KØBENHAVNS UNIVERSITET

Machine-learning methods applied to inverse problems in motor neuronal networks

MSc Thesis in Computational Physics

Elias Najarro

Supervised by Henrik Lindén, Rune Berg & James Avery

Abstract

Balanced neuronal populations consisting of both excitatory and inhibitory neurons can generate slow and robust oscillations that are used to model the activity the spinal cord in mammals. A potential explanation comes from a recent finding that seemingly unstructured networks with an excitation-inhibition balance can generate slow and robust oscillations when the overall strength of synaptic connections are strong. In this regime, the network enters a limit cycle with distributed phases that is highly reminiscent of activity experimentally recorded in the spinal cord of turtles. However, previous models have used a rate-based formalism that neglects fast temporal dynamics due to action potential generation in individual neurons. Here we show that, under strong coupling conditions, spiking populations can display the same type of oscillatory behaviour previously seen in rate-based population models.

Neural motor populations generate activity in a functionally distributed manner, consequently, the high dimensionality of their dynamics makes traditional approaches for neural decoding insufficient. In order to tackle this shortcoming, we explore statistical and machine learning methods to better understand the dynamics and information processing regimes of simulated network models and investigate whether deep-learning approaches can provide a fruitful framework to build inverse models of simulated neuronal populations. Specifically, we make use of recurrent network models of biological neural populations -both rate and spiking- known to exhibit rich dynamics, and apply a combination of dimensionality reduction techniques and deep network architectures to build backward models. We show that these models are capable of decoding their input stimuli and the relevant model parameters, both from the raw neural activity as well as from the compact manifold representation of their dynamics.

Contents

List of Abbreviations

ANN	Artificial neural network
CNN	Convolutional neural network
CPG	Central Pattern Generators
DL	Deep learning
E/I balance	Excitatory-Inhibitory balance
E/I ratio	Excitatory-Inhibitory ratio
ESN	Echo state network
EPSP	Excitatory postsynaptic potential
FFT	Fast Fourier transform
GLIF	Generalized leaky integrate-and-fire
H&H	Hodgkin and Huxley (model)
IF	Integrate-and-fire (neuron)
IPSP	Inhibitory postsynaptic potential
ISI	Interspike interval
JIT	Just-in-time (compilation)
KL	Kullback-Leibler (divergence)
LFP	Local field potential
LIF	Leaky integrate-and-fire (neuron)
LSTM	Long short-term memory
ML	Machine Learning
MLP	Multilayer perceptron
MN	Motor neuron
PCA	Principal component analysis
PSP	Postsynaptic potential
RNN	Recurrent neural network
SL	Supervised learning
SNN	Spiking neural network
STDP	Spike-timing-dependent plasticity
t-SNE	t-distributed stochastic neighbour embedding

Acknowledgements

My gratitude goes to Henrik, Rune and the rest of the BergLab who provided with an enthusiastic and open research environment for writing this thesis; and to James who gracefully took me in after being nonchalantly abandoned by Brian, and whose contribution per-meeting ratio to this thesis could not have been greater. And finally, to Chunt, Kevin, Rugé, Ernesto, Usidore and, specially, some Buffalo without whom my world would be much smaller.

"I woke up as the sun was reddening; and that was the one distinct time in my life, the strangest moment of all, when I didn't know who I was - I was far away from home, haunted and tired with travel, in a cheap hotel room I'd never seen, hearing the hiss of steam outside, and the creak of the old wood of the hotel, and footsteps upstairs, and all the sad sounds, and I looked at the cracked high ceiling and really didn't know who I was for about fifteen strange seconds. I wasn't scared; I was just somebody else, some stranger, and my whole life was a haunted life, the life of a ghost."

On the Road, Jack Kerouac

Chapter 1

Introduction

In this chapter we present the motivation for this work and the research questions we have addressed. We then give an outline of the structure of this report and, finally, present an overview of the codebase used to implement and run our models.

1.1 Motivation & Problem Statement

The ability to gracefully perform complex movements is one of the most remarkable characteristics of animals, however the underlying neural principles of how the nerve system gives rise to it are beyond our current understanding. More generally, the language that neurons speak -the *neural code*- remains largely undeciphered. This thesis deals with two distinct neuroscientific open questions.

The first one is **how can rhythmic movement be generated by the spiking populations of the spinal cord**. Rhythmic movement, such as walking, swimming or breathing, is performed by the alternation of activity of flexor-extensor muscle pairs which by acting in phase opposition produce periodic movements. Traditionally, the neural activity guiding these muscles has been understood in terms of alternation of activity between distinctive populations of neurons in the spinal cord. In this view, two independent neural population are coupled to each other and produce alternating pattern of activity such that when one population is active the other remains quiet. Recent work by Lindén et al. [1] has sought to challenge that view and shown that the generation of motor activity in the spinal cord can be understood by means of a single neuronal population performing *rotation* -instead of alternation- in reduced neural space. However, their model is based on a rate-model formalism of neurons, which only accounts for the average firing rate of neurons, rather than describing the population dynamics in terms of spiking activity. This projects builds on the work and build a model of spiking neurons capable of exhibiting rotational dynamics in neural space analogous to that generated by their rate-based model and observed in experimental recordings in the spinal cord of turtles.

The second question addressed by this project is **how can we control the activity of populations of neurons such that we can generate different activity patterns at will** or, the equivalent problem, **how can we decode the parameters of a neural population model from its activity**. The question of the neural coding, namely, how to read and understand what a pattern of neural activity means, is one of the most fundamental questions in neuroscience. Traditionally, the decoding of neural activity -for instance, finding what movement is represented by a specific activity pattern- has been done using tools from control theory such as Wiener or Kalman filters. Recent work [2] has shown that deep learning models can outperform traditional decoders and can be effectively used to test whether a specific information is encoded in a signal [3]. We build on these works and formulate the decoding problem using the inverse problem formalism and use a class of computational models, known as *deep neural networks* to develop a framework to train models capable of both control, and decode, the simulated activity of neuronal populations.

1.2 Contributions

This project has resulted in three main contributions:

1.2.1 Spiking model of spinal cord circuitry

We have built a spiking neural population model which, under certain conditions, displays rotational dynamics, that is, the low dimensional representation of its spiking activity form a series of continuous circular orbits in reduced neural-space; this contrasts with *alternating dynamics* which, when represented in a low-dimensional space, abruptly jump between two state-space regions. To do so, we have incorporated insights from previous rate-based models shown to also exhibit rotational dynamics, as well as from existing spiking architectures, namely balanced spiking networks. As we show in Chapter 3, the resulting spiking population model is capable of exhibiting rotational dynamics. Although the generated activity does not yet fully match the electrophysiological activity seen in experimental recordings, this contribution is a stepping-stone towards spiking models of motor-generation circuitry in the spinal cord.

1.2.2 Deep-learning inversing models

We have developed a codebase to train inversing deep-learning models on neural activity. In Chapter 5, we demonstrate that the resulting model can be effectively used to decode the model parameters (ie., *inverse problems*) as well as be used to generate stimulation capable of inducing specific activity patterns in the neuronal populations (ie. *inverse optimal control*); even when the connectivity of the neural population is unknown to the model. The inversing models can be inputted both continuous activity, such as the oscillatory patterns generated by rate neurons, as well as the discrete activity produced by spiking populations.

We have conceived an online-learning variation of the backpropagation algorithm which disregard the notion of training vs validation sets and is specifically adapted to work with synthetic data, that is, when a generative model is available. Our developed codebase also includes the functionality needed to train models on with non-synthetic finite datasets.

Finally, although we demonstrate the model by decoding the activity of a neural population, the same codebase can be used to train an inverse model on any kind of sequential data by adapting a few functions.

1.2.3 Inverse models for neuroprosthetics & other applications

The intrinsic difficulty of making measures in *in vivo* systems and the invasiveness of perturbation methods have resulted in neuroscience seeing modest progress since its birth as an independent field. However, recent developments in multi-electrode neural probes and less-invasive perturbation techniques such as optogenetics have resulted in a explosion in the amount of neural data generated. However, the field is still adapting to this new data-driven era and it is still unclear how to best make use of this data abundance. In this context, we have designed an experimental protocol that utilizes the inverse modelling approach presented in this thesis to train end-to-end neuroprosthetics models. We present the protocol in Chapter 6 along with other applications of our models, such as how to empirically test a popular conjecture in motor neuroscience known as *manifold hypothesis*[1]

1.3 Report outline

In **Chapter** 2 we introduce the basic concepts from neuroscience needed to understand neural population models.

In **Chapter** 3 we start by briefly introducing the basics of how movement is generated in mammals. We then analyse the different dynamical regimes of rate-based modes and show how

[1]We further expand on the *neuroscience manifold hypothesis* in Chapter 3.

they can give rise to rotational dynamics and alternating muscle activity. Finally, we present our spiking population model, introduce two metrics to quantify the properties of its dynamical regimes, and demonstrate that the model can generate rotational dynamics and alternating muscle activity.

In **Chapter** 4 we introduce the notion of *manifold* as used in the neuroscience literature. We then discuss the relation between dynamics and the computations done by neural populations, and illustrate the different geometries generated by different dimensionality reduction techniques. We conclude by introducing the notion of *manifold induction* which is the base of our proposed manifold-hypothesis testing protocol that we introduce in Chapter 6.

In **Chapter** 5 we present the inverse-problem formalism as well as the basics of machine learning applied to supervised learning tasks. We then present our optimisation algorithm for applying DL to synthetic data. Finally, we demonstrate how the trained models can indeed decode the neural population parameters from their raw activity for both rate and spiking populations.

In **Chapter** 6 we introduce the two experimental protocols we have designed to apply our inverse-modelling approach to create neural prosthetics and to empirically test the *neuroscience manifold hypothesis*.

In **Chapter** 7 we provide with an outlook of potential future work.

1.4 Codebase overview

This project has been done as a thesis for a master in computational physics, as such, a big part of the work done has consisted in developing the code for the neural population simulations and for the machine learning models. Great attention has been paid to optimise the performance of the code. Although written in *Python*, computationally heavy operations have been vectorised using *Numpy* and, whenever possible, pre-compiled using *Numba* just-in-time (JIT) compiler; furthermore, the neural population simulation have been parallelised across CPU cores using Python's multiprocessing library and, as it's common practise, the training of the deep-learning models is executed in parallel in a GPU using Pytorch library.

The code is split across three GitHub repositories. In order to train a model, the three repositories should be placed in the same directory.

- The repository with the code to simulate the rate population. [2]
- The repository with the code to simulate the spiking population. [3]
- The repository with the code to train inversing DL-models. [4]

1.4.1 Rate-population simulations code overview

The rate-population repository contains the following files:

```
Rate network
├─ simulate_rate_model.py          # Main script to simulate populations
├─ read_nerve_activity.py          # Generates rotational dynamics and nerve read-out
└─ spinalsim.py                    # Main simulation and helper functions
```

[2]https://github.com/enajx/rate_network
[3]https://github.com/enajx/spiking_network
[4]https://github.com/enajx/predictive_network

```
  │
  ├── chaos_experiments_rate_model.py   # Run chaos experiments
  ├── time_simulation_benchmarking.py   # Run benchmarking experiments
  ├── requirements.txt                  # Libraries dependencies
  └── README.md                         # Instructions on how to the run code
```

The functions in `spinalsim.py` are based on the codebase from BergLab's former member Henrik Lindén. The main function is `simulate_rate_network` which implements the Euler method to solve the differential equation for each of the rate neurons of a population with connectivity `W` for a simulation duration of `t_steps`.

The file `simulate_rate_model.py` is the main script that allows to run simulations of rate neurons as well as generate different dimensionality-reduction techniques such as PCA, Kernel PCA or tSNE on the generated population activity.

The main options of the simulation script are shown below and can be display with `python simulate_rate_model.py --help`:

```
simulate_rate_model.py options:

  --number_neurons      Number of neurons being simulated. Not relevant for N experiment.
  --simulation_time     Simulation times, in ms.
  --connect_normal_dist If true, it samples weights from a normal distribution
  --connect_sparsity    Only applies if sampling connectivity from normal distribution
  --resampling_rate     Only applied to compute PCA/tSNE: resampling rate [1.0, 0[
  --PCA                 Run PCA on simulated activity
  --kernel_PCA          Run kernel PCA on simulated activity
  --tSNE                Run tSNE on simulated activity
  --dimensions          Dimensionality of the basis used to represent the generated activity
  --standarised         Standardised PCA dimensions
  --FT Run              Computes Fourier transform on simulated activity
  --burnout             % of initial points remove from PCA analysis
  --save_plots          Save plots
  --limited-input       Run simulation with only a brief 200ms input stimulus
                        driving the neurons
```

See readme.md in the repository for full instructions on how to run the code.

1.4.2 Spiking-population simulations code overview

The spiking population repository contains the following files:

```
  Spiking network
  ├── spiking_models.py        # Main script to run simulations
  ├── spiking_experiments.py   # Generate figures from section 4.3.2.2
  ├── utils_spiking.py         # Helper functions
  ├── other/*                  # Deprecated code
  ├── sync_regularity_data     # Results from sync/regularity experiments
  ├── requirements.txt         # Libraries dependencies
  └── README.md                # Instructions on how to the run code
```

The file `spiking_models.py` contains the spiking model and allows to run simulations for different network configurations as well as running PCA and generate nerve read-outs from the generated population activity.

The main options of the simulation script are summarised below and can be displayed with `python simulate_rate_model.py --help`:

```
spiking_models.py options:

  --network_type            Network model, only 'spinal' available
  --network_size            Number of neurons in the network
  --t_steps                 Simulation duration in ms
  --sparsity                Network sparsity - 1.0 fully connected, 0.1: 10% per neuron
  --coupling                Synaptic coupling between neuron [mV], if random matrix
                            it works as the std of the sampling distribution.
  --relative_inhib_strength Ratio between inhibition and excitation strength
  --plot                    Generate plots
  --convstd                 Std of Gaussian function used for convolving spikes
  --downsampling_convolved  Downsamples spike signals after having convolved them
  --plot_connectivity       Plot connectivity
  --random_connectivity     If true, a random connectivity matrix is manually generated
  --random_stim_current     If true, each neuron has a random constant stimulation (v_rest)
  --random_vm_init          Neuron membrane potentials are randomly initialised
  --save_results            Save results
  --sync_regularity         Computes synchronicity and regularity metrics
  --nerve_readout           Computes nerve activity from population activity
  --run_PCA                 Computes PCA on membrane and convolved spiking activity
  --PCA_components          Dimension of new basis for PCA
  --remove_mean             Removes mean from convolved spikes
  --high_pass_filtered      Apply high pass filer to convolved spikes
```

See readme.md in the repository for full instructions on how to run the code.

1.4.3 Machine-learning code overview

The code repository to train the inverse models contains the following files:

```
  Predictive network
   |__train_predictive.py                    # Main script to train the models
   |__evaluate_predictive.py                 # Script to evaluated trained models
   |__predictive_models.py                   # DL model classes
   |__predictive_pytorch.py                  # Code to train DL models
   |__utils.py                               # Helper functions
   |__population_models_inverse_wrappers.py  # Helper functions
   |__ESN/*                                  # ESN models (pre-alpha)
   |__experiments.py                         # Script with experiments
   |__experiments/                           # Results from experiments
   |__saved_models/*                         # Trained models checkpoints
   |__requirements.txt                       # Libraries dependencies
   |__README.md                              # Instructions on how to run the code
```

The script `train_predictive.py` is the main interface to train the models. It allows to select the type and size of the neural population that we want to simulate (ie. rate or spiking) as well as the model parameters that we want the DL-model to inverse (eg. stimulation current).

Below we present a summary of the main functionalities of the training script. These options can be display with `python train_predictive.py --help`:

```
train_predictive.py options:

  --population_type         Population type: rate or spiking
  --number_neurons          Number of biological neurons simulated.
                            For spiking version it needs to be a multiple of 100
  --time_recording          Time the simulation runs for in ms
  --resampling_rate         Resampling rate [1.0, 0[. If 1.0 no resampling is done
                            and default 1ms resolution is used. If 0.1 resution is 1/10
                            of the original.
  --remove_transient        Percentage of the initial timepoints of signal to be removed:
                            eg. 0.0 does not do anything, 0.2 removes the first 20% datapoints, etc.
  --num_epochs              Number of epochs the DL network is trained for
  --batch_size              Number of simulations per batch
  --lr                      Optimizer initial learning rate
  --ANN_architecture        ANN architecture: CNN1D, LSTM, CNN1D_small, CNN1D_medium
  --loss_function           Loss function: L1, SmoothL1, L2
  --inversing_parameter     Model parameters that we want to recover. For RATE model:
                            gains or stimulation_current. For
                            SPIKING: stimulation_current_through_spikes
                            or stimulation_current_membrane_potentials.
  --dim_reduced             If true, the output space is transformed onto a lower
                            dimensional representation (default: False)
  --DR_technique            If DR true, this argument specifies the technique to use:
                            PCA, tSNE
  --DR_dimensions           If DR true, number of dimensions of the new basis
                            Standardizes features by removing the mean and scaling to
                            unit variance (default: False)
  --single_manifold         If true, the used stimulation is the same random vector
                            for all simulations (default: False)
  --random_connectivity     If true, the connectivity of the network will be random,
                            if false: a single coupling value (positive for exct. and
                            negative for inhib.) will be used. (default: True)
  --random_connectivity_sparse
                            If true, the connectivity of the network that is sampled will
                            be sparse. (default: False)
  --random_connectivity_each_instance
                            If true, the connectivity of the network is randomised at each
                            simulation instance (along with the inversing parameter)
                            (default: False)
  --random_connectivity_from_normal_distribution
                            If true, the connectivity of the network is sampled from a normal
                            random distribution. Only applies to the rate model,
                            spiking is always sampled from a normal distribution. (default: True)
  --save_model              Save resulting model (default: True)
  --checkpoint_id           Checkpoint id, allows to resume training. (default: False)
  --new_lr                  If resuming training, uses a new learning rate. (default: True)
  --JIT_freq                If !=0, it will generate a new simulation at each epochs
                            rather than using a pre-generated dataset. Makes optimisation
                            slower but requires less memory and prevents overfitting on
                            training set. Its value indicates how many epochs
                            new data is simulated, if 0 a single dataset is produced.
```

The deep-learning model classes are found in `predictive_models.py`, where we have implemented an LSTM model and three convolutional networks of different sizes (CNN1D_small, CNN1D_medium and CNN1D) by correspondingly increasing the number of linear layers after the convolutions. The functions to train the models are implemented in Pytorch and can be found in `predictive_pytorch.py`. For the detailed functionality of all the classes and function, please refer to the code, where all the classes functions have been comprehensively commented using docstrings. See readme.md in the repository for full instructions on how to run the code.

1.4.3.1 Trained models checkpoints

In the repository we have included the checkpoints of the trained models. These are the models for which we report the results in Chapter 5. We have included the following trained models, whose results can be obtained by simply running:
`python evaluate_predictive.py --id <model id>`:

```
saved_models/ # Trained models checkpoints
├── 1636572579     # Rate (CNN)
├── 1636646979     # Rate (LSTM)
├── 1636645220     # Rate connectivity-agnostic
├── 1636579660     # Spiking
├── 1636296822     # Spiking connectivity-agnostic
└── 1634823405     # Manifold induction
```

Each model folder also includes a human-readable configuration file (`.yml`) file and the training curves `losses.pdf`.

Chapter 2

Background

In this chapter we introduce a few basic concepts from computational neuroscience: how neurons communicate through synapses, the different abstractions available to build neuron and population models, and the known types of neural coding.

2.1 A Brief History of Neuroscience

The Nobel Prize in Physiology or Medicine 1906 was awarded jointly to Camillo Golgi and Santiago Ramón y Cajal ”in recognition of their work on the structure of the nervous system”. Golgi had invented a method to stain nervous tissue using potassium bichromate and silver nitrate and Cajal made a series of observations that resulted in the *Neuron doctrine*: the nervous system is composed of discrete individual cells with gaps between them which are used as communication channels [4], [5]. Since then, the history of neuroscience has been the history of the technical innovations that have enabled us to simultaneously capture the activity of increasing number of neurons at high temporal and spatial resolutions. Over the last 20 years, this technical progress has percolated down to the theoretical approaches used to to understand neurons, metamorphosing the neuron doctrine into a *neural network doctrine* making neural populations, rather than single neurons, the meaningful unit of computation and, therefore, the center of study of neuroscience [6], [7].

2.2 Dynamical models in Neuroscience

In this section we will introduce the basic notions needed to investigate neuronal systems as well as the main model families currently used to study them.

2.2.1 Elements of Neural Dynamics

The activity of nervous systems span across many spatial and temporal orders of magnitudes. From the molecular dynamics governing the cell's ion channels up to the dynamics of neuronal populations made of billions of neurons. The choice of the granularity at which we want to explain the phenomena will determine the suitable type of model, see Fig. 2.1.

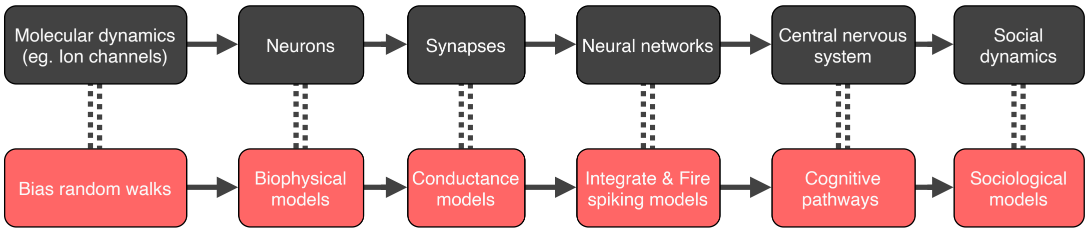


Figure 2.1: Scales at which the activity of neurons can be studied and examples of their corresponding model families.

2.2.2 Neurons, point-neurons and synapses

2.2.2.1 Neurons

It will not come as a surprise: the basic unit of study in neuroscience is the neuron. The neuron acts as the epistemological center of gravity around which all modelling scales orbit. Neurons are specialised cells that possess an electrically excitable membrane and whose main purpose is integrating and transmitting information. They can present very different morphologies, each endowing them of distinct computational capabilities. In Fig. 2.2 we can see two well-known neuron types, a pyramidal neuron from the cerebral cortex and a Purkinje cell, typically located in the cerebellum.

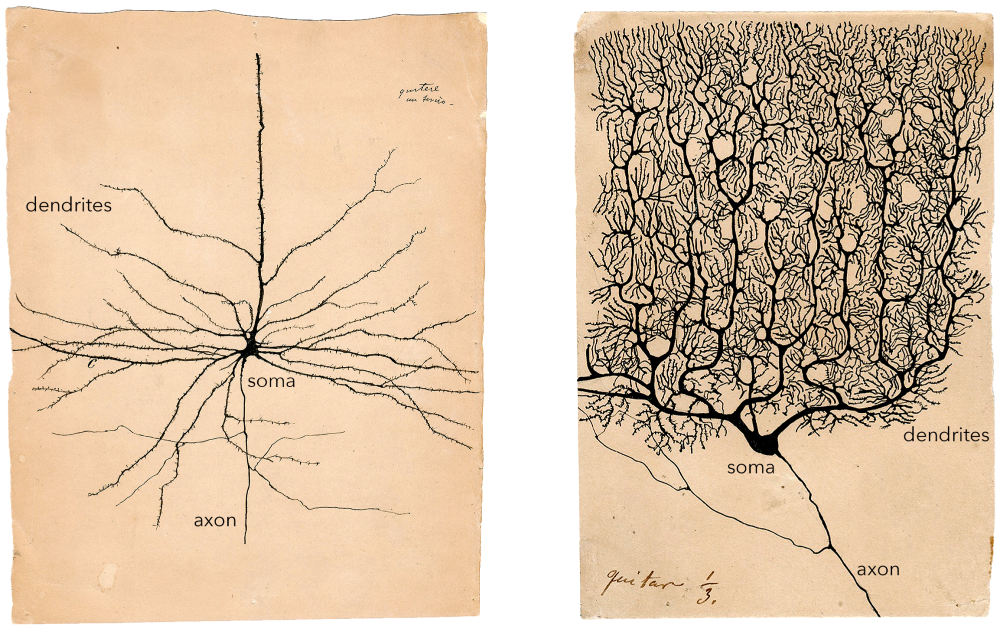


Figure 2.2: Example of two types of neuron morphologies: a pyramidal neuron (left) and a Purkinje neuron (right) drawn by Cajal in 1904 from *de visu* microscopy observations. The neurites projections extending from the soma are referred as *dendrites*, they are responsible for integrating incoming information. The Purkinje neuron is characterized by the intricate morphology of its dendritic arbor. The *axon* is the somatic projection whose basic purpose is to transmit the integrated information onto other cells.

Dendrites and signal integration The word dendrite derives from greek *δευδρου* /*déndron*/, meaning *tree*. Dendrites are the branch-shaped membrane extensions (see Fig. 2.2) through which neurons receive and *integrate* information from incoming neurons before forwarding the signal towards the neuron's soma. Therefore they constitute an defining element of the computational properties of a neuron.

2.2.2.2 Synaptic communication

In their homeostatic state, neuron membranes observe a potential of $\sim -70mV$ between the interior and the exterior of the cell; in this state the cell is said to be polarised. Neurons' electric excitability arises from the presence of ion channels and ion pumps in their membranes. Ion channels are proteins that regulate the flux of ions between the inside and outside of the cell. By doing so, they enable neurons to communicate: when positively charged ions enter the cell they reduce the membrane potential, a process referred as depolarisation. If the depolarisation reaches a threshold, it will trigger a positive feedback by which more ion channels are opened and the neuron generates an electric binary signal referred to as *spike* or *action potential*. Spikes are pulse-like currents that constitute the fundamental unit of communication between neurons [8], [9]. See Fig. 2.4 for a schematic of the spiking process. When a spike reaches the postsynaptic axon terminal, it dissolves the vesicles containing neurotransmitters, releasing them into the postsynaptic cleft, see Fig. 2.3.

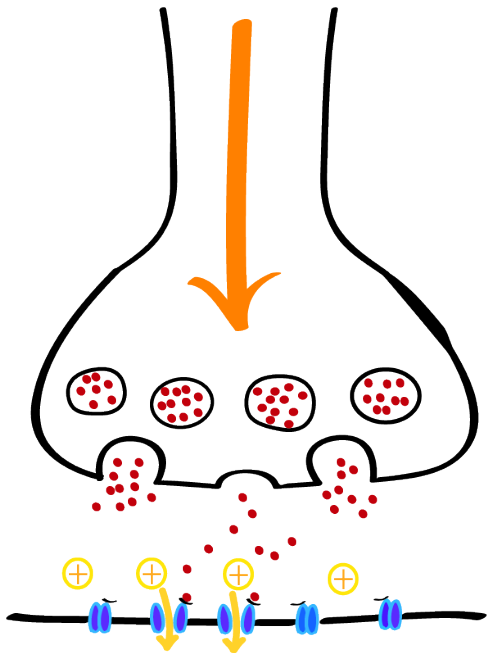


Figure 2.3: Schematic of a chemical synapse. A spike (orange arrow) arrives at the synapse, causing a release of neurotransmitters (red dots) onto the synaptic cleft. When the neurotransmitters bind to the adjacent neuron ion channels' receptors (blue), they open up the ion channels letting in an influx of extra-cellular ions (yellow). The main ion-channels present in neurons are sodium Na^{+}, potassium K^{+}, calcium Ca^{2+} and chloride Cl^{-}. Image adapted from [10].

The region where two neurons meet is called the synapse and the signal is said to be transmitted from the pre-synaptic to the postsynaptic neuron. Depending on the mechanism triggering the opening of the ion channels we will distinguish two main types of synapses: chemical and electrical. In chemical synapses, the pre-synaptic neuron releases a specific type of molecule -a *neurotransmitter*- into the synaptic cleft, this neurotransmitter will bind to the postsynaptic neuron ion-channels, changing their structure into a conformation that allows for the influx of ions; chemical are the major kind of synapses in the nervous system. In the case of electric synapses, the neurotransmitter layer is skipped and ions travel directly from the pre to the postsynaptic neuron, allowing for faster communication speeds but at the expense of loosing versatility.

If the influx of ions into the postsynaptic neuron is positively charged, we talk about excitatory postsynaptic potential (EPSP), conversely, an influx of negatively charged ions that will further polarise the cell is referred to as an inhibitory postsynaptic potential (IPSP). Whether a presynaptic neuron generates EPSPs or IPSPs depends on its neurotransmitter profile which will determine which ion channels open up and, hence, if we have an inhibitory -i.e. polarizing- current (negatively-charged ions) or a excitatory -i.e. depolarizing- current (positively-charged ions). The neurotransmitter profile of a neuron -ie. if it releases inhibitory or excitatory neurotransmitters when activated- determine its inhibitory/excitatory nature and, along with its morphology, determines the computational properties of the neuron. In our computational models, we abstract the notion of inhibitory/excitatory synapses by negative/positive weights in the connectivity matrix of the neural population. For the spiking models these connectivity weights have Volt units [V], reflecting how much the membrane potential of the post-synaptic neuron is decreased/increased when the pre-synaptic neuron is activated; for rate-based models, the connectivity has hertz units [Hz], indicating how much the firing rate of the post-synaptic neuron decreases/increases when the pre-synaptic neuron is activated.

Both chemical and electrical synapses are modelled in the same fashion: as a set of coupled non-linear differential equations capturing the activity of the ion channels.

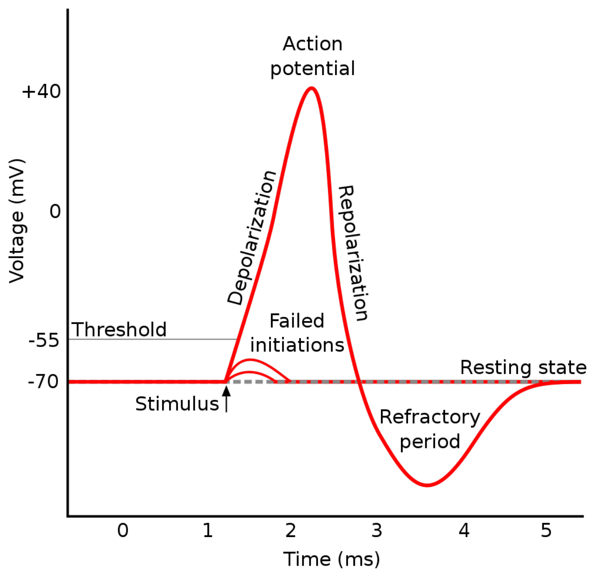


Figure 2.4: Schematic of an action potential. The curve indicates the change in the voltage membrane potential. Stimulus that fail to reach the firing threshold will not elicit a spike. After spiking there is a refractory period where the membrane is hyperpolarised and an action potential during this phase is highly unlikely. Image source [11].

2.2.2.3 Point-neurons

The dynamics of the activations between two neurons require taking into account the distribution of ion-channels across the different sections of the membrane surface as well as the morphology of the membrane itself. This results in non-linear differential models which are highly faithful but significantly complex. An useful abstraction is to consider neurons to be point-like, that is, disregarding the spatial properties of the cell membrane similarly to the point particle approximation in physics. In point neurons, dendrites are abstracted into the connectivity the neuron. This abstraction is often used in computational neuroscience but has its limitations, for instance, neurons have been shown to perform crucial computations on their dendrites [12]–[14]. Throughout this work, we will restrict our models to point-neurons.

2.2.3 Neural dynamics models

2.2.3.1 Hodgkin and Huxley model

The first electrical model that successfully produced a quantitative description of synaptic commutation is the Hodgkin and Huxley model (H&H) . H&H (Eq.2.1) is a current conservation model that describes the cell's lipid layer as a capacitance and the ion-channels activity as a set of nonlinear conductances. Gradients driving ion fluxes are represented by batteries and ion pumps as currents. The membrane electric activity is therefore modelled as a set of four coupled ordinary differential equations: one describing the membrane potential in relation to the sodium, potassium, and leak currents, and three equations characterising the gating the ion-channels. This model, originally published in 1952, was used to describe the activity dynamics of squid giant axon and, provided that the equations constant parameters are adjusted, is capable of precisely predict the spike generation across a number of neuronal types. But more importantly, it paved the way for the accession of theoretical and computational neuroscience.

$$C_m \frac{dV(t)}{dt} + I_{Na}(t) + I_K(t) + I_{leak}(t) = I_{ext}(t) \tag{2.1}$$

Since then, a myriad of mathematical models following the H&H paradigm have been developed. We will mention the most relevant for our work here:

2.2.3.2 Integrate-and-fire

Integrate-and-fire (IF) models capture the integration of postsynaptic potentials (PSPs) as a simple summation and combine them. If the cumulative sum of PSPs crosses a threshold then the neuron fires a spike. This model can be described with a differential equation used to characterise an electrical RC circuit where the cell membrane acts as a capacitor insulating the conductive intra-cellular and extra-cellular fluids 2.2; V is the membrane potential, $I(t)$ is the sum of ionic input currents from presynaptic neurons and C_m is the membrane capacitance. It is the ability of the cell membrane to accumulate charges over time that enables the neuron to integrate inputs over time and therefore it is able to perform computations.

$$I(t) = C_m \frac{dV(t)}{dt} \tag{2.2}$$

As previously discussed, in order to fire, the membrane voltage of a neuron needs to attain a threshold which, once reached, will inescapably trigger a spike. These activation dynamics, depicted in Fig. 2.4, can be fully described with the Hodgkin and Huxley ionic currents model. However, IF models do not intrinsically give rise to spikes, instead, a firing-threshold mechanism is added *ad hoc* to generate spikes once the defined threshold is reached and subsequently reset the membrane voltage to its resting value. This approach has the advantage of greatly reducing the analytical and numerical complexity of differential equation models, which is especially convenient when simulating networks with hundreds of thousands of neurons.

Leaky integrate-and-fire

Leaky integrate-and-fire (LIF) is a variation of Integrate-and-fire which adds a term describing the diffusion -or leak- of ions across the cell's membrane that is experimentally observed in neurons. We start with Kirchhoff's conservation of current law for the current arriving to the cell membrane:

$$I(t) = I_{C_m} + I_{leak} \tag{2.3}$$

The cell membrane is not a perfect insulator, therefore besides the current charging the capacitor I_{C_m}, we have another *leak current* I_{leak} representing the movement of charges -in the form of ions- through the cell membrane. This translates into adding the extra leaky term $I_{leak} = \frac{V(t)}{R_m}$ to Eq. 2.2 which results in the following equation:

$$I(t) = C_m \frac{dV(t)}{dt} + \frac{V(t)}{R_m} \tag{2.4}$$

In the case $R_m \to \infty$, we retrieve the non-leaky IF model. The spiking dynamics of the LIF model as a response to a constant and a stochastic input are shown below in Fig. 2.5. Finally, to account for the resting potential observed on neurons in the absence of any input current, we add a resting potential term V_{rest} and replace the capacitance term by τ_m/R_m:

$$I(t) = \frac{\tau_m}{R_m} \frac{dV(t)}{dt} + \frac{V(t) - V_{rest}}{R_m} \tag{2.5}$$

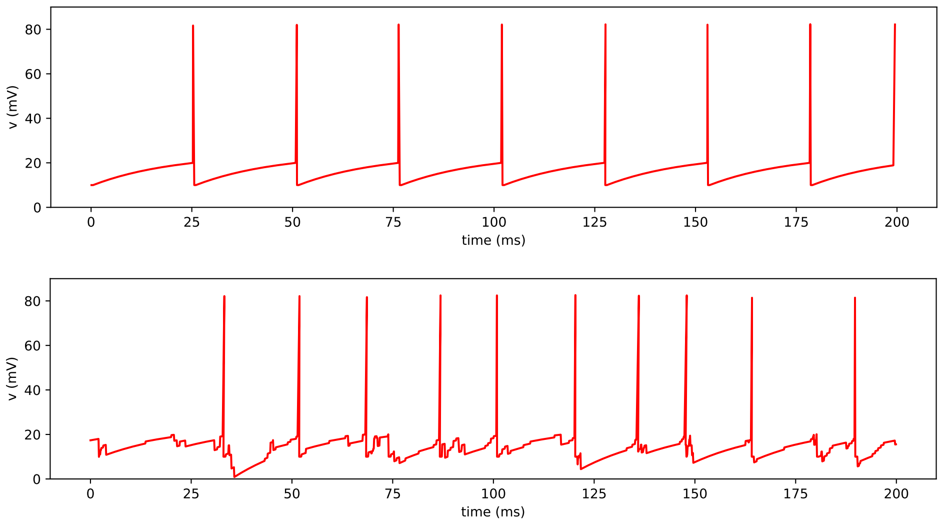


Figure 2.5: Membrane voltage dynamics of a LIF neuron in response to (above) a constant input current and (below) a stochastic spiking input. The non-spike as oscillations of the membrane observed on the lower figure are typically referred as *subthreshold dynamics*.

Generalized leaky integrate-and-fire models If our working model proves to be incapable of reproducing the spiking behaviours of some real neurons, we can proceed by introducing additional non-linearities or variables as we did with the leaky term in the LIF model. Additionally we include stochastic terms to account for the intrinsic noise present at the cellular level. Generalized leaky integrate-and-fire models (GLIF) are the family of models that extend from the aforementioned LIF model up to more complex models including terms either conjectured or fitted to electrophysiological data [15].

Other point-neuron models The spanned spectrum between Hodgkin and Huxley and Integrate and Fire models is inhabited by a population of models, each favouring different trade-offs between the accuracy of H&H and the tractability of LIF. Additionally, the computational cost of simulating the model and its parallelizability have become critical factors with the growth of simulation-based experiments [16]. Finally, other models like the *Stochastic Point Process model* directly abandon the notion of representing the cell's membrane potential and instead describe the neuron as an inhomogeneous Poisson process whose instantaneous firing rate depends on its inputs.

2.2.4 Network dynamics models

Neurons are found in living organisms forming networks. The size of these networks spans anywhere from the 302 neurons and ∼7000 synapses of the C. Elegans nematode [17] to the ∼86 billions neurons and $\sim 10^{14}$ synapses of the human cerebrum [18]. Neural network models attempt to characterize the neurons activity driven by both external stimulus as well as internally generated activity [19]. We will choose an appropriate neuron model based on which aspects of the network behaviour we want to capture with our model. Two main approaches to modding neuronal populations are spiking and firing-rate models.

2.2.4.1 Spiking networks

Spikes are the fundamental unit of communication between neurons [9], [20]. As such, spiking neural networks (SNNs) are the canonical model for simulating the behaviour of neural populations.

The dynamics of a SNN is characterised by two time scales, the fast oscillations of the sub-threshold potentials and the slower scale of spikes. The specific activation dynamics are determined by the choice of the neuron model (eg. H&H, IF, LIF, etc.). Regardless of the neuron model, spikes are binary events -i.e. the amplitude of the spike is independent of the magnitude of the input current the cell receives-, and are mathematically represented as Dirac delta function $\delta(t)$.

The activity of the network is fully characterised by a set of spike times and the location and types of synapses where they occurred, i.e., which two neurons the synapse connects and the strength of the excitatory or inhibitory synapse. When simulating neural populations, the excitatory/inhibitory nature of the synapse is abstracted out from biophysical ion-channel profiles and instead represented as a positive/negative synaptic coupling.

The standard way of visualising a SNN activity is through a *raster plot*, see top image in Fig. 2.6. The sequence of times at which a given neuron has spiked is called the neuron's *spike train*. From the spiking activity of a network, it is straightforward to obtain the population activity by computing the population mean firing rate over time (middle image in Fig. 2.6).

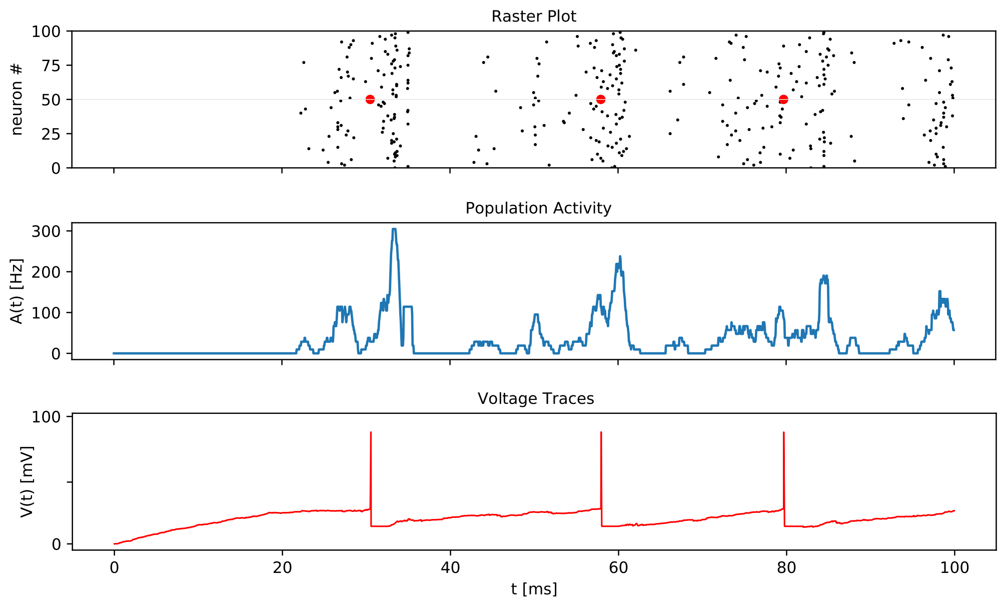


Figure 2.6: Visualisation of the activity of a simulated Brunel network of 100 neurons during a 100ms. Top: a raster plot showing the spiking times of each of the neurons. Middle: the population rate average over time. Bottom: the membrane potentials of one neuron over time, the spiking produced by the membrane potential reaching the firing threshold is visualised on the top raster plot as red dots.

2.2.4.2 Firing-Rate networks

Population rate models are an abstraction of spiking models that use mean firing rates, rather than spike-timings, as the central variable of the model [21], [22]. The firing rate can be defined in three ways: 1. temporal average: the number of spikes of one neuron occurring during a long time window; 2. spatial average: the number of spikes in N neurons over a short time window; 3. firing probability: the number of spikes of one neuron across N trails during a short time window.

Firing-rate models can be seen as a course-grained version of spiking models where each of the differential equations characterizes not a single cell but a small neuronal population. By adopting this abstraction, we largely reduce the number of model parameters, we disregard sub-threshold dynamics and lose the spiking threshold non-linearity [16]. Therefore, rate models can be convenient as long as the information of interest does not live on the fast sub-threshold oscillations or in the timing of the spikes.

The differential equations governing firing-rate models are similar those previously introduced for spiking networks. However, than characterising the voltage potential of the membrane, rate models describe the evolution of the firing rate f. The neuron firing rate is taken as an approximation to its response function and is modelled as a function of its weighted synaptic input spike trains $\boldsymbol{w} \cdot \boldsymbol{f}_{pre}$, where $\boldsymbol{f}_{pre}$ is the vector of pre-synaptic activity of all the neurons connected to a given neuron with weights $\boldsymbol{w}$. A defining parameter of a rate model is its *synaptic kernel* which is the function over which the presynaptic spike train is convolved to obtain the induced current response I_S of the postsynaptic neuron. Assuming a typical exponential synaptic kernel and a linear dendritic integration, we can derive the rate model equation:

$$\tau_S \frac{dI_S(t)}{dt} = \boldsymbol{w} \cdot \boldsymbol{f}_{pre} \tag{2.6}$$

where τ_S is he time constant describing the decay of the synaptic conductance, $\boldsymbol{w}$ represent the synaptic weights and $\boldsymbol{f}_{pre}$ are the firing rates of the presynaptic neurons. To complete the rate model, it remains to replace the synaptic current I_S by the postsynaptic firing rates. This is achieved by using a non-linear activation function $\mathcal{F}(I_S)$ which maps the current onto firing rates. A typical activation function like sigmoid also bounds the firing rates, resulting in a more stable network activity. The resulting equation is:

$$\tau_r \frac{df(t)}{dt} = \mathcal{F}(I_S) \tag{2.7}$$

where τ_r how rapidly the firing rate approaches its steady-state value for constant synaptic input I_S (not to be confused to the time constant of the membrane potential). The system of equations 2.6 and 2.7 is the firing-rate model approximation of a spiking network [23]. If we assume that Eq. 2.6 relaxes to its steady state much faster than Eq. 2.7, i.e. the input spike trains are integrated over longer time windows than the time scale over which the output spike trains are generated ($\tau_r >> \tau_S$), then we can simplify the system onto a single equation:

$$\tau_r \frac{df(t)}{dt} = \boldsymbol{w} \cdot \mathcal{F}(\boldsymbol{f}_{pre}) \tag{2.8}$$

Rate models have seen some success in describing aspects of memory, visual, and motor processing [22]. Additionally, they are analogous to the popular artificial neural networks models used in machine learning.

2.2.4.3 Balanced networks

A neural network is said to be E/I balanced if the magnitude of its excitatory currents matches that of its inhibitory currents on short time scales. In this regime, input stimuli and transient fluctuations are able to drive the global activity of the network even when their magnitudes are small. This balance between inhibition and excitation has been found experimentally in cortical circuits and is conjectured to be a critical element of how neurons perform computations and crucial for the neural coding [24]. E/I balance allows to explain how a network made of millions of neurons can produce rich internal dynamics while simultaneously remaining sensitive to brief sensory inputs [19].

A simple approach to generate balanced networks is the one used by the Brunel network [25]. It consists of using two populations of LIF neurons, one excitatory and one inhibitory (i.e., positive and negative synaptic couplings respectively). Both populations are connected to each other as well to a third external excitatory population, see Fig. 2.7. To obtain E/I balance, the Brunel network has a ratio of four excitatory neurons for each inhibitory one while making the synaptic strengths of inhibitory synapses to be four times stronger than that of the excitatory ones; these numbers are loosely inspired by neurophysiological observations of cortical populations. The Brunel network is an archetypal model of spiking network whose properties and dynamical regimes have been thoroughly studied both analytically [25] and numerically [26].

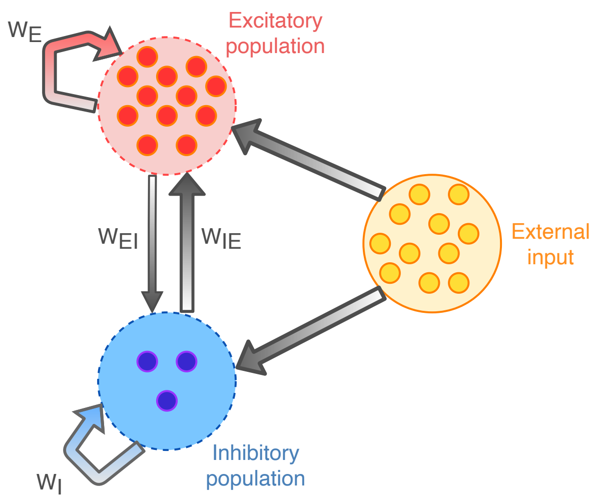


Figure 2.7: Diagram of a Brunel network. The excitatory population (red) is coupled to itself with synaptic strength W_E and to the inhibitory population (blue) with strength W_{EI}, an equivalent connectivity applies to the inhibitory population resulting in a balanced network. A third population (yellow) acts as an external source of stimulation to both populations.

2.2.5 Neural coding

The question of how neurons encode information is a long-standing quest at the heart of neuroscience. Like with most hard problems in neuroscience, the epistemic bottleneck is our inability to track and perturb the activity of every individual neurons in a population. This technical difficulty has resulted in most encoding models being correlational rather than causal. Moreover, the question is additionally challenging due to the observation that different neurons appear to employ different coding schemes. Here we present an overview of our current knowledge on how neuronal populations encode information.

2.2.5.1 Rate coding

Firing frequency is the simplest coding mechanism for which there is experimental evidence. Sensory inputs such as temperature receptors on the skin are translated onto a firing scheme, for instance the firing rate of the neurons in the skin is linear with the temperature perceived by the skin thermoreceptors. This way, a sensory quality like temperature is translated onto neural activity.

Another example is that of pitch encoding by the neurons at the base of hair cells in the inner ear. Sound waves make hair cells release neurotransmitters which make the neurons place at their base to fire at a frequency that, in the range 20Hz to 1KHz, directly codes the frequency, that is, a 100Hz sound produces a 100Hz spiking rate [27]. Interestingly, because most spikes extend temporally 1ms, this direct frequency coding allows only to encode up 1KHz and evolution has endowed the inner ear with another coding mechanism for pitch above this bound.

2.2.5.2 Topological coding

Building on the inner ear example previously discussed: how can it encode sound frequencies above 1KHz? Now, instead of using a direct encoding, the morphology of the inner ear makes it so that different regions of the basilar membrane -where the hair cells are embed- resonate more or less to sound waves depending on the frequency of the sound. This enables a *topological encoding* of the frequency based on the location of the firing neurons. In this regime, the spiking frequency now encodes for the intensity of the sound.

2.2.5.3 Temporal encodings

In temporal codes, the relative timings of the spikes serve as an encoding mechanism of the relevant information.

Phase coding Adaptation to sensory input is a commonly experienced phenomenon, if one puts on a ring on a finger, the pressure is perceived while putting it on, however after some minutes the sensation disappears. Neurons connected to pressure skin receptors are only excited by *changes* in the pressure not by the absolute magnitude of the stimulus, this is known a as phasic receptor response. In this situation, the information is encoded in the derivative of the firing frequency with respect to time rather than the absolute rate. Phase encoding has the advantage of being extremely fast, only one spike is needed to signify change with a single spike ($\sim$1ms), however it is less suitable for continuous quantities for which the firing rate is a more suitable code.

Latency coding In the *Latency* or *Time-to-First-Spike* scheme, a signal magnitude is inversely mapped onto the time that it requires to make the neuron spike after stimulus is present, i.e., a strong stimulus can be encoded as a neuron firing shortly after being exposed to it, while a weak one would require a longer time to elicit a spike.

Temporal patterns Information can also be encoded in a specific temporal pattern of activity, where the set of relative firing times between every spike pair carries the information.

2.2.5.4 Population coding

Encodings can be categorised within the *sparseness vs distributed representation* spectrum. On the sparse extreme of it, the now defunct idea of the *grandmother cell* was a popular coding model that suggested that individual neurons *encode* for high-level entities like the image of one's grandmother, the voice of a loved one or a specific smell [28]. Over time, as electrophysiologists could progressively record from an increasing number of neurons, the code modelling paradigm gradually shifted from single neuron toward population encodings where the function of a neural circuit emerges from the ensemble activity [6], [7], [29], [30]. [1]

Nowadays, neural ensembles are considered to be the fundamental unit of computation capable of explaining neural circuitry functionality and organism behaviour [6], [7].

Neurons connected to skin thermoreceptor are *tuned* to different temperature ranges, that enables to have a population of neurons capable of encoding the whole range of temperature the receptor can be exposed using a finite section the frequency spectrum. Furthermore, individual neurons are noisy and

[1] The now-obvious bias toward single-neuron models of information encoding that was hegemonic during the dawn of neuroscience when only single-cell recordings were possible can be seen as instanciation of the *law of the instrument* introduced by Abraham Kaplan's in 1964 and popularised by Abraham Maslow in the its hammer form: *"if all you have is a hammer, everything looks like a nail"* [31], [32].

can therefore be unreliable. To make them more robust, redundancy can be introduced by having several neurons to transmit the same information and then averaging over them.

Manifolds and population encoding In the language of coding theory -without considering biological priors- we can describe coding schemes in a spectrum spanning from efficient to robust. Efficient codes capture the information in orthogonal dimensions, this property makes them efficient but highly sensitive to noise and perturbations. On the other end, robust codes will have many correlated dimensions which result in a redundant encoding of information. Redundancy makes them resilient to noise and perturbations but less efficient. In neural circuits, the number of dimensions is the number of neurons present in the population, a neuron is said to be redundant if its activity is correlated to that of another neuron of the population. Correlated activity between neurons is typically observed experimentally and constitutes the mechanism enabling a population of neurons to perform computations despite noise or malfunctions of a subset of neurons [33]–[36]. Furthermore, the emergence of certain dynamical patterns -like attractor models of associative memory or decision-making- can only be understood from the perspective of an ensemble computation [6]. In sections 4.1.1 and 4.4 we will detail how to make use of the notion of manifolds to quantitatively study neural activity.

Recurrent and Hierarchical population encodings Certain motor or cognitive tasks require of a sequence of neural activities to be performed. For instance, the motor control of a muscle requires the synchronous activation of different fiber bundles, therefore requiring a mechanism to orchestrate the oscillations timely. To do so, neural populations can form recurrent networks enabling them to produce complex sequential patterns of intrinsic self-sustained activity. In the cognitive realm, eg. in the visual cortex, different sub-populations may extract different features of the visual stimulus which is then feedforwarded through the hierarchy of populations (known as the ventral stream) to form a high-level perception of the visual input.

2.2.5.5 Representational drift

Let us notice that there are no reasons to believe that neither neural representations nor their underlying network circuits ought to be static over time. The notion of *representational drift* refers to the observation that representations change over time [37], ie. that the activity of individual neurons of a population do not elicit the same activity when subjected to the input a two different times. This apparent paradox lives off the epistemic aftershocks of the *single-neuron doctrine.* As we have discussed, the computational function of a neural network derives from its connectivity, therefore a dynamic connectivity that preserves the whole network functionality would produce the apparent variability at a single-cell level. Therefore, within our current understanding, the only constrain is that population representations must remain invariant when perceived by independent downstream regions.

2.2.5.6 From quantity to quality

All spikes are born free and indistinguishable from each other. Spikes being completely analogous, no matter which sensory input triggered them, the different sensory quality they elicit in perception (temperature vs sound) derives from the brain region to which they project to; somatosensory cortex in the case of touch inputs and auditory cortex sound stimuli.

Chapter 3

Spinal motor control and simulations of spinal populations

In this chapter we briefly introduce the basics notions about how movement is generated in mammals. We then analyse the different dynamical regimes of rate-based models and show how they can give rise to rotational dynamics and alternating muscle activity. Finally, we present our spiking population model, introduce two metrics to the properties of its dynamical regimes, and demonstrate that it also can generate rotational dynamics and alternating muscle activity.

3.1 Motor control

Movement, or the ability of changing the position of some or all -i.e. *locomotion*- of their parts, is one of the defining features of living organisms. Furthermore, it has been argued that movement is at the root of the evolutionary origin of cognition [38], [39]. The executive regulation of movement by the nervous system is referred as *motor control*.

3.1.1 Motor control in vertebrates

In all vertebrates, motor control follows a pathway that involves the following components:

3.1.1.1 Motor cortex

The motor cortex region is one of the earliest cortical structures to have been functionally investigated [40]. It is involved in planning and executing of voluntary movements.

3.1.1.2 Spinal cord

The spinal cord is the region of the central nervous system that extends from the medulla down to the bottom of the vertebral column. While historically seen as a simple relay for signals originating from and heading to the brain, it has been recognized in recent decades the importance of the computations occurring in the spinal cord circuitry. Notably, circuits involved in rhythmic movement known as Central Pattern Generators (CPG). The human spinal cord is estimated to contain 69 millions nerve cells [41], and the rat's spinal cord 7 millions [42].

3.1.1.3 Central pattern generators (CPG)

Central pattern generators (CPGs) are neural circuits present in the spinal cord which are capable of generating periodic output signals without requiring periodic inputs constantly driving them. Instead, CPGs are driven by modulatory signals which trigger or inhibit their activity. CPGs are then connected to muscle fibers through motor neurons. CPGs' periodic activity controls rhythmic motor behaviors such as walking, running, breathing, or chewing [43]–[46]. Evidence of the autonomy of CPG from motor cortex to generate these rhythmic behaviours originates from ablation studies showing that animals can produce complex motor movements despite having been decapitated [44], [47], [48]. Traditionally, CPGs have been modelled as a set of modular sub-networks whose interconnections can be adjusted to select an appropriate motor pattern [45], [49], [50]. Conversely, following the approach introduced by [1], we will investigate CPGs as dynamical systems whose behaviour emerge from its different dynamical regimes rather than from the coordinated interaction of its sub-modules. This approach, unlike the modular one, is consistent with both functionality and experimental recordings of neural activity [1].

3.1.1.4 Motor Neurons

Motor neurons (MNs) are any neuronal cells located in the motor cortex, the brainstem or the spinal cord. MNs located in the cortex project to the spinal cord, while those whose soma is in the spinal cord project to muscles. CPGs control muscles through the intermediation of motor neurons [51]. MNs stimulate muscle fibers making them contract, or in the absence of stimulation, relax. Each motor neuron only controls a subset of fibers that are controlled by the same motor unit, enabling animals to perform fine-grain but robust movements. Neurons that do not directly project to sensory input nor to muscles are referred as *interneurons*, eg. the neurons that make up CPG circuits would be considered motor interneurons.

3.1.1.5 Muscles

Muscles are made of set of longitudinal fibers which are associated in bundles –or *muscle fascicles*. Muscles are connected to bones through tendons, therefore when a muscle contracts it pulls the bone resulting in movement. Locomotion is the result of groups of muscles working in coordination. In its simplest form, we can express movements as the result of the coordinated combination of flexion and extension of skeletal joints [52], eg. walking involves the extension and flexion of the knee, hip, and ankle joints.

3.1.1.6 Motor Units

A motor neuron and the set of muscle fibers it innervates are known as a motor unit. Motor units are the basic unit by which the nervous system controls movement [53]. A standard muscle is made up of a few hundred of motor units, each motor neuron innervating several thousand muscle fibers [54].

We summarise the motor control pathway in Fig. 3.1. Starting from the executive decision initiated from the motor cortex up to the resulting movement.

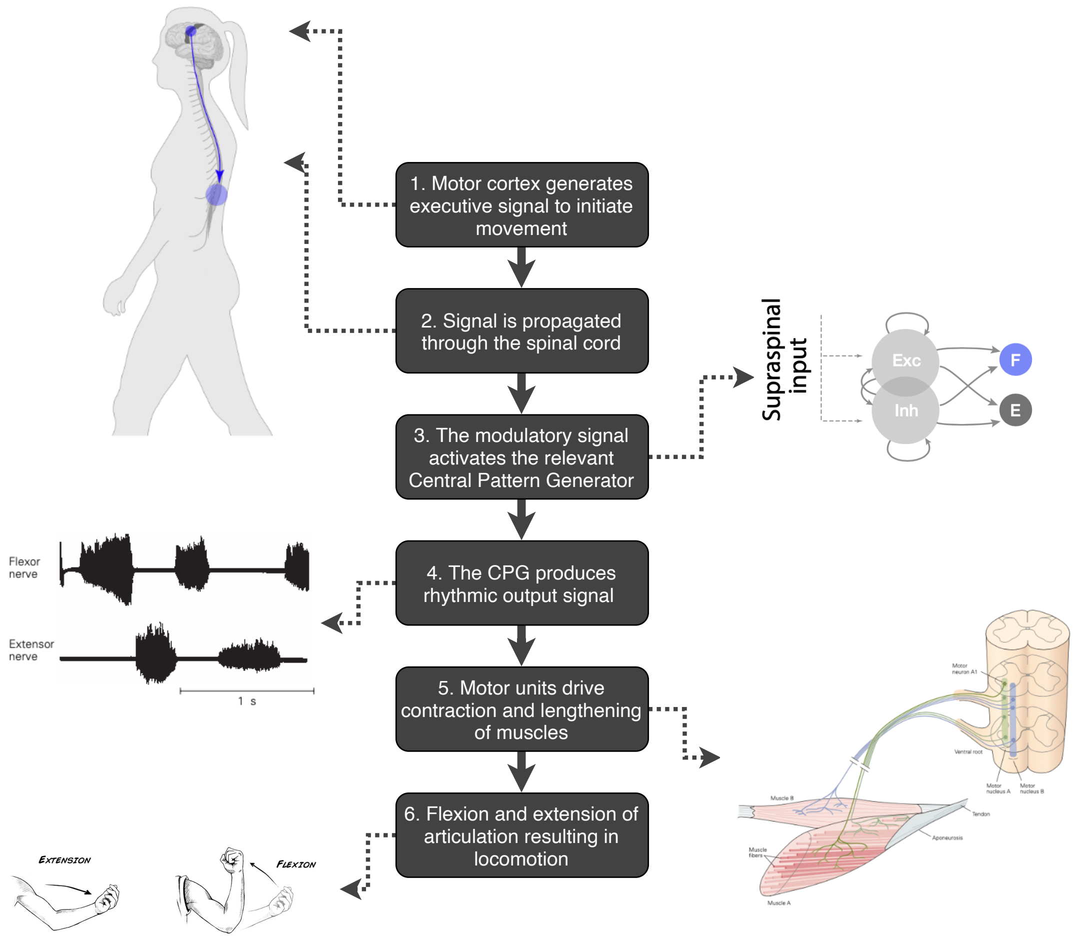

1. Motor cortex generates executive signal to initiate movement
2. Signal is propagated through the spinal cord
3. The modulatory signal activates the relevant Central Pattern Generator
4. The CPG produces rhythmic output signal
5. Motor units drive contraction and lengthening of muscles
6. Flexion and extension of articulation resulting in locomotion
Supraspinal input
Exc
Inh
F
E
Flexor nerve
Extensor nerve
1 s
Extension
Flexion
Motor neuron A1
Ventral root
Motor nucleus A
Motor nucleus B
Muscle B
Tendon
Aponeurosis
Muscle fibers
Muscle A

Figure 3.1: Diagram of executive motor control pathway in vertebrates. Following the evidence supporting the autonomy of spinal CPG circuits to drive movement 3.1.1.3, step 1 and 2 are skipped in non-executive movement.

3.2 Simulation of neural populations

In silico experiments are a crucial component in neuroscience, bridging the gap between theoretical models and behavioural recordings.

The computational power available in present-day computers and super-computers allows for the simulation of large neural networks, even when implemented with fine-grained biological mechanisms. Like with any model, the more detailed our model, the higher its computational cost. The choice of the appropriate level of model abstraction is critical and requires answering the underlying scientific question: what is the model built for? For instance, for certain behaviour, the computations occurring in the dendrites may play a crucial role and hence would require a compartmental model of the neuron that takes into account its morphology. For other tasks, a simple point-neuron model may suffice. A simulation with limited biological details will hardly be useful to explain experimental data, while an overly detailed network may contain too many unconstrained free parameters making it capable of reproducing any phenomena and hence producing little insight on the system. The model goal will also condition the solution method and the analysis methods employed.

In this project, we build and simulate models with two end goals. Firstly, to generate bio-plausible synthetic data to train our inverse models on. To do so, we constrain our model with known biological parameter ranges, such that the models activity is compatible with observed experimental data [1].

The number of neurons in a typical animal brain has the same order of magnitude as the number of stars in a typical galaxy. Therefore, if we strive to make simulations at biological-faithful scale, we are required to take the performance of the simulations into account. Several libraries exist to efficiently simulate neural populations at different scales. Our implementation of the rate networks built on those used by Lindén et al. [1]. For the spiking networks, we use Brian2, a spiking neural network simulator that offers a good balance between a friendly interface and high performance [55].

3.2.1 From random networks to behaviour

The activity pattern of a neural network -and hence the resulting organism behaviour- is thought to arise from the connectivity of the network, rather than from a fine-tuning of specific parameters of the neurons. This can be seen as consequence of the stochastic nature of neural tissue development where the global network architecture arises from neurons projecting neurites in a process exclusively guided by local interactions. We use these observations to guide our modelling approach: we simulate biological neural networks whose bio-physical parameters are informed by experimental measurements found in the literature and whose connectivity is sampled from a random distribution. We then study if ensemble averages of these networks can produce patterns of activity compatible with the animal behaviours we seek to reproduce. Another approach that we explore is to directly optimise the connectivity of a single network in order to produce the relevant activation patterns.

3.3 Population models of spinal motor control

Following the experimental evidence for central pattern generators in the spinal cord, a candidate neural network model needs to be able to generate a sustained activity in response to a transient input. The temporary stimulus representing the signal triggering the rhythmic motor behaviour, eg. a signal from the motor cortex. Furthermore, the generated sustained activity should be able to exhibit oscillatory patterns in order rhythmic motor behaviour.

3.3.1 Simulations of rate-based models of spinal motor control

Our simulations are based on a neural model that has previously been used to model the activity of motor cortex populations [56], [57]:

$$\tau \frac{d\boldsymbol{x}(t)}{dt} = -\boldsymbol{x}(t) + \boldsymbol{W} \cdot \mathcal{F}(\boldsymbol{x}, \boldsymbol{g}) + R\boldsymbol{I}(t) \tag{3.1}$$

with the activation function $\mathcal{F}$ being a sigmoid:

$$\mathcal{F}(x_i, g_i) = f_{max} \cdot \frac{1}{1 + e^{\,g_i(f_0 - x_i)}} \tag{3.2}$$

where x_i is the activity value of the neuron i whose image $\mathcal{F}(x_i, g_i)$ is its firing rate for a given input activation x_i and gain g_i 3.2. The synaptic couplings between the neurons are represented by the weight matrix $\boldsymbol{W}$ –positive for excitatory connection and negative for inhibitory ones–, $\boldsymbol{I}$(t) is the external input to the neurons, R represents the cell membrane resistance[1], τ is the time constant of the model, f_{max} is the maximum firing rate of the neurons and f_0 controls the threshold below which inputs do not elicit significant responses on the neurons. The specific values of the parameters are indicated in the table 3.1.

[1]In rate models, the input to each rate-neuron represents a firing rate, therefore, unlike in the conductance models that we will introduce in the spiking model section, here the resistance does not have Ohm units but instead can be freely chosen as long as the product of $R \cdot I(t)$ has hertz units. In our implementation, R is chosen to be adimensional and I is in Hz.

Parameter	Value
Network size	200
Excitatory-Inhibitory ratio	1
Activation gain	1 Hz^{-1}
Membrane time scale τ_m	50 ms
Minimum firing rate f_0	0 Hz
Maximum firing rate f_{max}	100 Hz
Activation function threshold f_0	50 Hz
Activation sigmoid firing rate f_{max}	100 Hz
Connectivity sparsity	10%
External input range	[0, 70] Hz

Table 3.1: Parameters used to define the rate population model (unless stated otherwise).

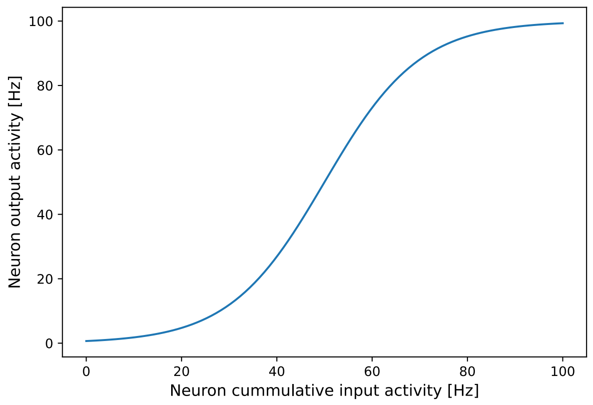


Figure 3.2: Sigmoid activation function used for the rate network simulations. Each neuron integrates the incoming activities it receives from its pre-synaptic neurons, it then applies a sigmoid to the resulting cumulative activity and broadcast the resulting signal to its post-synaptic neurons. In reality, different types of neurons have different activation functions; unless stated otherwise, our rate model uses this single activation function for all the neurons.

3.3.1.1 Random network models and eigenvalue analysis

Eigenvalue analysis provides a method to build networks with random connectivity but predictable stability. We construct the connectivity matrix **W** following the method introduced by [58] which allow us to build **W** with a specific spectral radius.

The stability of a network refers to whether its fixed points are resilient to the presence of small perturbations. In our model, the stability of the network depends on the presence of eigenvalues with a positive real part [59]. Therefore, by constructing a connectivity matrix with a specific spectral radius ρ we can produce stable networks which will have a fix point if $\rho < 1$ and will be unstable if $\rho > 1$. Unstable networks will observe self-sustained activity, that is, a network for which a short-duration stimulation can trigger an activity that remains beyond the presence of the stimulus, see Fig. 3.3. From a neuroscientific point of view, network models capable of self-sustained activity ($\rho \gtrsim 1$) have a great interest since this is a property observed in biological neural networks, for instance, upon deciding to start walking the motor cortex send a signal to the spinal cord which produce a self-sustained activity in the spinal cord that drive the movement of the muscles without the motor cortex having to keep sending any signal.

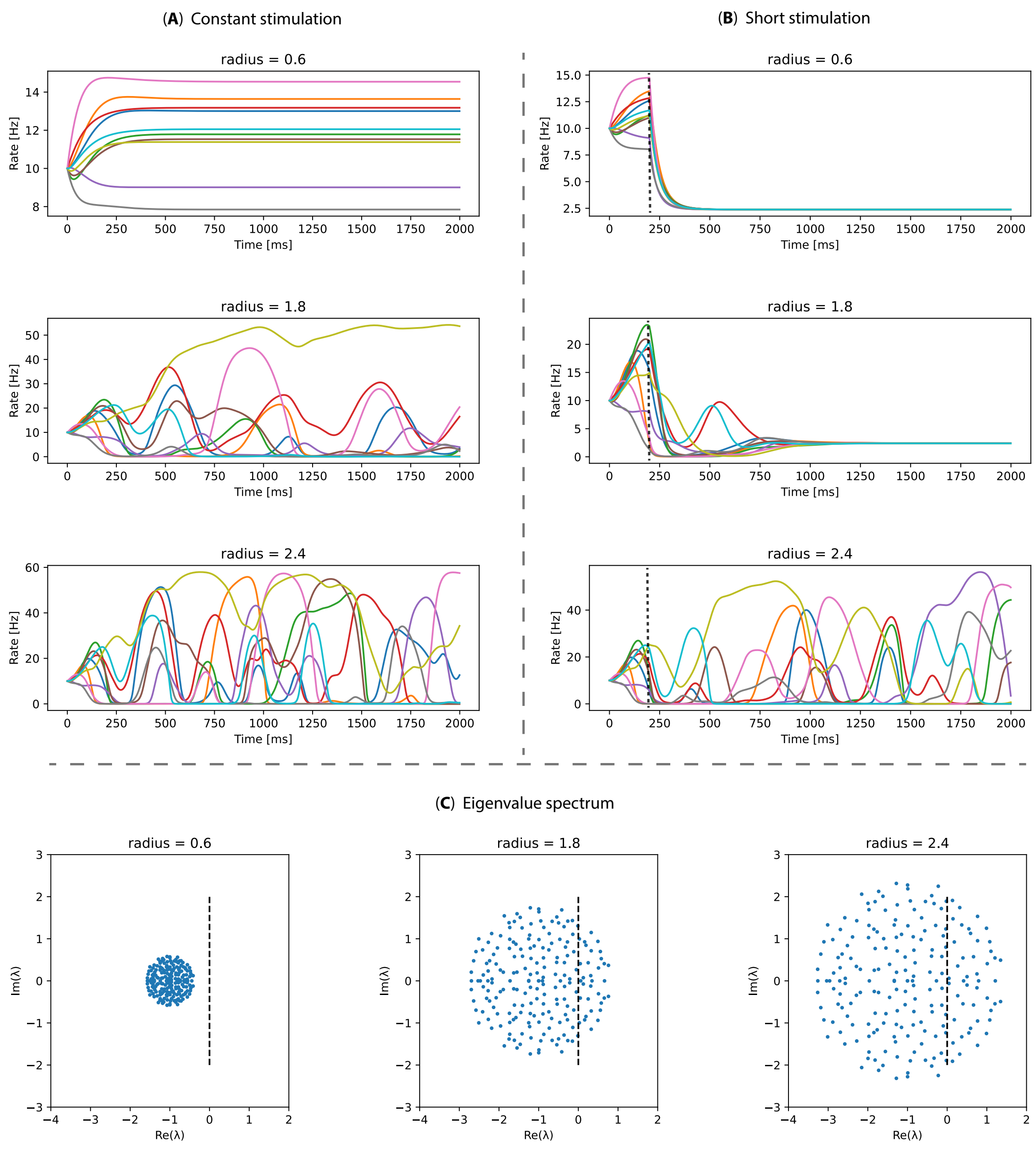


Figure 3.3: Simulation of a population of 200 rate neurons for three different connectivity matrices. *Left* (**A**): Each neuron receives a constant input the whole time, the curves represent the activity of 10 randomly picked neurons. *Right* (**B**): The neurons receive a constant input only during the first 200ms, the vertical dash line on the plots indicates when the stimulus stop. 10 randomly picked neurons are visualised. *Below* (**C**): The eigenvalues of each of the of the networks $\mathcal{EV}(\mathbf{W} - I)$.

3.3.1.2 Network activity analysis

As we show in Fig. 3.3, the stability of the network depends on the eigenvalue spectrum of its connectivity matrix. That is, asymptotic behaviour of the network as $n \to \infty$ is resilient to small perturbations in the initial conditions when most of its Jacobian matrix eigenvalues have a real part smaller than zero, ie. $\mathcal{EV}(\mathbf{W} - I)$, where $\mathbf{W}$ is the connectivity matrix and I is the identity matrix.

We now would like to inquire if the network displays chaotic behaviour understood as exponential sensitivity to small input perturbations. In order to do so, we run a series of simulations where we stimulate the network with random stimulation sampled from an uniform distribution $\mathcal{U}(0, 70)$ to which we add a small amount of normal noise $\mathcal{N}(0, 0.01)$. If we run both simulations for a network with a small spectral radius $\mathcal{R} = 0.5$ we do not observe divergence on the network activity while the network activity quickly diverge for radius 3.0:

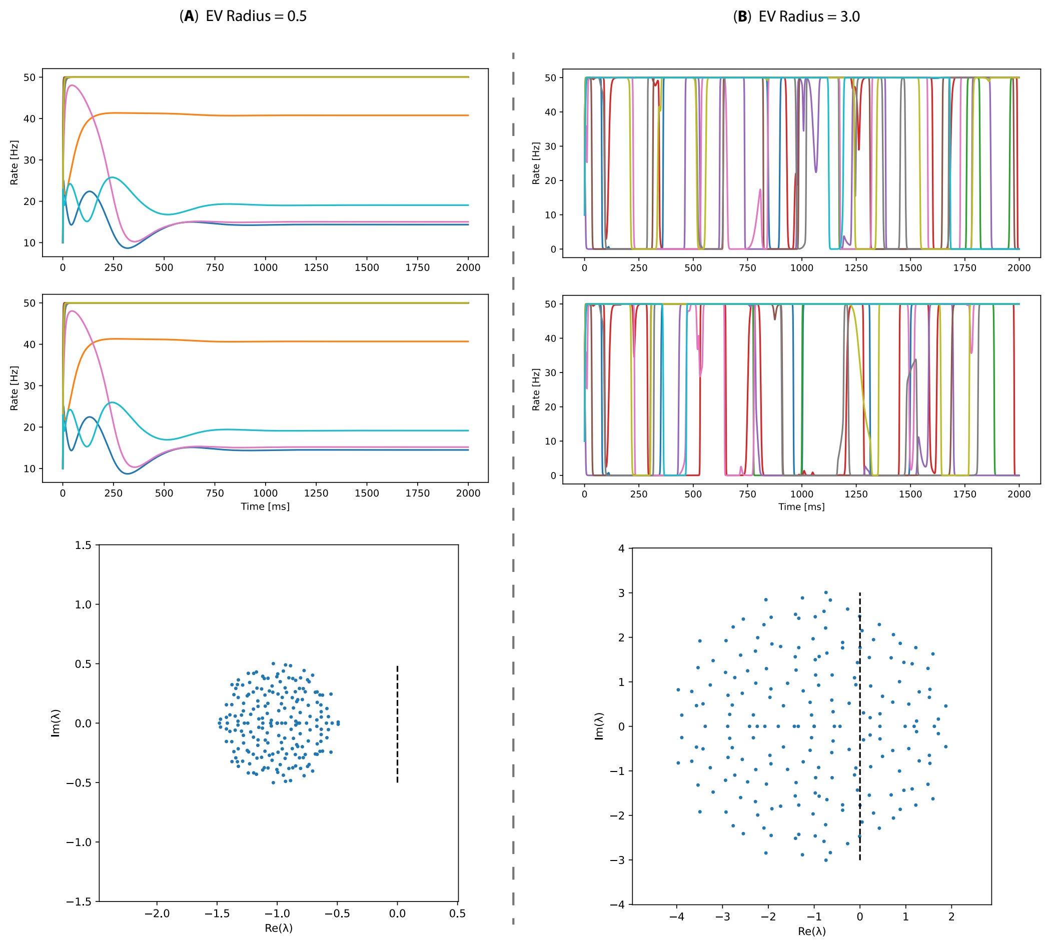


Figure 3.4: Two simulations of a population of 200 rate neurons with two connectivity matrix built such that their spectral radius $\mathcal{R} = 0.5$. (**A**) and 3.0 (**B**) respectively. In each case, the network is stimulated with an random stimulation $\mathcal{U}(0, 70)$ and the same stimulations plus a small perturbation $\mathcal{U}(0, 70) + \mathcal{N}(0, 0.01)$; for this simulation the stimulation mean is 35.80 and mean absolute magnitude of the added noise is 0.007. For better visualisation, only the activity of 10 neurons out of the 200 of the populations are displayed; the divergence between the original and perturbed activity is visible after timestep 500ms.

The same results are observed if instead of using a constant stimulus, we stimulate only the network for a short period of time of 200ms and then remove the input 3.5:

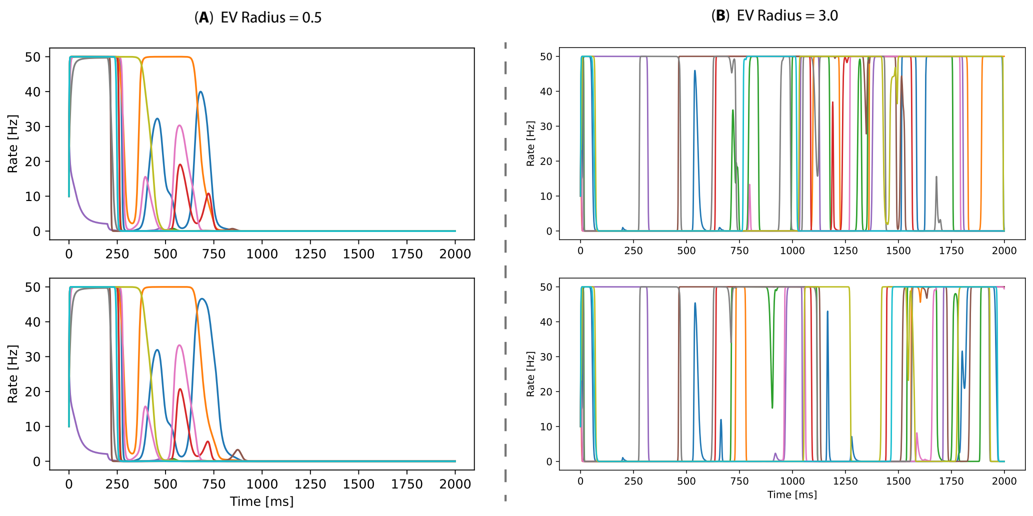


Figure 3.5: Similar simulation as depicted in Fig. 3.4, but with only a limited initial stimulation of 200ms. In the case of spectral radius of 3.0, we also observe divergence on the network activity in the presence of small perturbation despite the limited time duration of the stimulus; the divergence between the original and perturbed activity is visible after timestep 750ms.

Stability and chaotic behaviour In the Appendix 8.0.2, we have included a series of mode detailed experiments showing the relation between the spectral radius and network size, and the dynamical regime of the population. Namely, we observe exponential sensitivity to small perturbations in the stimulation for spectral radius $\rho > 1$ and population sizes larger than 200 neurons.

From rate activity to movement We use the method followed by Lindén et al. [1] to generate the nerve activity from the rate population activity. This results in a signal that can be physiologically interpreted as the signals in the motor nerves that can be read with electromyography devices. As we explained on section 3.1.1.6, these motor nerves innervate with muscle fibers forming motor units whose activity results in movement. Rhythmic movements, such as walking or breathing, is characterised by the activity of flexor-extensor muscles in phase opposition. Similarly to previous results [1], we also observe these phase-opposition activities in our simulation, see Fig. 3.6.

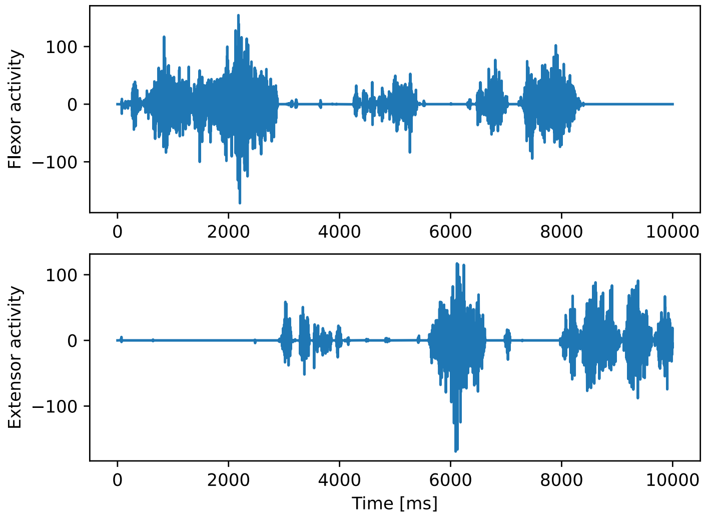


Figure 3.6: Flexor-Extensor nerve activities showing slow alternating oscillations. Activity computed from a 1000 rate-neurons population simulation with a connectivity of spectral radius $\mathcal{R} = 5$.

3.3.2 Simulations of spiking models of spinal motor control

3.3.2.1 Defining a spiking neural network

In order to simulate a neural population, the following elements need to be defined.

Neuron model In our network model, we restrict ourselves to point-neurons, and use the leaky-and-integrate model presented in section 2.2.3.2, that is, a model that abstracts the low-level dynamics of neurons and solely reproduce their spiking activity. In their non-stochastic version, LIF networks present analytical solutions for some parameter ranges and can therefore be solved either exactly or through numerical integration of its coupled differential equations.

Neuron morphology In our case, we restrict our simulations to point-neurons with no morphological information. This model choice enables us to run simulations with tens of thousands of neurons on a regular desktop computer in minutes.

Connectivity and network size The connectivity of a network consists of its topology -which neurons are connected to which neurons- and the synaptic strength of each of these synapses. We use both random connectivities as well as networks with a single coupling parameter (positive for excitatory connections and negative for inhibitory ones). Biological neural population in mammals are highly sparse, that is, most neurons are not connected to most neurons, we incorporate this sparsity in our model by using a sparse connectivity matrix. For example, in a simulated population of 1000 neurons with sparsity of 10%, each neuron will be connected, on average, to 100 neurons.

An important issue is the choice of the number of neurons in the simulated population. To capture the dynamics of neural circuits, prevent boundary effects and obtain statistically significant results, a high enough number of neurons is necessary. However, high network sizes have a computational cost that needs to be taken into account.

Other parameters The LIF model requires to define the membrane time scale τ, the strength of the external input as well as the experimentally observed synaptic delay. A full list of the parameters used for each model is provided in table 3.2. Other significant parameters such as temperature are simply not accounted for in our model.

3.3.2.2 Our model

Our spiking model is based on Brunel and Ostojic's models of the cortical populations [25], [26]. Similarly to them, we use two populations of leaky integrate-and-fire neurons, one excitatory and one inhibitory, sparsely connected to each other. Our model has a ratio between inhibitory and excitatory neurons (*E/I ratio*) to better reflect the properties of spinal cord circuits. Since we have an E/I ratio of 1, we use a relative inhibitory strength of 1 in order to preserve a E/I balance of the network. We use membrane time scale of 50ms both for excitatory and inhibitory neurons. See the network diagram in Fig. **??** and the full list of the model parameters in Table 3.2.

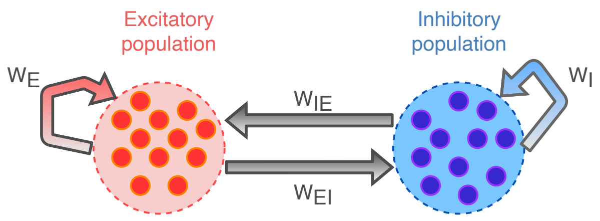


Figure 3.7: Diagram of our model. The excitatory LIF population (red) is coupled to itself with synaptic strength W_E and to the inhibitory LIF population (blue) with strength W_{EI}, an equivalent connectivity applies to the inhibitory population resulting in a balanced network.

Parameter	Value
Neuron type	LIF
Excitatory-Inhibitory ratio	1
Relative inhibitory ratio g	1
Synaptic delay	0.55 ms
Resting potential	-24 mV
Firing threshold V_{Fire}	20 mV
Reset potential V_{Reset}	10 mV
Membrane time scale τ_m	50 ms
Refractory period	0.5 ms
Connectivity sparsity	10%
Initial membrane potentials	$\mathcal{U}(V_{reset}, V_{Fire})$ mV
Simulation resolution	0.1-1 ms

Table 3.2: Parameters used to define the spiking population model (unless stated otherwise).

3.3.2.3 Network activity

The activity of a spiking network can be investigated at three levels: the spiking time of the population's neurons -visualised as raster plot-, the sub-threshold membrane dynamics (not to be confused with extracellular local field potentials or LFPs) and the average activity of the population. In Fig. 3.8 we show these three levels of activity for a simulation of our spiking model with 100 neurons over 2000 ms.

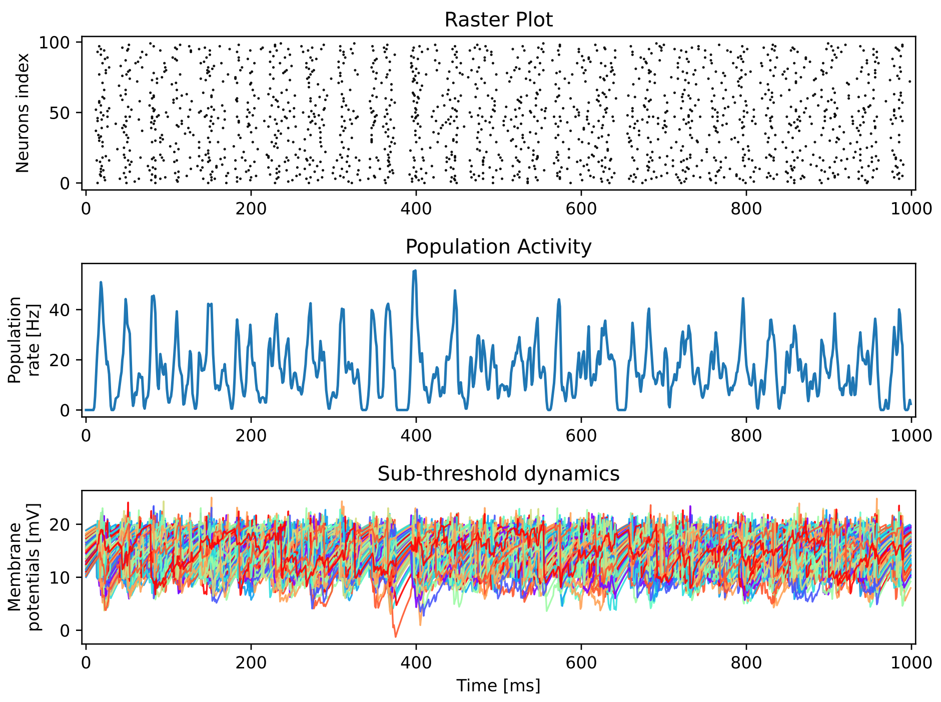


Figure 3.8: Simulation of a spiking population of 100 LIF neurons for 1000ms. The raster plot shows the spiking times of each of the neurons. The population rate plot displays the population-average over a Gaussian time-window of 1ms std. The sub-threshold dynamics plot shows the membrane potentials throughout the simulations. See Table 3.2 for the whole set of simulation parameters.

The connectivity of the network is sampled from a normal distribution of mean zero, this results in a E/I balance where we total strength of excitatory synapses is approximately equal to that of the inhibitory ones. The connectivity of the population shown in 3.8 can be seen in Fig. 3.9.

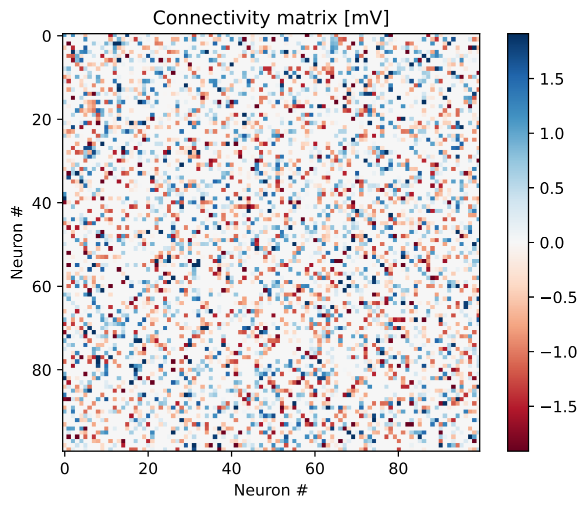


Figure 3.9: Connectivity (sampling distribution $\mathcal{N}(0,1)$mV, sparsity 40%) used to simulate the population of 100 LIF neurons whose activity is shown in Fig. 3.8. Red value indicate excitatory synapses (positive coupling) while blue ones are inhibitory ones (negative coupling). White ones reflect no connection between two neurons (zero coupling).

Types of population activity

The activity of spiking populations is typically described using the categories of synchronicity (population level) and regularity (neuron level) [25]. A synchronous activity involves a larger percentage of neurons in the population often firing at the same time, conversely the activity is said to be asynchronous when the spiking times do not coincide; this translate into synchronous population mean-firing rate displaying clear oscillations (clear peaks on its FFT) while staying approximately constant in an asynchronous regime (broad FFT peaks). We show that our spiking model can exhibit both synchronous and asynchronous activity, see Fig. 3.10.

Regularity refers to the variance of the **I**nter**S**pike **I**ntervals (known as ISI), that is, the time between successive spikes of each neuron of the population; the activity is considered regular when interspike variance is small, and irregular in the case of high variance, see Fig. 3.12. Regularity can thus be quantified out of the ISI variance distribution.

The presence of synchrony in neural activity is believed to have large influence on the computational regimes of neuronal populations and, more generally, has a strong epistemic pull in neuroscience. For instance, the synchronous activity of cortical populations is used to characterise sleep state and is widely believed to be associated with distinctive states of mind in humans [60], [61]. The study of synchronicity through power spectrum analysis, although prominent in electroencephalography studies, is not common in computational models of neural populations. We propose the use of the Fano factor of the Fourier transform magnitude F_{FFT} and of population rate F_{Rate} activity as metrics of synchrony in simulated networks.

$$F_{FFT} = \frac{\sigma^2(|FFT(w)|}{\mu(|FFT(w)|)}, \qquad w \in \text{frequency domain} \tag{3.3a}$$

$$F_{Rate} = \frac{\sigma^2(Rate(t))}{\mu(Rate(t))}, \qquad t \in \text{time domain} \tag{3.3b}$$

To quantify the regularity of the activity, we compute the variance of the ISI of each neuron and treat the resulting sequence as a random variable for which we compute its variance, this final quantity is what we define as irregularity:

$$Irregularity \propto \sigma(\sigma(ISI_i), \qquad \text{for each neuron } i \text{ of the population} \tag{3.4}$$

With these two metrics defined, we can numerically explore the different networks regimes for different set of parameters, see Fig. 3.13 for an analysis of our spiking population.

Ostojic [26] further distinguishes asynchronous activity between homogeneous asynchronicity and heterogeneous synchronicity, the later being characterised by higher instability and present in highly coupled networks and the former reflecting stable activity and emerging from weakly coupled populations.

The reason behind these criteria categorisation for the stationary activity of spiking populations is that asynchronous activity in balanced networks is conjectured to be the primary medium for propagation and processing of information in neural circuits [26], furthermore, asynchronous activity can be further split into two dynamical regimes with which define their computational properties: a weakly-coupled network leads to a dynamical regime which displays regular asynchronous activity and is capable of transmitting information but can not perform complex transformation on the incoming signal; conversely a strong-coupled network leads to an irregular regime which observes richer dynamics and therefore computationally more expressive [26], [62], [63]. Notice how the irregularity plot in Fig. 3.13 shows the activity of individual neurons in balanced networks (relative inhibitory strength ≈ 1) can display both regular and irregular patterns of firing activity. Hence, demonstrating how balance populations may exhibit distinctive computational regimes.

In terms of spinal circuit modelling, this computational bistability could enable the same common circuit architecture (3.7) to specialise to perform both *1)* forwarding of incoming signals along the spinal cord, and *2)* to generate the activity patterns that result in movement (3.1.1.3), by simply adjusting the synaptic coupling, for instance, through a activity-guided plasticity mechanism during motor development. We further develop this hypothesis in the future work section.

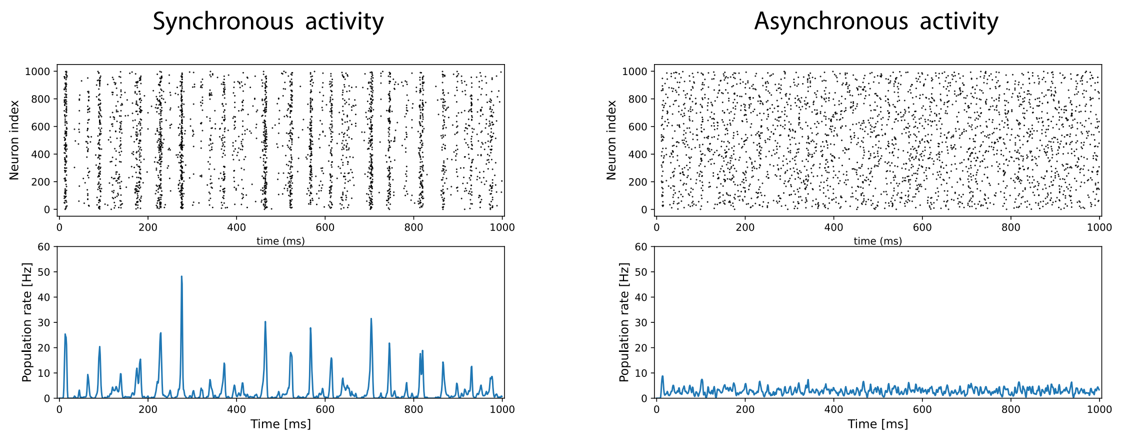


Figure 3.10: Examples of synchronous (left) and asynchronous (right) activity exhibited by the spiking population of 1000 Leaky Integrate-and-Fire neurons. **Synchronous**: Population rate Fano factor : 9.52Hz. Population parameters: sparsity 20%, connectivity : single coupling of ±1.0 mV (positive for excitatory, negative for inhibitory synapses), V_{rest} : -24 mV, relative inhibitory strength : 2. **Asynchronous**: Population rate Fano factor : 0.55 Hz. Population parameters: sparsity 10%, connectivity : single coupling of 0.1 mV, V_{rest} : -24 mV, relative inhibitory strength : 10. Non-stated parameters follow those in Table 3.2.

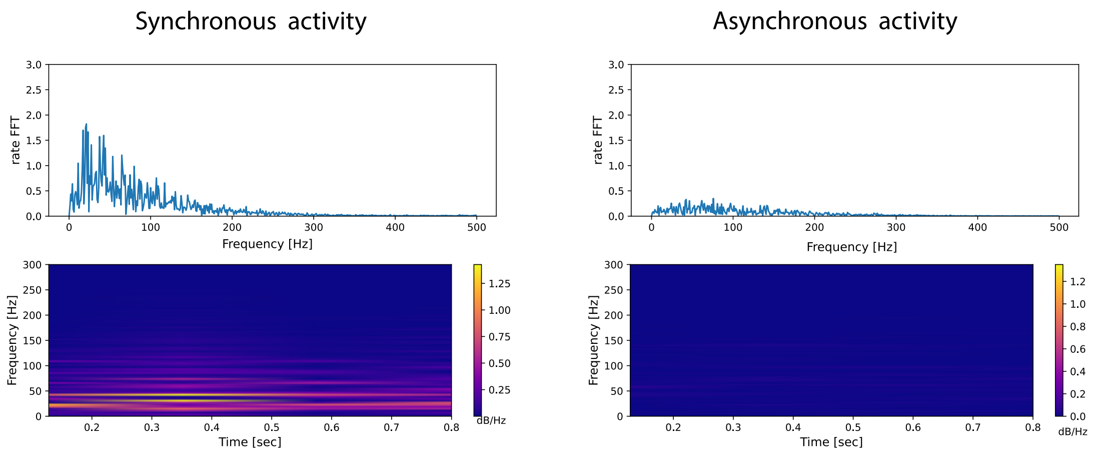


Figure 3.11: Fourier transform (magnitude) and density spectrogram of the population activities shown in Fig. 3.10. We can see that the dispersion of the magnitude of the Fourier transform is significantly larger when the network exhibits synchronous rate activity, thus justifying our choice of the Fano factor as a suitable metric to quantify the population rate synchronicity. **Synchronous**: FFT Fano factor : 0.48 Hz. **Asynchronous**: FFT Fano factor : 0.08 Hz. Population parameters same as Fig. 3.10.

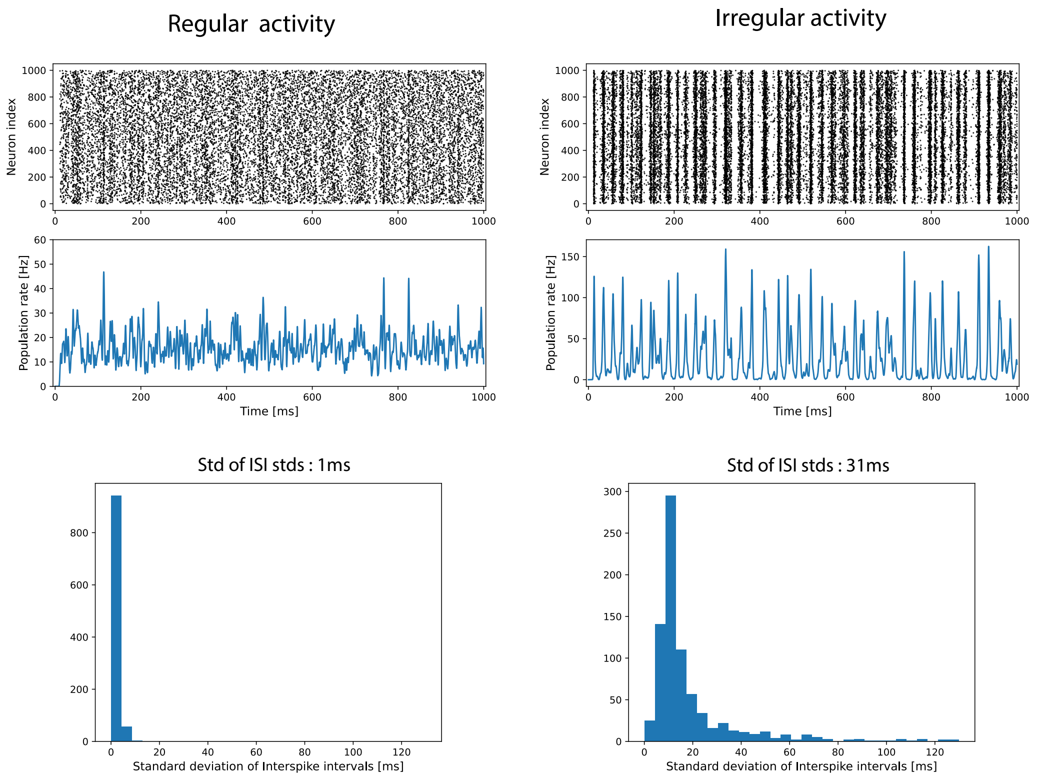


Figure 3.12: Examples of regular (left) and irregular (right) activity. Although the regularity of the spiking times is hardly visible on the population activity, computing the standard deviation of the InterSpike Interval distribution (ISI) swiftly reveals that neurons in highly coupled networks observe a much more irregular firing activity pattern. In this example, an increment of the coupling parameter from 0.1 mV to 1 mV results in a increase of the regularity metric from 1 ms to 31 ms. **Regular**: Standard deviation of ISI standard deviations : 1 ms. Population parameters: sparsity 10%, connectivity : single coupling of 0.1 mV, V_{rest} : -24 mV, relative inhibitory strength : 1. **Irregular**: Population parameters: sparsity 10%, connectivity : single coupling of 1.0 mV, V_{rest} : -24 mV, relative inhibitory strength : 1.

3.3.2.4 Effect of coupling and relative inhibitory strength on synchronicity

The dominant factors in determining the type of network behaviour are the coupling strength and the relative inhibitory strength. We investigate the synchronicity and regularity regimes by simulating a spiking population for a range of parameters and computing the metrics we have introduced. The results are shown in Fig. 3.13.

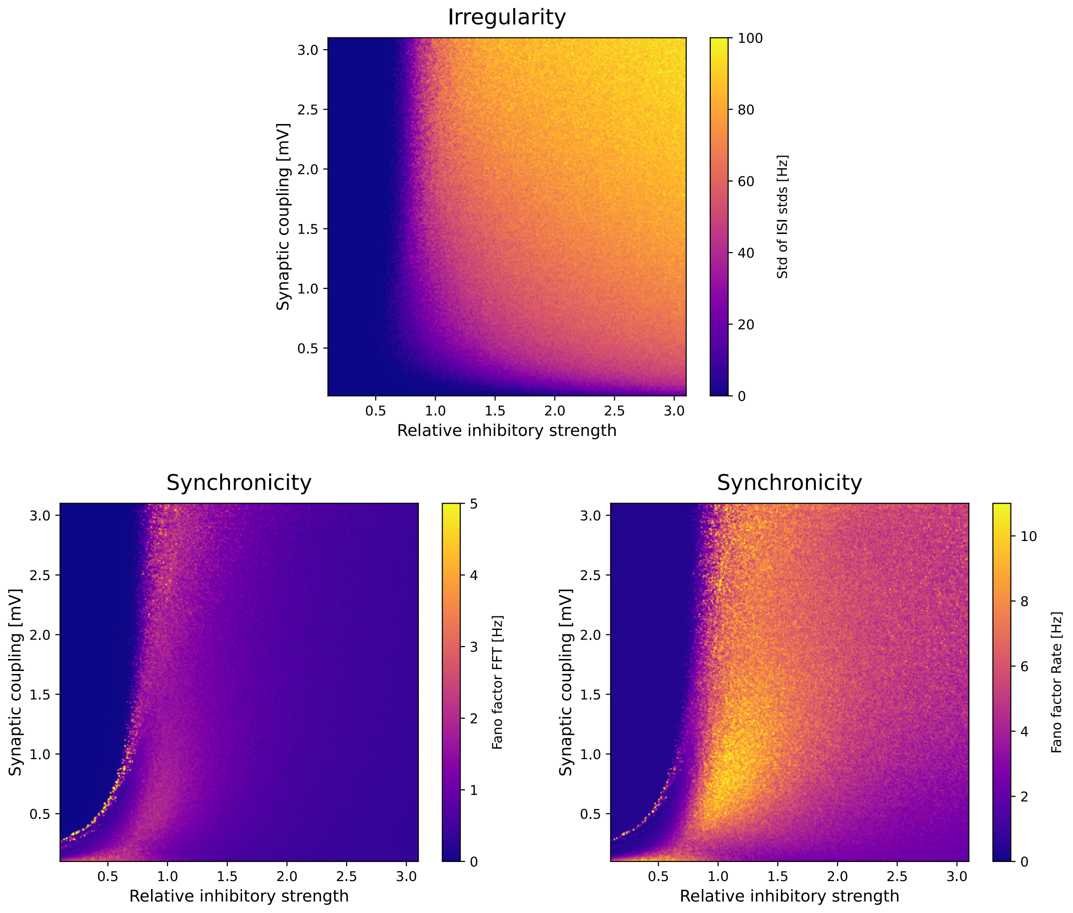


Figure 3.13: Effect of coupling and relative inhibitory strengths on the regularity and synchronicity of the spiking population activity. Each point represent a simulation of a 1000 LIF neurons for 1000ms (population parameters: Table 3.2 for which the irregularity (Eq. 3.4) and synchronicity metrics (Eq. 3.3) are computed. Notice how both the Fano factor of the population rate and FFT display a similar pattern, attesting that both metrics capture synchronicity similarly.

3.3.2.5 From rate activity to movement

As we showed in a previous section 3.6, rate models are capable of generating slow oscillatory activity similar to that experimentally observe in the spinal cord of turtles. We follow the method introduce by Lindén et al. [1] to generate nerve activity from the rate population activity and apply it the activity of spiking networks. To do so, we simulate a population of 1000 leaky integrate-and-fire neurons for 6400 ms, then convolve the generated spiking trains by a gaussian filter with a standard deviation of 350 ms, high-passed it with a Butterworth filter (order 3, cut-off 0.3 Hz) and compute its PCA and the nerve activity. The resulting activity can show slow alternating phase-opposing oscillations as we show in Fig. 3.14. The rest of the population parameters are reported in Table 3.2.

As with the rate-population, the resulting activity can be interpreted as the signals of the motor nerves that can be read with electromyography devices. As we explained on section 3.1.1.6, these motor nerves innervate with muscle fibers forming motor units whose activity results in movement.

Although the generated activity does not yet fully match the electrophysiological activity seen in experimental recordings, this contribution is a stepping-stone towards building spiking models of motor-generation circuitry in the spinal cord.

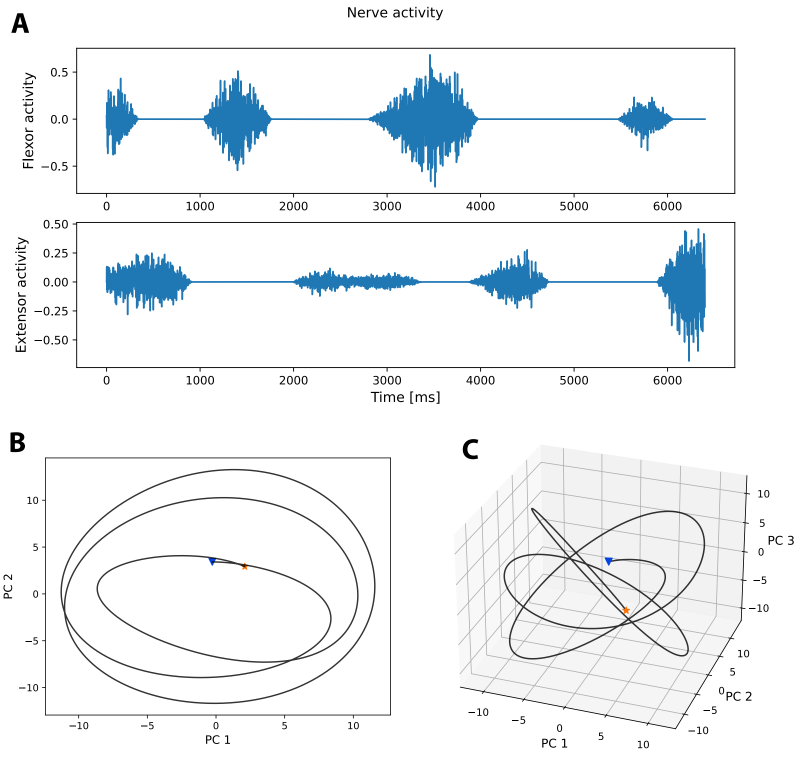


Figure 3.14: **A**: Flexor-Extensor nerve activities generated by a population of 1000 spiking neurons showing slow alternating oscillations; alternation between flexor and extensor muscles activity form the basics of biomechanical movement in mammals. **B** and **C**: The PCA of the convolved spiking activity exhibits rotational dynamics, however the first three components only account for 53% of the variance (23%, 20%, 15% respectively); yellow start indicates the initial timestep and blue triangle the last one. Population parameters: 1000 LIF neurons, connectivity distribution $\mathcal{N}(0,1)$mV, sparsity 10%, relative inhibitory strength 1. The rest of parameters are shown in Table 3.2.

Chapter 4

Neural Manifolds & Network's Dynamics as Computations

In this chapter we introduce the notion of *manifold* as used in the neuroscience literature. We then discuss the relation between dynamics and the computations done by neural populations and illustrate the different geometries generated by different dimensionality reduction techniques. We conclude by introducing the notion of *manifold induction* which is the base of our proposed manifold-hypothesis testing protocol that we introduce in Chapter 6.

4.1 Neural Dynamics, Computations and Behaviour

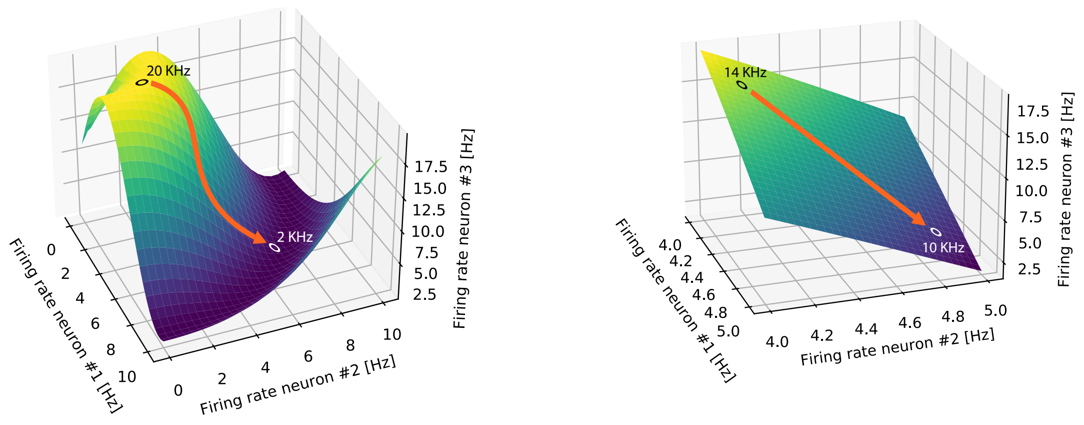
4.1.1 Manifolds in Neuroscience

Mathematically, a manifold is a topological space which is locally Euclidean [64]. This means that manifold space can be approximated by a –potentially infinite– set of overlapping Cartesian maps. A geometric intuition can be developed by thinking of the earth surface which –at a human scale– can be considered Euclidean. Intuitively, this is why walking on a city feels like moving on a flat surface and can be navigated using flat maps, despite the Earth being spherical. In many cases, we can find scales where the Euclidean approximation is effective enough to characterise a system.

In neuroscience, the term *manifold* suffers of an abuse of terminology and does not rigorously reflect the mathematical definition of manifolds. In this section we try to convey the notion of manifold as it is used in neuroscience: a low-dimensional representation of the state space of neural activity spanned by the low-dimensional representations of a collection of neural-activity trajectories. Therefore the manifold can be seen as the space capturing the variability of the neural population activity during a given behaviour or sensory processing; that is, a neural activity representation from which we can compute the low-rank approximation of the covariance matrix of the high-dimensional neural activity. This is the sense of expressions found in the literature such as *discovering the neural activity manifold of a behavioural task* [65]–[71]. In this framework, the geometry of the manifold represents the computation being performed by the its underlying neural ensemble.

A fundamental question in neuroscience is how to relate very high-dimensional neural activity to the few relevant variables during a behavioural task, eg. how the collective activity of thousands of neurons in the spinal cord control the movement of a few muscles. Therefore, the sculpting of a manifold out of neural data is done with the (potentially deceitful) constrain of making it reflect the dimensionality of the behaviour. Although manifolds of neural activity are typically presented in 2D or 3D spaces, from purely descriptive studies, it can not be assumed that such a low-dimensional representations can characterise the dynamics of the system. We will discuss a perturbative approach in Chapter 6 to test the meaningfulness of manifolds; for instance, testing what information from the system can be decoded from its manifold or whether inducing the activity manifold in the neuronal population also induces the behaviour that we claim that the manifold characterises; we build an approach toward this in Sections 4.4 and 5.5.

Let us explore the notion of manifolds with a thought experiment. Suppose we want to study how the pitch of a sound is encoded by three neurons in the auditory cortex. To do so, we produce a single tone of frequency 20KHz and progressively bring it down to 2KHz while we record the spiking rates of those three neurons. The recorded spiking rates of the neurons over many trials results in the Fig. 4.1 where the arrow represents a single trial. [1]



(a) Global state space (b) Zoomed-in to a local region of the state-space

Figure 4.1: A neural activity manifold. The 2D surface is the neural manifold, the initial and final points are indicated as circles, and the arrows indicates the change in pitch on the stimulus sound. In 4.1a we can see that the neural activity state space is non-Euclidean, however if we zoom in over a subregion 4.1b, we can see that the region travelled by the neural activity pattern is approximately Euclidean and we can connect two points by a straight line.

The 2D surface represents a neural activity's manifold. Each of the points of this surface is a neural response to the sound stimulus. The orange arrow represents the temporal evolution of the neural activity over a single experiment trial -also referred to as *latent activities* or *latent dynamics*-. We can see that although the global geometry of the state space is non-Euclidean 4.1a, locally it becomes effectively linear when we zoom-in in a small space subregion as we can see in Fig. 4.1b. That is, we expect small changes in the sound pitch to only produce linear changes in the neurons' firing rates.

The *intrinsic dimensionality* of a manifold is defined as the minimal number of variables needed to parameterize it, while its *embedding dimensionality* is the number of dimension of the space in which the manifold lives. The former reflects the behavioural variables (stimulus, movement, etc.) being encoded while the the latter is a manifestation of the transformations and processing applied to these variables –and other latent information like memories or attention– occurring through the dynamics of the neuronal activity [72].

From the Fig. 4.1a, we can observe that all possible state-space trajectories live on a 2D dimensional surface, therefore there exists a lower-dimensional space (2D rather than 3D) that fully captures the neural activity elicited by the pitch-shifting sound. What is more, the pitch seems to be highly correlated with the firing rate of neuron #3, a dimensionality reduction analysis of the data could potentially yield a 1D manifold which would explain most the variance. Naturally, this is a toy example with purpose of exemplifying the notion of manifold, real neural datasets are made of spaces of hundreds or thousands neurons recorded simultaneously over many time-points resulting in very high dimensional spaces whose

[1]This model assumes that pitch is encoded using a rate-coding formalism. For cognitive computations relying on temporal encodings each point of the manifold -eg. pitch 20KHz- would be captured by a trajectory in neural-space rather than a single point.

dynamics we try to capture with manifolds of intrinsic dimensionality as small as possible.

Neural populations feeding from sensory stimuli should observe activity whose dimensionality mainly reflects the input stimuli, with the auditory cortex responses representing the latent structure of sound. On the other hand, neurons projecting to motor centers will be best characterize by the dimensionality of the outputting motor behaviour. Intermediate areas between sensory and motor will have the highest dimensionality since they do not only need to rely the sensory signal but also need to incorporate latent information such as memory, attention, or context that mediates behaviour [72].

4.1.2 The Manifold Hypothesis in Neuroscience

In the context of animal neural activity, *manifold hypothesis* states that high-dimensional data tend to lie in the vicinity of a low-dimensional manifold. This conjecture is not exclusively applied to neuroscience, but it is also highly investigated in data-centric fields such as machine learning or data-science [73]–[77]. In the context of neural activity, this would mean that the number of degrees of freedom of a neural network is significantly smaller than the number of neurons that make up the network. The possible patterns of neural activations are constrained by the connectivity of the network and its parameters, in other words, it is the specific connectivity of the network, along with its bio-physical parameters and the input received by its neurons, that gives rise to the shape of the manifold. Accordingly, we can see the manifold as the space of possible activity patterns of a network under natural conditions [78], [79]. The hypothesis is backed by experimental results across different brain regions and behaviours [70], [80]–[83].

As discussed in section 2.2.5.4, neuroscience has seen a paradigm shift from studying single neurons to now considering populations of neurons as the meaningful unit of neural activity capable of explaining behaviour. In this context, manifolds are a useful concept that allows to reconcile the observations that the neural code appears to be distributed across many neurons but also low dimensional in relation to the total number of neurons present in neural tissue.

4.1.3 Computation as Neural Population Dynamics

The goal of neuroscience is to understand the casual relation between the dynamics of the central nervous system at different scales and behaviour, both in healthy and pathological individuals. The challenge of this endeavour derives from two elements: firstly, the technical difficulty of performing measurements on *in vivo* animals in a non-invasive manner. Secondly, the difficulty of understanding high-dimensional non-linear phenomena. The first of these challenges has seen a steady progress over the last 70 years [84]. Tungsten electrodes in the 50s allowed for the first single-cell recordings while today's micro-electrode arrays like *Neuropixel 2* are capable of recording 5120 sites per probe at sub-millisecond resolution [85]. At slower temporal resolutions of the order of 1Hz to 30Hz, optical methods such as fluorescent two-photon calcium imaging allows to record tens of thousands of neurons simultaneously [86], [87].

Regarding the challenge of understanding high-dimensional non-linear phenomena, we still lack the conceptual tools to bridge the gap between the dynamics of complex systems like biological neural networks and the behaviours they beget. The *Computation Through Neural Population Dynamics* framework proposes to understand the computations performed by neuronal populations *as* the dynamics of their activity [71]. That is, using dynamical systems theory as a proxy to understand the computations enabling animal behaviour. See diagram in Fig. 4.2.

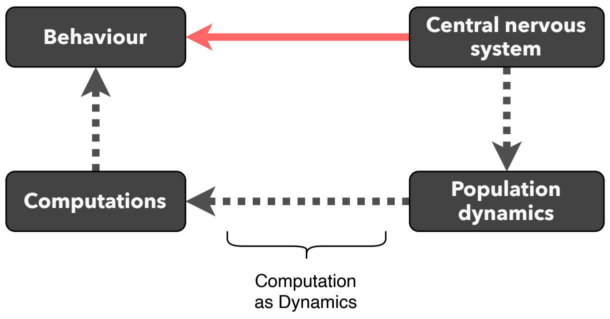


Figure 4.2: Computation as Neural Population Dynamics. The red arrow indicates the final goal; black dashed arrows indicate the proxies approach of using dynamical systems methods. Diagram adapted from Mehrdad Jazayeri's talk at Emory TMLS workshop [88].

The notion of computation refers to the set of operations or transformation that an agent applies to its sensory *input* or to its own internal state to produce an action or behaviour. The resulting *output* of the computation may be in the form of perceptual, cognitive[2] or motor action. Let us write this notion in the form of a differential equation 4.1, where $\boldsymbol{S}$ is the state-vector of the neural population (eg. membrane voltages of each neuron), $\boldsymbol{Input}$ is the stimulation they receive and $\mathcal{F}$ is the non-linear function that, through its temporal evolution, encodes the computation performed by the neural population.

$$\frac{d\boldsymbol{S}(t)}{dt} = \mathcal{F}(\boldsymbol{S}(t), \boldsymbol{Input}(t)) \tag{4.1}$$

Under the dynamical system lens, we can look at the state space of the neural population during a task and investigate the presence of fix points, limit cycles, oscillators, attractors, etc. How is this useful to reveal the computation performed by a neural circuit? Let us illustrate it with two examples, a cognitive and a motor computation:

4.1.3.1 Cognition

We want to discover which neural circuits enable the recalling of memories in an animal executing a memory-recalling task. We perform recordings in several brain regions and examine their state spaces. The presence attractor region with a fix-point representing the memory being recalled would hint at that area as candidate for being involved in the task's computation.

4.1.3.2 Motor control

If we think that a given circuit is responsible for generating a periodic muscle movement -eg. running-, we would expect to observe a limit cycle on its neural activity state space driving the activity of the muscles; furthermore, we can see the necessary preparation phase of movement performed before the movement is executed as the the neural activity being positioned in a state-space sub-region -i.e. initial condition- from where it can generate the state-space trajectory that will give rise to the movement [71].

On the grounds of the notion of manifold and the *manifold hypothesis* discussed in 4.1.1 and 4.1.2, we can investigate the function of a neural circuit by investigating the geometry of the neural-activity trajectory across the manifold -as well as the flow field- during behaviour [89]–[91].

[2]By *perception* we understand changes in the internal state of agent (i.e. the state of the neural population) that are largely driven by input stimulus while we use *cognition* to refer to changes in the agent's internal state largely predominantly driven by other internal states.

4.2 Dimensionality reduction techniques

Dimensionality reduction techniques are unsupervised algorithms that allow to reduce the dimensionality of the representation of data by finding new basis to represent it in meaningful ways. When applied to data visualisation, dimensionality reduction is applied to high dimensional data and typically project onto a 2D or 3D space such that nearby points share similarities while far-away points do not; in the best case, this projection will lead to the emergence of clusters that can be used to gain insights about the data. We present here three popular dimensionality reduction techniques:

4.2.1 PCA

Principal component analysis (PCA) is a linear non-parametric technique for reducing the dimensionality of a dataset by finding a new lower-dimensional basis to represent it [92]. The dimensions of new basis will be orthogonal, and in the directions that maximise the captures variance. The process of finding such a basis –or principal components–, comes down to find the eigenspectrum of the dataset, therefore many techniques can be used to implement it, typically being done thorough singular value decomposition which is numerically more stable than the covariance method [93], [94].

4.2.2 Kernel PCA

By design, PCA would fail to capture a low-dimensional basis of datasets with non-linear structures. Kernel PCA is an extension of the PCA algorithm that allows to find *non-linear* low-dimensional representations by projecting the data onto a higher dimensional space -or *implicit feature space*- through a kernel function from which a linear lower-dimensional can be constructed [95]. This method requires the user to specify the projective kernel, hence requiring to conjecture some priors about the dataset.

4.2.3 t-SNE

t-distributed stochastic neighbor embedding (t-SNE) is a non-linear non-parametric method used to visualise high-dimensional data into a 2D/3D space [96].

The t-SNE algorithm is made of three steps. Firstly, t-SNE builds a Gaussian joint probability distribution over the pairs of the high-dimensional input data in such a way that similar data-points are assigned a high probability while dissimilar points are assigned a low probability. Secondly, t-SNE defines a Student t-distribution over the points in the low-dimensional embedding space[3]. Thirdly, it uses gradient descent to minimize the Kullback-Leibler divergence (KL divergence) between the two probability distributions with respect to the locations of the points in the low-dimensional embedding space for a given metric, typically euclidean distance.

As indicated, t-SNE builds the low-dimensional representation of the data by minimizing the KL divergence of the joint probabilities in the original space and the low-dimensional embedding space. This KL divergence is a non-convex loss function that needs to be minimized to build the final representation, therefore, different optimisation runs may yield different results. To palliate this, the use of several seeds is advised. The relative position of clusters and their sizes on the t-SNE plot is largely arbitrary and depends on the random initialisation of the optimiser.

Modern neural recording techniques allow to capture up to 40000 neurons at the same time, each with thousands of time points. Since its introduction in 2008, t-SNE has become one of the most popular dimensionality reduction techniques used to analyze and visualise these high-dimensional neural recordings. By observing the formation of clusters and correlating them to properties of the neurons, eg. if we have a recording of neurons from a living animal while producing two types of motor behaviour, we would expect the emergence of two clusters each associated to each of the movements performed by the animal. Similarly, clustering techniques can be used to distinguish between neuron types or more abstract quantities such as the uncertainty of the animal during several trials of learning task. The

[3]An embedding space is the space in which the data is embedded after dimensionality reduction. Its dimensionality is typically lower that than of the ambient space [97].

resulting low-dimensional embedding spaces are often referred in the neuroscientific literature as neural activity *manifolds*.

In our analysis we use the two most popular implementations of t-SNE which are *OpenTSNE* [98] and *Scikit-learn t-SNE* [99]; acording to our results both yield the same results with the Scikit's version being significantly faster.

With all non-linear dimensionality techniques, there is a trade-off between preserving local information versus preserving the global structure of the data, t-SNE often fails to preserve the global geometry of the data [100], [101]. However, it provides a *perplexity* hyperparameter that can be tuned to modulate the focus given to local and global features of the data, that is, perplexity is a proxy measure of the number of neighbors each data point will be attracted to. More technically, the perplexity parameter defines the width of the Gaussian kernel used to compute the similarity between points; the larger the perplexity, the more non-local information will be retained in the dimensionality reduction result.

Non-linear dimensionality reduction beyond t-SNE Other algorithms, such as UMAP -or its variation densMAP- put a higher focus on preserving the global structure of the data while others like Rastermap have a stronger emphasis on sustaining the local geometry of the data [36], [102]–[104]. However, the extent to which the global vs local structure is preserved highly depends on the initialisation and configuration of the gradient descent optimiser [105].

4.3 Manifold representation of rate populations

The dimensionality reduced representation of the activity of three simulations of the rate-model is presented in Fig. 4.3. We can see that all methods are capable of capturing rotational dynamics in the network activity, with the exception of Kernel PCA for the network with spectral radius $\rho = 1.5$. This seems to justify the standard practice in the motor neuroscience field of mainly employing PCA to characterise the activity of neural activity. However, it remains to be explored what are the limits of linear methods to capture certain more complex dynamical regimes.

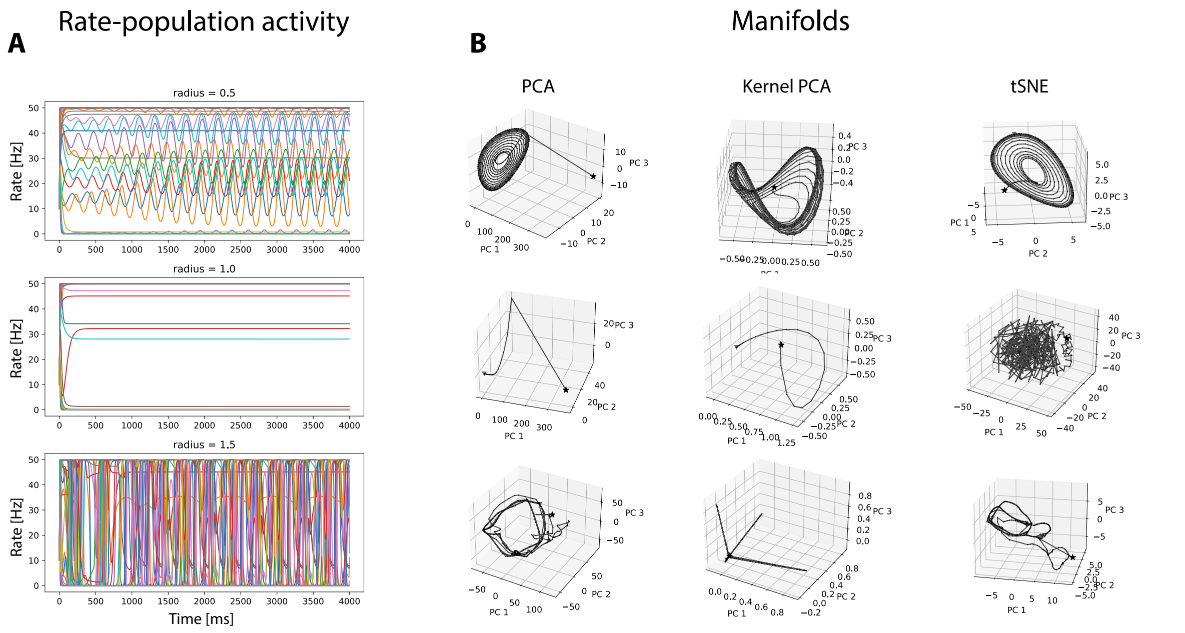


Figure 4.3: **A**: The simulated activity of three rate populations with spectral radius `0.5, 1.0, 1.5` and their corresponding dimensionality reduced representation using PCA, Kernel PCA and tSNE.

4.4 Manifold Induction

A crucial question in neural control theory is how to stimulate a population of neurons in order to trigger a specific behaviour, in our case, a gait movement. In order to do so, rather seeking to generate the specific high-dimensional activity pattern known to be correlated to the target gait, we instead exploit the observation that the latent activity is confined to a neural subspace -i.e., the manifold hypothesis- and attempt to induce a latent activity pattern whose low-dimensional representation is correlated to the behaviour we want to generate in the animal.

This perturbation can either slightly modify the current location of the activity in the phase space in order to exploit the presence of attractors in the state-space -i.e., trigger a decision which would itself trigger a motor behaviour-, or to *manually* induce the complete phase-space trajectory correlated to the behaviour we want to induce.

While the specific activity recorded from different subsets of neurons during the same task will be different, there is evidence that their collective activity, and hence their activity manifolds, is the same up to an alignment transformation [83]. In the context of motor control, this endow the manifold induction approach with the critical advantage of not having to stimulate the same subset of neurons each time as well as a potentially higher robustness across time [83]. Furthermore, recent evidence [106] shows that the the geometry of the manifold itself intrinsically reflects the nature of the computation being performed, suggesting that there might be a robust invariant component also across individuals.

In silico, we are unconstrained to simulate any modality of stimulation conceivable on our neural populations simulations. *In vivo*, our possibilities are more restricted, however the recent developments of perturbation tools such as optogenetics, which enables us to selectively stimulate neural populations with a laser-diode on freely moving animals [107], [108], open up the possibility of using the manifold induction approach in the context of motor control, and more generally for causal perturbative analyses [78] as well as a potential approach for neuroprosthetics development.

In chapter 5, we will show that a deep-learning approach can effectively be used to induce specific dynamical patterns on rate and spiking populations, an emerging approach in the field of inverse optimal control. However, as we have discussed in this chapter, there is growing interesting in understanding populations dynamics through dimensionality-reduced representations. Hence, we in section 5.5 we extend our DL-inversing framework to, rather than producing particular activity patterns at a neuron level, induce a specific activity manifold. We refer to this approach as *deep inverse optimal control through dimensionality reduction* or, more succinctly, *manifold induction*. In the context of motor control, the idea consists in employing a deep-learning architecture to find stimulations capable of inducing activity dynamics in the neural populations whose manifolds are correlated with a target behaviour, in our case, neural population activity exhibiting rotational dynamics.

Chapter 5

Deep Learning Techniques for Inverse Problems

In this chapter we present the inverse-problem formalism as well as the basics of machine learning applied to supervised learning tasks. We then present our optimisation algorithm for applying DL to synthetic data. We demonstrate how the trained models can indeed decode the neural population parameters from the raw -and dimensionally reduced- activity for both rate and spiking populations.

5.1 Inverse problems

The study of a physical system can be divided in three steps [109]:

1. **Parameterization** of the system: finding the minimal set of *model parameters* capable of fully characterizing the system at a given scale.
2. **Forward modeling**: find the system relations that, for a given set of *model parameters*, enable us to make prediction about the *model observables*. Here the observables can be seen as the input to the *forward* model.
3. **Inverse modeling**: using the system's *observables* values to infer the *model parameters*. That is, finding the casual or correlational relations that produced the observations. Here the observables can be seen the input to the *backward* or *inverse model*.

Whether a given parameter is a *model parameters* or a *observable parameters* is a somewhat accidental epistemic distinction derived from the way humans perceive causal relations. For a detailed discussion on this topic, see [110] and [111].

In our case, we are interested in the following inverse problem: controlling the dynamics of spiking neural networks with few model parameters, ideally, a set of parameters that could be perturbed in *in vivo* systems. As we have discussed, SNNs are highly non-linear systems and -like most recurrent network systems- not injective, therefore controlling their resulting dynamics is non-trivial.

5.2 Deep learning

Deep learning (DL) is a sub-field of machine learning that uses artificial neural networks (ANN) as function approximators to solve a diverse set of algorithmic tasks. ANNs are analogous to rate-based networks introduced in 2.2.4.2 and are considered *deep* when several layers are present between the input and output layers.

5.2.1 Supervised Learning (SL)

Unlike *expert systems*, where a set of hand-coded rules determine what the output should be for a given input, a supervised learning algorithm is trained on a hand-crafted dataset of domain-codomain pairs by varying the ANN weights until a suitable mapping is found. A deep learning system applied to SL consists of four elements:

- A dataset $\mathcal{D}$ of domain-codomain pairs $\{X - Y\}$
- An artificial neural network whose connection weights $\{W\}$ act as free parameters that can be tuned to produce the right mapping $f : X \to Y$
- A *loss function* $\mathcal{L}$ that allows us to compute a notion of error in a metric space between a prediction $\hat{y}$ and a ground truth y, $\mathcal{L}(\hat{y}, y)$.
- An optimisation algorithm responsible for guiding the search for optimal weights that minimise the loss function. In SL, the standard procedure is to compute the gradient of the loss function with respect to the weights of the network for the set inputoutput pairs $\frac{\partial \mathcal{L}}{\partial W}$ -also known as *backpropagation*- and subsequently update the weights through *gradient descent* or any other first-order optimisation method. This process is repeated iteratively until the loss magnitude becomes small enough. Although less popular in a SL context, black-box methods [1] can also be used to optimise the weights.

5.2.2 The artificial neural network architecture zoo

The choice of the ANN architecture depends on the nature of the dataset $\mathcal{D}$. For instance, if the data contain sequential relations, a network with recurrent connections will more suitable. If instead, the data have some sort of translation or scale invariance, a convolutional network would be pertinent. In other words, the choice of the network architecture reflects the priors over the dataset. We now briefly introduce the major artificial neural network architecture currently used to solve function approximation problems:

- **Multilayer Perceptron (MLP)** A perceptron consists of a single layer of artificial neurons which weight and add their input. An MLP is simply the feedforward network that results from stacking multiple perceptron layers and applying a non-linear transformation to each neuron output. MLPs consist of at least 3 fully connected layers: an input, a hidden, and an output layer, although it is not uncommon for MLPs to have several hundreds hidden layers [112]. The input layer dimension of the MLP reflects the data modality and size while the output dimension depends on the task being solved.

[1]Black-box methods, also known as *gradient-free* methods, are optimisation algorithms that do not have access explicit access to the function being maximised/minimised, instead they only observe a the evaluation of function for a set of inputs. They tend to be less computational efficient than derivative-based methods but have the advantage of not requiring a differentiable loss function.

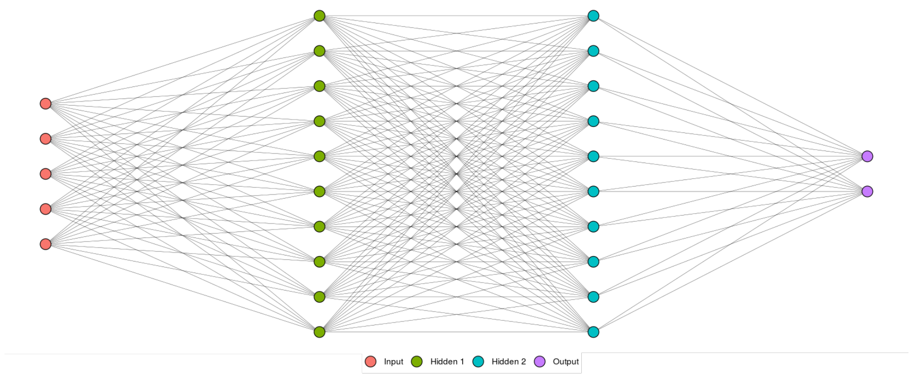


Figure 5.1: Diagram of a Multilayer Perceptron with two hidden layers.

- **Convolutional Neural Network (CNN)** In the context of image processing, a kernel -or convolution filter-, is a small matrix that can be applied to an image to in order to highlight some visual features. For instance, the Sobel filter results in an image emphasising edges. Convolutional networks apply filter kernels -aka convolutional layers- to their inputs to extract relevant features, but unlike standard kernels in image processing, the filter matrix are learned during the training process of the network. The resulting output from the convolutional layers is fed onto a MLP. Convolutional layers have the property of being scale and transnational invariant, therefore they are suitable for data that also exhibits these properties.

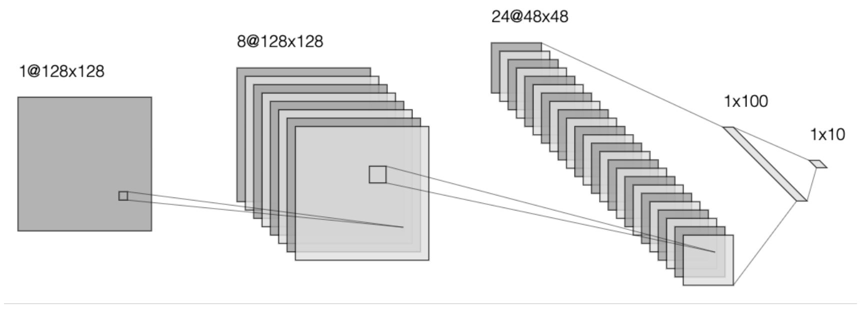


Figure 5.2: Diagram of a convolutional architecture with 2D-convolution filters. The network receives an input image, applies a series of eight kernel filters, downsamples the resulting feature activations, flattens them, and feeds them onto a linear layer with 10 output neurons.

- **Recurrent Neural Network (RNN)** Unlike MLPs and CNNs, which are purely feedforward, RNNs present recurrent connections between the neurons. These feedback connections enable RNNs to capture and exhibit temporal dynamics. They are specially suitable for sequential data. RNNs are computationally difficult to train through gradient descent due to the *vanishing/exploding gradient problem*, that is, the challenge of numerically keeping track of the loss function gradient with respect to the weights when using floating-point numbers. A model introduced to tackle this issue are **Long short-term memory (LSTM)** networks, a recurrent architecture that uses gates to modulate the flow of recurrent activations within the network. The presence of these gates results in a well-behaved gradient.
- **Beyond CNNs and RNNs**: In the section 7.1 we discuss other neural network architectures such as *Transformers* or *Echo State Networks*.

5.3 Deep learning applied to inverse problems

Deep learning provides a suitable framework to tackle inverse problems. Deep neural networks have the property of being universal function approximators. Thus, by designing a neural network model that takes the output signals of a system as input and outputs the model parameters, we can flexibly build a backward model for any parameters of interest.

Previous work has shown that deep learning can be used to solve the inverse problem of estimating the model parameters of a simulated neural population based on their extracellular local field potentials (LFP) [3]. This approach provides a method for testing whether a given information is encoded in some arbitrary signal, i.e., if the network connectivity parameters can be recovered from the LFP, this shows that the connectivity information is embedded in the observed signal.

A strong advantage of this approach is that it does not require to make assumptions about how information is encoded in the neural activity (see section 2.2.5). Instead, we feed the deep-learning system with the raw neural activity and train it to find correlations in the data, allowing us to skip over the coding question.

Moreover, inverse problems can be easily be translated to the sister problem of inverse optimal control [2]. In Chapter 6, we further explore the research applications of our framework in the context of neuroscience research.

5.4 Network inverse problem with deep-learning (DL) models

5.4.1 Inversing population stimulation

Here we simulate a population of biological neurons for a time t to create a dataset $\mathcal{D}$ of stimulation-activity pairs. These pairs are used to train the inversing-DL network by optimising its weights until it can accurately map inputting activity to the needed stimulation to elicit that activity. [3]

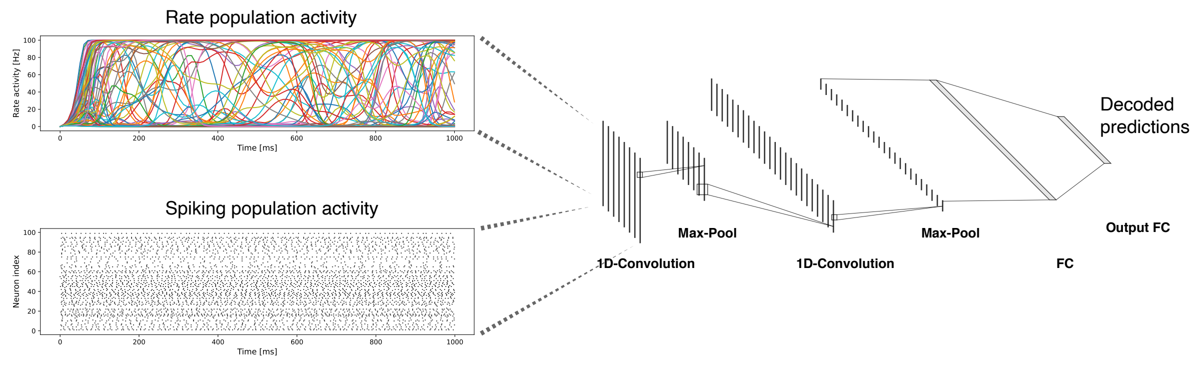


Figure 5.3: Diagram of the inverse deep-learning model. A neural population driven by a random stimulation generates an activity pattern. The population activity –which can be oscillatory activity or spikes– is fed into the convolutional network whose final layer outputs are the decoded stimulation driving the population activity.

[2]Inverse optimal control is the extension of inverse problems to dynamical systems where the model's observable are N-dimensional time-sequences rather than a single N-dimensional point.

[3]In reality, for highly coupled populations, the stimulation-activity mapping is not surjective, therefore several stimulations can potentially lead to the same pattern of activity. For simplicity, we only seek to decode the precise stimulation that was used to generate the activity.

5.4.1.1 Training deep-learning models on synthetic data

We use a version of the backpropagation algorithm that we adapted to make it operate efficiently with synthetic data. Unlike standard supervised learning problems where the dataset contains a finite number of examples, generative models allow to generate as much synthetic data as we can computationally afford. This type of algorithms are referred to as *online machine learning* –as opposed to batch methods, in which the full dataset is available from start. This has the advantage of enabling the model not to overfit to a particular subset of the data. However, it adds an extra training hyper-parameter, namely, the number of epochs[4] that we run a forward-backward pass on the network for each generated batch of data. Intuitively, if we have a very high number of epochs per batch, the overall training will be slower since we will be over-fitting for each batch and the network weights will have go under many updates when exposed to a new batch of data; on the other hand, if we do a single epoch per batch, we will be wasting useful information that could have been used to update the weights. This trade-off is specially relevant in our case since the generation of synthetic data from our generative model is relatively expensive. Research on how to address this issue empirically is limited, therefore we fine-tuned the ratio hyper-parameter by trial-and-error. This process on continual generation of new data results in a sawtooth shape of the training curve as can be seen in Fig. 8.5.

The rest of the training algorithm follows the backpropagation algorithm introduced by Rumelhart et al. in 1986 [113]. We run a number of simulations of biological neural populations to generate a batch of activity-simulation pairs, we then feed the activity to the DL-model and compute the error between the model output and the ground truth from the simulated data. Next, we update the DL-model weights following the computed gradient direction. This process is repeated until the error can not be further reduced or the cluster administrator cancels our job. The simulations of biological neural populations are implemented in parallel (CPU) using Python multiprocessing library, and the DL-model is trained in parallel using GPU accelerators [114] using Pytorch library. The algorithm is described as pseudocode in Algorithm 1.

Algorithm 1: Back propagation with unlimited synthetic data

Input: Generative model $\mathcal{G}$, Neural network model Π, training hyper-parameters Ω, a loss function $\mathcal{L}$
Output: Optimal neural network weights $\mathcal{W}$

```
Initialise network's $\mathcal{W}$ randomly;
while $Loss\ \mathcal{L} > target\ Loss$ do
    Generate a batch of N samples pairs $(x_n, y_n)$ from $\mathcal{G}$ ;          // Parallelized on CPU
    for $epoch = 1$ to $epochs\ number$ do
        for $i = 1$ to $N$ do                                                    // Parallelized on GPU
            Propagate the input $x_n$ through the network to compute the outputs $(\hat{y}_1, \ldots, \hat{y}_n)$;
        end
        Compute the loss $\mathcal{L}(y_n, \hat{y}_n)$ and its gradient $\frac{\partial \mathcal{L}_n}{\partial w_i}$ with respect to each weight $w_i$;
        Update the network weights $\mathcal{W}$ following the computed gradient $\frac{\partial \mathcal{L}_n}{\partial w_i}$;
        if $Loss\ \mathcal{L}$ has plateaued then
            Decay learning rate;
        end
    end
end
```

[4]An epoch refers to the moment where the training model has seen all the samples in the dataset once.

5.4.1.2 DL-model architectures

We have employed two different deep learning architectures, convolutional and a LSTM, the parameter details are shown in Fig. 5.4. Our experiments show that both architectures result in a similar performance (see appendix), however LSTM are considerably slower to train (see training curves in appendix Fig. 8.6), therefore, convolutional networks the preferred model choice to extract information from our time-series data. Like it is often the case in deep-learning, the number of parameters of our models is very large compared to traditional decoders such as Kalman filters. The number of parameters of the CNN depends on both the number of size of the population whose activity is being decoded as well as the length of the time-series; in our case the number of trained parameters range from 100 thousand to 10 millions. The number of parameters of the LSTM architecture only depends on the number of features –ie. the population size–, but not on the sequence length; for instance the LSTM model that we report in Fig. 8.6 has 593 400 trainable parameters.

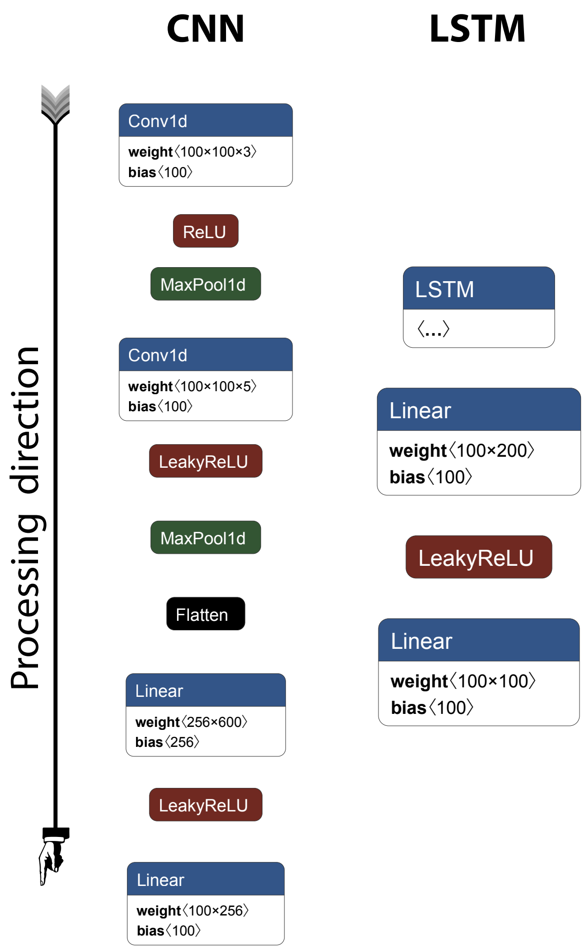


Figure 5.4: Details of the neural network architecture used as inverse models. **A**: The convolutional architecture consists of two 1D-convolutional layers followed by two linear layers. Here the `CNN1D_medium` model is displayed, which has two linear layers on top of the convolutional ones. **B**: The recurrent architecture has a Long short-term memory (LSTM) module with two hidden layers followed by two linear layers. In both architecture the linear layers are followed by leaky-ReLU activations.

5.4.1.3 Random-connectivity population

Rate population

In order to demonstrate our inverse model, we first simulate a population of 100 rate neurons with a fixed random connectivity matrix. The resulting oscillatory activity of each of the neurons are shown in Fig. 5.5.

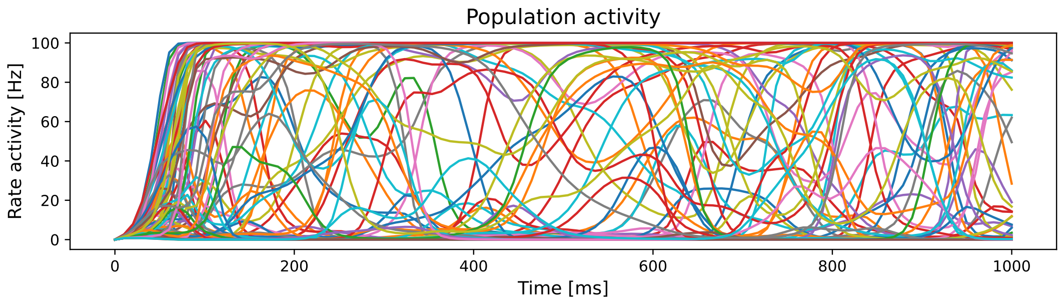
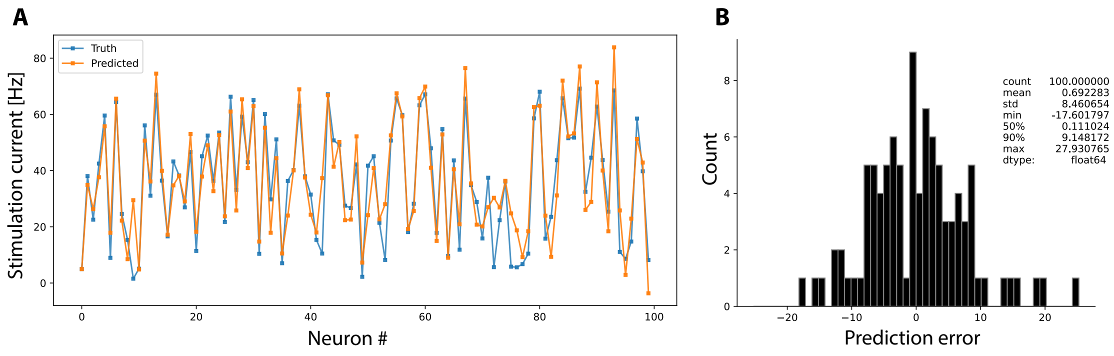


Figure 5.5: Example of the activity generated by a population of 100 rate-neurons with a random connectivity during 1000ms. We can see the fast oscillatory activity of each of 100 rate neurons. Although the stimulation information is encoded in these signals –as our inverse model shows–, it can not be easily be inferred *de visu*.

Consequently, the activity and the corresponding random stimulation that elicited it are used to train the inverse model. That is, we are trying to find the function f that maps the population activity of during T time-steps onto the set of stimulations driving each of the N rate neuron: $f\colon N \times R^t \mapsto R^N$. The results of comparing the predicted stimulations vs the original ones can are shown in Fig. 5.6. As we can see, the DL-model is capable of decoding the stimulation based on the population activity.



```
count     100.000000
mean        0.692283
std         8.460654
min       -17.601797
50%         0.111024
90%         9.148172
max        27.930765
dtype:       float64
```

Figure 5.6: **A**: Comparison between original stimulation and the prediction by the inverse model with the rate-neurons activity as input. **B**: Prediction errors histogram and error's statistics.

Spiking population

Unlike populations of rate-neurons, the activity of spiking networks is not characterised by continuous oscillatory signals, but rather, by a series of binary events in the form of spikes. As we saw in the background chapter 2.2.4.1, the activity of spiking neurons is visualised with a *raster plot* which displays the spiking times of each neuron. To demonstrate that our inverse DL-model can also operate with spiking data, we simulate a population of 100 LIF neurons with a fixed random connectivity. The resulting activity is shown in Fig. 5.7.

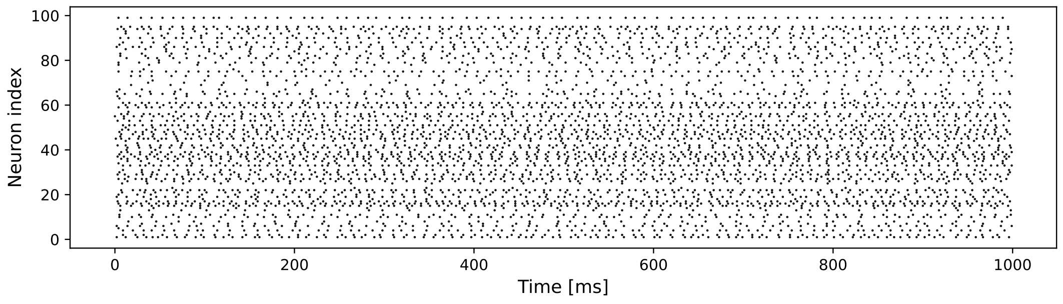
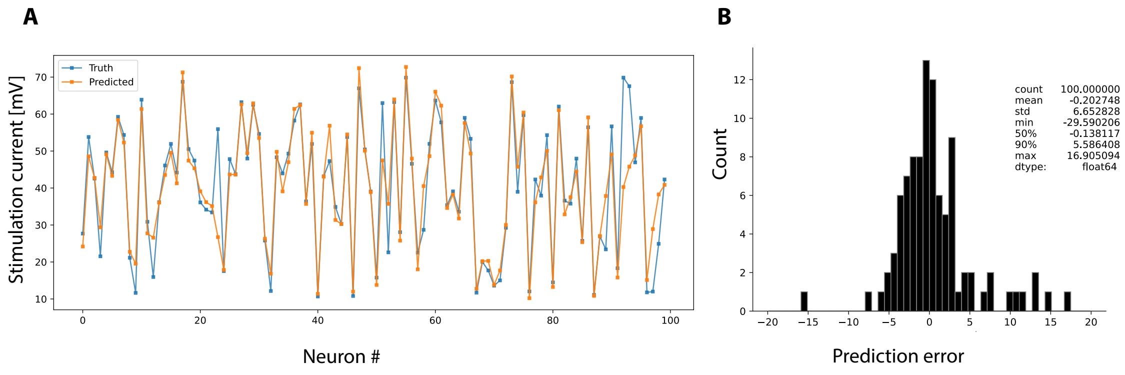


Figure 5.7: Raster plot with the activity of a balanced population of 100 LIF neurons. Each point represents the firing of one neuron.

We then use the generated population activity –and the corresponding random stimulation used to elicited it– to train the inverse DL-model. Now we are operating on a discrete input space, hence, we are trying to find the function f that maps the population activity of during T time-steps into the set of stimulations driving each of the N rate neuron: $f\colon N \times \{0,1\}^t \mapsto R^N$. The results comparing the predicted stimulations vs the original ones are shown in Fig. 5.8. As we can see, the DL-model is indeed capable of decoding the stimulation based on the population activity.



```
count     100.000000
mean       -0.202748
std         6.652828
min       -29.590206
50%        -0.138117
90%         5.586408
max        16.905094
dtype:       float64
```

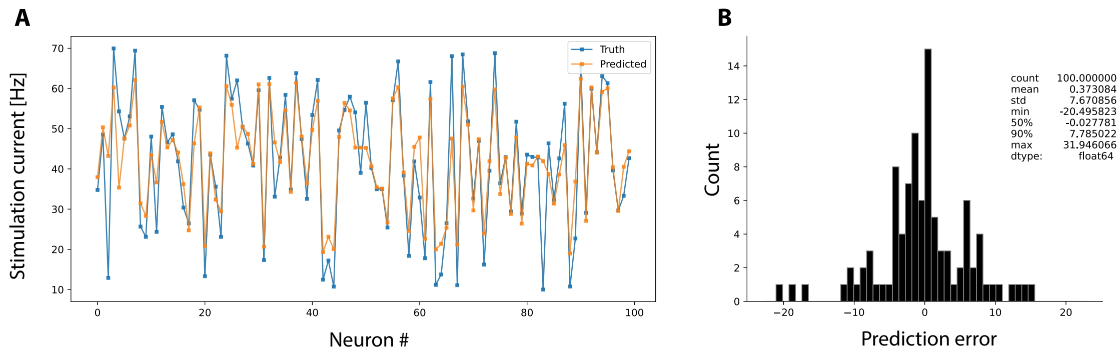
Figure 5.8: **A**: Comparison between original stimulation and the prediction by the inverse model with the spiking times as input. **B**: Prediction errors histogram and error's statistics.

5.4.1.4 Connectivity-agnostic inverse model

The results we have just presented for rate and spiking populations show that the inverse DL-model is capable of decoding the input stimulation for a randomly connected population of neurons. The connectivity of these populations –although random– is fixed during training. The previous models will not be capable of decoding the stimulation if presented with the activity of a neural population with a different connectivity than the one seen during training.

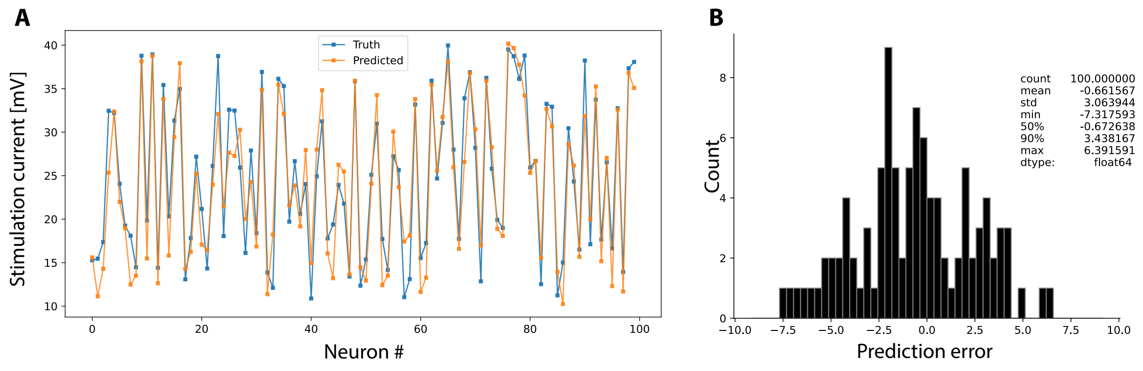
A natural extension of the previous result is to explore whether we could train a single inverse model capable of decoding the stimulation to a neural population whose connectivity is –not only random– but different at each simulation. Lo and behold, we show that it is indeed possible to build such models, which we name *connectivity-agnostic* inverse models.

In Fig. 5.9 and Fig. 5.10 we show that the inverse model is capable of decoding the input stimulus for a rate and spiking population whose connectivities have not been seen by the inverse model during training.



```
count    100.000000
mean       0.373084
std        7.670856
min      -20.495823
50%       -0.027781
90%        7.785022
max       31.946066
dtype:      float64
```

Figure 5.9: **A**: Activity generated by a population of 100 rate neurons with random connectivity at each instance. **B**: prediction errors histogram and error's statistics.

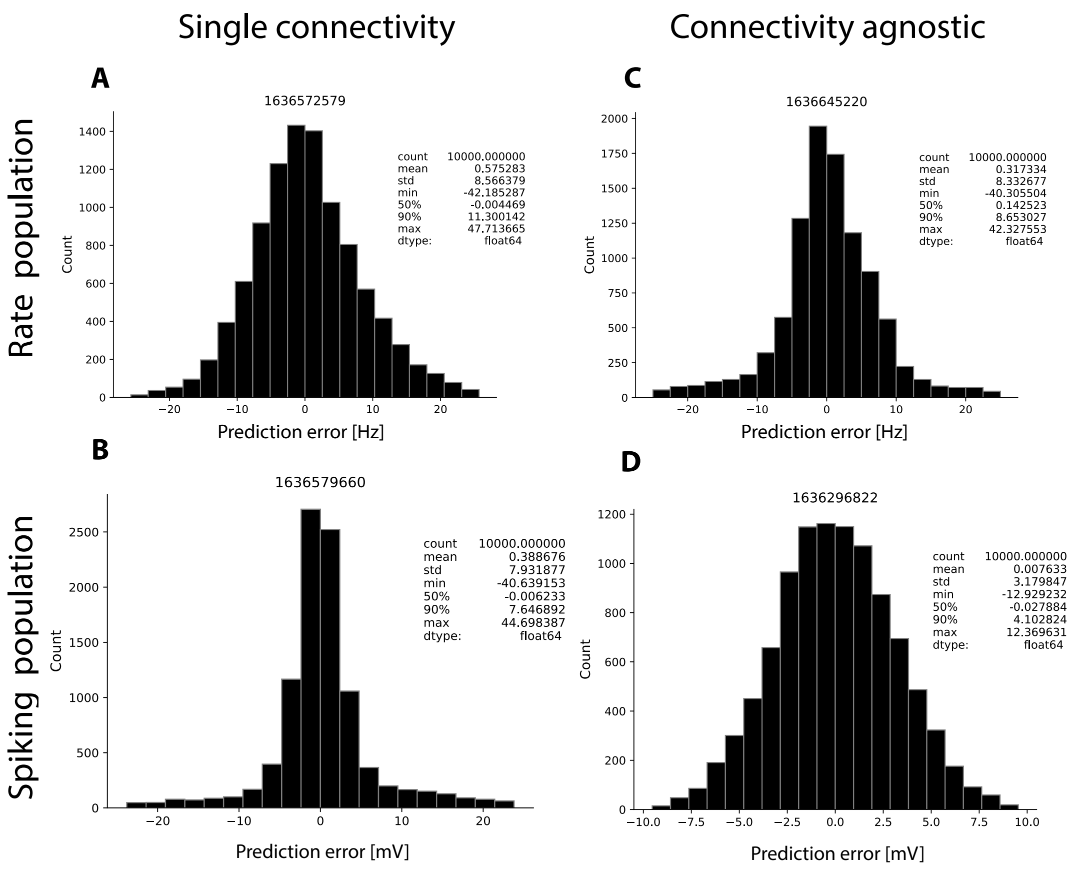


```
count    100.000000
mean      -0.661567
std        3.063944
min       -7.317593
50%       -0.672638
90%        3.438167
max        6.391591
dtype:      float64
```

Figure 5.10: **A**: Activity generated by a population of 100 spiking LIF neurons with random connectivity at each instance. **B**: prediction errors histogram and error's statistics.

The fact that the inverse model is capable of decoding the input stimulation from a network of hundred coupled nodes for any random connectivity is remarkable. This result suggests that DL-models could also be use to extract information –and control– other network systems that can be modelled as systems of coupled differential equations, without having to be concerned about the specific network connectivity. As it is usually the case, there is no free lunch in machine learning and training the connectivity-agnostic models requires much more data and training time (for instance, while the reported single-random-connectivity inverse model for the rate network required 15000 epochs to reach an accuracy of 0.57 ± 8.5 Hz, its connectivity-agnostic counterpart required 50000 epochs for an similar accuracy of 0.31 ± 8.3. Furthermore, a more thoroughly investigation on the limitations connectivity-agnostic models should be done. For instance, it is foreseeable that extremely-coupled networks will exhibit dynamical regimes for which decoding their parameters –for randomised connectivities– may not be feasible. The same difficulty might be faced with networks whose connectivity is sampled from non-Gaussian distributions.

Evaluation of models accuracy A more systematic evaluation of the accuracy of the trained models is presented in Fig. 5.11, where we show the prediction errors over one hundred evaluations for each of the models. The prediction error for the inverse model trained on the rate population with unique random connectivity **(A)** is 0.57 ± 8.5 Hz, the model trained on the spiking population with unique random connectivity **(B)** is 0.38 ± 7.9 mV, the model trained on the rate population with random connectivity at each instance **(C)** is 0.31 ± 8.3 Hz and the model trained on the spiking population with random connectivity at each instance **(D)** is 0.01 ± 3.1 mV. At first, it might be surprising that the connectivity-agnostic models **(C,D)**, which are capable of decoding the stimulation for any connectivity, have smaller prediction error than their single-connectivity versions **(A,B)**. There are two reasons for this: different population models and training times. For instance, in the case of spiking models **(B,D)**, the stimulation range of **(B)** is $[10, 70]\, mV$ while for **(D)** is $[10, 40]\, mV$, this suggests that the activity induced by the smaller stimulation range might be easier to decode. In the case of the rate models **(A,C)**, the stimulation range is the same $[10, 70]\, Hz$, but the connectivity-agnostic model has been trained for longer as shown in the training curves (appendix Fig. 8.5); the monotonically decreasing loss suggests that a longer training period would have resulted in a smaller prediction error.



```
count     10000.000000
mean          0.575283
std           8.566379
min         -42.185287
50%          -0.004469
90%          11.300142
max          47.713665
dtype:         float64
```

```
count     10000.000000
mean          0.317334
std           8.332677
min         -40.305504
50%           0.142523
90%           8.653027
max          42.327553
dtype:         float64
```

```
count     10000.000000
mean          0.388676
std           7.931877
min         -40.639153
50%          -0.006233
90%           7.646892
max          44.698387
dtype:         float64
```

```
count     10000.000000
mean          0.007633
std           3.179847
min         -12.929232
50%          -0.027884
90%           4.102824
max          12.369631
dtype:         float64
```

Figure 5.11: Histograms showing prediction errors and associated statistics of the inverse models for **A**: rate population with a single random connectivity, **B**: spiking population with a single random connectivity, **C**: rate population with random connectivity at each evaluation instance, and **D**: spiking population with random connectivity at each evaluation instance. The parameters of the population models are not the same for the different inverse models.

5.4.1.5 Capturing information vanishing in highly-coupled regimes

An interesting application of the inverse model is to investigate how long a given information remains in a signal. In our case, if we stimulate the neural population with an input current for limited duration –eg. 200ms–, and then remove the current, how long does information about the current –ie. the magnitude of the stimulation to each neuron– remain in the population activity signals? We can investigate this question by training the inverse model while masking out increasingly longer parts of the signal. We expect the model to be capable of decoding the stimulus when provided with full signal. If the model is capable of decoding the stimulation $U[10, 70]$Hz once interrupted (200ms onwards), we can safely conclude that the stimulation information is present in the signal. On the other hand, if we can not decode it, we can not conclude that the information is not there since it could simply mean that our inverse model was not capable of decoding it.

In Fig. 5.12 we show the result of the describe experiment performed on population of rate neurons. As we can see, the loss of the inverse model abruptly increases after 0.2, which corresponds to the duration of the stimulation signal, and quickly converges to a loss of ~ 14 which means that the model is incapable of decoding the individual stimulations to each neuron and instead is simply outputting the same mean stimulation value for all the neurons —the standard deviation of the stimulation distribution is $\sigma(U[10, 70]) \approx 17$.

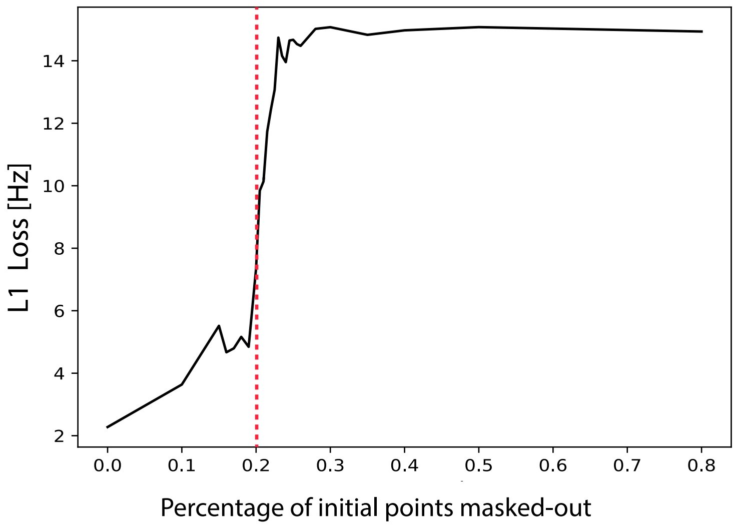


Figure 5.12: Test loss of the inverse model trained on the activity of the a 100-neuron rate population while masking out increasingly longer durations of the signal. Red dashed-line indicates when the stimulation ($\in U[10, 70]$Hz) stops. Each point indicates the final test loss (mean error per neuron of the test evaluation) of a model trained for 10000 epochs

As we have previously stated, the lack of success of the inverse model to decode the signal does not –necessarily– entitle the absence of the stimulus information in the signal. Nevertheless, this result demonstrates that inverse models can be used as a preliminary tool to interrogate signals about the information they hold before applying more rigorous analyses.

Externally-driven vs self-sustained activity

In section 3.3.1.2 we explored the stability and sensitivity to perturbations of rate populations, and empirically showed that networks with an spectral radius $\rho > 1$ display unstable and chaotic behaviour. Therefore, in the case of highly coupled networks, we expect the behaviour of the network being driven by self-sustained activity rather than by the input current used to stimulate the population. We can use our inverse models to empirically test this idea. To do so, we run an experiment where we train the same inverse model on the activity generated by a set of rate populations, each of which has a different spectral radius. The populations receive a constant stimulation $\in U[10, 70]$Hz over the whole duration of the simulation.

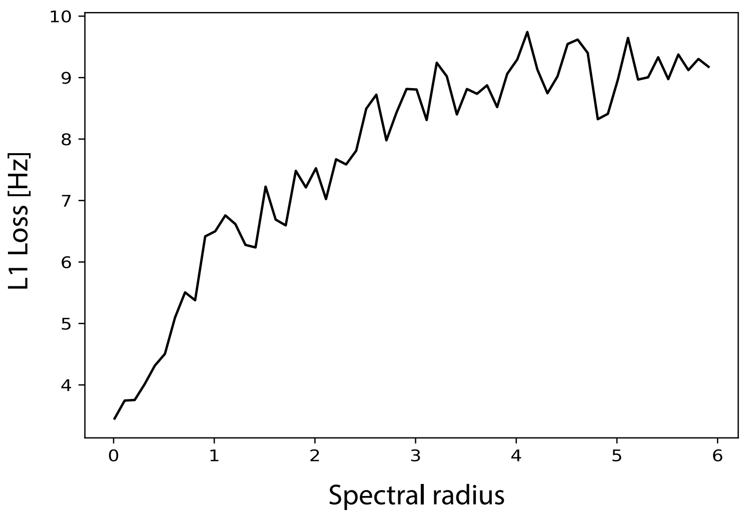


Figure 5.13: Test loss of the inverse model trained on the activity of a 100-neurons rate population with connectivity with different spectral radius. Each point indicates the final test loss (mean error per neuron of the test evaluation) of a model trained for 10000 epochs.

Similarly to the approach of using the decoding error of the trained inverse model as a way of probing for the presence of information in the signal, we can interpret Fig. 5.12 as evidencing that the dynamics of neural populations with spectral radius > 1 are dominated by their self-sustained activity. This approach that we have demonstrated here can naturally be used to study the dynamics of any network model –not just neural populations.

5.5 Manifold induction

Motivated by the prominence of manifolds in recent neuroscience research, we have introduced the notion of *manifold induction*[5] in Section 4.4. The approach is analogous as the one used to decode the input stimulus, but instead of providing the inverse model with the raw activity of the simulated neural populations, the inverse model is fed with the dimensionality-reduced representation of the activity –ie. its activity manifold.

To demonstrate the approach we simulate a population of 1000 rate-neurons for which, unlike in the previous case, we use a single stimulation vector such that the population generates an unique pattern of activity –and hence, an unique manifold– see Fig. 5.15 (upper). The raw activity is then mapped onto a lower dimensional basis with PCA to obtain its corresponding manifold, see Fig. 5.16 (left); as we saw in section 4.3, PCA already captures the rotational dynamics of the activity, therefore we restrict ourselves to this dimensionality reduction technique and leave non-linear ones such as Kernel PCA or tSNE for future work. Subsequently, the inverse-DL model is fed with the obtained manifold and trained to decode the original stimulation vector; since we are trying to predict a single stimulation, the inverse model can easily find a mapping from manifold representation to the stimulation, see Fig. 5.17. Finally, the predicted stimulation is used as input to a new simulation of a rate population whose activity –Fig. 5.15 (below)– is then used to compute its low-dimensional representation with PCA; this the final induced manifold shown in Fig. 5.16 (right). In the next chapter 6 we will discuss the applications and value of being able to induce specific activity manifolds in neuronal populations.

[5]Which could also be named *deep inverse optimal control with dimensionality reduction.*

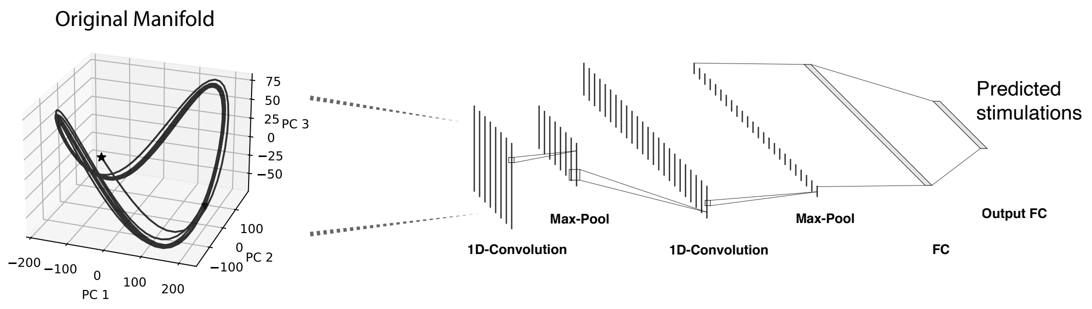


Figure 5.14: Diagram of the inverse deep-learning model. The dimensionality reduced representation of the population activity is fed onto the inversing network whose outputs is the stimulation that will induce that activity pattern.

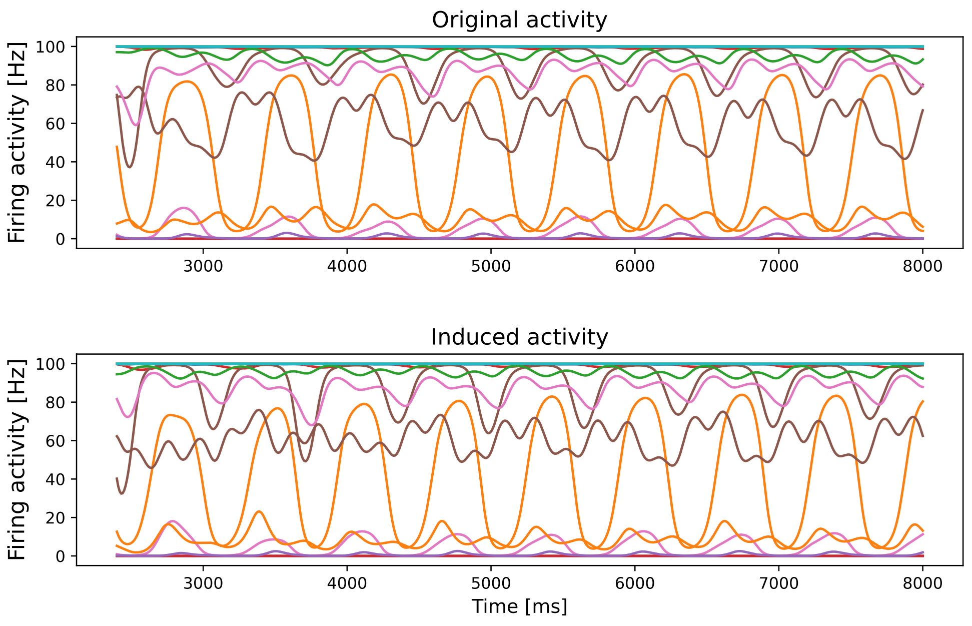


Figure 5.15: **Above**: Activity of a subset of 20 neurons generated by a population of 1000 rate-neurons. **Below**: Activity induced by simulating the same population with the decoded stimulation obtained from the inverse model. The initial 2.5s of the simulations were removed to avoid the initial transient phase.

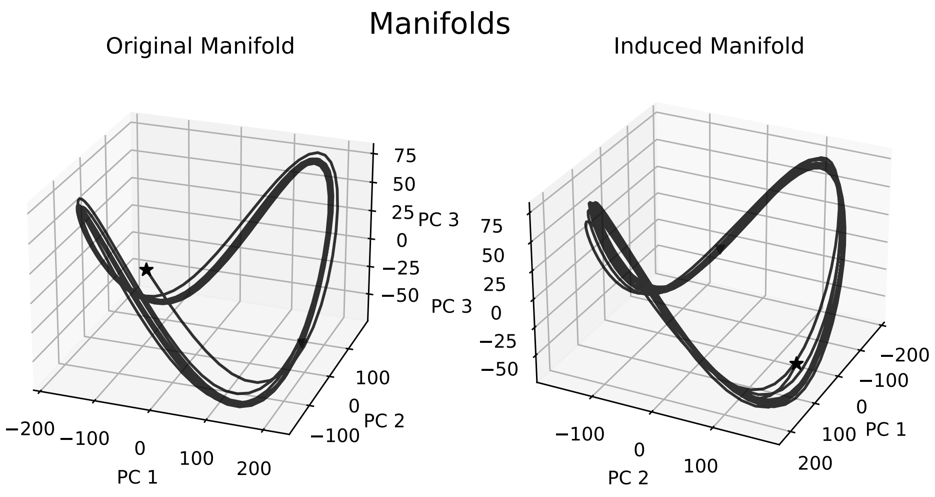
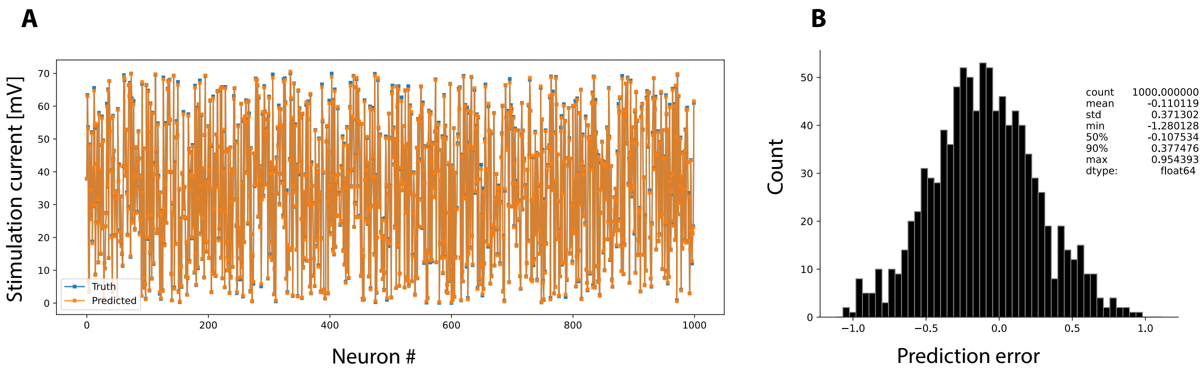


Figure 5.16: **Left**: Original manifold obtained by computing PCA from the raw activity shown in Fig. above 5.15. **Right**: Manifold resulting of generating a simulation with the predicted stimulus obtained from the inverse model. Notice that although the geometry of both manifolds is fairly similar, it is not identical and there is a rotation offset between both; this is due to the small prediction errors of the DL-model with respect to the original stimulation.



```
count     1000.000000
mean        -0.110119
std          0.371302
min         -1.280128
50%         -0.107534
90%          0.377476
max          0.954393
dtype:        float64
```

Figure 5.17: **A**: Comparison between original stimulation and the prediction by the inverse model with the manifold as input. **B**: Prediction errors histogram and error's statistics. Notice that in this set-up the discrepancies in **A** between the original and predicted stimulations are hardly visible since the errors of the substantially smaller (error standard deviation 0.37 mV) since our goal here is to overfit to a specific manifold.

Chapter 6

Inverse models for neuroprosthetics & other applications

Recent developments in multi-electrode neural probes –and less-invasive perturbation techniques such as optogenetics– have resulted in a explosion of the amount of neural data generated. However, the field is still adapting to this new data-driven era, and it is still unclear how to best make use of this data abundance. In this chapter we propose a protocol to build neuroprosthetic models making use of our ML-inverse models, and suggest other applications to utilize inverse models –as a research tool– to exploit the new data post-scarcity paradigm in neuroscience.

6.1 Inverse control for neuroprosthetics

Neural prostheses are technological devices that can replace –or enhance– motor, sensory or cognitive functions by building an active bridge between neural activity and sensory information or motor action. An example of widespread used neuroprosthetic devices are cochlear implants, a type of auditory implant which transform sound into electric signals which stimulate the cochlear nerve. The development of functional neural prostheses has strong clinical potential on subjects with motor or sensory impairments.

In the case of visual prostheses[1], the inverse problem consists in finding how to stimulate the visual cortex such that the induced neural activity elicits a faithful perception of the original image. As the reader will have noticed, this problem is largely analogous to the decoding problem we explored in Chapter 5.

Building on the demonstrated capability of ML-model to solve inverse problems on neural activity, we have devised a experimental protocol to build neuroprosthetic models potentially capable of bidirectionally translate neural activity to sensory or motor control using inverse models. The protocol consists on using a two close-loop optimisation set-up to train two inverse models. A first one that maps the to neural activity to cross-modality stimulation –eg. optogenetics–. And a second one which maps motor behaviour –or sensory input– to neural activity. The protocol is described in detail in Protocol 2.

[1]Visual prosthesis have seen weaker progress, partly because of the higher invasiveness of implanting electrodes on the visual cortex, and partly because of the higher information bandwidth of the human visual system.

```
Algorithm 2: End-to-end training of neuroprosthetic models
Set-up: Optogenetics stimulation, multi-electrode implant
1  Prepare the optogenetics and implant multi-electrode on a organoid or animal spinal cord;
Step: 1: Train inverse model to map neural-activity to stimulation
2  while neural activity loss > target neural activity loss do
3      Generate N random stimulation pulses;
4      Stimulate the tissue and record the elicit response;
5      Train inverse model on the generated data;
6  end
Step: 2: Train inverse model to map motor-behaviour to inverse model
7  while motor activity loss > target motor activity loss do
8      Record the neural activity from the animal spinal cord while performing a diversified set of
         motor tasks;
9      Map the motor behaviour into a continual space representation, for instance, a joint coordinate
         systems for motor movement;
10     Train an inverse model to find the neural activity that correlated to each movement;
11     Use the trained model (step 2) to find the pattern of neural activity that we know to be
         associated with the motor behaviour we want to trigger [a];
12     Use the inverse model trained in step 1 to generate the stimulation needed to induce the target
         activity pattern generated in line 11;
13     Compute the loss between elicit motor response and target movement;
14 end
```

[a]In the case of subjects with spinal cord injuries, the volitional intention to initiate the movement could be for instance be decoded from the motor cortex with a ECoG arrays.

Notice that the proposed protocol does not delve into understanding the neural code. Instead it builds a black-box model which blindly tells us how to stimulate the nerve system –eg. spinal cord, visual cortex, etc.– in order to induce a specific motor or sensory response. One of the strength of this protocol is its cross-modality. The sequential data used to characterise the neural activity can take any modal form –electrode recordings, calcium imaging, etc– and does not need to be of the same modality as the stimulation. Therefore, the devised protocol can be readily be used with any of the techniques available to experimentally characterise and stimulate neural activity.

6.2 Other applications of inverse models

Manifold hypothesis testing As we have discussed in Chapter 4.1.2, a central problem in neuroscience to explain how low-dimensional behaviours –such as movement or decision making– relates to the high-dimensional neural activity observed in the brain. An emerging conjecture is the *manifold hypothesis* which postulates that high-dimensional data tend to lie in the vicinity of a low-dimensional manifold. In neuroscience, this is used a as way to bridge the dimensional gap between behaviour and neural activity. However, it is yet unclear to which extent these manifolds –and their geometry– convey information about the behaviour they describe.

In Section 5.5, we have shown that we can use inverse models to induce specific activity manifolds in neural populations. Mutatis mutandis, the same models could be applied to *in vivo* neural populations and test whether inducing a specific manifold also induces its associated behaviour.

Inversing models as information distillers Last but not least, as we showed in Section 5.4.1.5, inverse models can be use to determine whether a given information is present on a signal. This constitutes a valuable tool to interrogate neural activity in order to develop our understanding of the neural code.

Chapter 7

Future Work

In this chapter we provide with an outlook of potential future work.

7.1 Neural networks beyond convolution and recurrent architectures

In this project we have shown that convolutional and recurrent architectures can be used to build inverse models. However, the deep-learning zoo is inhabited by many other architectures which have properties potentially beneficial for an inverse model. Let use mention two architectures:

- **Echo state networks (ESN)** are a type of *reservoir computing* architecture in which a pool of sparsely connected artificial neurons is placed between the input and output layers. Similarly to kernel methods, this recurrent reservoir acts as a high-dimensional space onto which the activations from the input layer are projected, in order to find representations that can be linearly separated by the output layer. Unlike with RNNs, the recurrent connections between the neurons in the reservoir are not trained but rather randomly initialised and fixed during the training of the input and output layers –making the training phase fast. ESN have been shown to be suitable for working with time-series data [115], [116].
- **Attention-based networks** such as transformers, tackle the difficulty of finding long-term dependencies in lengthy sequences of data by implementing a trainable attention mechanism, which guides the network *attention* to the relevant part of the input sequence. Transformers are the current state-of-the-art in dealing with natural language sequences, and could be particularly suitable for the type of sequential-data generate by neural populations.
- **Beyond deep-learning** Although dull, there is life in ML beyond deep-learning, and less buzzy techniques such as Bayesian methods or ensemble methods (random forest, boosting methods, etc.) are good candidates as inverse models.

7.2 Developmental hypothesis of spinal cord circuitry

As we discussed when we introduced our spiking model of the spinal cord circuitry in section 3.3.2.3, the model can exhibit several computation regimes by virtue of only adjusting their connectivity strength. This property of the model can used to formulate a hypothesis of developmental origin of the spinal cord circuit. The two computational regimes we have are: *the forward regime* which emerges from low connectivity strengths between the neurons, and is characterised by forwarding information without modifying. The second one is the *processing regime*, which results from a highly coupled connectivity and which significantly transform the signals it receives. Therefore, a potential mechanism to explain how these circuits develop could take the form of an activity-dependent plastic mechanism –eg. spike-timing-dependent plasticity (STDP)– during the early phases of motor development in animals. In this hypothesis, the spinal cord circuitry would be initially undifferentiated, all populations having exclusively low-coupling connections hence resulting in information only being feed-forwarded though through the spinal cord in neonates. Subsequently, based on the incoming projections of motor units, some populations of neurons would strengthen their connections through STDP, leading to the specialization –or development– of a particular computational function. This hypothesis could be tested by monitoring the evolution of the electrophysiological representations of the spinal cord activity from neonates until they reach maturity.

7.3 Topological analysis as modelling tool

In Chapter 4, we discussed the emerging approach in neuroscience of seeking to understand the computations performed by neural populations from a dynamical systems point of view. Specifically, it is conjectured that the geometry of dimensionality-reduced representations of neural activity can characterise the computation being performed. Most of current techniques focus on describing the geometry of the manifolds, however topological analysis also constitutes a promising approach to characterize manifolds. For instance, non-linear dimensionality techniques have been combined with the persistent homology method to compute the Betti numbers of neural activity patterns, and build low dimensional representations that accurately capture the relevant dimensions of animal behaviour [117].

Topological characterisation approaches could be combined with inverse models. Namely, rather than inducing a manifold with a specific geometry –as we showed in Section 5.5–, the inverse model could be trained to induce an activity manifold with certain topological properties. By doing so, the hypotheses regarding the relation between neural manifolds topologies and their associated computations could be tested experimentally.

7.4 Fully differentiable manifold induction models

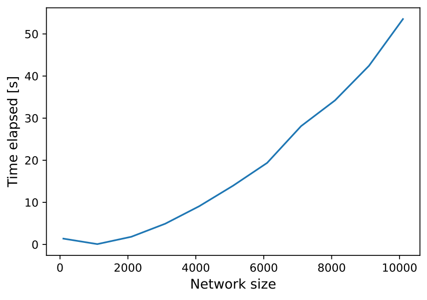
Our code to induce activity manifolds on neural populations works by taking the target manifold and finding the stimulation that evoke the underlying activity pattern that generated the manifold. However, in its current form, the resulting inverse models only works with a single manifold per trained model. In order to generalise it such that a single inverse model can take any manifold and find a valid stimulation –which needs not be the original one–, the model would need to be either trained in a cyclic manner using a black-box method or be fully-differentiable. That is, an inverse model that would take in the target manifold, output a stimulation, run a population simulation with that predicted stimulation, compute the manifold from the resulting activity, and compare the loss between the original and the final manifold. During our project, we unsuccessfully tried to implement such a model. We implemented a cyclic model trained with an evolutionary strategy known as CMA-ES, and fully differential model in Pytorch such that we could backpropagate end-to-end, including through the neural populations. Results were unimpressive, however the idea remains an interesting research direction.

Chapter 8

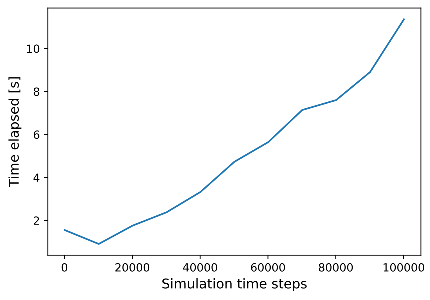
Appendix

8.0.1 Simulation performance

We run some benchmarks here to illustrate the performance of the model. The simulation is implemented on python and is based on the one used by Lindén et al. [1] with a some additional vectorisation of the operations in order to speed up the code and a few minor modifications. In Fig. **??** we show that the time needed to run the rate network simulation on a single-thread grows roughly linearly both with the network size and with the number of steps.



(a) $\propto$ Network Size



(b) $\propto$ Time-steps

Figure 8.1: Elapsed time (wall-clock) of the rate network model for different (a) network sizes and (b) simulation timesteps . The higher values of the initial first point of both graphs is an artifact produce by the Just-In-Time (JIT) compiler used by some functions which require to compile the code the first time the code is executed.

An benchmarking analysis of the simulation performance of Brian2 simulator can be found in [55].

8.0.2 Stability and chaotic behaviour

As we can see in the Fig. 8.2 we do observe exponential sensitivity to small perturbations. The exponential divergence is only observed over the first 200ms 8.2 (**B1**), over longer time the divergence appears to be linear; this due to the activity space being bounded, indeed, firing rates can only take values in the interval [0, 100]; this can be observed in Fig. 8.2 (**C**) where we plot the absolute difference between the last timestep of the activity between the network stimulated with a random stimulus and the resulting activity from the stimulating the network with same random stimulus plus a small Gaussian perturbation $\mathcal{N}(0, 0.01)$:

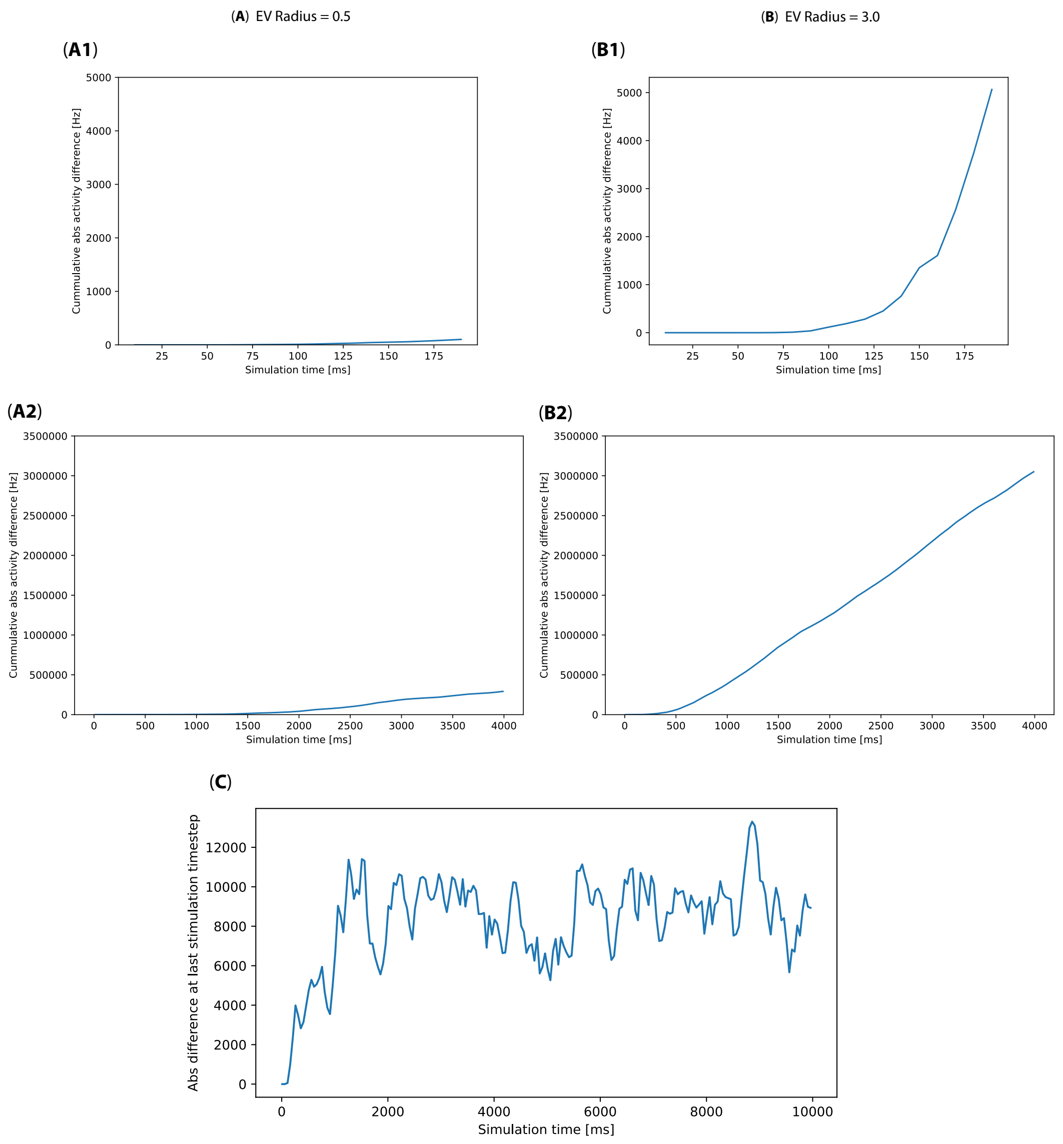


Figure 8.2: The network display exponential sensitivity to small perturbations for spectral radius $\mathcal{R} = 3.0$. In (**B1**) we can see an exponential divergence of the cumulative absolute difference between the unperturbed and perturbed network during the first 200ms. In (**B2**) we see that the initial exponential sensitivity disappears and become linear over longer simulation time. In (**C**) we can see that the difference between any two signals in firing rate space is bounded and plateaus after 2000ms. (**A1**) and (**A2**) show no exponential grow for spectral radius $\mathcal{R} = 0.5$.

Spectral radius effect We can investigate the effect of the spectral radius $\mathcal{R}$ by simulating unperturbed and perturbed network -$\mathcal{N}(0, 0.01)$- for different spectral radius $\mathcal{R}$ in the range [0, 3.0] and computing the cumulative difference between the networks activities. As we can see in Fig. 8.3, the divergence grows with $\mathcal{R}$, this is expected since higher $\mathcal{R}$ reflects higher coupling between the network neurons, hence increasing the divergence of the activity.

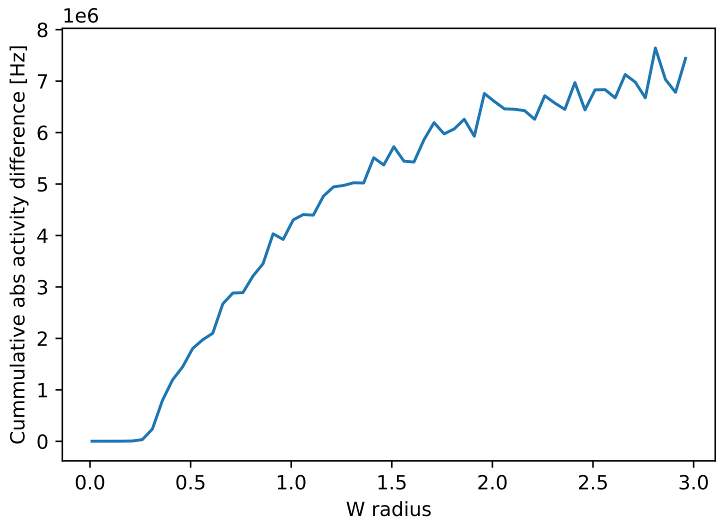


Figure 8.3: Cumulative absolute difference between the unperturbed and perturbed network during 2000ms simulation for different spectral radius $\mathcal{R}$. The plotted curved is the average of 5 simulations per $\mathcal{R}$.

Network size We now study the effect of the number of neurons in the network, -ie. the network size N- on the network sensitivity to small perturbations. To do so, we proceed in a similar manner as the the previous experiments but we now compute the ratio of the cumulative activity difference between the unperturbed and perturbed networks and its size N. As we can see in Fig. 8.4, the divergence plateaus for a network size of 200 neurons onwards. This suggests that a network size of 200 neurons is big enough for its activity to span over all the firing-rate space.

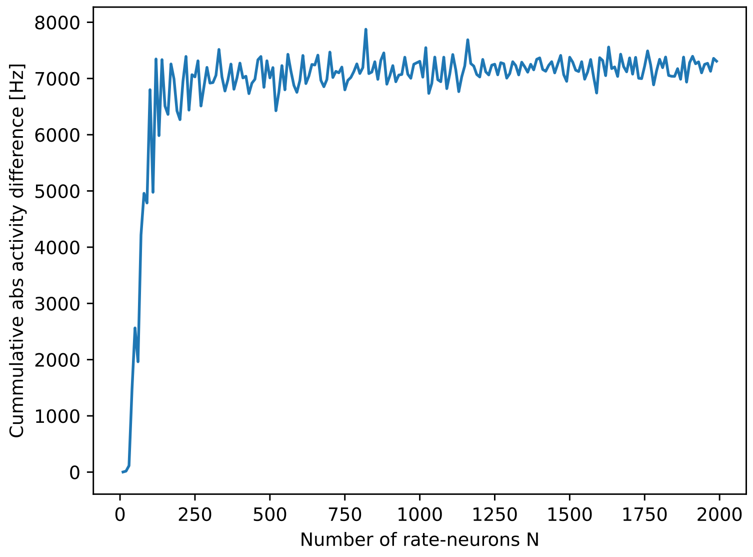


Figure 8.4: Ratio of the cumulative absolute difference between the unperturbed and perturbed network during 2000ms simulation and the network size for different network sizes N. The plotted curved is the average of 5 simulations per network size.

8.0.3 Training curves

Here we report the training curves for each of the presented models in Chapter 5. The loss indicates the mean error per neuron of the batch at each epoch. Notice that the number of epochs and population model parameters are different for each of the inverse models. Since we are using our proposed variation of backpropagation for synthetic data (Algorithm 1), the optimisation process of the model does not have a notion of validation loss since new data is periodically generated. The reported test loss is evaluated on a new simulation instance and with the model training features –such as dropout– deactivated.

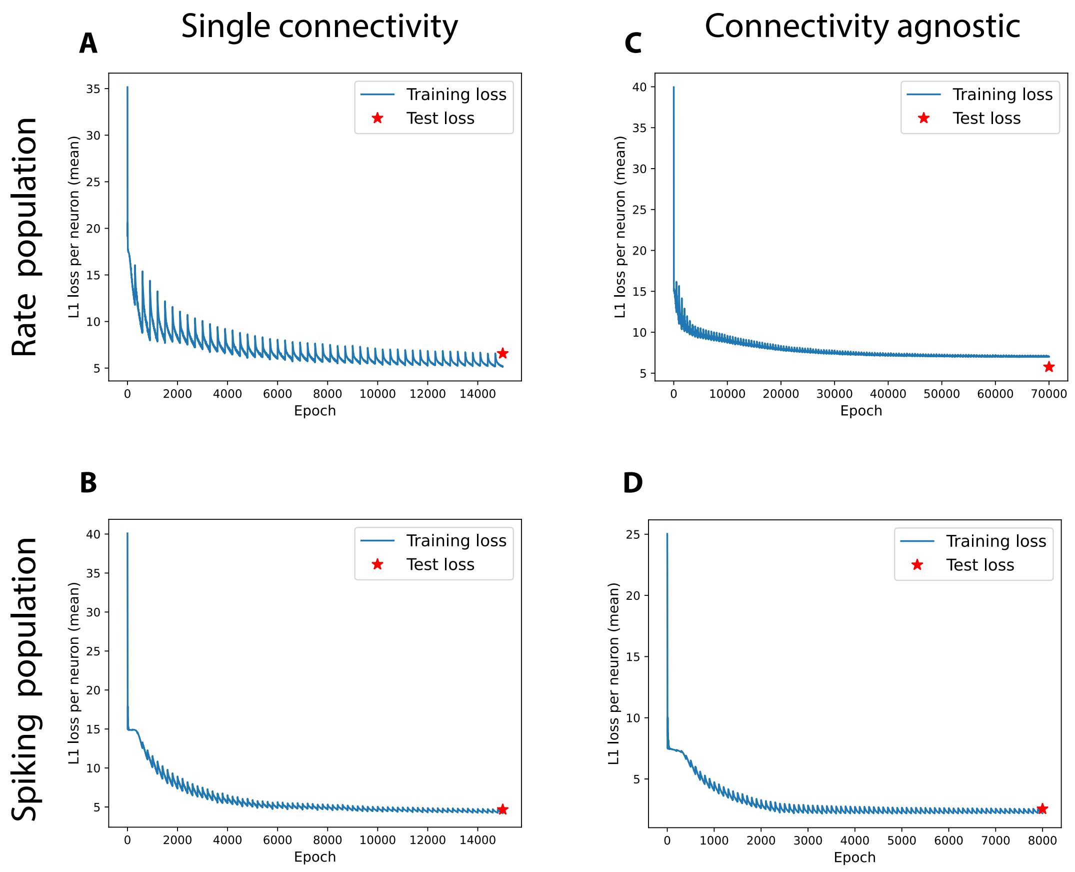


Figure 8.5: Training curves of the trained models shown in Fig. 5.11. Notice the sawtooth shape of the curves, each of the increases occurs when a new batch of data is generated.

8.0.4 CNN vs LSTM performance

Here we present a comparison of the training curves of two DL-inverse model trained on the same rate populations: one with convolutional architecture **A** and one Long short-term memory (LSTM) model **B**. The performance reached after 15000 epochs is similar although the convergence rate of the convectional model is faster. Furthermore, the training time of the LSTM was 25 minutes while the CNN only required 16 minutes. All things considered, the convolutional architecture seems a better choice for our data.

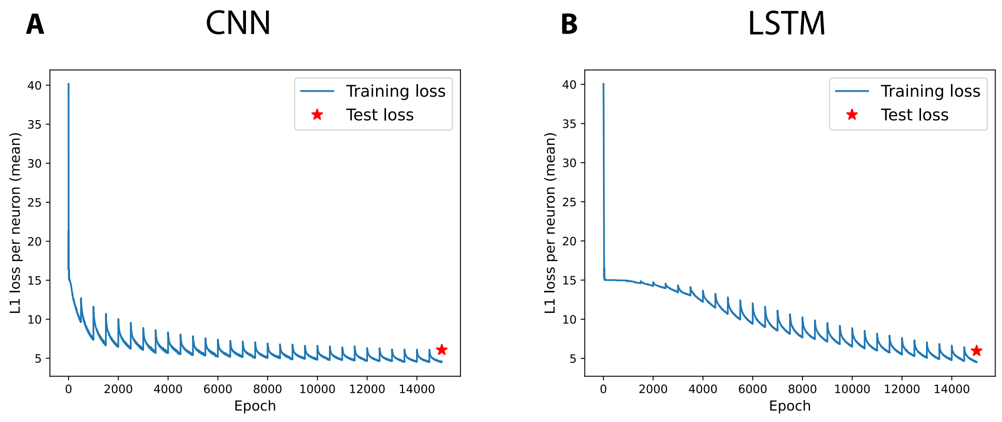


Figure 8.6: Training curves resulting from the training of a convolutional model **A** and a recurrent model **B** trained the same rate population.

Bibliography

[1] H. Lindén, P. C. Petersen, *et al.*, "Movement is governed by rotational population dynamics in spinal motor networks," Sep. 28, 2021, Company: Cold Spring Harbor Laboratory Distributor: Cold Spring Harbor Laboratory Label: Cold Spring Harbor Laboratory Section: New Results Type: article, p. 2021.08.31.458405. DOI: 10.1101/2021.08.31.458405.

[2] J. I. Glaser, A. S. Benjamin, *et al.*, "Machine learning for neural decoding," *arXiv:1708.00909 [cs, q-bio, stat]*, Sep. 19, 2019. arXiv: 1708.00909.

[3] J.-E. W. Skaar, A. J. Stasik, *et al.*, "Estimation of neural network model parameters from local field potentials (LFPs)," *PLOS Computational Biology*, vol. 16, no. 3, A. Roth, Ed., e1007725, Mar. 10, 2020. DOI: 10.1371/journal.pcbi.1007725.

[4] A. Dhawale and U. S. Bhalla, "The network and the synapse: 100 years after cajal," *HFSP Journal*, vol. 2, no. 1, pp. 12–16, Feb. 2008. DOI: 10.2976/1.2835214.

[5] M. Glickstein, "Golgi and cajal: The neuron doctrine and the 100th anniversary of the 1906 nobel prize," vol. 16, no. 5, p. 5,

[6] R. Yuste, "From the neuron doctrine to neural networks," *Nature Reviews Neuroscience*, vol. 16, no. 8, pp. 487–497, Aug. 2015, Number: 8 Publisher: Nature Publishing Group. DOI: 10.1038/nrn3962.

[7] S. Saxena and J. P. Cunningham, "Towards the neural population doctrine," *Current Opinion in Neurobiology*, vol. 55, pp. 103–111, Apr. 2019. DOI: 10.1016/j.conb.2019.02.002.

[8] E. D. Adrian and R. Matthews, "The action of light on the eye," *The Journal of Physiology*, vol. 63, no. 4, pp. 378–414, 1927, _eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1927.sp002410. DOI: https://doi.org/10.1113/jphysiol.1927.sp002410.

[9] F. Rieke, *Spikes: Exploring the Neural Code*. MIT Press, 1999, 418 pp., Google-Books-ID: W2dplgYWkkQC.

[10] P. Kraemer, *Synapse*, Jul. 1, 2020. DOI: 10.5281/zenodo.3925933.

[11] *Afterhyperpolarization*, in *Wikipedia*, Page Version ID: 989910924, Nov. 21, 2020.

[12] W. R. Taylor, S. He, *et al.*, "Dendritic computation of direction selectivity by retinal ganglion cells," *Science*, vol. 289, no. 5488, pp. 2347–2350, Sep. 29, 2000, Publisher: American Association for the Advancement of Science Section: Report. DOI: 10.1126/science.289.5488.2347.

[13] M. London and M. Häusser, "Dendritic computation," *Annual Review of Neuroscience*, vol. 28, no. 1, pp. 503–532, 2005, _eprint: https://doi.org/10.1146/annurev.neuro.28.061604.135703. DOI: 10.1146/annurev.neuro.28.061604.135703.

[14] L. L. Gollo, O. Kinouchi, *et al.*, "Single-neuron criticality optimizes analog dendritic computation," *Scientific Reports*, vol. 3, no. 1, p. 3222, Nov. 14, 2013, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/srep03222.

[15] C. Teeter, R. Iyer, *et al.*, "Generalized leaky integrate-and-fire models classify multiple neuron types," *Nature Communications*, vol. 9, no. 1, p. 709, Feb. 19, 2018, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/s41467-017-02717-4.

[16] E. De Schutter, Ed., *Computational modeling methods for neuroscientists*, Computational neuroscience, OCLC: ocn310075964, Cambridge, Mass: MIT Press, 2010, 419 pp.

[17] J. G. White, E. Southgate, *et al.*, "The structure of the nervous system of the nematode caenorhabditis elegans," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*,

vol. 314, no. 1165, pp. 1–340, Nov. 12, 1986, Publisher: Royal Society. DOI: 10.1098/rstb.1986.0056.

[18] Y. Tang, J. R. Nyengaard, *et al.*, "Total regional and global number of synapses in the human brain neocortex," *Synapse*, vol. 41, no. 3, pp. 258–273, 2001, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/syn.1083. DOI: https://doi.org/10.1002/syn.1083.

[19] T. P. Vogels, K. Rajan, *et al.*, "Neural network dynamics," *Annual Review of Neuroscience*, vol. 28, pp. 357–376, 2005. DOI: 10.1146/annurev.neuro.28.061604.135637.

[20] R. Brette, "Philosophy of the spike: Rate-based vs. spike-based theories of the brain," *Frontiers in Systems Neuroscience*, vol. 9, 2015, Publisher: Frontiers. DOI: 10.3389/fnsys.2015.00151.

[21] H. R. Wilson and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophysical Journal*, vol. 12, no. 1, pp. 1–24, Jan. 1, 1972. DOI: 10.1016/S0006-3495(72)86068-5.

[22] O. Shriki, D. Hansel, *et al.*, "Rate models for conductance-based cortical neuronal networks," *Neural Computation*, vol. 15, no. 8, pp. 1809–1841, Aug. 1, 2003. DOI: 10.1162/08997660360675053.

[23] Dayan, *Theoretical Neuroscience*. 2005.

[24] S. Denève and C. K. Machens, "Efficient codes and balanced networks," *Nature Neuroscience*, vol. 19, no. 3, pp. 375–382, Mar. 2016, Number: 3 Publisher: Nature Publishing Group. DOI: 10.1038/nn.4243.

[25] N. Brunel, "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons," p. 26, 2000.

[26] S. Ostojic, "Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons," *Nature Neuroscience*, vol. 17, no. 4, pp. 594–600, Apr. 2014. DOI: 10.1038/nn.3658.

[27] C. Holscher and M. Munk, "Information processing by neuronal populations," p. 485, 2009.

[28] C. G. Gross, "Genealogy of the grandmother cell," *The Neuroscientist*, vol. 8, no. 5, pp. 512–518, Oct. 1, 2002, Publisher: SAGE Publications Inc STM. DOI: 10.1177/107385802237175.

[29] H. S. Seung and H. Sompolinsky, "Simple models for reading neuronal population codes," *Proceedings of the National Academy of Sciences*, vol. 90, no. 22, pp. 10 749–10 753, Nov. 15, 1993, Publisher: National Academy of Sciences Section: Research Article. DOI: 10.1073/pnas.90.22.10749.

[30] N. Brunel and J.-P. Nadal, "Mutual information, fisher information and population coding," *Neural Computation*, vol. 10, no. 7, pp. 1731–1757, Oct. 1998, Publisher: Massachusetts Institute of Technology Press (MIT Press).

[31] A. Kaplan, *The Conduct of Inquiry: Methodology for Behavioral Science*. Chandler, 1964, 452 pp.

[32] A. H. Maslow, *The Psychology of Science: A Reconnaissance*. Harper & Row, 1966, 200 pp.

[33] E. P. Simoncelli and B. A. Olshausen, "Natural image statistics and neural representation," *Annual Review of Neuroscience*, vol. 24, no. 1, pp. 1193–1216, 2001. DOI: 10.1146/annurev.neuro.24.1.1193.

[34] J. J. Atick and A. N. Redlich, "Towards a theory of early visual processing," *Neural Computation*, vol. 2, no. 3, pp. 308–320, Sep. 1, 1990. DOI: 10.1162/neco.1990.2.3.308.

[35] H. B. Barlow, "Possible principles underlying the transformations of sensory messages," in *Sensory Communication*, W. A. Rosenblith, Ed., The MIT Press, Sep. 28, 2012, pp. 216–234. DOI: 10.7551/mitpress/9780262518420.003.0013.

[36] C. Stringer, M. Pachitariu, *et al.*, "High-dimensional geometry of population responses in visual cortex," *Nature*, vol. 571, no. 7765, pp. 361–365, Jul. 2019, Number: 7765 Publisher: Nature Publishing Group. DOI: 10.1038/s41586-019-1346-5.

[37] M. E. Rule, T. OLeary, *et al.*, "Causes and consequences of representational drift," *Current opinion in neurobiology*, vol. 58, pp. 141–147, Oct. 2019. DOI: 10.1016/j.conb.2019.08.005.

[38] G. Jékely, "Origin and early evolution of neural circuits for the control of ciliary locomotion," *Proceedings of the Royal Society B: Biological Sciences*, vol. 278, no. 1707, pp. 914–922, Mar. 22, 2011, Publisher: Royal Society. DOI: 10.1098/rspb.2010.2027.

[39] G. Leisman, A. A. Moustafa, *et al.*, "Thinking, walking, talking: Integratory motor and cognitive brain function," *Frontiers in Public Health*, vol. 4, May 25, 2016. DOI: 10.3389/fpubh.2016.00094.

[40] G. Fritsch and E. Hitzig, "Uber die elektrische erregbarkeit des grosshirns," *Arch, anat. Physiol. Wiss. Med.*, vol. 37, pp. 300–332, 1870.

[41] J. BAHNEY and C. S. VON BARTHELD, "The cellular composition and glia-neuron ratio in the spinal cord of a human and a non-human primate: Comparison with other species and brain regions," *Anatomical record (Hoboken, N.J. : 2007)*, vol. 301, no. 4, pp. 697–710, Apr. 2018. DOI: 10.1002/ar.23728.

[42] R. Bjugn and H. J. Gundersen, "Estimate of the total number of neurons and glial and endothelial cells in the rat spinal cord by means of the optical disector," *The Journal of Comparative Neurology*, vol. 328, no. 3, pp. 406–414, Feb. 15, 1993. DOI: 10.1002/cne.903280307.

[43] M. Wilson, *The central nervous control of flight in a locust.* 1961.

[44] D. Wilson and R. Wyman, "Motor output patterns during random and rhythmic stimulation of locust thoracic ganglia," *Biophysical Journal*, vol. 5, no. 2, pp. 121–143, Mar. 1965. DOI: 10.1016/S0006-3495(65)86706-6.

[45] S. Grillner and P. Wallén, "Central pattern generators for locomotion, with special reference to vertebrates," *Annual Review of Neuroscience*, vol. 8, no. 1, pp. 233–261, 1985, _eprint: https://doi.org/10.1146/annurev.ne.08.030185.001313. DOI: 10.1146/annurev.ne.08.030185.001313.

[46] E. Marder and D. Bucher, "Central pattern generators and the control of rhythmic movements," *Current Biology*, vol. 11, no. 23, R986–R996, Nov. 27, 2001. DOI: 10.1016/S0960-9822(01)00581-4.

[47] Y. I. Arshavsky, I. M. Gelfand, *et al.*, "Messages conveyed by spinocerebellar pathways during scratching in the cat. II. activity of neurons of the ventral spinocerebellar tract," *Brain Research*, vol. 151, no. 3, pp. 493–506, Aug. 11, 1978. DOI: 10.1016/0006-8993(78)91082-X.

[48] Y. Sqalli-Houssaini, J. R. Cazalets, *et al.*, "Oscillatory properties of the central pattern generator for locomotion in neonatal rats," *Journal of Neurophysiology*, vol. 70, no. 2, pp. 803–813, Aug. 1, 1993. DOI: 10.1152/jn.1993.70.2.803.

[49] C. Perret and J.-M. Cabelguen, "Main characteristics of the hindlimb locomotor cycle in the decorticate cat with special reference to bifunctional muscles," *Brain Research*, vol. 187, no. 2, pp. 333–352, Apr. 14, 1980. DOI: 10.1016/0006-8993(80)90207-3.

[50] P. S. G. Stein, "Central pattern generators in the turtle spinal cord: Selection among the forms of motor behaviors," *Journal of Neurophysiology*, vol. 119, no. 2, pp. 422–440, Feb. 1, 2018. DOI: 10.1152/jn.00602.2017.

[51] S. Grillner, "Interaction between central and peripheral mechanisms in the control of locomotion," in *Progress in Brain Research*, ser. Reflex Control of Posture And Movement, R. Granit and O. Pompeiano, Eds., vol. 50, Elsevier, Jan. 1, 1979, pp. 227–235. DOI: 10.1016/S0079-6123(08)60823-7.

[52] T. G. Brown, "On the nature of the fundamental activity of the nervous centres; together with an analysis of the conditioning of rhythmic activity in progression, and a theory of the evolution of function in the nervous system," *The Journal of Physiology*, vol. 48, no. 1, pp. 18–46, 1914, _eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1914.sp001646. DOI: https://doi.org/10.1113/jphysiol.1914.sp001646.

[53] C. S. Sherrington, "Remarks on some aspects of reflex inhibition," *Proceedings of the Royal Society of London. Series B, Containing Papers of a Biological Character*, vol. 97, no. 686, pp. 519–545, Apr. 1, 1925, Publisher: Royal Society. DOI: 10.1098/rspb.1925.0017.

[54] Kandel, *Principles of Neural Science*, 5th.

[55] M. Stimberg, R. Brette, *et al.*, "Brian 2, an intuitive and efficient neural simulator," *eLife*, vol. 8, e47314, Aug. 20, 2019. DOI: 10.7554/eLife.47314.

[56] G. Hennequin, T. P. Vogels, *et al.*, "Optimal control of transient dynamics in balanced networks supports generation of complex movements," *Neuron*, vol. 82, no. 6, pp. 1394–1406, Jun. 18, 2014. DOI: 10.1016/j.neuron.2014.04.045.

[57] J. P. Stroud, M. A. Porter, *et al.*, "Motor primitives in space and time via targeted gain modulation in cortical networks," *Nature Neuroscience*, vol. 21, no. 12, pp. 1774–1783, Dec. 2018, Number: 12 Publisher: Nature Publishing Group. DOI: 10.1038/s41593-018-0276-0.

[58] K. Rajan and L. F. Abbott, "Eigenvalue spectra of random matrices for neural networks," *Physical Review Letters*, vol. 97, no. 18, p. 188 104, Nov. 2, 2006. DOI: 10.1103/PhysRevLett.97.188104.

[59] R. M. May, "Will a large complex system be stable?" *Nature*, vol. 238, no. 5364, pp. 413–414, Aug. 1972, Bandiera_abtest: a Cg_type: Nature Research Journals Number: 5364 Primary_atype: Research Publisher: Nature Publishing Group. DOI: 10.1038/238413a0.

[60] P. Uhlhaas, G. Pipa, *et al.*, "Neural synchrony in cortical networks: History, concept and current status," *Frontiers in Integrative Neuroscience*, vol. 3, p. 17, 2009. DOI: 10.3389/neuro.07.017.2009.

[61] E. Baar, C. Baar-Erolu, *et al.*, "Chapter 2 - brain's alpha, beta, gamma, delta, and theta oscillations in neuropsychiatric diseases: Proposal for biomarker strategies," in *Supplements to Clinical Neurophysiology*, ser. Application of Brain Oscillations in Neuropsychiatric Diseases, E. Baar, C. Baar-Erolu, *et al.*, Eds., vol. 62, Elsevier, Jan. 1, 2013, pp. 19–54. DOI: 10.1016/B978-0-7020-5307-8.00002-8.

[62] B. Kriener, H. Enger, *et al.*, "Dynamics of self-sustained asynchronous-irregular activity in random networks of spiking neurons with strong synapses," *Frontiers in Computational Neuroscience*, vol. 8, p. 136, 2014. DOI: 10.3389/fncom.2014.00136.

[63] R. W. Berg, A. Willumsen, *et al.*, "When networks walk a fine line: Balance of excitation and inhibition in spinal motor circuits," *Current Opinion in Physiology*, Motor Control Systems, vol. 8, pp. 76–83, Apr. 1, 2019. DOI: 10.1016/j.cophys.2019.01.006.

[64] B. Riemann, "Grundlagen für eine allgemeine Theorie der Functionen einer veränderlichen complexen Grösse," *Zweiter unveränderter Abdruck, Göttinger*, p. 44, 1851.

[65] H. S. Seung and D. D. Lee, "The manifold ways of perception," *Science*, vol. 290, no. 5500, pp. 2268–2269, Dec. 22, 2000, Publisher: American Association for the Advancement of Science Section: Perspective. DOI: 10.1126/science.290.5500.2268.

[66] R. A. Peters, R. E. Bodenheimer, *et al.*, "Sensory-motor manifold structure induced by task outcome: Experiments with robonaut," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, ISSN: 2164-0580, Dec. 2006, pp. 484–489. DOI: 10.1109/ICHR.2006.321317.

[67] D. Derdikman and E. I. Moser, "A manifold of spatial maps in the brain," in *Space, Time and Number in the Brain*, Elsevier, 2011, pp. 41–57. DOI: 10.1016/B978-0-12-385948-8.00004-9.

[68] R. Huys, D. Perdikis, *et al.*, "Functional architectures and structured flows on manifolds: A dynamical framework for motor behavior," *Psychological Review*, vol. 121, no. 3, pp. 302–336, 2014, Place: US Publisher: American Psychological Association. DOI: 10.1037/a0037014.

[69] J. A. Gallego, M. G. Perich, *et al.*, "Neural manifolds for the control of movement," *Neuron*, vol. 94, no. 5, pp. 978–984, Jun. 7, 2017. DOI: 10.1016/j.neuron.2017.05.025.

[70] J. A. Gallego, M. G. Perich, *et al.*, "Cortical population activity within a preserved neural manifold underlies multiple motor behaviors," *Nature Communications*, vol. 9, no. 1, p. 4233, Oct. 12, 2018, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/s41467-018-06560-z.

[71] S. Vyas, M. D. Golub, *et al.*, "Computation through neural population dynamics," *Annual Review of Neuroscience*, vol. 43, no. 1, pp. 249–275, Jul. 8, 2020, Publisher: Annual Reviews. DOI: 10.1146/annurev-neuro-092619-094115.

[72] M. Jazayeri and S. Ostojic, "Interpreting neural computations by examining intrinsic and embedding dimensionality of neural activity," *arXiv:2107.04084 [q-bio]*, Jul. 8, 2021. arXiv: 2107.04084.

[73] J. B. Tenenbaum, V. d. Silva, *et al.*, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 22, 2000, Publisher: American Association for the Advancement of Science Section: Report. DOI: 10.1126/science.290.5500.2319.

[74] B. Mathew, "Charting a manifold," in *Neural Information Processing Systems*, 2002.

[75] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, Jun. 1, 2003. DOI: 10.1162/089976603321780317.

[76] H. Narayanan and S. Mitter, "Sample complexity of testing the manifold hypothesis," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'10, Red Hook, NY, USA: Curran Associates Inc., Dec. 6, 2010, pp. 1786–1794.

[77] C. Fefferman, S. Mitter, *et al.*, "Testing the manifold hypothesis," *Journal of the American Mathematical Society*, vol. 29, no. 4, pp. 983–1049, Feb. 9, 2016. DOI: 10.1090/jams/852.

[78] M. Jazayeri and A. Afraz, "Navigating the neural space in search of the neural code," *Neuron*, vol. 93, no. 5, pp. 1003–1014, Mar. 8, 2017, Publisher: Elsevier. DOI: 10.1016/j.neuron.2017.02.019.

[79] O. G. Sani, H. Abbaspourazad, *et al.*, "Modeling behaviorally relevant neural dynamics enabled by preferential subspace identification," *Nature Neuroscience*, vol. 24, no. 1, pp. 140–149, Jan. 2021, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/s41593-020-00733-0.

[80] P. T. Sadtler, K. M. Quick, *et al.*, "Neural constraints on learning," *Nature*, vol. 512, no. 7515, pp. 423–426, Aug. 2014, Number: 7515 Publisher: Nature Publishing Group. DOI: 10.1038/nature13665.

[81] J. D. Murray, A. Bernacchia, *et al.*, "Stable population coding for working memory coexists with heterogeneous neural dynamics in prefrontal cortex," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, no. 2, pp. 394–399, Jan. 10, 2017. DOI: 10.1073/pnas.1619449114.

[82] E. Wärnberg and A. Kumar, "Perturbing low dimensional activity manifolds in spiking neuronal networks," *PLOS Computational Biology*, vol. 15, no. 5, P. E. Latham, Ed., e1007074, May 31, 2019. DOI: 10.1371/journal.pcbi.1007074.

[83] J. A. Gallego, M. G. Perich, *et al.*, "Long-term stability of cortical population dynamics underlying consistent behavior," *Nature Neuroscience*, vol. 23, no. 2, pp. 260–270, Feb. 2020. DOI: 10.1038/s41593-019-0555-4.

[84] G. Hong and C. M. Lieber, "Novel electrode technologies for neural recordings," *Nature Reviews Neuroscience*, vol. 20, no. 6, pp. 330–345, Jun. 2019, Number: 6 Publisher: Nature Publishing Group. DOI: 10.1038/s41583-019-0140-6.

[85] N. A. Steinmetz, C. Aydin, *et al.*, "Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings," *Science*, vol. 372, no. 6539, Apr. 16, 2021, Publisher: American Association for the Advancement of Science Section: Research Article. DOI: 10.1126/science.abf4588.

[86] C. Stosiek, O. Garaschuk, *et al.*, "In vivo two-photon calcium imaging of neuronal networks," *Proceedings of the National Academy of Sciences*, vol. 100, no. 12, pp. 7319–7324, Jun. 10, 2003, Publisher: National Academy of Sciences Section: Biological Sciences. DOI: 10.1073/pnas.1232232100.

[87] J. T. Russell, "Imaging calcium signals in vivo: A powerful tool in physiology and pharmacology," *British Journal of Pharmacology*, vol. 163, no. 8, pp. 1605–1625, Aug. 2011. DOI: 10.1111/j.1476-5381.2010.00988.x.

[88] M. Jazayeri, *What could neural dynamics have to say about neural computation and do we know how to listen?* Dec. 4, 2020.

[89] (Apr. 7, 2021). "Geometrical thinking offers a window into computation," Simons Foundation, [Online]. Available: https://www.simonsfoundation.org/2021/04/07/geometrical-thinking-offers-a-window-into-computation/ (visited on 06/02/2021).

[90] S. Chung and L. F. Abbott, "Neural population geometry: An approach for understanding biological and artificial neural networks," *arXiv:2104.07059 [cs, q-bio]*, Apr. 14, 2021. arXiv: 2104.07059.

[91] S. Saxena, A. A. Russo, *et al.*, "Motor cortex activity across movement speeds is predicted by network-level strategies for generating muscle activity," *bioRxiv*, p. 2021.02.01.429168, Feb. 2, 2021, Publisher: Cold Spring Harbor Laboratory Section: New Results. DOI: 10.1101/2021.02.01.429168.

[92] J. Shlens, "A tutorial on principal component analysis," *arXiv:1404.1100 [cs, stat]*, Apr. 3, 2014. arXiv: 1404.1100.

[93] K. P. F.R.S, "LIII. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, Nov. 1, 1901. DOI: 10.1080/14786440109462720.

[94] I. T. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20 150 202, Apr. 13, 2016, Publisher: Royal Society. DOI: 10.1098/rsta.2015.0202.

[95] B. Schölkopf, A. Smola, *et al.*, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, Jul. 1, 1998. DOI: 10.1162/089976698300017467.

[96] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[97] "Embedding space," in *Encyclopedia of Biometrics*, S. Z. Li and A. Jain, Eds., Boston, MA: Springer US, 2009, pp. 259–259. DOI: 10.1007/978-0-387-73003-5_573.

[98] P. Poliar. (). "openTSNE: Extensible, parallel implementations of t-SNE openTSNE 0.3.13 documentation," [Online]. Available: https://opentsne.readthedocs.io/en/latest/ (visited on 05/16/2021).

[99] G. Lemaitre. (). "Sklearn.manifold.TSNE scikit-learn 0.24.2 documentation," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html (visited on 05/16/2021).

[100] D. Kobak and P. Berens, "The art of using t-SNE for single-cell transcriptomics," *Nature Communications*, vol. 10, no. 1, p. 5416, Nov. 28, 2019, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/s41467-019-13056-x.

[101] M. Wattenberg, F. Viégas, *et al.*, "How to use t-SNE effectively," *Distill*, vol. 1, no. 10, e2, Oct. 13, 2016. DOI: 10.23915/distill.00002.

[102] A. Narayan, B. Berger, *et al.*, "Density-preserving data visualization unveils dynamic patterns of single-cell transcriptomic variability," *bioRxiv*, p. 2020.05.12.077776, May 14, 2020, Publisher: Cold Spring Harbor Laboratory Section: New Results. DOI: 10.1101/2020.05.12.077776.

[103] L. McInnes, J. Healy, *et al.*, "UMAP: Uniform manifold approximation and projection for dimension reduction," *arXiv:1802.03426 [cs, stat]*, Sep. 17, 2020. arXiv: 1802.03426.

[104] C. Stringer, M. Pachitariu, *et al.*, "Spontaneous behaviors drive multidimensional, brainwide activity," *Science*, vol. 364, no. 6437, Apr. 19, 2019, Publisher: American Association for the Advancement of Science Section: Research Article. DOI: 10.1126/science.aav7893.

[105] D. Kobak and G. C. Linderman, "UMAP does not preserve global structure any better than t-SNE when using the same initialization," Bioinformatics, preprint, Dec. 19, 2019. DOI: 10.1101/2019.12.19.877522.

[106] A. A. Russo, R. Khajeh, *et al.*, "Neural trajectories in the supplementary motor area and motor cortex exhibit distinct geometries, compatible with different classes of computation," *Neuron*, vol. 107, no. 4, 745–758.e6, Aug. 19, 2020. DOI: 10.1016/j.neuron.2020.05.020.

[107] O. Yizhar, L. E. Fenno, *et al.*, "Optogenetics in neural systems," *Neuron*, vol. 71, no. 1, pp. 9–34, Jul. 14, 2011. DOI: 10.1016/j.neuron.2011.06.004.

[108] K. Deisseroth, "Optogenetics," *Nature Methods*, vol. 8, no. 1, pp. 26–29, Jan. 2011, Number: 1 Publisher: Nature Publishing Group. DOI: 10.1038/nmeth.f.324.

[109] A. Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, Jan. 2005. DOI: 10.1137/1.9780898717921.

[110] M. Bunge, "Inverse problems," *Foundations of Science*, vol. 24, no. 3, pp. 483–525, Sep. 1, 2019. DOI: 10.1007/s10699-018-09577-1.

[111] P. A. Valdes-Sosa, A. Roebroeck, *et al.*, "Effective connectivity: Influence, causality and biophysical modeling," *NeuroImage*, vol. 58, no. 2, pp. 339–361, Sep. 15, 2011. DOI: 10.1016/j.neuroimage.2011.03.058.

[112] K. He, X. Zhang, *et al.*, "Deep residual learning for image recognition," *arXiv:1512.03385 [cs]*, Dec. 10, 2015. arXiv: 1512.03385.

[113] D. E. Rumelhart, G. E. Hinton, *et al.*, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, Bandiera_abtest: a Cg_type: Nature Research Journals Number: 6088 Primary_atype: Research Publisher: Nature Publishing Group. DOI: 10.1038/323533a0.

[114] S. Hooker, "The hardware lottery," *arXiv:2009.06489 [cs]*, Sep. 21, 2020. arXiv: 2009.06489.

[115] C. Gallicchio and A. Micheli, "Deep echo state network (DeepESN): A brief survey," *arXiv:1712.04323 [cs, stat]*, Feb. 25, 2019. arXiv: 1712.04323.

[116] N. Heim and J. E. Avery, "Adaptive anomaly detection in chaotic time series with a spatially aware echo state network," *arXiv:1909.01709 [cs, stat]*, Sep. 2, 2019. arXiv: 1909.01709.

[117] R. J. Low, S. Lewallen, *et al.*, "Probing variability in a cognitive map using manifold inference from neural dynamics," Neuroscience, preprint, Sep. 16, 2018. DOI: 10.1101/418939.