networks. In a recurrent network, a neuron population can have self-connections, and thus the output rate of the neurons depends on the rate of the population itself. Figure 1.4 shows a scheme of a recurrent network.

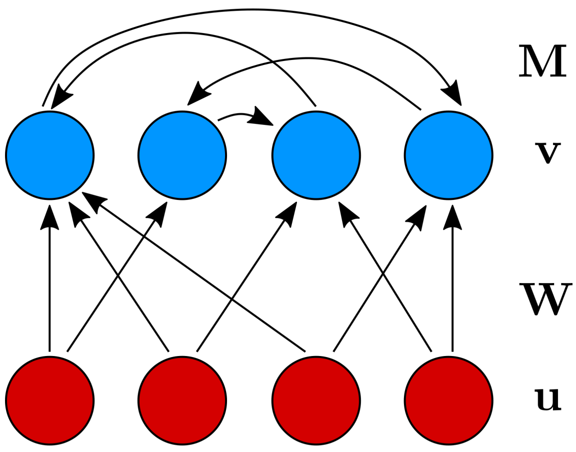


Figure 1.4: Scheme of a recurrent network with two neuron populations. The synaptic weights of the feed-forward connections are indicated with the matrix $\mathbf{W}$, whereas $\mathbf{M}$ indicates the set of synaptic weights of the recurrent connections.

Calling $\mathbf{M}_{a,a'}$ the matrix which contains the synaptic weights from output neurons a and a', Equation (1.24) becomes

$$\tau_v \frac{d\mathbf{v}}{dt} = -\mathbf{v} + \mathcal{F}(\mathbf{W} \cdot \mathbf{u} + \mathbf{M} \cdot \mathbf{v}) \tag{1.25}$$

Indeed, the matrix $\mathbf{M}$ here considers only self-connections of the output population, but it can also be considered in a similar way as the contribution of a second population (e.g. an inhibitory one) which is stimulated by the output population and then projects to it an inhibitory feedback. Such an architecture is widely used when a realistic architecture of a neural circuit has to be represented since the addition of an inhibitory population grants the stability of the excitatory (i.e. output) population and can also lead to mechanisms of competition through lateral inhibition that take place in the human brain.

Chapter 2

Synaptic plasticity and cognitive processes

Summary

Here the synaptic models studied in compliance with this thesis are presented, i.e., short-term synaptic plasticity and structural plasticity. The description of these plasticity mechanisms is then followed by a discussion of the relationship between them and cognitive processes such as working memory and learning.

2.1 Short-term synaptic plasticity (STP)

In this section, short-term plasticity and its phenomenological description are introduced. For further details please see [18–22].
Short-term plasticity (STP) is a mechanism in which the synaptic efficacy temporarily changes with a timescale on the order of hundreds or thousands of milliseconds. This phenomenon is regulated by the amount of synaptic resources (i.e. the neurotransmitters) available in the synapse at the moment of spike emission and by the calcium levels in the presynaptic terminal.
Indeed, the spike arrival at the presynaptic terminal elicits an influx of calcium ions that is responsible for the release of the vesicles in which neurotransmitters are stored. Higher calcium concentration in the terminal leads to a higher fraction of neurotransmitters released. This mechanism is called short-term facilitation (STF). The neurotransmitter release is then followed by a mechanism of calcium removal from the presynaptic terminal to restore its baseline concentration. The amount of neurotransmitters a synapse can contain is limited and the emission of a spike diminishes the number of neurotransmitters available in the presynaptic terminal for further stimulation. This mechanism is called short-term depression (STD). Without synaptic activity, the number of available neurotransmitters in the presynaptic terminal returns to its baseline level. The coupling of these two phenomena leads to a temporary modulation of the synaptic efficacy (i.e. short-term

plasticity).
The first phenomenological model for short-term plasticity was presented in [18]. The model describes mechanisms that take place in the presynaptic terminal, such as the neurotransmitter dynamics, taking into account the fact that neurotransmitters are limited and also the mechanism of neurotransmitter release is triggered by calcium ions. The neurotransmitter dynamics can be described by three normalized quantities that represent the fraction of neurotransmitters in three different states:

- recovered (x), i.e., contained in synaptic vesicles, ready to be released;
- active (y), i.e., released in the synaptic cleft;
- inactive (z), i.e., in the process of returning inside the synaptic vesicles.

The mechanism related to calcium dynamics is described by a normalized quantity called u, which represents the fraction of opened calcium channels. Short-term plasticity can thus be described by the following set of equations:

$$\begin{aligned}
\frac{dx}{dt} &= \frac{z}{\tau_d} - u(t_s)x(t_s-\epsilon)\delta(t-t_s)\\
\frac{dy}{dt} &= -\frac{y}{\tau_{\text{syn}}} + u(t_s)x(t_s-\epsilon)\delta(t-t_s)\\
\frac{dz}{dt} &= \frac{y}{\tau_{\text{syn}}} - \frac{x}{\tau_d}\\
\frac{du}{dt} &= -\frac{u}{\tau_f} + U(1-u)\delta(t-t_s)
\end{aligned} \tag{2.1}$$

Where t_s is the time at which a spike is emitted, $\delta(\cdot)$ is a Dirac delta and τ_{syn}, τ_f and τ_d are the synaptic time constant and the time constants for facilitation and depression mechanism, respectively.
Synaptic modulation is driven by the variable $y(t)$, ergo considering a synaptic efficacy J, the postsynaptic potential would have an amplitude of $Jy(t)$. In this equation, for simplicity, we neglect the indexed i, j indicating the indices of presynaptic and postsynaptic neurons. The quantity u increases by a factor $U(1-u)$ whenever a presynaptic spike arrives, since the spike triggers the opening of calcium channels.
Is it possible to show that such a model can be simplified by adopting a system of two differential equations that describe the behavior of the synaptic resources (x) and the one of utilization factor (u) [18]. Let x be the normalized amount of available resources in the presynaptic terminal and let u be the fraction of resources used in a spike emission. The spike arrival to the synaptic terminal raises the variable u by a quantity $U(1-u)$ (so that u remains normalized) and the amount of resources released is equal to ux. Considering a synapse connecting the presynaptic neuron i and the postsynaptic neuron j, this dynamics can be described by the

following equations [23]:

$$
\begin{aligned}
\frac{du_{i,j}}{dt} &= -\frac{u_{i,j} - U}{\tau_f} + U(1 - u_{i,j}) \sum_s \delta(t - t_s^{(i)}) \\
\frac{dx_{i,j}}{dt} &= \frac{1 - x_{i,j}}{\tau_d} - u_{i,j} x_{i,j} \sum_s \delta(t - t_s^{(i)})
\end{aligned}
\tag{2.2}
$$

where $\delta(\cdot)$ is the Dirac delta function and the sum is over the spike times $t_s^{(i)}$ of the presynaptic neuron i. The synaptic modulation takes place during the spike emission, so that

$$
J_{i,j}(t) = J_{i,j}^{(\mathrm{abs})} u_{i,j}(t - \hat{\delta}_{i,j}) x_{i,j}(t - \hat{\delta}_{i,j}) \tag{2.3}
$$

where $J_{i,j}^{(abs)}$ is the absolute synaptic efficacy for the synapse connecting neurons i to neuron j and $\hat{\delta}_{i,j}$ is the synaptic delay. Thus, when a spike is fired, the synaptic efficacy is described by the product Jux.
Short-term plasticity can show STD-dominated or STF-dominated behaviors depending on the choice of the time constants for the mechanism of depression and facilitation. The former can be observed when the mechanism of neurotransmitter restoration is slower with respect to the mechanism of residual calcium removal after spike emission and vice versa. To give a phenomenological description of STP we can define, as in Equation (2.2), τ_d as the time constant of the process of neurotransmitter restoring and τ_f the time constant for calcium removal mechanism. Thus we can observe STD-dominated dynamics when $\tau_d > \tau_f$ and STF-dominated dynamics when $\tau_d < \tau_f$. Figure 2.1 shows an example of STD-dominated and STF-dominated dynamics, in which two postsynaptic neurons receive signals from a presynaptic neuron through synaptic connections with STP with different facilitation and depression time constants.
As can be seen, the neuron connected with the STF-dominated synapse is able to show a change in the postsynaptic potential after a relatively long period of time (around one second). This is achieved by the fact that the synaptic resources (x) can return quickly to the presynaptic terminal, whereas the calcium concentration in the terminal (u) decreases slowly. Since the STP modulation is driven by the factor ux, this results in a temporary potentiation of the synaptic efficacy. The STD-dominated synapse has the opposite effect, according to which the efficacy strongly decreases when a series of spikes have to be transmitted and slowly returns to the baseline efficacy.

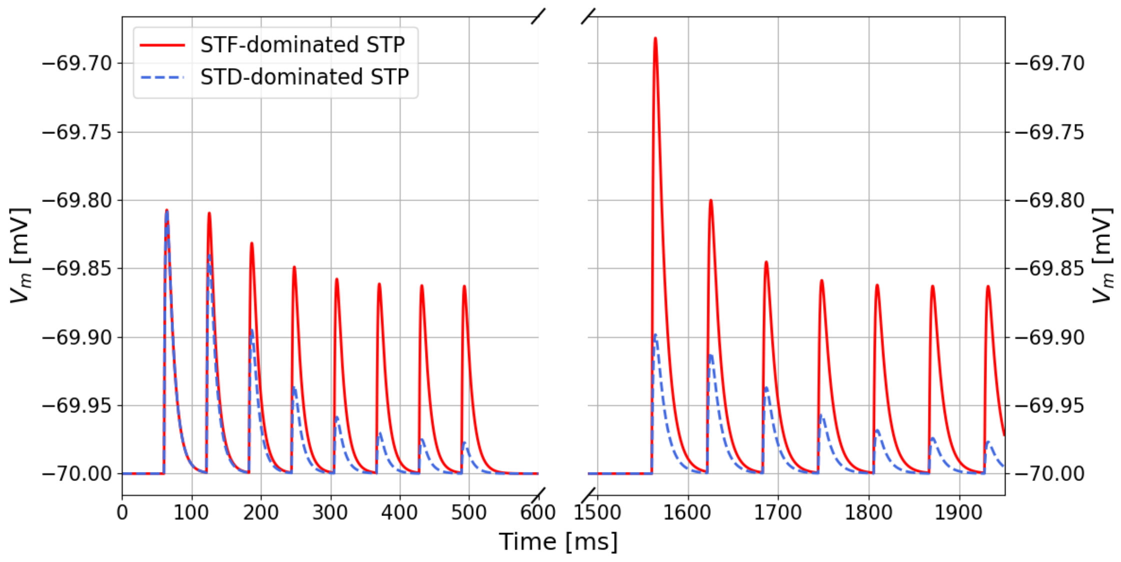


Figure 2.1: Voltage trace of two neurons connected with the same presynaptic neuron with a STP connection. A neuron has a STF-dominated connection (red line), with $\tau_f = 1.5$ s and $\tau_d = 0.2$ s, whereas the other neuron shows a STD-dominated behavior (blue-dotted line), with $\tau_f = 0.2$ s and $\tau_d = 1.5$ s. The time axis is interrupted between 600 and 1500 ms, during which the presynaptic neuron is not active.

2.1.1 STP and Working Memory

We notice in Figure 2.1 that facilitated (i.e., STF-dominated) synapses can show, after a relatively long time of inactivity, an increase in synaptic efficacy. Indeed, it has been estimated that, in our brain, certain regions show a predominance of facilitated synapses, whereas many others show mostly STD-dominated synapses. Among the regions with more relevant occurrence of facilitated synapses, we have the prefrontal cortex (PFC), which is believed to be one of the most important locations of the brain in which Working Memory (WM), which will be briefly introduced here, takes place. The question that arises is whether a synaptic mechanism such as STP, and in particular synaptic facilitation, would be a synaptic correlate of working memory.
Working Memory is a cognitive process able to hold and manipulate information for a short time. It is involved in a vast number of cognitive tasks [24–27] which span from speech to visual and spatial processing. Differently from long-term memory, working memory is a transient phenomenon and it is also believed that it does not entail structural changes to the network.
A classic procedure for studying working memory relies on the so-called delay response tasks. In such a framework, a stimulus is presented for a short time and the related execution of the task can take place only after a delay period. During the delay period, it is experimentally observed, especially in the prefrontal cortex, a neuronal selective persistent spiking activity able to maintain the information previously presented by the stimulus [28–30]. When this activity is somehow suspended (e.g. because of a noise stimulus during the delay period or a too-long

delay) the task is not correctly executed.
The first computational models assumed that this peculiar activity could be entirely maintained with prior long-term synaptic modifications so that, when a stimulus was given into the network, the population encoding for the presented stimulus exhibited a persistent spiking activity [31, 32]. Thus, according to these models, the information was only stored in the spiking activity. However, experimental pieces of evidence show that memory can be also maintained when the enhanced activity is interrupted, suggesting that information is not only stored in the population spiking activity [33] and also that working memory processes can exhibit discrete periodic bursts instead of a persistent activity [34, 35].
In this framework, many studies were conducted to enlighten the role of synaptic plasticity in working memory [36] and some of the proposed models rely on short-term synaptic facilitation [21, 23, 37, 38]. The work of [37] shows that employing synaptic facilitation enables a spiking network to maintain a relevant number of memories at the same time, whereas the same network lacking this kind of plasticity can maintain many fewer memories. Moreover, [38] argues that the nonlinearity of short-term facilitation is essential for displaying a reasonable persistent activity able to retain memory during a delay period.
One of the models that posit a dominant role of synaptic facilitation in working memory is the one of [23], which shows that a spiking network with synaptic facilitation is able to exhibit a bi-stable regime in which it can autonomously retain memories with periodic spiking activity without a significant firing rate increase. Thus, according to this model, memories are stored in a synaptic fashion, with spiking activity functional for synaptic facilitation upkeep. The model is further employed in [39] to study how working memory capacity can be modulated by synaptic facilitation and the network's external excitation.
More recently, [40] (see also [41]) proposed a spiking network model based on a fast expression of Hebbian plasticity, in which memory is retained by oscillatory bursts. Here, the authors proposed a synaptic plasticity model based on a Hebbian learning rule flanked by a short-term plasticity mechanism. This kind of implementation can enable a network to learn new memory representations, whereas using non-Hebbian plasticity needs prior long-term network training.
In the context of this thesis we focused on the theory proposed by [23], first reproducing the results of the work using the spiking network simulator NEST [3] and then evaluating the model itself, to shed light on the limitations and the strength of such approach when comparing it to other working memory models that rely on different neural mechanisms. This study is presented in Chapter 7.
Indeed, the mechanisms involved in the modification of the synaptic efficacy of a synapse are many and take place at different time scales. The next sections present mechanisms that describe synaptic changes at longer time scales, from minutes to days.

2.2 Spike timing-dependent synaptic plasticity (STDP)

As we noticed in the previous section, STP has a time scale of the order of seconds, and it only depends on the emission of a spike from the presynaptic terminal. Such a process is not related to an activity-driven potentiation of the synapses, which is believed to be a key process for learning according to the Hebb postulate [31], which states that there should be a metabolic process triggered by a repetitive and persistent firing from a presynaptic neuron to a postsynaptic one able to increase the efficacy of the signals transmitted.
In this regard, spike timing-dependent plasticity (STDP) is a mechanism that changes the synaptic efficacy of a synapse based on the times at which the presynaptic and the postsynaptic neurons emit a spike. In particular, when the presynaptic neuron fires slightly before the postsynaptic neuron, the synaptic efficacy increases following a long-term potentiation (LTP) process. This is related to the fact that between these events there could be a causal relation, and after repeated events in which presynaptic spikes are followed by postsynaptic activity some metabolic processes are triggered leading to a strengthening of the synaptic efficacy. On the contrary, a synapse that connects neurons that do not show this behavior can decrease its synaptic efficacy, leading to long-term depression (LTD).
The general behavior of STDP can be described by the following [42, 43], given $\Delta t = t_j^f - t_i^f$ the spike times difference between the postsynaptic neuron j and the presynaptic neuron i

$$\begin{aligned} \Delta\omega^+ &= F_+(\omega)e^{-|\Delta t|/\tau_+}\text{if } \Delta t > 0 \\ \Delta\omega^- &= F_-(\omega)e^{-|\Delta t|/\tau_-}\text{if } \Delta t \leq 0 \end{aligned} \tag{2.4}$$

where ω represents the synaptic efficacy and $F(\omega)$ is a time independent function. The only part of the equation that is time dependent is represented by the exponential functions, each one provided with a specific time constant, meaning that this rule is asymmetric with respect to Δt (it is a form of Temporal Asymmetric Hebbian learning).
In 1998, the work of [44] measured the synaptic modifications in neurons of the hippocampus, showing that the spike timing between the presynaptic and postsynaptic neuron determined "the direction and the extent of synaptic changes". Figure 2.2 depicts the experimental results measured in that work, showing that there is an exponential dependence of the change in EPSC with respect to the spike timing. Further, a positive or negative Δt determines an increase or a decrease of the postsynaptic current amplitude, respectively, and a different time constant for the time-dependent exponential behavior, as also reported in Equation (2.4). STDP is here presented as a single phenomenological model, however, several STDP models have been developed in recent times to take into account the several experimental observations regarding the role of the spike timing in long-term plasticity mechanisms (see [42] for a review).

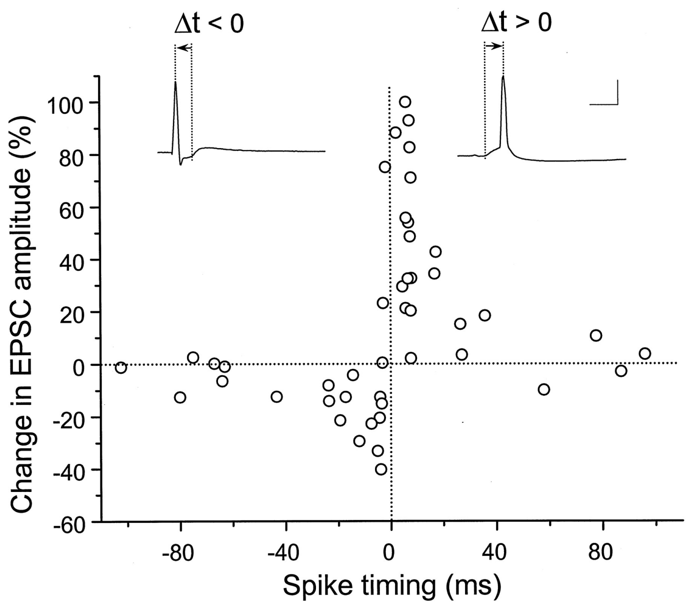


Figure 2.2: Experimental measurements of synaptic potentiation and depression depending on the spike timing. The difference in the excitatory postsynaptic current amplitude (EPSC) is measured after repetitive correlated spiking activity. Spike timing is defined as the time difference between the beginning of the postsynaptic potential and the arrival of the presynaptic spike on the axonal terminal. Figure from [44]. Copyright ©1998 by the Society for Neuroscience.

2.3 Structural plasticity and learning

While in the previous sections we discussed temporary and reversible changes in the synaptic efficacy of an instantiated synapse, here we discuss the structural changes in the synaptic morphology of the brain neuronal network, which takes place during the lifetime, constantly changing the architecture of our brain.
These changes occur at longer time scales than the short or long-term mechanisms mentioned above and consist in the consolidation, creation of new synapses, or erasure of synapses that have not been consolidated. This kind of plasticity is called *structural plasticity*. It can be spontaneous, but also activity-based [45], and it has a key role in the stabilization of new concepts that have to be kept in memory after learning [46].
Structural plasticity is underlain by several biochemical and biophysical mechanisms and, according to [47], we can distinguish between two main categories: *activity-dependent* and *homeostatic* structural plasticity.
Activity-dependent structural plasticity (*Hebbian* in [47]) is related to the growth or retraction of neurites and the formation of new synapses to increase the connectivity between two neurons with high and correlated activity, but also to the deletion of synapses which connect neurons with low and uncorrelated activity. It is known that LTP induced by a strong neural activation is followed by an increase in the number of synapses. Moreover, LTP causes the growth of dendritic spines, which is related to the efficacy of the synapses. The growth of a dendritic

spine is associated with its stability in a way that a grown spine, to which corresponds a higher synaptic efficacy, is more stable, i.e. it is not likely to be removed. Thus, synaptic efficacy can somehow be related to structural plasticity and the probability for a synapse to be consolidated or removed [48,49]. Accordingly, it is also shown in several studies (see, for example, [50]) that inhibiting LTP prevents the increase of the number of dendritic spines, corroborating the link between LTP, structural plasticity and synaptic efficacy changes in previously instantiated synapses. On the other hand, LTD is linked to structural plasticity, showing that it leads to a reduction in the number of synapses [51].
These activity-dependent modifications are then flanked by a homeostatic kind of structural plasticity, which consists of a decrease in connectivity between neurons with high neuronal activities and an increase within neurons with low activity. Thus, this mechanism results in a balancing effect that, through dendritic and axonal modifications, leads to a regime of intermediate postsynaptic neuronal activity (see [45] for a review). These effects are summarized in Figure 2.3.

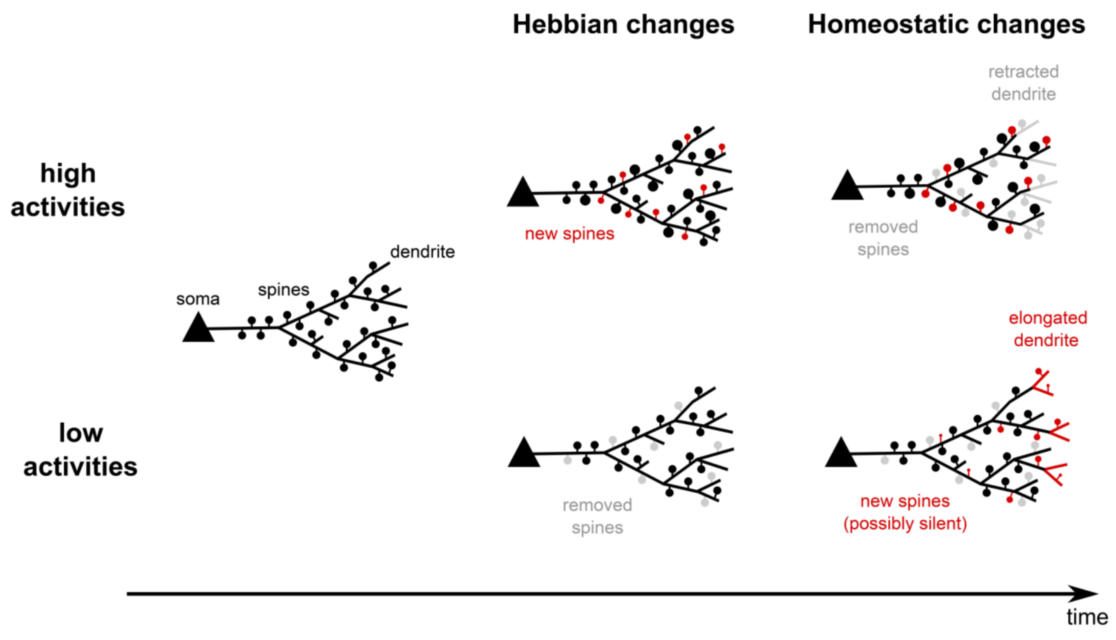


Figure 2.3: Summary of the effects of activity-dependent (i.e. Hebbian) and homeostatic structural plasticity when having high or low neuronal activity. In case of high activity, new dendritic spines are formed following the Hebbian mechanisms and, to maintain homeostasis some connections are removed afterward. An opposite behavior can be seen for low neuronal activity. Figure from [47].

In addition, the number of synapses in the brain can change over time. In [52] it is shown that synaptic density reaches the highest values at 1–2 years age, it drops during adolescence and stabilizes between age 16–72, followed by a slight decline. However, although synaptic density remains approximately stable during adulthood, rewiring of network connections occurs as well. Indeed, the increase of connections in high activity regions (and vice versa), together with the rearrangement of synapses leads to a fine-tuning of the brain's circuits [53], because on one hand some synapses are strengthened through long-term potentiation (LTP) and new connections are formed next to the already potentiated ones to further enhance synaptic transmission, on the other hand when the presynaptic and postsynaptic neuron activities have low correlation their connection are likely to be removed. Indeed, synaptic pruning is considered essential for optimizing activity propagation and memory capacity [54–56]. Additionally, it is commonly believed that synaptic pruning and rewiring dysfunction are among the neural correlates of developmental disorders such as autism or schizophrenia [57, 58], leading to a higher or lower synaptic density with respect to neurotypical subjects, respectively [59–61].

Only in recent times, computational models of structural plasticity and connectivity rearrangements during learning were developed, showing intriguing results. [55] and [56] describe a model of structural plasticity based on "effectual connectivity", defined in these works as the fraction of synapses able to represent a memory stored in a network. By structural plasticity, effectual connectivity is improved, since synapses that do not code for the memory are moved in order to optimize the network's connectivity. Their model defines synapses using a Markov model of three states: potential (i.e. not instantiated), instantiated but silent, or instantiated and consolidated. Structural plasticity is thus related to the passage of the synapses from a potential state to an instantiated state (and vice versa), whereas changes only related to the synaptic weight are described by the consolidation of the instantiated synapses. With such a model, it is possible to show that networks with structural plasticity have higher or comparable memory capacity to networks with dense connectivity and it is possible to explain some cognitive mechanisms such as the spacing effect [55].

[62] simulated a spiking neural network with structural plasticity and STDP, showing that structural plasticity reduces the amount of noise of the network after a learning process, thus making the network able to have a clearer output. Furthermore, such a network with structural plasticity shows higher learning speed than the same network with only STDP implemented.

Some new insights about the importance of synaptic pruning are also shown in [63], in which different pruning rates were studied suggesting that a slowly decreasing rate of pruning over time leads to more efficient network architectures. Chapters 8 and 9 of this thesis will expand this discussion by proposing a new theoretical model for learning through structural plasticity supported by firing-rate-based simulations. The model is designed to consider the effects of both activity-dependent and homeostatic structural plasticity as described in this section and can help shed light on the role of structural modifications of brain circuits during learning.

Part II

GPU-based simulations of spiking networks

Chapter 3

Simulation technology: CPU, GPU-based and beyond

Summary

This chapter introduces some spiking network simulators, with particular regard to NEST GPU, i.e., the GPU version of the NEST simulator. After presenting the currently most used simulators, classified according to the type of hardware that supports them, the dissertation will focus on GPU computing and NEST GPU accompanied by a description of some of its algorithms.

3.1 From CPUs to GPUs and neuromorphic computing: a brief introduction

Spiking neural networks are used to study the behavior of neuron populations with single neuron resolution. They are establishing themselves as a flexible and powerful tool to study the brain function resulting from neuronal dynamics. Moreover, the development of simulation technology is leading to more complex and realistic network architecture being able to simulate the dynamics of large networks of neurons more accurately and more efficiently in terms of performance.
Currently, the simulation of spiking neural networks can greatly benefit from a large variety of hardware: from conventional hardware like CPUs and GPUs to chips specifically designed for this kind of computation, such as FPGA and neuromorphic hardware. From a software perspective, in the last decades, many simulators have been developed to provide the neuroscientific community with powerful yet flexible and intuitive platforms to simplify the simulations of such networks and produce good quality data. Indeed, the development of simulation software able to take advantage of the most recent computing technology is vital for paving the way for more detailed and realistic models. For this reason, many simulation software have recently introduced different back-ends in order to benefit from the largest variety of hardware.
Among the most established spiking network simulators we can mention NEST

[64], NEURON [65], Brian 2 [66] and ANNarchy [67]. These simulators are capable of simulating a large variety of neuron and synapse models and support multi-thread parallel execution on general-purpose CPU-based systems. In addition, NEST and NEURON support distributed computing via MPI.
Neuromorphic hardware is an umbrella term for research oriented towards brain-inspired hardware architectures. In this regard, many projects have been developed; we can mention Loihi [68] and TrueNorth [69], which are entering the realm of large-scale neural network simulations, and BrainScaleS [70], which is based on analog emulations of simplified models of spiking neurons and synapses, with digital connectivity. The system enables energy-efficient neuronal network simulations, offering highly accelerated operations. Another promising project in this field is SpiNNaker [71], which recently achieved biological real-time simulations of a cortical microcircuit model [72] proposed in [12]. This model has since been simulated sub-realtime with NEST [73], on FPGA-based neural supercomputer [74,75] and also with NEST GPU [4], as will be discussed later. SpiNNaker's result was made possible by its architecture designed for efficient spike communication, performed with an optimized transmission system of small data packets. BrainScaleS and SpiNNaker are freely available to the scientific community through the EBRAINS Neuromorphic Computing service.
Nevertheless, neuromorphic systems still require a significant amount of system-specific skills. Even if the simulation speed they can provide is impressive, the flexibility and simplicity of programming environments available for such neuromorphic systems are still low compared to their general-purpose counterparts. On neuromorphic systems adopting analog design techniques, advantages in speed, area, and energy consumption are associated with the difficulties of managing manufacturing fluctuations, unavoidable in analog substrates, and with the effects of electronic noise emerging in the dynamics of analog circuits. Porting neural simulations from digital systems to analog neuromorphic platforms is not a trivial task. Overcoming such difficulties and turning them into advantages is an emerging field of research [76]. Furthermore, as soon as the number of synapses per neuron reaches biological scales (i.e., several thousand per neuron), the current generation of neuromorphic systems can experience a significant slowdown. For example, in its maximum configuration, the first-generation BrainScaleS system hosts 1 billion synapses and 4 million neurons (250 synapses per neuron) on 20 silicon wafers [77], and a similar synapse-per-neuron ratio is a sweet spot for optimal execution on SpiNNaker, well below the typical $10,000$ synapses per neuron characteristic for pyramidal cortical neurons. However, an FPGA-based cluster recently showed promising results in the simulation of the cortical microcircuit model of [12], being around 20 times faster than real-time in a network with natural density, paving the way for a new generation of a neuromorphic system that will have to simulate even larger scale networks [75].

3.1.1 GPU-based simulation

Over the past years, the use of GPUs (Graphical Processing Units) for parallel computing had a significant surge in popularity across various research fields, from scientific simulation to data analysis, deep learning, and machine learning in general. Indeed, this hardware is provided with thousands of cores executing multiple threads simultaneously, resulting in hardware capable of a very high degree of parallelism. Types of computation based on SIMD parallelism (i.e., Single Instruction Multiple Data) can largely benefit from GPU parallelism. For this reason, GPU manufacturers developed specific frameworks such as CUDA or OpenCL to take full advantage of the GPU capabilities. Moreover, GPUs are becoming more and more affordable and can be easily found in the consumer market, even inside high-end laptops. Supercomputers are also taking advantage of the GPU availability since they are reaching for exascale by deriving a significant portion of their computing power from GPUs.
For neuroscience to benefit from these systems, efficient GPU-based back-end for neural network simulators have been developed. For instance, simulation codes such as ANNarchy [67], NENGO [78], Arbor [79], and CARLsim [80] provide a GPU back-end. Moreover, simulators such as Brian and NEURON have simulation codes that take advantage of GPUs, i.e., Brian2CUDA [81] and CoreNEURON [82] respectively.
However, there are simulators such as GeNN [83], or the previously mentioned CARLsim, that are primarily designed for GPUs, with CUDA [84] being the most widely used scripting language for the device code (thus NVIDIA GPUs are employed) and `C/C++` for host code. Among GPU-based simulators, we can say that GeNN is currently one of the most popular. Integrated also in the Brian simulator as Brian2GeNN [85], GeNN was able to outperform CPU-based and neuromorphic code in the simulation of a highly-connected large-scale network [86] and recently managed to simulate a multi-area spiking network model [87–89] on a single GPU card [90]. This result was achieved with the procedural connectivity approach, consisting of generating the model connectivity and its synaptic weights only when spikes need to be transmitted, without storing any connectivity data in the GPU memory. Nonetheless, such an approach can be efficient from a memory point of view, since one of the most constraining features of GPUs is the size of the built-in memory, which in spiking neural network simulations can be a severe limitation. However, this method can be suitable only with static synapses, since plastic synapses require storage of the synaptic weights, which change their value during the simulation.
One interesting topic for spiking network simulation codes, no matter the back-end, is the way a network is instantiated and prepared for the simulation. Indeed, every simulation code employs different solutions, with algorithms that are adapted to the type of simulation hardware the simulator takes advantage of. From now on, we will call this task *network construction*. Generally speaking, we can identify two macro-categories of network construction management:

- network construction during runtime using scripting languages;
- code-generation approaches, which require the code to be first generated

through a high-level description provided by the user before it is compiled and executed.

On one hand, code-generation approaches offer an advantage since, after code-generation and compilation, the time for model construction is generally negligible. On the other hand, a new compilation is needed whenever there is a modification of the model to be applied, with times needed for this task typically much longer than network construction times [91]. Runtime network construction network can offer more ease in model implementation and can be also an advantage in some contexts like parameter scan applications. However, they can be less performing than code-generation approaches [1].
Hitherto we presented a plethora of simulation codes, lastly focusing on GPU-based codes. Now the GPU-based simulation code NEST GPU will be introduced.

3.2 NEST GPU

NEST GPU [2], previously named NeuronGPU [1], is a GPU library for the simulation of large-scale networks of spiking neurons. Written in `CUDA-C++`, it is provided with a Python interface, with most commands being similar, if not identical, to the Python interface of the NEST simulator. Indeed, it has been recently included in the NEST Initiative with the aim of integrating it with the NEST simulator to allow for GPU-based simulations. To open for large-scale simulations of spiking networks, it is able to exploit multiple GPUs on MPI clusters. In this regard, NEST GPU implements a novel MPI-optimized spike delivery algorithm that has been tested on an MPI-GPU cluster in [2] on the simulation of the multi-area model of [87–89]. One of the greatest advantages of the simulations of such a model is the algorithm used for the neuron distribution among the different MPI processes. Indeed, NEST GPU exploits the neuron distribution described in [92], according to which the spatial locality of the neurons is taken into account. To be more specific, this means that neurons belonging to the same population (or cortical circuit in general) are simulated in the same MPI process. This is a potentially great advantage since intra-population connections are generally higher in number than inter-population connections. Thus, this approach leads to a limited load on MPI communication with respect to other algorithms. For example, one of the most utilized approaches for neuron distribution in MPI simulations is the round-robin, for which neurons are evenly distributed across the MPI processes, meaning that in each MPI process there are neurons belonging to every neuron population simulated. This approach grants an optimal load balance across MPI processes but can be less efficient depending on the network architecture since it can lead to a much higher compute load on MPI communication.
Regarding network construction, NEST GPU falls in the category in which the network is built on runtime. In previous versions of NEST GPU (see for example the prototype library NeuronGPU [1]), the network construction process was not handled by the GPU, but the network was built in the CPU. However, the network had to be transferred from CPU to GPU for the simulation of the network dynamics to start. This copy operation from RAM to GPU memory was indeed a bottleneck

for the network construction, which resulted comparable to or even slower than CPU-based simulators such as NEST [1,2]. Even if the network construction time is a fixed overhead with respect to the time-to-solution, a relatively slow network construction approach can limit the efficiency of applications such as parameter scans, where several simulations have to be done, resulting in a significant contribution to the overall execution time. For this reason, we came up with a novel algorithm for dynamic network construction directly in the GPU devices that are able to reach state-of-art performance for network construction [4]. The following subsections give a brief description of the spike delivery and communication algorithm and the novel network construction algorithm.

3.2.1 Spike delivery algorithm

Spike delivery algorithms can be significantly different among different simulation codes, mainly depending on the hardware the simulators use. NEST GPU exploits a specific spike delivery algorithm, which is here briefly introduced. For a more detailed description, please refer to [1,2].
In NEST GPU, each neuron is provided with an output spike buffer, which holds the spikes emitted by the neuron. Output connections of the neuron are organized in groups, with each group collecting connections with the same synaptic delay. Each spike is identified by three values:

- multiplicity, i.e., an integer that indicates how many spikes correspond to the spike event under consideration. Generally, this value is 1, but can be greater in the case, for example, of spiking devices with Poisson statistics;

- time index t_s, which starts from 0 and is increased by 1 at every simulation time step;

- connection-group index i_g, which starts from 0 and is incremented every time the time index matches a connection-group delay.

Given these values for each spike event, the spike is sent when the time index of the spike t_s matches a connection-group delay. This way, the spike is sent in that simulation time step to all the target nodes of the connection group associated with the matching delay. In particular, when a spike has to be transmitted, it is first sent to a spike array. Finally, the spike is sent from this array to the target neurons using a CUDA kernel. Figure 3.1 depicts the workflow of the spike buffer.

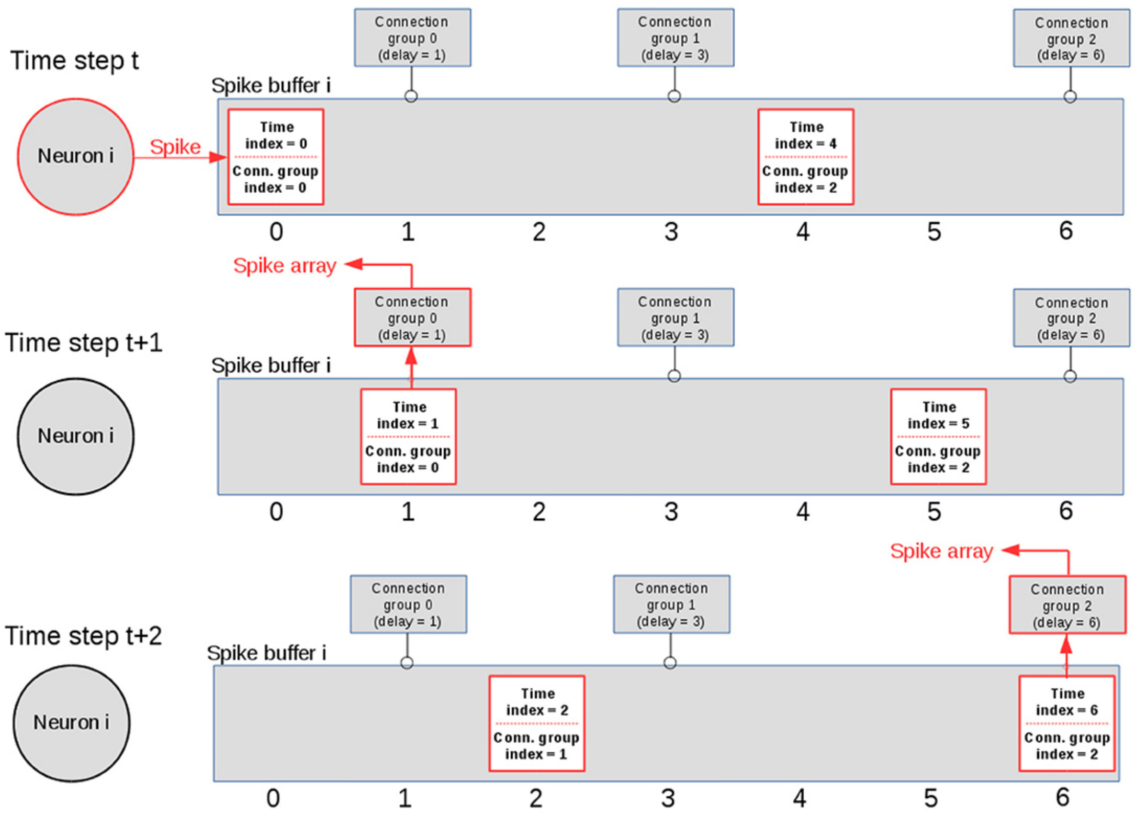


Figure 3.1: Schematic representation of the spike buffer. At time step t a neuron emits a spike, which starts with the time index and the connection group index equal to 0. In the next time step, the spike increases t_s by 1, and, matching the delay of the connection group 0, a spike event is sent to the spike array. This process is iterated until i_g matches the last connection group and the spike event reaches it. Figure adapted from [1].

Regarding multiple MPI process simulations, when a source node is connected to a target neuron belonging to a different MPI process, the spike buffer is created in the remote process, so that when the source node fires a spike, it is sent to the remote spike buffer at the end of the simulation time step through MPI communication.

3.2.2 Network construction algorithm

As mentioned in this section, NEST GPU has recently been equipped with a new algorithm for network construction that dynamically builds the network directly in the GPU [4]. Since the GPU memory capacity is one of the most important bottlenecks in GPU computing, an algorithm that handles such a task should be able to optimize the available memory as much as possible. In the creation of the spiking neural network, a crucial role in this regard is held by the creation of the connections between network nodes. Indeed, in network models with natural connection density, the number of connections greatly exceeds the number of spiking devices: for instance, the multi-area model of [87–89] includes 4 million neurons and 24

billion synapses. We can thus understand the importance of having an efficient method for instantiating and organizing connections during network creation.
NEST GPU structures connections in the following way: each connection contains the source node index, the target node index, the receptor port index, synaptic weight and delay, and the synaptic group index (which is non-zero only for plastic connections). Additional parameters can exist depending on the synapse model employed. Connections are stored in GPU memory using dynamically allocated blocks with a fixed number of connections per block (B), which can be set by the user. It should not be too small to avoid having too many blocks and thus increase the execution time, and it should not be too big to avoid wasting memory leaving the last block not completely filled with connections. A reasonable value, that we usually use for large-scale simulations, can be $B = 10^7$.
Every time a connection command is executed, if the last allocated block does not have enough free slots, one or more new blocks are created to store the remaining connections. After blocks have been allocated, connections are created by launching CUDA kernels (i.e., functions executed on GPU devices exploiting multiple CUDA-thread blocks) to set the connection parameters. In case some parameters have to be driven by a probability distribution, the cuRAND library[1] is used to get the pseudo-random numbers given the distribution parameters. Figure 3.2 depicts an example of connection creation using this algorithm.

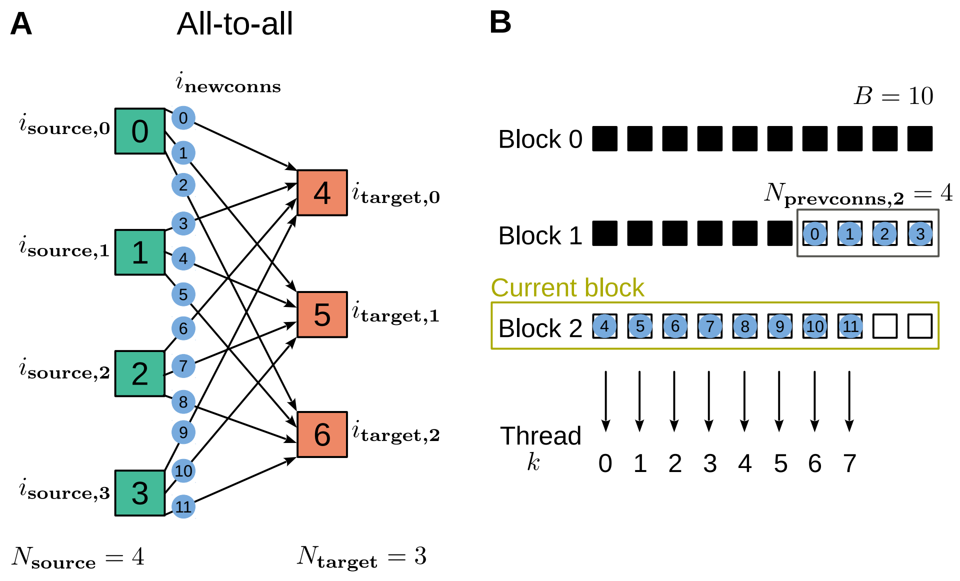


Figure 3.2: Example of connection creation. **(A)** All-to-all connection rule is used, so every source neuron (green) is connected with each target neuron (orange). Here, 12 connections are created and the same amount of slots in the connection blocks have to be allocated. **(B)** Memory slots allocated in the blocks, that have already allocated memory for previous connections (full black squares). To store all the new connections, a new block is created. Finally, an appropriate number of CUDA-threads per block (k) is used to create the connections. Figure from [4].

[1]`https://developer.nvidia.com/curand`

Once connections are created, they must be organized properly to optimize the spike transmission using the spike buffer. For this reason, connections are divided into groups, so that connections from the same group have the same source node and the same synaptic delay. Those groups are, indeed, the connection groups discussed in the previous section and depicted in Figure 3.1. In order to organize connections in this way, connections have to be sorted using two sorting keys hierarchically, i.e., the index of the source node as the first key and the delay as the second key. Such a process is done after the dynamical creation of the connections, in a stage called *calibration* phase. The sorting algorithm employed is an extension of radix-sort [93] applied to an array organized in blocks, based on the implementation available in the CUB library[2].
Once the connections are sorted, their groups must be adequately indexed, so that when a neuron emits a spike, the code has quick access to the groups of connections outgoing from this neuron and to their delays. This indexing is done in parallel using CUDA kernels on connection blocks with one CUDA thread for each connection. Finally, for each source node, an array of size equal to the number of outgoing connection groups is constructed, and the arrays are then concatenated for all the source nodes into a single one-dimensional array. This avoids the creation of a large number of separate arrays, whose retrieval and usage would be more demanding than having a single one. This structure is used during spike delivery to properly index the spike event and send them to the proper connection groups. More details about the network construction algorithm can be found in [4].

[2]`https://nvlabs.github.io/cub`

Chapter 4

Spiking network models and validation techniques

Summary

Here we will introduce the spiking network models we used for benchmarking and validating the NEST GPU simulator. After the description of the models, it will be described the statistical parameters used to evaluate their behavior, and how these parameters can be compared between simulators to ensure that different implementations of the same model provide a statistically compatible result.

4.1 Balanced network

The simplest network model used in this work is the so-called balanced network model. It is a sparsely connected network composed of an excitatory and an inhibitory population of integrate-and-fire neurons [94]. Under suitable conditions, this model can exhibit a stable average firing rate with asynchronous irregular spiking activity. External driving is granted by independent input spike trains targeting all neurons of both excitatory and inhibitory populations. In both NEST and NEST GPU this can be achieved through a single spiking device, called a `poisson_generator`, which is able to send independent spike trains following a Poisson distribution through all its outgoing connections. In the considered model, a single Poisson generator is connected by excitatory connections to all neurons of both populations.
Neurons are connected through static synapses, which can be excitatory or inhibitory. Each neuron receives $\mathcal{C}_\mathrm{E}$ connections from excitatory neurons and $\mathcal{C}_\mathrm{I} = \mathcal{C}_\mathrm{E}/4$ connections from inhibitory neurons using the `fixed_indegree` connection rule, ergo each neuron is provided with $\mathcal{C}_\mathrm{E}$ excitatory and $\mathcal{C}_\mathrm{I}$ inhibitory input connections from other neurons of the model (see [95] for a detailed description of the connection rule). Furthermore, the network is provided of $\mathcal{N}_\mathrm{E}$ excitatory neurons and $\mathcal{N}_\mathrm{I} = \mathcal{N}_\mathrm{E}/4$ inhibitory neurons.

Figure 4.1 shows the structure of the network model following the graphical notation of [95] for nodes and connections.

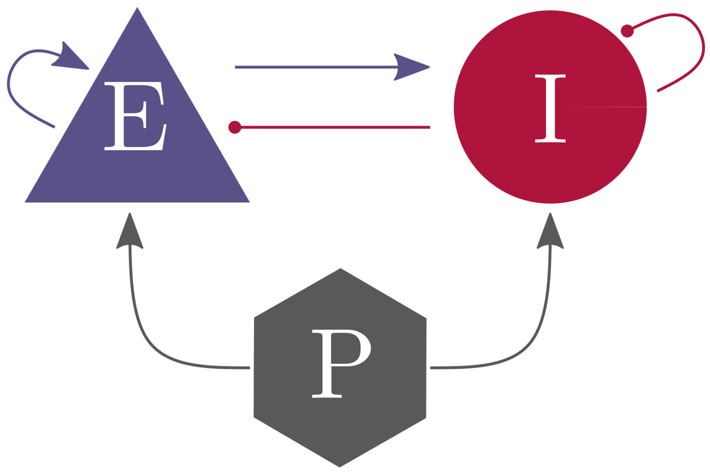


Figure 4.1: Schematic representation of a balanced network model. Here the graphical notation proposed in [95] is used. Thus, the triangle, the circle, and the hexagon represent, respectively, an excitatory, an inhibitory, and a stimulating device population (a Poisson signal generator in this case). Moreover, arrowhead connections are excitatory, and circle-headed connections are inhibitory.

Such a network was also used in [1] in order to test the weak scaling performance of NeuronGPU, the prototype library of NEST GPU. Here we propose a similar scaling using the most recent version of NEST GPU.
The neuron model implemented is the adaptive-exponential integrate-and-fire (AdEx) [14] with conductance-based synapses and synaptic conductance modeled by an alpha function [15]. An introduction to the neuron model can be found in Appendix A, together with neuron and network model parameters.

4.2 Cortical microcircuit

The cortical microcircuit model of [12] describes the neural circuit under $1\,\text{mm}^2$ of cortical surface. It is composed of around $77,000$ neurons and 3×10^8 static connections with connectivity originating from an integrated connectivity map. Moreover, it is characterized by a layered structure that is based on the stratification of the cerebral cortex. Each layer (i.e., 2/3, 4, 5, or 6) has an excitatory (E) and an inhibitory (I) neuron population, for a total of eight populations: 2/3I, 2/3E, 4I, 4E, 5I, 5E, 6I, and 6E. All neurons are externally stimulated by a background input which originates from Poisson-distributed spike input or a DC input. The external input is needed to take into account the signal coming from other regions of the cortex that are not included in the model. Moreover, neurons of layers 4 and 6 are additionally stimulated by a thalamo-cortical input designed as the previously described external input, in order to mimic the input coming from the thalamus. We can consider it as a multilaminar extension of the balanced network model described in Section 4.1.

The neuron model employed is a leaky integrate-and-fire (LIF) with exponentially decaying postsynaptic currents. The differential equations underlying the neuron dynamics are integrated using the exact integration method of [13]. Synapses are static and the weight is chosen in order to have a defined average postsynaptic potential amplitude (see Chapter 1 for more details in this regard).
Such a model is able to show an asynchronous irregular spontaneous activity, compatible with the experimentally observed cell-type and layer-specific firing statistics. As we will see in the next section, it has been used as a building block for larger models [89], and has been used as a benchmark in several studies [73, 74, 86, 96–98].
The model has been implemented in NeuronGPU (and thus in NEST GPU) in [1] and we evaluate the performance of the libraries on this model in [1, 4].
Neuron and network model parameters are described in detail in Tables 1-4 of [99] following the guidelines proposed in [100]. Figure 4.2 shows a scheme of the microcircuit model.

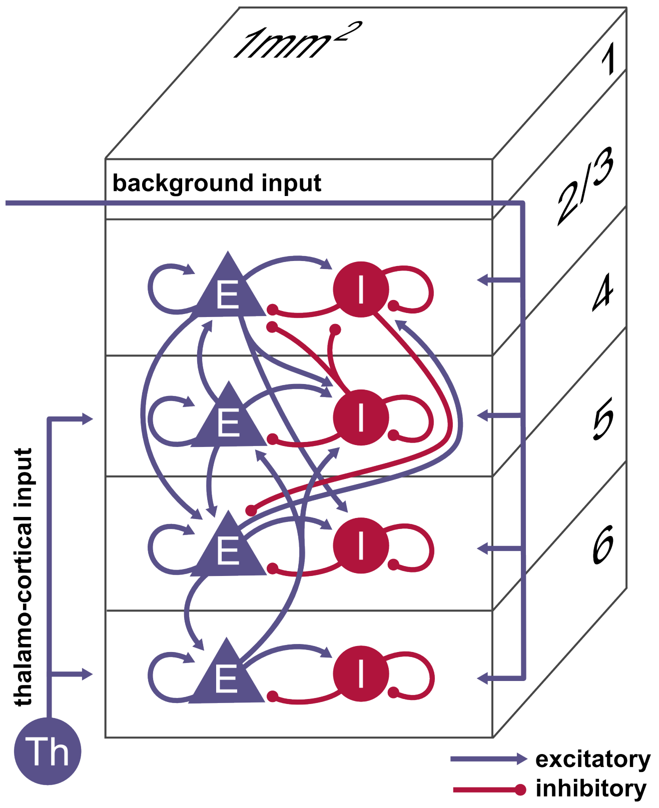


Figure 4.2: Scheme of the cortical microcircuit. The scheme exhibits the layered structure of the network model, together with intra- and inter-layer connections. Here, only connections with probability > 0.04 are shown. Figure from [97].

4.3 Multi-area model

The multi-area model is a spiking network model able to describe the dynamics of the vision-related areas of one hemisphere of the macaque cortex [87–89]. In particular, the model is composed of 32 areas, each one described by a patch of $1\,\mathrm{mm}^2$ size with full local density of neurons and synapses. This way, about 4 million neurons and 24 billion synapses are simulated. Each area is designed similarly to the cortical microcircuit model of [12] described in the previous section. Thus, areas have a laminar structure with 4 layers, each one with an excitatory and an inhibitory population of neurons. Since one area (i.e., area TH) lacks one layer, the total number of neuron populations in the network is 254. The local connectivity (i.e., within the same area) is a result of a scaling of the one of the cortical microcircuit of [12]. Also, the number of neurons for each microcircuit depends on experimental measurements done on every modeled area (e.g., neuron density and thickness of the cortical layer). The inter-area connectivity is based on the database CoCoMac [101] of axonal tracing data, complemented with additional axonal tracing studies [102, 103] and statistical estimations reported in detail in [87].
The neuron model used is, as in the cortical microcircuit, the current-based LIF model with exponentially decaying postsynaptic potential. Moreover, the input coming from non-modeled regions of the brain is mimicked through Poisson spike trains, with frequency adapted for each neural population of the model.
Figure 4.3 represents an overview of the multi-area model.

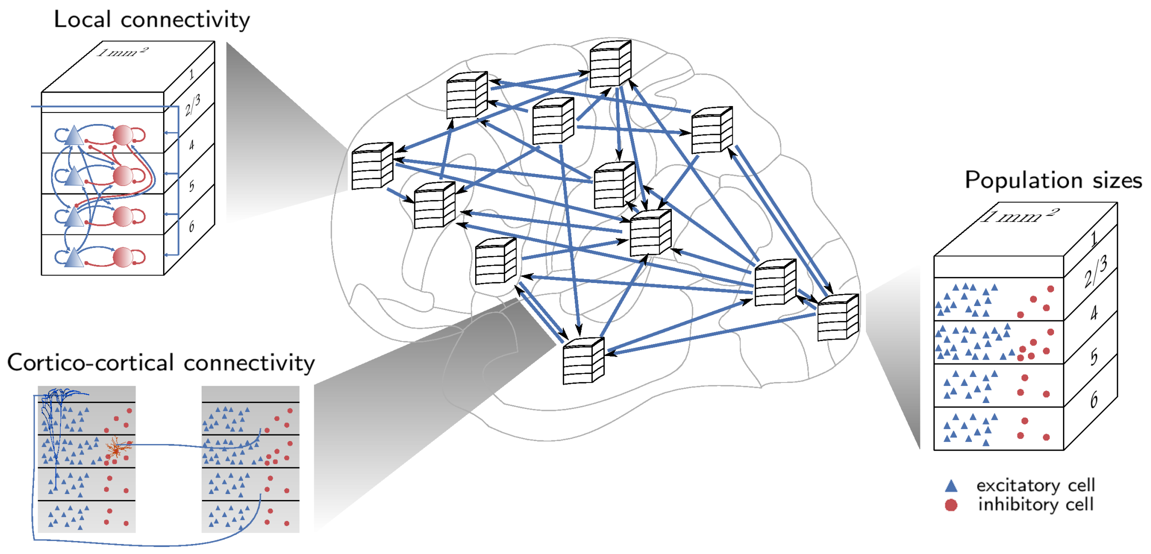


Figure 4.3: Schematic representation of the multi-area model, adapted from [2].

Such a model can show two activity states based on the synaptic strength of the cortico-cortical synapses with respect to the local synaptic weights. When the cortico-cortical synaptic strength is the same as the local one, the network reaches the so-called *ground state*, in which the activity is asynchronous and irregular without any relevant rate fluctuation. Indeed, such a state does not represent the resting-state recordings for spiking activity and functional connectivity.

However, when cortico-cortical connections are increased, the network can reach the *metastable state*, in which the network activity matches the recordings of the resting-state activity of lightly anesthetized monkeys. For further details on the multi-area model, we refer to the original publications [88, 89].

4.4 Validation techniques and benchmarking

The described network models were implemented on NEST and the result of the simulations of these models was compared with experimental measurements (see for example [12, 89]). Thus, NEST is a reliable and well-established simulator we can use as a reference to ensure the correctness of the simulations of the same network models with NEST GPU. When implementing the same model on a different simulator, the first question that we need to answer is: *did we build the right system?* To answer such a question, after preliminary checks (e.g., verify that a single neuron responds in the same way when the same input is given), we have to establish a validation protocol able to assess whether the outcomes of the simulators are statistically compatible. To do so we need some quantitative estimations for each population, and a metric to compare these estimations obtained from the same population of the model under consideration using different simulators.

4.4.1 Getting the distributions

The simulation outcome that has been used to compare these models with experimental observations is the spiking activity. As proposed in several other works (such as [74, 86, 97, 99]) we choose a set of statistical distributions related to the spiking activity of the neuron populations. In order to provide enough details on the statistical distributions we extracted from the data, we need to define what we actually measure from simulations and identify as spiking activity. Indeed, the outcome we get from the simulations is the spike train from each neuron of the population, identified with an ID. A spike train can be described by the equation

$$s_i(t) = \sum_k \delta(t - t_k^{(i)}) \tag{4.1}$$

where $\delta(\cdot)$ is a Dirac delta. Thus, a neuron spike train is just the collection of time values t_j at which the neuron i has emitted a spike. In addition, we need to describe another quantity functional for the estimation of the statistical distributions, which is the inter-spike interval (ISI). Given a spike train, the inter-spike interval (ISI) is defined as the time difference between two consecutive spikes. Thus

$$\mathrm{ISI}_i(t) = t_{k+1}^{(i)} - t_k^{(i)} \tag{4.2}$$

Now that we have defined the spike train and the ISI, for each population we computed the following statistical distributions:

- the average firing rate. For a single neuron it can be defined as

$$\nu_i(t) = \frac{1}{t}\int_0^t s_i(t)dt \tag{4.3}$$

The distribution is then formed by the firing rates of the neurons belonging to the population calculated using the equation above. The average single-neuron firing rate for a population is

$$\nu(t) = \frac{1}{N}\sum_{i=1}^{N} \nu_i(t) \tag{4.4}$$

- the coefficient of variation of the inter-spike intervals (i.e., CV ISI), thus the ratio between standard deviation and average of the inter-spike intervals

  $$\mathrm{CV}_i = \frac{\sum_k (\mathrm{ISI}_k^{(i)} - \langle \mathrm{ISI}_k^{(i)} \rangle)^2}{\sum_k \mathrm{ISI}_k^{(i)}} \tag{4.5}$$

  Like the firing rate, this is a quantity extracted from a single neuron, thus the population distribution is the collection of the CV_i for all the neurons of the population;

- the pairwise Pearson correlation between spike trains, obtained by considering only a subset of 200 neurons for each population to have a reasonable computing time. To properly compute this statistical distribution, spike trains are binned using a time step of 2 ms (i.e., the neurons refractory time used when simulating the network models), so that at most one spike can occur at each bin. We call b_i the binned spike train of the neuron i and μ_i a vector with the same length of b_i having all the entries equal to the average of the binned spike train. Then, Pearson correlation can be computed as follows

  $$\mathcal{C}[i,j] = \frac{\langle b_i - \mu_i, b_j - \mu_j \rangle}{\sqrt{\langle b_i - \mu_i, b_i - \mu_i \rangle \cdot \langle b_j - \mu_j, b_j - \mu_j \rangle}} \tag{4.6}$$

  where $\langle , \rangle$ represents a scalar product. Considering 200 neurons per population, the result of Equation (4.6) is a 200×200 matrix, and the distribution of the pairwise Pearson correlation is formed by the non-diagonal entries of the matrix.

The quantities we have just described are computed using the Elephant package [104]. To form the raw distributions, it would be sufficient to apply proper binning to the data and create the histograms. In this regard, distributions are binned following the Freedman-Diaconis rule, according to which the bin width is defined as

$$2\frac{\mathrm{IQR}}{N^{1/3}} \tag{4.7}$$

where IQR is the interquartile range of the data and N is the number of observations for each distribution. This way, we can obtain the raw distributions, which are indeed dependent on the random fluctuations of single simulations. In order to reduce such fluctuations, we applied a smoothing using the kernel density estimation (KDE) [105, 106] with a Gaussian kernel. Having a set of values x_i with $i = 1, \ldots, N$ and a bandwidth h, a Gaussian kernel $K(x,h)$ can be defined as

$$K(x,h) = A \exp\left(-\frac{x^2}{2h^2}\right) \tag{4.8}$$

where A is a constant needed for normalization. The KDE for a point y is then

$$\rho_K(y) = \sum_{i=1}^{N} K(y - x_i, h) \tag{4.9}$$

Hence, if x_i are the set of values extracted from the spike data, each value can now be considered as a Gaussian function with a fixed width and properly normalized. The sum of the Gaussian functions obtained for each value produces the smoothed distribution.
To get the distributions, we still have to define the values for the normalization factor A and the bandwidth h. Regarding the normalization, each Gaussian function returned by Equation (4.9) is normalized to $1/N$ so that the final distribution is normalized to unity. Moreover, the bandwidth is optimized through the Silverman method [107], so that

$$h = 0.9 \min\left(\hat{\sigma}, \frac{\text{IQR}}{1.349}\right) N^{-1/5} \tag{4.10}$$

with $\hat{\sigma}$ standard deviation of the values and IQR the interquartile range. In the framework developed for this thesis, KDE is obtained using the function `sklearn.neighbors.KernelDensity` of the `scikit-learn` Python library [108]. For more clarity, Figure 4.4 depicts a simple application of KDE.

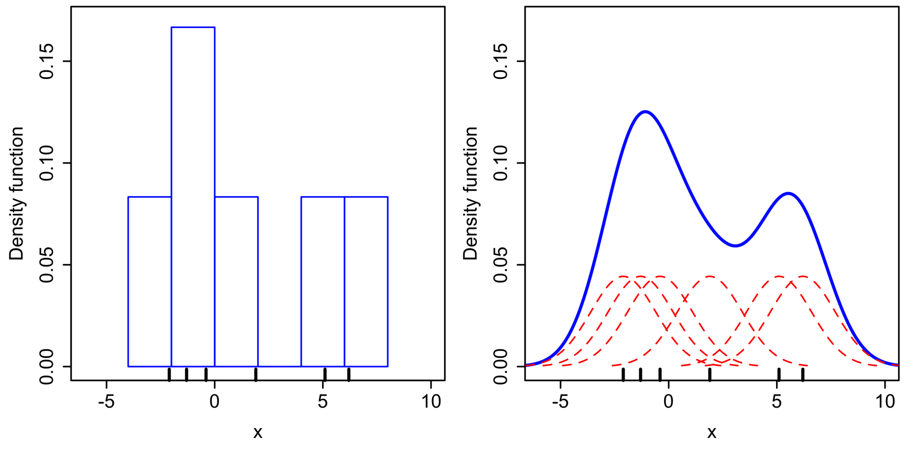


Figure 4.4: Example of application of KDE to a simple data set. The left panel shows the histogram obtained from the data shown below as vertical black dashes. On the right, KDE is applied to data. Red-dotted lines represent the Gaussian kernels from each value that belongs to the data set, whereas the blue line is the KDE, defined as the sum of the single Gaussian kernels. Figure from [109].

This way we are able to obtain a continuous function able to reduce the small variability observed by single simulations that used different seeds for random number generation. Indeed, the seed plays a crucial role in our validation framework, and the next subsection describes its role in detail.

4.4.2 Getting the metric to compare the distributions

The validation protocol adopted to statistically compare the results of the spiking activity obtained from different simulation codes (i.e., NEST and NEST GPU) is described below:

1. we perform a set of simulations (e.g., 10) using NEST, and we do the same with NEST GPU. Each simulation should use a different seed for random number generation;

2. we perform an additional set of simulations using NEST (i.e., the simulator taken as a reference) using a different set of seeds, in order to estimate the variability of NEST when using different seeds;

3. we compute, using a proper metric, the pairwise comparison between the distributions obtained by the two different simulators (e.g., the first simulation of NEST compares the distributions with the first simulation of NEST GPU, etc). This way we collect a set of values that quantifies the difference between the two simulators for each distribution, which we can call *NEST-NEST GPU*;

4. we can do the same but comparing the two sets of simulations using NEST, obtaining the *NEST-NEST* dissimilarity measure (or distance);

5. we plot side by side the *NEST-NEST GPU* and *NEST-NEST* values to evaluate the differences that arise because of the different simulator employed and the one that can arise because of a different seed for random number generation when using the same simulator.

With this approach, we have a more quantitative method for the validation of the NEST GPU simulator than a simple comparison of the distributions that can be done by simply showing the distributions in the same plot.
Now a question remains to be answered: *which metric, or dissimilarity measure, should we choose?* For example, in [1] for the comparison of NEST and NeuronGPU in the simulation of the cortical microcircuit model, we used the Kullback-Leibler divergence (D_{KL}) [110], which is defined as

$$D_{KL}(p, q) = -\sum_i p_i \log\left(\frac{p_i}{q_i}\right) \tag{4.11}$$

where p and q are two probability distributions with the index i running on the sampling points of the distributions. The divergence measures the degree of dissimilarity of p with respect to a reference distribution q. However, in [2] we moved to the Earth Mover's Distance (EMD) because of the metric properties that make it symmetric and more specific in detecting the degrees of dissimilarity among the distributions. EMD computes the distance between two probability distributions and its name resembles a problem of soil reshaping. Indeed, the two distributions can be interpreted as two piles of soil, with one of them having to be arranged in order to have the same shape as the other one. In this case, EMD represents the

minimum amount of work needed to move the soil from one pile in order to have the desired distribution of soil. Returning to our distributions, EMD measures the work that has to be done to reshape a distribution in order to be like the target one. From a mathematical point of view, EMD is equivalent to the 1^{st} Wasserstein distance between two distributions.
The Wasserstein distance is defined as follows [111, 112]: let χ be a metric space endowed with a metric d and μ and ν be two probability measures on χ. Also let $\Gamma(\mu,\nu)$ be the set of probability measures γ on $\chi \times \chi$ so that the measure γ has marginals μ and ν on each axis. The p^{th}-Wasserstein distance can be written as

$$W_p(\mu,\nu) = \Big(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_{\chi\times\chi} d(x,y)^p d\gamma(x,y)\Big)^{1/p} \tag{4.12}$$

This relation yields the given interpretation of the Earth Mover's Distance. In particular, given a $\gamma \in \Gamma(\mu,\nu)$ and two locations (x,y), the quantity $d(x,y)^p$ quantifies the amount of work necessary to move a unit of mass from x to y, and the infimum of the integral thus returns the minimum amount of work needed to reshape μ distribution into ν.
Furthermore, Equation (4.12) can be rewritten when $\chi = \mathbb{R}^1$ (ergo $d(x,y) = |x-y|$). Letting F_X be the cumulative distribution function (CDF) of a distribution X and $F_X^{-1}(q) = \inf\{x : F_X(x) \geq q\}, q \in (0,1)$ the quantile distribution of X (i.e. the inverse cumulative distribution), it is shown in [113] that

$$W_p(\mu,\nu) = \Big(\int_0^1 |F_\mu^{-1}(\alpha) - F_\nu^{-1}(\alpha)|^p d\alpha\Big)^{1/p}. \tag{4.13}$$

In the particular case of the 1^{st} Wasserstein distance (i.e. $p = 1$), as discussed in [114], Equation (4.13) reduces to

$$W_1(\mu,\nu) = \mathrm{EMD}(\mu,\nu) = \int_{\mathbb{R}} |F_\mu(t) - F_\nu(t)| dt. \tag{4.14}$$

that is the equation used by the Python scientific library SciPy [115] to compute this metric with the `scipy.stats.wasserstein_distance` function, which takes the values of the distributions as input, computes their CDF and finally returns the result of the integral shown in Equation (4.14).
In conclusion, the validation process enables us to show side by side the distributions of the EMD values obtained from the comparisons *NEST-NEST GPU* and *NEST-NEST*. The next chapter describes the validation work we performed using NEST GPU for the simulation of the cortical microcircuit model and the multi-area model. Then, we analyze the performance of NEST GPU in the simulation of these models and the scaling performance of the simulator through simulations of the balanced network model.

Chapter 5

NEST GPU validation

Summary

Here NEST and NEST GPU are compared in terms of neural activity of single neurons and the models discussed in the previous chapter with the aim of providing a solid validation workflow to be easily employed when validating NEST GPU. In particular, NEST GPU is validated on the simulation of the cortical microcircuit model and the multi-area model previously introduced.

5.1 Single neuron simulation

A first control that should be done when checking the compatibility of NEST GPU is a simple simulation of single neuron models targeted by a few spikes or injected with a constant current, to be compared with an analogous NEST simulation. Indeed, one of the tests NEST GPU performs after installation regards the response of such a stimulus by the membrane potential of the several neuron models implemented in the simulator. For instance, Figure 5.1 shows the membrane potential of two different neuron models injected with two spikes delivered using connections with different delays. Panel A depicts the response of a LIF neuron with exponential current-based synapses (`iaf_psc_exp` in NEST and NEST GPU), whereas panel B shows the response of an AdEx neuron with conductance-based synapses and synaptic conductance modeled as an alpha function (`aeif_cond_alpha` in NEST and NEST GPU).
The same simulation is also performed in NEST, and the results are here compared. In particular, the first signal the neuron receives is excitatory (i.e., is transmitted through a connection with positive weight), whereas the following one elicits a drop in the membrane potential (ergo the synapse is inhibitory, and thus with a negative synaptic weight). When testing NEST GPU consistency in such simulations, we take the NEST values as a reference and we compute the normalized root-mean-square error (NRMSE) between the membrane potential obtained by the two simulators, normalized by the average of the membrane potential ob-

tained by the NEST GPU simulation (in modulus), as indicated in Equation (5.1):

$$\text{NRMSE} = \frac{\text{RMSE}}{|\bar{V}_\text{m}^\text{NEST GPU}|} = \frac{1}{|\bar{V}_\text{m}^\text{NEST GPU}|}\sqrt{\frac{\sum_{i=1}^{N}(V_\text{m}^\text{NEST GPU} - V_\text{m}^\text{NEST})^2}{N}} \tag{5.1}$$

Indeed, the NRMSE calculated from the data shown in panels A and B of Figure 5.1 are in the range of 10^{-6}.

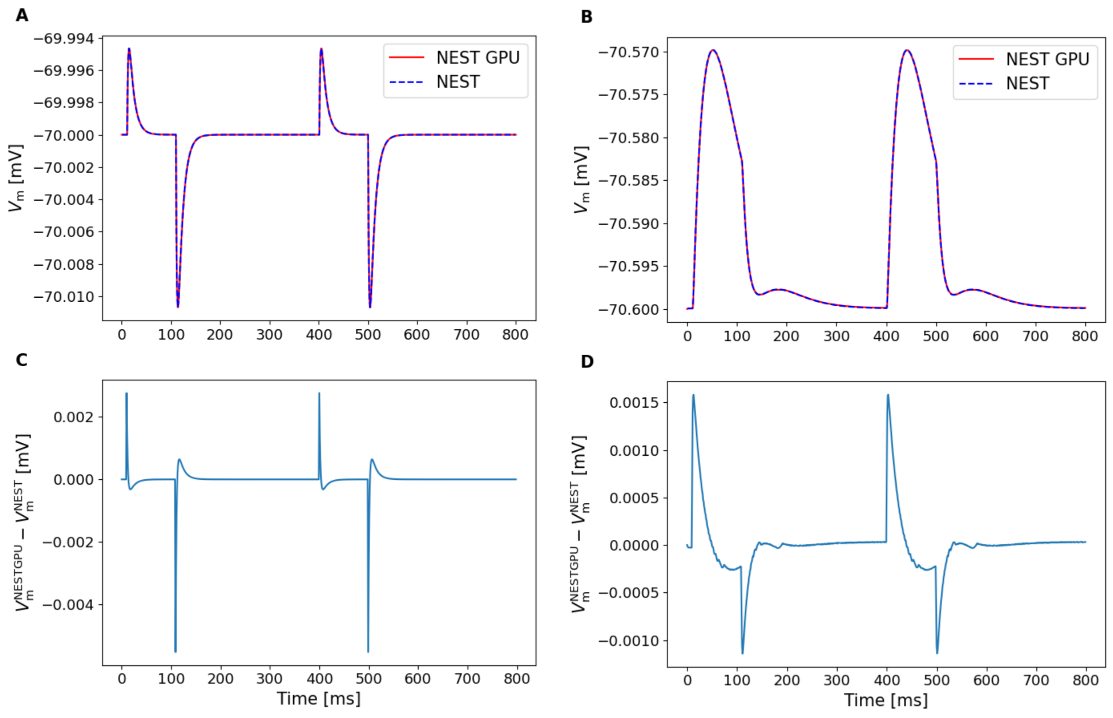


Figure 5.1: Membrane potentials of neurons injected with single spikes simulated using NEST (blue dotted line) and NEST GPU (red line). **(A)** LIF neuron with exponential current-based synapses. NRMSE $= 5.3 \times 10^{-6}$. **(B)** AdEx neuron with conductance-based synapses and synaptic conductance modeled as an alpha function. NRMSE $= 4.5 \times 10^{-6}$. **(C)** Difference between the LIF neuron membrane potential simulated using NEST GPU and NEST. **(D)** Same as **(C)**, but for the AdEx neuron membrane potential.

Clearly, such a test is not enough to ensure the compatibility of NEST and NEST GPU. The following sections will describe the validation processes on the simulation of two large-scale network models, i.e., a cortical microcircuit model and a multi-area model.

5.2 Simulations of the cortical microcircuit model

The first large-scale model implemented in NEST GPU was the cortical microcircuit model of [12] described in Section 4.2. It employs LIF neurons with exponential current-based synapses, which, as indicated in Figure 5.1 do not show any relevant difference in the dynamic with respect to the NEST implementation. Indeed, this model is also integrated using the same integration method, which is the exact integration scheme proposed in [13].
In order to validate the result of the NEST GPU simulations using NEST as a reference, we perform simulations using the Poisson spike generator as input for each area of the model, and we initialize the membrane potential following a normal distribution in order to avoid long transients. In addition, we perform a so-called pre-simulation phase of 500 ms without recording the spiking activity for the same reason. The simulation of the network dynamics (with enabled spike recording) is 600 s long, using a simulation time step of 0.1 ms. Indeed, such a long simulation of biological time to be simulated can be preferred, since it helps the activity statistics to converge, being able then to distinguish between the statistic of the spiking activity and random processes [99]. Moreover, the full-scale simulation of such a model can also be performed using GPUs mounted in high-end laptops, since it requires a bit less than 4950 MiB of GPU memory and returns output files of around 4 MB for each second of simulation of the network dynamics.
Following the validation protocol discussed in Section 4.4.2, we compute the distributions of firing rate, CV ISI and Pearson correlation for all the eight populations of the model for NEST GPU and NEST and, to produce the distributions, we used the `seaborn.violinplot` function of the Seaborn Python library [116], which returns KDE-smoothed optimized using the Silverman method. Indeed, the first validation between NEST GPU and NEST, done in [1], was not done using this package, but by extracting and smoothing the distributions autonomously through our analysis script, which performed the same procedure by computing the KDE method using the function `sklearn.neighbors.KernelDensity` available in the scikit-learn Python library [108]. By using the Seaborn Python library, we verified that the violin plot function uses the same protocol for binning and smoothing as in the previous analysis pipeline described in Section 4.4.1. Moreover, the violin plot has the advantage of showing side-by-side the distributions obtained using the two simulators, granting a better comparison of the results. Figure 5.2 shows the violin plot for a simulation of the cortical microcircuit model on NEST and NEST GPU.
As can be noticed, the distributions obtained with NEST and NEST GPU are visually indistinguishable. However, to provide a quantitative estimation of the difference between the distribution, and compare the fluctuations between the two simulators and NEST with different seeds for random number generation, we computed the EMD. In this regard, we created three sets of 100 simulations each using different seeds for random number generation, so that the comparison of NEST-NEST and NEST-NEST GPU can be done. Figure 5.3 shows how values of EMD for such a comparison fluctuate for each population of the model and for each distribution obtained.

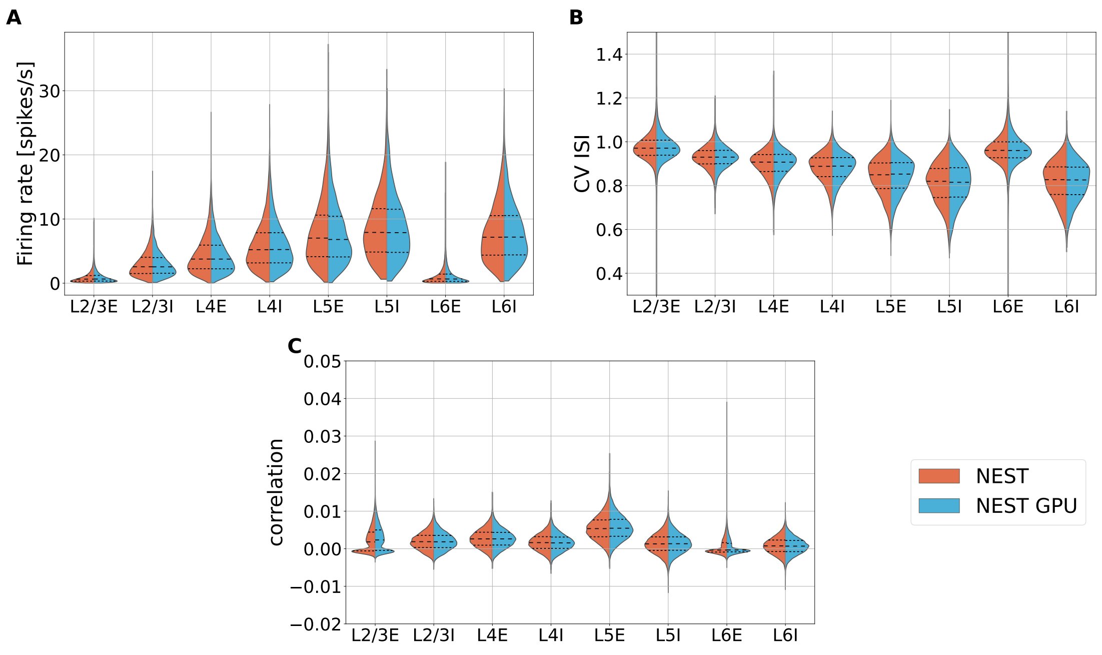


Figure 5.2: Violin plot of the distributions of firing rate **(A)**, CV ISI **(B)** and Pearson correlation **(C)** extracted during a simulation of the cortical microcircuit model performed on NEST (orange distributions) and NEST GPU (sky blue distributions). Distributions are obtained for each population of the model, whose label is indicated in the abscissa. The central dashed line represents the median of the distribution, whereas the two side dashed lines represent the interquartile range.

Figure 5.3, as previously discussed, compares the fluctuations we can expect by simulating the same model using two different simulators such as NEST and NEST GPU (i.e., NEST-NEST GPU comparison) and the ones we can have by comparing NEST simulations with different seeds for random number generation (i.e., NEST-NEST comparison). Thus, if the NEST-NEST GPU comparison fluctuates similarly with respect to the values of the NEST-NEST comparison, we can conclude that the usage of a different simulator does not have a significant variability when simulating the model, thus validating the simulator. In this case, we can notice that the fluctuations are compatible, and also the values of EMD are relatively low when compared to the values of the respective distributions in Figure 5.2. Thus, we can affirm that NEST GPU is able to reproduce results statistically identical with respect to the ones of NEST.
The validation here reported compares the version 3.3 of NEST [117] and the version of NEST GPU able to construct the network directly in the GPU memory, presented in [4] and available on the GitHub page of the library[1] under the git tag `nest-gpu_onboard`[2].

[1]`https://github.com/nest/nest-gpu`
[2]`https://github.com/nest/nest-gpu/releases/tag/nest-gpu_onboard`

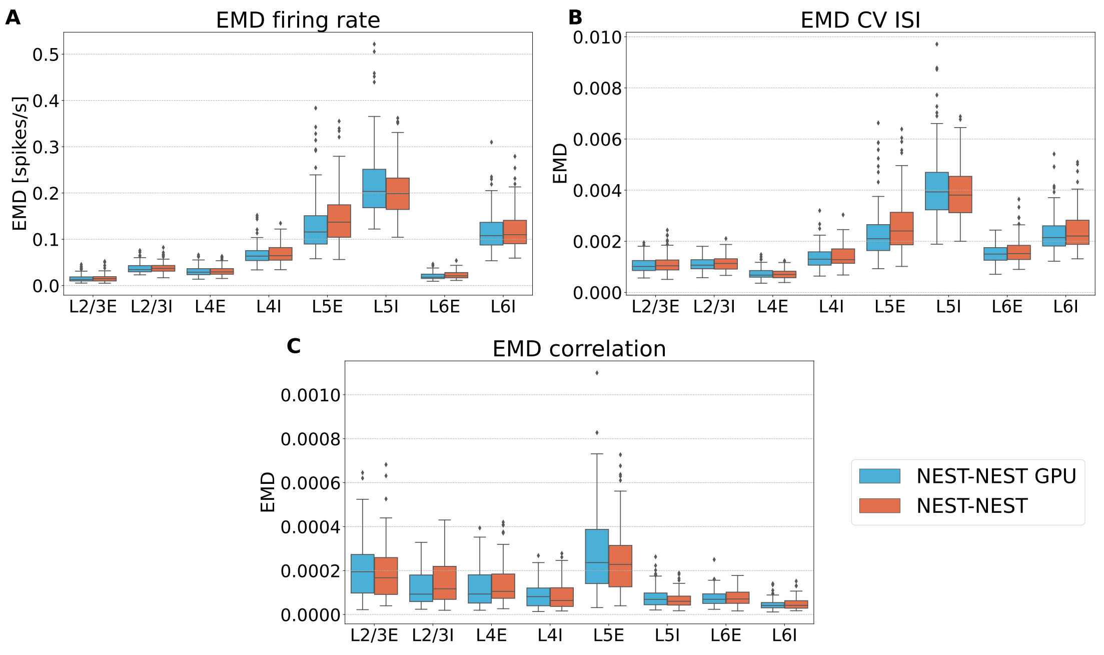


Figure 5.3: Box plots of the EMD obtained by comparing distributions of firing rate **(A)**, CV ISI **(B)** and Pearson correlation **(C)** for NEST and NEST GPU simulations (sky blue bars) or two sets of NEST simulations with different seeds (orange bars). The central line of the box plot represents the median of the distribution, whereas the extension of the boxes is determined by the interquartile range of the distribution formed by the values of EMD of each comparison. Whiskers extend to the farthest observations that lie within 1.5 times the interquartile range, and dots represent the outliers.

Indeed, this validation workflow was employed for different versions of NEST and NEST GPU. For instance, in [1] was compared NEST version 2.20.0 [118] and the prototype library NeuronGPU[3]. Moreover, during the development of the library, such a workflow is applied whenever a code modification results in a change of the spiking activity results, to ensure that the compatibility from a statistical point of view is always preserved.

5.3 Simulations of the multi-area model

In order to validate NEST GPU and the MPI implementation of its spike delivery algorithm, in [2] we implemented and simulated the multi-area model of [87–89]. As described in Section 4.3, the model simulates 32 areas of the macaque visual cortex, each one represented by a patch of $1\,\text{mm}^2$ surface designed as the cortical microcircuit model of [12]. Thus, every area (except for the TH, which lacks layer 4), has a total of 4 layers (i.e., 2/3, 4, 5, 6), each one having an excitatory (E) and

[3]`https://github.com/golosio/NeuronGPU`

an inhibitory (I) population.
In order to validate NEST GPU in the simulation of this model, we performed the same protocol described in Chapter 4 and applied in the previous section in the case of the cortical microcircuit model. However, some simulation parameters differ with respect to the validation workflow of the cortical microcircuit. For each set of simulations, we performed 10 simulations using different seeds for random number generation. The pre-simulation time is set to $500\,\mathrm{ms}$ as well, however, we simulated $10\,\mathrm{s}$ of biological activity, using a simulation time step of $0.1\,\mathrm{ms}$. The reason for the lower number of simulations and the shorter time of the simulation of the network dynamics is mainly linked to technical bottlenecks, mostly related to the limited compute time available to perform the simulations and the amount of data to be analyzed to produce the validation. Indeed, such a large-scale network, with about 4 million neurons and 24 billion synapses, requires an MPI-GPU cluster to be simulated, and in the case of these simulations, we opted for a simulation using 32 compute nodes, each equipped with a GPU device. Details on the computing platform used to perform the simulations will be provided in the next chapter.
As mentioned in Section 4.3, the multi-area model can exhibit two activity states: a stationary ground state with relatively low firing rate variability and a metastable state, more realistic and with an increase in the rate fluctuation and inter-area interactions. As a reference, Figure 5.4 shows the raster plot of two of the 32 areas of the model to show the difference in the spiking activity between these states.

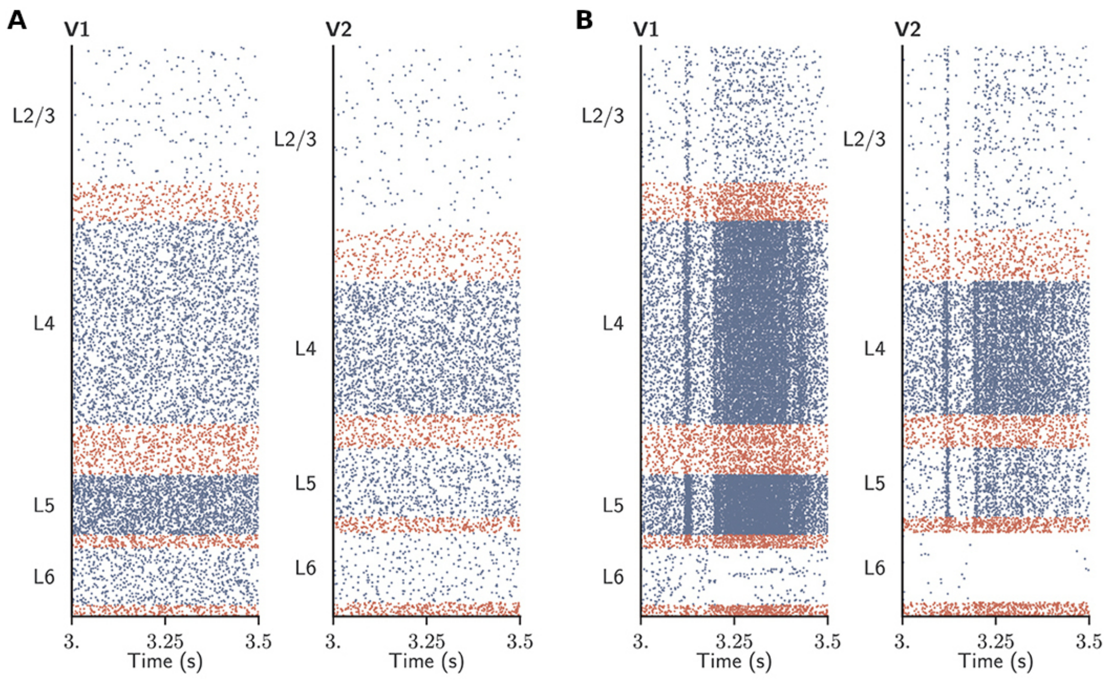


Figure 5.4: Raster plots for the areas V1 and V2 of the multi-area model when the network is in the ground state **(A)** and in the metastable state **(B)** simulated using NEST. Figure adapted from [2].

As can be seen, raster plots of the ground state of the model show a behavior without significant changes in the network activity, whereas the metastable state is quite the opposite. It should be noted that the only difference between the two states of the model is the strength of the cortico-cortical synapses: when the strength is the same with respect to the intra-area synapses the model is in its ground state, whereas the metastable state has the strength of the cortico-cortical connections increased.
For the purpose of this study, we validated both the ground state and the metastable state of the model. We computed the distributions of firing rate, CV ISI, and Pearson correlation for each population of the model, so that for each state we extracted a total of 762 distributions (254 distributions per statistical measure). Distributions were smoothed through the KDE with a Gaussian kernel using a bandwidth driven by the Silverman method. As explained in the previous section, such a task can be done by the Seaborn `violinplot` function, which is useful to compare at first glance distributions obtained, in this case, using NEST and NEST GPU.
Figure 5.5 represents the violin plots of firing rate, CV ISI, and Pearson correlation obtained from a simulation of the multi-area model using NEST and NEST GPU for the first area of the model (i.e., the V1) in the ground and metastable states. This is depicted as a sample, however, the distributions for all the populations and areas of the model are shown in Appendix B.
The distributions of NEST and NEST GPU are barely indistinguishable when simulating the ground state of the multi-area model.
In contrast, in the distributions of metastable state of the model, we can notice, apart from higher rates and variability indicated by the values of the statistical distributions, a larger difference between NEST and NEST GPU distributions, so that the distributions are not indistinguishable as in the case of the ground state. However, we have to consider that, in such a state, even a change of the seed for random number generation can lead to a large variability with respect to what we observe in the ground state. To make this clear, Figure 5.6 and 5.7 show the distributions of firing rate, CV ISI, and Pearson correlation just for two populations of the area V1 of the model in both the ground state and the metastable state. NEST and NEST GPU distributions are shown not using the violin plot function but using our analysis workflow that produces smoothed distributions without employing the Seaborn `violinplot` function. Moreover, to underline the variability due to the change of seed, here we show distributions averaged over the 10 seeds of random number generation employed, together with a shading representing the standard deviation of the mean.

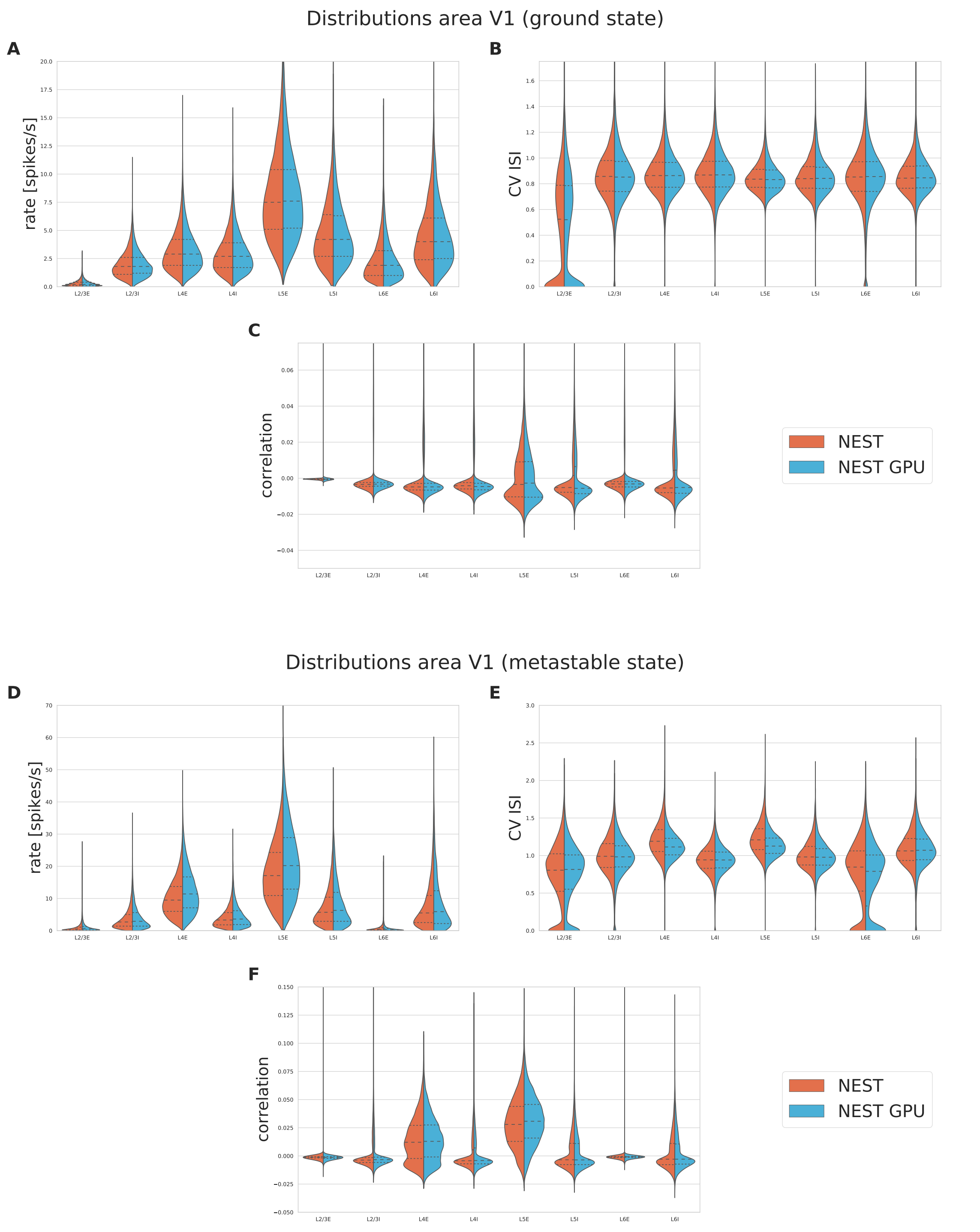


Figure 5.5: Violin plot of the distributions of firing rate, CV ISI, and Pearson correlation extracted during a simulation of the multi-area model of the ground state (**A, B, C**) and the metastable state (**D, E, F**) for the area V1. Simulations are performed on NEST (orange distributions) and NEST GPU (sky blue distributions). The Central dashed line represents the median of the distributions, and the other two dashed lines represent the interquartile range.

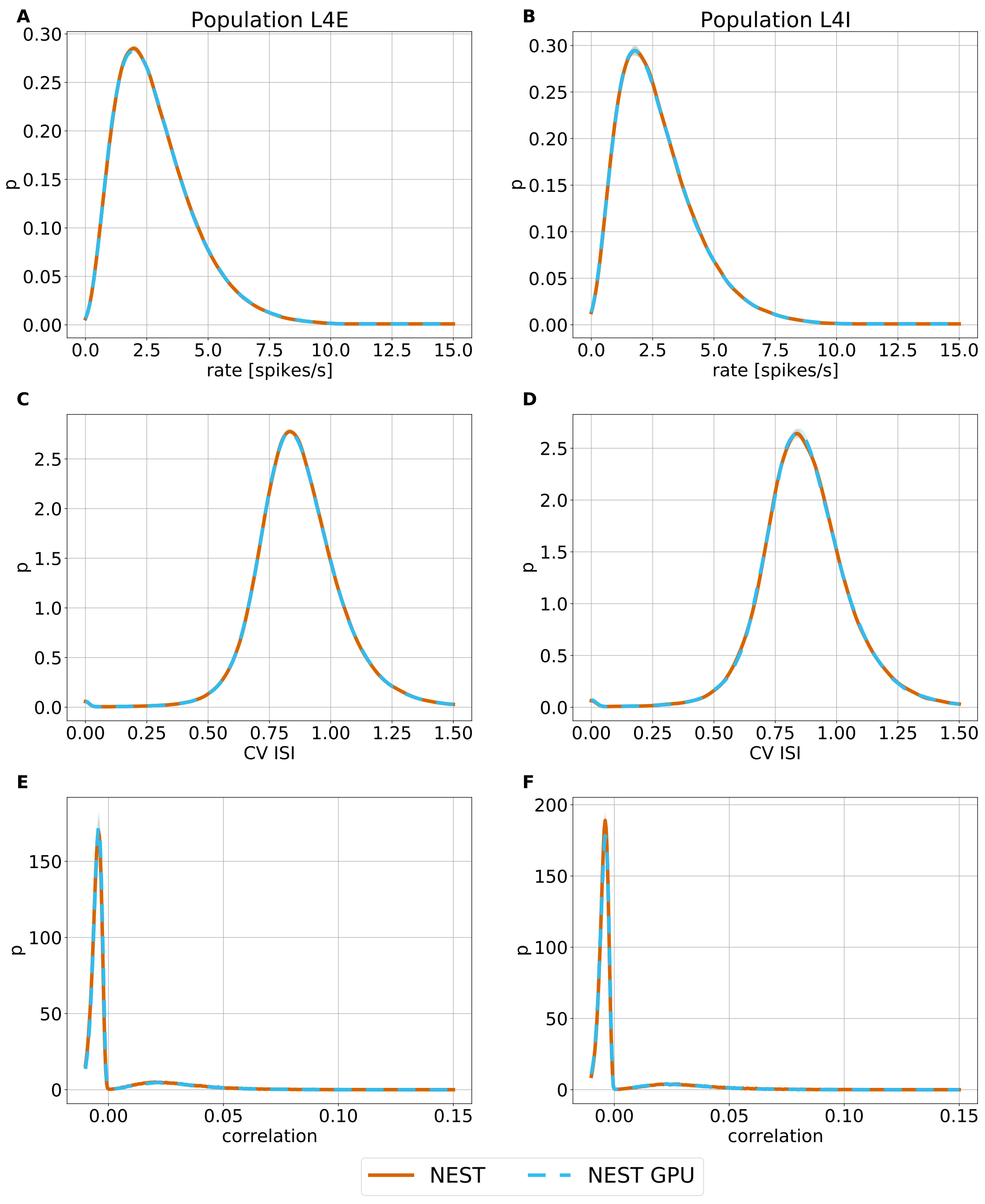


Figure 5.6: Ground state distributions of firing rate **(A,B)**, CV ISI **(C,D)** and Pearson correlations **(E,F)** for the populations L4E and L4I of the area V1 of the multi-area model. Distributions are averaged over 10 simulations with NEST (orange line) and NEST GPU (sky blue dotted line). Sky blue and orange shading represents the standard deviation of the distributions. Figure adapted from [2].

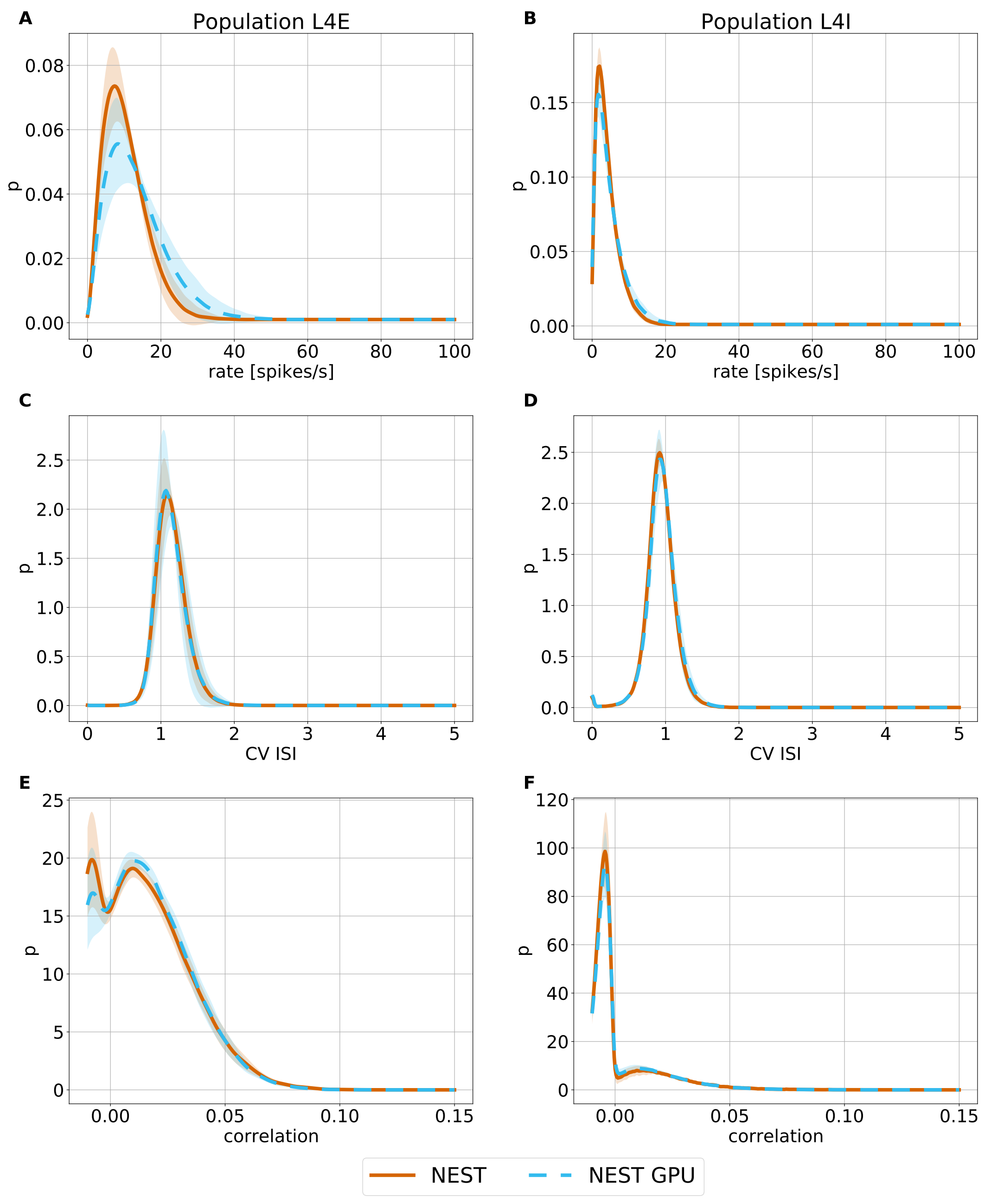


Figure 5.7: Metastable state distributions of firing rate **(A,B)**, CV ISI **(C,D)** and Pearson correlations **(E,F)** for the populations L4E and L4I of the area V1 of the multi-area model. Distributions are averaged over 10 simulations with NEST (orange line) and NEST GPU (sky blue dotted line). Sky blue and orange shading represents the standard deviation of the distributions. Figure adapted from [2].

As can be seen, metastable state simulations have a higher variability even when the seed is changed using the same simulator. In this regard, the usage of EMD to compare the fluctuations when using the simulators can be particularly useful, since it is not trivial to ensure the compatibility between NEST and NEST GPU

when having distributions qualitatively different. Thus, having 2 sets of NEST simulations and a set of NEST GPU simulations using 10 different seeds for random number generation, we perform 10 pairwise comparisons between NEST and NEST GPU and 10 comparisons between the two sets of NEST simulations. The EMD values related to these comparisons are then collected in box plots and plotted side by side for a more effective comparison of the fluctuations driven by the usage of a different seed and the ones driven by the usage of a different simulator. Figure 5.8 shows the EMD box plots for the comparison of, respectively, firing rate, CV ISI, and Pearson correlation distributions for each population of area V1 of the multi-area model in the case of ground state and metastable state simulations.
Figure 5.8 shows that the EMD values for the NEST-NEST GPU comparison are distributed similarly to those for the NEST-NEST comparison, meaning that the differences that arise due to the choice of the simulator are statistically similar to those between NEST simulations with different random number generator seeds. Thus, using NEST GPU instead of NEST (with different random seeds) does not add variability compared to using different random seeds with the same simulator. As a reference, only the results for the area V1 of the model are shown in this figure, however, the EMD box plots for all the populations and areas of the model are shown in Appendix B. This is a further indication that NEST and NEST GPU yield statistically closely similar results. EMD values obtained by the comparison of the ground state distributions are significantly smaller than the EMD values obtained for the metastable state. This is due to the increased fluctuations in the latter state of the model. In some cases, whiskers for the NEST-NEST and NEST-NEST GPU comparisons have different extents. This may be related to long-tailed distributions of the corresponding activity statistics. Indeed, differences in the tails of the distributions caused by only a few data points can lead to large differences in EMD values because the probability mass needs to be moved over large distances to turn one distribution into another. Moreover, each EMD box is only formed by 10 values of EMD, since each set of simulations comprises 10 simulations. Indeed, a larger amount of data would decrease the differences between the box plots shown, however, such a work would need a considerable amount of computed time which can be reduced by future hardware and developments of the library.

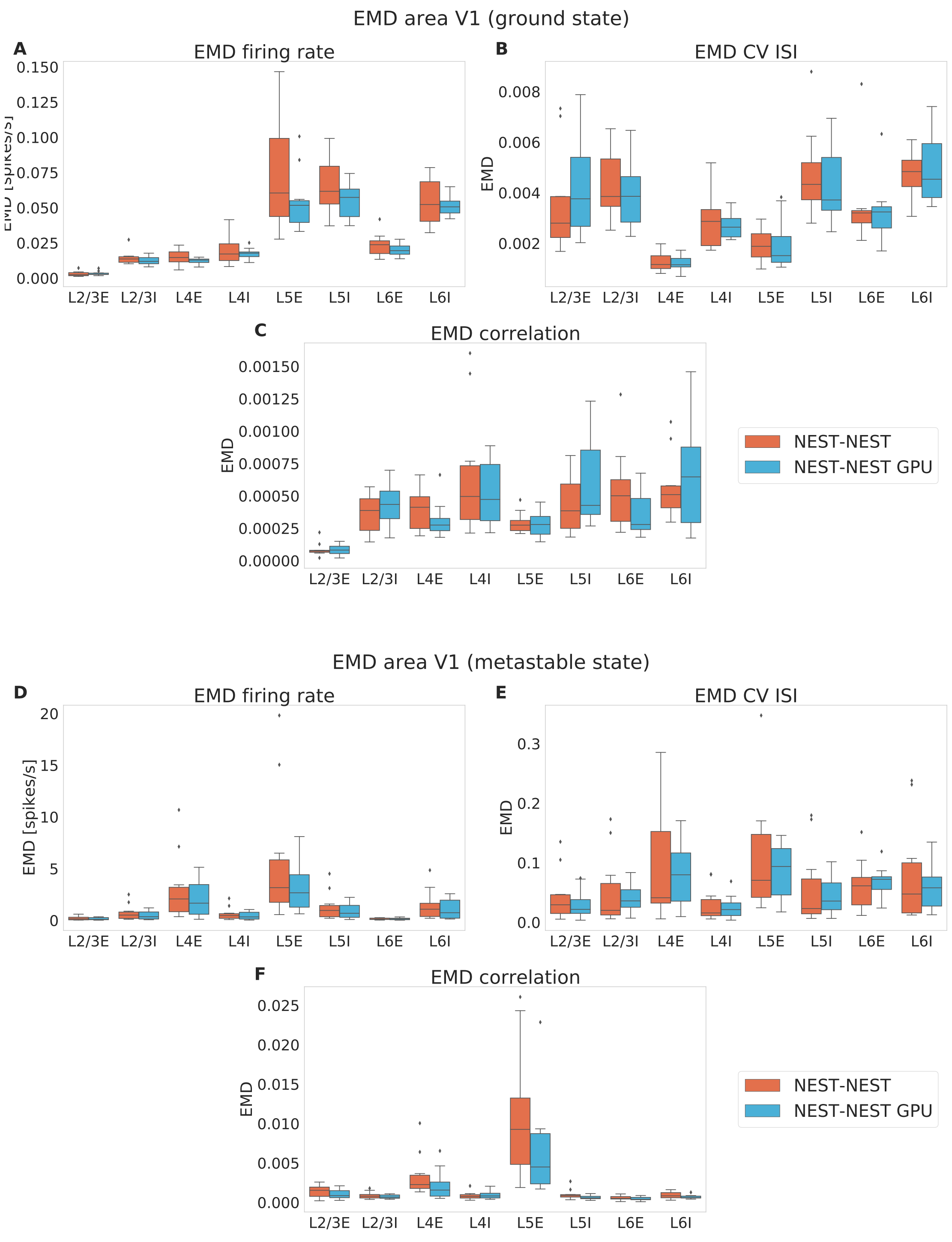


Figure 5.8: Earth Mover's Distance between distributions of firing rate, CV ISI and correlation of the spike trains obtained from the area V1 of the model in the ground state **(A, B, C)** and the metastable state **(D, E, F)** simulated with NEST and NEST GPU. EMD boxes are obtained comparing NEST using different seeds (NEST-NEST, orange) and NEST and NEST GPU (NEST-NEST GPU, sky blue). The high variability of the metastable state can be evidenced by the large difference in the scale of the plots **(A-C)** and **(D-F)**.

Chapter 6

NEST GPU performance evaluation

Summary

Here, NEST GPU performance is evaluated using different hardware and parallel configurations. First of all, benchmarks on the simulation of the cortical microcircuit model are presented, with a focus both on the network construction and on the simulation of the network dynamics. Then, performance on the simulation of the multi-area model on an MPI-GPU cluster is discussed. Finally, the scaling performance of the simulator is shown using the balanced network model with a different number of neurons and synapses.
Benchmarks oriented toward the estimation of the network construction time on NEST GPU are performed together with Jose Villamar, a PhD student at the Institute of Neuroscience and Medicine (INM-6) of the Jülich Research Center (Germany).

6.1 Phases of the simulation

To properly evaluate the performance of the library we have to distinguish at least two phases of the overall simulation time (or time-to-solution): *network construction* and *simulation*. The network construction phase handles all the steps until the actual simulation of the network dynamics starts, with the latter handled by the simulation phase. To go more into detail, the network construction phase is characterized by the following stages:

- *initialization*, which is a setup phase in the Python script that imports modules, sets the parameters, etc;
- *node creation*, which instantiates the neurons and the devices of the network model;
- *node connection*, which instantiates the connections between the network nodes;

- *calibration*, a preparation phase in which connections are properly ordered and organized suitably for the spike delivery algorithm, and spike buffers and spike arrays are initialized.

In the previous version of NEST GPU and in the prototype library NeuronGPU, the calibration phase was also used to move the data from RAM to GPU memory, since the network construction phase was done at the CPU level and then copied to GPU. However, the novel algorithm proposed in [4] directly performs the network construction phase on the GPU device, avoiding the copy operation.
As the network construction phase, also the simulation phase can be divided into subtasks, which are:

- *delivery*, which represents the time for local spike handling and delivery;
- *communication*, which describes the time for remote (i.e., through MPI) spike handling and delivery. This subtask is present only for multi-GPU simulations;
- *collocation*, which describes the time needed for preparing the MPI send buffers (ergo is present only in multi-GPU simulations);
- *update*, representing the time needed to update the dynamics of neurons and synapses;
- *other*, which is a general subtask that collects minor contributions to the simulation time that do not belong to the previous subtasks.

6.2 Simulations of the cortical microcircuit model

After having validated NEST GPU on the simulation of the cortical microcircuit model in Section 5.2, here we evaluate the performance of the library. Simulations of such a network model are usually performed on a single GPU since the full-scale model easily fits the GPU memory of relatively recent GPU cards, and even high-end laptops can simulate the model within a reasonable time-to-solution.
Here we divide the performance evaluation on the simulation of the cortical microcircuit into two subsections to distinguish between the performance evaluation of the network construction phase and the simulation of the network dynamics.

6.2.1 Network construction time performance

The network construction phase is an "intensive property" of the overall simulation, meaning that it does not depend on the biological time we want to simulate. Thus, an appropriate time for simulating the dynamics would lead to a minor contribution of this time to the overall simulation time. However, it should not be considered as a simple overhead not worthy to be optimized. Indeed, this phase can have a significant time contribution in certain applications such as parameter scans, in which several simulations are needed to be tested and thus a non-optimized network construction phase would strongly condition the overall

time.
In [4] we tested the new network construction algorithm able to construct the network at runtime on GPU. We validated the new algorithm, which currently can only be applied to single-GPU simulations, on the simulation of the cortical microcircuit model. While in the previous version of NEST GPU the network construction depended on the CPU of the workstation, now this time depends uniquely on the GPU card employed (as the simulation time).
To evaluate the performance at this stage, we used different systems employing different NVIDIA GPU cards. The table below fully describes the hardware these systems are equipped with.

System	CPU	GPU
JUSUF cluster (node)	2× AMD EPYC 7742, 2×64 cores, 2.25 GHz	NVIDIA V100, 1530 MHz, 16 GB HBM2e, 5120 CUDA cores
JURECA-DC cluster (node)	2× AMD EPYC 7742, 2×64 cores, 2.25 GHz	NVIDIA A100, 1410 MHz, 40 GB HBM2e, 6912 CUDA cores
Workstation 1	Intel Core i9-9900K, 8 cores, 3.60 GHz	NVIDIA RTX 2080 Ti, 1545 MHz, 11 GB GDDR6, 4352 CUDA cores
Workstation 2	Intel Core i9-10940X, 14 cores, 3.30 GHz	NVIDIA RTX 4090, 2520 MHz, 24 GB GDDR6X, 16384 CUDA cores

Table 6.1: Hardware configuration of the different systems used to evaluate the performances. Cluster information is given on a per-node basis.

First of all, we want to compare the performance of the new algorithm of network construction of NEST GPU with the previous one. We call *NEST GPU onboard* the version of NEST GPU which implements the novel algorithm since all the phases of the simulation take place on the GPU card. We call then *NEST GPU offboard* the previous version of the library for which the network construction phase was handled by the CPU and then the network was copied into the GPU memory for the simulation of the network dynamics.
For the comparison, we performed 10 simulations using different seeds for every system configuration shown in Table 6.1. Times are extracted using Python timers, in particular using the function `perf_counter_ns` of the `time` module, which returns each time measured in nanoseconds, i.e., the best possible resolution at Python level.
Figure 6.1 directly compares the two network construction approaches using the different computing systems described in Table 6.1.
As can be seen, the novel method for network construction directly on GPU memory outperforms the method applied in the previous version of the library, granting a speed-up of two orders of magnitude. With this method, the full-scale cortical microcircuit model can be constructed in less than a second both on consumer-grade GPUs and data center GPUs.

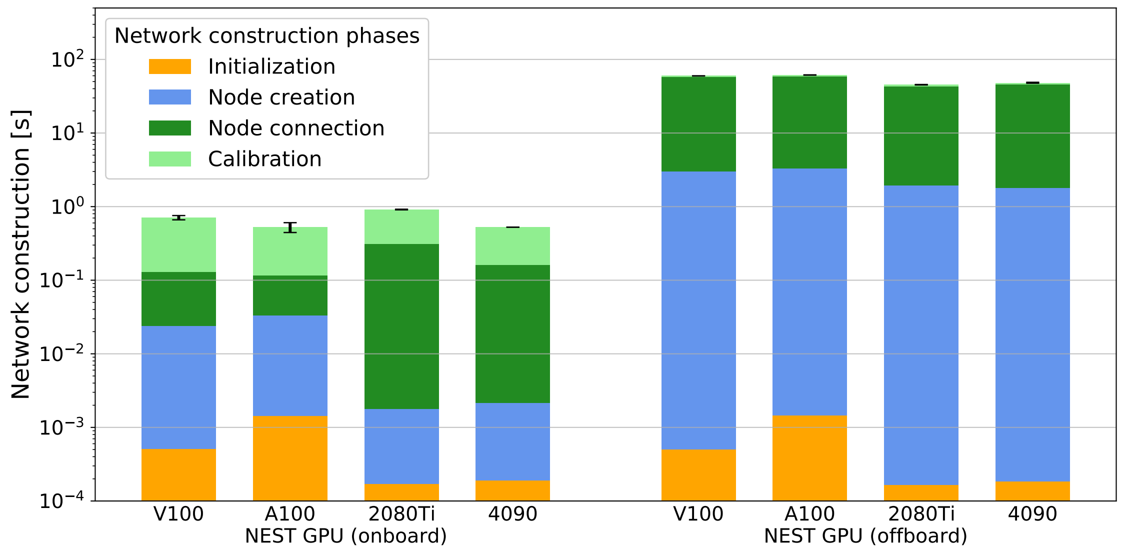


Figure 6.1: Comparison of the network construction phase for the two versions of NEST GPU in the simulation of the cortical microcircuit model. Simulations are performed on different hardware configurations, here indicated with the GPU device they equip. Error bars show the standard deviation of the time needed for network construction over ten simulations using different seeds. Figure adapted from [4].

Additionally, we also compared the performance of the novel algorithm with the CPU-based simulator NEST (version 3.3 [117]) and the GPU-based GeNN [83] (version 4.8.0[1]). In particular, the GeNN simulator differs in terms of the network construction approach from NEST GPU since it adopts a code-generation approach. Thus, GeNN code has to be generated and compiled before execution. In this simulator, the network construction phase can be decomposed into the following subtasks:

- *model definition*, which defines neurons, devices, and synapses of the network;
- *building*, which generates and compiles the simulation code;
- *loading*, which allocates memory and instantiates the network on the GPU.

Also, in this case, the time to complete the subtasks is measured through Python timers. Regarding NEST, the network construction is done at runtime as in NEST GPU, with analogous network construction subtasks. Figure 6.2 shows the network construction time for NEST GPU, NEST, and GeNN using different hardware configurations.
As can be noticed, NEST GPU results faster in the network construction with respect to NEST, and also to GeNN. However, it should be noted that the *building* phase of GeNN, related to code creation and compilation, take place only if code has not been generated and compiled or when network parameters have to be

[1]`https://github.com/genn-team/genn/releases/tag/4.8.0`

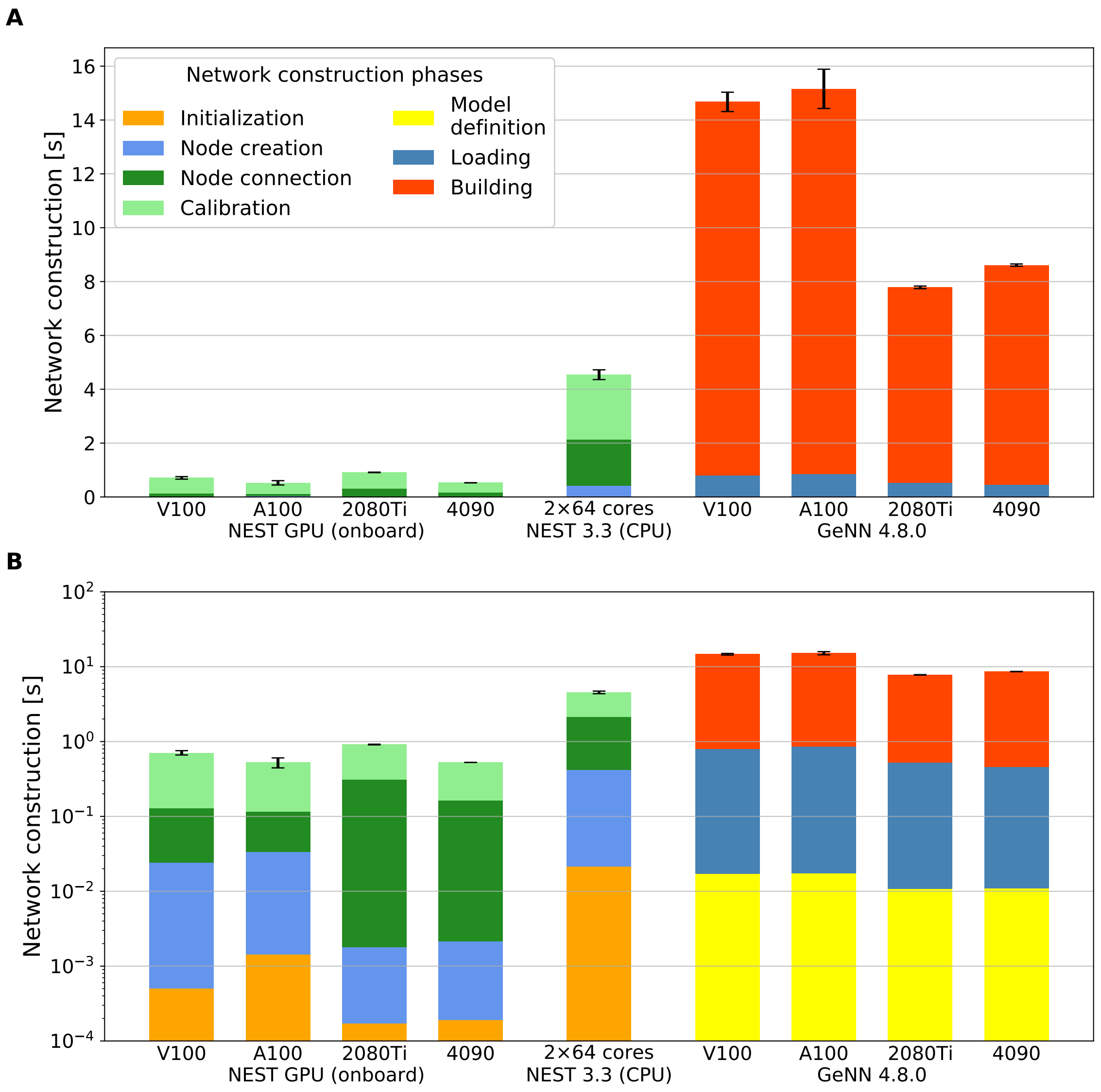


Figure 6.2: Performance comparison of the network construction phase for different simulators and hardware configurations on the cortical microcircuit model. Data for NEST GPU *(onboard)* is the same as in Figure 6.1. **(A)** Network construction time of the model in linear scale for different simulators and hardware configurations. **(B)** as in **A** but with logarithmic y-axis scale. In both panels, the *building* phase of GeNN is placed on top of the bar, breaking with the otherwise chronological order, because this phase is not always required and at the same time, this display makes the shorter *loading* phase visible in the plot with the logarithmic y-axis. Error bars show the standard deviation of the overall network construction phase over ten simulations using different random seeds. Figure from [4].

changed. If these are not the cases, this phase is not present, making the network construction times of NEST GPU and GeNN compatible. Nonetheless, NEST GPU approach works on runtime, so the time for code compilation is not needed anyway, increasing the flexibility of this phase.
Moreover, Figure 6.2 shows the performance for network construction when, in NEST and NEST GPU, external stimulation of the model is driven by Poisson spikes generators. Indeed, the NEST and NEST GPU implementations can have this kind of device or DC input as external stimulation. However, the network construction times obtained using the DC input in NEST and NEST GPU are similar to the one shown in the figure.
In summary, the new network construction of NEST GPU is able to achieve state-of-the art performance in the network construction of networks that can be simulated on single GPUs. Indeed, NEST GPU network construction times are compatible with code generation-based simulation codes such as GeNN. Currently, the major limitation of this algorithm is that it can be employed only for single-GPU simulations, however, an extension for the algorithm to multi-GPU simulations is in development. A discussion about the idea behind this extension of the algorithm and how should it work is presented in the outlook of this thesis.

6.2.2 Simulation performance

Evaluating the simulation performance is important to provide an efficient simulation code: indeed, codes that run at realtime are fundamental for several applications, such as robotics and studies of large-scale models oriented towards the study of cognitive processes and brain development in general. As a normal practice, when evaluating simulation performance spike recording is disabled. Differently from the network construction phase, the simulation phase is an "extensive measure", meaning that the length of the wall-clock time depends on the amount of biological time that has to be simulated. In order to provide a measure to assess the simulation speed we use the real-time factor (RTF), defined as

$$\mathrm{RTF} = \frac{T_{\mathrm{wall}}}{T_{\mathrm{model}}} \tag{6.1}$$

where T_{wall} is the wall-clock time, i.e., the time needed for the simulator to complete the simulation of the network dynamics, whereas T_{model} is the biological time that is simulated.
Here we evaluate the performance of the simulation of the network dynamics for NEST GPU (onboard) on the simulation of the cortical microcircuit model. To simplify the notation, from now on we will drop the term "onboard" so that for NEST GPU we refer to the most recent version of the library.
Since this model was the most tested one when optimizing the library, here we can provide benchmarks using different versions of the library. In particular, we will focus on the simulation of the model using the Poisson signal generators as external stimulation. Indeed, there is a relevant difference in the simulation time when Poisson signals or DC input are employed for external stimulation, with the latter leading to the best performance. However, since a stimulation through Poisson-like spike trains is the most plausible from a neuroscientific point of view, the plot,

if not specified, will show the performance when these devices are employed on NEST and NEST GPU simulations.
To show the improvement of the library over the last few years, Figure 6.3 shows the simulation time of the current version of NEST GPU and the prototype library NeuronGPU using the different hardware systems shown in Table 6.1.

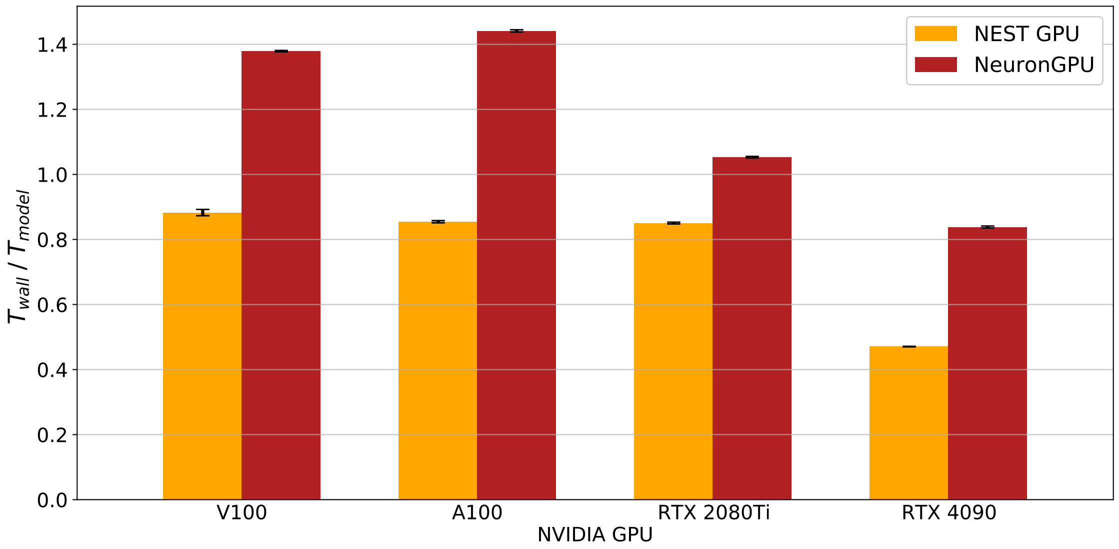


Figure 6.3: Real-time factor of cortical circuit model simulations for NEST GPU (amber) and NeuronGPU (dark red). Here, $T_{\text{model}} = 10\,\text{s}$ and 10 simulations for each hardware configuration are performed. Error bars show the standard deviation calculated over the 10 simulations.

As can be seen, NEST GPU library has been optimized in the simulation phase, with improvements that span from 55% to 75% with respect to NeuronGPU, the prototype library introduced in [1]. This improvement is mainly due to an optimization of the CUDA synchronous calls in the NEST GPU code, which were removed when unnecessary. This led to a removal of the latency due to the synchronization between CUDA threads during the simulation phase which were not necessary for performing simulations correctly. This work was done with the help of NVIDIA developers in the context of a GPU hackathon, during which I and the rest of the NEST GPU team were involved in several optimizations of the GPU code of the library. In that context, we also focused on the implementation of multiple algorithms to perform nested loops on GPU code in the most efficient way possible. Indeed, part of the performance improvement is also due to the choice of a more efficient algorithm for this operation. In this regard, further work is needed to evaluate the most efficient algorithm to perform this task and, more in general, optimization of the library using techniques such as performance profiling will be the object of future work to further improve NEST GPU performance. This topic will be discussed in the outlook of this thesis.
Together with the comparison of the results of the spiking activity, we also compare the performance of NEST and NEST GPU. Moreover, we also perform a comparison between NEST GPU and other GPU-based simulators such as GeNN [83],

which has established itself as one of the most used GPU-based simulation codes. In particular, in [86] the GeNN implementation of the cortical microcircuit model was presented, thus we decided to compare the two simulators in the simulation of this model. Indeed, results changed over the last years: Figure 6.4 shows the results of the comparison obtained in [1] and the one obtained in [4] for different versions of the GPU libraries.

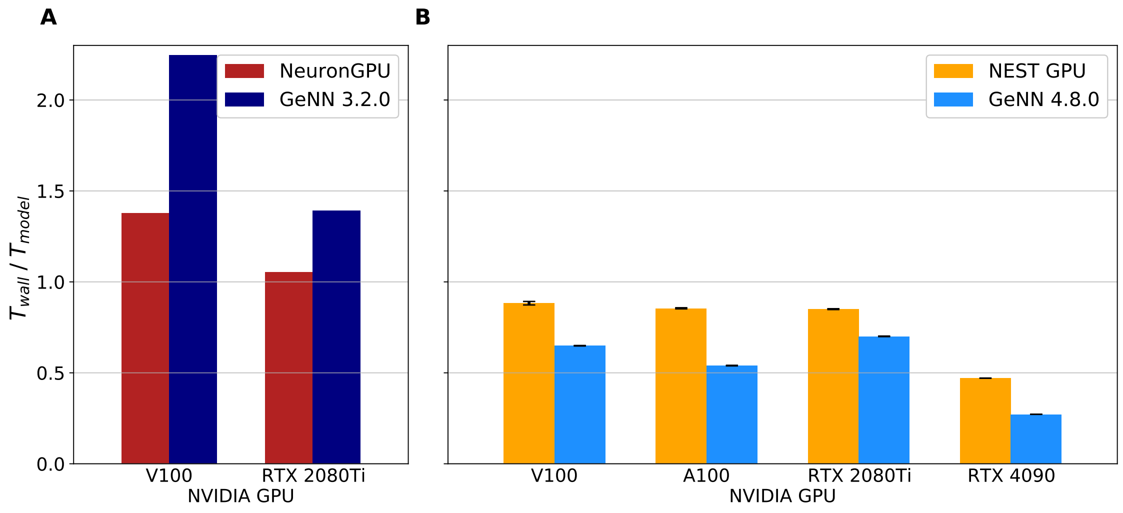


Figure 6.4: Real-time factor using different NVIDIA GPUs for different GPU-based simulation codes. **(A)** real-time factor of the prototype library NeuronGPU (red) and GeNN 3.2.0 (navy), from [1]. **(B)** real-time factor of the most recent version of NEST GPU (amber) and GeNN 4.8.0 (sky blue). Data for NEST GPU is the same as in Figure 6.3.

As can be seen, panel A shows that, using the same GPU hardware, NeuronGPU was able to outperform GeNN (version 3.2.0 [2]), whereas in the current version of the libraries (see panel B), GeNN performs the simulation of the cortical microcircuit faster than NEST GPU.
However, simulations with NEST GPU here use Poisson signal generators as external input. Indeed, using DC input can lead to a remarkable speed-up of the simulation. GeNN employs a different mechanism for external stimulation, which mimics incoming Poisson spike trains through a current directly applied to each neuron. For this reason, it is interesting to compare both the NEST GPU implementations with Poisson spike and the DC input-driven stimulation with the GeNN code.
To represent the current state-of-art for the simulation of the cortical microcircuit model on conventional computing architectures, Figure 6.5 represents the real-time factor for NEST GPU, GeNN 4.8.0, and NEST. The NEST (GPU and CPU) performance is measured both when employing the Poisson spike generators and DC inputs as external stimulation.

[2]`https://github.com/genn-team/genn/releases/tag/3.2.0`

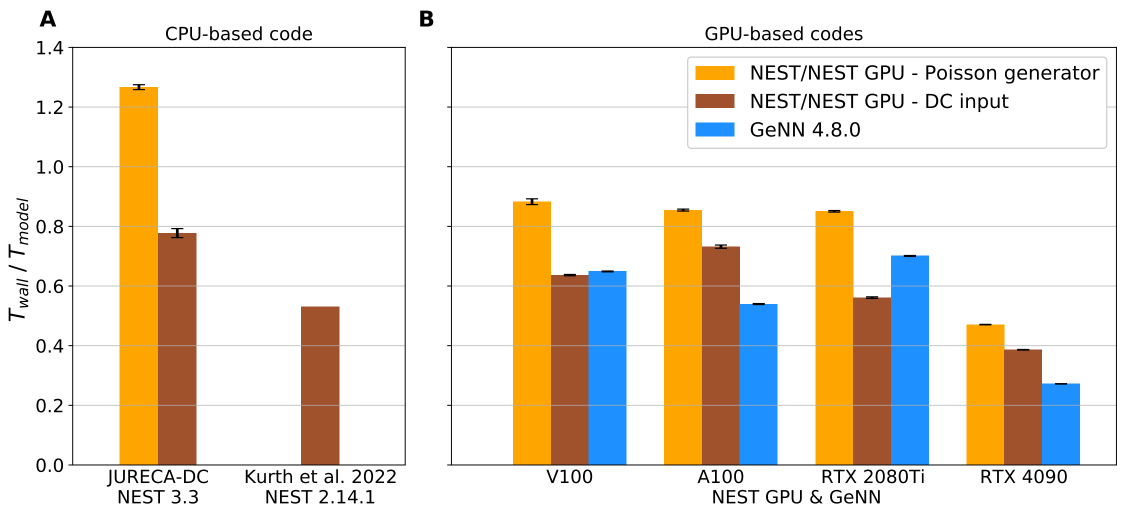


Figure 6.5: Real-time factor using different CPU-based and GPU-based simulation codes, with the model employing different external stimuli; i.e., Poisson spike trains (amber), DC input (sienna), or the one designed by GeNN (sky blue). **(A)** real-time factor of different versions of NEST using different hardware and parallelization approaches. Data for NEST 3.3 from [4], data for NEST 2.14.1 from [73]. **(B)** real-time factor of NEST GPU and GeNN 4.8.0 using different NVIDIA GPUs. Data from [4].

To provide more detail on the NEST performance, the one of NEST 3.3 is presented on [4] and uses a single compute node of the HPC cluster JURECA-DC [119] (see also Table 6.1), exploiting its 128 cores by 8 MPI processes each running 16 threads. The performance achieved using NEST 2.14.1 [120] is from [73] and is obtained using two nodes equipped with two AMD EPYC Rome 7702 with 64 cores each, with 2 MPI processes per node. This result is, at the time of writing, the best result achieved by NEST in the simulation of this model.
Figure 6.5 shows that GPU-based performances are all below real-time, with the best results achieved by GeNN or by NEST GPU when the external stimulation is driven by a DC input. It is also interesting to note that data center GPUs such as the NVIDIA V100 and the NVIDIA A100 show performance compatible (or even slower) with respect to recent consumer GPUs such as the RTX 2080Ti and the RTX 4090. In particular, the remarkable results on the latter can be related to the higher clock speed of this GPU card. Moreover, NEST GPU performs the computation using single-precision floating point (FP32), which explains the fact that data-center GPUs such as the A100, with better performance than consumer GPUs on double-precision floating point operations, show similar performance compared to RTX 2080. A further investigation of the relationship between the performance and the specifications of the GPU device will be the object of future work.

6.3 Simulations of the multi-area model

Here we present the performance evaluation of the multi-area model implementation for NEST GPU. The network is comprised of 32 areas, each one designed similarly to the cortical microcircuit. NEST GPU simulations were performed using the JUSUF computing cluster [121], whereas NEST simulations were done on JURECA-DC [119]. See Table 6.1 for the specifications of each node of these computing clusters.

Since at most one of the model areas can be simulated on a single NVIDIA V100 (i.e., the GPU card with which JUSUF nodes are equipped), we decided to parallelize the simulation as follows: we employ 32 nodes equipped with a GPU each, with a single MPI process per node. In this regard, we decided to simulate the model so that each area is simulated in a single MPI process, i.e. in a single GPU card.

Such an approach is possible thanks to the locality exploitation of NEST GPU, which enables the creation of entire neuron populations in a defined MPI process. Indeed, this is one of the major differences with respect to NEST, since the CPU code adopts a round-robin approach to evenly distribute the neurons across all the MPI processes. Indeed, this approach, adopted also in [92], can be significantly advantageous for multi-area models such as the one simulated here, since the density of inter-area connections greatly exceeds the density of intra-area connections. From a simulation point of view, this means that simulating each area in the same MPI process can greatly reduce the number of spikes that have to be sent to other areas of the model, thus reducing the compute load of MPI communication, which typically incurs higher overhead compared to intra-node communication.

To evaluate the performance of the simulation of this model, we simulated the ground state and the metastable state, without recording the spikes. While we employed 32 compute nodes with 1 MPI process each for the NEST GPU simulations, we performed a strong scaling using a variable amount of compute nodes for the NEST (version 3.0 [122]) simulations, finding that the fastest performance was achieved when simulating the model on 32 nodes of JURECA-DC, with 8 MPI processes per node and 16 threads per task. The strong scaling was done using the benchmarking framework beNNch [123].

Regarding the version of NEST GPU employed, as previously mentioned, the most recent version of NEST GPU can not be used since it employs the network construction algorithm to construct the network in GPU memory, which has not yet been implemented for multi-GPU simulations. Moreover, the version used here version still differs from the one previously called NEST GPU (*offboard*), since it lacks of the optimization mentioned in the previous section regarding asynchronous calls. This has the disadvantage of increasing the simulation time, but has the advantage of providing the exact times for the simulation subtasks, which the current version is still not able to provide[3]. This way, we can also evaluate the amount of time taken by each stage of the simulation, which can be advantageous when comparing different approaches for network simulation across different MPI processes.

[3]A solution for this issue is currently object of study from me and the rest of the NEST GPU team.

Figure 6.6 shows the real-time factor for the simulation of the multi-area model using NEST and NEST GPU.

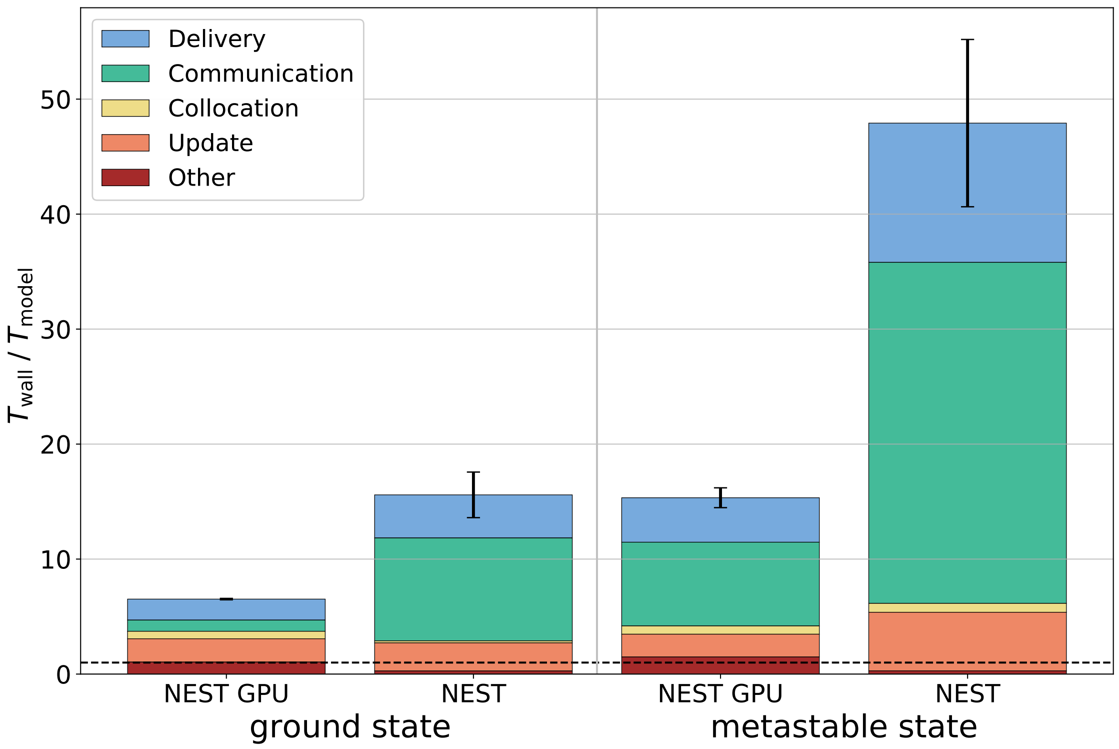


Figure 6.6: Real-time factor for ground and metastable state simulations of the multi-area model using NEST and NEST GPU. The time taken by each simulation subtask is averaged over the MPI processes and results are then averaged over 10 simulations using different seeds. Error bars show the standard deviation of the overall performance across the 10 simulations. Adapted from [2].

As can be seen in Figure 6.6, NEST GPU is able to achieve, for the ground state, a real-time factor of 6.5 ± 0.1, which is around 2.4 times faster than NEST. In the metastable state, NEST GPU has a speed-up factor of 3.1 times with respect to NEST, with a real-time factor of 15.3 ± 0.9. The longer simulation time taken for the metastable state is explained by the higher firing rates and synchrony in this state.

The main difference between the simulators appears in the time taken by spike communication, evidencing the advantage of exploiting a neuron distribution among MPI processes that takes into account spatial locality. Indeed, the round-robin distribution of neurons in NEST necessitates the communication between MPI processes of a larger amount of data since also intra-area spikes have to be delivered through MPI.

The relative contributions of the various phases do not differ strongly between the ground and metastable states. The contribution of the communication of spikes between different MPI processes for the metastable state of the model is around 8.0 s and 29.7 s per second of biological time for NEST GPU and NEST, respectively. The contribution of update, delivery, and other operations, excluding the commu-

nication of spikes between different MPI processes, is around 7.3 s for NEST GPU and 18.0 s for NEST. We can therefore observe that the better performance of NEST GPU compared to NEST is mainly due to a reduction in the communication time of the spikes between MPI processes, although there is an improvement also in the time associated with the update and delivery of local spikes.
Nevertheless, it could be argued that using this approach for simulating the areas of the model would lead to an unbalanced distribution of compute load across simulations, and thus to different simulation times. However, since MPI processes are synchronized at the end of every simulation time step, the overall simulation time shown by every MPI process is the same. Indeed, each MPI process shows different times for the simulation subtasks, depending on the number of neurons to be simulated or spikes delivered inside and outside the process, with latency due to synchronization between the processes at every time step. The resulting latency due to the difference between model areas is embedded in the Communication subtask.
For a better estimation of the subtask times in every MPI process, Figure 6.7 shows the relative contributions to the total simulation time of the simulation subtasks for each MPI process (i.e., model area).

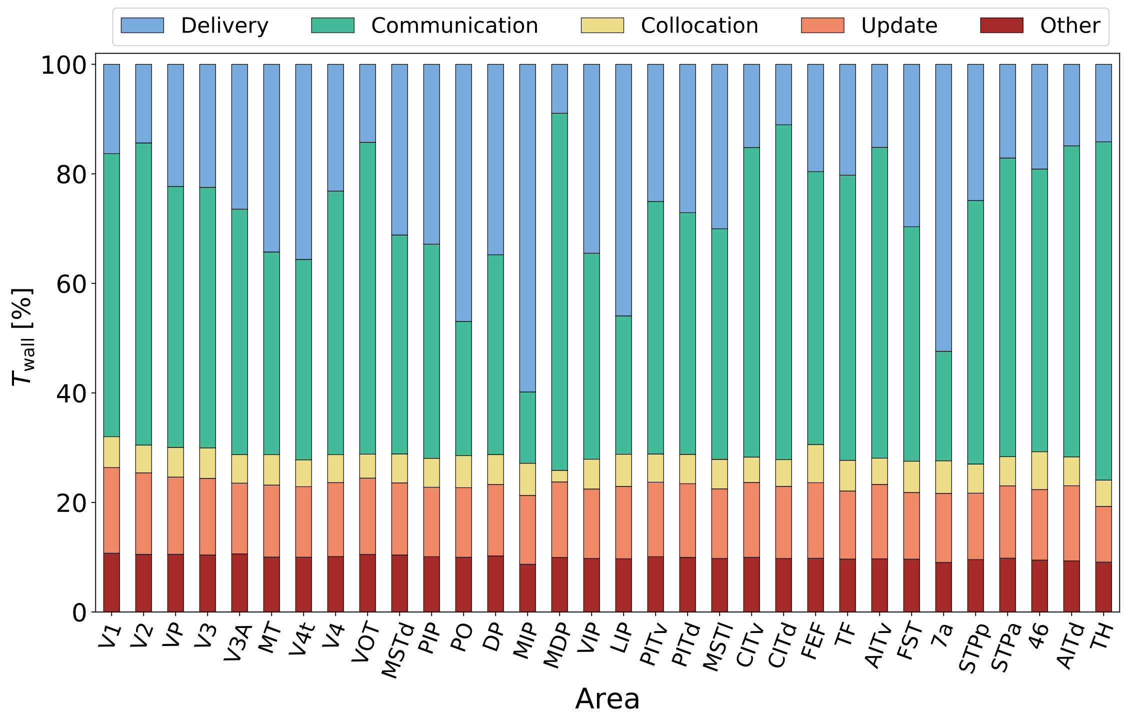


Figure 6.7: Relative contributions to the simulation time of the multi-area model in the metastable state for every area (i.e. for every MPI process) in a NEST GPU simulation. The latency due to the MPI synchronization is included in the Communication subtask. Figure from [2].

We measured that, within a simulation, the contribution of the spike communication between the 32 MPI processes (i.e. the 32 areas of the model) can vary up to 25% with respect to its average shown in Figure 6.6 and the contribution

of the local spike delivery subtask shows comparable variations. The rest of the subtasks (i.e. Collocation, Update, and Other) do not change significantly across the MPI processes, as shown in Figure 6.7. Moreover, we also evaluate the amount of spikes that have to be delivered through MPI by the multi-area model, as a fraction of the total number of spikes delivered during a simulation.
Figure 6.8 shows that the vast majority of the spikes emitted in a second of biological time are delivered within the same area, and only a small fraction of the spikes is delivered to different areas because the inter-area connections represent only a minority of the total connectivity of the model [89]. Therefore, the NEST GPU neuron distribution, which exploits spatial locality, strongly reduces the number of spikes that have to be communicated between MPI processes and thus contributes to the overall simulation time reduction.

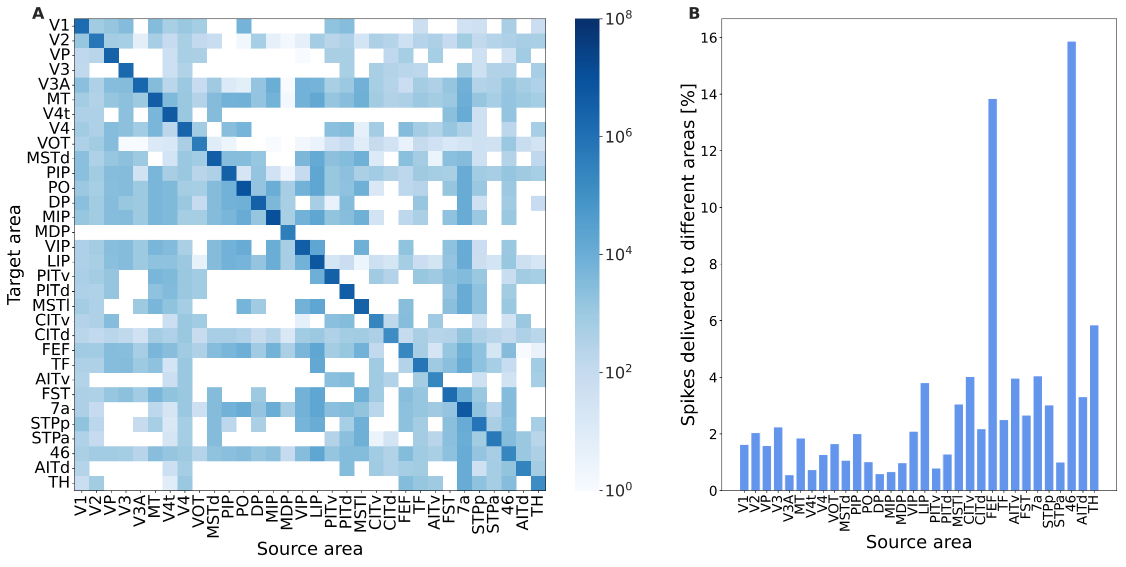


Figure 6.8: Spikes delivered in a simulation of the full-scale multi-area model in the metastable state. **(A)** Heatmap of the number of spikes delivered from each area of the model for a second of biological time. The diagonal elements show the total number of spikes fired by the neurons within the model area, whereas the off-diagonal elements show the number of spikes delivered to different areas of the model. **(B)** Fraction of spikes delivered to different areas of the model (i.e. to different MPI processes), obtained with the ratio between the number of spikes sent by a source area to every other area of the model and the total number of spikes emitted by the source area neurons. Figure from [2].

In case of enabled spike recording using NEST GPU the simulation time increases up to 5% when recording from all neurons. In these simulations, packing of recorded spikes and transfer to the CPU memory is performed every 2000 simulation time steps (i.e. every 200 ms of biological time). This overhead is strongly dependent on the model simulated and the amount of GPU memory available. Indeed, a larger GPU memory would support larger buffers of recorded spikes, diminishing the frequency of copy operations from GPU memory to CPU memory. Furthermore, the overhead can be reduced by recording spikes from only a fraction of the neurons.

Regarding the network construction phase, the algorithm employed by NEST GPU is the one adopted by NeuronGPU. Indeed, simulations of this model reveal the limits of the previous approach: the network, both in the ground and in the metastable state, was constructed in around 955 s on NEST GPU (more precisely, 951 ± 29 s for the ground state and 957 ± 41 s for the metastable state), whereas the same network was constructed by NEST in around 75 s (80 ± 7 s for the ground state and 69.5 ± 0.4 s for the metastable state). These results suggest that an extension of the network construction algorithm proposed in [4] for multi-GPU simulation can significantly reduce the time-to-solution when simulating a large-scale model requiring an MPI-GPU cluster to be simulated. Indeed, preliminary tests suggest that an MPI implementation of this algorithm can reduce the network construction time to around 60 s, but additional work is required to validate this new implementation.

6.4 Simulations of the balanced network model: NEST GPU scaling performance

In order to evaluate the scaling performance of NEST GPU, we used the balanced network model described in Section 4.1. Indeed, the neurons or the connections per neuron simulated can differ, with a scaling that preserves the average spiking activity of the network. This way, we are able to have an idea about the times taken by the simulator to perform simulations of networks with a certain number of neurons and connections per neuron on a single GPU[4].
In particular, here we evaluate the NEST GPU scaling on the simulation of the balanced network model using Workstation 1 described in Table 6.1 (i.e., the one with the NVIDIA RTX 2080 GPU). Among the compute systems used, this is the most affordable one, and using a consumer-grade GPU which is not necessarily the most performing device in the market can be of broader interest for users that want to use this library on local systems.
The balanced network model is composed of AdEx neurons, which are integrated using the fifth-order Runge-Kutta (5th-RK) method with adaptive step size. Usually, the adaptive step size for GPU simulators is not used, since GPU simulations would largely benefit from fixed step size because of the SIMD parallelism. Indeed, it is not obvious a priori that a GPU simulator would be faster than a CPU-based code using this kind of integration method. Indeed, one of the next steps for the simulation of AdEx models in NEST GPU would include the possibility of using the 5th-RK method without the adaptive step size in order to speed-up simulations with these neuron models.
To perform the scaling, we chose to simulate a network with 5000 (4000 excitatory and 1000 inhibitory) connections per neuron. This is a reasonable number of connections for a natural density network, which can span from a few thousand to 10000 connections per neuron. The cortical microcircuit model of [12], for exam-

[4]Having a similar evaluation for multi-GPU simulations is an object of current work; indeed, we are currently implementing of the so-called *HPC benchmark* introduced in [124]. However, this object will not be discussed in this thesis.

ple, has on average 4000 connections per neuron and the multi-area model of [89] has 6000 connections per neuron on average[5]. The total number of neurons spans from 10000 to 130000. A greater amount of neurons can not be simulated for GPU memory limits, which depends on the GPU device used for simulation (here, 11 GB). Figure 6.9 shows the weak scaling performance for the simulation of $1\,\mathrm{s}$ of biological activity of the balanced network, which shows an average firing rate of around $25\,\mathrm{Hz}$. The results are averaged over 10 simulations using different seeds for random number generation.

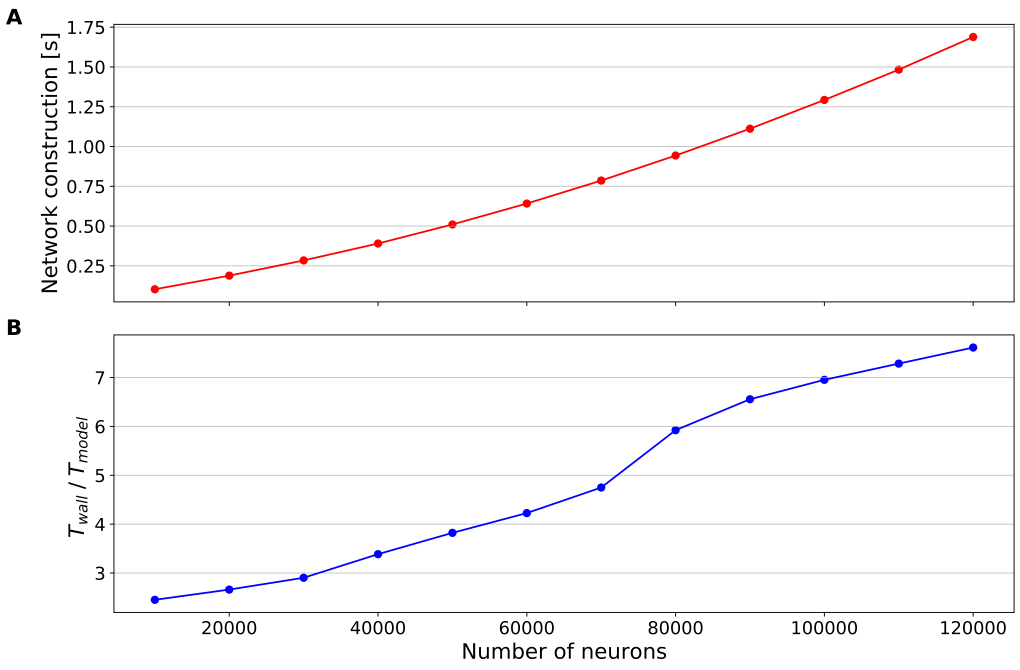


Figure 6.9: Scaling of the balanced network model for NEST GPU using an RTX 2080Ti GPU card. The network is simulated using a variable number of neurons and 5000 connections per neuron. **(A)** Network construction time as a function of the number of neurons simulated. **(B)** Real-time factor as a function of the number of neurons. All the values are averaged over 10 simulations using different seeds.

As can be seen in Figure 6.9, the behavior of network construction and real-time factor is approximately linear as a function of the number of neurons simulated. Moreover, when comparing these results to the one obtained in the simulation of the cortical microcircuit model (with around 80000 neurons), we notice that the network construction is compatible with the data shown in Figure 6.1, whereas the results of the real-time factor are different from the shown in Figure 6.5. Indeed, this is due to a different firing rate of the neurons of the simulations and also to the fact that the neuron models are different and, most importantly, integrated using different techniques. However, in this regard, it is interesting to provide the performance on a type of neuron model that is integrated differently that was not used before in the simulations of the large-scale models treated here in this thesis.

[5]These are just rough estimations; indeed, each layer or model area has a different indegree.

Part III

Synaptic mechanisms and cognitive processes: from the synaptic theory of Working Memory to a framework for learning through Structural Plasticity

Chapter 7

How to model Working Memory: an STP-driven spiking model

Summary

This chapter first introduces the synaptic theory of Working Memory. Then, we present and describe the reproduction of the working memory spiking network model with STP introduced in [23]. The model has been implemented using the Python interface of the NEST simulator with the aim of paving the way for further research oriented toward the study of the relationship between working memory and synaptic mechanisms.

7.1 Synaptic theory of Working Memory

As discussed in Section 2.1.1, Working Memory is a cognitive process that handles information manipulation and storage for a short time. Indeed, this cognitive process has a limited capacity, i.e., there is a limit on the number of items that can be kept in memory in order to manipulate them, and this number varies depending on the task that has to be performed [125–127]. The first computational models rely on attractor network dynamics to explain how neural circuits are able to encode pieces of information for a short amount of time [31, 32, 94], and thus these studies were all based on the fact that these items (or *chunks*) were only recalled by the persistent spiking activity of a neuron population, making the population converging to a high-rate attractor. This fact was also grounded by experimental results on delayed-response tasks [28, 29], according to which the neural activity is enhanced during the delay. However, a mechanism based only on spiking activity can be biologically expensive, and experimental observations suggested also that the item can be kept in memory even after a brief interruption of the enhanced neural activity during the delay [33]. Moreover, it was observed that the higher activity during the delay takes place in the prefrontal cortex (PFC), which shows an abundance of facilitating synapses with respect to other regions of the brain [128]. In 2008, Mongillo et al. [23] proposed that working mem-

ory can be sustained by a synaptic mechanism so that an item can be stored in synaptic connections as an activity-silent event. The item can then emerge in a spiking fashion after a small stimulation [129]. The synaptic theory of working memory relies uniquely on short-term synaptic plasticity, and in particular on the mechanism of facilitating synapses, described in Section 2.1, related to the neurotransmitter and presynaptic calcium concentration dynamics. Facilitation leads to a short-term enhancement of synaptic efficacy in synapses that were previously utilized (i.e. triggered by an external input) and, this way, synapses encode the item silently and retrieve it when a subsequent signal (called readout signal) is injected.
To understand the mechanism underlying the activity-silent item upkeep, let us consider a network formed by several neuron populations, each one able to encode a different item thanks to a previous learning process. When an external signal targets a neural population increasing its spiking activity, the synapses connecting neurons of the targeted population remain facilitated for a time in the order of the facilitation time constant τ_f. The connections between neurons inside the pre-stimulated population are potentiated because of prior long-term Hebbian learning, and also because of the STP modulation. On the other hand, connections between neurons belonging to other populations are relatively weaker because they lack short-term potentiation, while the connections between neurons belonging to different selective populations are weaker because they have not been previously potentiated by long-term Hebbian learning. A similar effect could be driven by random fluctuation of neuron activity, inducing an autonomous winner-take-all (WTA) mechanism, according to which the selective excitatory population with the highest firing rate stimulates the inhibitory population eliciting a suppression of the spiking activity of the other excitatory populations [130] due to the global inhibition. Indeed, the global inhibition is provided by the non-specific inhibitory connectivity, in agreement with experimental observations [131]. This mechanism decreases the amount of available neurotransmitters in the presynaptic terminal and increases the calcium concentration across the pre-stimulated population. Since the neurotransmitter dynamics is faster than the calcium one having synaptic facilitation, the connection strength becomes large enough to trigger again the WTA mechanism. This process can be reactivated periodically, and the period of reactivation is related to the neurotransmitter dynamics and in particular to the time constant of the synaptic depression τ_d. This oscillatory behavior can last as long as the net synaptic efficacy remains above a critical level, with a spiking outcome that strongly depends on the input provided to the network.
Indeed, such a mechanism has also a lower metabolic cost than a mechanism relying on persistent activity. As shown in [132] and [133], only about 8% of the ATP consumption for a neuron emitting at $4\,$Hz is related to the mechanism STP is inspired from, whereas spike transmission accounts for around 50%, meaning that activity-silent mechanisms are considerably more efficient from an energy consumption point of view. Moreover, since the items are stored in a synaptic fashion, a small input is enough to retrieve the memory.
The work of Mongillo was supported by spiking network simulations of integrate-and-fire neurons. In [3] we reproduced the results shown in that work, with

qualitative agreement, using the NEST simulator (version 3.1) [134]. The next section describes the spiking network model in detail, showing the results of the simulations and the tests performed on the network model. More details on the compatibility of the results with experimental observations are also provided.

7.2 Spiking model and reproduction of the results

7.2.1 Model description

For a better understanding, here is reported the system of differential equations of STP, as described in Section 2.1. Being x the normalized amount of available resources into the presynaptic terminal and u the fraction of resources used in a spike emission, and considering a synapse connecting the presynaptic neuron i and the postsynaptic neuron j, STP is described by

$$\begin{aligned} \frac{du_{i,j}}{dt} &= -\frac{u_{i,j}-U}{\tau_f} + U(1-u_{i,j})\sum_s \delta(t-t_s^{(i)}) \\ \frac{dx_{i,j}}{dt} &= \frac{1-x_{i,j}}{\tau_d} - u_{i,j}x_{i,j}\sum_s \delta(t-t_s^{(i)}) \end{aligned} \tag{7.1}$$

where $\delta(\cdot)$ is the Dirac delta function, U modulates the increase of calcium concentration due to the spike incoming into the presynaptic terminal and the sum is over the spike times $t_s^{(i)}$ of the presynaptic neuron i. The synaptic modulation takes place during the spike emission, so that

$$J_{i,j}(t) = J_{i,j}^{(\text{abs})}u_{i,j}(t-\hat{\delta}_{i,j})x_{i,j}(t-\hat{\delta}_{i,j}) \tag{7.2}$$

where $J_{i,j}^{(\text{abs})}$ is the absolute synaptic efficacy for the synapse connecting neurons i to neuron j and $\hat{\delta}_{i,j}$ is the synaptic delay. The synaptic efficacy is described by the product Jux.

Now the network model proposed in [23] and reproduced in [3] can be described. A simplified scheme of the spiking network architecture is depicted in Figure 7.1.

The network is composed of N_E excitatory and N_I inhibitory leaky integrate-and-fire (LIF) neurons with exponential postsynaptic currents. The sub-threshold dynamics of the LIF neuron model is described by the differential equation

$$\tau_\text{m}\frac{dV_j}{dt} = -V_j + R_\text{m}(I_j^\text{exc} + I_j^\text{inh} + I_{\text{ext},j}) \tag{7.3}$$

where τ_m is the membrane time constant, V_j is the neuron's membrane potential, R_m is the membrane resistance, I_j^exc and I_j^inh represent the excitatory and inhibitory synaptic current received as input from the connections within the other neurons of the network and $I_{\text{ext},j}$ represents the external input to the network.

The network external input is modeled with Gaussian white noise currents defined by the following

$$I_{\text{ext},j}(t-\hat{\delta}_j) = \mu_\text{ext} + \sigma_\text{ext}G_k \quad \text{for} \quad k\Delta t_\text{ng} \le (t-\hat{\delta}_j) \le (k+1)\Delta t_\text{ng} \tag{7.4}$$

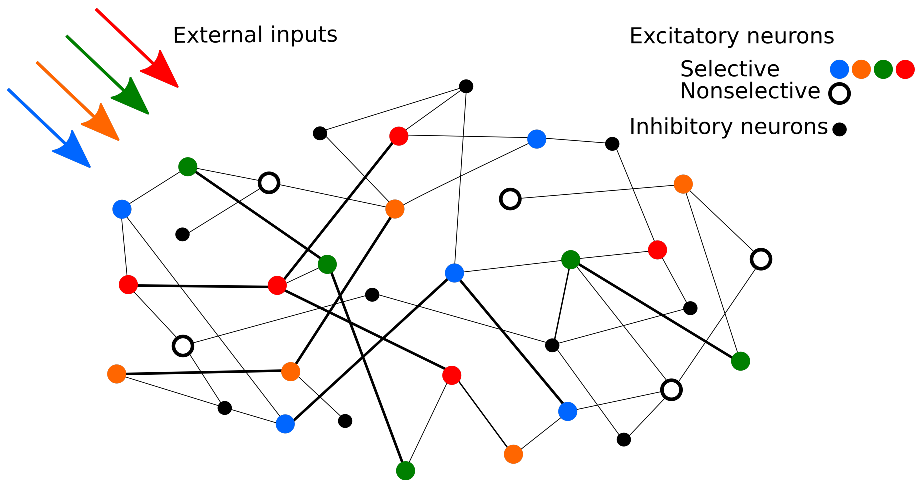


Figure 7.1: Schematic representation of the network. Colored circles represent excitatory neurons of different selective populations (i.e., populations that code for a certain item), whereas the black open circles represent excitatory neurons of the non-selective population. Black circles represent inhibitory neurons. Strengthened connections are represented by thicker black lines. Figure from [3].

In particular, the noise is approximated by a piecewise constant current with mean μ_{ext} and standard deviation σ_{ext}, with constant current during time intervals of length $\Delta t_{\text{ng}} = 1$ ms. Denoting the index of the time interval with k, for each interval the current is given by $\mu_{\text{ext}} + \sigma_{\text{ext}} G_k$, with G_k a random number extracted from a standard Gaussian distribution. The term $\hat{\delta}_j$ indicates the delays.
The synaptic current shown in Equation (7.3) is the sum of the contributions given by the connections with the neurons of the network, and it is characterized by excitatory and inhibitory contributions defined as $I_j^{\text{exc}}(t)$ and $I_j^{\text{inh}}(t)$ respectively. Thus, the synaptic input for a neuron j of the network, with exponential postsynaptic currents, is given by the following equations for excitatory and inhibitory currents respectively:

$$\begin{aligned} \tau_{\text{exc}} \frac{dI_j^{\text{exc}}}{dt} &= -I_j^{\text{exc}} + \sum_i \alpha J_{i,j}(t) \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j}) \\ \tau_{\text{inh}} \frac{dI_j^{\text{inh}}}{dt} &= -I_j^{\text{inh}} + \sum_i \alpha J_{i,j} \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j}) \end{aligned} \tag{7.5}$$

where i is the index of the presynaptic neurons targeting the neuron j. τ_{exc} and τ_{inh} represent the time constant of the excitatory and inhibitory synaptic currents respectively. In this model $\tau_{\text{exc}} = \tau_{\text{inh}} = 2$ ms. $\hat{\delta}_{i,j}$ is the synaptic delay for the synapse connecting neurons i and j. All the delays are uniformly distributed between 0.1 ms and 1.0 ms. The time dependence of the synaptic efficacy $J_{i,j}$ is only due to short-term plasticity modulation and it is described in Equation (2.3), whereas synapses not modulated by the STP dynamics have fixed values of $J_{i,j}$. Since in this model, only the connections between excitatory neurons employ short-term plasticity, the connections with inhibitory neurons do not show a time-dependent synaptic efficacy. In addition, since the synaptic efficacies $J_{i,j}^{(\text{abs})}$ are expressed in

mV, a factor α is needed in order to be consistent with the units of Equation (7.5). This term derives the variation of current input needed to elicit a unit of variation of the postsynaptic potential, and has been derived in Section 1.4 (see Equation (1.19)).
The excitatory neurons are organized in $p = 5$ selective populations, each of which including a fixed fraction of neurons, and a non-selective population that includes the rest of the excitatory neurons of the network. Selective populations code for a certain item so that when an item has to be recalled in the model, the respective selective population is targeted by an additional stimulus. In the base model, the selective populations of excitatory neurons have no overlap, so a neuron can not belong to different selective populations. However, in an extension of the model, it is possible to simulate it with overlapping selective populations. In such a framework, the neurons belonging to each selective population are randomly chosen from the whole excitatory population, enabling the possibility of having neurons belonging to more than one selective population.
Regarding network connectivity, short-term plasticity is implemented in all the excitatory-to-excitatory connections using the same time constants in order to show synaptic facilitation. These connections are thus characterized by the STP variables x and u and a weight J, that represents the absolute synaptic efficacy. The weights of the connections within excitatory neurons belonging to the same selective population are assigned a potentiated value J_p, emulating the result of prior long-term Hebbian learning. On the other hand, connections between excitatory neurons belonging to different selective populations, or linking a selective population with the non-selective one, are set to a baseline value J_b. The rest of the excitatory-to-excitatory connections have the baseline synaptic efficacy except for the 10% of them, randomly chosen, that show the potentiated value in order to provide additional "noisy" internal input. This percentage is expressed in the model by a factor γ. While the excitatory-to-excitatory connections show STP dynamics, the other connections are static connections with hard-coded synaptic weights. The overall connectivity is structured so that each neuron of the network receives a fixed amount of connections from the network's populations, both excitatory and inhibitory, with a non-specific inhibitory connectivity. The possibility of having more than one synapse with the same two neurons is also enabled.
The next tables describe, following [100] guidelines, the network model introduced above, providing additional details on the network architecture and parameters.

Summary	
Populations	excitatory population $\mathcal{E}$, inhibitory population $\mathcal{I}$
Connectivity	sparse random connectivity
Neurons	leaky integrate-and-fire (LIF)
Synapses	linear input integration with exponential postsynaptic currents (PSCs), short-term synaptic plasticity (STP) for connections between excitatory neurons
Input	Gaussian white noise-like currents

Populations		
Name	**Elements**	**Size**
$\mathcal{E} \equiv \mathcal{E}_{\text{ns}} \cup \mathcal{E}_{\text{s}} \forall s \in \{1, \ldots, p\}$	LIF neurons	N_{E}
$\mathcal{E}_{\text{s}} \forall s \in \{1, \ldots, p\}$	LIF neurons	$N_{\text{E}} \times f$
$\mathcal{E}_{\text{ns}}$	LIF neurons	$N_{\text{E}} \times (1 - fp)$
$\mathcal{I}$	LIF neurons	N_{I}

Neuron	
Type	leaky integrate-and-fire (LIF) dynamics
Description	dynamics of membrane potential $V_i(t)$ and spiking activity of neuron $i \in \{1, \ldots, N\}$: • emission of kth ($k = 1, 2, \ldots$) spike of neuron i at time $t_k^{(i)}$ if $$V_i\left(t_k^{(i)}\right) \geq \theta$$ with spike threshold θ; • reset and refractoriness: $$\forall k,\ \forall t \in \left(t_k^{(i)},\ t_k^{(i)} + \tau_{\text{ref}}\right] : \quad V_i(t) = V_{\text{reset}}$$ with refractory period τ_{ref} and reset potential V_{reset}; • subthreshold dynamics of membrane potential $V_i(t)$: $$\forall k,\ \forall t \notin \left[t_k^{(i)},\ t_k^{(i)} + \tau_{\text{ref}}\right) : \quad \tau_{\text{m}} \frac{\mathrm{d}V_i(t)}{\mathrm{d}t} = \left[E_{\text{L}} - V_i(t)\right] + R_{\text{m}} I_i(t)$$ with membrane time constant τ_{m}, membrane resistance R_{m}, resting potential E_{L}, and total synaptic input current $I_i(t)$.

Table 7.1: Description of the network model (continues on next page).

Connectivity (part 1)		
Source	**Target**	**Pattern**
$\mathcal{E}_\mathrm{s} \forall s \in \{1,\ldots,p\}$	$\mathcal{E}_{\mathrm{s'}} \forall s' \equiv s \in \{1,\ldots,p\}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = f c_\mathrm{EEsp} N_\mathrm{E}$ $(\forall j \in \mathcal{E}_{\mathrm{s'}})$; • static synaptic weights J_p with STP modulation $(\forall j \in \mathcal{E}_{\mathrm{s'}})$; • uniformly distributed spike-transmission delays $\delta_{ij} \in [0.1, 1.0]$ ms $(\forall i \in \mathcal{E}_\mathrm{s}, j \in \mathcal{E}_{\mathrm{s'}})$.
$\mathcal{E}_\mathrm{s} \forall s \in \{1,\ldots,p\}$	$\mathcal{E}_{\mathrm{s'}} \forall s' \neq s \in \{1,\ldots,p\}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = f c_\mathrm{EE} N_\mathrm{E}$ $(\forall j \in \mathcal{E}_{\mathrm{s'}})$; • static synaptic weights J_b with STP modulation $(\forall j \in \mathcal{E}_{\mathrm{s'}})$; • uniformly distributed spike-transmission delays $\delta_{ij} \in [0.1, 1.0]$ ms $(\forall i \in \mathcal{E}_\mathrm{s}, j \in \mathcal{E}_{\mathrm{s'}})$.
$\mathcal{E}_\mathrm{s} \forall s \in \{1,\ldots,p\}$	$\mathcal{E}_\mathrm{ns}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = f c_\mathrm{EE} N_\mathrm{E}$ $(\forall j \in \mathcal{E}_\mathrm{ns})$; • static synaptic weights J_b with STP modulation $(\forall j \in \mathcal{E}_\mathrm{ns})$; • homogeneous spike-transmission delays $d_{ij} = d$ $(\forall i \in \mathcal{E}_\mathrm{s}, j \in \mathcal{E}_\mathrm{ns})$.
$\mathcal{E}_\mathrm{ns}$	$\mathcal{E}_\mathrm{ns} \vee \mathcal{E}_\mathrm{s} \forall s \in \{1,\ldots,p\}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = c_\mathrm{EE}(1 - fp) N_\mathrm{E}$ $(\forall j \in \mathcal{E}_\mathrm{s})$; • static synaptic weights J_b with STP modulation $(\forall j \in \mathcal{E}_\mathrm{s})$; • homogeneous spike-transmission delays $d_{ij} = d$ $(\forall i \in \mathcal{E}_\mathrm{ns}, j \in \mathcal{E}_\mathrm{s})$.

Table 7.1: Description of the network model (continues on next page).

Connectivity (part 2)		
Source	**Target**	**Pattern**
$\mathcal{E}_\mathrm{s} \forall s \in \{1,\ldots,p\}$	$\mathcal{I}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = c_\mathrm{IE} f N_\mathrm{E}$ ($\forall j \in \mathcal{I}$); • static synaptic weights J_IE ($\forall j \in \mathcal{I}$); • homogeneous spike-transmission delays $d_{ij} = d$ ($\forall i \in \mathcal{E}_\mathrm{s}, j \in \mathcal{I}$).
$\mathcal{E}_\mathrm{ns}$	$\mathcal{I}$	• random, independent; homogeneous in-degree $K_{\mathrm{E},j} = c_\mathrm{IE}(1 - fp) N_\mathrm{E}$ ($\forall j \in \mathcal{I}$); • static synaptic weights J_IE ($\forall j \in \mathcal{I}$); • homogeneous spike-transmission delays $d_{ij} = d$ ($\forall i \in \mathcal{E}_\mathrm{s}, j \in \mathcal{I}$).
$\mathcal{I}$	$\mathcal{E}$	• random, independent; homogeneous in-degree $K_{\mathrm{I},j} = c_\mathrm{EI} N_\mathrm{I}$ ($\forall j \in \mathcal{E}$); • static synaptic weights J_EI ($\forall j \in \mathcal{E}$); • homogeneous spike-transmission delays $d_{ij} = d$ ($\forall i \in \mathcal{I}, j \in \mathcal{E}$).
$\mathcal{I}$	$\mathcal{I}$	• random, independent; homogeneous in-degree $K_{\mathrm{I},j} = c_\mathrm{II} N_\mathrm{I}$ ($\forall j \in \mathcal{I}$); • static synaptic weights J_II ($\forall j \in \mathcal{E}$) • homogeneous spike-transmission delays $d_{ij} = d$ ($\forall i \in \mathcal{I}, j \in \mathcal{I}$).
self-connections (“autapses”) and multiple connections (“multapses”) allowed		

Table 7.1: Description of the network model (continues on next page).

Stimulus	
Type	white noise-like Gaussian current
Description	External current is given by a Gaussian white noise current, with mean μ_{X}, standard deviation σ_{X} and constant current during time intervals of length Δt_{ng}. Denoting the index of the time interval with k, for each interval the current is given by $\mu_{\mathrm{X}} + \sigma_{\mathrm{X}} G_k$, with G_k a random number extracted from a standard Gaussian distribution. $$I_{\mathrm{X},i}(t-\hat{\delta}_i) = \mu_{\mathrm{X}} + \sigma_{\mathrm{X}} G_k \text{ for } k\Delta t_{\mathrm{ng}} \leq (t-\hat{\delta}_i) \leq (k+1)\Delta t_{\mathrm{ng}})$$ External stimulation such as the item loading signal and the read-out signal is designed in the same way and modulated using a contrast factor $\mathcal{A}$ so that the total current injected to the targeted population has mean $\mathcal{A}\mu_{\mathrm{X}}$.
Synapse: transmission	
Type	current-based synapses with exponential postsynaptic currents (PSCs)
Description	• total synaptic input current of neuron i $$I_i(t) = I_i^{\mathrm{E}} + I_i^{\mathrm{I}} + I_{\mathrm{X},i}$$ • excitatory, inhibitory synaptic input currents $$\tau_{\mathrm{E}} \frac{dI_i^{\mathrm{E}}}{dt} = -I_i^{\mathrm{E}} + \sum_i \alpha J_{i,j}(t) \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j})$$ $$\tau_{\mathrm{I}} \frac{dI_j^{\mathrm{I}}}{dt} = -I_j^{\mathrm{I}} + \sum_i \alpha J_{i,j} \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j})$$ where $\delta(t - t_s^{(i)} - \hat{\delta}_{i,j})$ represents the spike train considering also the synaptic delay of the connections ($\hat{\delta}$). τ_{E} and τ_{I} are the excitatory and inhibitory time constants and α is the PSC amplitude (that gets the value of the synaptic weight in pA in order to elicit a given depolarization of the membrane potential); • PSC amplitude (synaptic weight) α (see Equation (1.19)).

Table 7.1: Description of the network model (continues on next page).

Synapse: plasticity	
Type	short-term synaptic plasticity (STP) for connections between excitatory neurons
Description	dynamics of synaptic weights $J_{ij}(t)\ \forall i \in \mathcal{E}, j \in \mathcal{E}$: $\forall J_{ij}$, given the absolute weight (J_b or J_s in the model, here $J^{(\text{abs})}$ for simplicity) : $$\frac{du_{i,j}}{dt} = -\frac{u_{i,j} - U}{\tau_\text{f}} + U(1 - u_{i,j}) \sum_s \delta(t - t_s^{(i)})$$ $$\frac{dx_{i,j}}{dt} = \frac{1 - x_{i,j}}{\tau_\text{d}} - u_{i,j}x_{i,j} \sum_s \delta(t - t_s^{(i)})$$ $$J_{i,j}(t) = J_{i,j}^{(abs)} u_{i,j}(t - \hat{\delta}_{i,j}) x_{i,j}(t - \hat{\delta}_{i,j})$$ where • $u(t)$ and $x(t)$ are the STP variables, representing the normalized concentration of calcium ions in the presynaptic terminal and the normalized amount of neurotransmitters ready for release, respectively; • τ_d and τ_f are the time constant for short-term depression and facilitation. In this model $\tau_\text{f} \gg \tau_\text{d}$; • U is related to the increase of calcium concentration due to the spike incoming in the presynaptic terminal; • $\hat{\delta}$ is the synaptic delay.

Initial conditions	
Type	STP parameters, membrane potential, input current
Description	• STP variables: $x_{i,j}(0) = 1.0$, $u_{i,j}(0) = U$; • membrane potential: $V_i = 0.0\,\text{mV} \quad \forall i \in \mathcal{E}, \mathcal{I}$; • input current $I_{\text{X},i} = 0.0\,\text{pA} \quad \forall i \in \mathcal{E}, \mathcal{I}$.

Table 7.1: Description of the network model.

Network and connectivity		
Name	**Value**	**Description**
f	0.1	fraction of neurons in a synaptic population
p	5	number of selective populations
c_{EEsp}	0.2	connectivity level for neurons belonging to the same selective population
c_{EE}	0.2	connectivity level for other E$\rightarrow$E connections
c_{IE}	0.2	connectivity level for E$\rightarrow$I connections
c_{EI}	0.2	connectivity level for I$\rightarrow$E connections
c_{II}	0.2	connectivity level for I$\rightarrow$I connections
N_{E}	8000	number of excitatory neurons
N_{I}	2000	number of inhibitory neurons
Neuron		
Name	**Value**	**Description**
V_{th}	$20\,\mathrm{mV}$	spike threshold
E_{L}	$0\,\mathrm{mV}$	resting potential
τ_{m}	$15\,\mathrm{ms}$	membrane time constant
C_{m}	$250\,\mathrm{pF}$	membrane capacitance
V_{reset}	$0\,\mathrm{mV}$	reset potential
τ_{ref}	$2\,\mathrm{ms}$	absolute refractory period
Synapse		
Name	**Value**	**Description**
J_{b}	0.10 mV	baseline weight (PSP amplitude) for excitatory to excitatory synapses
J_{p}	0.45 mV	potentiated weight of excitatory to excitatory synapses
J_{IE}	0.135 mV	weight of excitatory to inhibitory synapses
J_{EI}	0.25 mV	weight of inhibitory to excitatory synapses
J_{II}	0.2 mV	weight of inhibitory to inhibitory synapses
γ_0	0.0	fraction of potentiated synapses across different excitatory sub-populations
$\hat{\delta}$	$\mathcal{U}(0.1, 1.0)$ ms	synaptic delays uniformly distributed
U	0.19	baseline STP utilization factor
u	U	(initial) fraction of resources released for a spike
x	1.0	(initial) fraction of resources ready to be released
τ_{f}	$1500.0\,\mathrm{ms}$	facilitation time constant
τ_{d}	$200.0\,\mathrm{ms}$	depression time constant

Table 7.2: Model parameters (continues on next page).

Stimulus		
Name	**Value**	**Description**
$\eta_{\mathrm{X}}^{\mathrm{E}}$	variable	mean external current for excitatory neurons
$\eta_{\mathrm{X}}^{\mathrm{I}}$	20.5 mV	mean external current for inhibitory neurons
$\Sigma_{\mathrm{X}}^{\mathrm{E}}$	1.0 mV	standard deviation of external current for excitatory neurons
$\Sigma_{\mathrm{X}}^{\mathrm{I}}$	1.0 mV	standard deviation of external current for inhibitory neurons
$\mathcal{T}_{\mathrm{cue}}$	350.0 ms	item loading signal duration
$\mathcal{A}_{\mathrm{cue}}$	1.15	item loading signal contrast factor
$\mathcal{T}_{\mathrm{reac}}$	250.0 ms	readout signal duration
$\mathcal{A}_{\mathrm{reac}}$	1.05	readout signal contrast factor
$\mathcal{T}_{\mathrm{reac}}^{\mathrm{period}}$	100.0 ms	periodic readout signal duration
$\mathcal{A}_{\mathrm{reac}}^{\mathrm{period}}$	1.075	periodic readout signal contrast factor

Table 7.2: Model parameters.

Table 7.2 illustrates the synaptic parameters used in this spiking network model to build the external stimuli. It should be noted that the external stimulus, modeled as a Gaussian white noise current, shows mean η_{ext} and standard deviation Σ_{ext} expressed in mV and not in pA. In [3] we derived[1] that, given the mean μ and standard deviation σ of a piecewise constant current injected to a neuron, with standard deviation changing at fixed intervals of length Δt_{ng}, the membrane potential will show a mean η and the standard deviation Σ described as follows:

$$\begin{aligned}\eta &= \mu\frac{C_m}{\tau_m}\\ \Sigma &= \sqrt{\frac{2}{\tau_{\mathrm{m}}\Delta t_{\mathrm{ng}}}}C_{\mathrm{m}}\sigma\end{aligned} \tag{7.6}$$

7.2.2 Results of the simulation

In this section, we present the results of the spiking network simulations performed using the NEST simulator (version 3.1) [134].

The network is composed of 8000 excitatory and 2000 inhibitory LIF neurons with exponential postsynaptic currents, whose dynamics are described by Equations (7.3), (7.4) and (7.5) (see also Equations 1, 2, 4 and 5 in [135] and Equation 3 in [136]). The neuron model differs from the one employed in the original work, as [23] employs a LIF neuron model with instantaneous rise and decay times for postsynaptic currents. As shown in Tables 7.1 and 7.2, the excitatory

[1]We followed the derivation of the NEST Documentation, please see `https://nest-simulator.readthedocs.io/en/v3.1/model_details/noise_generator.html?highlight=noise_generator#Hans-Ekkehard-Plesser,-2015-06-25`

population is further divided into $p = 5$ selective populations of 800 neurons each and a non-selective population that includes the rest of the excitatory neurons. All excitatory-to-excitatory connections follow a STP dynamics whereas the rest of the connections have fixed synaptic efficacies.
The simulations are performed using a time step of $0.05\,\mathrm{ms}$, with the system of Equation (7.3) and Equation (7.5) integrated following the exact integration scheme of [13] and assuming that the external current $I_{\mathrm{ext},j}$ is piecewise constant over time intervals of width Δt_{ng}. This is an additional difference with respect to [23], in which both Equation (7.3) describing the neuron sub-threshold dynamics and Equation (2.2) describing the STP mechanism are integrated using the Euler scheme. Indeed, the implementation here adopted is also different from the NEST default implementation[2]. Further details on these differences are described in Appendix C. In this model, the STP timescales are set so that the network shows synaptic facilitation, in fact, $\tau_d = 200\,\mathrm{ms}$ and $\tau_f = 1500$ ms in agreement with the parameters chosen in [23].
All the simulations begin with a time period of $3000\,\mathrm{ms}$ in which only the background input is injected into the whole network in order to allow the network to enter its baseline state illustrating spontaneous activity. This stimulation, as well as all the other external signals, is created using the NEST `noise_generator`, which injects a Gaussian white noise current as described in Equation (7.4). The background input targets both excitatory and inhibitory neurons with different mean current values. Later in this section, it will be shown how network behavior can be modulated by changing excitatory background activity.
After the network reaches its spontaneous activity, an additional current, designed as a Gaussian white noise current which is added to the background input, is injected only into a selective population for a time $\mathcal{T}_{\mathrm{cue}} = 350\,\mathrm{ms}$. As a result, an item is loaded into the model. This signal, called item loading, increases the synaptic activity of the target population and thus permits a temporary strengthening of synaptic efficacies by changing the STP variables u and x across the connections of neurons belonging to the target population. Thus, even after the end of the item loading signal, the reloaded memory can be synaptically facilitated especially because of the slow decaying dynamics of the variable u. Figure 7.2 shows the raster plot for two selective populations when an item is loaded in one of them, having different values of background input. The values of the mean excitatory background current (i.e., $\eta_{\mathrm{X^E}}$) used in Figure 7.2 are $22.7\,\mathrm{mV}$, $23.7\,\mathrm{mV}$ and $24.1\,\mathrm{mV}$ respectively for the panels A, B and C.
As can be seen, the memory-specific response of the network depends on the background activity level of the excitatory neurons. This figure shows the raster plot of two selective populations, one targeted by the additional current which re-loads the item and a non-targeted one, together with the STP variables x and u averaged over the connections outbound from the neurons of the targeted selective population. In Figure 7.2A, to reactivate a memory, a supplemental external signal targeting the entire excitatory population is given. Although this external signal is nonspecific, only the population in which the memory was previously restored

[2]The NEST version provided with the synapse model used here can be found at `https://github.com/gmtiddia/nest-simulator-3.1`.

responds with the emission of a single synchronized activity, called *population spike*. The network can also autonomously exhibit a memory-specific spiking activity when a higher excitatory background current is injected (see Figure 7.2B and 7.2C).

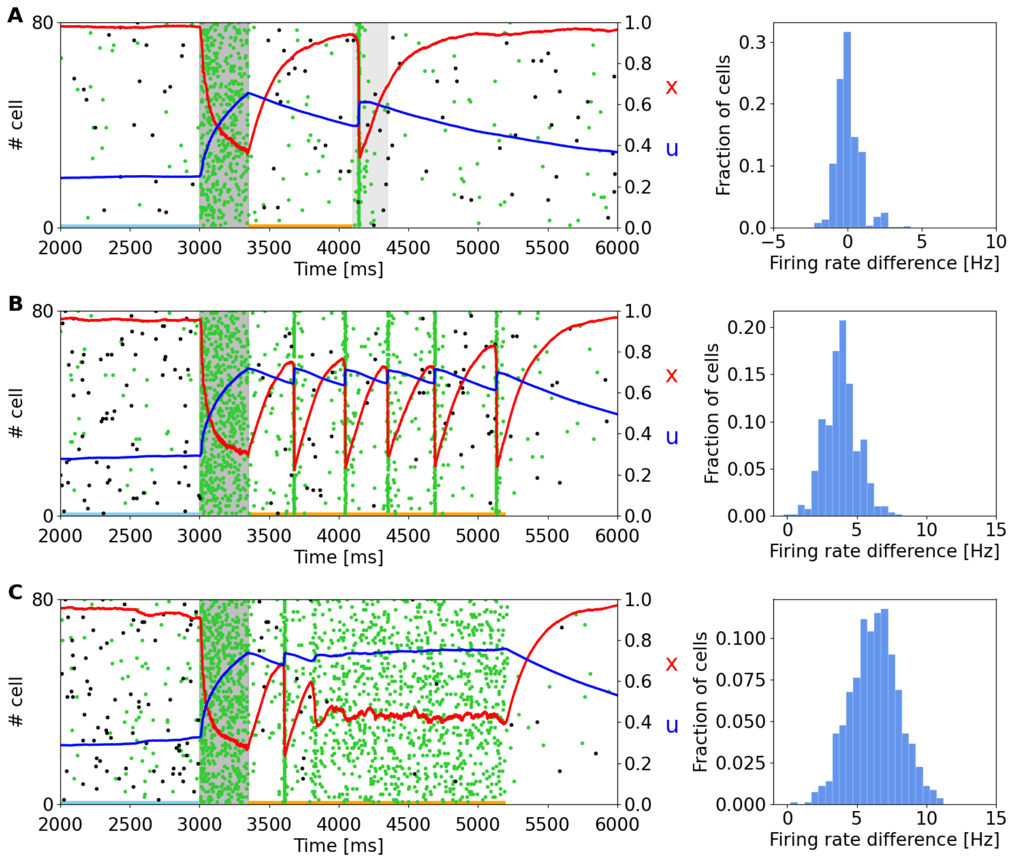


Figure 7.2: Raster plots of a subset of neurons of a targeted (green) and non-targeted (black) selective population for different values of excitatory background input. Averaged STP variables (x in red and u in blue) of the synaptic connections belonging to the target population are also shown. **(Left) (A)** The network needs the injection of a nonspecific input (lighter gray shading) to show a memory-specific response. **(B)** Increasing the background input, the network autonomously reactivates the memory by showing periodic synchronized events. **(C)** With a further increase in the background stimulus, the network shows an asynchronous enhanced spiking activity of the re-loaded memory. In (B) and (C), the network returns to its spontaneous state when background input diminishes. **(Right)** Histograms representing the difference in firing rate between the delay period (orange line at the bottom of the left panels) and the spontaneous state for the selective population targeted by the item loading signal (sky blue line at the bottom of the panels). In (A) the delay period is defined as the time between the end of the item loading stimulus and the beginning of the nonspecific external input, whereas in (B) and (C) is the time between the end of the item loading and the decreasing of the external background input (here at $5.2\,\mathrm{s}$). Figure from [3].

In Figure 7.2B the targeted selective population shows an autonomous and synchronous emission of population spikes. It should be noted that after each population spike the STP variable u increases and returns to similar values reached at the end of the item loading signal injection, interrupting the exponential decrease due to the calcium removal mechanism and thus enabling a new population spike to emerge. This behavior, together with the fast exponential growth of available resources described by the variable x, leads to a new stable state for the network together with the one representing spontaneous activity. To interrupt the network persistent activity we set the excitatory background current to the value of Figure 7.2A. In Figure 7.2C the background input is further increased, and the network spontaneously shows an asynchronous higher rate activity. In this state, the memory is maintained in both spiking and synaptic form since the STP parameters reach stable values during the high activity state followed by a population spike. As in the previously described state, the network could pass from the memory-specific activity state to the spontaneous state by diminishing the background input. Indeed, without the diminishing of the background input, the network would continue to behave showing the asynchronous higher rate activity or the synchronous emission of population spikes.
Moreover, we quantitatively estimate the difference in firing rate for the targeted selective population between the delay period and the spontaneous activity state. The difference in firing rate for a neuron population is obtained by measuring the spike-count rate for each neuron of the population in two time intervals. Naming r_s the firing rate measured during the spontaneous activity state and r_d the firing rate measured during the delay period, the firing rate difference for a neuron i of the population is

$$\Delta r^{(i)} = r_d^{(i)} - r_s^{(i)} = \frac{N_d^{(i)}}{\Delta t_d} - \frac{N_s^{(i)}}{\Delta t_s} \tag{7.7}$$

where $N^{(i)}$ is the number of spikes emitted by the neuron i in a certain time interval Δt. Those values are obtained for each neuron of the targeted selective population and are collected in the histograms on the right side of Figure 7.2. In panel 7.2A the delay period is defined as the time between the end of item loading and the beginning of the nonspecific signal (i.e., the stimulus targeting the whole excitatory population represented with the lighter gray shading), whereas in the other panels is identified between the end of the item loading and the decreasing of the external input (happening at $5.2\,\mathrm{s}$ for both panels). The time intervals related to the spontaneous activity and the delay period are indicated with horizontal lines (sky blue and orange respectively) in the left panels of Figure 7.2. It is possible to notice that in panel 7.2A there is no significant difference in firing rate, and a relevant part of the network shows a decrease in firing rate during the delay period. In panels 7.2B and 7.2C is observed an increase in firing rate of about $4\,\mathrm{Hz}$ and $7\,\mathrm{Hz}$ respectively, with an average baseline firing rate of about $0.7\,\mathrm{Hz}$. These changes in firing rate are lower with respect to the ones shown in network models relying only on persistent activity to show working memory behavior such as [94] and they are in agreement with experimental measures on single-cell activity during delay period [137], according to which the changes in firing rate are mostly below $5\,\mathrm{Hz}$.

In comparison with the work of [23], the network simulated with NEST shows qualitatively similar results, with comparable behavior when modulating the background input targeting the excitatory neurons. However, we noticed some relevant differences with respect to the original work. For instance, on the left side of Figure 7.2B, it is possible to see that the time interval between adjacent population spikes is around $300\,\mathrm{ms}$, whereas in [23] this value is closer to $200\,\mathrm{ms}$, same order of τ_d. Further, while the behavior of the variable u is mostly comparable to the one shown in the original article, the behavior of the variable x shows a considerably higher drop of the averaged variable in correspondence to a population spike. This pronounced drop in the value of x is probably the reason for the difference in the time interval between two population spikes previously mentioned.
Since one of the main features of a working memory network is the holding of multiple items, we load two items into two different selective populations at different times to analyze the behavior of the STP variables of the targeted populations and the capacity of such a network of maintaining multiple items. Figure 7.3 shows a subset of two selective populations targeted by the item loading signal in the single stable state regime (Figure 7.3A) and in the regime showing synchronous and autonomous reactivation (Figure 7.3B), obtained using the same values of background input used in Figure 7.2A and Figure 7.2B respectively.
Moreover, both simulations are provided with noise, injected into a fraction of all the excitatory neurons in order to check the robustness of the network state. The noise signal is designed as the item loading one but targets the 15% of the excitatory neurons randomly. In Figure 7.3A the reactivation of the selective populations is enabled by a periodic nonspecific input (with a period of $300\,\mathrm{ms}$). It can be noticed that in this framework the two targeted selective populations do not emit the population spike during the same periodic readout signal, but they alternate in order to reach suitable values of STP variables to enable the emission of a population spike in the following readout signal. This peculiar behavior can be seen also when the network autonomously shows the synchronous spiking activity (Figure 7.3B). In this case, similarly to Figure 7.3A, the synchronous activity of the targeted selective populations is alternated, increasing the average value of x for a population when the other one is emitting the population spike. However, in Figure 7.3B, this mechanism is completely autonomous. In both the network states the slow dynamics of u have a key role in holding the information, in particular when another selective population shows a higher spiking activity. In addition, it can be observed that the higher spiking activity of a selective population inhibits the other populations. This is due to the network's connectivity which enables a winner-take-all mechanism, i.e. the competition between different populations through a mechanism of global inhibition, as previously described. For this reason, it is not possible to correctly load multiple items at the same time and it is not possible to have population spikes from different selective populations at the same time. As can be seen in Figure 7.3A, even if the readout signal targets all the selective populations, only the targeted selective population which has the highest STP-modulated synaptic efficacy is capable of emitting a population spike, inhibiting the excitatory neurons of the competing selecting populations.

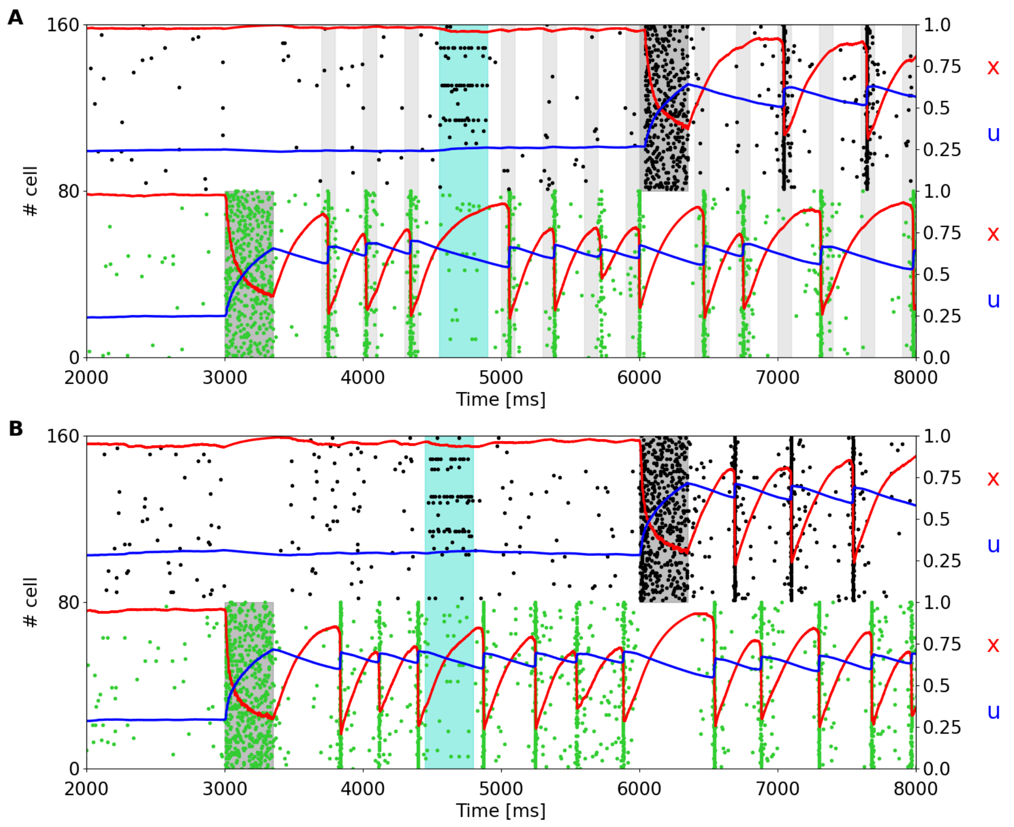


Figure 7.3: Raster plots of a subset of neurons of two targeted selective populations when two items are loaded into the network (gray shading). An additional noise (cyan shading), which targets 15% of the excitatory neurons, is injected to test the network's robustness. **(A)** Network showing a single stable activity state injected with a periodic readout signal targeting all the excitatory neurons. After the second memory is loaded into the network, the populations show alternating population spikes. **(B)** Network in the bi-stable regime showing synchronous spiking activity. Here the network does not receive the periodic input since the synchronous activity autonomously shows up after the item loading. After the second memory is loaded the population spikes of the stimulated selective populations alternates, refreshing the synaptic variables in order to maintain the synchronous spiking activity. Figure from [3].

The behavior of the network in Figure 7.3 is comparable with respect to the results shown in [23]. The main differences that emerge are related, as stated before, to the dynamics of the STP variable x which shows a more pronounced drop when neurons show synchronous firing activity. We slightly increased the time interval between two consecutive stimulations in Figure 7.3A, from $250\,\mathrm{ms}$ to $300\,\mathrm{ms}$, to make the STP variable x recover enough to enable the synchronous activity response as in Figure 7.2A. Shorter time intervals between subsequent readout signals could result in stimulation that leads to a population spike right after the end of the stimulus.

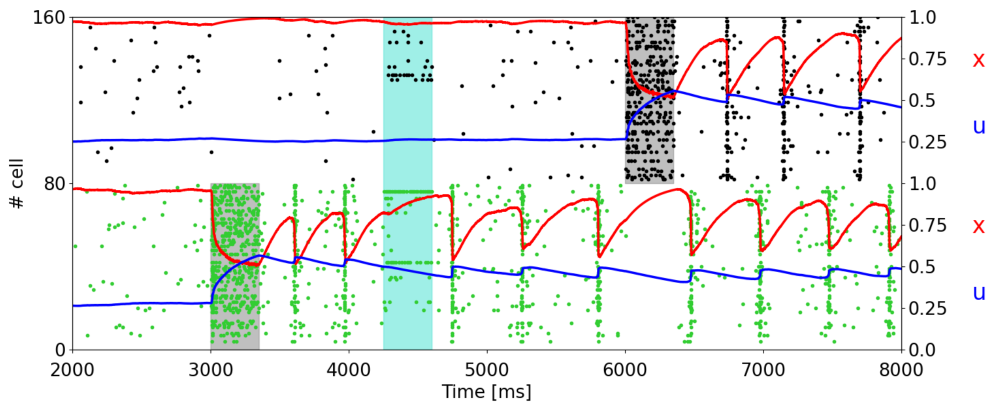


Figure 7.4: Raster plot for a simulation with overlapping populations. Here only a subset of the two targeted selective populations is shown. An additional noise (cyan shading) is also injected. The network is in the bi-stable regime showing synchronous spiking activity. Figure from [3].

Hitherto we presented the results of the simulations for the model with non-overlapping populations, ergo an excitatory neuron can only belong to a selective population at most. To verify the network's behavior in more realistic conditions we also performed simulations in which there is the possibility of having overlaps between the selective populations. Figure 7.4 shows the raster plot of a simulation with the same parameters used in Figure 7.3B but with overlapping populations. Here, the population spikes are less synchronized and not all the neurons belonging to the selective population emit a spike during the synchronous spiking activity. For this reason, the STP variable x drops caused by the population spikes are less pronounced. To obtain a qualitatively similar behavior with respect to the network with non-overlapping populations the value of the potentiated synaptic efficacy J_p has been slightly increased to $0.49\,\mathrm{mV}$.

Study on network's memory capacity

In [39] (see also [138] for a similar derivation) is provided an analytical expression to estimate the maximal number of items that can be maintained in a working memory model sustained by STP. This number is determined by the ratio between T_{max}, i.e. the maximal period of the limit cycle of the network, and t_s, i.e. the time interval between two successive population spikes. Indeed, the maximal period of the limit cycle is only dependent on STP parameters and can be expressed by [39]

$$T_{\mathrm{max}} \simeq \tau_d \ln \frac{\tau_f/\tau_d}{1-U} \tag{7.8}$$

This equation is then validated in [39] by performing simulations using a spiking neural network with a similar structure to the one shown in [23], thus we decided to perform simulations similar to the one of [39] using our implementation and test the validity of Equation (7.8). To this end, we perform simulations with additional item loading signals targeting other selective populations for the network state showing synchronous spiking activity (same value of background input as in

Figure 7.2B). We noticed that the network was able, using these parameters, to keep up to three items. The raster plot, together with the averaged STP variables for the three targeted selective populations, is depicted in Figure 7.5.

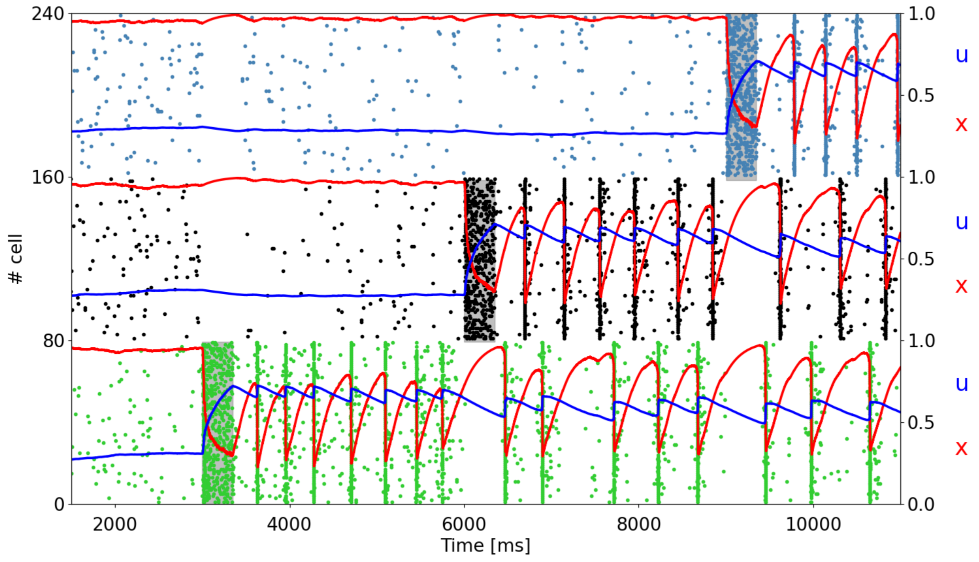


Figure 7.5: Raster plot of a subset of neurons of three targeted selective populations when three items are loaded into the network. The network is in the bi-stable regime showing synchronous spiking activity. Item stimuli are loaded into the network at 3000, 6000, and 9000 ms. Figure from [3].

As shown in Figure 7.5, the network is able to maintain three selective populations in the persistent activity state similar to Figure 7.3B so that, when all the items are loaded, population spikes alternate within the population keeping appropriate values for the STP variable. Moreover, it should be noted that each selective population in the synchronous spiking activity regime diminishes the average value for the STP variable u when other items are loaded into the network. This behavior is clearly visible for the first selective population of Figure 7.5. Indeed, this is due to the increased distance between population spikes related to the activity of the other targeted selective populations. For an increasing number of items loaded, this can lead to a loss of synchronicity, since persistent activity state needs relatively high values of u to be maintained. In this case, only a subset of the targeted selective population are able to keep showing the memory-specific response, with other populations that interrupt the emission of the population spikes and return in the low activity regime.
Using the parameters employed to produce Figure 7.5, in which up to three items can be stored, $T_{\text{max}} \simeq 445$ ms, whereas the time separation between the population spike, once the three items have been loaded into the network, is approximately $t_s \simeq 160$ ms, with the ratio between these times being

$$N_c \approx T_{\text{max}}/t_s \simeq 2.8$$

not far from the number of items stored at the same time, confirming the generality of the analytical estimation proposed in [39].

We also performed a similar calculation for a network able to store more items. To do so, an increase in the value of τ_f is needed to enhance facilitation, thus enabling the upkeep of more items. Moreover, we decided to increase the network size. The default network model has 10000 LIF neurons and has a total of $p = 5$ selective populations. Since in [39] a network of 20000 LIF neurons and ten selective populations is described, we decided to use similar parameters to perform simulations with a larger network having a higher number of selective populations. To do so, we chose $f = 0.4$, $p = 10$, $N_{\mathrm{E}} = 16000$, $N_{\mathrm{I}} = 4000$, $J_b = 0.05$ (as in [39]). A simulation with a single item loaded was performed to tune the external background input targeting the excitatory neurons in order to show synchronous spiking activity. We found that $\mu_{\text{ext}} = 26.5\,\text{mV}$ was able to show the desired behavior. We simulated the network using different values of τ_f to notice the change in the number of items that can be simultaneously maintained, and we observed a similar behavior discussed with the smaller network, so a higher value of τ_f is needed to keep a larger number of items in memory. In particular, we observed that with $\tau_f = 4000\,\text{ms}$, $\tau_d = 250\,\text{ms}$ seven memories can be loaded into the network at the same time. The raster plot of the selective populations in this configuration of the model is shown in Figure 7.6.

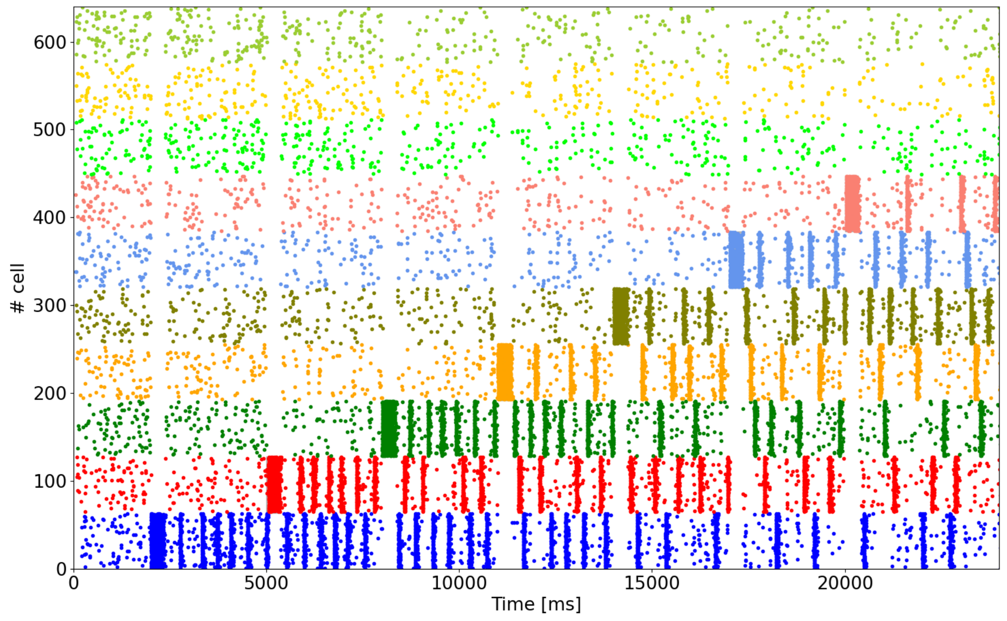


Figure 7.6: Raster plot of a subset of the ten selective populations of the model. An item loading signal is injected every $3000\,\text{ms}$ for each of the first seven selective populations starting from $2000\,\text{ms}$. The network, simulated using $\tau_f = 4000\,\text{ms}$, $\tau_d = 250\,\text{ms}$, is able to maintain seven targeted selective populations in the synchronous activity state. Figure from [3].

Regarding the theoretical estimation of the working memory capacity, using the parameters employed to obtain Figure 7.6, we estimate that the maximal period of the limit cycle of the network is $T_{max} \simeq 745\,\text{ms}$, whereas the time between two successive population spikes after having loaded all the seven items is $t_s = 140 \pm 40\,\mathrm{ms}$. Those values lead to an analytical estimation of working memory capacity between 4.1 and 7.5, with the average estimation being equal to 5.3, which is slightly smaller than the effective number of items kept in memory. However, it is still compatible with it considering the large uncertainty in this estimation.

7.3 Discussion of the results

In this chapter we have shown the results of the working memory spiking network model proposed by [23] reproduced in [3] using NEST and following the network description and the parameters shown in the original work. However, we have not limited to a reproduction of the model, and we explore the flexibility of the STP-driven mechanism through several tests.
Indeed, the spiking network model proposed here has some differences with respect to the original one of [23]. First, we employed a LIF neuron model with exponential postsynaptic currents, whereas the original one used a LIF model with instantaneous rise and decay times for postsynaptic currents. Furthermore, the neuron model is integrated following the exact integration method of [13], with synaptic variables for the neuron i synapses updated when the neuron i emits a spike. The implementation of the STP dynamics follows Equations (2.2) and (C.3). The implementation of the STP dynamics further differs with respect to the original work. In fact, in [23] and [39] the absolute synaptic efficacy is modulated using the values of the variables u and x immediately before the emission of the spikes. As described in Equation (C.3), the implementation used in this work considers the value of the variable x immediately before the spike emission, but with the variable u updated at the time of the emission of the spike, in agreement with [18]. This change in the implementation leads to higher modulated synaptic efficacies for the implementation employed in this work (see also [139] for a comparison of the two different implementations in a network of QIF neurons), and thus can be responsible for the more pronounced drop of the variable x noticed in the spiking model presented in this work. Despite these differences, we were able to obtain a similar behavior with respect to the original model by slightly adjusting some parameters. However, other parameters were missing, like the integration time step, which we set at $0.05\,\mathrm{ms}$, verifying that lower or higher time steps do not entail significant changes in the network behavior. Moreover, the connection scheme in the NEST simulator opens the possibility of having multiple connections with the same two neurons or self-connections. We thus checked that enabling or disabling this possibility does not change significantly the spiking activity, maintaining the same qualitative results as the one shown in this chapter. Furthermore, we also performed simulations with the same neuron model integrated with a different integration scheme with respect to the exact integration method of [13]. Specifically, we employed the stochastic Runge-Kutta method, more suitable in the presence of noise signals modeled as the background input employed in this network. We found that the results of the simulations are comparable with respect to the one presented here. The results of these studies are shown in Appendix D.
Figure 7.2, showing the raster plot of a selective population targeted by an item

loading signal and a non-targeted population, exhibits comparable results with respect to the original work, showing that the network can operate at relatively low firing rate and with modest (or even negligible) increase in firing rate during the delay period. These results are consistent with several experimental observations. For instance, [140, 141] show that, during the delay period, the information held in memory can be reactivated by a non-specific stimulus (as in Figure 7.2A). This result is also shown by [142], in which transcranial magnetic stimulation produced a brief reactivation of the held item. Moreover, the silent dynamics can lead to interference between information from different trials [143], and the relation between STP dynamics and the so-called serial effects in WM tasks has recently been explored in [144] and [145]. Furthermore, the firing rate changes between the spontaneous state and the delay period shown in the right panels of Figure 7.2 are in agreement with single-cell firing rates, which is mostly below 5 Hz and only rarely can reach values greater than 10 Hz [137]. Indeed, since a higher spiking activity would be more metabolically demanding, this behavior makes the model energetically efficient underlining the importance of activity-silent dynamics during working memory tasks, as suggested in [132] and [133].
On the other hand, this model has some limitations, the main one being that it assumes a prior long-term Hebbian learning. The way items are encoded in selective populations is extremely simplified, as all the connections within the same population have equal synaptic strength. Furthermore, this value remains constant during the simulation. A more realistic model would be a combination of long-term and short-term plasticity, enabling the learning of new items.
In conclusion, to provide a qualitative comparison of the results, this chapter shows simulation protocols and raster plots similar to the ones of the original work. Indeed, the original work does not provide the code or the data of the spiking activity of the network, thus only a qualitative comparison was possible. The work of [3] has reproduced the results, with the aim of providing the simulation code and the data needed for further studies oriented towards a better understanding of working memory mechanisms and the link between short-term synaptic plasticity and long-term cognitive processes such as learning[3].
Indeed, more studies on this model have been performed during my research stay at the Institute of Neuroscience and Medicine (INM-6) of the Jülich Research Center, in the group of Computational Neurophysics under the supervision of Dr. Tom Tetzlaff. These studies are focused on a direct comparison based on specific test protocols between working memory mechanisms sustained by attractor network dynamics or synaptic mechanisms such as STP. These tests are oriented towards the applicability of such mechanisms in hierarchical network models performing sequence learning. In [146] it is shown a spiking model able to reproduce the temporal-memory part of the Hierarchical Temporal Memory (HTM) algorithm of [147], which is able to learn sequences of inputs, making predictions and autonomously replay the sequence. In order for this model to assume a hierarchical structure and be able to encode sequences of sequences, a further mechanism is needed to keep a sequence in memory for a longer time. Thus, I worked on the

[3]The implementation of the model and the analysis code can be found at `https://github.com/gmtiddia/working_memory_spiking_network`.

possible implementation of the STP-driven working memory mechanism in such a model. However, such a project is still ongoing with some preliminary results, so I decided not to report them in the main body of this thesis. Some of the preliminary results are reported in Appendix D.

Chapter 8

Learning through structural plasticity: a theoretical framework

Summary

This chapter presents the framework for learning with structural plasticity described also in [5]. The chapter focuses in particular on the description of the theoretical framework and all the options that can be chosen to include some important features. In the first part of the chapter, a feed-forward network model is discussed, with the last part focusing on an extension of the latter provided with recurrent connections and additional neuron populations.

8.1 Model introduction

As described in Section 2.3, it is commonly believed that learning is influenced not only by a change in the synaptic efficacy of existing synapses but also by the change in the connection structure of neural circuits have a relevant role. In recent years, several computational models for structural plasticity were developed showing intriguing results regarding the relation of this mechanism in learning and other cognitive processes.
Here we propose a novel model for learning through a mechanism of structural plasticity. Since the biochemical and biophysical mechanisms underlying structural plasticity are extremely complex and only partially understood to date, this model exploits a relatively simple phenomenological model, including both the activity-driven and the homeostatic contributions according to what is discussed in Section 2.3. Despite the lower complexity, this model accounts for the effects of structural plasticity in terms of the consolidation of synaptic connections between neurons with a high activity correlation as well as those of pruning and rewiring the connections for which this correlation is lower. This approach is also justified by the requirement for a simple and effective computational model suitable for simulating networks with a relatively large number of neurons and connections and for representing learning processes with sizable numbers of training and vali-

dation patterns.
This model serves as the foundation for the creation of a mean-field-based theoretical framework for learning through synaptic plasticity capable of accounting for a variety of biological network properties. This framework is used in a training and validation procedure to characterize the learning and memory capacity of plastic neuronal networks as the number of training patterns and other model parameters vary. The proposed approach is capable of accounting for different probabilistic connection rules, firing rate probability distributions, presence of noise in stimuli, thus providing a general framework to study the impact of structural plasticity in learning on large-scale neuronal network models. The next section introduces the theoretical framework and how we implemented structural plasticity.

8.2 Model overview

Here we present the general model of two neuron populations connected feed-forwardly, and then the theoretical framework is divided into two possible approaches: a simple, discrete rate model in which neurons can only assume two possible firing rate values and a more realistic continuous rate model in which firing rates follow a continuous probability distribution. The following framework has been discussed in [5], and the following description entirely refers to this work.
The neuronal network model consists of two neuron populations, $\mathcal{P}_1$ and $\mathcal{P}_2$, of 10^5 neurons each. This number corresponds roughly to the number of neurons in a 1 mm^2 section of the cerebral cortex, considering its cortical thickness. The exchange of information between the two populations takes place through the connections from the population $\mathcal{P}_1$ to the population $\mathcal{P}_2$ (i.e. in a feed-forward fashion), which in the model are on average 5×10^3 per neuron of $\mathcal{P}_2$, for a total of 5×10^8 connections. Each connection has an initial synaptic weight of $\mathcal{W}_\text{b}$. The first population ($\mathcal{P}_1$) receives an input stimulus (e.g., visual) mimicked by the activation of the neurons with a firing rate pattern associated with it.
During the training stage, in addition to the signal from $\mathcal{P}_1$, the second population receives another stimulus (e.g., auditory), that we identify as a *contextual stimulus*. Figure 8.1 depicts a simple scheme of the network. Structural plasticity is modeled following the categories described in Section 2.3 and [47], i.e., activity-dependent and homeostatic. The firing rate patterns of the two populations have a role in the activity-dependent structural plasticity. The consolidation of a synaptic connection occurs when the firing rates of both the presynaptic and the postsynaptic neurons are concurrently above a certain threshold. The synaptic weight of a consolidated connection increases from $\mathcal{W}_\text{b}$ to a value $\mathcal{W}_\text{c} > \mathcal{W}_\text{b}$. This is an effect of LTP, according to which we have the formation of new synapses, but also the growth of dendritic spines that increases synaptic efficacy of already existing synapses. Thus, the mechanism of consolidation proposed here embraces both structural and functional synaptic modifications, both mediated by LTP [148]. Moreover, consolidated connections not only show an increased connection weight, but the connection is also prevented from being pruned in further steps of the simulation. We flank this mechanism with synaptic rewiring, which handles the redistribution

of the connections during the simulation. Indeed, this is a relevant mechanism, which relocates connections that have not been consolidated yet, mimicking the mechanism of connection pruning together with the creation of new connections handled by homeostatic structural plasticity. The mechanism of synaptic rewiring will be discussed in detail later in this chapter. Once a synaptic connection has been consolidated, it will maintain the synaptic weight $\mathcal{W}_c$, without the possibility of returning to the initial state $\mathcal{W}_b$. This approach is a computationally-effective way of representing the biological structural changes that make a consolidated connection strong and durable.
To model the injection of input and contextual stimuli, neurons of the two populations show independent firing-rate patterns randomly generated from predefined firing rate probability distributions, which can be either discrete or continuous. The training process is performed using $\mathcal{T}$ independent input patterns, together with the corresponding contextual stimuli, evolving the dynamics in discrete steps in which single input patterns are provided. A diagram of the training and validation processes is shown in Figure 8.1.

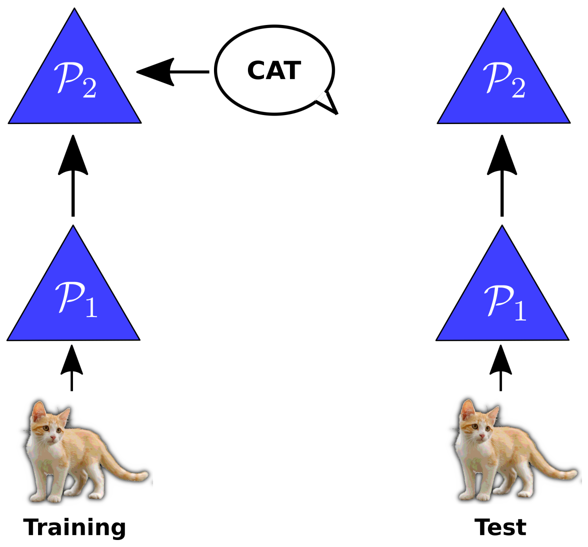


Figure 8.1: Schematic representation of the network model through a block diagram. During training (left), a visual stimulus is injected into $\mathcal{P}_1$, and an auditory stimulus is injected into $\mathcal{P}_2$ as contextual stimulus. In the test phase (right) one of the visual patterns is injected without the corresponding contextual stimulus. The cat image is adapted from Golden tabby and white kitten by Marie-Lan Nguyen / Wikimedia Commons / CC-BY 2.5. Figure from [5].

During training, when both input and contextual stimulus are used, a fraction of the neurons in the population $\mathcal{P}_2$ will assume a rate above the threshold. These neurons, called *coding*, or *selective* neurons, play a vital role in input coding. The existence of neurons showing selective firing rates in response to specific stimuli is largely confirmed by experimental results. The average input signal to these neurons will be called $\langle \mathcal{S}_2 \rangle$. The non-selective neurons of $\mathcal{P}_2$ will instead be called *non-coding* or *background* neurons, and their average input signal will be indicated with $\langle \mathcal{S}_b \rangle$. The proposed model accounts for the ability of the network to learn

the association between input patterns and the corresponding contextual stimuli by measuring the input signals incoming to *coding* or *non-coding* neurons of $\mathcal{P}_2$. Indeed, the output signals of $\mathcal{P}_2$ neurons can be calculated, as will be seen in Section 8.3.
We may distinguish two different types of models based on the potential values that the neuron firing rate can assume:

- **Discrete model**, in which neurons rate can assume only two discrete values ν_ℓ (low rate) and ν_h (high rate);

- **Continuous model**, in which neurons rate follow a continuous probability distribution (here lognormal).

In the next section, we will derive the mean-field equations for these two models, which are summarized in the following tables.

Summary		
Populations	$\mathcal{P}_1$, $\mathcal{P}_2$	
Connectivity	sparse random connectivity	
Neurons	firing-rate-based models of point-like neurons	
Synapses	structural plasticity	
Input	firing rate pattern extracted from a discrete or continuous probability distribution	
Populations		
Name	**Elements**	**Size**
$\mathcal{P}_1$	point-like neurons	$\mathcal{N}_1$
$\mathcal{P}_2$	point-like neurons	$\mathcal{N}_2$
Neuron		
Type	firing-rate-based neuron model	
Firing rate distribution	• discrete model: the firing rates can assume only two discrete values ν_h (high rate) and ν_ℓ (low rate); • continuous model: the firing rates can assume values from a continuous probability distribution (here lognormal).	

Table 8.1: Description of the network model (continues on next page).

Synapse	
Type	structural plasticity
Description	initial synaptic weight are set to $\mathcal{W}_\mathrm{b}$ for all the instantiated connections; when a training pattern is used, considering a connection between a $\mathcal{P}_1$ neuron i and a $\mathcal{P}_2$ neuron j: • (discrete model) $\mathcal{W}_\mathrm{b} \rightarrow \mathcal{W}_\mathrm{c}$ if $\nu_\mathrm{i} = \nu_\mathrm{j} = \nu_\mathrm{h}$; • (continuous model) $\mathcal{W}_\mathrm{b} \rightarrow \mathcal{W}_\mathrm{c}$ if $\nu_\mathrm{i} > \nu_{\mathrm{t},1}$ and $\nu_\mathrm{j} > \nu_{\mathrm{t},2}$. Once a connection is consolidated, it cannot return to the initial weight.

Connectivity		
Source	**Target**	**Pattern**
$\mathcal{P}_1$	$\mathcal{P}_2$	• random, independent; in-degrees can be homogeneous, with a fixed number of $\mathcal{C}$ connections per neuron of $\mathcal{P}_2$, or extracted using a Poisson distribution of mean $\mathcal{C}$; • synaptic weights are $\mathcal{W}_\mathrm{b}$ for unconsolidated connections and $\mathcal{W}_\mathrm{c}$ for consolidated ones, with $\mathcal{W}_\mathrm{c} > \mathcal{W}_\mathrm{b}$; • multiple connections between the same couple of presinaptic and postsinaptic neurons ("multapses") are allowed by default, but they can be disabled.

Connection rewiring	
Description	periodically, unconsolidated connections are pruned, and new connections are created: if h is the number of consolidated incoming connections of a neuron of $\mathcal{P}_2$, $\mathcal{C} - h$ new connections will be created, were $\mathcal{C}$ is a fixed number if the fixed-indegree connection rule is used, while it is extracted from a Poisson distribution if the Poisson-indegree rule is selected; in both cases, the presynaptic neurons are randomly extracted from $\mathcal{P}_1$.

Input stimulus	
Description	firing rate pattern of the neurons of $\mathcal{P}_1$ selected from the training or from the test set.

Contextual stimulus	
Description	firing rate pattern of the neurons of $\mathcal{P}_2$ selected from the training set; used only in the training phase.

Table 8.1: Description of the network model (continues on next page).

Train set	
Type	set of T independent firing-rate patterns of the neurons of $\mathcal{P}_1$ (input stimulus) and $\mathcal{P}_2$ (contextual stimulus).
Description	each pattern is randomly generated from predefined firing rate probability distributions, which can be discrete or continuous: • discrete model: the firing rates can assume only two discrete values ν_h (high rate) and ν_ℓ (low rate) with probabilities p_1 and $q_1 = 1 - p_1$ in the $\mathcal{P}_1$ population, p_2 and $q_2 = 1 - p_2$ in the $\mathcal{P}_2$ population. • continuous model: the firing rates can assume values on the basis of a continuous probability distribution; this work uses a lognormal distribution.
Test set	
Type	set of V firing-rate patterns of the neurons of $\mathcal{P}_1$ (input stimulus)
Description	each pattern is randomly extracted from the train set and eventually altered by adding noise from a predefined distribution: • discrete model: the pattern is left unchanged; • continuous model: the firing rate of each neuron is modified by adding a random deviation extracted from a predefined probability distribution; in this work, we used a truncated Gaussian distribution.

Table 8.1: Description of the network model (continues on next page).

Regarding the choice of the test set, since we provide the network sets of independent firing-rate patterns, it is not possible to generate a proper test set in the case of the discrete model. For this reason, we decided to test the network with the same firing-rate pattern used during training, even if this clearly leads to a bias in the learning process. However, in the continuous model, it is possible to create sets of firing-rate patterns and separate training and test sets. To generate the test set, we thus added Gaussian noise to the training patterns in order to mimic a different input to the one provided during training (i.e., a different element belonging to the same class).

8.3 Discrete rate model

As previously mentioned, in this model the input and contextual stimuli are represented by discrete firing-rate patterns of the two populations, $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, in which the firing rate of each neuron can assume only two possible values, ν_h (high rate) or ν_ℓ (low rate). Each training example consists of two patterns, one representing the input stimulus to the population $\mathcal{P}_1$, the other representing the contextual stimulus to the population $\mathcal{P}_2$. The pattern representing the input stimulus is generated by randomly setting the firing rate of each neuron of $\mathcal{P}_1$ from the two values, ν_{h} and ν_ℓ, with probabilities p_1 and $q_1 = 1 - p_1$, respectively. The corresponding pattern for the contextual stimulus is generated in a similar way, extracting the values of the firing rates of the $\mathcal{P}_2$ neurons, ν_{h} or ν_ℓ, with probabilities p_2 and $q_2 = 1 - p_2$, respectively.

Network and connectivity		
Name	**Value**	**Description**
$\mathcal{N}_1$	100000	number of neurons of $\mathcal{P}_1$
$\mathcal{N}_2$	100000	number of neurons of $\mathcal{P}_2$
$\mathcal{C}$	5000	number of connection in-degrees per neuron of $\mathcal{P}_2$
$\mathcal{T}$	variable	number of training patterns
s	100	connection rewiring step
Neuron		
Name	**Value**	**Description**
ν_ℓ	2.0 spikes/s	low firing rate
ν_{h}	50 spikes/s	high firing rate
Synapse		
Name	**Value**	**Description**
$\mathcal{W}_{\mathrm{b}}$	0.1 pA	baseline synaptic weight
$\mathcal{W}_{\mathrm{c}}$	1 pA	consolidated synaptic weight
Stimulus		
Name	**Value**	**Description**
p_1	0.001	probability for a neuron of $\mathcal{P}_1$ of having high rate when an input stimulus is injected
p_2	0.001	probability for a neuron of $\mathcal{P}_2$ of having high rate when a contextual stimulus is injected

Table 8.2: Model parameters.

The mean number of high-rate neurons in the two populations will be:

$$\begin{aligned} N_{h,1} &= \mathcal{N}_1 p_1 \\ N_{h,2} &= \mathcal{N}_2 p_2 \end{aligned} \tag{8.1}$$

where $\mathcal{N}_1$ and $\mathcal{N}_2$ indicate the number of neurons of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. A connection will be consolidated in a training example if both the presynaptic and the postsynaptic neuron assume a high firing rate ν_{h}. The probability that a generic connection is consolidated in a single example is $p_1 p_2$, and thus the probability that it is not consolidated after $\mathcal{T}$ training examples is $(1 - p_1 p_2)^{\mathcal{T}}$. The probability p

that a connection is consolidated in at least one of the $\mathcal{T}$ training examples is given by the complement of the previous expression:

$$p = 1 - (1 - p_1 p_2)^{\mathcal{T}} \tag{8.2}$$

The product of this expression by the number of incoming connections per neuron $\mathcal{C}$ gives us the average number of consolidated connections per neuron:

$$\langle k \rangle = \mathcal{C}\Big[1 - (1 - p_1 p_2)^{\mathcal{T}}\Big] = \mathcal{C}p \tag{8.3}$$

For each neuron, we will therefore have on average $\langle k \rangle$ consolidated connections with synaptic weight $\mathcal{W}_\text{c}$ and $\mathcal{C} - \langle k \rangle$ unconsolidated connections with synaptic weight $\mathcal{W}_\text{b}$.
The test set consists of V firing-rate patterns of the neurons of $\mathcal{P}_1$, randomly extracted from the $\mathcal{T}$ input patterns of the train set. In the discrete rate model, the patterns are unaltered, thus each input pattern of the test set is identical to an input pattern of the train set. The contextual stimuli are not used in the validation phase.
The average rate of the neurons in the population $\mathcal{P}_1$ is

$$\langle \nu \rangle = p_1 \nu_\text{h} + (1 - p_1)\nu_\ell \tag{8.4}$$

The input signal targeting a non-coding neuron of $\mathcal{P}_2$ (i.e., a background neuron) is equal to the weighted sum of the signals coming from the $\mathcal{C}$ connections:

$$\mathcal{S}_\text{b} = \mathcal{W}_\text{c} \sum_{i=1}^{k} \nu_i + \mathcal{W}_\text{b} \sum_{i=1}^{\mathcal{C}-k} \xi_i \tag{8.5}$$

where $\mathcal{C}$ is the number of incoming connections, k is the number of consolidated connections, ν_i are the firing rates of the neurons connected to the consolidated connections, and ξ_i are the firing rates of the neurons connected to the unconsolidated connections. From the linearity of $\mathcal{S}_\text{b}$ with respect to ν_i and ξ_i and from the fact that the rates of presynaptic neurons have the same mean value $\langle \nu \rangle$, it follows that

$$\langle \mathcal{S}_\text{b} \rangle = [\mathcal{W}_\text{c} \langle k \rangle + \mathcal{W}_\text{b}(\mathcal{C} - \langle k \rangle)]\langle \nu \rangle \tag{8.6}$$

In this equation, we can clearly observe two distinct contributions: one related to consolidated connections, which depends on the mean value $\langle k \rangle$, and the other related to unconsolidated connections, which depends on $\mathcal{C} - \langle k \rangle$. From this result, we can now calculate the variance on the background signal, which is defined as:

$$\sigma_b^2 = \langle (\mathcal{S}_\text{b} - \langle \mathcal{S}_\text{b} \rangle)^2 \rangle \tag{8.7}$$

Using Equations (8.5) and (8.6), we can compute the variance as:

$$\sigma_b^2 = \langle \Big[\mathcal{W}_\text{c} \sum_{i=1}^{k} \nu_i + \mathcal{W}_\text{b} \sum_{i=1}^{\mathcal{C}-k} \xi_i - [\mathcal{W}_\text{c} \langle k \rangle + \mathcal{W}_\text{b}(\mathcal{C} - \langle k \rangle)]\langle \nu \rangle \Big]^2 \rangle \tag{8.8}$$

Taking advantage of the equality $\langle k\rangle = k + (\langle k\rangle - k)$, we can rewrite:

$$\mathcal{W}_\mathrm{c}\langle k\rangle + \mathcal{W}_\mathrm{b}(\mathcal{C} - \langle k\rangle) = \mathcal{W}_\mathrm{c}k + \mathcal{W}_\mathrm{c}(\langle k\rangle - k) + \mathcal{W}_\mathrm{b}\Big[(\mathcal{C} - k) + (k - \langle k\rangle)\Big] = \tag{8.9}$$

$$= \mathcal{W}_\mathrm{c}k + \mathcal{W}_\mathrm{b}(\mathcal{C} - k) + \mathcal{W}_\mathrm{c}(\langle k\rangle - k) + \mathcal{W}_\mathrm{b}(k - \langle k\rangle)$$

Inserting this last expression in Equation (8.8) and rewriting the terms with the multiplicative factors k and $\mathcal{C} - k$ with summations, such as for example $\mathcal{W}_\mathrm{c}k\langle\nu\rangle = \mathcal{W}_\mathrm{c}\sum_{i=1}^{k}\langle\nu\rangle$, we obtain:

$$\sigma_b^2 = \langle\Big[\mathcal{W}_\mathrm{c}\sum_{i=1}^{k}(\nu_i - \langle\nu\rangle) + \mathcal{W}_\mathrm{b}\sum_{i=1}^{\mathcal{C}-k}(\xi_i - \langle\nu\rangle) + (k - \langle k\rangle)(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle\nu\rangle\Big]^2\rangle \tag{8.10}$$

Taking into account that the mixed terms go to zero since $\sum_i\langle(x_i - \langle x\rangle)\rangle = 0$, setting $\sum_i(x_i - \langle x\rangle)^2 = \sigma_x^2$, we will have that:

$$\sigma_b^2 = \Big[\mathcal{W}_\mathrm{c}^2\langle k\rangle + \mathcal{W}_\mathrm{b}^2(\mathcal{C} - \langle k\rangle)\Big]\sigma_\nu^2 + (\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})^2\sigma_k^2\langle\nu\rangle^2 \tag{8.11}$$

where $\sigma_k^2 = \langle(k - \langle k\rangle)^2\rangle$. In the previous formula, we note two contributions depending respectively on the variance of the firing rate and on the variance of the number of consolidated connections. The value of the variance of k is not shown here, but is derived in Appendix F, whereas the variance of the rate is, by definition, $\sigma_\nu^2 = \langle\nu^2\rangle - \langle\nu\rangle^2$.
Now we estimate the average input to a coding neuron of $\mathcal{P}_2$. The neuron receives signals from neurons of $\mathcal{P}_1$ coming from both consolidated and unconsolidated connections. If $\mathcal{C}$ is the number of incoming connections, the average number of high-rate presynaptic neurons will be $p_1\mathcal{C}$, while those with low rates will be on average $\mathcal{C}' = \mathcal{C}(1 - p_1)$. Since the input pattern used for validation is identical to the corresponding training pattern, the $p_1\mathcal{C}$ connections coming from high rate neurons will certainly be consolidated. The remaining $\mathcal{C}'$ connections come from neurons of $\mathcal{P}_1$ at low rate, however, they may have been consolidated in other training examples. The average number of consolidated connections from low-rate neurons can be calculated using Equation (8.3):

$$\langle k'\rangle = \mathcal{C}'p = \mathcal{C}(1 - p_1)p = \langle k\rangle(1 - p_1) \tag{8.12}$$

For a better understanding, the following scheme represents an example of coding and non-coding neurons during the test phase.

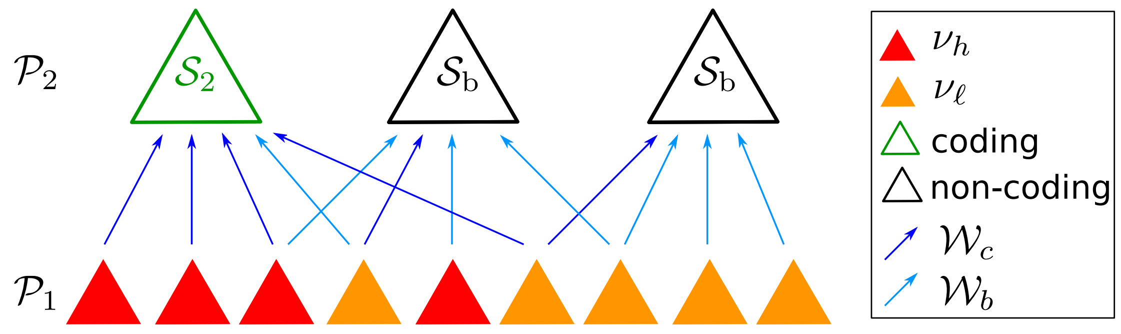


Figure 8.2: Scheme of a subset of the network during the test phase. Red and amber triangles represent single neurons of $\mathcal{P}_1$ having high or low rates, respectively. Dark and light blue arrows represent consolidated and unconsolidated connections, whereas green and black triangles represent coding or non-coding neurons of $\mathcal{P}_2$.

As we can see, a coding neuron receives, from high rate neurons of $\mathcal{P}_1$ only consolidated connections, but it can have consolidated or unconsolidated connections from low rate neurons. Summing up all the contributions, the average value of $\mathcal{S}_2$ is

$$\begin{aligned}\langle\mathcal{S}_2\rangle &= \mathcal{W}_\mathrm{c}\mathcal{C}p_1\nu_\mathrm{h} + \mathcal{W}_\mathrm{c}\langle k'\rangle\nu_\ell + \mathcal{W}_\mathrm{b}(\mathcal{C}' - \langle k'\rangle)\nu_\ell = \\ &= \mathcal{W}_\mathrm{c}\mathcal{C}p_1\nu_\mathrm{h} + \mathcal{W}_\mathrm{c}\langle k\rangle(1-p_1)\nu_\ell + \mathcal{W}_\mathrm{b}(\mathcal{C} - \langle k\rangle)(1-p_1)\nu_\ell = \\ &= \mathcal{W}_\mathrm{c}\mathcal{C}p_1\nu_\mathrm{h} + \Big[(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle k\rangle + \mathcal{C}\mathcal{W}_\mathrm{b}\Big](1-p_1)\nu_\ell\end{aligned} \tag{8.13}$$

The previous formula does not consider the rewiring of the connections; the effect of rewiring will be described in Section 8.4.2, where we will derive the expression of $\mathcal{S}_2$ that takes it into account. We identify this case as "with rewiring" to distinguish it from the case in which unconsolidated connections are not pruned and rewired. Indeed, this distinction is useful to estimate the contribution of this mechanism on the input signal on coding neurons of $\mathcal{P}_2$.
Now, it is possible to obtain the input signal-difference-to-noise-ratio (SDNR) using the formula

$$\mathrm{SDNR} = \frac{|\langle\mathcal{S}_2\rangle - \langle\mathcal{S}_\mathrm{b}\rangle|}{\sigma_b} \tag{8.14}$$

Here the SDNR is calculated on the input signal to the coding and non-coding neurons of the $\mathcal{P}_2$ population due to the connections coming from the $\mathcal{P}_2$ population, rather than on the firing rates of the $\mathcal{P}_2$ neurons. This choice is justified by the need to evaluate the memory capacity associated with the plasticity of the connections from $\mathcal{P}_1$ to $\mathcal{P}_2$. In general, in addition to the signal from the $\mathcal{P}_1$ population, the $\mathcal{P}_2$ neurons will receive other excitatory and inhibitory signals in input. As described in Section 1.5, in rate-based models, the response of neurons to the overall input signal is generally described by an activation function that expresses the firing rate as a function of that signal. A common choice is the threshold-linear (or ReLU) function described also in Equation (1.21):

$$\Phi(x) = \alpha \max\{0, x\} \tag{8.15}$$

where α is a multiplicative coefficient. With this choice, the average rates of coding and non-coding neurons of $\mathcal{P}_2$ can be written as

$$\begin{aligned} \langle \nu_\text{c} \rangle &= \Phi(\langle \mathcal{S}_2 \rangle + \langle S_\text{o} \rangle - S_\text{thresh}) \\ \langle \nu_\text{nc} \rangle &= \Phi(\langle \mathcal{S}_\text{b} \rangle + \langle S_\text{o} \rangle - S_\text{thresh}) \end{aligned} \tag{8.16}$$

where $\langle S_\text{o} \rangle$ is the average input signal from (excitatory and/or inhibitory) neuron populations different from $\mathcal{P}_1$ and S_thresh is the activation threshold. Assuming that the total input signal is above the threshold for both coding and non-coding neurons, the average rates will be linear functions of the input signals:

$$\begin{aligned} \langle \nu_\text{c} \rangle &= \alpha(\langle \mathcal{S}_2 \rangle + \langle S_\text{o} \rangle - S_\text{thresh}) \\ \langle \nu_\text{nc} \rangle &= \alpha(\langle \mathcal{S}_\text{b} \rangle + \langle S_\text{o} \rangle - S_\text{thresh}) \end{aligned} \tag{8.17}$$

while the variance of the non-coding neuron rate will be

$$\sigma_\text{nc}^2 = \alpha^2(\sigma_\text{b}^2 + \sigma_\text{o}^2) \tag{8.18}$$

The SDNR calculated on the rate will therefore be

$$\text{SDNR}_\nu = \frac{|\langle \nu_\text{c} \rangle - \langle \nu_\text{nc} \rangle|}{\sigma_\text{nc}} = \frac{|\langle \mathcal{S}_2 \rangle - \langle \mathcal{S}_\text{b} \rangle|}{\sqrt{\sigma_\text{b}^2 + \sigma_\text{o}^2}} \tag{8.19}$$

which has an expression similar to that reported in Equation (8.14), with the only difference that there is an additional contribution to the noise due to the signal coming from other populations.
Note also that the definitions of SDNR reported in Equations (8.14) and (8.19) refer to the mean signal difference between single coding and non-coding neurons. However, the overall memory capacity of the synaptic connections between the two populations can be best quantified by the SDNR evaluated on the total input signal to coding neurons and to an equivalent number of non-coding neurons. Calling $N_{h,2}$ the mean number of coding neurons in the population $\mathcal{P}_2$, we can define

$$\text{SDNR}_\text{pop} = \frac{|N_{h,2}\langle \mathcal{S}_2 \rangle - N_{h,2}\langle \mathcal{S}_\text{b} \rangle|}{\sqrt{N_{h,2}}\sigma_\text{b}} = \frac{\sqrt{p_2 N_2}|\langle \mathcal{S}_2 \rangle - \langle \mathcal{S}_\text{b} \rangle|}{\sigma_\text{b}} \tag{8.20}$$

where we used Equation (8.1) and $\sqrt{N_{h,2}}\sigma_\text{b}$ is the standard deviation of the total input signal to $N_{h,2}$ non-coding neurons. Thus, SDNR_pop scales with the square root of $p_2 N_2$.
Table 8.3 summarizes the equations of the discrete rate model.

Discrete rate model		
Name	**Symbol**	**Equation**
Rate mean	$\langle \nu \rangle$	$p_1 \nu_\mathrm{h} + (1-p_1)\nu_\ell$
Rate variance	σ_ν^2	$\Big(p_1 \nu_\mathrm{h}^2 + (1-p_1)\nu_\ell^2\Big) - \Big(p_1 \nu_\mathrm{h} + (1-p_1)\nu_\ell\Big)^2$
Average background signal	$\langle \mathcal{S}_\mathrm{b} \rangle$	$\langle k \rangle \mathcal{W}_\mathrm{c} \langle \nu \rangle + (\mathcal{C} - \langle k \rangle)\mathcal{W}_\mathrm{b} \langle \nu \rangle$
Variance of background signal	σ_b^2	$\Big[\mathcal{W}_\mathrm{c}^2 \langle k \rangle + \mathcal{W}_\mathrm{b}^2 (\mathcal{C} - \langle k \rangle)\Big]\sigma_\nu^2 + (\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})^2 \sigma_k^2 \langle \nu \rangle^2$
Average coding neuron signal (without rewiring)	$\langle \mathcal{S}_2 \rangle$	$\mathcal{W}_\mathrm{c} \mathcal{C} p_1 \nu_\mathrm{h} + \Big[(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle k \rangle + \mathcal{C}\mathcal{W}_\mathrm{b}\Big](1-p_1)\nu_\ell$

Table 8.3: Summary of the equations for the discrete model.

8.4 Continuous model

In this model, the firing rate patterns are generated from a continuous probability distribution, $\rho(\nu)$. The distinction between high-rate and low-rate neurons is based on two rate thresholds, $\nu_{\mathrm{t},1}$ for the population $\mathcal{P}_1$ and $\nu_{\mathrm{t},2}$ for the population $\mathcal{P}_2$. The values of these thresholds are related to the fraction of neurons above the threshold for the two populations, p_1 and p_2, respectively, by the equations:

$$\begin{aligned} p_1 &= \int_{\nu_{\mathrm{t},1}}^{\infty} \rho(\nu) d\nu \\ p_2 &= \int_{\nu_{\mathrm{t},2}}^{\infty} \rho(\nu) d\nu \end{aligned} \tag{8.21}$$

The average rates of the neurons of $\mathcal{P}_1$ below and above the threshold, $\langle \nu_{\ell,1} \rangle$ and $\langle \nu_{\mathrm{h},1} \rangle$, can be computed from the firing rate distribution as:

$$\begin{aligned} \langle \nu_{\ell,1} \rangle &= \int_0^{\nu_{\mathrm{t},1}} \nu \rho(\nu) d\nu \Big/ \int_0^{\nu_{\mathrm{t},1}} \rho(\nu) d\nu = \frac{1}{q_1} \int_0^{\nu_{\mathrm{t},1}} \nu \rho(\nu) d\nu \\ \langle \nu_{\mathrm{h},1} \rangle &= \int_{\nu_{\mathrm{t},1}}^{\infty} \nu \rho(\nu) d\nu \Big/ \int_{\nu_{\mathrm{t},1}}^{\infty} \rho(\nu) d\nu = \frac{1}{p_1} \int_{\nu_{\mathrm{t},1}}^{\infty} \nu \rho(\nu) d\nu \end{aligned} \tag{8.22}$$

where $q_1 = 1 - p_1$. Similar equations can be used to compute $\langle \nu_{\ell,2} \rangle$ and $\langle \nu_{\mathrm{h},2} \rangle$. From these equations the average firing rate of $\mathcal{P}_2$ can be expressed as

$$\langle \nu \rangle = \int_0^{\infty} \nu \rho(\nu) d\nu = q_1 \langle \nu_{\ell,1} \rangle + p_1 \langle \nu_{\mathrm{h},1} \rangle = q_2 \langle \nu_{\ell,2} \rangle + p_2 \langle \nu_{\mathrm{h},2} \rangle \tag{8.23}$$

In a training example, a connection will be consolidated if both the presynaptic and the postsynaptic neuron assume a firing rate above the threshold. Figure 8.3

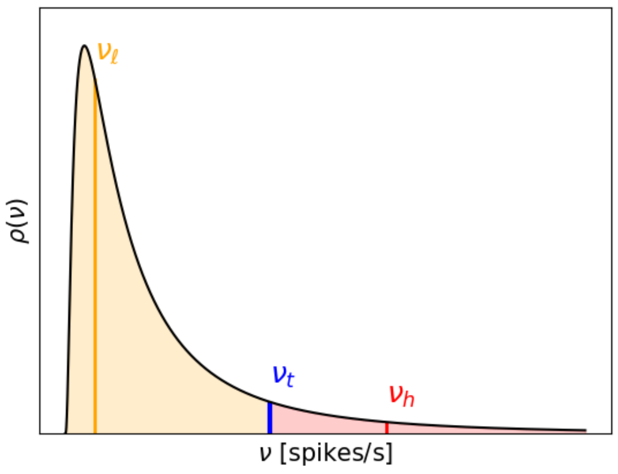


Figure 8.3: Lognormal distribution of firing rate. The black solid line indicates the probability distribution, which is divided into two sections by the rate threshold ν_t (blue, vertical line). The amber band represents the distribution of rate below the threshold, whose mean is ν_ℓ (amber, vertical line). The red band represents the distribution of neurons whose rate is above threshold. Here the average of this section is ν_h (red, vertical line). Figure from [5].

depicts an example of firing rate distribution, with the threshold and the average values of low and high firing rates. Adopting such a threshold mechanism, we are able to identify high and low rate neurons as in the discrete case, resembling thus the scheme shown in Figure 8.2 for the discrete model during test phase. The only difference is that since in this case the rate is not discrete, instead of having ν_h or ν_ℓ we have $\langle\nu_\mathrm{h}\rangle$ and $\langle\nu_\ell\rangle$.

As in the discrete model, the test set consists of V firing-rate patterns of the neurons of $\mathcal{P}_1$, randomly extracted from the $\mathcal{T}$ input patterns of the train set. Here we consider the case where the patterns are unchanged, thus each input pattern of the test set is identical to an input pattern of the train set. In a later section, we will discuss the effect of altering these patterns by adding noise. To estimate the values of $\langle\mathcal{S}_2\rangle$, $\langle\mathcal{S}_\mathrm{b}\rangle$, and σ^2_b we proceed in a similar way as for the discrete model. First of all, we calculate the input signal to a generic non-coding neuron of $\mathcal{P}_2$. Let $\mathcal{C}$ be the number of incoming connections to this neuron, $P(k)$ the probability that k of these connections are consolidated, $\nu_1,\cdots,\nu_k$ the firing rates of the presynaptic neurons of the consolidated connections and $\xi_1,\cdots,\xi_{\mathcal{C}-k}$ the firing rates of the presynaptic neurons of the unconsolidated connections. The probability of having k consolidated connections and rates in the range $(\nu_1,\nu_1+d\nu_1),\cdots,(\nu_k,\nu_k+d\nu_k)$, $(\xi_1,\xi_1{+}d\xi_1),\cdots,(\xi_{\mathcal{C}-k},\xi_{\mathcal{C}-k}{+}d\xi_{\mathcal{C}-k})$ is $P(k)\rho(\nu_1)\cdots\rho(\nu_k)\rho(\xi_1)\cdots\rho(\xi_{\mathcal{C}-k})d\nu_1\cdots d\nu_k d\xi_1\cdots d\xi_{\mathcal{C}-k}$. To calculate the average background signal we should average the expression of $\mathcal{S}_\mathrm{b}$, given by Equation (8.5), over all the possible values of k and of the firing rates, thus

$$\begin{aligned}\langle\mathcal{S}_\mathrm{b}\rangle &= \sum_k P(k)\int d\nu_1\cdots\int d\nu_k\int d\xi_1\cdots\int d\xi_{\mathcal{C}-k}\rho(\nu_1)\cdots\rho(\nu_k)\rho(\xi_1)\cdots\rho(\xi_{\mathcal{C}-k})\cdot\\ &\cdot\Big[\mathcal{W}_\mathrm{c}(\nu_1+\cdots+\nu_k)+\mathcal{W}_\mathrm{b}(\xi_1+\cdots+\xi_{\mathcal{C}-k})\Big]=\\ &=\sum_k P(k)\Big[\mathcal{W}_\mathrm{c}k\langle\nu\rangle+\mathcal{W}_\mathrm{b}(\mathcal{C}-k)\langle\nu\rangle\Big]=[\mathcal{W}_\mathrm{c}\langle k\rangle+\mathcal{W}_\mathrm{b}(\mathcal{C}-\langle k\rangle)]\langle\nu\rangle\end{aligned}\tag{8.24}$$

where we used the fact that $\int\nu\rho(\nu)d\nu=\int\xi\rho(\xi)d\xi=\langle\nu\rangle$. Note that the result obtained in Equation (8.24) is the same as the one obtained for the discrete model (see Equation (8.6)).
The variance of the background signal can be similarly derived:

$$\begin{aligned}\sigma_\mathrm{b}^2 &= \langle(\mathcal{S}_\mathrm{b}-\langle\mathcal{S}_\mathrm{b}\rangle)^2\rangle=\sum_k P(k)\int d\nu_1\cdots\int d\nu_k\int d\xi_1\cdots\int d\xi_{\mathcal{C}-k}\rho(\nu_1)\cdots\rho(\xi_{\mathcal{C}-k})\cdot\\ &\cdot\Big[\mathcal{W}_\mathrm{c}\sum_{i=1}^{k}\nu_i+\mathcal{W}_\mathrm{b}\sum_{i=1}^{\mathcal{C}-k}\xi_i-[\mathcal{W}_\mathrm{c}\langle k\rangle+\mathcal{W}_\mathrm{b}(\mathcal{C}-\langle k\rangle)]\langle\nu\rangle\Big]^2=\\ &=\sum_k P(k)\int d\nu_1\cdots\int d\xi_{\mathcal{C}-k}\rho(\nu_1)\cdots\rho(\xi_{\mathcal{C}-k})\Big[\mathcal{W}_\mathrm{c}\sum_{i=1}^{k}(\nu_i-\langle\nu\rangle)+\mathcal{W}_\mathrm{b}\sum_{i=1}^{\mathcal{C}-k}(\xi_i-\langle\nu\rangle)+\\ &+(k-\langle k\rangle)(\mathcal{W}_\mathrm{c}-\mathcal{W}_\mathrm{b})\langle\nu\rangle\Big]^2\end{aligned}\tag{8.25}$$

Where in the last line we used the substitution $\langle k\rangle=k+(\langle k\rangle-k)$ as done for Equation (8.10). The mixed terms of the equation above are null because $\int\rho(x)(x-\langle x\rangle)dx=0$, ergo we can write the variance of the background signal as follows:

$$\begin{aligned}\sigma_\mathrm{b}^2 &= \sum_k P(k)\int d\nu_1\cdots\int d\xi_{\mathcal{C}-k}\rho(\nu_1)\cdots\rho(\xi_{\mathcal{C}-k})\Big[\mathcal{W}_\mathrm{c}^2k\langle(\nu-\langle\nu\rangle)^2\rangle+\mathcal{W}_\mathrm{b}^2(\mathcal{C}-k)\langle(\nu-\langle\nu\rangle)^2\rangle+\\ &+(\mathcal{W}_\mathrm{c}-\mathcal{W}_\mathrm{b})^2(k-\langle k\rangle)^2\langle\nu\rangle^2\Big]=\Big[\mathcal{W}_\mathrm{c}^2\langle k\rangle+\mathcal{W}_\mathrm{b}^2(\mathcal{C}-\langle k\rangle)\Big]\sigma_\nu^2+(\mathcal{W}_\mathrm{c}-\mathcal{W}_\mathrm{b})^2\sigma_k^2\langle\nu\rangle^2\end{aligned}\tag{8.26}$$

The variance of k has the same expression as for the discrete case, and is derived in Appendix F. $\mathcal{S}_2$ can be derived similarly to the discrete case as well. We have a contribution from $p_1\mathcal{C}$ neurons of $\mathcal{P}_1$ at high rate connected by consolidated connections. The remaining $\mathcal{C}'=\mathcal{C}-p_1\mathcal{C}$ neurons have a low rate and can be connected either by unconsolidated connections or by connections that have been consolidated in input patterns different from the current one. Calling $\langle k'\rangle$ the average number of consolidated connections outgoing from the $\mathcal{C}'$ low-rate neurons and using the definition of $\langle\nu_\ell\rangle$ and $\langle\nu_\mathrm{h}\rangle$ given by Equation (8.22) we can write

$$\begin{aligned}\langle\mathcal{S}_2\rangle &= \mathcal{W}_\mathrm{c}p_1\mathcal{C}\langle\nu_\mathrm{h}\rangle+(\mathcal{W}_\mathrm{c}-\mathcal{W}_\mathrm{b})\langle k'\rangle\langle\nu_\ell\rangle+\mathcal{W}_\mathrm{b}\mathcal{C}(1-p_1)\langle\nu_\ell\rangle=\\ &=\mathcal{W}_\mathrm{c}p_1\mathcal{C}\langle\nu_\mathrm{h}\rangle+\Big[(\mathcal{W}_\mathrm{c}-\mathcal{W}_\mathrm{b})\langle k\rangle+\mathcal{C}\mathcal{W}_\mathrm{b}\Big](1-p_1)\langle\nu_\ell\rangle\end{aligned}\tag{8.27}$$

where we used the expression of $\langle k' \rangle$ from Equation (8.12). As can be seen, Equation (8.27) differs from (8.13) only because the rates are not discrete but can assume continuous values, thus the discrete values $\nu_{\rm h}$ and ν_ℓ are replaced by the averages $\langle \nu_{\rm h} \rangle$ and $\langle \nu_\ell \rangle$. The modified formula of $\mathcal{S}_2$ that takes into account connection rewiring will be derived in Section 8.4.2.
In this work, we use a lognormal distribution of the firing rates for the continuous model. Indeed, it is known that rate distribution in the cortex is long-tailed and skewed with a lognormal shape [149]. The lognormal distribution is a continuous probability distribution of a random variable ν whose logarithm $\ln(\nu)$ is normally distributed. The probability density function of this distribution is

$$\rho_{\rm LN}(\nu) = \frac{1}{\sqrt{2\pi}\sigma\nu} \cdot \exp\Big(-\frac{(\ln(\nu)-\mu)^2}{2\sigma^2}\Big) \tag{8.28}$$

where μ and σ are the mean and standard deviation of $\ln(\nu)$. The expressions of the mean value and standard deviation of $\rho_{\rm LN}(\nu)$ will be derived in Appendix E.

8.4.1 Poisson distribution of incoming connections per neuron

Hitherto we considered a model in which each neuron of $\mathcal{P}_2$ has a fixed number of incoming connections, i.e., a fixed in-degree, $\mathcal{C}$. However, a more general and realistic approach would consider $\mathcal{C}$ as a variable across the neurons of $\mathcal{P}_2$ according to an appropriate probability distribution $P(\mathcal{C})$. Here we focus on the case where the number of incoming connections follows a Poisson distribution (i.e. a Poisson-indegree connection rule), however, the approach we will present can be easily extended to other distributions. The values of $\langle \mathcal{S}_2 \rangle$ and $\langle \mathcal{S}_{\rm b} \rangle$, previously averaged over the rate ν and the number of consolidated connections k, should be also averaged over the number of incoming connections, so that

$$\begin{aligned} \langle \langle \mathcal{S}_{\rm b} \rangle_{\nu,k} \rangle_{\mathcal{C}} &= \sum_{\mathcal{C}} P(\mathcal{C}) \langle \mathcal{S}_{\rm b} \rangle_{\nu,k} \\ \langle \langle \mathcal{S}_2 \rangle_{\nu,k} \rangle_{\mathcal{C}} &= \sum_{\mathcal{C}} P(\mathcal{C}) \langle \mathcal{S}_2 \rangle_{\nu,k} \end{aligned} \tag{8.29}$$

where $\langle \mathcal{S}_2 \rangle_{\nu,k}$ is given by Equation (8.27) and $\langle \mathcal{S}_{\rm b} \rangle_{\nu,k}$ is given by Equation (8.24). Since these equations are linear in $\mathcal{C}$ and since $\sum_{\mathcal{C}} \mathcal{C} P(\mathcal{C}) = \langle \mathcal{C} \rangle$, Equations (8.27) and (8.24) would show $\langle \mathcal{C} \rangle$ instead of $\mathcal{C}$ when averaged over the number of incoming connections per neuron.
The variance can be obtained from the equation:

$$\mathrm{Var}(\langle \langle \mathcal{S}_{\rm b} \rangle_{\nu,k} \rangle_{\mathcal{C}}) = \sigma^2_{\nu,k,\mathcal{C}} = \langle \langle \mathcal{S}^2_{\rm b} \rangle_{\nu,k} \rangle_{\mathcal{C}} - \langle \langle \mathcal{S}_{\rm b} \rangle_{\nu,k} \rangle^2_{\mathcal{C}} \tag{8.30}$$

Knowing that $\langle \sigma^2_{\rm b} \rangle_{\mathcal{C}} = \langle \langle \mathcal{S}^2_{\rm b} \rangle_{\nu_k} - \langle \mathcal{S}_{\rm b} \rangle^2_{\nu,k} \rangle_{\mathcal{C}} = \langle \langle \mathcal{S}^2_{\rm b} \rangle_{\nu,k} \rangle_{\mathcal{C}} - \langle \langle \mathcal{S}_{\rm b} \rangle^2_{\nu,k} \rangle_{\mathcal{C}}$ and that $\langle k \rangle = p\mathcal{C}$ we can write

$$\begin{aligned} \sigma^2_{\nu,k,\mathcal{C}} &= \langle \sigma^2_{\rm b} \rangle_{\mathcal{C}} + \langle \langle \mathcal{S}_{\rm b} \rangle^2_{\nu,k} \rangle_{\mathcal{C}} - \langle \langle \mathcal{S}_{\rm b} \rangle_{\nu,k} \rangle^2_{\mathcal{C}} = \\ &= \langle \sigma^2_{\rm b} \rangle_{\mathcal{C}} + \Big\{ \langle \nu \rangle \Big[\mathcal{W}_{\rm b} + p(\mathcal{W}_{\rm c} - \mathcal{W}_{\rm b}) \Big] \Big\}^2 \Big[\langle \mathcal{C}^2 \rangle - \langle \mathcal{C} \rangle^2 \Big] = \\ &= \langle \sigma^2_{\rm b} \rangle_{\mathcal{C}} + \langle \nu \rangle^2 \Big[\mathcal{W}_{\rm b} + p(\mathcal{W}_{\rm c} - \mathcal{W}_{\rm b}) \Big]^2 \sigma^2_{\mathcal{C}} \end{aligned} \tag{8.31}$$

This equation is also valid for the discrete model.

8.4.2 Connection Rewiring

In the proposed approach, rewiring is implemented by periodically pruning unconsolidated connections and creating new ones. Indeed, the former resembles the LTD effects of pruning synapses from neurons with uncorrelated activity, and the process of new connection creation is related to the homeostatic mechanism. We decided to model these two mechanisms together as a single process of connection rewiring, in which connections that are not yet consolidated are redistributed across the network. These procedures are performed with a fixed step on the number of training examples, which we will call *rewiring step*, denoted by the letter s. The redistribution of connections is made in such a way as to keep the distribution of the number of incoming connections per neuron unchanged. If h is the number of consolidated incoming connections of a neuron of $\mathcal{P}_2$, after pruning all the unconsolidated connections, $\mathcal{C} - h$ new connections will be created. $\mathcal{C}$ is a fixed number if the fixed-indegree connection rule is used, while it is extracted from a Poisson distribution if the Poisson-indegree rule is selected; in both cases, the presynaptic neurons are randomly extracted from $\mathcal{P}_1$. For this reason, rewiring leaves the expressions of the background signal and of the variance on this signal unchanged, while, as we will see, it modifies the input signal to coding neurons. A diagram of the rewiring process is shown in Figure 8.4, which illustrates the activity of a high-rate neuron of $\mathcal{P}_2$ and of the presynaptic neurons of its incoming connections in a training example, and the effect of connection rewiring.

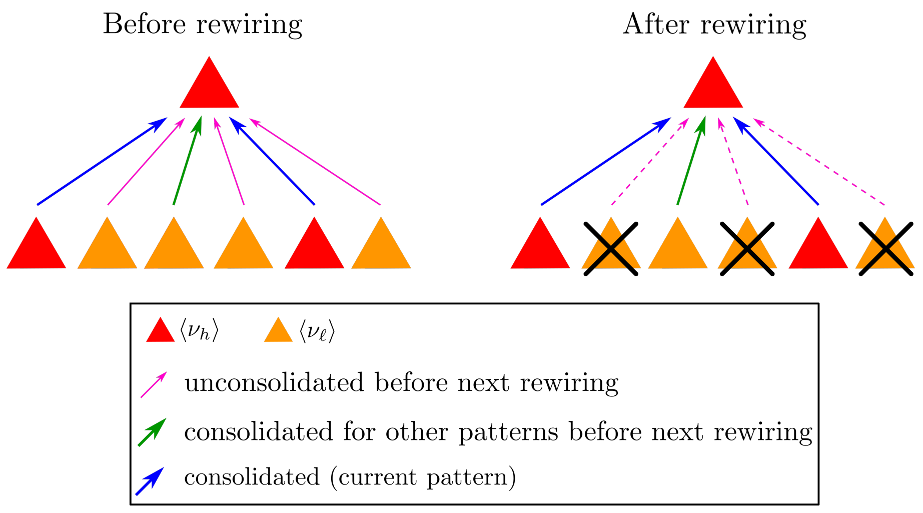

Before rewiring
After rewiring
$\langle\nu_h\rangle$
$\langle\nu_\ell\rangle$
unconsolidated before next rewiring
consolidated for other patterns before next rewiring
consolidated (current pattern)

Figure 8.4: Scheme of the rewiring process and of its effect on the average signal in input to coding neurons of the population $\mathcal{P}_2$. Red and amber triangles represent high and low rate neurons respectively. Blue and green connections are consolidated for the current example and for a previous example, whereas pink connections are not consolidated. After rewiring these connections are pruned (here indicated with a dotted arrow), thus the relative presynaptic neurons do not project anymore to the postsynaptic neuron under consideration. For better readability, the new instantiated connections are not shown in this scheme. Figure from [5].

The average number of incoming connections that are consolidated in the current example (blue lines) is equal to the average number of high-rate presynaptic neurons, $p_1\mathcal{C}$. The average number of incoming connections that are consolidated in other examples after the entire training, $\langle k' \rangle$, is given by equation (8.12):

$$\langle k' \rangle = p\mathcal{C}(1 - p_1) \tag{8.32}$$

Let t be the next training index for which rewiring will be applied, and k'_t the number of connections from low-rate neurons that are consolidated before t (green lines in the figure). These connections will not be affected by rewiring, so even in the test phase with the same input pattern they will have low-rate presynaptic neurons. The average value of k'_t is

$$\langle k'_t \rangle = p_t\mathcal{C}(1 - p_1) \tag{8.33}$$

where p_t is given by an expression analogous to the one obtained for p (Equation (8.2))

$$p_t = 1 - (1 - p_1 p_2)^t \tag{8.34}$$

On the other hand, there will be $k' - k'_t$ connections displaced by rewiring and consolidated in training examples of index greater than t. Putting all the contributions together, we obtain the following expression for $\mathcal{S}_2$:

$$\begin{aligned}\langle \mathcal{S}_2 \rangle &= p_1\mathcal{C}\mathcal{W}_\mathrm{c}\langle \nu_\mathrm{h} \rangle + (\langle k' \rangle - \langle k'_t \rangle)\mathcal{W}_\mathrm{c}\langle \nu \rangle + \langle k'_t \rangle \mathcal{W}_\mathrm{c}\langle \nu_\ell \rangle + \mathcal{W}_\mathrm{b}(\mathcal{C} - p_1\mathcal{C} - \langle k' \rangle)\langle \nu \rangle = \\ &= p_1\mathcal{C}\mathcal{W}_\mathrm{c}\langle \nu_\mathrm{h} \rangle + \langle k' \rangle \mathcal{W}_\mathrm{c}\langle \nu \rangle + \mathcal{W}_\mathrm{b}[\mathcal{C}(1 - p_1) - \langle k' \rangle]\langle \nu \rangle - \langle k'_t \rangle \mathcal{W}_\mathrm{c}(\langle \nu \rangle - \langle \nu_\ell \rangle)\end{aligned} \tag{8.35}$$

To obtain the average value of $\mathcal{S}_2$ over all examples, $\langle k'_t \rangle$ must be averaged over all values of the index t for which rewiring is done. This calculation is shown in Appendix G.1. Equation (8.35) is also valid in the case for which neurons can assume discrete values of firing rate. In that case, $\langle \nu_\ell \rangle$, $\langle \nu_\mathrm{h} \rangle$ and $\langle \nu \rangle$ have to be replaced with the discrete values ν_ℓ, ν_h and the average rate $\langle \nu \rangle$ shown in Table 8.3.

8.4.3 Introduction of noise into input patterns

In a realistic learning model, the test patterns will never be exactly the same as the training ones. The ability of a learning model to generalize is linked to the ability to recognize which training pattern or patterns are most similar to a given test pattern, according to appropriate metrics. To study the generalization capacity of the model proposed in this work, the test input patterns were generated starting from the corresponding training input patterns by adding noise, which is represented by a deviation extracted from a given probability distribution with an assigned standard deviation. In Appendix G we describe the effect that noise with a truncated Gaussian distribution has on the firing rates and on the variables $\mathcal{S}_\mathrm{b}$, $\mathcal{S}_2$, σ^2_b, and SDNR, and we derive the modified equations.

8.4.4 Summary of theoretical model equations

Table 8.4 summarizes the equations of the continuous rate model in case of a fixed number of incoming connections per neuron. As mentioned before, when the value of $\mathcal{C}$ follows a Poisson distribution, $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ are given by the same expression obtained for the fixed in-degree connection rule with $\mathcal{C}$ replaced by $\langle \mathcal{C} \rangle$. The variance of the background signal has an additional term in that case and is given by Equation (8.31).

Continuous rate model		
Name	**Symbol**	**Equation**
Rate distribution	$\rho(\nu)$	$\frac{1}{\sqrt{2\pi}\sigma\nu} \cdot \exp\left(-\frac{(\ln(\nu)-\mu)^2}{2\sigma^2}\right)$
Mean of the normal distribution of $\ln(\nu)$	μ	$\ln(\langle\nu\rangle) - \frac{\sigma^2}{2}$
Standard deviation of the normal distribution of $\ln(\nu)$	σ	$\mathrm{erf}^{-1}(q_1) - \mathrm{erf}^{-1}\left(\frac{q_1\langle\nu_\ell\rangle}{\langle\nu\rangle}\right)$
Rate threshold	ν_t	$\exp\left(\mathrm{erf}^{-1}(q_1)\sigma + \mu\right)$
Average high rate	$\langle\nu_\mathrm{h}\rangle$	$\frac{1}{p_1}\int_{\nu_\mathrm{t}}^{\infty} \nu\rho(\nu)d\nu$
Average low rate	$\langle\nu_\ell\rangle$	$\frac{1}{q_1}\int_{0}^{\nu_\mathrm{t}} \nu\rho(\nu)d\nu$
Average rate	$\langle\nu\rangle$	$q_1\langle\nu_\ell\rangle + p_1\langle\nu_\mathrm{h}\rangle$
Rate standard deviation	σ_ν^2	$\left(e^{\sigma^2} - 1\right)e^{2\mu+\sigma^2}$
Average background signal	$\langle\mathcal{S}_\mathrm{b}\rangle$	$(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle k\rangle\langle\nu\rangle + \mathcal{W}_\mathrm{b}\mathcal{C}\langle\nu\rangle$
Variance of background signal	σ_b^2	$\left[\mathcal{W}_\mathrm{c}^2\langle k\rangle + \mathcal{W}_\mathrm{b}^2(\mathcal{C} - \langle k\rangle)\right]\sigma_\nu^2 + (\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})^2\sigma_k^2\langle\nu\rangle^2$
Average coding-neuron signal (without rewiring)	$\langle\mathcal{S}_2\rangle$	$\mathcal{W}_\mathrm{c}p_1\mathcal{C}\langle\nu_\mathrm{h}\rangle + \left[(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle k\rangle + \mathcal{C}\mathcal{W}_\mathrm{b}\right](1-p_1)\langle\nu_\ell\rangle$
Average coding-neuron signal (with rewiring)	$\langle\mathcal{S}_2\rangle$	$\mathcal{W}_\mathrm{c}p_1\mathcal{C}\langle\nu_\mathrm{h}\rangle + \left[(\mathcal{W}_\mathrm{c} - \mathcal{W}_\mathrm{b})\langle k\rangle + \mathcal{C}\mathcal{W}_\mathrm{b}\right](1-p_1)\langle\nu\rangle - \langle k_t'\rangle\mathcal{W}_\mathrm{c}(\langle\nu\rangle - \langle\nu_\ell\rangle)$

Table 8.4: Summary of the equations for the continuous model. Equations shown here refer to the case in which the number of incoming connections per neuron $\mathcal{C}$ is constant.

8.5 Recurrent neuron model

Here we describe the recurrent network model, in which population $\mathcal{P}_2$ shows recurrent connections and projects to an inhibitory population $\mathcal{P}_i$ from which it receives a global inhibitory signal[1]. Indeed, self-connections and inhibitory feedback are fundamental ingredients to realize realistic networks oriented towards the study of learning process [130], and several other spiking learning models are based on a similar structure to the one depicted in Figure 8.5 (see, for example, [150, 151]). Figure 8.5 depicts a scheme of the network.

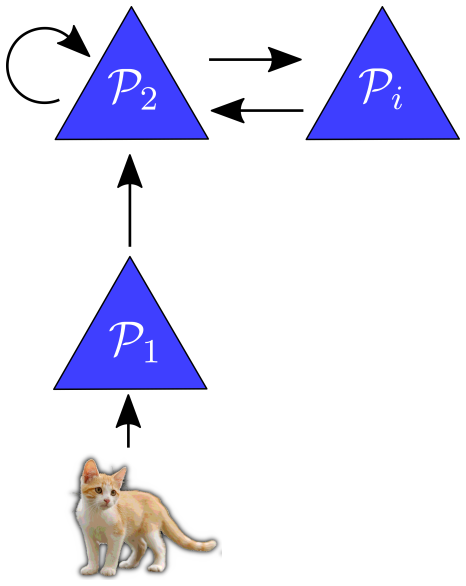


Figure 8.5: Schematic representation of the recurrent network model through a block diagram.

In this case, we follow an approach similar to the one proposed in Section 1.5, in particular describing each population with a differential equation of the average rate similar to Equations (1.24) and (1.25), in which every population is provided with time constant τ needed to describe the time needed for the population to reach a steady state firing rate. In this case, only the time constant between excitatory and inhibitory population is distinguished.
In this section, we derive the expressions for the background signal $\mathcal{S}_b$ and coding neuron signal $\mathcal{S}_2$ to the neurons of population $\mathcal{P}_2$. To this aim, several input average rates y_i have to be considered:

- y_0: average input rate from non-coding neurons and non-consolidated connections;
- y_1: average input rate from non-coding neurons and consolidated connections;

[1]N.B. this theoretical framework is still in development, additional work is required to provide an appropriate framework to be compared with computational simulations.

- y_2: average input rate from coding neurons and non-consolidated connections;
- y_3: average input rate from coding neurons and consolidated connections;
- y_4: average input rate from inhibitory neurons.

The reason for such an input grouping will be clear with the following derivation of the equations for $\mathcal{S}_\text{b}$ and $\mathcal{S}_2$. The differential equations describing the listed average rates are, according to Section 1.5

$$\tau_i \frac{dy_i}{dt} = -y_i + \mathcal{F}\Big(\sum_{j=0}^{4} W_{ij} y_j + E_i - S_{\text{th},i}\Big) \tag{8.36}$$

Where τ_i is the time constant for the input i, W_{ij} is the matrix of the synaptic weights multiplied by the average number of indegrees from the population j to population i, E_i is the input rate from other populations, including the input coming from $\mathcal{P}_1$, and $S_{\text{th},i}$ represent the activation threshold for the neuron population i and, eventually, $\mathcal{F}$ is the activation function. Choosing the ReLU (see Equation (1.21)) as activation function, a reasonable assumption we can make is considering to be over threshold for every neuron population since both excitatory and inhibitory neurons show significant activity also during rest. Thus, the ReLu can be reduced to a linear function

$$\mathcal{F}(x) = \alpha x H(x) = \alpha x \qquad \text{with} \quad x = \sum_{j=0}^{4} W_{ij} y_j + E_i - S_{\text{th},i} \quad \text{and} \quad x > 0 \tag{8.37}$$

Hence, Equation (8.36) becomes

$$\tau_i \frac{dy_i}{dt} = -y_i + \alpha\Big(\sum_{j=0}^{4} W_{ij} y_j + E_i - S_{\text{th},i}\Big) \tag{8.38}$$

Now we can rewrite the equation in a matrix form. Given $C_{ij} = \frac{1}{\tau_i}(-\delta_{ij} + \alpha W_{ij})$ and $b_i = \frac{\alpha}{\tau_i}(E_i - S_{\text{th},i})$ we have

$$\frac{dy_i}{dt} = \sum_{j=0}^{4} C_{ij} y_j + b_i \tag{8.39}$$

which can be written as follows

$$\frac{d\vec{y}}{dt} = C\vec{y} + \vec{b} \tag{8.40}$$

Here we focus on the study of the steady state of the previous equation, in which the average firing rates of the different neuron populations do not change anymore over time. Thus, we can set the derivative of Equation (8.36) to zero, obtaining

$$0 = -y_i + \alpha(\sum_{j=0}^{4} W_{ij} y_j + E_i - S_{\text{th},i}) \tag{8.41}$$

from which we can obtain a matrix equation in which we have to solve for the average rates y_i:

$$\begin{aligned} y_i - \alpha \sum_{j=0}^{4} W_{ij} y_j &= \alpha(E_i - S_{\text{th},i}) \\ (I - A)\vec{y} = \vec{b} \qquad &\Longrightarrow \qquad \vec{y} = M^{-1}\vec{b} \\ \text{with} \qquad A_{ij} = \alpha W_{ij} \quad \text{and} \quad & b_i = \alpha(E_i - S_{\text{th},i}) \quad \text{and} \quad M = I - A \end{aligned} \tag{8.42}$$

Thus, we can solve this equation by computing M^{-1}. Before obtaining the equations for $\langle \mathcal{S}_\text{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ we should motivate the introduction of the four different rates y_i instead of having neurons with only high or low rates ν_h and ν_ℓ. The distinction between these four input rates is necessary because of self-connections. Indeed, there is a correlation between connection consolidation and average input rate. We present an example to make this clear.
Let us consider some neurons of population $\mathcal{P}_2$ when the network is trained with $\mathcal{T} = 10^5$ patterns. Having a probability $p_2 = 10^{-3}$ for a neuron to be representative of a pattern, this means that, on average, a neuron will be representative for $\langle m \rangle = 10$ patterns. The variable m will have a variance $\sigma_m^2 \propto \sqrt{m}$, hence some neurons can be representative for more than 10 patterns, and consequently these neurons will have more consolidated connections both in input and output due to the self-connections. Thus, neurons with higher m have more consolidated connections, ergo they are likely to show a higher firing rate. This way a correlation between the average rate and the number of synaptic connections comes about, and the theoretical framework should take account of such a correlation. In order to show that, we need to do some calculations similar to the ones reported in Appendix F in which we compute the variance of the number of consolidated connections k for the feed-forward model. In particular, it is important to remember that $Q(m, h)$ (see Equation (F.1)) represents the probability that a neuron of $\mathcal{P}_2$ is representative of m patterns (i.e., is trained for recognizing m input patterns) and has h consolidated indegrees from neurons of $\mathcal{P}_1$. Analogously, we can derive an expression to represent the probability for a neuron of $\mathcal{P}_1$ to be representative of m patterns and with h consolidated outdegree to neurons of $\mathcal{P}_2$. In the following, we call C_{12}^{in} the average number of connections in input for a neuron of $\mathcal{P}_2$ coming from population $\mathcal{P}_1$, and C_{12}^{out} the number of outcoming connections to neurons of $\mathcal{P}_2$ from each neuron of $\mathcal{P}_1$. Generally speaking, $C_{12}^{\text{in}} \neq C_{12}^{\text{out}}$ since the number of neurons of the two populations might differ. However, the outcoming connections from $\mathcal{P}_1$ to $\mathcal{P}_2$ should equate the incoming connections to $\mathcal{P}_2$ from $\mathcal{P}_1$, hence

$$\mathcal{N}_2 C_{12}^{\text{in}} = \mathcal{N}_1 C_{12}^{\text{out}} \tag{8.43}$$

Similarly, we can define C_{22}^{in} and C_{22}^{out}, but since input and output population is the same we should have

$$C_{22}^{\text{in}} = C_{22}^{\text{out}} = C_{22} \tag{8.44}$$

Now let us define $Q_{12}^{\text{out}}(m, h)$ the probability that a neuron of $\mathcal{P}_1$ is selective for m patterns over $\mathcal{T}$ and with h outcoming consolidated connections over C_{12}^{out} to $\mathcal{P}_2$, and $Q_{12}^{\text{in}}(m, h)$ the probability for a neuron of $\mathcal{P}_2$ to be selective for m patterns

over $\mathcal{T}$ and with h incoming consolidated connections over C_{12}^{in} from $\mathcal{P}_1$. Self-connections of $\mathcal{P}_2$ will be discussed considering that input and output populations are the same.
The number of outcoming connections from neurons of $\mathcal{P}_1$ representative for m patterns over $\mathcal{T}$ is

$$H_{12}^{\text{out}}(m) = \mathcal{N}_1 \sum_h h Q_{12}^{\text{out}}(m,h) = N_1 \xi_{12}^{\text{out}}(m) \tag{8.45}$$

where

$$\xi_{12}^{\text{out}}(m) = \sum_h h\, Q_{12}^{\text{out}}(m,h) \tag{8.46}$$

The total number of consolidated connections is thus

$$H_{12}^{\text{tot}} = \sum_m H_{12}^{\text{out}}(m) = N_1 \sum_{m,h} h Q_{12}^{\text{out}}(m,h) = \mathcal{N}_1 \langle h_{12}^{\text{out}} \rangle \tag{8.47}$$

This way, we can compute the probability $p_{12}^{\text{out}}(m)$ that a consolidated connection has a presynaptic neuron which is representative of m patterns

$$p_{12}^{\text{out}}(m) = \frac{H_{12}^{\text{out}}(m)}{H_{12}^{\text{tot}}} = \frac{\mathcal{N}_1 \xi_{12}^{\text{out}}(m)}{N_1 \langle h_{12}^{\text{out}} \rangle} = \frac{\xi_{12}^{\text{out}}(m)}{\langle h_{12}^{\text{out}} \rangle} \tag{8.48}$$

Similarly, the number of non-consolidated connections outgoing from neurons of $\mathcal{P}_1$ representative for m patterns is

$$Z_{12}^{\text{out}}(m) = \mathcal{N}_1 \sum_h (C_{12}^{\text{out}} - h) Q_{12}^{\text{out}}(m,h) = \mathcal{N}_1 \zeta_{12}^{\text{out}}(m) \tag{8.49}$$

where

$$\zeta_{12}^{\text{out}}(m) = \sum_h (C_{12}^{\text{out}} - h) Q_{12}^{\text{out}}(m,h) \tag{8.50}$$

The total number of connections is

$$Z_{12}^{\text{tot}} = \sum_m Z_{12}^{\text{out}}(m) = \mathcal{N}_1 \sum_{m,h} (C_{12}^{\text{out}} - h) Q_{12}^{\text{out}}(m,h) = \mathcal{N}_1 (C_{12}^{\text{out}} - \langle h_{12}^{\text{out}} \rangle) \tag{8.51}$$

and thus the probability $q_{12}^{\text{out}}(m)$ that a non-consolidated connection is outgoing from a neuron representative for m patterns is

$$q_{12}^{\text{out}}(m) = \frac{Z_{12}^{\text{out}}(m)}{Z_{12}^{\text{tot}}} = \frac{\mathcal{N}_1 \zeta_{12}^{\text{out}}(m)}{N_1 (C_{12}^{\text{out}} - \langle h_{12}^{\text{out}} \rangle)} = \frac{\zeta_{12}^{\text{out}}(m)}{C_{12}^{\text{out}} - \langle h_{12}^{\text{out}} \rangle} \tag{8.52}$$

The probability that a neuron of $\mathcal{P}_1$ is selective for m patterns is

$$G_1(m) = \sum_{h=0}^{C_{12}^{\text{out}}} Q_{12}^{\text{out}}(m,h) \tag{8.53}$$

and multiplying the latter with $\mathcal{N}_1$ we get the total number of neurons of $\mathcal{P}_1$ representative for m patterns. Remembering that $\mathcal{N}_1\xi_{12}^{\text{out}}(m)$ represents the total number of consolidated connections from neurons of $\mathcal{P}_1$ representative for m patterns, the average number of consolidated connections outcoming from a neuron of $\mathcal{P}_1$ representative for m patterns is

$$\kappa_{12}^{\text{out}}(m) = \frac{\mathcal{N}_1\xi_{12}^{\text{out}}(m)}{\mathcal{N}_1 G_1(m)} = \frac{\xi_{12}^{\text{out}}(m)}{G_1(m)} \tag{8.54}$$

The equation above can be reduced to the following

$$\kappa_{12}^{\text{out}}(m) = C_{12}^{\text{out}}\Big[1-(1-p_2)^m\Big] \tag{8.55}$$

where p_2 represents the probability for a neuron of $\mathcal{P}_2$ of being selective for a pattern.
Using a similar procedure, it is possible to obtain the average number of consolidated connections incoming to a neuron of $\mathcal{P}_2$ representative for m patterns from neurons of $\mathcal{P}_1$:

$$\kappa_{12}^{\text{in}}(m) = C_{12}^{\text{in}}\Big[1-(1-p_1)^m\Big] \tag{8.56}$$

and the average number of consolidated connections to neurons of $\mathcal{P}_2$ coding for m patterns and coming from neurons of the same population is

$$\kappa_{22}(m) = C_{22}\Big[1-(1-p_2)^m\Big] \tag{8.57}$$

Notice that in this case, the average number of incoming or outcoming consolidated connections is the same.
Previous expressions show that the average number of consolidated connections, both incoming and outcoming, is higher for neurons representative of more patterns (i.e., higher m), as anticipated.
Now that we have an expression for the average number of consolidated connections coming from different neuron populations, we can derive the values of $\mathcal{S}_\text{b}$ and $\mathcal{S}_2$. We use the following notation to refer to the average input rates for the self-connections of $\mathcal{P}_2$:

- $r_{l,2}^{\text{nc}}$: average input rate to non-consolidated connections outgoing from non-representative neurons for the current pattern;
- $r_{l,2}^{\text{c}}$: average input rate to consolidated connections outgoing from non-representative neurons for the current pattern;
- $r_{h,2}^{\text{nc}}$: average input rate to non-consolidated connections outgoing from representative neurons for the current pattern;
- $r_{h,2}^{\text{c}}$: average input rate to consolidated connections outgoing from representative neurons for the current pattern;
- r_I: average input rate of inhibitory neurons.

The average input to a non-representative neuron of $\mathcal{P}_2$ having κ_{22} consolidated indegrees from the same population and κ_{12} consolidated indegrees from neurons of $\mathcal{P}_1$ can be computed as discussed in the feed-forward model. The only difference here is the addition of the different average rates for consolidated or non-consolidated connections. Thus, we can compute the background signal $\mathcal{S}_\mathrm{b}$ in input to a neuron of $\mathcal{P}_2$ as follows:

$$\begin{aligned}\mathcal{S}_{b,2}(\kappa_{12},\kappa_{22}) = \mathcal{W}_\mathrm{c}\kappa_{22}r^\mathrm{c}_{\mathrm{av},2} + \mathcal{W}_\mathrm{b}(C_{22}-\kappa_{22})r^\mathrm{nc}_{\mathrm{av},2} + \\ + \mathcal{W}_\mathrm{c}\kappa_{12}r_{\mathrm{av},1} + \mathcal{W}_\mathrm{b}(C^\mathrm{in}_{12}-\kappa_{12})r_{\mathrm{av},1} + \mathcal{W}_\mathrm{IE}C_\mathrm{IE}r_\mathrm{I} + E\end{aligned} \tag{8.58}$$

where $r_{\mathrm{av},1}$ is the average rate of the neurons of $\mathcal{P}_1$, and $r^\mathrm{c}_{\mathrm{av},2}$ is the average rate from other neurons of $\mathcal{P}_2$ that are representative for the current pattern and connected with consolidated connections. Similarly, $r^\mathrm{nc}_{\mathrm{av},2}$ is the average rate from neurons of $\mathcal{P}_2$ connected with non consolidated connections. $\mathcal{W}_\mathrm{IE}$ is the synaptic weight for the inhibitory-to-excitatory connections, and C_IE is the average number of connections to a neuron of $\mathcal{P}_2$ coming from the inhibitory population $\mathcal{P}_i$. Finally, E is the average input signal from other populations. The first two terms of Equation (8.58) represent the contribution due to the self-connections, whereas the third and the fourth are the contributions of $\mathcal{P}_1$ to the input, ergo the same expression that represented $\mathcal{S}_\mathrm{b}$ in the feed-forward model. Eventually, the last two terms are related to external stimulation and inhibitory feedback.
The two average rates $r^\mathrm{c}_{\mathrm{av},2}$ and $r^\mathrm{nc}_{\mathrm{av},2}$ can be written in term of the rate previously listed (i.e., $r^\mathrm{c}_{h,2}$, $r^\mathrm{c}_{l,2}$,$r^\mathrm{nc}_{h,2}$ and $r^\mathrm{nc}_{l,2}$) following the expression:

$$\begin{cases} r^\mathrm{c}_{\mathrm{av},2} = p_2 r^\mathrm{c}_{h,2} + (1-p_2)r^\mathrm{c}_{l,2} \\ r^\mathrm{nc}_{\mathrm{av},2} = p_2 r^\mathrm{nc}_{h,2} + (1-p_2)r^\mathrm{nc}_{l,2} \end{cases} \tag{8.59}$$

The average rates have a linear dependence from κ_{12} and κ_{22} because of the fact that in Equation (8.41) we have a term like $\mathcal{S}_{b,2}(\kappa_{12},\kappa_{22}) - S_{\mathrm{th},i}$. The linearity enables us to write the different rates as

$$\begin{cases} r^\mathrm{c}_{l,2} = r_{l,2}(\langle\kappa_{12}\rangle_\mathrm{cons}, \langle\kappa_{22}\rangle_\mathrm{cons}) \\ r^\mathrm{c}_{h,2} = r_{h,2}(\langle\kappa_{12}\rangle_\mathrm{cons}, \langle\kappa_{22}\rangle_\mathrm{cons}) \\ r^\mathrm{nc}_{l,2} = r_{l,2}(\langle\kappa_{12}\rangle_\mathrm{nocons}, \langle\kappa_{22}\rangle_\mathrm{nocons}) \\ r^\mathrm{nc}_{h,2} = r_{h,2}(\langle\kappa_{12}\rangle_\mathrm{nocons}, \langle\kappa_{22}\rangle_\mathrm{nocons}) \end{cases} \tag{8.60}$$

where expectation values refer to presynaptic neurons of consolidated or non-consolidated self-connections of $\mathcal{P}_2$.
To compute $\langle\kappa_{12}\rangle_\mathrm{cons}$ we can refer to Equation (8.56) for $\kappa^\mathrm{in}_{12}(m)$, representing the average number of consolidated connections from neurons of $\mathcal{P}_1$ to a neuron of $\mathcal{P}_2$ representative for m patterns. Starting from that equation, we have to do a weighted average over m taking account of the probability that a consolidated self-connection of $\mathcal{P}_2$ comes from a representative neuron for m patterns, to that

$$\langle\kappa_{12}\rangle_\mathrm{cons} = \sum_{m=0}^{\mathcal{T}} p_{22}(m)\kappa^\mathrm{in}_{12}(m) = \sum_{m=0}^{\mathcal{T}} \frac{\xi_{22}(m)}{\langle h_{22}\rangle}\kappa^\mathrm{in}_{12}(m) \tag{8.61}$$

Analogously, we can derive the expectation value of κ_{12} for presynaptic neurons of non consolidated self-connections

$$\langle\kappa_{12}\rangle_{\text{nocons}} = \sum_{m=0}^{\mathcal{T}} q_{22}(m)\kappa_{12}^{\text{in}}(m) = \sum_{m=0}^{\mathcal{T}} \frac{\zeta_{22}(m)}{C_{22} - \langle h_{22}\rangle}\kappa_{12}^{\text{in}}(m) \tag{8.62}$$

The expressions for $\langle\kappa_{22}\rangle_{\text{cons}}$ and $\langle\kappa_{22}\rangle_{\text{nocons}}$ can be obtained in a similar way:

$$\begin{aligned}\langle\kappa_{22}\rangle_{\text{cons}} &= \sum_{m=0}^{\mathcal{T}} \frac{\xi_{22}(m)}{\langle h_{22}\rangle}\kappa_{22}(m)\\ \langle\kappa_{22}\rangle_{\text{nocons}} &= \sum_{m=0}^{\mathcal{T}} \frac{\zeta_{22}(m)}{C_{22} - \langle h_{22}\rangle}\kappa_{22}(m)\end{aligned} \tag{8.63}$$

After a few calculations (not reported here) it is possible to obtain the expressions for $\langle\kappa_{12}\rangle_{\text{cons}}$, $\langle\kappa_{12}\rangle_{\text{nocons}}$, $\langle\kappa_{22}\rangle_{\text{cons}}$ e $\langle\kappa_{22}\rangle_{\text{nocons}}$ as a function of the probability of being representative for a pattern as a neuron of $\mathcal{P}_1$ or $\mathcal{P}_2$, i.e., p_1 and p_2:

$$\begin{aligned}\langle\kappa_{12}\rangle_{\text{cons}} &= C_{12}^{\text{in}} \frac{1 - (1 - p_1 p_2)^{\mathcal{T}} - (1 - p_2^2)^{\mathcal{T}} + (1 - p_1 p_2 - p_2^2 + p_1 p_2^2)^{\mathcal{T}}}{1 - (1 - p_2^2)^{\mathcal{T}}}\\ \langle\kappa_{12}\rangle_{\text{nocons}} &= C_{12}^{\text{in}}\left[1 - \left(\frac{1 - p_1 p_2 - p_2^2 + p_1 p_2^2}{1 - p_2^2}\right)^{\mathcal{T}}\right]\\ \langle\kappa_{22}\rangle_{\text{cons}} &= C_{22}\frac{1 - 2(1 - p_2^2)^{\mathcal{T}} + (1 - 2p_2^2 + p_2^3)^{\mathcal{T}}}{1 - (1 - p_2^2)^{\mathcal{T}}}\\ \langle\kappa_{22}\rangle_{\text{nocons}} &= C_{22}\left[1 - \left(\frac{1 - 2p_2^2 + p_2^3}{1 - p_2^2}\right)^{\mathcal{T}}\right]\end{aligned} \tag{8.64}$$

These expressions should replace the term κ_{12} and κ_{22} in Equation (8.58) for $\mathcal{S}_{b,2}$ to find the average input signal for non-representative neurons of $\mathcal{P}_2$. The calculation for the average input signal to representative neurons of $\mathcal{P}_2$, i.e., $\mathcal{S}_{2,2}$ can be obtained using similar techniques and properly replacing the rate contributions due to the self-connections of $\mathcal{P}_2$.

Chapter 9

Learning through structural plasticity: firing rate-based network simulations

Summary

This chapter presents the simulations of the network of firing-rate-based neurons designed following the theoretical framework described in the previous chapter with the aim of validating the theoretical estimations. The simulation framework has been formalized by Bruno Golosio, me, and Luca Sergi, a Master's student. Luca Sergi and I performed the simulations and cured the data analysis.

The validation of the equations derived in the previous chapter is done through simulations with firing-rate-based neuronal network models. The code of the simulator was written in the `C++` programming language and was compiled with GCC (`https://github.com/gcc-mirror/gcc`) (version 10.2.0) and with GSL (`https://www.gnu.org/software/gsl/`) (version 2.7) scientific libraries. The simulations have been performed using the supercomputers Galileo 100 and JUSUF [121]. The networks used for the simulations are generated according to the selected connection rule. In particular, in the case of the fixed-indegree rule, $\mathcal{C}$ incoming connections are created for each neuron of the $\mathcal{P}_2$ population, where $\mathcal{C}$ has a fixed value. In the case of the Poisson-indegree rule, for each neuron of the population $\mathcal{P}_2$ the number of incoming connections C is extracted from a Poisson distribution with mean $\langle C \rangle$. In both cases, the indices of the presynaptic neurons are randomly extracted on the $\mathcal{P}_1$ population. The connection weights are initially set to the baseline value, $\mathcal{W}_\mathrm{b}$. Each training input pattern of the discrete model is generated by extracting, for each neuron of $\mathcal{P}_1$, a random number r from a uniform distribution in the interval $[0, 1]$; if $r < p_1$ the rate of the neuron is set to the high level, ν_h, otherwise it is set to ν_ℓ. An analogous procedure is used to generate the corresponding contextual stimulus pattern on the neurons of the population $\mathcal{P}_2$. A connection is consolidated in a training example if both the presynaptic and the postsynaptic neuron are in the high-rate level, ν_h. In the continuous case, the firing

rates of the input pattern and those of the contextual pattern are extracted from a log-normal distribution. In this case, a connection is consolidated if the firing rates of both the presynaptic and postsynaptic neurons are above the thresholds $\nu_{t,1}$ and $\nu_{t,2}$, respectively. Connection rewiring is performed every s training steps, as described in Section 8.4.2. The test set is generated by randomly extracting V input patterns from the train set. In the discrete case, the patterns are not modified. In the continuous case, the patterns of the test set are altered by adding noise extracted from a truncated Gaussian distribution.
To estimate $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ we compute the input of each $\mathcal{P}_2$ neuron as the sum of the rate of the presynaptic neurons of its incoming connections multiplied by the synaptic weights (i.e., $\mathcal{W}_\mathrm{c}$ or $\mathcal{W}_\mathrm{b}$). The variance σ^2_b is evaluated by the formula

$$\sigma^2_\mathrm{b} = \langle \mathcal{S}^2_\mathrm{b} \rangle - \langle \mathcal{S}_\mathrm{b} \rangle^2 \tag{9.1}$$

where the mean values are calculated over the input signals to all non-coding neurons of $\mathcal{P}_2$.

9.1 Two neuron population simulations

This section, presented also in [5], compares the results of the simulations of the firing rate model with the theoretical predictions described in the previous chapter. Since we proposed several versions of the model, with different features implemented, the section is divided into different parts that sum up the main characteristics of the model. We present the results of the approaches employing discrete and continuous values for neuron firing rates, comparing the theoretical values of the average input signal to background neurons $\langle \mathcal{S}_\mathrm{b} \rangle$, the average input signal to coding neurons $\langle \mathcal{S}_2 \rangle$, the variance of the background input signal σ^2_b and the signal-difference-to-noise ratio with the values obtained from the simulations. This way, we are able to assess the capacity of the population $\mathcal{P}_2$, and thus of the network, to recognize a pattern memorized in the training phase. Here, we present simulation results with a Poisson-driven number of incoming connections, with $\langle \mathcal{C} \rangle = 5000$. We opted for such an approach since it is more realistic for biological neural circuits with respect to a fixed amount of connections per neuron. Each simulation is repeated 10 times using a different seed for random number generation to ensure the robustness of the simulation results. The values shown in the plots are a result of averaging over the different seeds.

9.1.1 Comparison between continuous and discrete firing rate

As previously mentioned, the main difference in calculating $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ with the continuous rate model versus the discrete model is that the discrete values of ν_l and ν_h are replaced, respectively, by the average values of the rate below and above threshold, calculated on the continuous probability distribution. On the other hand, the variance of the background signal differs in the two models, because it depends on the variance of the rate, σ^2_ν, which is different in the two cases.

The first study we present is oriented towards the estimation of these parameters as a function of the number of training patterns $\mathcal{T}$. As the number of training patterns increases, so does the number of patterns encoded by each individual neuron. Since p_2 is the probability that a neuron of $\mathcal{P}_2$ is in a high-rate level for a single training pattern, on average such neuron will encode $p_2\mathcal{T}$ patterns of the entire training set. This multiple selectivity of individual neurons is also present in biological neural networks, in which the same neuron can be selective for several stimuli [152].
The test set consists of $V = 1000$ input patterns, generated as described in Section 8.2. Thus, the simulation outcome used for our analysis is an average over the entire test set of the $\mathcal{S}_\mathrm{b}$, $\mathcal{S}_2$, σ^2_b, and SDNR values obtained for each test pattern. Figure 9.1 shows the comparison of the simulation outcomes using discrete and continuous rate values.

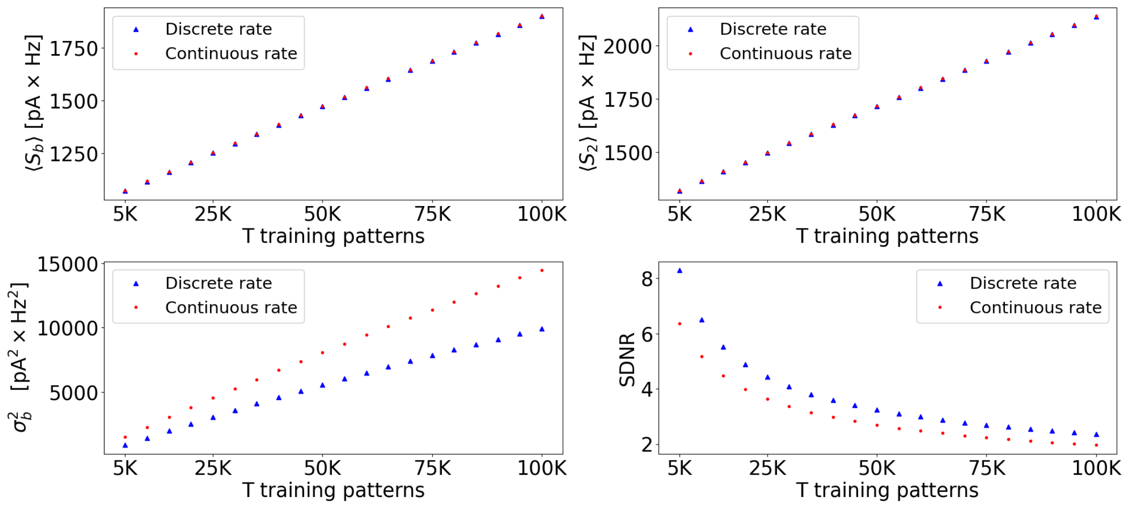


Figure 9.1: Comparison between $\langle\mathcal{S}_\mathrm{b}\rangle$, $\langle\mathcal{S}_2\rangle$, σ^2_b, and SDNR obtained from simulations using discrete (blue triangles) or continuous (red dots) firing rate distribution. The values are given as a function of the number of training patterns $\mathcal{T}$. Background and coding signals are expressed in the unit pA $\times$ Hz since the input received from the neurons of $\mathcal{P}_2$ is defined by the product between the firing rate of the presynaptic neurons (in Hz) and the synaptic weight (in pA, since the input received by a neuron is an electric current).

We can see that the curves of $\langle\mathcal{S}_\mathrm{b}\rangle$ and $\langle\mathcal{S}_2\rangle$ obtained from the simulations using the continuous firing rate distribution are superimposed on those obtained using the discrete model; this is due to the fact that the choice of the threshold on the log-normal distribution is done so that the average values for low and high rates, $\langle\nu_l\rangle$ and $\langle\nu_h\rangle$, correspond to the values adopted in the discrete rate model. On the other hand, the variance of the background signal σ^2_b differs in the two models because it depends on the variance of the firing rate, σ_ν, which is different in the two cases. This leads also to the different behavior of the SDNR.

9.1.2 Comparison between theoretical predictions and simulation results

The results shown in the previous section are obtained from simulations. This section presents a comparison between simulation results and theoretical expectations. To provide a quantitative estimation of the discrepancy between the theoretical predictions and the simulations, we evaluate their relative error, using the theoretical values as a reference.

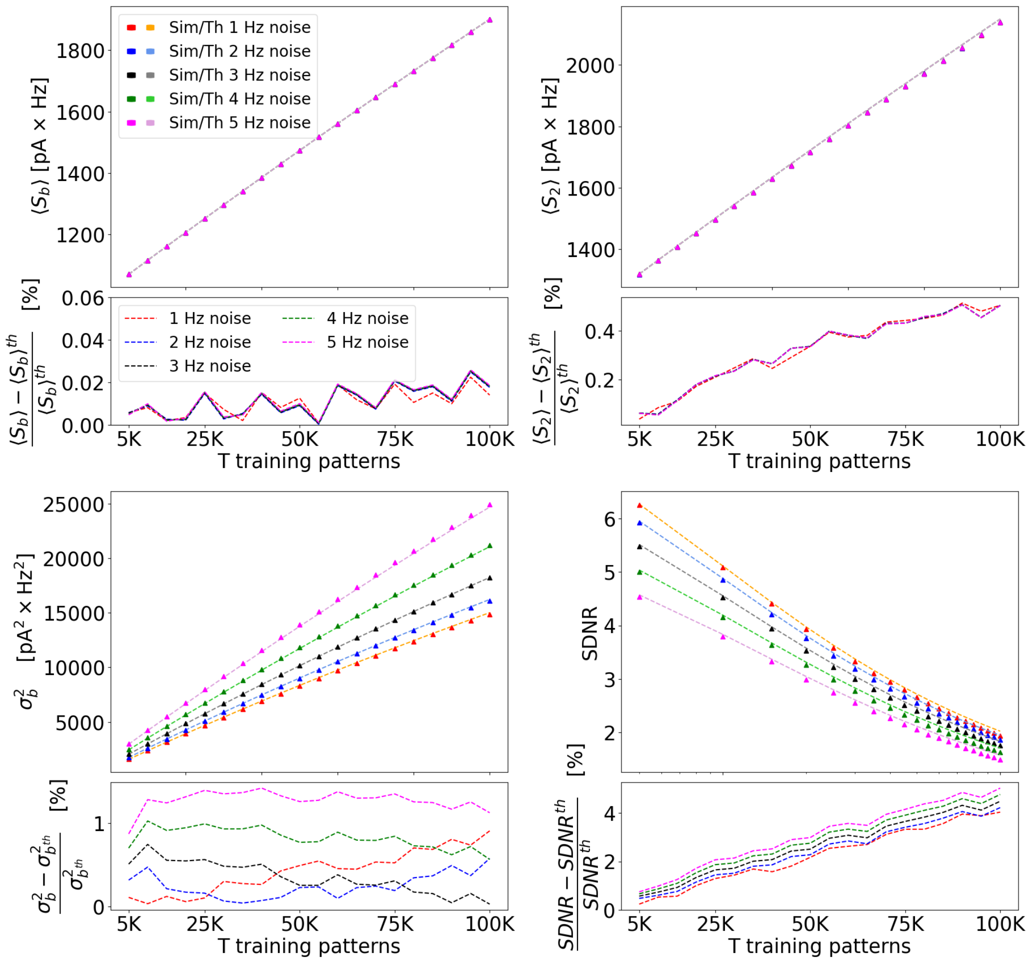


Figure 9.2: Values of $\langle \mathcal{S}_b \rangle$, $\langle \mathcal{S}_2 \rangle$, σ_b^2 and the SDNR and percent error with respect to the theoretical prediction, as a function of the number of training patterns $\mathcal{T}$. Upper subplots represent the values of the quantities considered as a function of $\mathcal{T}$ for different noise levels, whereas each lower subplot represents the percentage error of the values shown in the upper subplot. Simulation data is represented with triangles, whereas the theoretical predictions are depicted with dotted lines. The different color families identify the simulation and theory results when having a noise standard deviation of 1 Hz (red-orange), 2 Hz (blue-light blue), 3 Hz (black-grey), 4 Hz (green-light green) and 5 Hz (magenta-pink).

As previously described, the test input patterns used in the continuous rate model are altered from the corresponding training input patterns by adding noise extracted from a truncated Gaussian distribution, with assigned standard deviation. In this section, we present simulation results and comparisons with theoretical predictions for standard deviation values ranging from $1\,\mathrm{Hz}$ to $5\,\mathrm{Hz}$. These values are relatively high in relation to the average firing rate of $\mathcal{P}_1$, which is slightly higher than $2\,\mathrm{Hz}$.
Figure 9.2 shows the curves obtained for the continuous rate model using different values for the standard deviation of the noise, together with the relative error with respect to theoretical predictions.
It can be observed that the curves obtained from the simulations are compatible with the theoretical ones for all the noise levels.
Regarding $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$, the curves corresponding to different noise levels appear perfectly superimposed. This is due to the fact that the noise is driven by a distribution with zero mean, and thus the addition of noise to the quantities represented in the curves does not alter their average (see Appendix G for the details). Regarding σ^2_b, the values corresponding to different noise levels differ from each other and increase with the standard deviation of the noise, in agreement with the theoretical model.
The relative error between simulation results and theoretical prediction is quite small: for $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ the errors span between 0.01% and 0.4%, whereas σ^2_b shows a relative error of around 1% for all the simulations performed with a different number of training patterns.
The addition of noise with fluctuations greater than or comparable to the average firing rate can produce negative rate values for a fraction of the neurons. Considering that negative rate values are not physically possible, this behavior can be corrected in the simulations by simply replacing negative values of the firing rates with zero, i.e. saturating negative rates to zero. This correction is equivalent to having a different noise distribution, with an average value greater than zero. However, the current theoretical model is not able to take this effect into account. Since negative values are replaced by zeros, we would expect the average values of $\mathcal{S}_\mathrm{b}$ and $\mathcal{S}_2$ evaluated by the simulations that exploit saturation to be greater than the values predicted by the theoretical model. Figure 9.3 shows the behavior of the model with this correction on the neurons' firing rate.
As can be seen from the figure, the discrepancies between simulations and theoretical predictions are much higher and can arrive at 40%. This is due to the fact that the current theoretical framework is not able to take this correction into account. However, it should be considered, as discussed above, that the noise levels are relatively high when compared with the average rate used in these simulations. Indeed, a different choice for the values of $\langle \nu_\ell \rangle$ and $\langle \nu_\mathrm{h} \rangle$ (and thus a different average rate of the whole distribution) would have an impact on the discrepancies shown here. In particular, a higher average rate would strongly diminish the amount of neurons having negative firing rate as a result of the noise addition.
The relative error of σ^2_b is greater than that shown for $\langle \mathcal{S}_\mathrm{b} \rangle$ and $\langle \mathcal{S}_2 \rangle$ in Figure 9.2; this is due to a simplification used in the theoretical model to derive the expression of the variance. The values of $\mathcal{S}_\mathrm{b}$ from which we compute the variance are

obtained by incoming connections from neurons of $\mathcal{P}_1$, but since connections are created randomly, different neurons of the $\mathcal{P}_2$ population may have presynaptic neurons in common, and therefore their input signals are correlated. The theoretical model does not take this correlation into account. The average number of presynaptic neurons in common to two arbitrary neurons of $\mathcal{P}_2$ depends on the total number of neurons of $\mathcal{P}_1$ and on the number of incoming connections per neuron of $\mathcal{P}_2$.

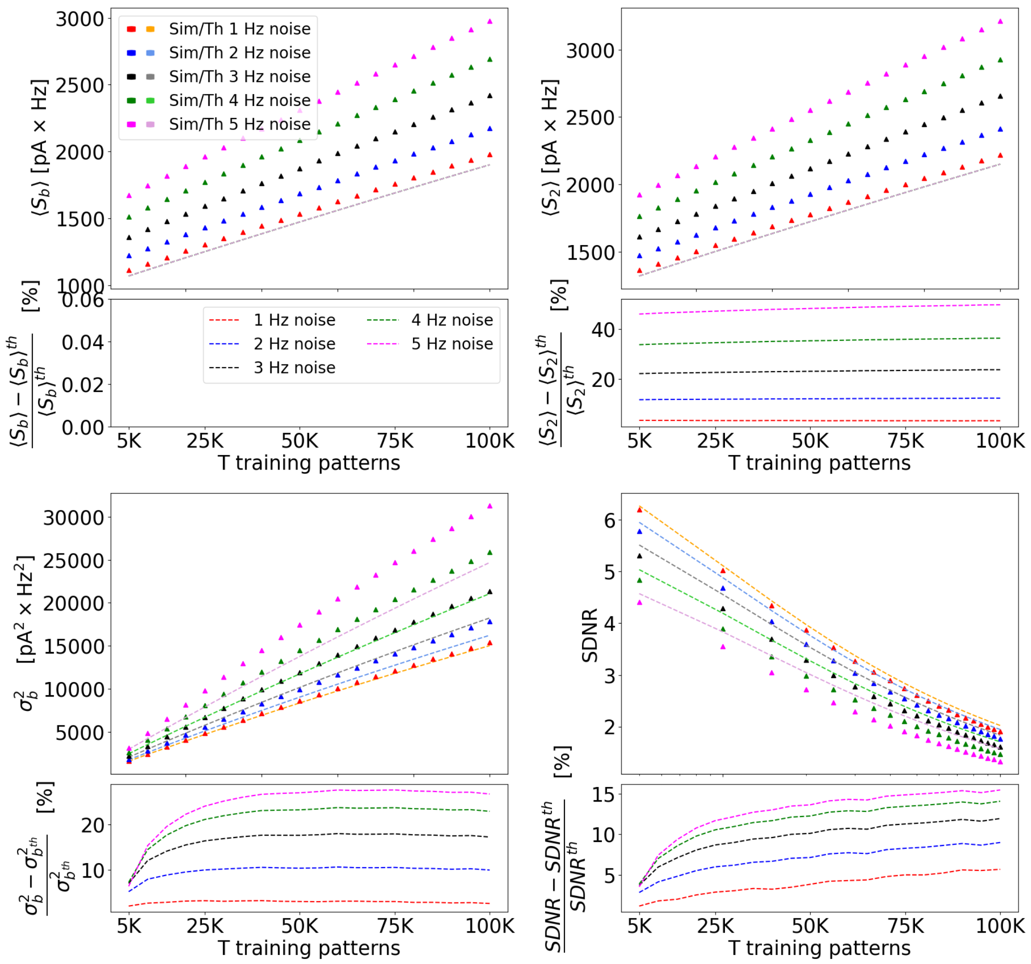


Figure 9.3: Values of $\langle\mathcal{S}_\mathrm{b}\rangle$, $\langle\mathcal{S}_2\rangle$, σ_b^2 and SDNR, and percent error with respect to the theoretical predictions, as a function of the number of training patterns $\mathcal{T}$ when negative rates due to noise addition are saturated to zero. Simulation data is represented with triangles, whereas the theoretical predictions are depicted with dotted lines. The different color families identify the simulation and theory results when having a noise standard deviation of $1\,\mathrm{Hz}$ (red-orange), $2\,\mathrm{Hz}$ (blue-light blue), $3\,\mathrm{Hz}$ (black-grey), $4\,\mathrm{Hz}$ (green-light green) and $5\,\mathrm{Hz}$ (magenta-pink).

Calling $\mathcal{N}_1 = \mathcal{N}$, we can state that the bias due to this simplification becomes more and more relevant when the ratio $\mathcal{C}/\mathcal{N}$ increases. In order to estimate this bias as a function of the $\mathcal{C}/\mathcal{N}$ ratio, we performed a series of simulations with a

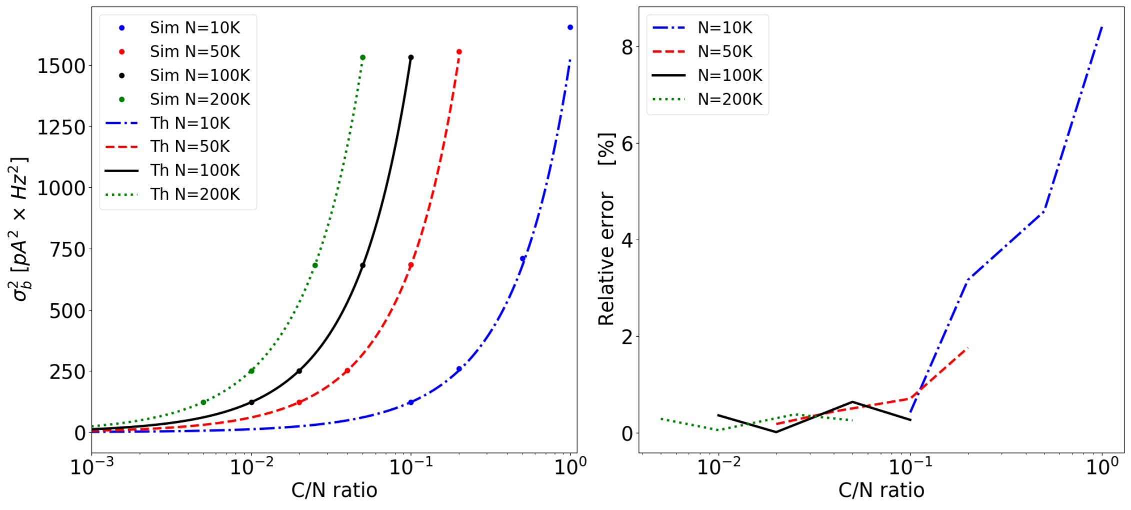


Figure 9.4: Values of σ_{b}^2 as a function of the $\mathcal{C}/\mathcal{N}$ ratio for different values of $\mathcal{N}$. On the left panel, lines represent the theoretical prediction (Th), whereas dots represent the values obtained from the simulation (Sim). On the right panel, dotted lines represent the relative error between simulation results and theoretical prediction.

fixed number of training patterns, $\mathcal{T} = 5000$, changing the $\mathcal{C}/\mathcal{N}$ ratio. Figure 9.4 shows the results of this analysis.

As can be seen in the right panel of Figure 9.4, a greater value of $\mathcal{C}/\mathcal{N}$ leads to a higher discrepancy between theoretical prediction and simulation. However, such a ratio, for natural density circuits in the brain, is very far from values of $\mathcal{C}/\mathcal{N}$ near unity. Indeed, a plausible value of the ratio would be less than 0.1, resulting in negligible relative errors.

9.1.3 Impact of synaptic rewiring

In the simulations discussed so far, the rewiring mechanism was always performed with a rewiring step $s = 100$. This means that every 100 training patterns, all the unconsolidated connections are removed, and new connections are created. This operation represents the effect of homeostatic structural plasticity, which aims at keeping the network balanced by reorganizing connections, while activity-dependent structural plasticity focuses on the consolidation of connections.

To motivate the choice of this step for connection rewiring, we show here the results for networks trained for $\mathcal{T} = 5000$ patterns with a different rewiring step s. We also show the results of a simulation that does not perform rewiring, in order to highlight the different behavior of a network that combines connection consolidation with periodic rewiring and that of a network that exploits only connection consolidation. Figure 9.5 shows the results obtained by these simulations using different rewiring intervals.

As can be noticed, the values of $\mathcal{S}_{\mathrm{b}}$, $\mathcal{S}_2$, σ_{b}^2, and SDNR do not change significantly as the rewiring step varies. This means that the value of the step s chosen for the connection rewiring has no impact on the results of the simulations. On the other

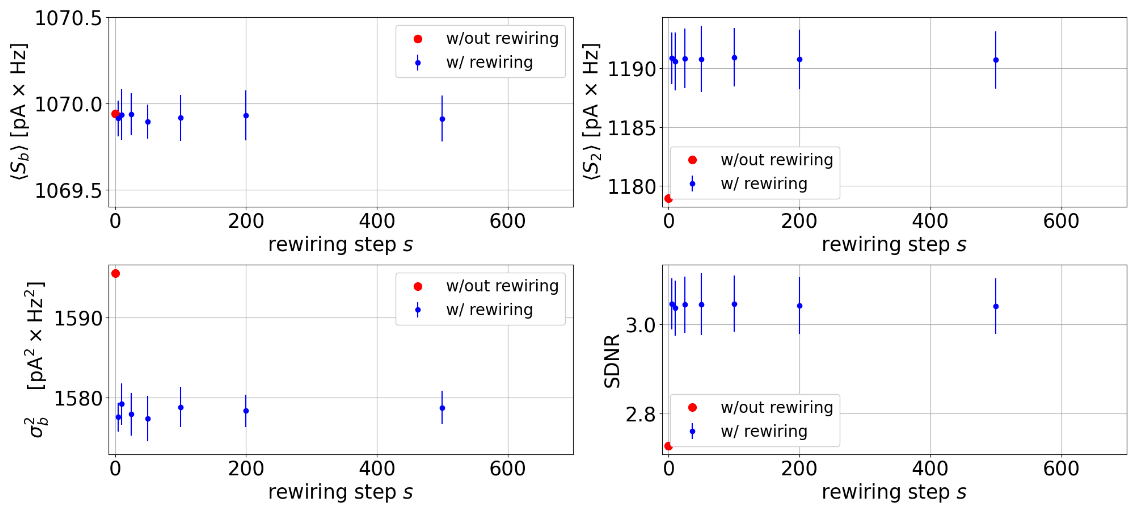


Figure 9.5: Values of $\langle \mathcal{S}_\mathrm{b} \rangle$, $\langle \mathcal{S}_2 \rangle$, σ_b^2 and SDNR for a network trained with 5000 patterns as a function of the rewiring step s. The simulations used a continuous rate distribution and a noisy input driven by truncated Gaussian distribution with a standard deviation of $1\,\mathrm{Hz}$. The red dot indicates the simulation outcome when connection rewiring is disabled, whereas the blue dots show the simulation results with connection rewiring, using different values of s. Blue error bars represent the standard deviation of the mean obtained from 10 simulations using different seeds for random number generation.

hand, significant differences emerge when comparing the results of simulations with or without connection rewiring; it can be observed that the signal-difference-to-noise ratio has a lower value when the rewiring is disabled. This confirms that connection rewiring grants a higher capability of recognizing an input pattern among the several patterns for which the network was trained.

We also applied a similar protocol for simulations enabling or disabling connection rewiring as a function of the number of training patterns $\mathcal{T}$. The results are shown in Figure 9.6.

We can say that the performance of the model is improved when connection rewiring is enabled, and the relative difference between a rewired or just consolidated connectivity increases when the number of training patterns increases as well.

The effect of rewiring can become more relevant when a greater number of connections is consolidated at every step (i.e., with greater values of p_1 and p_2). Furthermore, the importance of the rewiring mechanisms can significantly change when the average number of connections is not constant but increases or decreases as a result of rewiring itself. This aspect will be explored in future work.

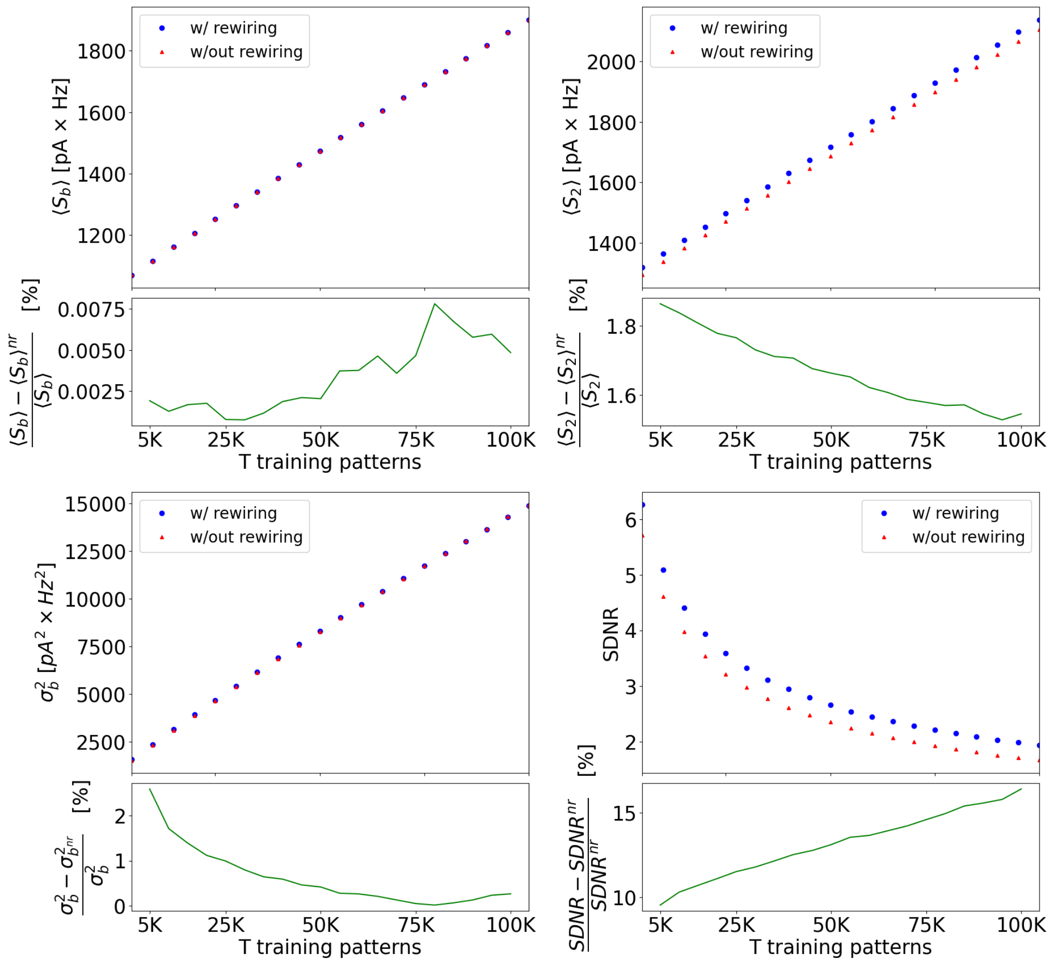


Figure 9.6: Comparison between simulations exploiting the connection rewiring mechanism performed every 100 training patterns (blue dots) and simulations without rewiring (red triangles) for a network trained with a variable number of training patterns $\mathcal{T}$. Green lines in the lower panels show the difference between the values obtained with connections rewired or not rewired (indicated with nr) as percentage error.

Discussion

This thesis covers several aspects of computational neuroscience, from simulation technology to a better understanding of the synaptic mechanisms underlying high-level cognitive processes. To better organize the discussion of the results presented in this thesis, different topics are presented in different sections.

GPU-based simulations using NEST GPU

In Part II of this thesis we focused on the simulation of spiking networks accelerated by the GPU simulation code NEST GPU. In the context of this thesis, we implemented network models such as the cortical microcircuit of [12] and the multi-area model of [88, 89] and we validated the results of the network activity with the ones of the spiking network simulator NEST through the comparison of some relevant statistical distributions. The validation protocol is described in Chapter 4 and the results of the validation of these models are shown in Chapter 5, showing optimal compatibility between NEST GPU and NEST. In particular, the validation protocol aims to perform a statistical comparison between the fluctuation in some distributions of the spiking activity, which have been employed in several other works (see, for example, [74, 86, 97, 99]). First, the fluctuations within NEST simulations (i.e., when different seeds for random number generation are employed) and with NEST and NEST GPU are evaluated. Comparability of these fluctuations would lead to the conclusion that the simulation on a different simulator does not add additional variability in the simulation of the neural activity. The results shown in Chapter 5 show an optimal compatibility with the statistical distributions of the neural activity of NEST, both in the simulation of the cortical microcircuit model and in the simulations of the multi-area model. This is a theme of fundamental importance for better integration of NEST GPU in the software of the NEST Initiative. Indeed, this work focuses also on the integration of neuron, synapse models and connection rules already integrated in NEST. Furthermore, to enable the users to take advantage of all the features of the library, a documentation similar to the one of NEST has been written and periodically updated.
Another relevant topic is the evaluation of the performance of NEST GPU. To better study the different simulation stages, we divided the total time-to-solution into two phases: network construction (i.e., the time needed to create and organize the network before the simulation of the dynamics) and simulation (i.e., the actual simulation of the network dynamics). Indeed, these phases can be further divided into different stages, as described in Chapter 6. Regarding network construction time, this thesis and [4] presented a novel algorithm for network construction, which dynamically creates the network exploiting the high degree of parallelism of GPU devices. In the first version of the library (see [1]), connections were first created on the CPU side and then copied from the RAM to GPU memory. This approach benefited from the standard C++ libraries and, particularly, the dynamic allocation of container classes of the C++ Standard Template Library. However, it had the drawback of relatively long network construction times, not only due to the costly copying of connections and other CPU-side initialization, but also because the connection creation process was performed serially. The new approach,

so far applied to single-GPU simulations, enables much faster connection creation, initialization, and organization while preserving the advantages of dynamic connection building, particularly the ability to create and initialize the model at runtime without the need for compilation. With the current version, it is possible to construct a network such as the cortical microcircuit model of [12] in around half of a second, two orders of magnitude less than the previous method. Comparing this result with other simulation codes, NEST GPU network construction times are also shorter compared to the CPU version of NEST and the code generation framework GeNN (see Figure 6.2); if code generation and compilation are not required in GeNN, the results of NEST GPU and GeNN are compatible. We also verified that the time to simulate the network dynamics after network construction is not compromised by the novel approach. Regarding the network dynamics, Figure 6.4 shows the improvement in the simulation performance of the full-scale cortical microcircuit model of [12] between the first version of the GPU library (i.e., NeuronGPU [1]) and the most recent version of the library employed in [4] using both data center and consumer GPUs. We can notice that the optimization performed on the library on many aspects of the code led to an improvement in the simulation performance spanning from 55% to 75% depending on the GPU hardware employed. In particular, the best result is achieved with the NVIDIA RTX 4090, with a real-time factor, averaged over 10 simulations with different seeds, of 0.4707 ± 0.0008 [4]. This result is obtained with the external drive of the model being a Poisson signal generator. Figure 6.5 shows the real-time factor when a DC input is given to the network instead of the Poisson stimulation, obtaining a real-time factor of 0.386 ± 0.001 when employing the same GPU. We can thus notice that in both the configurations, NEST GPU can perform sub-realtime simulation of this model, even outperforming the CPU simulator NEST in an optimal parallelization configuration (see Figure 6.5A and [73] for additional details). Moreover, the same Figure shows the results for the simulation of the same model performed using the GPU code GeNN (version 4.8.0), which mimics the incoming Poisson spike trains via a current directly applied to each neuron. Thus, the implementation of the external stimulation is not the same as the one of NEST or NEST GPU. However, it can be seen that GeNN currently simulates faster than NEST GPU. Indeed, this indicates that there is room for improvement, which could be exploited via further parallelizations of the simulation kernel of NEST GPU. Moreover, the algorithm for runtime network construction directly on GPU devices is being extended in order to be exploited into multi-GPU simulations. This would lead to a further increase in NEST GPU performance on network construction for large-scale models that have to be simulated on MPI-GPU clusters.
Indeed, NEST GPU was also designed to perform an efficient algorithm for spike delivery and communication across different compute nodes via MPI-GPU clusters. Furthermore, NEST GPU distributes neurons across MPI processes exploiting their locality, i.e., an entire neuron population is created and simulated on a single MPI process, to be specified by the user. In this regard, in [2] we validated NEST GPU with the simulation of the multi-area model of [88, 89] describing the dynamics of the vision-related areas of the macaque cortex. The results of the validation (see Chapter 5 and Appendix B) show optimal compatibility with the results ob-

tained with the NEST simulator, both for the ground and in the metastable state of the model. Moreover, Figure 6.6 shows the real-time factor of the CPU and GPU code of NEST in both the ground and metastable state, noticing that the locality exploitation of NEST GPU is one of the major contributors to the better performance of NEST GPU with respect to NEST, in which neurons are distributed in a round-robin fashion. Indeed, both these approaches show advantages and disadvantages: in NEST every MPI process handles the same amount of compute load, whereas in NEST GPU the compute load differs (see for example Figure 6.7). However, in the case of the multi-area model, the locality exploitation is a more efficient approach since the vast majority of the spikes emitted in a simulation of the network is exchanged between neurons belonging to the same area, i.e., the same MPI process according to the NEST GPU approach. This way, only a fraction of the spikes have to be delivered remotely. In summary, NEST GPU is able to perform simulations of spiking networks taking advantage of both data center and consumer GPUs. The library achieves network construction times comparable to or shorter than those obtained with other state-of-the-art simulation technologies while still meeting the flexibility of runtime network construction. The library is able to take advantage of multi-GPU systems and thanks to an efficient algorithm for spike delivery and network locality exploitation, it is particularly suitable in the simulation of modular large-scale models that can not be simulated on a single GPU card.
To be thorough, in [90], it is shown that GeNN was able to simulate the multi-area model previously mentioned on a single GPU by exploiting the procedural connectivity approach mentioned in Chapter 3. With this method, the network generates the network connectivity on the go, without saving data on the GPU memory and thus being able to simulate a larger network on a single card. However, such a network would not be suitable when plastic connections have to be employed, thus is not applicable in cases in which learning or the interplay between synaptic changes and brain dynamics are of interest (e.g., in [150, 151, 153]).
Without any doubt, NEST GPU has room for improvement in several aspects, from additional optimization of the simulation phase (e.g. increasing the level of parallelization of several subtasks) to further integration of the library into the NEST Initiative. A further alignment to the NEST simulator is needed to enable the same code to be run on both the CPU and GPU simulators, and also additional neuron, synapse models and connection rules have to be implemented on NEST GPU to have an appropriate variety and flexibility. Moreover, the support of NESTML, a domain-specific language that allows the modification or the creation of neuron or synapse models, would be fundamental for implementing new models in a simple format. In addition, to fully benefit from the features of the library, updated documentation is needed to guide the user in the choice of the most appropriate neuron, synapse models and parallelization approaches.
Finally, from a general perspective, recent advances in GPU technology and the development of exascale MPI-GPU clusters would surely contribute to an increase in the performance on the simulation of large-scale spiking network models, but most importantly would open to the development of even larger network models, with several millions of neurons and billions of synapses. In this regard, with NEST

GPU we provided a tool able to exploit the most recent technology, paving the way for more and more detailed network models, that could be able to explore, with a neuron resolution, the dynamics of brain networks handling high-level cognitive processes.

Short-term plasticity and working memory

In Part III of this thesis we focused on the study of synaptic mechanisms underlying cognitive processes. In Chapter 7 we present a spiking network model able to show a typical working memory behavior only by implementing short-term synaptic plasticity in the excitatory-to-excitatory synapses. The hypothesis of a synaptic theory of working memory was initially presented in the work of Mongillo [23], together with the simulations of a spiking network that we reproduce in the context of this thesis to provide a well-documented spiking model implemented on a widely used spiking network simulator (i.e., NEST). To this aim, we edited the implementation of the short-term plasticity model of NEST (more details on Appendix C) and we propose a similar network model based on the instructions provided in the original work. Despite some differences in the implementation and in the value of the parameters, we were able to qualitatively reproduce the results of [23], and we were also able to estimate the working memory capacity according to the work of [39], in which a similar network model was employed to estimate the working memory capacity as a function of synaptic and neuron parameters. As discussed in Section 7.3, this work aims to pave the way for further studies on the relation between synaptic processes such as short-term plasticity and working memory. In this regard, such studies can shed light on the mechanisms that contribute to high-level cognitive processes.
Furthermore, the model reproduced in this thesis, in which the only ingredient to obtain a memory-specific response is an STP-modulated synaptic efficacy and prior long-term Hebbian learning, has several limitations. For instance, the previously mentioned prior learning process is needed, and the synapses can assume only two values of absolute synaptic efficacy (i.e., J_b or J_p). A more realistic network would implement a proper training protocol in order to mimic a realistic learning process, with connections having heterogeneous synaptic strengths. Additionally, the STP variables u and x have the same initial values for all the connections of the network. An interesting work would thus add heterogeneity to synapse parameters, from synaptic weights to STP time constants and variables, to provide the results of a more realistic network able to underlie working memory. Indeed, a study related to a different model relying on persistent activity, revealed that heterogeneity in neurons and synapse parameters is able to prevent the appearance of persistent activity [154], showing that persistent activity regime can be restored when some compensating mechanisms are added (e.g., homeostatic synaptic plasticity, an increase in the number of neurons or in the external input). A similar study can thus be interesting for our model, in particular adding heterogeneity to synaptic efficacies and STP parameters.
Another limit of the model reproduced in this thesis is related to the fact that once the network enters the regime of persistent population spikes or higher-rate ac-

tivity (see panels B and C of Figure 7.2) it can return to the lower activity state only with a decrease of the background current. Thus, for such an activity to have a selected duration, another mechanism has to be added to the network. Recent work of [155] suggests that, together with neurons, astrocytes can modulate neural activity. Thus, interactions between glia and synapses contribute to synaptic transmission and can underlie multiple forms of working memory. Another interesting work on astrocytes involved in working memory was able to show that astrocytic mechanisms are involved in the duration modulation of the persistent activity state [156]. In particular, this work employed the model of Mongillo [23] with the addition of an astrocytic mechanism with a proper time constant and a parameter that influences the calcium dynamics described by the variable u. This way the network is able to show the population spike or the high-asynchronous activity regime for a limited time, determined by the dynamics of the astrocytic mechanism.
As we can see, the quest to account for synaptic mechanisms in working memory is leading to several models in which many processes should be taken into account to better model a high-level cognitive process such as working memory. Indeed, the synaptic theory of working memory suggests that activity-silent processes take place to encode chunks for a limited time, whereas the theory proposed at the beginning relied on persistent activity as the key mechanism underlying working memory. Several experimental findings support both these hypotheses, and the idea that both regimes coexist has also been proposed [157, 158]. Novel experiments will be needed to shed light on the neural and synaptic mechanisms underlying this high-level cognitive process, and more detailed computational modeling of different processes that can contribute to working memory will be of fundamental importance to flank experiments and provide tools to further analyze the impact of these mechanisms in working memory.

Structural plasticity and learning

The last chapters of Part III presented a theoretical framework for learning through a structural plasticity mechanism, described also in [5]. The predictions of the theoretical framework, based on a mean-field approach, have been compared with the results of simulations performed using firing-rate-based neuronal networks. The comparison shows that the proposed framework is able to accurately predict the values of various quantities relevant for assessing learning and memory capacity in the presence of structural plasticity mechanisms, taking into account numerous characteristics of biological neuronal networks. The rates of neurons in the training and test patterns can be distributed according to an arbitrary probability distribution. Here, two cases have been considered: a simplified one in which the rates can assume only two values, high rate and low rate, and a more realistic one in which the rates follow a log-normal distribution in agreement with [149]. Connectivity between regions can be achieved through different connection rules, with a fixed number of connections per target neuron or, more realistically, with random connectivity in which the number of incoming connections of each target neuron follows a Poisson distribution.

Since the biochemical and biophysical mechanisms underlying structural plasticity are multiple and extremely complex, we opted for a phenomenological approach to capture their main aspects. This way, we exploited a simple model of structural plasticity able to represent plasticity processes driven by neuronal activity as well as mechanisms that lead to homeostasis, in agreement with [47], which divides structural plasticity mechanisms into these two categories. Structural plasticity driven by neuronal activity is achieved through the consolidation of synapses connecting neurons that are concurrently at a high-rate level. This process can be triggered by other forms of plasticity that modify synaptic efficacy, such as STDP, followed by mechanisms involving cytoarchitectural changes, such as the creation of novel connections next to the already existing ones.

The homeostatic form of structural plasticity involves a balance between pruning connections that are not utilized over time and the creation of novel connections. This is achieved in the simulations through periodic connection rewiring, which consists of the removal of unconsolidated connections followed by the creation of new connections. The results show that connection rewiring leads to an increase in SDNR, conducting to a higher capability of recognizing the input patterns when this mechanism is enabled.

In order to evaluate the generalization capability of the framework with the continuous firing-rate distribution model, the test patterns were generated by altering the training input patterns through the addition of noise from a given probability distribution and assigned standard deviation. In particular, a truncated Gaussian distribution has been used for the noise. The results of the simulations are compatible with the theoretical predictions, with differences in the order of 1–2%, which is a remarkable result for our purpose.

However, such an approach can lead to a fraction of the neurons with a negative firing rate as a result of noise addition. Since negative firing rates are not physically possible, a more realistic model would apply a saturation to zero for these rates. Saturation can be activated in the neuronal network simulations, but the current version of the theoretical framework is still unable to account for it. This leads to a discrepancy between the simulations with saturation turned on and the theoretical predictions, which grows as the noise increases and becomes relevant when the noise gets significantly greater than the average firing rate. Future work on this model should be devoted to the development of a theoretical framework capable of taking into account the saturation of negative firing rates.

Another limitation of the framework comes from a simplification in the calculation of the background signal variance, which does not take into account the correlations between the contributions of presynaptic neurons to the input signals to distinct neurons of the target population. However, the results show that the impact of this simplification is very small, at most in the order of a few percent.

The theoretical model can be surely extended. It can potentially provide a powerful tool to describe the impact of structural plasticity in cognitive processes such as learning in a large-scale model of the cortex with natural density and plausible characteristics. For instance, the consolidation mechanism can be probability-driven, with a probability depending on the rate of pre-and postsynaptic neurons. This would replace the current deterministic mechanism that requires a firing rate

threshold to be exceeded by both neurons to have synaptic consolidation. Moreover, the probability could depend on other variables not necessarily related to the firing rate. In particular, it has been hypothesized that plasticity mechanisms may also depend on the bursting activity of neurons [159,160]. The consolidation probability of a connection could therefore depend, in addition to the firing rate of the presynaptic and postsynaptic neurons, also to their bursting activity. In this regard, it would be interesting to expand this work through simulations of spiking neural networks. Indeed, simulators such as NEST [64] and its GPU implementation [2, 4] can lead to fast and efficient simulations of large-scale models on supercomputer clusters.
In Chapter 8 we have mainly considered the connections between two distinct populations $\mathcal{P}_1$ and $\mathcal{P}_2$. However, the proposed framework also lends itself to the study of structural plasticity in the self-connections of population $\mathcal{P}_2$, together with an inhibitory population in order to have a more realistic architecture of excitatory and inhibitory neurons. Indeed, it is known that the mechanisms of competition through lateral inhibition play a key role in biological learning [130]. Section 8.5 of this thesis is devoted to the description of such a recurrent network model. In this extended model, the theoretical framework allows to obtain the differential equations governing the dynamics of the activity of the population $\mathcal{P}_2$ and the dependence of the coefficients of these equations on the number of training patterns and on the other model parameters [161]. Such an extension is currently under development and additional work has to be done to provide an extensive description of all the features of this model and for an appropriate choice of simulation parameters.
Another extension of the model could describe more in detail the mechanisms of synaptic pruning and rewiring. Indeed, connection rewiring as intended in the feed-forward model preserves the total number of connections over time, which is a typical behavior of a healthy adult brain [52]. However, to shed light on the importance of this mechanism in neurological disorders, or to perform studies focused on this mechanism in different life stages, this mechanism should be extended to enable different "speed" for the processes embedded in structural plasticity.
In conclusion, this work intends to provide a sufficiently general theoretical framework for learning through structural plasticity. This framework is able to describe synaptic consolidation, pruning, and rewiring, and includes several features that can be added in a modular fashion. The validation has been performed through simulations with firing-rate-based neuronal network models, showing remarkable compatibility between the results of the simulations and theoretical predictions.

Outlook

This thesis makes several contributions to computational neuroscience, with a particular focus on simulation technology and the understanding of synaptic mechanisms in high-level cognitive processes.
Chapters 3–6 are devoted to the validation of a new simulation code (i.e., NEST GPU) able to exploit the most recent GPU technology advancements. Indeed,

GPUs, with their high degree of parallelism, are particularly suitable to handle the simulation of spiking neural networks and the rapid growth of the GPU industry is leading to more performing cards and, most importantly, gradual improvements in the GPU memory. Moreover, supercomputers are entering the exascale era and entrust GPUs with a significant part of their computing power. In this framework, a simulator able to take advantage of this technology is fundamental to open to more detailed simulations of the neuronal dynamics of large-scale networks. Most importantly, this simulator should be easily accessible to the neuroscientific community and the compatibility of the results with respect to affirmed and widely used simulators should be ensured. In this context, the integration of the simulator in the NEST Initiative and the work performed in [1,2,4] go in this direction, by assessing the compatibility of NEST and NEST GPU for different models running both on high-end PCs and MPI-GPU clusters with competitive simulation performance both on network construction and simulation of the network dynamics.
Chapters 7–9 focus on two synaptic mechanisms that are able to underlie high-level cognitive processes: short-term synaptic plasticity and structural synaptic plasticity. The works described in this thesis can be intended as starting points for future works devoted to clarifying the relationship between these mechanisms and brain functioning. The first of the two aforementioned chapters analyzes the relation between short-term plasticity and working memory through spiking network model simulations, whereas the second sheds light on the role of structural plasticity during learning through a theoretical framework based on a mean-field approach supported by simulations of firing-rate based neuronal networks.
In conclusion, this thesis aims to provide the instruments needed to face the challenge of the study of high-level cognitive processes through simulations of large-scale neuronal networks. In the near future, simulation codes such as NEST GPU will be able to simulate network models of substantial portions of the brain that have never been studied on a single neuron scale, and a better understanding of the role of synaptic scale mechanisms will be needed to evaluate their impact on such a large-scale perspective, or the impact of their inhibition. In this regard, neural scale simulation frameworks could be seen as virtual tools able to be employed for testing novel treatments or to evaluate the large-scale effect of a drug able to affect a certain brain mechanism. For this reason, it is fundamental to benefit from the novel technological progress and pave the way for an era in which computational neuroscience can develop network models and theoretical frameworks able to flank experiments and advance our knowledge of brain functioning.

Bibliography

[1] Bruno Golosio, Gianmarco Tiddia, Chiara De Luca, Elena Pastorelli, Francesco Simula, and Pier Stanislao Paolucci. Fast simulations of highly-connected spiking cortical models using GPUs. *Frontiers in Computational Neuroscience*, 15, 2021. `doi:10.3389/fncom.2021.627620`.

[2] Gianmarco Tiddia, Bruno Golosio, Jasper Albers, Johanna Senk, Francesco Simula, Jari Pronold, Viviana Fanti, Elena Pastorelli, Pier Stanislao Paolucci, and Sacha J. van Albada. Fast simulation of a multi-area spiking network model of macaque cortex on an MPI-GPU cluster. *Frontiers in Neuroinformatics*, 16, 2022. `doi:10.3389/fninf.2022.883333`.

[3] Gianmarco Tiddia, Bruno Golosio, Viviana Fanti, and Pier Stanislao Paolucci. Simulations of working memory spiking networks driven by short-term plasticity. *Frontiers in Integrative Neuroscience*, 16, 2022. `doi:10.3389/fnint.2022.972055`.

[4] Bruno Golosio, Jose Villamar, Gianmarco Tiddia, Elena Pastorelli, Jonas Stapmanns, Viviana Fanti, Pier Stanislao Paolucci, Abigail Morrison, and Johanna Senk. Runtime construction of large-scale spiking neuronal network models on GPU devices. *Applied Sciences*, 13(17), 2023. `doi:10.3390/app13179598`.

[5] Gianmarco Tiddia, Luca Sergi, and Bruno Golosio. A theoretical framework for learning through structural plasticity, 2023. `arXiv:2307.11735`.

[6] Frederico A.C. Azevedo, Ludmila R.B. Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata E.L. Ferretti, Renata E.P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, April 2009. `doi:10.1002/cne.21974`.

[7] Bente Pakkenberg, Dorte Pelvig, Lisbeth Marner, Mads J. Bundgaard, Hans Jørgen G. Gundersen, Jens R. Nyengaard, and Lisbeth Regeur. Aging and the human neocortex. *Experimental Gerontology*, 38(1):95–99, 2003. Proceedings of the 6th International Symposium on the Neurobiology and Neuroendocrinology of Aging. `doi:10.1016/S0531-5565(02)00151-1`.

[8] 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, 1952. doi:10.1113/jphysiol.1952.sp004764.

[9] Nicolas Brunel and Mark C. W. van Rossum. Quantitative investigations of electrical nerve excitation treated as polarization. *Biological Cybernetics*, 97(5-6):341–349, November 2007. doi:10.1007/s00422-007-0189-6.

[10] Nicolas Brunel and Mark C. W. van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biological Cybernetics*, 97(5-6):337–339, October 2007. doi:10.1007/s00422-007-0190-0.

[11] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, USA, 2014.

[12] Tobias C. Potjans and Markus Diesmann. The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. *Cerebral Cortex*, 24(3):785–806, 12 2012. arXiv:https://academic.oup.com/cercor/article-pdf/24/3/785/14099777/bhs358.pdf, doi:10.1093/cercor/bhs358.

[13] Stefan Rotter and Markus Diesmann. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological Cybernetics*, 81(5-6):381–402, November 1999. doi:10.1007/s004220050570.

[14] Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 94(5):3637–3642, 2005. PMID: 16014787. doi:10.1152/jn.00686.2005.

[15] Arnd Roth and Mark C. W. van Rossum. Modeling Synapses. In *Computational Modeling Methods for Neuroscientists*. The MIT Press, 09 2009. doi:10.7551/mitpress/9780262013277.003.0007.

[16] Erik De Schutter, editor. *Computational Modeling Methods for Neuroscientists*. The MIT Press, September 2009. doi:kDeSchutter,.

[17] P. Dayan and L.F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Computational Neuroscience Series. MIT Press, 2005.

[18] Misha Tsodyks, Klaus Pawelzik, and Henry Markram. Neural Networks with Dynamic Synapses. *Neural Computation*, 10(4):821–835, 05 1998. doi:10.1162/089976698300017502.

[19] Henry Markram, Yun Wang, and Misha Tsodyks. Differential signaling via the same axon of neocortical pyramidal neurons. *Proceedings of the National*

Academy of Sciences, 95(9):5323–5328, 1998. doi:10.1073/pnas.95.9.5323.

[20] Misha Tsodyks, Asher Uziel, and Henry Markram. Synchrony generation in recurrent networks with frequency-dependent synapses. *Journal of Neuroscience*, 20(1):RC50–RC50, 2000. doi:10.1523/JNEUROSCI.20-01-j0003.2000.

[21] Omri Barak and Misha Tsodyks. Persistent activity in neural networks with dynamic synapses. *PLOS Computational Biology*, 3(2):1–1, 02 2007. doi:10.1371/journal.pcbi.0030035.

[22] Alessandro Barri and Gianluigi Mongillo. *Short-Term Synaptic Plasticity: Microscopic Modelling and (Some) Computational Implications*, pages 105–121. Springer International Publishing, Cham, 2022. doi:10.1007/978-3-030-89439-9_5.

[23] Gianluigi Mongillo, Omri Barak, and Misha Tsodyks. Synaptic theory of working memory. *Science*, 319(5869):1543–1546, 2008. doi:10.1126/science.1150769.

[24] George A. Miller, Eugene Galanter, and Karl H. Pribram. *Plans and the structure of behavior.* Henry Holt and Co, 1960. doi:10.1037/10039-000.

[25] Alan D. Baddeley and Graham Hitch. Working memory. volume 8 of *Psychology of Learning and Motivation*, pages 47–89. Academic Press, 1974. doi:10.1016/S0079-7421(08)60452-1.

[26] Nelson Cowan. *Attention and Memory*. Oxford University Press, February 1998. doi:10.1093/acprof:oso/9780195119107.001.0001.

[27] Bruno Golosio, Angelo Cangelosi, Olesya Gamotina, and Giovanni Luca Masala. A cognitive neural architecture able to learn and communicate through natural language. *PLOS ONE*, 10(11):1–37, 11 2015. doi:10.1371/journal.pone.0140866.

[28] S. Funahashi, C. J. Bruce, and P. S. Goldman-Rakic. Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. *Journal of Neurophysiology*, 61(2):331–349, February 1989. doi:10.1152/jn.1989.61.2.331.

[29] P.S Goldman-Rakic. Cellular basis of working memory. *Neuron*, 14(3):477–485, March 1995. doi:10.1016/0896-6273(95)90304-6.

[30] Mark D'Esposito and Bradley R. Postle. The cognitive neuroscience of working memory. *Annual Review of Psychology*, 66(1):115–142, January 2015. doi:10.1146/annurev-psych-010814-015031.

[31] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, 1949.

[32] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982. `doi:10.1073/pnas.79.8.2554`.

[33] Mark G. Stokes. 'Activity-silent' working memory in prefrontal cortex: a dynamic coding framework. *Trends in Cognitive Sciences*, 19(7):394–405, July 2015. `doi:10.1016/j.tics.2015.05.004`.

[34] Roosa Honkanen, Santeri Rouhinen, Sheng H. Wang, J. Matias Palva, and Satu Palva. Gamma oscillations underlie the maintenance of feature-specific information and the contents of visual working memory. *Cerebral Cortex*, 25(10):3788–3801, November 2014. `doi:10.1093/cercor/bhu263`.

[35] Mikael Lundqvist, Jonas Rose, Pawel Herman, Scott L. Brincat, Timothy J. Buschman, and Earl K. Miller. Gamma and beta bursts underlie working memory. *Neuron*, 90(1):152–164, April 2016. `doi:10.1016/j.neuron.2016.02.028`.

[36] Omri Barak and Misha Tsodyks. Working models of working memory. *Current Opinion in Neurobiology*, 25:20–24, April 2014. `doi:10.1016/j.conb.2013.10.008`.

[37] Edmund T. Rolls, Laura Dempere-Marco, and Gustavo Deco. Holding multiple items in short term memory: A neural mechanism. *PLoS ONE*, 8(4):e61078, April 2013. `doi:10.1371/journal.pone.0061078`.

[38] D. Hansel and G. Mato. Short-term plasticity explains irregular persistent activity in working memory tasks. *Journal of Neuroscience*, 33(1):133–149, January 2013. `doi:10.1523/jneurosci.3455-12.2013`.

[39] Yuanyuan Mi, Mikhail Katkov, and Misha Tsodyks. Synaptic correlates of working memory capacity. *Neuron*, 93(2):323–330, 2017. `doi:10.1016/j.neuron.2016.12.004`.

[40] Florian Fiebig and Anders Lansner. A spiking working memory model based on hebbian short-term potentiation. *The Journal of Neuroscience*, 37(1):83–96, November 2016. `doi:10.1523/jneurosci.1989-16.2016`.

[41] Florian Fiebig, Pawel Herman, and Anders Lansner. An indexing theory for working memory based on fast hebbian plasticity. *eneuro*, 7(2):ENEURO.0374–19.2020, March 2020. `doi:10.1523/eneuro.0374-19.2020`.

[42] Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98(6):459–478, May 2008. `doi:10.1007/s00422-008-0233-1`.

[43] J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362, 2010. revision #184913. `doi:10.4249/scholarpedia.1362`.

[44] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, 18(24):10464–10472, December 1998. doi:10.1523/jneurosci.18-24-10464.1998.

[45] Markus Butz, Florentin Wörgötter, and Arjen van Ooyen. Activity-dependent structural plasticity. *Brain Research Reviews*, 60(2):287–305, 2009. doi:10.1016/j.brainresrev.2008.12.023.

[46] Min Fu and Yi Zuo. Experience-dependent structural plasticity in the cortex. *Trends in Neurosciences*, 34(4):177–187, 2011. doi:10.1016/j.tins.2011.02.001.

[47] Michael Fauth and Christian Tetzlaff. Opposing effects of neuronal activity on structural plasticity. *Frontiers in Neuroanatomy*, 10, 2016. doi:10.3389/fnana.2016.00075.

[48] Haruo Kasai, Masanori Matsuzaki, Jun Noguchi, Nobuaki Yasumatsu, and Hiroyuki Nakahara. Structure–stability–function relationships of dendritic spines. *Trends in Neurosciences*, 26(7):360–368, July 2003. doi:10.1016/s0166-2236(03)00162-0.

[49] Travis C. Hill and Karen Zito. LTP-induced long-term stabilization of individual nascent dendritic spines. *The Journal of Neuroscience*, 33(2):678–686, January 2013. doi:10.1523/jneurosci.1404-12.2013.

[50] N. Toni, P.-A. Buchs, I. Nikonenko, C. R. Bron, and D. Muller. LTP promotes formation of multiple spine synapses between a single axon terminal and a dendrite. *Nature*, 402(6760):421–425, November 1999. doi:10.1038/46574.

[51] J. Simon Wiegert and Thomas G. Oertner. Long-term depression triggers the selective elimination of weakly integrated synapses. *Proceedings of the National Academy of Sciences*, 110(47):E4510–E4519, 2013. doi:10.1073/pnas.1315926110.

[52] Peter R. Huttenlocher. Synaptic density in human frontal cortex — developmental changes and effects of aging. *Brain Research*, 163(2):195–205, 1979. doi:10.1016/0006-8993(79)90349-4.

[53] Jill Sakai. How synaptic pruning shapes neural wiring during development and, possibly, in disease. *Proceedings of the National Academy of Sciences*, 117(28):16096–16099, 2020. doi:10.1073/pnas.2010281117.

[54] D. B. Chklovskii, B. W. Mel, and K. Svoboda. Cortical rewiring and information storage. *Nature*, 431(7010):782–788, October 2004. doi:10.1038/nature03012.

[55] Andreas Knoblauch, Edgar Körner, Ursula Körner, and Friedrich T. Sommer. Structural synaptic plasticity has high memory capacity and can explain

graded amnesia, catastrophic forgetting, and the spacing effect. *PLOS ONE*, 9(5):1–19, 05 2014. doi:10.1371/journal.pone.0096485.

[56] Andreas Knoblauch and Friedrich T. Sommer. Structural plasticity, effectual connectivity, and memory in cortex. *Frontiers in Neuroanatomy*, 10, 2016. doi:10.3389/fnana.2016.00063.

[57] Thomas Bourgeron. A synaptic trek to autism. *Current Opinion in Neurobiology*, 19(2):231–234, 2009. Development. doi:10.1016/j.conb.2009.06.003.

[58] Caitlin E. Moyer, Micah A. Shelton, and Robert A. Sweet. Dendritic spine alterations in schizophrenia. *Neuroscience Letters*, 601:46–53, 2015. Dendritic Spine Dysgenesis in Neuropsychiatric Disease. doi:10.1016/j.neulet.2014.11.042.

[59] Jeffrey J. Hutsler and Hong Zhang. Increased dendritic spine densities on cortical projection neurons in autism spectrum disorders. *Brain Research*, 1309:83–94, 2010. doi:10.1016/j.brainres.2009.09.120.

[60] Marco Pagani, Noemi Barsotti, Alice Bertero, Stavros Trakoshis, Laura Ulysse, Andrea Locarno, Ieva Miseviciute, Alessia De Felice, Carola Canella, Kaustubh Supekar, Alberto Galbusera, Vinod Menon, Raffaella Tonini, Gustavo Deco, Michael V. Lombardo, Massimo Pasqualetti, and Alessandro Gozzi. mTOR-related synaptic pathology causes autism spectrum disorder-associated functional hyperconnectivity. *Nature Communications*, 12(1), October 2021. doi:10.1038/s41467-021-26131-z.

[61] Leisa A. Glantz and David A. Lewis. Decreased Dendritic Spine Density on Prefrontal Cortical Pyramidal Neurons in Schizophrenia. *Archives of General Psychiatry*, 57(1):65–73, 01 2000. doi:10.1001/archpsyc.57.1.65.

[62] Robin Spiess, Richard George, Matthew Cook, and Peter U. Diehl. Structural plasticity denoises responses and improves learning speed. *Frontiers in Computational Neuroscience*, 10, 2016. doi:10.3389/fncom.2016.00093.

[63] Saket Navlakha, Alison L. Barth, and Ziv Bar-Joseph. Decreasing-rate pruning optimizes the construction of efficient and robust distributed networks. *PLOS Computational Biology*, 11(7):1–23, 07 2015. doi:10.1371/journal.pcbi.1004347.

[64] M. Gewaltig and M. Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007. doi:10.4249/scholarpedia.1430.

[65] Nicholas T. Carnevale and Michael L. Hines. *The NEURON Book*. Cambridge University Press, 2006. doi:10.1017/cbo9780511541612.

[66] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314, August 2019. doi:10.7554/eLife.47314.

[67] Julien Vitay, Helge Ü. Dinkelbach, and Fred H. Hamker. ANNarchy: a code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, 9, July 2015. doi:10.3389/fninf.2015.00019.

[68] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018. doi:10.1109/MM.2018.112130359.

[69] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015. doi:10.1109/TCAD.2015.2474396.

[70] Andreas Grübl, Sebastian Billaudelle, Benjamin Cramer, Vitali Karasenko, and Johannes Schemmel. Verification and design methods for the BrainScaleS neuromorphic hardware system. *Journal of Signal Processing Systems*, 92(11):1277–1292, jul 2020. doi:10.1007/s11265-020-01558-7.

[71] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. doi:10.1109/JPROC.2014.2304638.

[72] Oliver Rhodes, Luca Peres, Andrew G. D. Rowley, Andrew Gait, Luis A. Plana, Christian Brenninkmeijer, and Steve B. Furber. Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164):20190160, 2020. doi:10.1098/rsta.2019.0160.

[73] Anno C Kurth, Johanna Senk, Dennis Terhorst, Justin Finnerty, and Markus Diesmann. Sub-realtime simulation of a neuronal network of natural density. *Neuromorphic Computing and Engineering*, 2(2):021001, mar 2022. doi:10.1088/2634-4386/ac55fc.

[74] Arne Heittmann, Georgia Psychou, Guido Trensch, Charles E. Cox, Winfried W. Wilcke, Markus Diesmann, and Tobias G. Noll. Simulating the cortical microcircuit significantly faster than real time on the IBM INC-3000 neural supercomputer. *Frontiers in Neuroscience*, 15, 2022. doi:10.3389/fnins.2021.728460.

[75] Kevin Kauth, Tim Stadtmann, Vida Sobhani, and Tobias Gemmeke. neuroAIx-framework: design of future neuroscience simulation systems exhibiting execution of the cortical microcircuit model 20× faster than biological real-time. *Frontiers in Computational Neuroscience*, 17, April 2023. doi:10.3389/fncom.2023.1144143.

[76] Timo Wunderlich, Akos F. Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, Christian Pehle, Sebastian Billaudelle, Gerd Kiene, Christian Mauch, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici. Demonstrating advantages of neuromorphic computation: A pilot study. *Frontiers in Neuroscience*, 13, 2019. doi:10.3389/fnins.2019.00260.

[77] Gilbert Maurice Güttler. Achieving a higher integration level of neuromorphic hardware using wafer embedding. 2017. doi:10.11588/HEIDOK.00023723.

[78] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48):1–13, 2014. doi:10.3389/fninf.2013.00048.

[79] N. Abi Akar, B. Cumming, V. Karakasis, A. Küsters, W. Klijn, A. Peyser, and S. Yates. Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 274–282, feb 2019. doi:10.1109/EMPDP.2019.8671560.

[80] Lars Niedermeier, Kexin Chen, Jinwei Xing, Anup Das, Jeffrey Kopsick, Eric Scott, Nate Sutton, Killian Weber, Nikil Dutt, and Jeffrey L. Krichmar. CARLsim 6: An open source library for large-scale, biologically detailed spiking neural network simulation. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2022. doi:10.1109/ijcnn55064.2022.9892644.

[81] Denis Alevi, Marcel Stimberg, Henning Sprekeler, Klaus Obermayer, and Moritz Augustin. Brian2CUDA: Flexible and efficient simulation of spiking neural network models on GPUs. *Frontiers in Neuroinformatics*, 16, October 2022. doi:10.3389/fninf.2022.883700.

[82] Pramod Kumbhar, Michael Hines, Jeremy Fouriaux, Aleksandr Ovcharenko, James King, Fabien Delalondre, and Felix Schürmann. CoreNEURON : An optimized compute engine for the NEURON simulator. *Frontiers in Neuroinformatics*, 13, 2019. doi:10.3389/fninf.2019.00063.

[83] Esin Yavuz, James Turner, and Thomas Nowotny. GeNN: a code generation framework for accelerated brain simulations. *Scientific Reports*, 6(1), January 2016. doi:10.1038/srep18854.

[84] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley, Upper Saddle River, NJ, 2010.

[85] Marcel Stimberg, Dan F. M. Goodman, and Thomas Nowotny. Brian2GeNN: accelerating spiking neural network simulations with graphics hardware. *Scientific Reports*, 10(1), January 2020. doi:10.1038/s41598-019-54957-7.

[86] James C. Knight and Thomas Nowotny. GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model. *Frontiers in Neuroscience*, 12, 2018. doi:10.3389/fnins.2018.00941.

[87] Jannis Schuecker, Maximilian Schmidt, Sacha J. van Albada, Markus Diesmann, and Moritz Helias. Fundamental activity constraints lead to specific interpretations of the connectome. *PLOS Computational Biology*, 13(2):e1005179, February 2017. doi:10.1371/journal.pcbi.1005179.

[88] Maximilian Schmidt, Rembrandt Bakker, Claus C. Hilgetag, Markus Diesmann, and Sacha J. van Albada. Multi-scale account of the network structure of macaque visual cortex. *Brain Structure and Function*, 223(3):1409–1435, April 2018. doi:10.1007/s00429-017-1554-4.

[89] Maximilian Schmidt, Rembrandt Bakker, Kelly Shen, Gleb Bezgin, Markus Diesmann, and Sacha Jennifer van Albada. A multi-scale layer-resolved spiking network model of resting-state dynamics in macaque visual cortical areas. *PLOS Computational Biology*, 14(10):1–38, 10 2018. doi:10.1371/journal.pcbi.1006359.

[90] James C. Knight and Thomas Nowotny. Larger GPU-accelerated brain simulations with procedural connectivity. *Nature Computational Science*, 1(2):136–142, February 2021. doi:10.1038/s43588-020-00022-7.

[91] Bruno Golosio, Chiara De Luca, Elena Pastorelli, Francesco Simula, Gianmarco Tiddia, and Pier Stanislao Paolucci. Toward a possible integration of NeuronGPU in NEST. In *NEST Conference*, volume 7, 2020. doi:10.5281/zenodo.4501616.

[92] Elena Pastorelli, Cristiano Capone, Francesco Simula, Maria V. Sanchez-Vives, Paolo Del Giudice, Maurizio Mattia, and Pier Stanislao Paolucci. Scaling of a large-scale simulation of synchronous slow-wave and asynchronous awake-like activity of a cortical model with long-range interconnections. *Frontiers in Systems Neuroscience*, 13:33, 2019. doi:10.3389/fnsys.2019.00033.

[93] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[94] Nicolas Brunel. Persistent activity and the single cell frequency-current curve in a cortical network model. *Network Computation in Neural Systems*, 11, 01 2000. doi:10.1088/0954-898X/11/4/302.

[95] Johanna Senk, Birgit Kriener, Mikael Djurfeldt, Nicole Voges, Han-Jia Jiang, Lisa Schüttler, Gabriele Gramelsberger, Markus Diesmann, Hans E. Plesser, and Sacha J. van Albada. Connectivity concepts in neuronal network modeling. *PLOS Computational Biology*, 18(9):1–49, 09 2022. doi:10.1371/journal.pcbi.1010086.

[96] James C. Knight, Anton Komissarov, and Thomas Nowotny. PyGeNN: A Python library for GPU-enhanced neural networks. *Frontiers in Neuroinformatics*, 15, 2021. doi:10.3389/fninf.2021.659005.

[97] Sacha J. van Albada, Andrew G. Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B. Stokes, David R. Lester, Markus Diesmann, and Steve B. Furber. Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model. *Frontiers in Neuroscience*, 12, 2018. doi:10.3389/fnins.2018.00291.

[98] Oliver Rhodes, Luca Peres, Andrew G. D. Rowley, Andrew Gait, Luis A. Plana, Christian Brenninkmeijer, and Steve B. Furber. Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164):20190160, December 2019. doi:10.1098/rsta.2019.0160.

[99] Stefan Dasbach, Tom Tetzlaff, Markus Diesmann, and Johanna Senk. Dynamical characteristics of recurrent neuronal networks are robust against low synaptic weight resolution. *Frontiers in Neuroscience*, 15, 2021. doi:10.3389/fnins.2021.757790.

[100] Eilen Nordlie, Marc-Oliver Gewaltig, and Hans Ekkehard Plesser. Towards reproducible descriptions of neuronal network models. *PLOS Computational Biology*, 5(8):1–18, 08 2009. doi:10.1371/journal.pcbi.1000456.

[101] Rembrandt Bakker, Wachtler Thomas, and Markus Diesmann. CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics*, 6:30, 2012. doi:10.3389/fninf.2012.00030.

[102] Nikola T Markov, P Misery, Arnaud Falchier, C Lamy, J Vezoli, R Quilodran, MA Gariel, Pascale Giroud, Maria Ercsey-Ravasz, LJ Pilaz, et al. Weight consistency specifies regularities of macaque cortical networks. *Cerebral Cortex*, 21(6):1254–1272, 2011. doi:10.1093/cercor/bhq201.

[103] Nikola T Markov, Maria M Ercsey-Ravasz, AR Ribeiro Gomes, Camille Lamy, Loic Magrou, Julien Vezoli, Pierre Misery, Arnaud Falchier, Rene Quilodran, Marie-Alice Gariel, et al. A weighted and directed interareal connectivity matrix for macaque cerebral cortex. *Cerebral Cortex*, 24(1):17–36, 2014. doi:10.1093/cercor/bhs270.

[104] M. Denker, A. Yegenoglu, and S. Grün. Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework. In *Neuroinformatics 2018*, page P19, 2018. doi:10.12751/incf.ni2018.0019.

[105] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, September 1956. doi:10.1214/aoms/1177728190.

[106] Emanuel Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, September 1962. doi:10.1214/aoms/1177704472.

[107] B. W. Silverman. *Density estimation for statistics and data analysis*. Chapman and Hall, London, 1986. doi:10.1201/9781315140919.

[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL: https://dl.acm.org/doi/10.5555/1953048.2078195.

[109] Wikipedia contributors. Kernel density estimation — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Kernel_density_estimation&oldid=1158176361, 2023. [Online; accessed 25-June-2023].

[110] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. doi:10.1214/aoms/1177729694.

[111] Victor M. Panaretos and Yoav Zemel. Statistical aspects of Wasserstein distances. *Annual Review of Statistics and its Application*, 6(1):405–431, 2019. doi:10.1146/annurev-statistics-030718-104938.

[112] Andrew Frohmader and Hans Volkmer. 1-Wasserstein distance on the standard simplex. *Algebraic Statistics*, 12(1):43–56, Apr 2021. doi:10.2140/astat.2021.12.43.

[113] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On Wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2), 2017. doi:10.3390/e19020047.

[114] S. S. Vallender. Calculation of the Wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974. doi:10.1137/1118101.

[115] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. `doi:10.1038/s41592-019-0686-2`.

[116] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. `doi:10.21105/joss.03021`.

[117] Sebastian Spreizer, Jessica Mitchell, Jakob Jordan, Willem Wybo, Anno Kurth, Stine Brekke Vennemo, Jari Pronold, Guido Trensch, Mohamed Ayssar Benelhedi, Dennis Terhorst, Jochen Martin Eppler, Håkon Mørk, Charl Linssen, Johanna Senk, Melissa Lober, Abigail Morrison, Steffen Graber, Susanne Kunkel, Robin Gutzen, and Hans Ekkehard Plesser. NEST 3.3, March 2022. `doi:10.5281/zenodo.6368024`.

[118] Tanguy Fardet, Stine Brekke Vennemo, Jessica Mitchell, Håkon Mørk, Steffen Graber, Jan Hahne, Sebastian Spreizer, Rajalekshmi Deepu, Guido Trensch, Philipp Weidel, Jakob Jordan, Jochen Martin Eppler, Dennis Terhorst, Abigail Morrison, Charl Linssen, Alberto Antonietti, Kael Dai, Alexey Serenko, Binghuang Cai, Piotr Kubaj, Robin Gutzen, Hanjia Jiang, Itaru Kitayama, Björn Jürgens, and Hans Ekkehard Plesser. NEST 2.20.0, January 2020. `doi:10.5281/zenodo.3605514`.

[119] Philipp Thörnig. JURECA: Data centric and booster modules implementing the modular supercomputing architecture at jülich supercomputing centre. *Journal of large-scale research facilities JLSRF*, 7, October 2021. `doi:10.17815/jlsrf-7-182`.

[120] Alexander Peyser, Johanna Senk, Jari Pronold, Ankur Sinha, Stine Brekke Vennemo, Tammo Ippen, Jakob Jordan, Steffen Graber, Abigail Morrison, Guido Trensch, Tanguy Fardet, Håkon Mørk, Jan Hahne, Jannis Schuecker, Maximilian Schmidt, Susanne Kunkel, David Dahmen, Jochen Martin Eppler, Sandra Diaz, Dennis Terhorst, Rajalekshmi Deepu, Philipp Weidel, Itaru Kitayama, Sepehr Mahmoudian, David Kappel, Martin Schulze, Shailesh Appukuttan, Till Schumann, Hünkar Can Tunç, Jessica Mitchell, Michael Hoff, Eric Müller, Milena Menezes Carvalho, Barna Zajzon, and Hans Ekkehard Plesser. NEST 2.14.1, September 2021. `doi:10.5281/zenodo.4018724`.

[121] Benedikt Von St. Vieth. JUSUF: Modular tier-2 supercomputing and cloud infrastructure at jülich supercomputing centre. *Journal of large-scale research facilities JLSRF*, 7, October 2021. `doi:10.17815/jlsrf-7-179`.

[122] Jan Hahne, Sandra Diaz, Alexander Patronis, Wolfram Schenck, Alexander Peyser, Steffen Graber, Sebastian Spreizer, Stine Brekke Vennemo, Tammo Ippen, Håkon Mørk, Jakob Jordan, Johanna Senk, Sara Konradi, Philipp Weidel, Tanguy Fardet, David Dahmen, Dennis Terhorst, Jonas Stapmanns, Guido Trensch, Alexander van Meegen, Jari Pronold, Jochen Martin Eppler, Charl Linssen, Abigail Morrison, Ankur Sinha, Jessica Mitchell, Susanne Kunkel, Rajalekshmi Deepu, Espen Hagen, Tom Vierjahn, Nilton Liuji Kamiji, Robin de Schepper, Pedro Machado, Jasper Albers, Wouter Klijn, Alex Myczko, William Mayner, Pooja Nagendra Babu, Hanjia Jiang, Sebastian Billaudelle, Benedikt S. Vogler, Guilherme Miotto, Lionel Kusch, Alberto Antonietti, Aitor Morales-Gregorio, Joris Dolderer, Younes Bouhadjar, and Hans Ekkehard Plesser. NEST 3.0, June 2021. doi:10.5281/zenodo.4739103.

[123] Jasper Albers, Jari Pronold, Anno Christopher Kurth, Stine Brekke Vennemo, Kaveh Haghighi Mood, Alexander Patronis, Dennis Terhorst, Jakob Jordan, Susanne Kunkel, Tom Tetzlaff, Markus Diesmann, and Johanna Senk. A modular workflow for performance benchmarking of neuronal network simulations. *Frontiers in Neuroinformatics*, 16, 2022. doi:10.3389/fninf.2022.837549.

[124] Abigail Morrison, Ad Aertsen, and Markus Diesmann. Spike-timing-dependent plasticity in balanced random networks. *Neural Computation*, 19(6):1437—1467, June 2007. doi:10.1162/neco.2007.19.6.1437.

[125] George A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, March 1956. doi:10.1037/h0043158.

[126] Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, 2001. doi:10.1017/S0140525X01003922.

[127] Nelson Cowan. The magical mystery four: How is working memory capacity limited, and why? *Current Directions in Psychological Science*, 19(1):51–57, 2010. doi:10.1177/0963721409359277.

[128] Yun Wang, Henry Markram, Philip H. Goodman, Thomas K. Berger, Junying Ma, and Patricia S. Goldman-Rakic. Heterogeneity in the pyramidal network of the medial prefrontal cortex. *Nature Neuroscience*, 9(4):534–542, Apr 2006. doi:10.1038/nn1670.

[129] Stefano Fusi. A quiescent working memory. *Science*, 319(5869):1495–1496, March 2008. doi:10.1126/science.1155914.

[130] Robert Coultrip, Richard Granger, and Gary Lynch. A cortical model of winner-take-all competition via lateral inhibition. *Neural Networks*, 5(1):47–54, 1992. doi:10.1016/S0893-6080(05)80006-1.

[131] Elodie Fino and Rafael Yuste. Dense inhibitory connectivity in neocortex. *Neuron*, 69(6):1188–1203, 2011. doi:10.1016/j.neuron.2011.02.025.

[132] David Attwell and Simon B. Laughlin. An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, 21(10):1133–1145, 2001. PMID: 11598490. doi:10.1097/00004647-200110000-00001.

[133] Peter Lennie. The cost of cortical computation. *Current Biology*, 13(6):493–497, 2003. doi:10.1016/S0960-9822(03)00135-0.

[134] Rajalekshmi Deepu, Sebastian Spreizer, Guido Trensch, Dennis Terhorst, Stine Brekke Vennemo, Jessica Mitchell, Charl Linssen, Håkon Mørk, Abigail Morrison, Jochen Martin Eppler, Nilton Liuji Kamiji, Robin de Schepper, Itaru Kitayama, Anno Kurth, Aitor Morales-Gregorio, Pooja Nagendra Babu, and Hans Ekkehard Plesser. NEST 3.1, September 2021. doi:10.5281/zenodo.5508805.

[135] A. N. Burkitt. A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input. *Biological Cybernetics*, 95(1):1–19, Jul 2006. doi:10.1007/s00422-006-0068-6.

[136] Alexander Hanuschkin, Susanne Kunkel, Moritz Helias, Abigail Morrison, and Markus Diesmann. A general and efficient method for incorporating precise spike times in globally time-driven simulations. *Frontiers in Neuroinformatics*, 4, 2010. doi:10.3389/fninf.2010.00113.

[137] M. Shafi, Y. Zhou, J. Quintana, C. Chow, J. Fuster, and M. Bodner. Variability in neuronal activity in primate cortex during working memory tasks. *Neuroscience*, 146(3):1082–1108, 2007. doi:10.1016/j.neuroscience.2006.12.072.

[138] Halgurd Taher, Alessandro Torcini, and Simona Olmi. Exact neural mass model for synaptic-based working memory. *PLOS Computational Biology*, 16(12):1–42, 12 2020. doi:10.1371/journal.pcbi.1008533.

[139] Richard Gast, Thomas R. Knösche, and Helmut Schmidt. Mean-field approximations of networks of spiking neurons with short-term synaptic plasticity. *Phys. Rev. E*, 104:044310, Oct 2021. doi:10.1103/PhysRevE.104.044310.

[140] Michael Wolff, Jacqueline Ding, Nicholas Myers, and Mark Stokes. Revealing hidden states in visual working memory using electroencephalography. *Frontiers in Systems Neuroscience*, 9, 2015. doi:10.3389/fnsys.2015.00123.

[141] Michael J. Wolff, Janina Jochim, Elkan G. Akyürek, and Mark G. Stokes. Dynamic hidden states underlying working-memory-guided behavior. *Nature Neuroscience*, 20(6):864–871, Jun 2017. doi:10.1038/nn.4546.

[142] Nathan S. Rose, Joshua J. LaRocque, Adam C. Riggall, Olivia Gosseries, Michael J. Starrett, Emma E. Meyering, and Bradley R. Postle. Reactivation of latent working memories with transcranial magnetic stimulation. *Science*, 354(6316):1136–1139, 2016. doi:10.1126/science.aah7011.

[143] Zachary P. Kilpatrick. Synaptic mechanisms of interference in working memory. *Scientific Reports*, 8(1):7879, May 2018. doi:10.1038/s41598-018-25958-9.

[144] Anastasia Kiyonaga, Jason M Scimeca, Daniel P Bliss, and David Whitney. Serial dependence across perception, attention, and memory. *Trends Cogn. Sci.*, 21(7):493–497, July 2017. doi:10.1016/j.tics.2017.04.011.

[145] Joao Barbosa, Heike Stein, Rebecca L. Martinez, Adrià Galan-Gadea, Sihai Li, Josep Dalmau, Kirsten C. S. Adam, Josep Valls-Solé, Christos Constantinidis, and Albert Compte. Interplay between persistent activity and activity-silent dynamics in the prefrontal cortex underlies serial biases in working memory. *Nature Neuroscience*, 23(8):1016–1024, June 2020. doi:10.1038/s41593-020-0644-4.

[146] Younes Bouhadjar, Dirk J. Wouters, Markus Diesmann, and Tom Tetzlaff. Sequence learning, prediction, and replay in networks of spiking neurons. *PLOS Computational Biology*, 18(6):1–36, 06 2022. doi:10.1371/journal.pcbi.1010233.

[147] Jeff Hawkins, Subutai Ahmad, Donna Dubinsky, et al. Cortical learning algorithm and hierarchical temporal memory. *Numenta Whitepaper*, 1(68):2, 2011.

[148] Dominique Muller, Irina Nikonenko, Pascal Jourdain, and Stefano Alberi. LTP, memory and structural plasticity. *Current Molecular Medicine*, 2(7):605–611, November 2002. doi:10.2174/1566524023362041.

[149] Alex Roxin, Nicolas Brunel, David Hansel, Gianluigi Mongillo, and Carl van Vreeswijk. On the distribution of firing rates in networks of cortical neurons. *Journal of Neuroscience*, 31(45):16217–16226, 2011. doi:10.1523/JNEUROSCI.1677-11.2011.

[150] Bruno Golosio, Chiara De Luca, Cristiano Capone, Elena Pastorelli, Giovanni Stegel, Gianmarco Tiddia, Giulia De Bonis, and Pier Stanislao Paolucci. Thalamo-cortical spiking model of incremental learning combining perception, context and NREM-sleep. *PLOS Computational Biology*, 17(6):1–26, 06 2021. doi:10.1371/journal.pcbi.1009045.

[151] Chiara De Luca, Leonardo Tonielli, Elena Pastorelli, Cristiano Capone, Francesco Simula, Cosimo Lupo, Irene Bernava, Giulia De Bonis, Gianmarco Tiddia, Bruno Golosio, and Pier Stanislao Paolucci. NREM and REM: cognitive and energetic gains in thalamo-cortical sleeping and awake spiking model, 2023. arXiv:2211.06889, doi:10.48550/ARXIV.2211.06889.

[152] Mattia Rigotti, Omri Barak, Melissa R. Warden, Xiao-Jing Wang, Nathaniel D. Daw, Earl K. Miller, and Stefano Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590, May 2013. doi:10.1038/nature12160.

[153] Cristiano Capone, Elena Pastorelli, Bruno Golosio, and Pier Stanislao Paolucci. Sleep-like slow oscillations improve visual classification through synaptic homeostasis and memory association in a thalamo-cortical model. *Scientific Reports*, 9(1):8990, Jun 2019. doi:10.1038/s41598-019-45525-0.

[154] Joachim Hass, Salva Ardid, Jason Sherfey, and Nancy Kopell. Constraints on persistent activity in a biologically detailed network model of the prefrontal cortex with heterogeneities. *Progress in Neurobiology*, 215:102287, 2022. doi:10.1016/j.pneurobio.2022.102287.

[155] Maurizio De Pittà and Nicolas Brunel. Multiple forms of working memory emerge from synapse–astrocyte interactions in a neuron–glia network model. *Proceedings of the National Academy of Sciences*, 119(43):e2207912119, 2022. doi:10.1073/pnas.2207912119.

[156] Sophia Becker, Andreas Nold, and Tatjana Tchumatchenko. Modulation of working memory duration by synaptic and astrocytic mechanisms. *PLOS Computational Biology*, 18(10):1–25, 10 2022. doi:10.1371/journal.pcbi.1010543.

[157] Nicolas Y. Masse, Guangyu R. Yang, H. Francis Song, Xiao-Jing Wang, and David J. Freedman. Circuit mechanisms for the maintenance and manipulation of information in working memory. *Nature Neuroscience*, 22(7):1159–1167, June 2019. doi:10.1038/s41593-019-0414-3.

[158] Jan Kamiński and Ueli Rutishauser. Between persistently active and activity-silent frameworks: novel vistas on the cellular basis of working memory. *Annals of the New York Academy of Sciences*, 1464(1):64–75, August 2019. doi:10.1111/nyas.14213.

[159] Daniel A Butts, Patrick O Kanold, and Carla J Shatz. A burst-based “hebbian” learning rule at retinogeniculate synapses links retinal waves to activity-dependent refinement. *PLoS Biology*, 5(3):e61, March 2007. doi:10.1371/journal.pbio.0050061.

[160] Alexandre Payeur, Jordan Guerguiev, Friedemann Zenke, Blake A. Richards, and Richard Naud. Burst-dependent synaptic plasticity can coordinate learning in hierarchical circuits. *Nature Neuroscience*, 24(7):1010–1019, May 2021. doi:10.1038/s41593-021-00857-x.

[161] Luca Sergi. Teoria di campo medio dell’apprendimento nei sistemi neurali biologici attraverso la plasticità sinaptica strutturale. Master’s thesis, Department of Physics, University of Cagliari, Italy, July 2023.

[162] Vigirdas Mackevičius. *Introduction to Stochastic Analysis*. John Wiley & Sons, Ltd, 2011. doi:10.1002/9781118603338.

[163] A. J. Roberts. Modify the improved euler scheme to integrate stochastic differential equations, 2012. doi:10.48550/ARXIV.1210.0933.

[164] N. G. van Kampen. Itô versus Stratonovich. *Journal of Statistical Physics*, 24(1):175–187, January 1981. doi:10.1007/bf01007642.

[165] Behrad Noudoost, Mindy H Chang, Nicholas A Steinmetz, and Tirin Moore. Top-down control of visual attention. *Current Opinion in Neurobiology*, 20(2):183–190, 2010. Cognitive neuroscience. doi:10.1016/j.conb.2010.02.003.

Appendix A

Additional information for the balanced network model

Here are reported the tables that describe the balanced network model of Section 4.1. The tables follow the guidelines of [100] to provide all the information that can be needed for the reproducibility of the model. The neuron model, i.e., the adaptive-exponential integrate-and-fire with conductance-based synapses can be described by the following equations

$$\begin{aligned} C_\mathrm{m}\frac{dV}{dt} &= -g_\mathrm{L}(V - E_\mathrm{L}) + g_\mathrm{L}\Delta_\mathrm{T}\exp\left(\frac{V - V_\mathrm{th}}{\Delta_\mathrm{T}}\right) + g_\mathrm{ex}(t)(V - E_\mathrm{rev\_ex}) + \\ &+ g_\mathrm{in}(t)(V - E_\mathrm{rev\_in}) - w + I_\mathrm{e} \end{aligned} \tag{A.1}$$

$$\tau_w \frac{dw}{dt} = a(V - E_\mathrm{L}) - w \tag{A.2}$$

where g_ex and g_in are the excitatory and inhibitory conductances, modeled as an alpha function as described in Equation (1.9). The behavior of the *adaptation current* w is described in Equation (A.2) and as soon as the neuron emits a spike the membrane potential resets to a value V_reset and $w \to w + b$.

After the description of the model, the table of network and neuron parameters will be presented.

Model Summary	
Populations	excitatory neurons (E), inhibitory neurons (I), external input (P)
Connectivity	Random connectivity
Neuron model	adaptive exponential integrate-and-fire (AdEx) neuron model
Synapse model	conductance-based synapses, conductance modeled by an alpha function
Plasticity	–
Input	Independent fixed-rate Poisson spike trains to all neurons
Measurements	Spiking activity and average firing rate

Populations		
Name	**Elements**	**Size**
E	AdEx neurons	$N_E = 4N_I$
I	AdEx neurons	N_I
P	Poisson generator	$N_E + N_I$

Connectivity		
Source	**Target**	**Pattern**
P	E ∪ I	• `one_to_one` connectivity (each neuron receives an independent Poisson spike train); • synaptic weights W_{poisson}.
E	E ∪ I	• random, independent; homogeneous indegree $\mathcal{C}_E$; • static synaptic weights W_{ex}; • normally distributed delay with mean μ_d and standard deviation σ_d.
I	E ∪ I	• random, independent; homogeneous indegree $\mathcal{C}_I$; • static synaptic weights W_{in}; • normally distributed delay with mean μ_d and standard deviation σ_d.

Table A.1: Description of the network model.

Network and connectivity		
Name	**Value**	**Description**
N_{E}	Variable	Number of excitatory neurons
N_{I}	$N_{\mathrm{E}}/4$	Number of inhibitory neurons
$\mathcal{C}_{\mathrm{E}}$	4000	Number of excitatory indegrees per neuron
$\mathcal{C}_{\mathrm{I}}$	1000	Number of inhibitory indegrees per neuron
Neuron		
Name	**Value**	**Description**
C	$281\,\mathrm{pF}$	Membrane capacitance
g_{L}	$30\,\mathrm{nS}$	Leak conductance
E_{L}	$-70.6\,\mathrm{mV}$	Leak reversal potential
V_{th}	$-50.4\,\mathrm{mV}$	Spike initiation threshold
Δ_{T}	$2\,\mathrm{mV}$	Slope factor
τ_{w}	$144\,\mathrm{ms}$	Adaptation time constant
a	$4\,\mathrm{nS}$	Subthreshold adaptation
b	$80.5\,\mathrm{pA}$	Spike-triggered adaptation
V_{reset}	$-60\,\mathrm{mV}$	Reset value of the membrane potential after a spike
$E_{\mathrm{rev\_ex}}$	$0\,\mathrm{mV}$	Excitatory reversal potential
$E_{\mathrm{rev\_in}}$	$-85\,\mathrm{mV}$	Inhibitory reversal potential
τ_{syn}	$1\,\mathrm{ms}$	Synaptic time constant
Synapses and external stimuli		
Name	**Value**	**Description**
W_{ex}	$0.05\,\mathrm{pA}$	Excitatory connection weight
W_{in}	$0.35\,\mathrm{pA}$	Inhibitory connection weight
W_{poisson}	$0.37\,\mathrm{pA}$	Poisson spiking devices connection weight
μ_{d}	$0.5\,\mathrm{ms}$	Mean synaptic delay
σ_{d}	$0.25\,\mathrm{ms}$	Standard deviation of synaptic delay
ν_{ext}	$20000\,\mathrm{Hz}$	Fixed Poisson spike rate

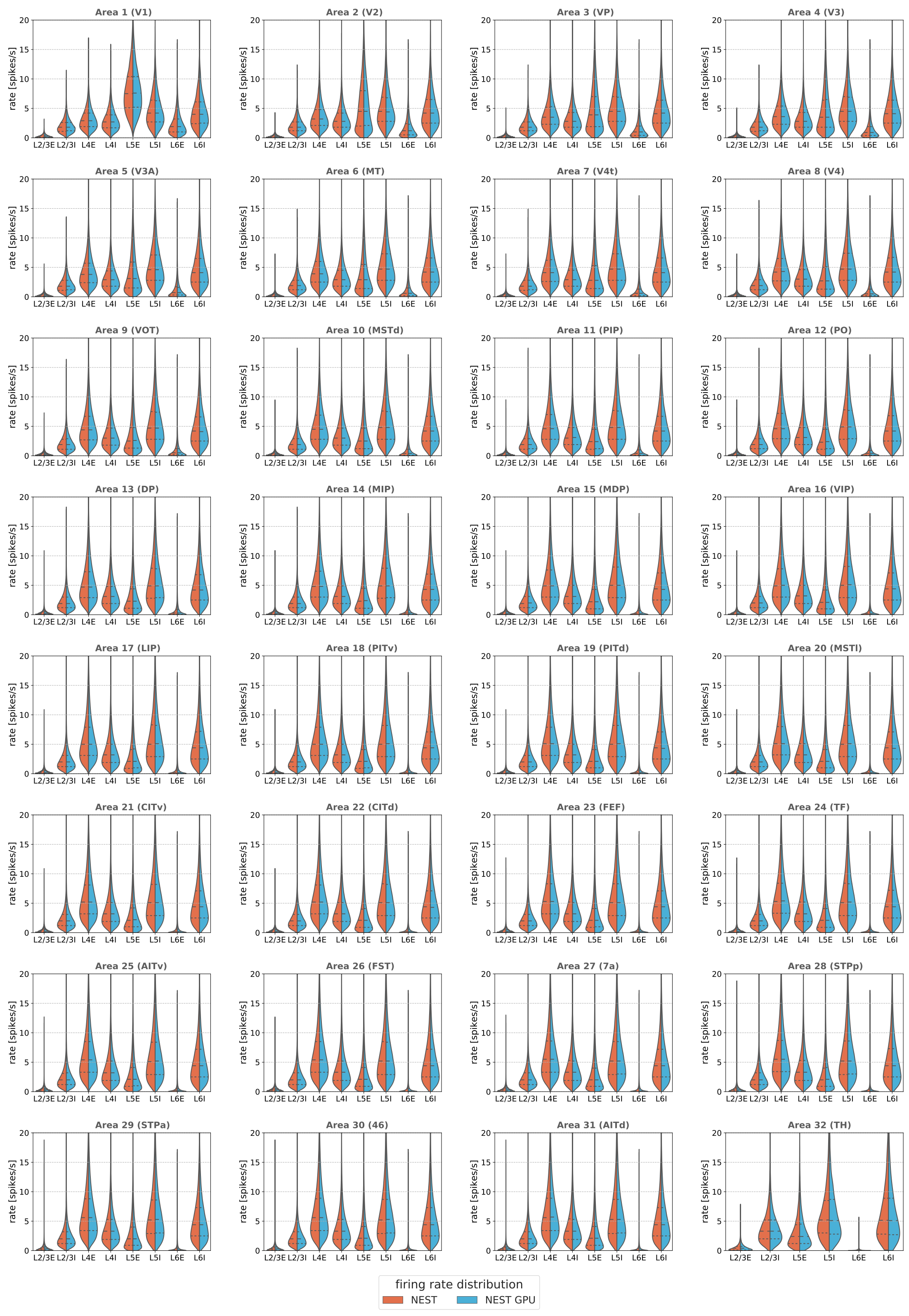
Table A.2: Model parameters.

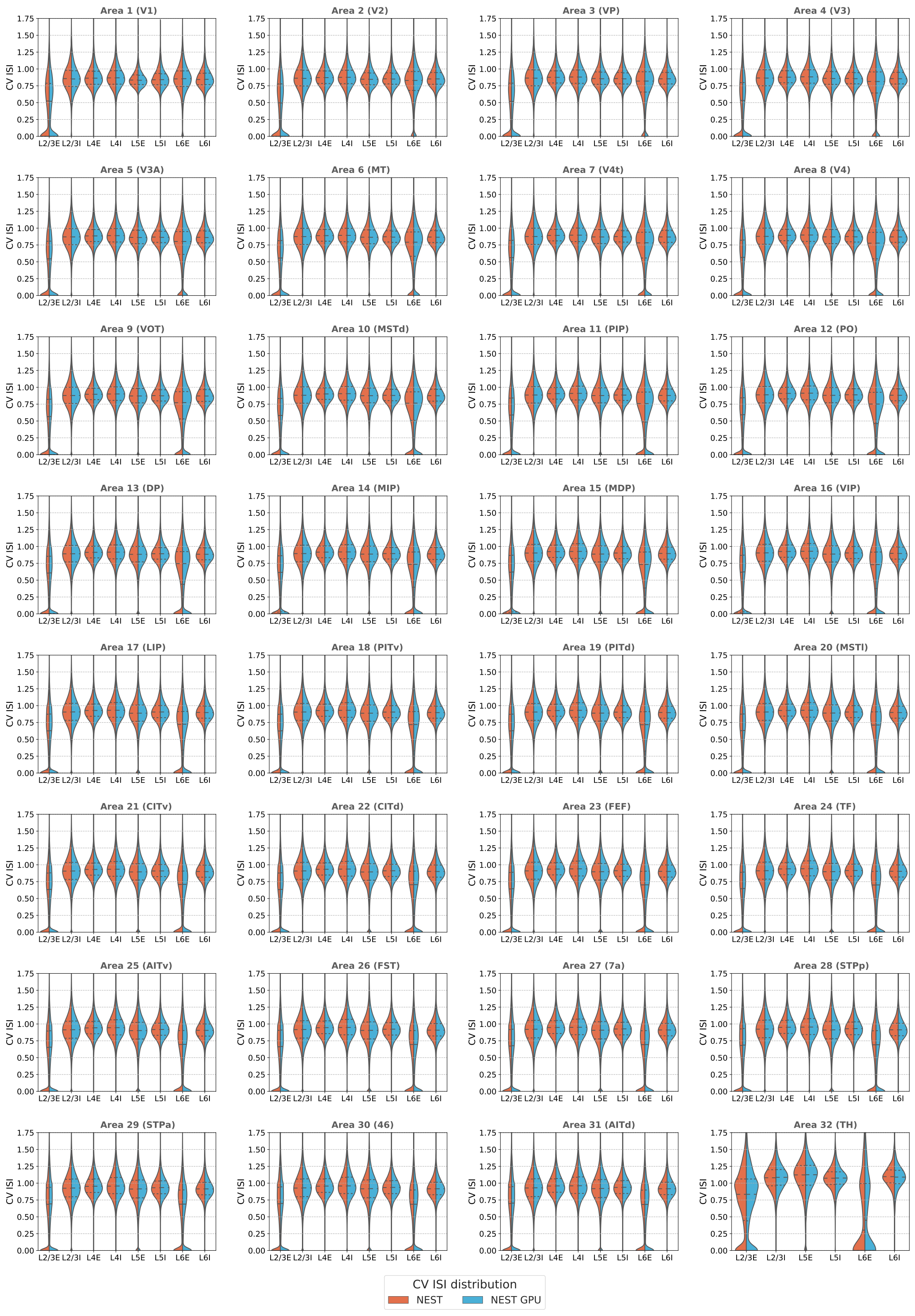
Appendix B

Validation of the multi-area model

In this appendix are depicted the violin plots of distributions and the EMD box plots for all the populations and the areas of the multi-area model. The model is provided with 32 areas with 8 populations each (except for the last area, which lacks layer 4). The following plots indicate the area and the relative label as a subplot title, with each subplot obtained using the same framework needed for Figure 5.5 and Figure 5.8.

Figure B.1 represents the violin plots of firing rate, CV ISI, and Pearson correlation obtained from a simulation of the ground state of the multi-area model using NEST and NEST GPU for each area of the model. Figure B.2 represents, instead, the violin plots of the same distributions obtained simulating the metastable state of the model with NEST and NEST GPU. Similarly, Figures B.3 and B.4 represent the EMD metric when comparing two sets of NEST simulations or a set of NEST and NEST GPU simulations.


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
rate [spikes/s]
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
firing rate distribution
NEST
NEST GPU


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
CV ISI
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
L2/3E
L2/3I
L4E
L4I
L5E
L5I
L6E
L6I
CV ISI distribution
NEST
NEST GPU

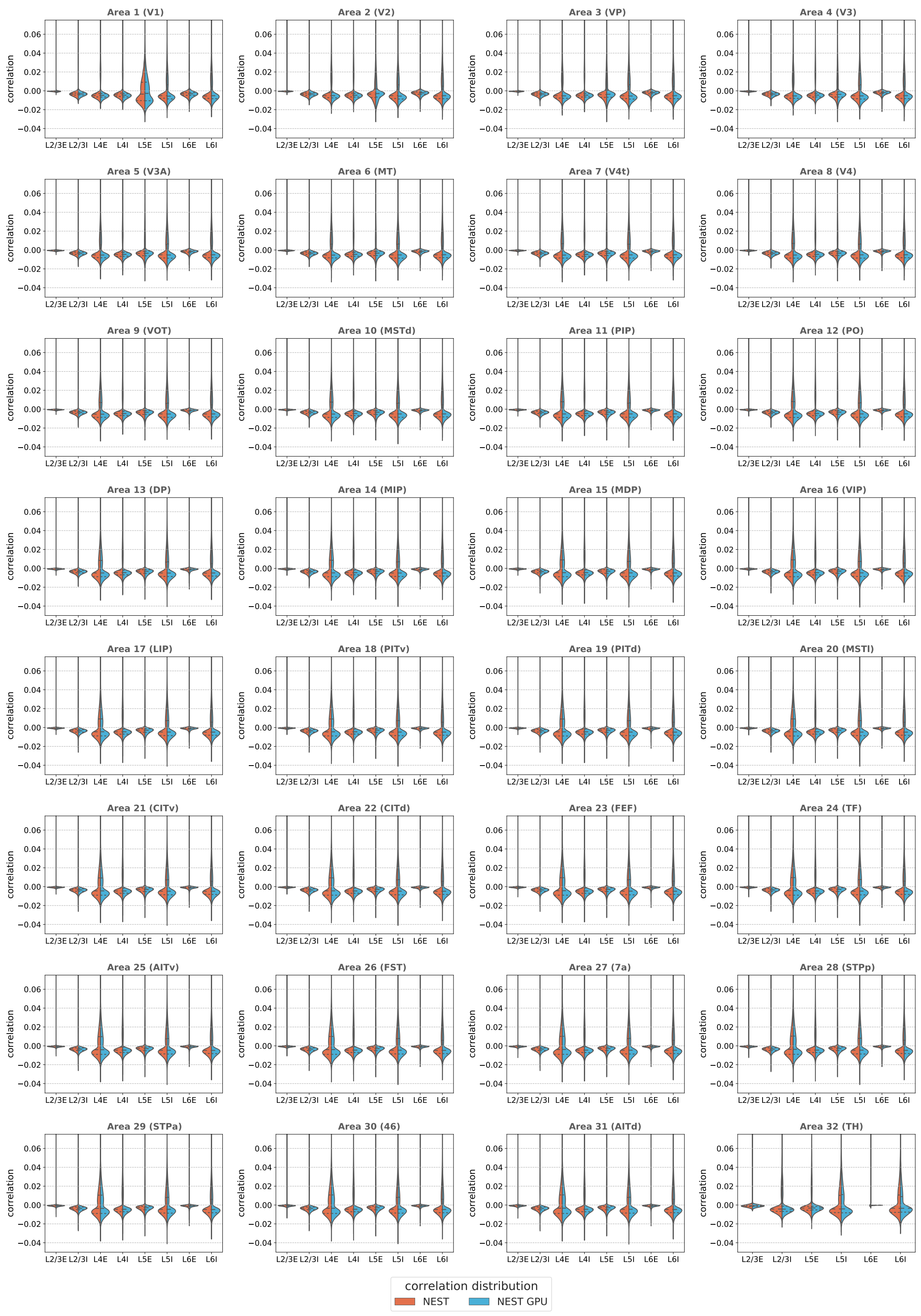
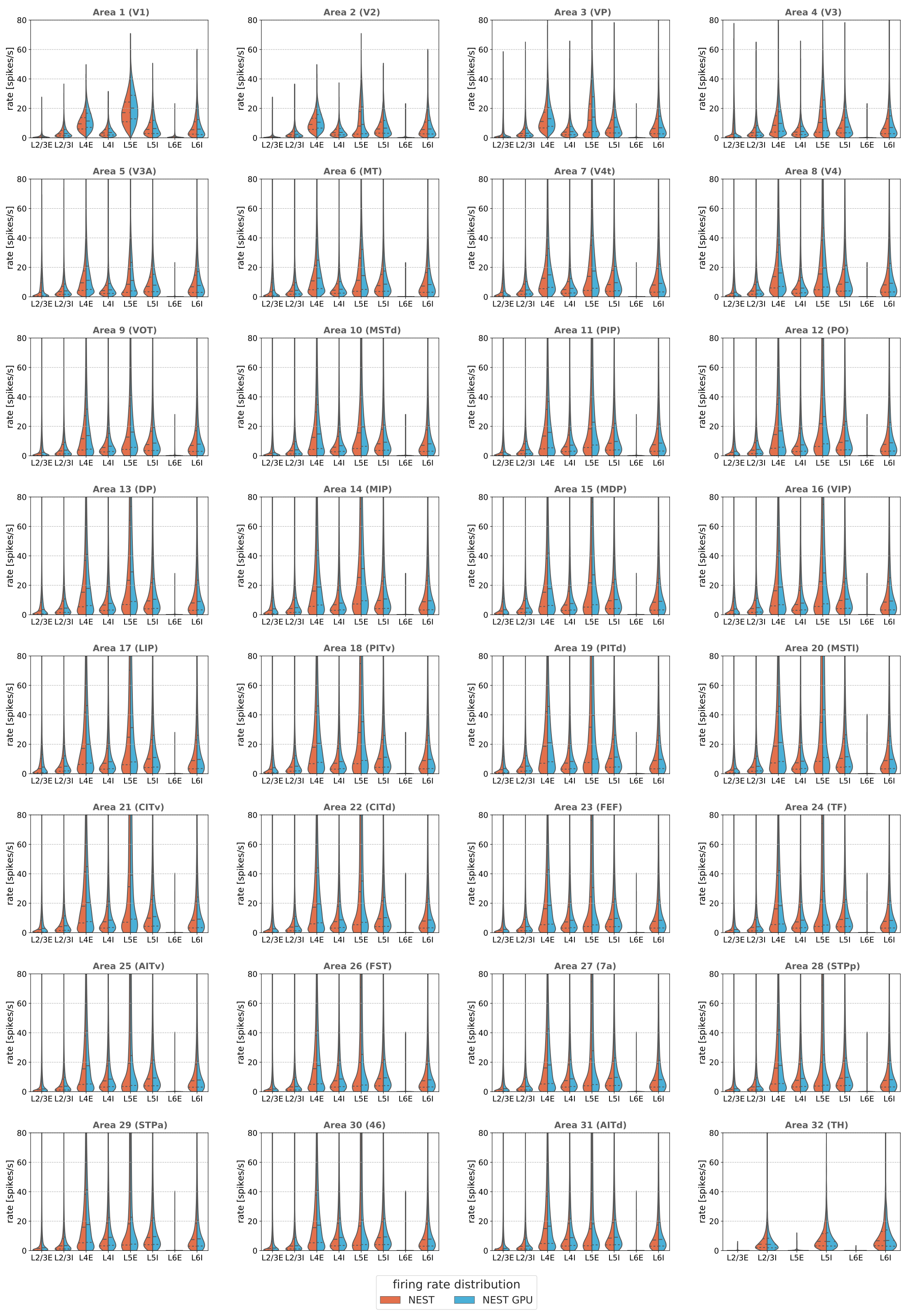
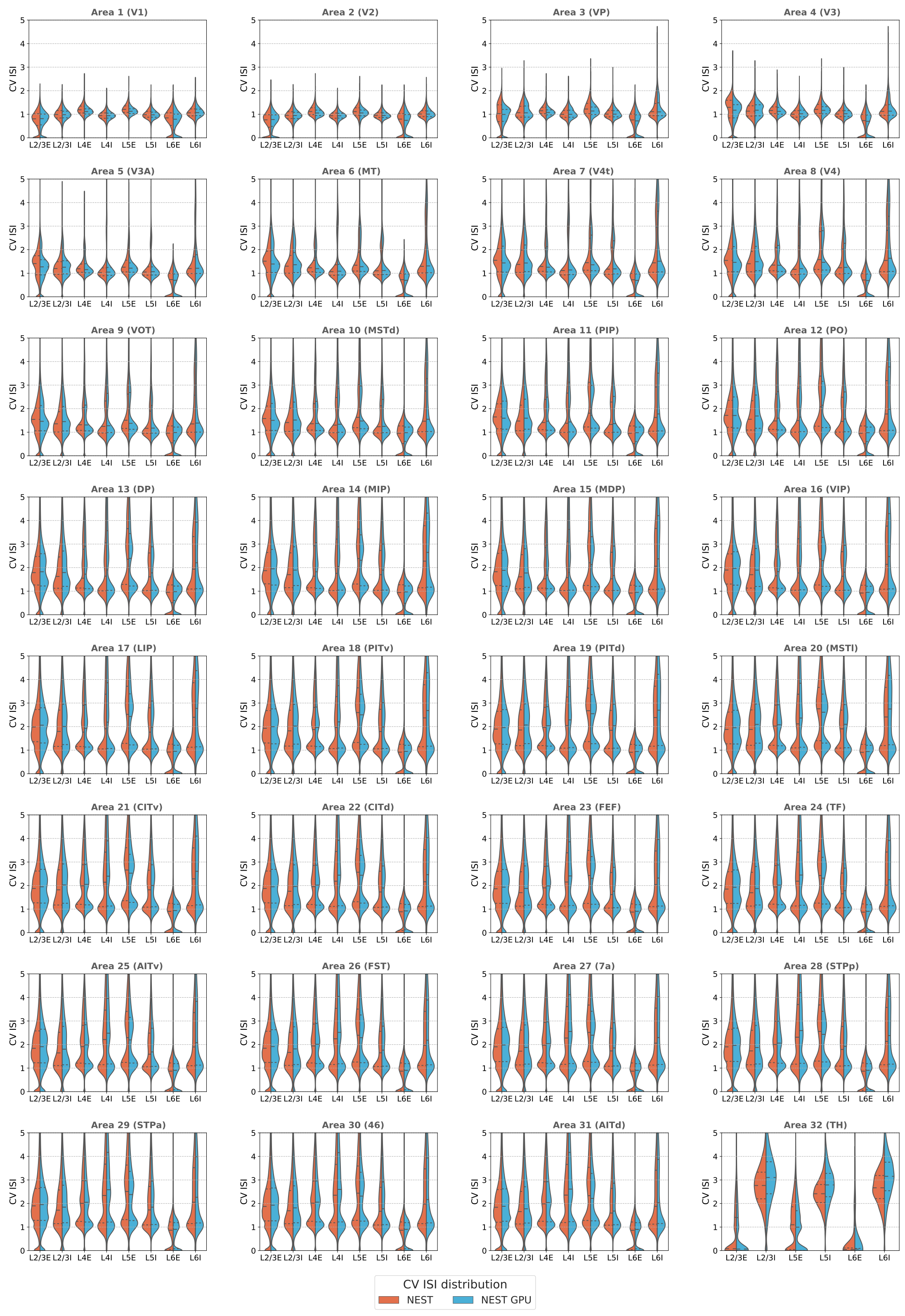


Figure B.1: Violin plot of the distributions of firing rate, CV ISI, and Pearson correlation extracted during a simulation of the ground state of the multi-area model performed on NEST (orange distributions) and NEST GPU (sky blue distributions). The Central dashed line represents the median of the distributions, and the other two dashed lines represent the interquartile range.


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
rate [spikes/s]
0
20
40
60
80
L2/3E
L2/3I
L4E
L4I
L5E
L5I
L6E
L6I
firing rate distribution
NEST
NEST GPU


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
CV ISI
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
CV ISI distribution
NEST
NEST GPU

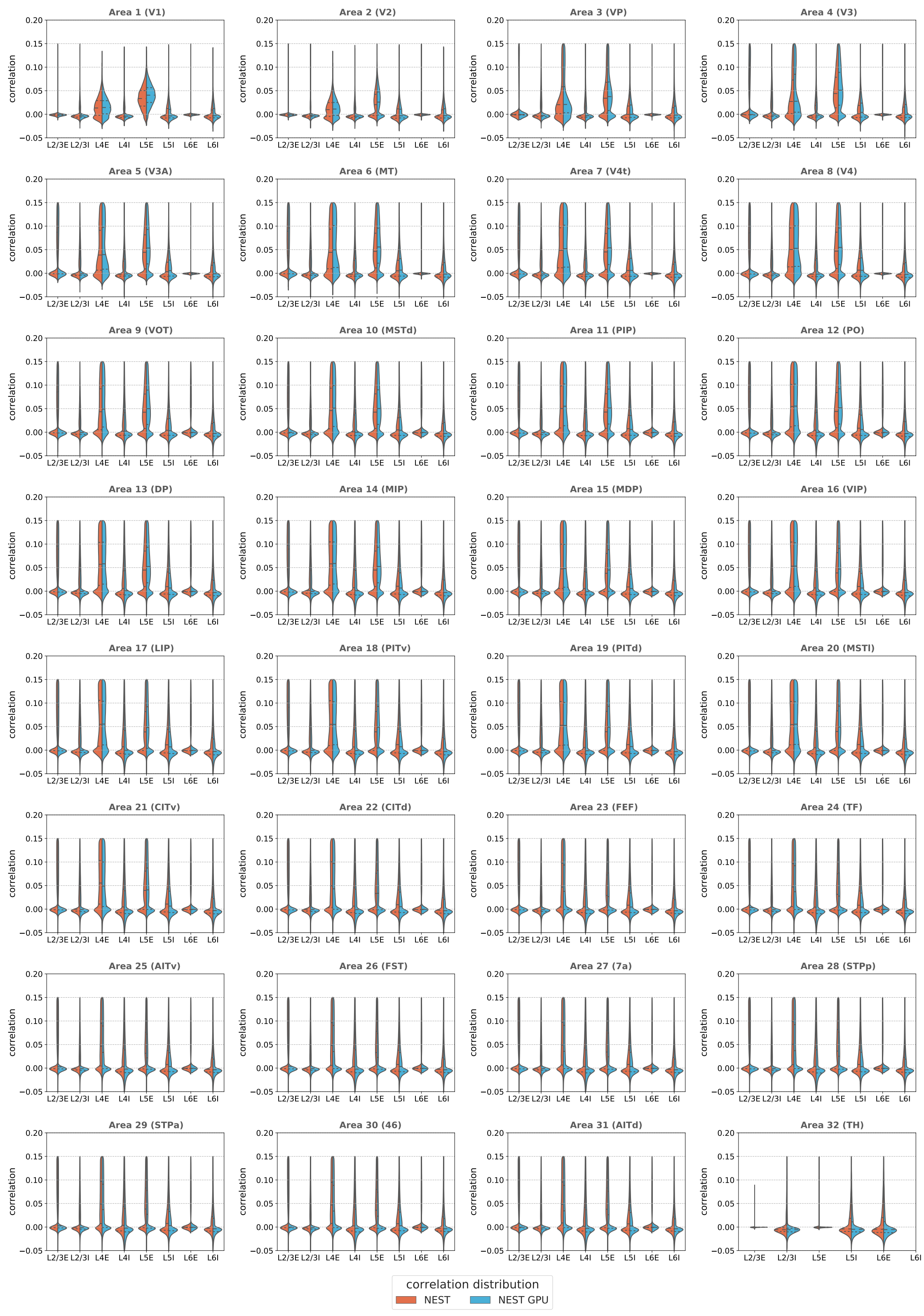
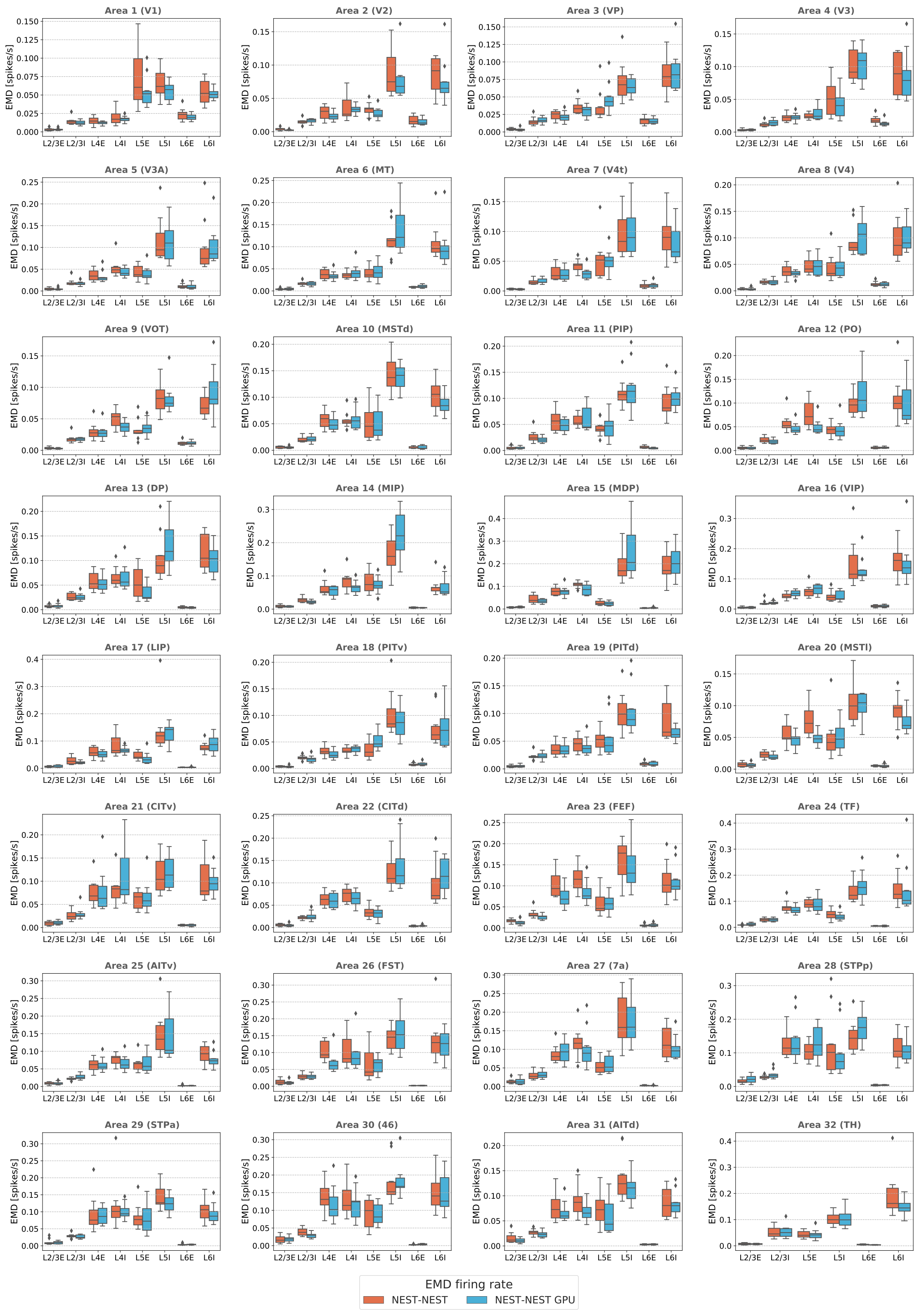
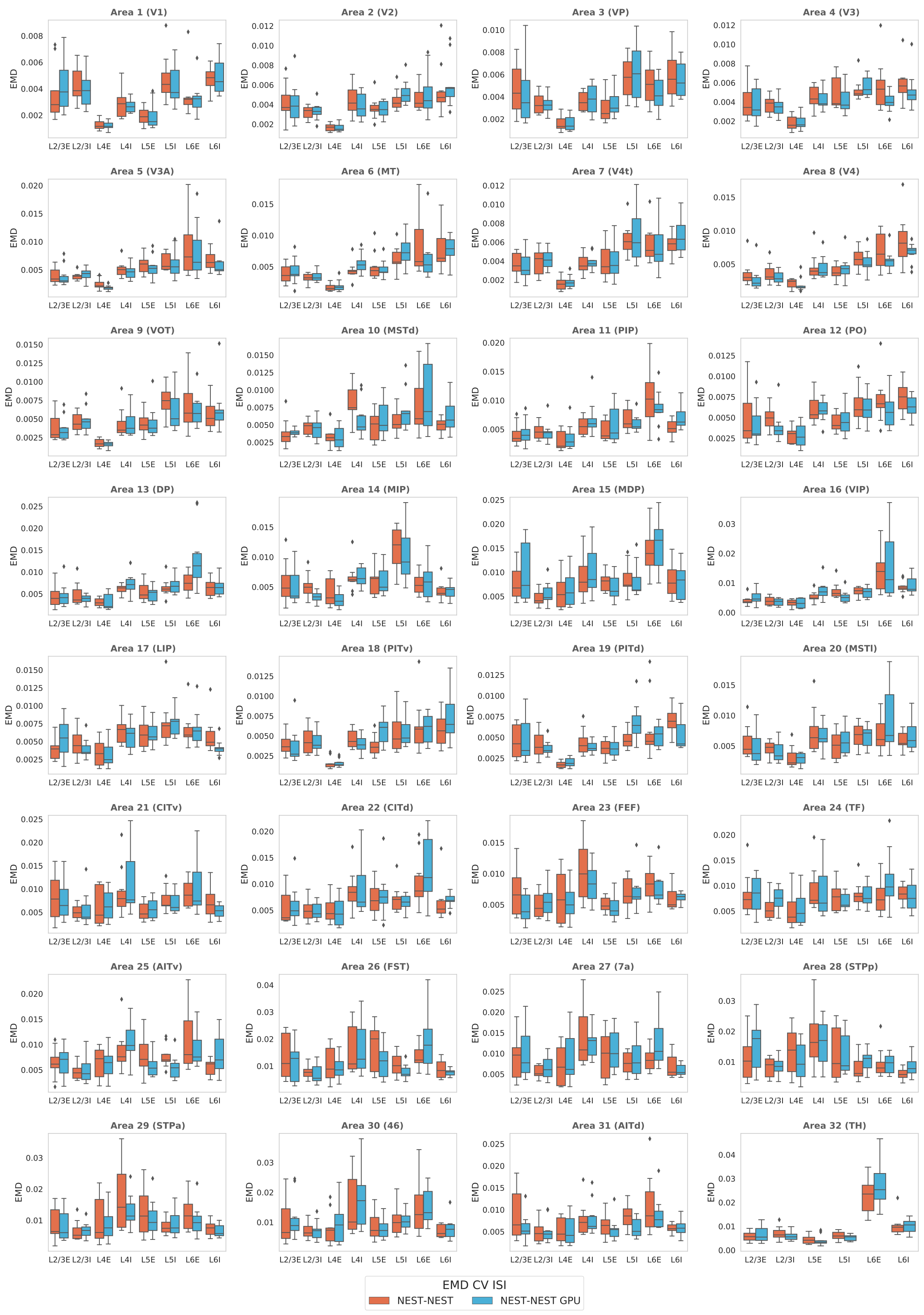


Figure B.2: Violin plot of the distributions of firing rate extracted during a simulation of the metastable state of the multi-area model performed on NEST (orange distributions) and NEST GPU (sky blue distributions). The Central dashed line represents the median of the distributions, and the other two dashed lines represent the interquartile range.


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
EMD [spikes/s]
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
EMD firing rate
NEST-NEST
NEST-NEST GPU


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
EMD
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
EMD CV ISI
NEST-NEST
NEST-NEST GPU

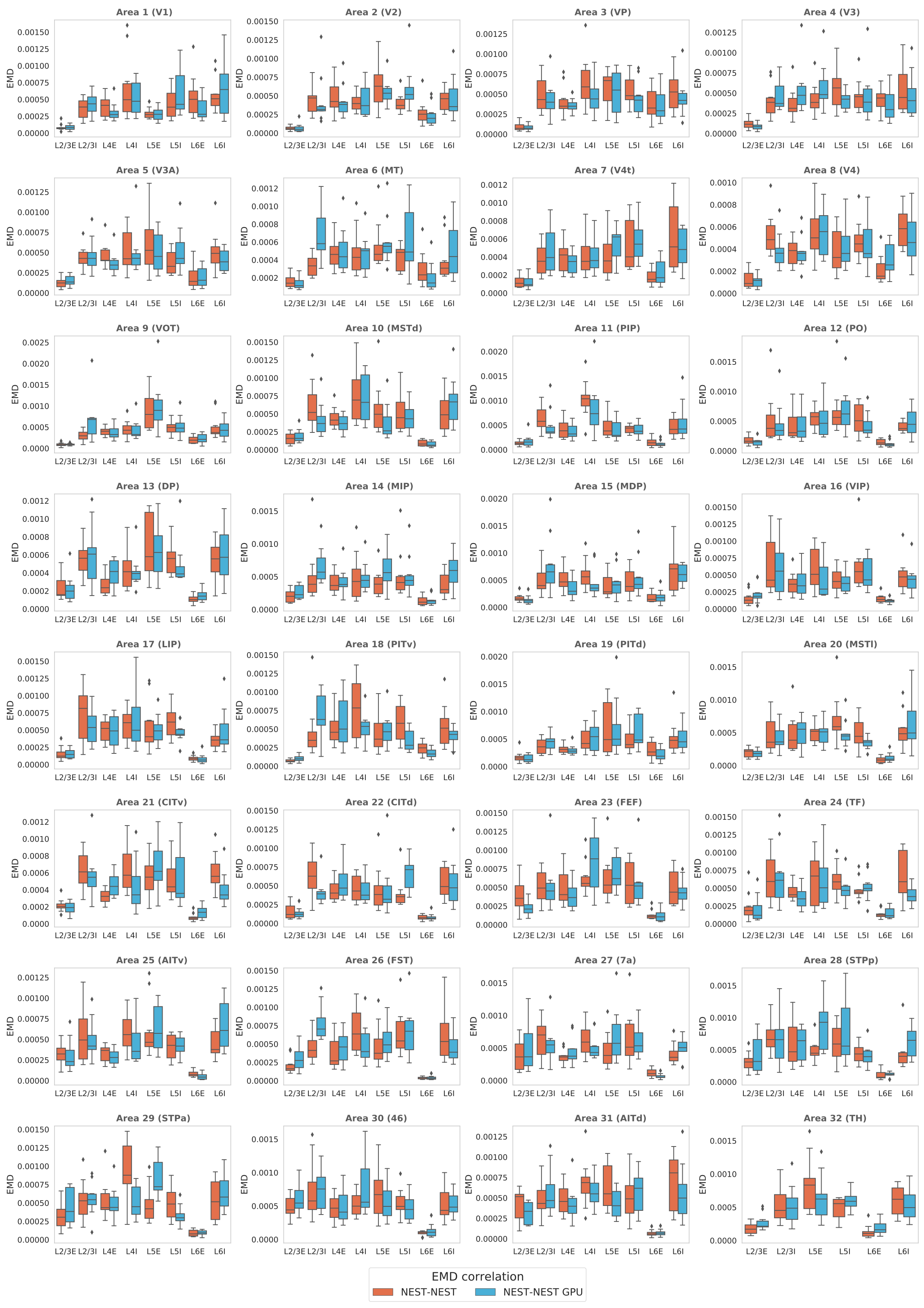
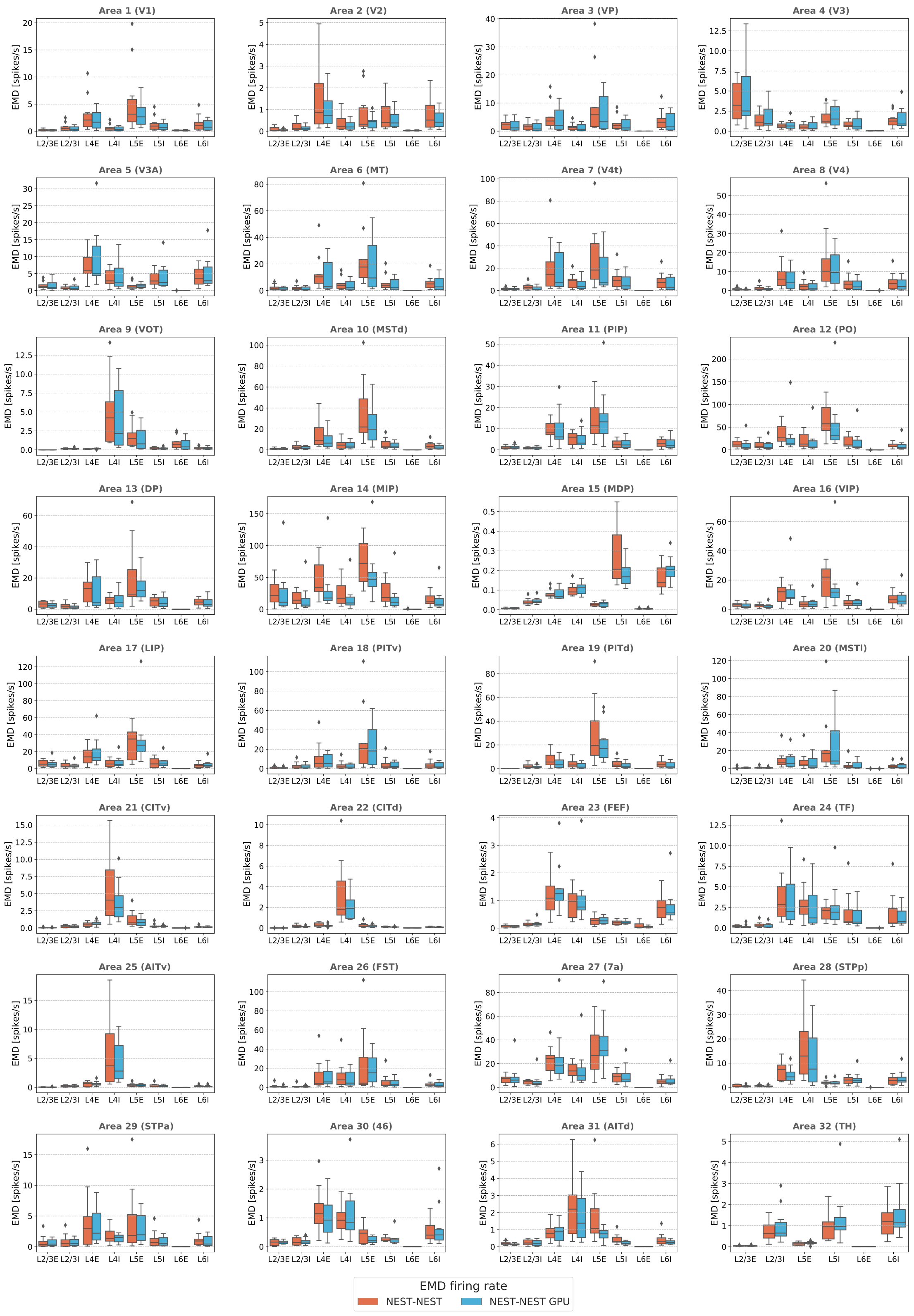
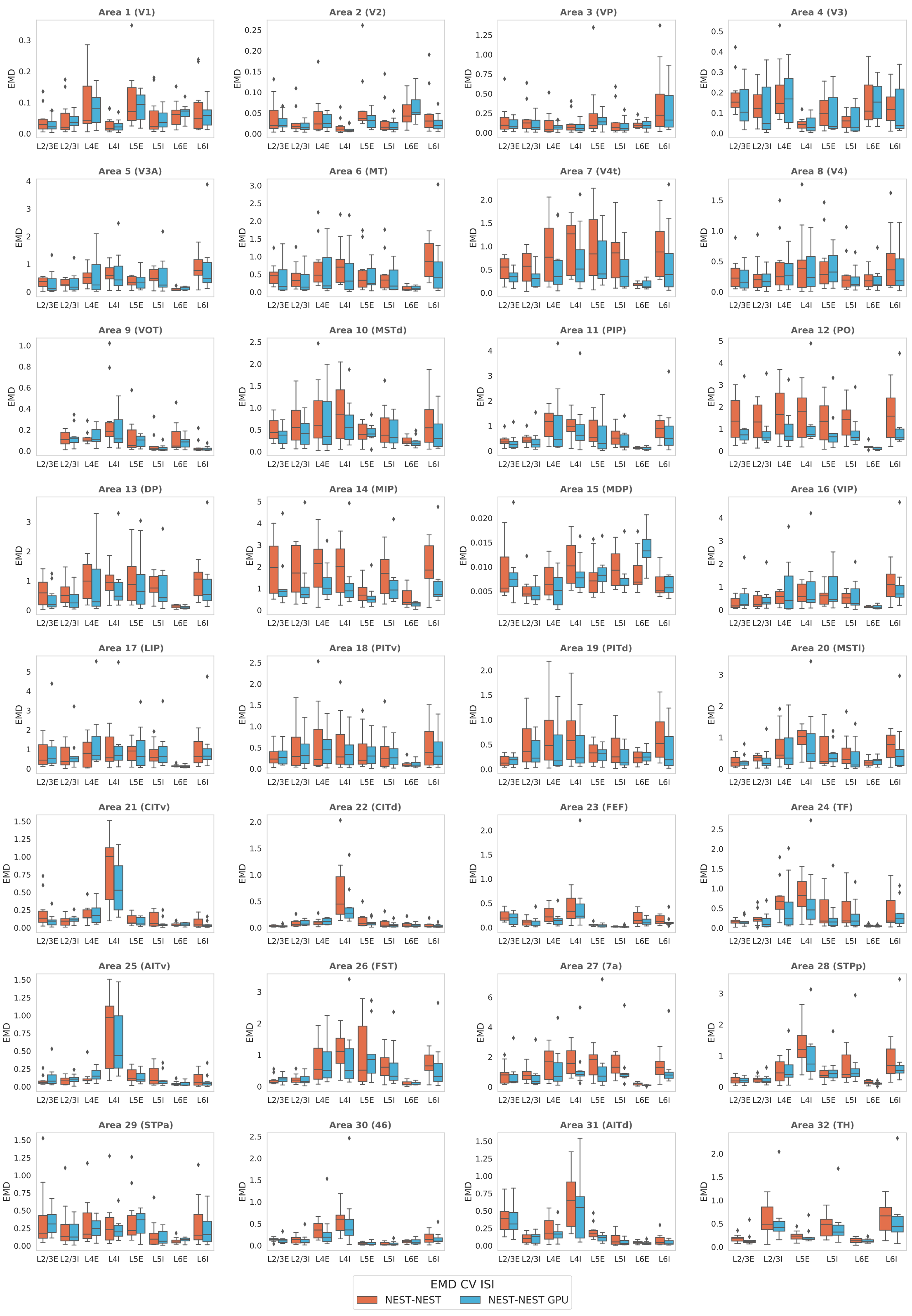


Figure B.3: Earth Mover's Distance between distributions of firing rate, CV ISI, and correlation of the spike trains obtained from all the areas of the model in the ground state simulated with NEST and NEST GPU. EMD boxes obtained comparing NEST using different seeds (NEST-NEST, orange) and NEST and NEST GPU (NEST-NEST GPU, sky blue) are placed side by side.


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
EMD [spikes/s]
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
EMD firing rate
NEST-NEST
NEST-NEST GPU


Area 1 (V1)
Area 2 (V2)
Area 3 (VP)
Area 4 (V3)
Area 5 (V3A)
Area 6 (MT)
Area 7 (V4t)
Area 8 (V4)
Area 9 (VOT)
Area 10 (MSTd)
Area 11 (PIP)
Area 12 (PO)
Area 13 (DP)
Area 14 (MIP)
Area 15 (MDP)
Area 16 (VIP)
Area 17 (LIP)
Area 18 (PITv)
Area 19 (PITd)
Area 20 (MSTl)
Area 21 (CITv)
Area 22 (CITd)
Area 23 (FEF)
Area 24 (TF)
Area 25 (AITv)
Area 26 (FST)
Area 27 (7a)
Area 28 (STPp)
Area 29 (STPa)
Area 30 (46)
Area 31 (AITd)
Area 32 (TH)
EMD
L2/3E L2/3I L4E L4I L5E L5I L6E L6I
EMD CV ISI
NEST-NEST
NEST-NEST GPU

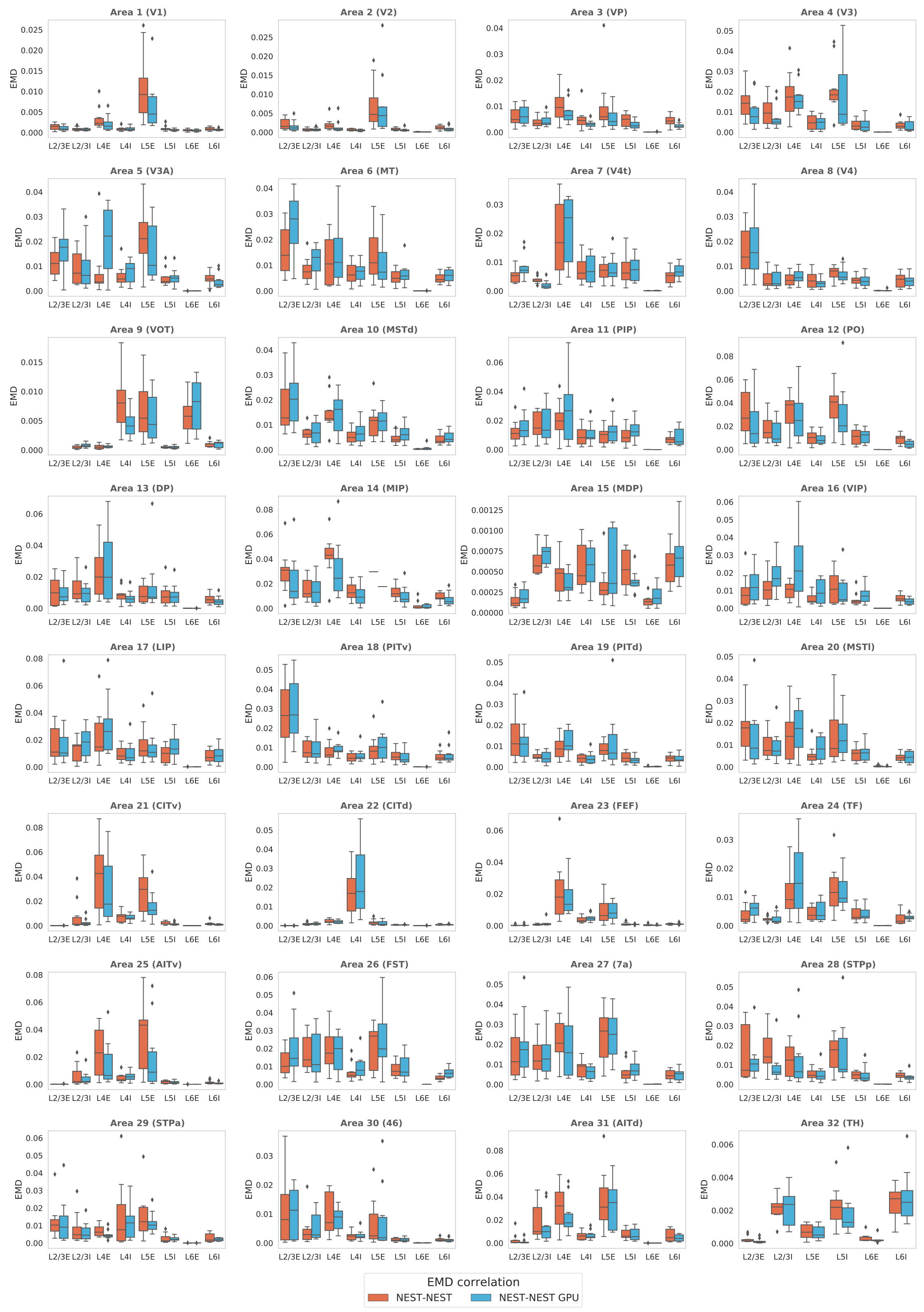


Figure B.4: Earth Mover's Distance between distributions of firing rate, CV ISI, and correlation of the spike trains obtained from all the areas of the model in the metastable state simulated with NEST and NEST GPU. EMD boxes obtained comparing NEST using different seeds (NEST-NEST, orange) and NEST and NEST GPU (NEST-NEST GPU, sky blue) are placed side by side.

Appendix C

Implementation of STP model

The STP model of [18, 20] describes the behavior of the amount of available resources in the presynaptic terminal by using a system of three differential equations, together with the differential equation describing the behavior of the utilization factor u:

$$\begin{aligned}
\frac{dx}{dt} &= \frac{z}{\tau_d} - u(t_s)x(t_s-\epsilon)\delta(t-t_s) \\
\frac{dy}{dt} &= -\frac{y}{\tau_{syn}} + u(t_s)x(t_s-\epsilon)\delta(t-t_s) \\
\frac{dz}{dt} &= \frac{y}{\tau_{syn}} - \frac{x}{\tau_d} \\
\frac{du}{dt} &= -\frac{u}{\tau_f} + U(1-u)\delta(t-t_s)
\end{aligned} \tag{C.1}$$

where x, y, and z are respectively the (normalized) amount of resources in the recovered, active state and inactive state. Here, the synaptic modulation is driven by the variable $y(t)$. In this equation and in the following, for simplicity, we neglect the indexed i, j indicating the presynaptic and the postsynaptic neuron. The model is already implemented in the NEST simulator under the name of `tsodyks_synapse` and it exactly solves Equation (C.1). As mentioned in Section 2.1, this model can be simplified by adopting a system of two differential equations which describe the behavior of the synaptic resources (x) and the one of the utilization factor (u) [18] so that

$$\begin{aligned}
\frac{du}{dt} &= -\frac{u-U}{\tau_f} + U(1-u)\delta(t-t_s) \\
\frac{dx}{dt} &= \frac{1-x}{\tau_d} - ux\delta(t-t_s)
\end{aligned} \tag{C.2}$$

The model described in Equation (C.2) is the same adopted in [21] and [23] (see Equations 5 and 6 of the Supporting Material of [23]). Further, such a model is included in NEST as well under the name of `tsodyks2_synapse`. In the NEST implementation, Equation (C.2) is not integrated at every time step, but the values of the variables $x_{i,j}$ and $u_{i,j}$ are analytically obtained whenever a spike is emitted by the presynaptic neuron i. Thus, it performs the temporal evolution from t_{s-1} and t_s.

Indeed, the model used in [3] to describe STP dynamics is a modification of the latter NEST synapse model, which we named `tsodyks3_synapse`. The modification involves only the order of the variables updates, in agreement with Equation (3.1) of [18], which leads to a difference in synaptic modulation.

In particular, having two consecutive spikes emitted at times t_s and t_{s+1} and knowing $x(t_s)$ and $u(t_s)$, the variables evolution is computed as follows:

$$\begin{aligned}
x(t_{s+1}^-) &= 1 + \big(x(t_s^+) - 1\big)e^{-(t_{s+1}-t_s)/\tau_d} \\
u(t_{s+1}^-) &= U + \big(u(t_s^+) - U\big)e^{-(t_{s+1}-t_s)/\tau_f} \\
u(t_{s+1}^+) &= u(t_{s+1}^-) + U\big(1 - u(t_{s+1}^-)\big) \\
x(t_{s+1}^+) &= x(t_{s+1}^-) - u(t_{s+1}^+)x(t_{s+1}^-)
\end{aligned} \tag{C.3}$$

where t_s represents the spike time, while t_s^- and t_s^+ represent the times immediately before and immediately after the spike emission, respectively. More formally, $x(t_s^-)$ and $u(t_s^-)$ can be intended as the left-side limits:

$$\begin{aligned}
x(t_s^-) &= \lim_{\epsilon\to 0} x(t_s - \epsilon) \quad \text{with}\, \epsilon \in \mathbb{R}^+ \\
u(t_s^-) &= \lim_{\epsilon\to 0} u(t_s - \epsilon) \quad \text{with}\, \epsilon \in \mathbb{R}^+
\end{aligned} \tag{C.4}$$

while $x(t_s^+)$ and $u(t_s^+)$ can be intended as the right-side limits:

$$\begin{aligned}
x(t_s^+) &= \lim_{\epsilon\to 0} x(t_s + \epsilon) \quad \text{with}\, \epsilon \in \mathbb{R}^+ \\
u(t_s^+) &= \lim_{\epsilon\to 0} u(t_s + \epsilon) \quad \text{with}\, \epsilon \in \mathbb{R}^+
\end{aligned} \tag{C.5}$$

Because of the discontinuity due to the spike emission, in general, the left-side and right-side limits differ from each other for the variables x and u. On the other hand, the exponential functions appearing in the first two lines of Equation (C.3) are continuous everywhere, therefore the left and right limits are equal to each other for these functions. Therefore, the modulation led by short-term plasticity shown in Equation (2.3) is given by $u(t_{s+1}^+)x(t_{s+1}^-)$, so considering variable x immediately before the spike emission and the variable u updated at the time of the spike emission as described in [18]. Only after spike emission, the variable x is decreased because of neurotransmitter release. This order of update stems from the fact that the presynaptic spike triggers facilitation (i.e. the increasing of the variable u) just before the spike emission to the postsynaptic neuron. Equation (C.3) is implemented in the NEST simulator with the `tsodyks3_synapse` model, a modified version of the NEST synapse model `tsodyks2_synapse` model, which describes the STP dynamics according to Equation (2.2) as well, but modulating the synaptic efficacy using the term $u(t_{s+1}^-)x(t_{s+1}^-)$. Indeed, such a difference in the implementation can be relevant, especially with neurons having low firing rates [139], with `tsodyks3_synapse` model showing higher modulated synaptic efficacies than `tsodyks2_synapse` model.

To verify the reliability of `tsodyks3_synapse` model, we compare it with the NEST models `tsodyks_synapse` and `tsodyks2_synapse`. To do so, we simulate four LIF neurons with exponential postsynaptic currents (same model as the one used in the network): the first (presynaptic) neuron is injected with a constant current able to induce the emission of spikes during the first $500\,\mathrm{ms}$ and the last $500\,\mathrm{ms}$ of simulation. Between the two current injections a time interval of $1000\,\mathrm{ms}$ is simulated without injecting any current to observe the weight change after a short period of inactivity. The neuron is then connected to the other neurons using one of the three synaptic models with the same parameters. Then, the membrane potential of the three postsynaptic neurons is recorded to analyze the differences in the peak amplitude (i.e. of the modulated synaptic weight). The postsynaptic potentials are shown in Figure C.1.

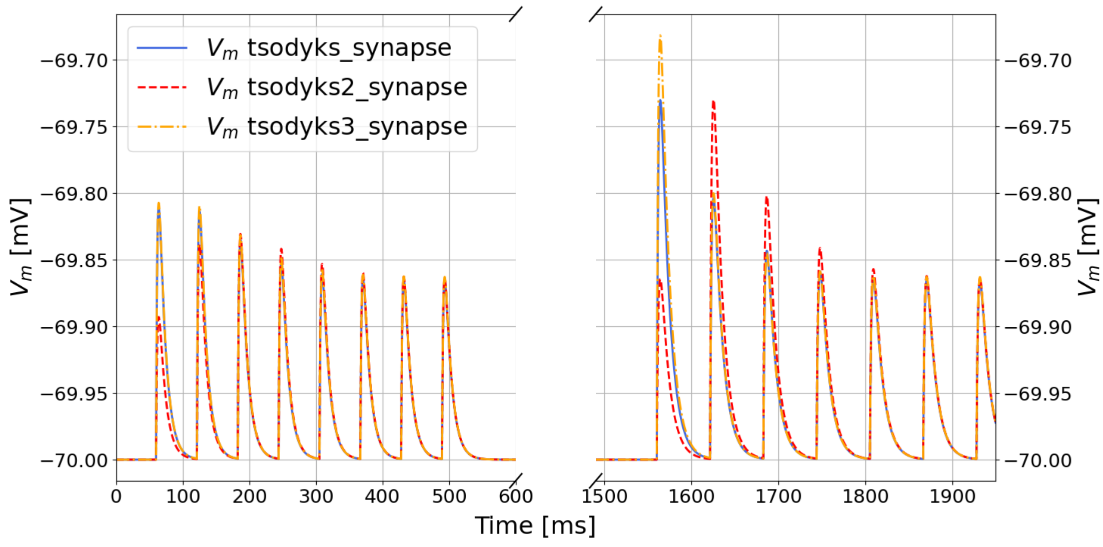


Figure C.1: Excitatory postsynaptic potential of neurons connected to the same presynaptic neurons using different STP synapses. The blue line is obtained using the NEST synapse model `tsodyks_synapse`, the red dashed line is obtained using the `tsodyks2_synapse` and the orange line is obtained with `tsodyks3_synapse`. The time axis has been adjusted to show only the time intervals in which the presynaptic neuron is stimulated.

It is possible to notice that the difference in peak amplitude between the neurons is significant only for the first spikes, after which the models modulate the synaptic weights producing comparable postsynaptic potentials. The differences that arise between `tsodyks2_synapse` and its modification used in this work are only due to the difference in the order of update of the variable u and the synaptic weight. Moreover, the differences between such a model and the `tsodyks_synapse` are less significant and can be justified by the different behavior of u dynamics. In fact, the only relevant difference between these models can be observed after a relatively long-lasting time (in the order of τ_f) during which the STP variable u can evolve and reach values near to the resting ones, which differs within the two models. In this regard, the first postsynaptic potential after the second of biological time during which the presynaptic neuron does not emit spikes shows the mentioned

difference.
In summary, we decided to make this modification in order to be consistent with the order of variables update shown in Equation (3.1) of [18]. The modifications made of the `tsodyks2_synapse` model go in this direction, leading to a more comparable behavior with respect to the `tsodyks_synapse` model.

Appendix D

Additional tests on the working memory spiking model

D.1 Changes of simulation time step and connectivity

The working memory spiking network model described in Chapter 7 is designed in order to be similar to the one proposed in the original work of [23], however, some information regarding the network was missing. For instance, it is not known the simulation time step employed in the original work, in which both neuron and synaptic dynamics are integrated using the Euler scheme. In this work, the neuron model employed is integrated using the exact integration method of [13]. We decided to perform network simulations using a time step of $dt = 0.05\,\mathrm{ms}$, but we performed some tests in order to see whether a change in the value of the time step would have entailed a significant difference in the network dynamics. In particular, we performed simulations with $dt = 0.1\,\mathrm{ms}$ and $dt = 0.01\,\mathrm{ms}$. In the following Figure D.1 will be shown a similar simulation to the one proposed in Figure 7.2B, when the network is able to show an autonomous and synchronous spiking activity (i.e. the population spikes) using $dt = 0.1\,\mathrm{ms}$, $dt = 0.05\,\mathrm{ms}$ and $dt = 0.01\,\mathrm{ms}$.

As can be noticed, the behavior of the network using different simulation time steps is almost identical, with small differences regarding the time at which the population spikes occur. Also, the histograms of the firing rate difference between the spontaneous activity and the delay period are comparable.

Regarding connectivity, in NEST it is possible to have more than one connection between two neurons (called *multapses*) and self-connections (called *autapses*). We performed all the simulations enabling them (as it is the default option in the NEST simulator), however in the original model the possibility of having such connections was not specified. In Figure D.2 it will be shown that disabling this option when building the model connectivity, the results are comparable.

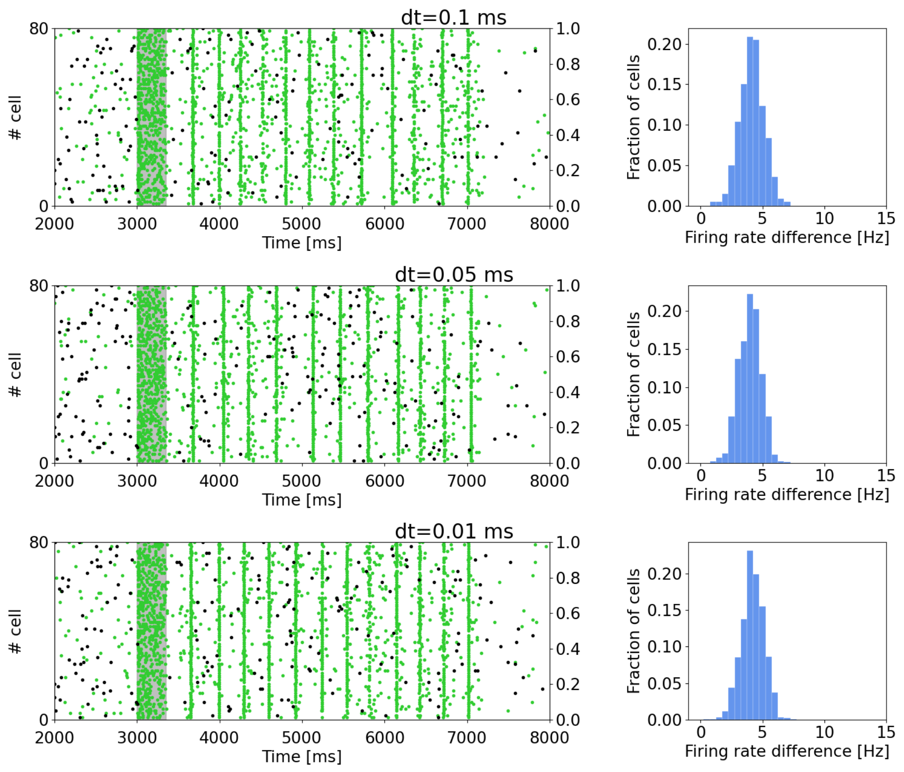


Figure D.1: **(Left)** Raster plots of a subset of neurons of a targeted selective population (green) and a non-targeted one (black) for different values of simulation time step dt. **(Right)** Histograms representing the difference in firing rate between the delay period and the spontaneous state for the selective population targeted by the item loading signal. The external background input diminishes at 7.2 s. Figure from [3].

D.2 Stochastic integration methods for neuronal dynamics integration: a results' comparison

As reported in Section 7.2.1, the LIF neuron model employed in the network is described by the ODE system:

$$
\begin{aligned}
\tau_m \frac{dV_j}{dt} &= -V_j + R_m(I_j^{\text{exc}} + I_j^{\text{inh}} + I_{\text{ext},j}) \\
\tau_{\text{exc}} \frac{dI_j^{\text{exc}}}{dt} &= -I_j^{\text{exc}} + \sum_i \alpha J_{i,j}(t) \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j}) \\
\tau_{\text{inh}} \frac{dI_j^{\text{inh}}}{dt} &= -I_j^{\text{inh}} + \sum_i \alpha J_{i,j} \sum_s \delta(t - t_s^{(i)} - \hat{\delta}_{i,j})
\end{aligned}
\tag{D.1}
$$

where τ_m is the membrane time constant, V_j is the neuron's membrane potential, R_m is the membrane resistance, I_j^{exc} and I_j^{inh} the excitatory and inhibitory synaptic current received as input from the connections within the other neurons of the network and $I_{\text{ext},j}$ represents the external input to the network, modeled as Gaussian white noise. The ODE system (D.1) is integrated following the exact

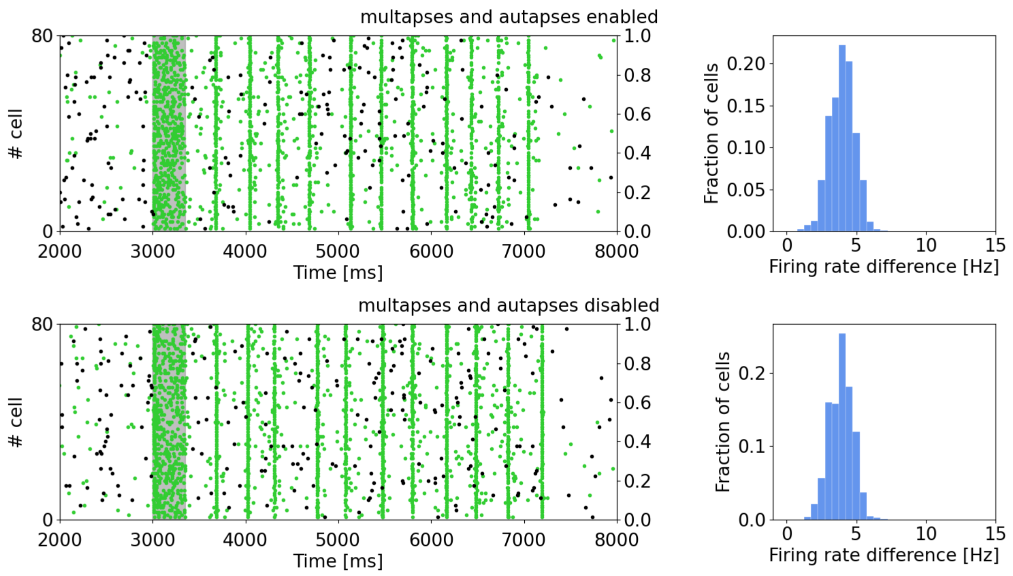


Figure D.2: **(Left)** Raster plots of a subset of neurons of a targeted selective population (green) and a non-targeted one (black) enabling or disabling the possibility of having multiple connections between two neurons and self-connections (called multapses and autapses respectively). **(Right)** Histograms representing the difference in firing rate between the delay period and the spontaneous state for the selective population targeted by the item loading signal. The external background input diminishes at $7.2\,\mathrm{s}$. Figure from [3].

integration scheme of [13], assuming that the external current $I_{\text{ext},j}$ is piecewise constant over time intervals of width Δt_{ng}. An alternative approach can be developed by exploiting the theory of stochastic differential equations and the methods for their numerical solution.

Stochastic differential equations are generally defined as:

$$dX = a(t, X)dt + b(t, X)dW \tag{D.2}$$

where dW represents the stochastic term and refers to the infinitesimal increment of the random walk of a Wiener process W, so that $dW = \sqrt{dt}G_n$, with G_n random number extracted from a Gaussian standard distribution. Terms $a(t, X)$ and $b(t, X)$ are respectively called drift coefficient and diffusion coefficient.

Indeed, the LIF model sub-threshold dynamics shown in Equation (D.1) can be described with a sum between a stochastic term (i.e. the Gaussian fluctuation of the external input) and a non-stochastic term. Expanding the term $I_{\text{ext},j}$ and taking the finite difference of V_j over a small time step Δt, the first equation of the system (D.1) can be rewritten as:

$$\tau_m \Delta V_j = (-V_j + R_m I_j^{\text{exc}} + R_m I_j^{\text{inh}} + R_m \mu_{\text{ext}})\Delta t + R_m \sigma_{\text{ext}} G_n \Delta t \tag{D.3}$$

where G_n is a random number extracted from a standard Gaussian distribution, whereas μ_{ext} and σ_{ext} are respectively mean and standard deviation of the external input signal. Defining $\sigma_{\text{ext}} = \kappa_{\text{ext}}/\sqrt{\Delta t}$, the last term in the left side of Equation (D.3) can be written as:

$$R_m \kappa_{\text{ext}} \Delta W_n \tag{D.4}$$

where ΔW_k is the variation of a Wiener process

$$\Delta W_n = \sqrt{\Delta t} G_n \tag{D.5}$$

Within each time step, equations for I_j^{exc} and I_j^{inh} can be solved analytically:

$$\begin{aligned} I_j^{\text{exc}}(t) &= I_j^{\text{exc}}(t_n) e^{-(t-t_n)/\tau_{\text{exc}}} \quad \text{for}\, t_n < t < t_{n+1} \\ I_j^{\text{inh}}(t) &= I_j^{\text{inh}}(t_n) e^{-(t-t_n)/\tau_{\text{inh}}} \quad \text{for}\, t_n < t < t_{n+1} \end{aligned} \tag{D.6}$$

By substituting the solutions for $I_j^{\text{exc}}(t)$ and $I_j^{\text{inh}}(t)$, Equation (D.3) assumes the form of Equation (D.2) with $a(V_j, t) = (-V_j + R_m I_j^{\text{exc}} + R_m I_j^{\text{inh}} + R_m \mu_{\text{ext}})/\tau_m$ and $b = R_m \kappa_{\text{ext}}/\tau_m$.
Indeed, Equation (D.3) is a stochastic differential equation (SDE), which can be numerically integrated using specifically designed numerical techniques. In this Section we present the comparison of the network behavior integrated following the exact scheme of [13] (as in Chapter 7) with respect to the behavior observed by using a specific SDE numerical integration technique, which will be presented in the next few lines.
Such a differential equation can be numerically integrated using the Euler-Maruyama method, which is an extension of the Euler method for SDE. According to this method, the solution for X in Equation (D.2) is defined as follows:

$$X(t_{n+1}) = X(t_n) + a(t_n, X(t_n))\Delta t + b(t_n, X(t_n))\Delta W_n \tag{D.7}$$

where $\Delta W_n = \sqrt{\Delta t} G_n$, and the time interval $[0, T]$ in which the equation is integrated is divided into N equal intervals t_n, so that $t_{n+1} - t_n = \Delta t = T/N$. As can be noticed from Equation (D.7), this integration method leads to a systematic error, since drift and diffusion coefficient are considered constant over each time interval $[t_n, t_{n+1}]$ with the values that they have at the beginning of the interval t_n. This error can be mitigated by reducing the time step, at the expense of the computational cost of the simulations. The Euler–Maruyama method has a weak order of convergence 1, however, it has a strong order of convergence $1/2$ [162].
A more precise class of methods has been derived through extensions of the Runge-Kutta method. In particular, [163] proposes an implementation of the Runge-Kutta method for SDEs which has both weak and strong order of convergence 1, and does not produce the systematic error of the Euler–Maruyama method mentioned above. Given a SDE as the one of Equation (D.2), a time step Δt so that $t_{n+1} = t_n + \Delta t$ and the value $X(t_n) = X_n$, the value for the subsequent time step $X(t_{n+1}) = X_{n+1}$ is determined by:

$$X_{n+1} = X_n + \frac{1}{2}(K_1 + K_2) \tag{D.8}$$

where

$$\begin{aligned} K_1 &= a(t_n, X_n)\Delta t + (\Delta W - S_n\sqrt{h}) b(t_n, X_n) \\ K_2 &= a(t_{n+1}, X_n + K1)\Delta t + (\Delta W - S_n\sqrt{h}) b(t_{n+1}, X_n + K1) \end{aligned} \tag{D.9}$$

where $\Delta W_n = \sqrt{\Delta t} G_n$, with G_n random number extracted from a standard normal distribution and S_n is a coefficient whose value depends on the SDE integration scheme. In particular, the Itô integration scheme can be achieved by assigning S_n the values $+1$ and -1 with equal probability, while the Stratonovich integration scheme is obtained by setting $S_n = 0$ [163]. In this work, we used the Stratonovich scheme, which is more appropriate for systems with external noise [164].

In Figure D.3 the raster plot for the network simulation using SDE Runge-Kutta scheme is compared with respect to the simulation obtained using the exact integration method of [13]. In particular, the simulation is performed using the background current needed to show the synchronous spiking activity during the delay period (as the one shown in Figure 7.2B).

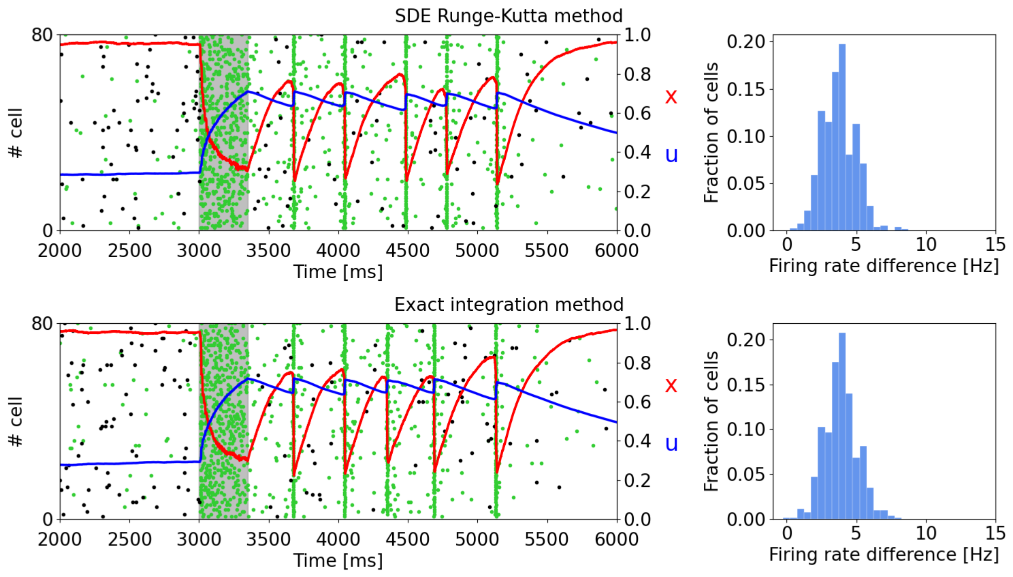


Figure D.3: Simulations of the network using different numerical integration techniques. **(Left)** Raster plots of a subset of neurons of a targeted selective population (green) and a non-targeted one (black). **(Right)** Histograms representing the difference in firing rate between the delay period and the spontaneous state for the selective population targeted by the item loading signal. Figure from [3].

As can be noticed, the network behavior observed using the SDE Runge-Kutta integration method (top panel of Figure D.3) is comparable to the behavior of the network obtained using the exact integration method of [13] (bottom panel of Figure D.3), which is the method used for all the simulations presented in Chapter 7.

D.3 Applicability of this model on a sequence learning network

As discussed in Chapter 7, during my research stay at the Jülich Research Center I worked on a possible implementation of a working memory spiking network such as the one described in this thesis on a hierarchical spiking model of sequence learning. Indeed, such a model (see [146]) is based on some biological assumptions: elements of the sequence are learned through a form of structural Hebbian plasticity and neurons are able to predict the following element of a sequence through a mechanism of nonlinear synaptic integration mimicking the generation of dendritic action potentials (dAPs) which, together with a winner-take-all mechanism, enables the neuron encoding the following element to be the only ones showing a synchronous event of spiking activity.

For this model to be able to process sequences of sequences, a further mechanism able to encode an entire sequence for a limited time is needed. In this regard, we provide here a biologically plausible simulation protocol according to which the STP mechanism for working memory can be employed. In Figure 7.2A it is shown that a memory-specific response can be triggered by a readout signal (i.e., a small nonspecific excitatory signal). Furthermore, the dAP can last for around 100 ms and can be used as a mechanism to predict the next element of a sequence (or the next sequence, depending on the hierarchy).

Let us assume that in the neurons of the working memory model the dAP mechanism is implemented[1]. By injecting a readout signal in the working memory model, we have a memory-specific response that can lead to the formation of dAPs. If we load an element or sequence right after, the dAP previously created can lead to the emission of spikes from specific neurons that represent the following element of the sequence (or the following sequence, depending on the hierarchical level), providing a prediction. To better explain the mechanism, Figure D.4 shows a schematic representation of the simulation protocol for a sequence of two elements.

This mechanism can be explained by the fact that attention control might be of thalamic origin. According to [165], which focused on visual attention, visual stimuli are projected, from the thalamus, not only to the visual area V1 (i.e., the first hierarchical level) but also to the visual area V2 (i.e., the second hierarchical level). Thus V2 receives a fast unspecific excitatory signal from the thalamus, and a slow specific excitatory stimulus from V1 (elaborated from the thalamic stimulus of V1). These signals can serve, respectively, as readout signals that activate the population encoding the current element of a sequence and item loading signal for the following element.

In order to make the working memory spiking network suitable for such a process, we have to adapt it to the TM model presented in [146]. For instance, the model should be targeted by a sequence-like item loading signal to mimic the encoding of a sequence of elements. Further, the TM model is able to encode an item in populations of a few tens of neurons, which implies that the size of the selective populations of the working memory model has to be strongly diminished. The next sections will discuss the results of the tests performed in this regard.

[1]Currently, the neurons of the working memory model are not provided with the dAP mechanism, and all the simulations shown here are based on the model described in Chapter 7.

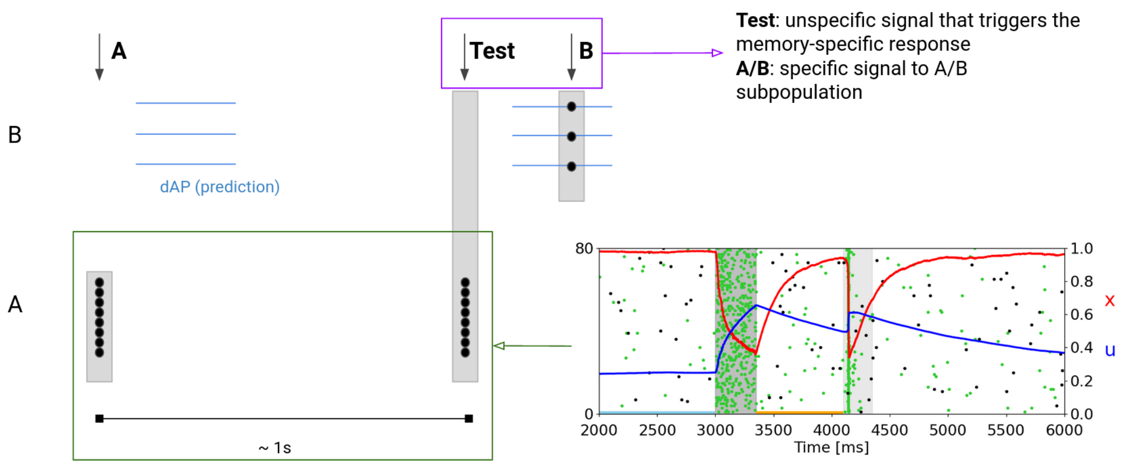

A
Test
B
Test: unspecific signal that triggers the memory-specific response
A/B: specific signal to A/B subpopulation
B
dAP (prediction)
A
~ 1s
80
0
2000
2500
3000
3500
4000
4500
5000
5500
6000
Time [ms]
1.0
0.8
0.6
0.4
0.2
0.0
x
u

Figure D.4: Schematic representation of the usage of the STP-driven mechanism in a sequence learning model. Let us consider a sequence of two elements (A, B). The element *A* is injected into the network at the beginning of the simulation and the network can retrieve a memory-specific response when a readout signal is injected. When the element *B* is injected, a first unspecific excitatory signal targets the whole network, triggering the memory-specific response (playing the role of a readout signal). This activates the dAP (sky blue lines) for the neurons of the selective population representing *B* connected with neurons representing the previous element. Thus, when the specific signal arrives at the neurons encoding the element *B*, the subset of neurons is able to emit a spike before the rest of the population, winning the WTA-like competition.

D.3.1 Sequential stimulation

Here selective populations are targeted with "sequential" item loading signals. Thus, having an item as a set of N sequences, the item loading time is divided into N intervals during which only a fraction of $1/N$ neurons of the population are targeted with the stimulus. Right after the time interval, the targeted neurons stop having the additional current and another subset of neurons is targeted, until the very end of the time dedicated to the item loading signal. Figure D.5 shows how the new item loading is designed.

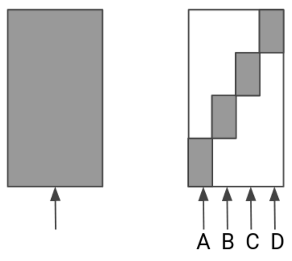


Figure D.5: Item loading signal using a sequential stimulus of 4 elements. The standard item loading signal (left), targets all the neurons of the selective population. The sequential input (right) targets a subset of neurons at different times. The total duration of the input is the same as the standard item loading signal.

This way, looking at the average values of the STP variables u and x, the value of u of the first set of targeted neurons rises only during the injection of the signal A (see Figure D.5), and so on. This means that the amplitude of the sequence signal can need to be increased to grant an enhancement of the synaptic efficacy suitable for triggering the memory-specific response. Figure D.6 shows an example of sequence-like item loading.

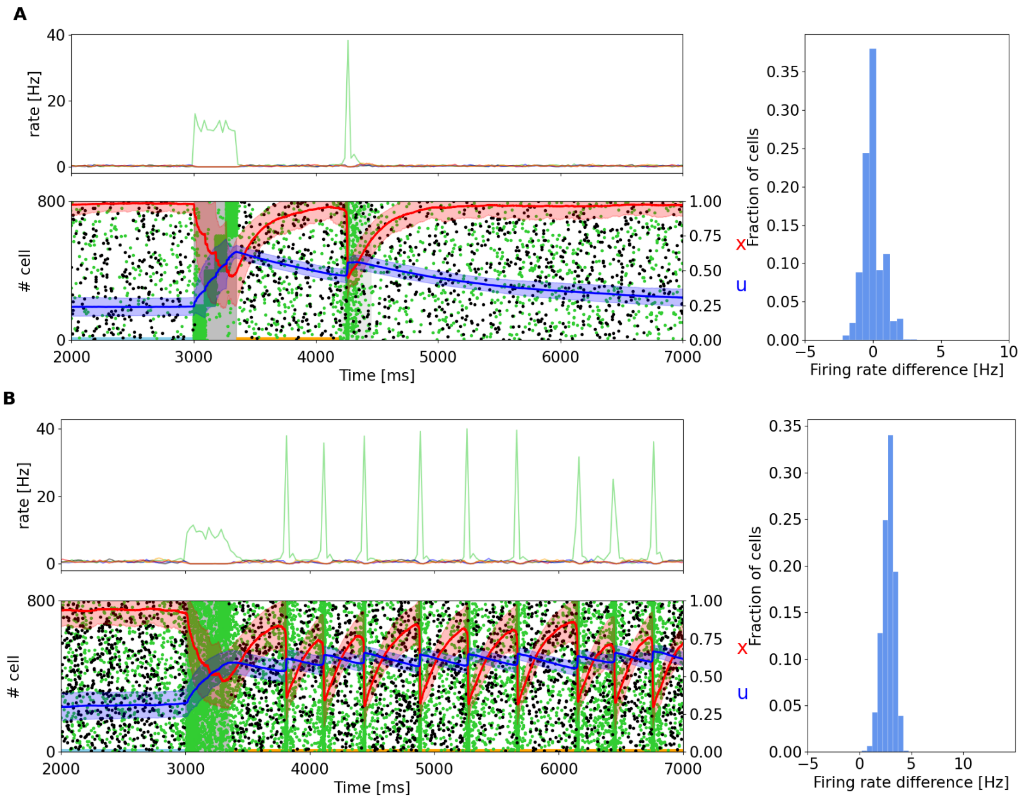


Figure D.6: Network behavior using a sequence-like item loading stimulus. Bottom-left and right panels are the same as Figure 7.2, whereas the top-left panel indicates the firing rate of each selective population. **(A)** Simulation where the readout signal is needed. Here, having a lower background input current, the sequential input is clearly visible. We used $\mathcal{A}_{\text{cue}} = 1.6$. **(B)** Simulation where the population spikes show up autonomously. Here, $\mathcal{A}_{\text{cue}} = 1.15$.

For 5 elements per item loading signal, the results would be almost identical. Generally, when increasing the number of elements, it should be increased the total time of the item loading, or the amplitude, or both.

We also studied the case in which a sequence element is missing. First of all, we verified that the position of the missing element does not have an influence on the final result. Indeed, the prediction mechanism of the TM model is sensitive to the element position, but the only STP mechanism should not have an order-dependent response. Here is an example of network behavior when the third element is missing.

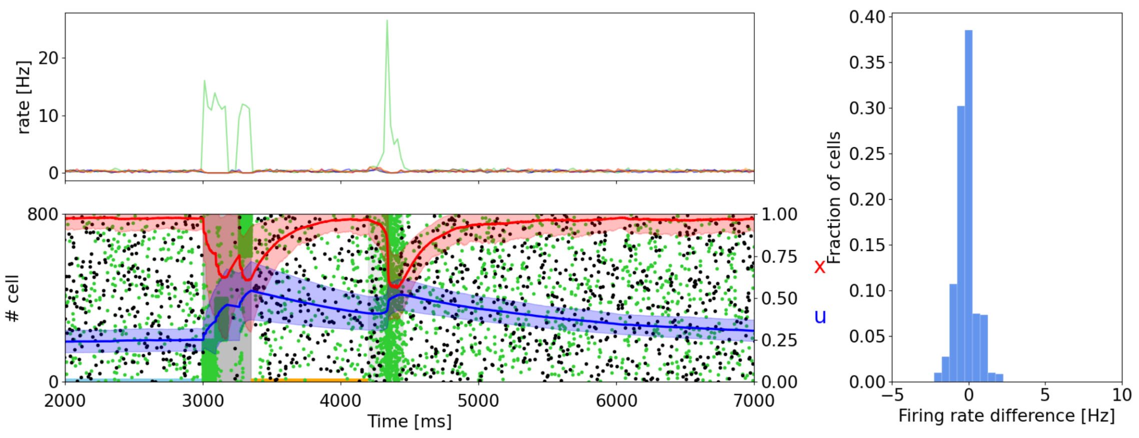


Figure D.7: Network behavior using a sequence-like item loading stimulus with a missing element of the sequence.

Interestingly, even if an element is missing, and thus part of the neurons is not stimulated by the item loading signal, the whole neuron population retrieves the memory when triggered by a readout signal. Indeed, the further missing element is not able to properly encode the sequence in the selective population. We thus decided to evaluate the readout performance. 5-elements item loading signal are given with a variable number of missing elements. The missing elements are at random position, since it was previously verified that the order of the element is independent from the final response. We changed slightly the duration of the total stimulus in order to see how much its length can be relevant, and we performed 10 simulations using different seeds for random number generation to see how many simulations show the memory-specific response. The next plot shows the results obtained.

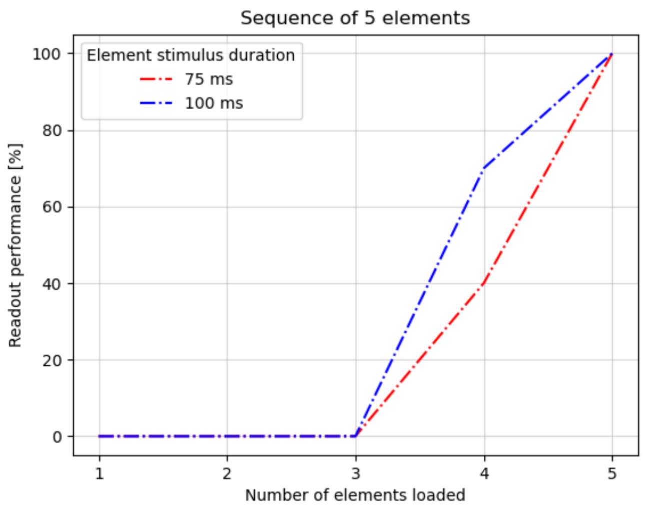


Figure D.8: Readout performance of the network when giving sequences of 5 elements as input with a different number of elements of the sequence actually loaded. The two lines are obtained by simulating the network using 10 different seeds for random number generation and using different element durations.

As we can expect, a shorter stimulation leads to a minor stimulation of the selective population, leading to a lower readout performance. Indeed, these performances can change as a function of the duration of a sequence element stimulus, the number of elements of the sequence, and the amplitude of the stimulus, and further tests will be needed to measure the readout performance with parameters similar to the ones of the TM model.

D.3.2 Selective populations down-scaling

We also worked on the down-scaling of the selective populations. The TM model encodes a sequence element using a population of around 20-30 neurons. Since the working memory network has a total of 10000 neurons, with 800 excitatory neurons per selective population, the idea is to fix the total size of the network and diminish the size of the selective populations gradually until we reach a few tens of neurons. This choice leads to encoding a larger number of items and also to retrieve a memory using even fewer neurons, resulting in a more efficient mechanism.

Together with this change, we also decided to shorten the duration of the item loading and readout stimuli (i.e., $\mathcal{T}_{\text{cue}}$ and $\mathcal{T}_{\text{reac}}$ respectively), compensating this change by increasing the contrast factors (ergo the amplitude of the external stimuli). This was done to use parameters more similar to the one of the TM model, which receives shorter but stronger inputs. To do so, we tried fixing the same amount of charge deposited when injecting the external signal, to keep constant the product of duration and contrast factor.

The network described in Chapter 7 has $p = 5$ selective populations, so that using the parameters of [3], half of the excitatory population is part of selective populations, with the other half being non-selective. However, even when increasing the number of selective populations, the network works in a similar way. In order to properly downscale the selective populations, we adjusted the following parameters:

- potentiated and baseline excitatory-to excitatory synaptic weights, usually increasing J_{p};
- synaptic time constant (default $\tau_s = 2\,\text{ms}$), to be increased to enhance the effect on the postsynaptic potential from a single spike;
- excitatory background current, to be increased;
- contrast factors for item loading and readout signals, to be increased.

Basically, since this model relies on WTA mechanism and population spikes are led to random fluctuation of spiking activity, increasing the excitatory drive to the selective populations is fundamental. This can be achieved both by increasing the background and the external signals and increasing the weight of the spikes coming from neurons belonging to the same population, since a decrease in the number of neurons leads to a decrease of the in-degrees from the same selective population. To this end, another potentially useful parameter is the connectivity level, which can be changed from 0.2 (i.e., the default value) to 1.0 (i.e., an all-to-all connectivity for the selective population).

The following figure shows the raster plot of the network when selective populations have a size of 50 neurons. To obtain such a result, we used the following parameters: $\eta_X^E = 26.5\,\text{mV}$, $\tau_{\text{syn}} = (5.0, 5.0, 2.0)\,\text{ms}$ (respectively, the synaptic time constant for intra and inter-population excitatory-to-excitatory connections, and connections between excitatory and inhibitory neurons), $J_b = 0.01\,\text{mV}$, $J_p = 1.15\,\text{mV}$.

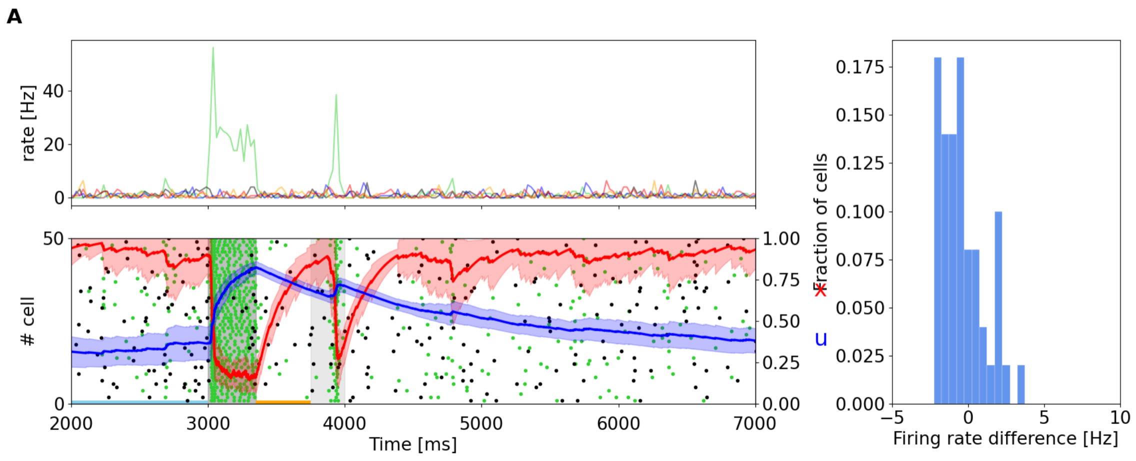


Figure D.9: Raster plot of the spiking activity for a network with selective populations of 50 neurons. The readout signal starts at $3750.0\,\text{ms}$.

Such a result can also be obtained by increasing the connectivity level, which in this example has not been changed. Moreover, although the STP parameters have not been changed, the memory specif response here starts showing up around $550\,\text{ms}$ after the item loading signal, instead of the usual $900\,\text{ms}$.

Appendix E

Lognormal distribution of the firing rate

The theoretical treatment of the continuous model proposed in this thesis is valid for a generic firing rate probability distribution. However, the model validation presented in the result section is focused on a lognormal distribution, which is a continuous probability distribution of a random variable ν whose logarithm $\ln(\nu)$ is normally distributed. The probability density function of this distribution is

$$\rho_{\mathrm{LN}}(\nu) = \frac{1}{\sqrt{2\pi}\sigma\nu} \cdot \exp\Big(-\frac{(\ln(\nu) - \mu)^2}{2\sigma^2}\Big) \tag{E.1}$$

where μ and σ are the mean and standard deviation of $\ln(\nu)$. Expanding Equation (8.22) using Equation (E.1) we have

$$\begin{aligned} \langle\nu_\ell\rangle &= \frac{1}{q_1}\int_{-\infty}^{y_t} \nu(y)G_{\sigma,\mu}(y)dy = \frac{1}{q_1}\int_{-\infty}^{y_t} e^y \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} dy \\ \langle\nu_{\mathrm{h}}\rangle &= \frac{1}{p_1}\int_{y_t}^{\infty} \nu(y)G_{\sigma,\mu}(y)dy = \frac{1}{p_1}\int_{y_t}^{\infty} e^y \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} dy \end{aligned} \tag{E.2}$$

where y is a variable representing the logarithm of the firing rate, $y = \ln(\nu)$, and follows a normal distribution $G_{\sigma,\mu}(y)$, while y_t represents the value linked to the threshold value on the rate ν_t ($y_t = \ln(\nu_t)$).
In the logarithmic representation the area of the portion of the Gaussian $G_{\sigma,\mu}(y)$ having $y < y_t$ corresponds to the probability that a neuron has a low rate, q_1. Therefore we can write:

$$q_1 = \int_{-\infty}^{y_t} G_{\sigma,\mu}(y)dy = \frac{1}{2} + \int_{\mu}^{y_t} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy \tag{E.3}$$

Substituting $x = \frac{y-\mu}{\sqrt{2}\sigma}$ we obtain:

$$q_1 = \frac{1}{2} + \int_0^{\frac{y_t-\mu}{\sqrt{2}\sigma}} \frac{1}{\sqrt{\pi}} e^{-x^2} dx = \frac{1}{2} + \frac{1}{2}\mathrm{erf}\Big(\frac{y_t-\mu}{\sqrt{2}\sigma}\Big) \tag{E.4}$$

where with erf(x) we indicate the error function, defined as:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{E.5}$$

and then

$$y_t = \mu + \sqrt{2}\sigma \mathrm{erf}^{-1}(2q_1 - 1) \tag{E.6}$$

where erf^{-1} is the inverse of the erf function. By substituting $z = y - \mu$ we can rewrite ν_h from Equation (E.2) as:

$$\begin{aligned}\langle \nu_h \rangle =& \frac{1}{p_1} \int_{y_t-\mu}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{z+\mu-\frac{z^2}{2\sigma^2}} dz = \frac{1}{p_1} \frac{e^{\mu}}{\sqrt{2\pi\sigma^2}} \int_{y_t-\mu}^{\infty} e^{-\frac{z^2-2\sigma^2 z}{2\sigma^2}} dz = \\ =& \frac{1}{p_1} \frac{e^{\mu}}{\sqrt{2\pi\sigma^2}} \int_{y_t-\mu}^{\infty} e^{-\frac{(z-\sigma^2)^2-\sigma^4}{2\sigma^2}} dz = \frac{1}{\sqrt{2\pi}} \frac{e^{\mu+\frac{\sigma^2}{2}}}{\sigma p_1} \int_{y_t-\mu}^{\infty} e^{\frac{-(z-\sigma^2)^2}{2\sigma^2}} dz\end{aligned} \tag{E.7}$$

Making a further substitution $\xi = \frac{z-\sigma^2}{\sqrt{2}\sigma}$ finally we find:

$$\begin{aligned}\langle \nu_h \rangle &= \frac{1}{\sqrt{\pi}} \frac{e^{\mu+\frac{1}{2}\sigma^2}}{p_1} \int_{\frac{y_t-\mu-\sigma^2}{\sqrt{2}\sigma}}^{\infty} e^{-\xi^2} d\xi = \\ &= \frac{1}{\sqrt{\pi}} \frac{e^{\mu+\frac{1}{2}\sigma^2}}{p_1} \Big(\int_0^{\infty} e^{-\xi^2} d\xi - \int_0^{\frac{y_t-\mu-\sigma^2}{\sqrt{2}\sigma}} e^{-\xi^2} d\xi \Big) = \\ &= \frac{e^{\mu+\frac{1}{2}\sigma^2}}{p_1} \Big[\frac{1}{2} - \frac{1}{2} \mathrm{erf}\Big(\frac{y_t-\mu-\sigma^2}{\sqrt{2}\sigma} \Big) \Big] = \\ &= \frac{\langle \nu \rangle}{2p_1} \Big[1 - \mathrm{erf}\Big(\mathrm{erf}^{-1}(2q_1-1) - \frac{\sigma}{\sqrt{2}} \Big) \Big]\end{aligned} \tag{E.8}$$

with $\langle \nu \rangle$ indicating the average rate, which for the lognormal distribution is given by the known expression

$$\langle \nu \rangle = e^{\mu+\frac{1}{2}\sigma^2} \tag{E.9}$$

With similar steps we obtain the expression of $\langle \nu_l \rangle$

$$\langle \nu_l \rangle = \frac{\langle \nu \rangle}{2q_1} \Big[1 - \mathrm{erf}\Big(\mathrm{erf}^{-1}(2p_1-1) - \frac{\sigma}{\sqrt{2}} \Big) \Big] \tag{E.10}$$

From these two equations we can finally derive the relationships between σ, q_1, ν and ν_h or ν_l respectively:

$$\sigma = \sqrt{2}\Big[\mathrm{erf}^{-1}(2q_1-1) - \mathrm{erf}^{-1}\Big(1 - \frac{2p_1\langle \nu_h \rangle}{\langle \nu \rangle} \Big) \Big] \tag{E.11}$$

$$\sigma = \sqrt{2}\Big[\mathrm{erf}^{-1}(2p_1-1) - \mathrm{erf}^{-1}\Big(1 - \frac{2q_1\langle \nu_l \rangle}{\langle \nu \rangle} \Big) \Big] \tag{E.12}$$

Using the equation (E.9) we can rewrite μ as:

$$\mu = \ln(\langle\nu\rangle) - \frac{\sigma^2}{2} \tag{E.13}$$

The average rate $\langle\nu\rangle$ can also be expressed as a function of $\langle\nu_{\mathrm{h}}\rangle$ and $\langle\nu_\ell\rangle$:

$$\langle\nu\rangle = p_1\langle\nu_{\mathrm{h}}\rangle + q_1\langle\nu_\ell\rangle \tag{E.14}$$

The latter equations allow us to express the parameters of the lognormal distribution σ and μ as a function of the parameters of the model, p_1, $\langle\nu_{\mathrm{h}}\rangle$ and $\langle\nu_\ell\rangle$.

Appendix F

Estimation of the variance of k

In this appendix, we compute the variance of the number of consolidated connections in input to a neuron of $\mathcal{P}_2$ (i.e., σ_k^2) which, as we have seen previously, enters the formula for the variance on the background signal. For the calculation we use the table below which represents the two states, rate high (1) or rate low (0), for a single neuron of the population $\mathcal{P}_2$ and for the presynaptic neurons of its input connections in a complete simulation over $\mathcal{T}$ patterns.

t	$\mathcal{O}$	$\mathcal{I}_0$	$\mathcal{I}_1$	....	$\mathcal{I}_{k-1}$	$\mathcal{I}_k$	....	$\mathcal{I}_{\mathcal{C}-1}$
0	1	x_{00}			$x_{k-1,0}$	0		0
1	1	..			..	0		0
2	1	..			..	0		0
..	..	..			..	..		..
m-1	1	x_{0m}			$x_{k-1,m-1}$	0		0
..	0							
..	..							
$\mathcal{T}-1$	0							

Table F.1: Table representing the two states rate high (1) or rate low (0) for a single neuron of the population $\mathcal{P}_2$ and for the presynaptic neurons of its input connections in a complete simulation. Each row represents a training pattern, with index ranging from 0 to $\mathcal{T}-1$. The first two columns represent the training pattern index t and the rate level $\mathcal{O}$, high or low, of the $\mathcal{P}_2$ neuron. The other columns $\mathcal{I}_j$ represent the rate level, high or low, of the presynaptic neurons connected to the neuron of $\mathcal{P}_2$ through its C incoming connections. The entries for rate levels can be 0 or 1 for low rate and for high rate respectively; in case of continuous distribution of the rate, the two levels correspond to a rate over or under the threshold ν_{t}. The table shows the case in which the $\mathcal{P}_2$ neuron is in the high-rate level for the first m examples and in the low-rate level for $\mathcal{T} - m$ examples, while the last $\mathcal{C} - k$ presynaptic neurons are in the low-rate level for the first m examples.

Given the scheme of Table F.1, we call:

- p_1: probability that a neuron of $\mathcal{P}_1$ is in the high-rate level, i.e. probability that a cell of a column $\mathcal{I}_j$ is equal to one;
- p_2: probability that the neuron of $\mathcal{P}_2$ is in the high-rate level for a given example, i.e., probability that a cell of the column $\mathcal{O}$ is equal to one;
- p_2^m: probability that the neuron of $\mathcal{P}_2$ is in the high-rate level for the first m patterns;
- $(1-p_2)^{\mathcal{T}-m}$: probability that the neuron of $\mathcal{P}_2$ is in the low-rate level for the remaining $\mathcal{T}-m$ patterns;
- $(1-p_1)^m$: probability that a neuron of $\mathcal{P}_1$ is in the low-rate level for the first m patterns;
- $1-(1-p_1)^m$: probability that a neuron of $\mathcal{P}_1$ is in the high-rate level for at least one pattern out of the first m;
- $[1-(1-p_1)^m]^k$: probability that every neuron of $\mathcal{P}_1$ of the columns $\mathcal{I}_0$, ..., $\mathcal{I}_{k-1}$ is above threshold for at least one pattern among the first m;
- $(1-p_1)^{m(\mathcal{C}-k)}$: probability that every neuron of $\mathcal{P}_1$ of the last $\mathcal{C}-k$ columns is below threshold for the first m patterns.

Now we can combine all these results to calculate the probability that one neuron of $\mathcal{P}_2$ and k presynaptic neurons of its input connections are high level for m generic patterns (i.e., not necessarily the first m). To do this we have to take into account that the neuron of $\mathcal{P}_2$ will not necessarily be at the high level in the first m examples and that the neurons of $\mathcal{P}_1$ at the high level will not necessarily be the first k (as in the case shown in the table). For this, we have to use binomial coefficients that will take into account all possible combinations in the choice of m patterns out of all possible $\mathcal{T}$ patterns and in the choice of k presynaptic neurons out of a total of $\mathcal{C}$ connections:

$$Q(m,k) = \binom{\mathcal{T}}{m} p_2^m (1-p_2)^{\mathcal{T}-m} \binom{\mathcal{C}}{k} \; [1-(1-p_1)^m]^k \; (1-p_1)^{m(\mathcal{C}-k)} \tag{F.1}$$

The probability that k connections of a generic neuron of $\mathcal{P}_2$ are consolidated can be calculated by adding $Q(m,k)$ over all possible values of m:

$$P(k) = \sum_{m=0}^{\mathcal{T}} Q(m,k) \tag{F.2}$$

and the average number of consolidated connections can be calculated as:

$$\begin{aligned}\langle k\rangle &= \sum_{m,k} kQ(m,k) = \sum_m \binom{\mathcal{T}}{m} p_2^m (1-p_2)^{\mathcal{T}-m} \sum_{k=0}^{\mathcal{C}} k \binom{\mathcal{C}}{k} (1-q_1^m)^k q_1^{m(\mathcal{C}-k)} = \\ &= \sum_m \binom{\mathcal{T}}{m} p_2^m (1-p_2)^{\mathcal{T}-m} \mathcal{C}(1-q_1^m) = \\ &= \mathcal{C}\Big[\ \sum_m \binom{\mathcal{T}}{m} p_2^m (1-p_2)^{\mathcal{T}-m} - \sum_m \binom{\mathcal{T}}{m} (p_2 q_1)^m (1-p_2)^{\mathcal{T}-m}\Big] = \\ &= \mathcal{C}\Big[\ 1 - \sum_m \binom{\mathcal{T}}{m} (p_2 - p_1 p_2)^m (1-p_2)^{\mathcal{T}-m}\Big]\end{aligned} \tag{F.3}$$

where $q_1 = 1 - p_1$ and we have used the formula for the mean value of a binomial distribution:

$$\sum_{k=0}^{n} k \binom{n}{k} p^k (1-p)^{n-k} = np \tag{F.4}$$

Using the relationship:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} \tag{F.5}$$

we can get the expression of $\langle k\rangle$:

$$\langle k\rangle = \mathcal{C}[1 - (1-p_1p_2)^{\mathcal{T}}] \tag{F.6}$$

To calculate σ_k^2 we must also calculate $\langle k^2\rangle$:

$$\begin{aligned}\langle k^2\rangle &= \sum_{m,n} Q(m,k)k^2 \\ \langle k^2\rangle &= \sum_m \binom{\mathcal{T}}{m} p_2^m (1-p_2)^{T-m} \cdot \sum_{k=0}^{C} \binom{\mathcal{C}}{k} (1-q_1^m)^k q_1^{m(C-k)} k^2\end{aligned} \tag{F.7}$$

After some calculations, analogous to the case of $\langle k\rangle$, we obtain the following formula:

$$\langle k^2\rangle = C(C-1)[1 + p_1p_2(p_1-2)]^T - C(2C-1)(1-p_1p_2)^T + C^2 \tag{F.8}$$

Finally, the variance can be calculated from equations (F.6) and (F.8) as

$$\sigma_k^2 = \langle k^2\rangle - \langle k\rangle^2 \tag{F.9}$$

Appendix G

Noise addition during test phase

As anticipated in Section 8.4.3, in order to assess the generalization capacity of the model proposed in this theoretical framework, the test input patterns were generated starting from the corresponding training input patterns by adding noise with a given probability distribution. More specifically, each test pattern is generated by adding to the rate of the corresponding training pattern the contribution of a further extraction from a truncated Gaussian distribution $G(\eta)_{\mu_{\mathrm{T}},\sigma_{\mathrm{T}}}$. Therefore the single neuron rate in a test pattern will be given by the following formula:

$$\nu_{\mathrm{tot}} = \nu + \eta \tag{G.1}$$

where η is a rate driven by the distribution $G(\eta)_{\mu_{\mathrm{T}},\sigma_{\mathrm{T}}}$. The input signal to a neuron of the population $\mathcal{P}_2$ can be expressed as the scalar product between the vector $\vec{\mathcal{W}}$ of the weights and the vector $\vec{\nu}_{\mathrm{tot}}$ of the rates of the presynaptic neurons:

$$\vec{\mathcal{W}} \cdot \vec{\nu}_{\mathrm{tot}} = \vec{\mathcal{W}} \cdot \vec{\nu} + \vec{\mathcal{W}} \cdot \vec{\eta} \tag{G.2}$$

Since the noise distribution has zero mean, its contribution to the average values of the signals in input to the coding and non-coding neurons, $\langle \mathcal{S}_2 \rangle$ and $\langle \mathcal{S}_{\mathrm{b}} \rangle$, is zero. On the other hand, it affects the variance of the background signal, σ_{b}^2. Since ν and η are independent and random variables, the overall variance will be equal to the sum of the variance in the absence of noise σ_{b}^2 (see Equation (8.26)) plus the variance due to noise. Thus

$$\sigma_{\mathrm{b}}^{*2} = \sigma_{\mathrm{b}}^2 + \langle k \rangle (\mathcal{W}_{\mathrm{c}} \sigma_\eta)^2 + (C - \langle k \rangle)(\mathcal{W}_{\mathrm{b}} \sigma_\eta)^2 = \sigma_{\mathrm{b}}^2 + \sigma_\eta^2 \mathcal{C} \Big[p\mathcal{W}_{\mathrm{c}}^2 + (1-p)\mathcal{W}_{\mathrm{b}}^2 \Big] \tag{G.3}$$

where $\sigma_\eta^2 = \sigma_{\mathrm{T}}^2$ is the variance of $G(\eta)_{\mu_{\mathrm{T}},\sigma_{\mathrm{T}}}$. Truncating the Gaussian distribution in the symmetric interval $[-2\sigma, 2\sigma]$, the mean is zero, whereas the variance is

$$\sigma_{\mathrm{T}}^2 = \sigma^2 \Big[1 - \frac{4 \cdot e^{-2}}{\sqrt{2\pi}\,\mathrm{erf}(\sqrt{2})} \Big] \tag{G.4}$$

G.1 Calculation of the mean value of k'_t over the rewiring steps

In Section 8.4 we obtained the expression of $\mathcal{S}_2$ in the presence of rewiring and we observed that this depends on the parameter k'_t, given by (Equation (8.33)):

$$\langle k'_t \rangle = p_t \mathcal{C}(1 - p_1) \tag{G.5}$$

where, according to Eq. (8.34), p_t is given by

$$p_t = 1 - (1 - p_1 p_2)^t \tag{G.6}$$

In order to calculate the mean value of $\mathcal{S}_2$ for all patterns, k'_t should be averaged over all the values of the training index t for which the rewiring is performed, i.e.

$$t = si \qquad i = 0, \dots, \frac{\mathcal{T}}{s} \tag{G.7}$$

where s is the rewiring step and for simplicity we assume that $\mathcal{T}$ is a multiple of s and that there is a final rewiring after the last training step. The average of p_t over the rewiring values of t is

$$\langle p_t \rangle = \frac{\sum_{i=0}^{\mathcal{T}/s} 1 - [(1 - p_1 p_2)^s]^i}{\frac{\mathcal{T}}{s} + 1} = 1 - \frac{\sum_{i=0}^{\mathcal{T}/s} [(1 - p_1 p_2)^s]^i}{\frac{\mathcal{T}}{s} + 1} = 1 - \frac{bs}{\mathcal{T} + s} \tag{G.8}$$

where we introduced a parameter b defined as

$$b = \sum_{i=0}^{T/s} [(1 - p_1 p_2)^s]^i = \frac{1 - [(1 - p_1 p_2)^s]^{T/s+1}}{1 - (1 - p_1 p_2)^s} = \frac{1 - (1 - p_1 p_2)^{T+s}}{1 - (1 - p_1 p_2)^s} \tag{G.9}$$

Acknowledgements

I am deeply grateful to all those who have contributed to the successful completion of this thesis. Their support and guidance have been invaluable throughout this research journey.
First and foremost, I want to thank the referees of this thesis, Dott. Elena Pastorelli and Prof. Dr. Hans Ekkehard Plesser, for their meticulous review of the work and invaluable feedback.
I want to extend thanks to my supervisor, Bruno Golosio, for his support, expert guidance, and invaluable mentorship not only for the PhD but for my whole academic path. Your dedication to my academic and professional growth has been fundamental, and a simple acknowledgment section wouldn't be enough to express my gratitude. I would also like to express my sincere thanks to Pier Stanislao Paolucci, my co-supervisor, for his insightful feedback and support.
I want to thank the rest of NEST GPU team for their continuous collaboration during these years. In this regard, a special thanks goes to Jose Villamar, for his support during my stay in Germany and for all the time spent together first remotely and then in person.
A special thanks goes also to the ones that made my period abroad just great. First of all, I want to thank Tom Tetzlaff for his supervision and the numerous insights, and together with him I want to deeply thank Markus Diesmann, Johanna Senk, Melissa Lober, Anno Kurth, and Jasper Albers for welcoming me into their group, in which I spent one of the best times of my PhD.
Coming back to Cagliari, I want to deeply thank my PhD colleagues and Luca Sergi for sharing lunches, coffees, and lots of nonsense.