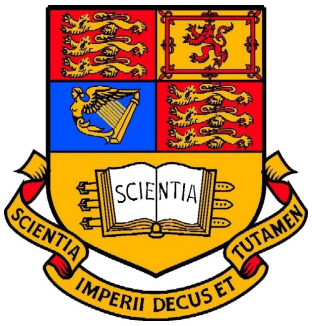
IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Structure and dynamics of large networks of interacting neurons

Author:
Alejandro Gilson Campillo
CID:
01112712

Supervisor:
Prof. Pier Luigi Dragotti
Second marker:
Dr. Wei Dai



A thesis submitted for the degree of

MEng Electrical and Electronic Engineering

June 19, 2019

Abstract

This project tries to understand how real biological neural networks are connected by treating the system as a diffusion process. By selecting the connectivity weights of each of the neurons that maximize the likelihood of some recorded spikes occurring the structure of the network is estimated. For this project, the speed and scalability of NetRate (the implemented algorithm) is improved and its constraints are analysed. Moreover, the model of neural network and inference algorithm are changed for it to be used on a real mouse's spike data-set and a benchmark is presented for analysing inference accuracy when no ground truth is available.

Acknowledgements

I would firstly like to express my immense gratitude to my supervisor, Dr. Pier Luigi Dragotti, for his guidance during the course of this project and Ms. Roxana Alexandru for her invaluable advice.

.

This journey would not have been the same without the care and support of my closest friends here at Imperial. For them I offer my most sincere wish for a fruitful and happy future.

Finally, I would like to show my heartfelt gratitude to my parents and my brother for being the pillars of my strength and motivation during the whole of my studies. Without their support I would not have attained a fulfilling education at Imperial College London.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The brain is a complex machine, it allows the human being to think, communicate and feel. It does so thanks to the billions of neurons that communicate in a dense network through synapses. Simple interactions between neurons build large structures of complex intelligence. However, little is known about how it works. By studying how the neurons structure to store and process information we can begin to understand how the brain as a whole functions. This could have important applications in medicine for curing diseases such as Parkinson [1] and epilepsy [2], and in machine learning for the development of more intelligent neural networks.

In order to infer the network structure of a set of neurons, they are treated as a diffusion network. In this scenario, a neuron's likelihood of spiking is increased if it is connected to another neuron that has spiked in the past. This is similar to the transmission of a disease where an infection (signal) is propagated within a population of transmitters. By evaluating the time of "infection", the relationship between two neurons can be probabilistically estimated. After computing the connections between all the neurons, an estimate of the topology of the network can be obtained.

Previous work on this topic [3, 4] evaluated the feasibility of using a maximum-likelihood estimator algorithm, NetRate [5], for the inference of the structure of biological neural networks. A network was simulated using the Izhikevic neuron model [6] and the Brian simulator [7]. The connections between the neurons were then estimated, compared to the original network and the performance of the algorithm was evaluated. Recent developments in technology now allow scientists to obtain individual neuron spike information from the brain tissue [8, 9, 10]. This data is very useful and serves as a mean of evaluating the performance of the algorithm with real neurons. Moreover, this information can help in creating simulated networks that resemble more the real biological ones.

1.1 Motivations

The aim of this project is to improve on the state of the art research of network inference and the understanding of the underlying structure of the brain. There are many ways in which this can be done such as scalability, increasing the similarity between simulated and real neural networks or changing NetRate so that it is more adapted to the problem at hand. Ultimately, the goal is to employ the algorithm on real spiking recordings and make topology estimations out of them. For this project, some of these objectives will be attempted and evaluated.

1.1.1 Increasing the speed of the algorithm

In [4], the size of the studied networks ranged from 10 to 30 nodes with the exception of one experiment with 50 neurons. This was due to the fact that the algorithm is very computationally expensive and it takes a long time for it to provide results. NetRate can be parallelized by nature since it computes an optimization problem for each column in a matrix. However, it makes use of CVX, a package for specifying and solving convex programs [11, 12]. Until this date, CVX cannot be naturally parallelized from within MATLAB.

In this project a novel approach is implemented where the algorithm can be parallelized and the size of the network that can be computed increased. Instead of attempting to parallelize CVX from within MATLAB, several MATLAB instances are run at the same time from the terminal and thus achieving parallelism. The aim of the project is, thus, to explain this approach and to measure accurately how much faster the algorithm is with different sizes of networks.

1.1.2 Inferring the connectivity of a real dataset

The objective of testing the algorithm on simulated networks is to eventually be able to use it on real spiking data. The information obtained from such an experiment would help researchers in their study of the brain.

A simulation that resembles more the behaviour of a network in a real world scenario provides a more meaningful insight to the performance of the algorithm on real brain tissue recordings. Moreover, it allows to test whether the Network Inference algorithm would still work under the conditions seen in the real dataset. For this project, such a simulation is computed and the connections within a real neural network are estimated.

Chapter 2

Background

2.1 Definition of connectivity

The definition of connectivity between neurons has a history of lack of consensus among the scientific community. Connectivity studies from different researchers may lead to different results depending on how they define it, as they may be looking at different aspects of connectivity. The two main accepted definitions that are used are functional and effective connectivity [13].

Functional connectivity is the temporal correlation between spatially remote neurophysiological events [14]. Studies on this topic began with electroencephalography (EEG) measurements. Some methods to measure functional connectivity include the evaluation of the correlation in the frequency domain between EEG signals at different scalp locations [15], and the cross-correlation of the time series measurements from a pair of electrodes [16]. However, due to the volume conduction of brain tissue, the electrical activity from the scalp cannot infer the individual neuron behaviour below the electrode [13].

Effective connectivity was defined in [14] as the influence that one neural system exerts on another. Effective connectivity can be measured in terms of efficacy and contribution. At a synaptic level it can be expressed as in Eq.2.1, where x_j is the post-synaptic response to many pre-synaptic inputs x_i and $\mathbf{W}_{ij}$ is the efficacy of the connections between neurons i and j. Contribution is reflected in Eq.2.2 as the effect of i on j relative to all pre-synaptic inputs. Using this definition, directional effects are taken into account and a richer representation of the network can be attained. Following the approach in [4], this project will focus on the effective connectivity of neurons in a network.

$$x_j = \Sigma \mathbf{W}_{ij} \times x_i \tag{2.1}$$

$$\frac{\mathbf{W}_{ij}}{\Sigma \mathbf{W}_{ij}} \tag{2.2}$$

2.2 Izhikevich neuron model

In order to understand how the brain works we must be able to replicate the behaviour of individual neurons applying simple and accurate models. However, as explained in [6], meeting both criteria can be challenging. The Hodgkin–Huxley model [17] is very accurate as it can emulate the rich firing patterns of many types of neurons. However, it is very computationally expensive and only a few neurons can be computed in real time. The integrate-and-fire model [18] has the opposite problem: it is computationally simple but it is an unrealistic representation of the neuron since it does not capture the firing patterns with sufficient accuracy [6]. In contrast, the Izhikevich neuron model [6] meets both criteria. Tens of thousands of spiking cortical neurons can be simulated in real time by simplifying the Hodgkin-Huxley model into the two dimensional system of differential equations shown below.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.3}$$

$$u' = a(bv - u) \tag{2.4}$$

with the auxiliary after-spike reseting

$$\text{if } v \geq 30\text{mV, then } \begin{cases} v & \leftarrow c \\ u & \leftarrow u + d \end{cases} \tag{2.5}$$

Here, the dimensionless variables v and u represent the membrane potential of the neuron and the membrane recovery, respectively. When a spike reaches its apex (30 mV), both these variables are reset according to Eq. 2.5. The differentiation is taken with respect to time. Synaptic or injected DC currents are represented by the variable I. Just as with real neurons, the threshold is not fixed and it's based on previous spikes.
On the other hand, a, b, c and d are dimensionless parameters. a determines the speed of the recovery variable u, b defines the sensitivity of the recovery variable u to sub-threshold fluctuations of the membrane potential v. Finally, c and d determine the after-spike reset value of the recovery variables v and u, respectively.

The relevance of this algorithm stems from the fact that, different combinations of the parameters provide the model with a rich variety of firing patterns. When analysing the neocortical neurons in the mammalian brain, a number of different classes of excitatory neurons can be found [19, 20] such as RS (regular spiking), IB (intrinsically bursting) and CH (chattering). From the inhibitory type of neurons, two classes can be found: FS (fast spiking and LTS (low-threshold spiking). Other interesting classes of neurons are the TC (thalamo-cortical) and the RZ (resonator). A visual representation of these neurons can be observed in figure 2.1. It is of great importance to understand what types of neurons can be found so that a simulated network can become a closer representation of what can be found on a real brain. In order to simplify the network to be inferred, the only type of neurons simulated in the network were the excitatory regular spiking neurons. This was achieved by setting the parameters to $a = 0.02$, $b = 0.2$, $c = -65$ and $d = 8$. This type of neuron

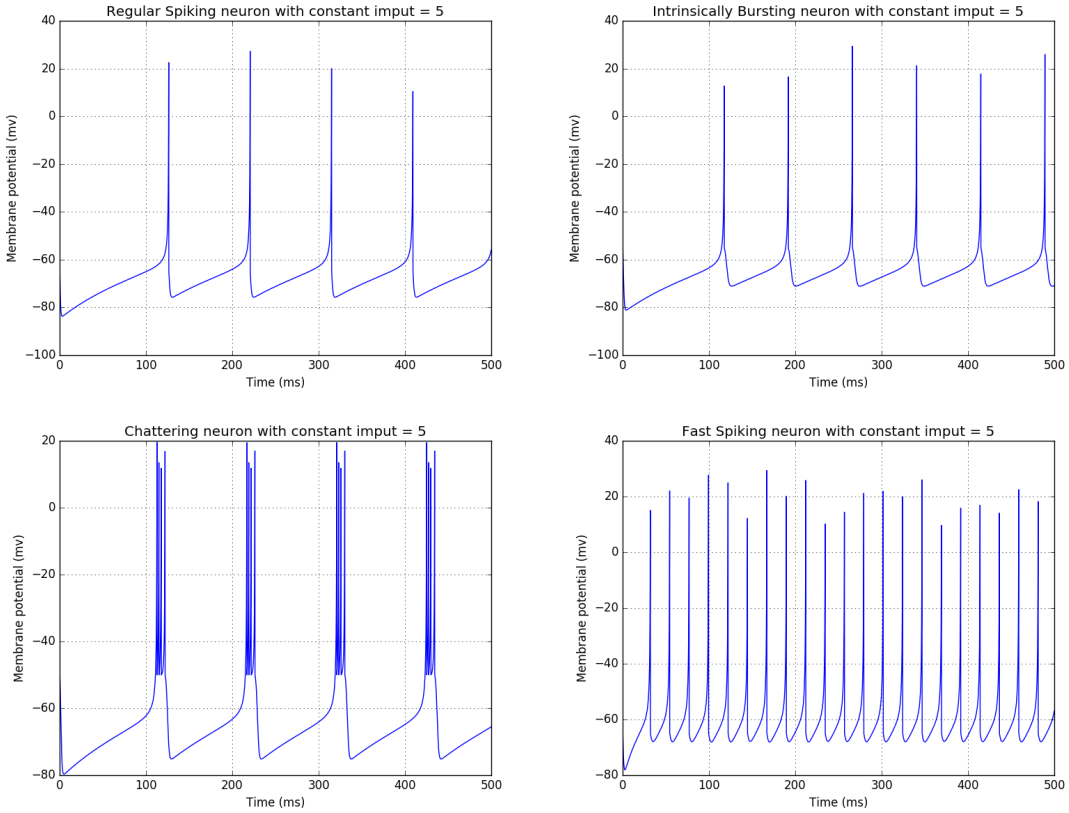


Figure 2.1: Types of neurons in the mammalian brain. Generated with the Brian Simulator [7] using the Izhikevich neuron model [6]

is the most common type of excitatory neuron in the brain. There is also a ratio of excitatory and inhibitory neurons of 4 to 1 in the mammalian brain, respectively [6].

In order to make use of the Izhikevich neuron model, Eq.2.3 is input to the Brian Simulator. This library computes the membrane potential voltage of the interacting neurons in the network and outputs all of their spiking times. This data will then be used to compute the NetRate algorithm.

2.3 Netrate

2.3.1 Diffusion processes

In order to infer the underlying structure of a network, [4] employed the NetRate algorithm developed by Rodriguez [5] by treating the network as a diffusion process.

The study of diffusion networks is based on the observation of the nodes in a system when they take a certain action: get infected by a virus, share a piece of information, etc. A problem concerning this kind of studies lies on the fact that we can only understand when and where these nodes propagate but not how or why the do so. An example of this is the propagation of a virus

in a population. We can tell who and when somebody got infected but not who infected him. For the rest of this section we will refer to the propagation of an infection as the object of study of the network.

To infer the mechanisms behind diffusion processes the time of infection is analysed. A model needs to be created with some assumptions about the structures that generate diffusion processes:

- The network in a diffusion process is fixed, unknown and directed: Connections do not change in time, it is not known what the connections are and connections are not bilateral.
- Infections are binary, they can only be infected or not infected, no partial infections are considered. For real neurons this means that there is either a spike or there is not.
- Infections across the edges of the network occur independently from one another. The probability of node i being infected by node j is not dependent on what the probability of node k infecting node i is.
- The likelihood of a node a infecting node b at time t is modelled by a probability distribution dependent on a, b and t.
- All infections in a network are observed during a recorded time window. This time frame is called horizon [5]. The larger the horizon, the higher the probability of more infections.

NetRate aims to describe how infections occur during a period of time in a fixed network. This is achieved by finding the optimal network and transmission rates that maximize the likelihood of a set of observed cascades to occur. The mathematical definitions that construct this model will be explained in the following section.

2.3.2 Mathematical definitions

The following definitions in this section are necessary for the construction of the model with which we intend to infer the connectivity of the network. First, the data that is going to be analysed will be defined:

Observations are carried out on a population of N nodes that have created a set of C cascades $\{\mathbf{t}^1, \cdots, \mathbf{t}^{|C|}\}$. Each of the cascades $\mathbf{t}^c$ contains the infection times of all the population within a time period T^c. Each of these cascades is an N-dimensional vector with recordings of when the nodes were infected in the cascade. If a node was not infected during the time period $[0, T^c]$, a symbol ∞ is assigned. This does not mean that the node never gets infected. For simplicity, we define $T^c = T$. Node i is parent of node j if $t_i < t_j$ within the cascade.

$$\mathbf{t}^c := (t_1^c, \cdots, t_N^c), \quad t_k^c \in [0, T^c] \cup \infty \tag{2.6}$$

The pairwise interactions are to be studied in order to obtain the pairwise transmission likelihood between nodes in the network. It will be assumed that infections can occur at different rates along different edges in the network.

- $f(t_i|t_j, \alpha_{j,i})$ is the conditional likelihood of transmission between nodes j and i. It depends on the infection times (t_i, tj) and pairwise transmission rate $\alpha_{j,i}$.
- A node cannot be infected by a healthy node. Node j, infected at t_j, can only infect node i at time t_i if and only if $t_j < t_i$.
- Transmission rate $\alpha_{j,i} \geq 0$.

The cumulative density function is defined as $F(t_i|t_j; \alpha_{j,i})$ and is obtained from the transmission likelihood. If a node j was infected at time t_j, the probability that node i is not infected by node j by time t_i is given by the survival function of the edge $j \to i$:

$$S(t_i|t_j; \alpha_{j,i}) = 1 - F(t_i|t_j; \alpha_{j,i}) \tag{2.7}$$

The instantaneous infection rate, or hazard function, of the edge $j \to i$ is the ratio of the transmission likelihood over the survival function as shown in Eq.2.8.

$$H(t_i|t_j; \alpha_{j,i}) = \frac{f(t_i|t_j; \alpha_{j,i})}{S(t_i|t_j; \alpha_{j,i})} \tag{2.8}$$

With a complete set of definitions, it will now be possible to derive the algorithm behind NetRate as it will be shown in the next section.

2.3.3 Derivation of NetRate

Rodriguez [5] derives NetRate by studying the individual probability of infection of the nodes and then building the whole of the network. The probability of survival of any cascade is the probability that a node is not infected until time T, given that the parents are infected at the beginning of the cascade. For a non-infected node i, the probability that any of the nodes $1 \cdots N$ does not infect node i by time T is given by the product of the survival functions of each of the infected nodes k targeting node i because the different probabilities of infection are considered independent. This is illustrated in Eq.2.9.

$$\prod_{t_k \leq T} S(T \mid t_k; \alpha_{k,i}) \tag{2.9}$$

To compute the likelihood of a cascade $\mathbf{t} := (t_1, \cdots, t_N | t_i \leq T)$ we require the the likelihood of the recorded infections $\mathbf{t}^{\leq T} = (t_1, \cdots, t_N | t_i \leq T)$. Again, using independence, the likelihood factorizes as seen in 2.10. The likelihood of the cascade then becomes the conditional likelihood of the infection time given the rest of the cascade.

$$f(\mathbf{t}^{\leq T}; \mathbf{A}) = \prod_{t_i \leq T} f(t_i \mid t_1, \cdots, t_N \backslash t_i; \mathbf{A}) \tag{2.10}$$

As in [21], a node gets infected when the first parent infects the node. We now compute the likelihood of a potential parent j of being the first one by using Eq.2.9.

$$f(t_i \mid t_j; \alpha_{j,i}) \times \prod_{j \neq k, t_k < t_i} S(t_i \mid t_k; \alpha_{k,i}) \tag{2.11}$$

In this step, we calculate the conditional likelihood of Eq.2.10 by adding all the likelihoods of the mutually disjoint likelihoods that each potential parent is the first parent:

$$f(t_i \mid t_1, \cdots, t_N \backslash t_i; \mathbf{A}) = \sum_{j:t_j<t_i} f(t_i \mid t_j; \alpha_{j,i}) \times \prod_{j\neq k, t_k<t_i} S(t_i \mid t_k; \alpha_{k,i}) \tag{2.12}$$

Using Eq.2.10 and removing the condition $k \neq j$, the likelihood of infections then becomes:

$$f(\mathbf{t}^{\leq T}; \mathbf{A}) = \prod_{t_i \leq T} \prod_{k:t_k<t_i} S(t_i \mid t_k; \alpha_{k,i}) \times \sum_{j:t_j<t_i} \frac{f(t_i \mid t_j; \alpha_{j,i})}{S(t_i \mid t_j; \alpha_{j,i})} \tag{2.13}$$

However, Eq.2.13 needs to consider also the nodes that are not infected during the observation window. For this reason we add the multiplicative survival term from Eq.2.9 and replace the ratios from Eq.2.13 with hazard functions:

$$f(\mathbf{t}; \mathbf{A}) = \prod_{t_i \leq T} \prod_{t_m > T} S(T \mid t_i; \alpha_{i,m}) \times \prod_{k:t_k<t_i} S(t_i \mid t_k; \alpha_{k,i}) \sum_{j:t_j<t_i} H(t_i \mid t_j; \alpha_{j,i}) \tag{2.14}$$

The likelihood of a set of independent set of cascades $C = \{t^1, \cdots, t^{|C|}\}$ is the product of the likelihoods of all the individual cascades given by Eq.2.14:

$$\prod_{\mathbf{t}^c \in C} f(\mathbf{t}^c; \mathbf{A}) \tag{2.15}$$

The goal of the algorithm is to find the transmission rates $\alpha_{j,i}$ of all the edges in the network such that the likelihood of the set of cascades is maximized.

$$\text{minimize}_{\mathbf{A}} - \sum_{c \in C} \log f(\mathbf{t}; \mathbf{A}) \tag{2.16a}$$

$$\text{subject to } \alpha_{j,i} \geq 0, i, j = 1, \cdots, N, i \neq j, \tag{2.16b}$$

.

Here, $\mathbf{A} := \{\alpha_{j,i} \mid i, j = 1, \cdots, n, i \neq j\}$ are the variables and the edges of the network are defined as the pairs of nodes whose transmission rates $\alpha_{i,j} > 0$.

The solution to Eq.2.16 found in [5] is given by Eq.2.17a. The survival and hazard functions are concave in the parameter(s) of the transmission likelihoods and, therefore, convexity of Eq.2.16 follows from linearity. The network inference problem from Eq.2.16 is thus convex for Power-Law,

Rayleigh and Exponential models of the likelihood function.

$$L(\{\mathbf{t}^1 \cdots \mathbf{t}^{|C|}\}; \mathbf{A}) = \sum_c \Psi_1(\mathbf{t}^c; \mathbf{A}) + \Psi_2(\mathbf{t}^c; \mathbf{A}) + \Psi_3(\mathbf{t}^c; \mathbf{A}) \tag{2.17a}$$

$$\Psi_1(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \sum_{t_m > T} \log S(T \mid t_i; \alpha_{i,m}) \tag{2.17b}$$

$$\Psi_2(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \sum_{j:t_j < t_i} \log S(t_i \mid t_j; \alpha_{j,i}) \tag{2.17c}$$

$$\Psi_3(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \log\Big(\sum_{j:t_j < t_i} H(t_i \mid t_j; \alpha_{j,i}) \Big) \tag{2.17d}$$

The terms in Eq.2.17a depend only on the infection time differences $(t_i - t_j)$ and the transmission rates $\alpha_{j,i}$. Each of the terms adds a property to the solution of NetRate.

- The terms Ψ_1 and Ψ_2 apply a positively weighted norm on $\mathbf{A}$, thus encouraging sparse solutions.
- Ψ_2 penalizes the edges that transmit infections slowly and promotes edges that infect quickly.
- Ψ_1 penalizes edges to uninfected nodes until the time horizon. With a longer observation window the penalties become larger, but so does the probability of nodes becoming infected.
- Ψ_3 makes sure that all infected nodes have a minimum of one parent to avoid $\log 0 = -\infty$. Since the logarithm grows slowly, it slightly encourages infected nodes to have many parents.

2.3.4 Cascade generation

The maximization of the likelihood function requires data in the form of cascades in order to be computed. The time of infection of each of the nodes can be obtained from the network simulation but it must then be formatted into cascades. The rest of this section is based on [3], where cascade generation is explained.

1. At time $t = 0$, a random node is selected to carry the disease.
2. The disease propagates for a T amount of time (horizon) based on the pairwise transmission likelihood $f(t_i \mid t_j; \alpha_{j,i})$ of the edges in the network.
3. At the end of the simulation, a cascade is generated with the information from the times at which the nodes were infected.

As an example, let there be a network of 6 nodes ($N = 6$) and a horizon of $T = 20$. Let us select node 5 at time $t = 0$ to be the starting point of the experiment. The simulation begins and the disease spreads out. It infects node 2 at $t = 3$ and node 6 at $t = 5$. The resulting cascade has the form of Eq.2.6 would look like this:

$$\mathbf{t}^c = \{\infty, 3, \infty, \infty, 0, 5\} \tag{2.18}$$

Remember from Eq.2.6 that the symbol ∞ represents a node that is not infected during the cascade. Nodes 2 and 6 were infected while nodes 1, 3 and 4 remained healthy for the duration of the cascade. Since at time $t = 3$ the only infected node was 3, this node must have infected node 2. However, it becomes inconclusive as to which node infected 6 at time $t = 6$. We could be lead to believe that the uninfected nodes are not connected to any of the other infected nodes. However, cascades are probabilistic models and no one cascade can tell us what the values of $\alpha_{j,i}$ are. We would, therefore, require a large number of cascades in order to infer those values with a high confidence.

2.3.5 Performance metrics

Evaluating the performance of NetRate involves analysing the inferred network $\hat{G}$: which edges have been correctly inferred, which ones have been missed and what weights have been assigned to the inferred edges. These questions are answered in [5] by calculating the accuracy, precision, recall and MAE against the true network G^*:

1. Precision is the proportion of edges in the inferred network that exist in the true network.
2. Recall is the proportion of edges in the true network that exist in the inferred network.
3. Accuracy $= 1 - \frac{\sum_{i,j}|I(\alpha^*_{i,j}) - I(\hat{\alpha}_{i,j}|}{\sum_{i,j} I(\alpha^*_{i,j}) + \sum_{i,j} I(\hat{\alpha}_{i,j})}$, where $I(\alpha) = 1$ if $\alpha > 0$ and $I(\alpha) = 0$, otherwise.
4. MAE $= E[\mid \alpha^* - \hat{\alpha} \mid / \alpha^*]$, where α^* and $\hat{\alpha}$ are the true and estimated transmission rates, respectively.

2.4 Biological Neural Network

2.4.1 Structure of the Network

In order to understand how neural networks are connected, a clear visual representation is required. This is achieved with an adjacency matrix plot. These matrices can be of three different types: binary ($\alpha_{i,j} \in \{0, 1\}$), ternary ($\alpha_{i,j} \in \{-1, 0, 1\}$) or real ($\alpha_{i,j} \in \mathbb{R}$). Due to the characteristics of biological neuron connections, real adjacency matrices are employed. In figure **??** an adjacency matrix of a network of 20 nodes can be observed. When $\alpha^{BNN}_{j,i} \neq 0$, the target neuron i and the source neuron j are connected and a dot appears on the graph. The source nodes are indexed in the x-axis and the target nodes on the y-axis. The weight of each of the edges is represented by the diameter of the dot.

The superscript BNN in $\alpha^{BNN}_{j,i}$ is employed to differentitate between the weights in biological neural networks and the analogous transmission rates $\alpha_{j,i}$ in diffusion networks [3]. When neurons j and i are connected with a weight $\alpha^{BNN}_{j,i}$, everytime j spikes, it causes the membrane potential v from Eq.2.3 to increase by $\alpha^{BNN}_{j,i}$. If neuron i crosses the threshold of 30mV, it spikes. This phenomenon is proved in [4] and explained in section 2.4.2. Using the example in figure **??**, when node 10 spikes, it causes nodes 13 and 9 to increase by $\alpha^{BNN}_{10,13}$ and $\alpha^{BNN}_{10,9}$, respectively.

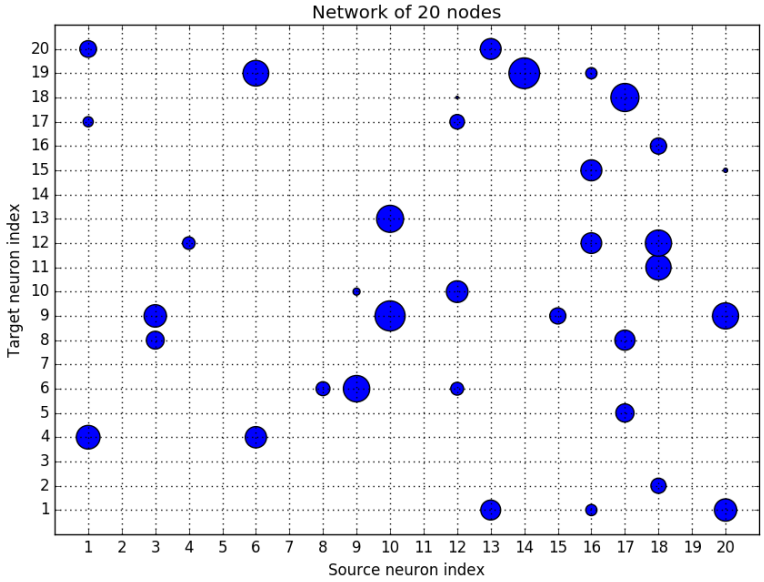


Figure 2.2: Adjacency matrix of a network of 20 nodes

The type of networks that were simulated in [4] are generated by Erdös & Reny random graphs, where each possible edge in the network has an independent probability p of being present. The weights assigned to the edges are the output of a uniform distribution in (0,30], the threshold value in Eq.2.3. It can be observed that the adjacency matrix does not contain weights where $i = j$ because spikes do not increase the membrane potential of the source neuron.

2.4.2 Input stimulus model for cascade generation

The neurons in the brain are susceptible to input stimuli from the rest of the neurons in the network. This is represented in the Izhikevich neuron model with the I component in Eq.2.3. This term can be employed to model injected current in the form of a DC input or can be normally distributed[1] to represent noise from interactions with neurons that are outside the network being recorded.

The selection of an appropriate input stimulus for the neurons was a challenge encountered by Malhotra [4]. With a Gaussian input I, neurons spike at random times and there is no systematic way of selecting the beginning of a cascade. In a cascade where nodes 1, 3 and 5 spike in chronological order, each of them could have in turn their own cascade: 1, 3 and 5; 3 and 5; and 5. However, this breaks the requirement of independence of cascades seen in section 2.3.1, where no same spike can appear in two cascades. For this reason, a well studied approach was required for the selection of cascades.

[1] In the Izhikevich neuron model [6], the mean and standard deviation are equal to 0 and 5 for excitatory neurons and to 0 and 2 for inhibitory neurons, respectively.

The solution Malhotra found to this problem was to provide one neuron at a time with a constant input of 12mV and the rest of the neurons with Gaussian noise. This caused the selected neuron to spike periodically. With an appropriate selection of the DC input, the spiking frequency could be changed, and with a sufficiently long time between spikes, the network could settle into a steady state. Unlike infections in diffusion networks, neurons can spike more than once. For this reason the horizon was arbitrarily limited so as to not allow two spikes from the same cascade to occur in the same cascade and, therefore, obey the law of binary infections imposed by NetRate [4].

This method provides a systematic way of generating cascades: every time the node with DC input spiked, a new independent cascade was created. In order to obtain cascade information from all the nodes in the network, all nodes are stimulated over the course of an experiment. Otherwise, if only one node was selected, no information would be extracted from the nodes with no direct connection to it [3].

The experimental results obtained by Malhotra show that an optimal amount of spiking information that achieves a high inferring performance is achieved with a stimulation time of 4,000 ms. This is due to the underlying probabilistic nature of NetRate [3]. In other words, more data does not result in a higher performance.

2.4.3 Likelihood function

The ability of NetRate to infer the weights in the adjacency matrix **A** stems from the fact that the shape of $f(t_i \mid t_j; a_{j,i})$ provides a probabilistic description of $\alpha_{j,i}$. The Izhikevich spiking neuron is modelled deterministically while the propagation of infections is probabilistic. For this reason the suitability of NetRate for the biological network inference was proved in [4].

As was explained in section 2.3.4, it is not possible to determine exactly which node caused some other node to become infected. However, this is not true for the Izhikevich neuron spikes. It was shown in [4] that the time it takes from a neuron i becoming unstable to the time it spikes is directly related to $\alpha_{j,i}$. A neuron becomes unstable when it crosses the threshold membrane value of 30mV. This can be caused by another neuron that spikes at that exact time or by random noise. In order to determine the shape that the likelihood function takes for different values of $\alpha_{j,i}$, an histogram of time was employed. It was observed that the likelihood function had an exponential and Rayleigh shape for low and large values of $\alpha_{j,i}$, respectively. Both of these distributions are convex for the solution of the optimization problem in Eq.2.16. NetRate only allows the use of one model, and because it is more relevant to infer the connections with larger weights, it was decided in [4] to employ the Rayleigh distribution.

	Accuracy	Recall	Precision	MAE
Average performance	0.667	0.633	0.704	0.997
Best performance	0.778	0.7	0.875	0.996
Worst performance	0.596	0.567	0.63	0.994

Table 2.1: 10 neuron network inference results obtained in [4]

2.4.4 Network inference results

The final test to determine the feasibility of the proposed algorithm in [4] was to compare an original simulated network and the resulting inferred network. Due to the underlying probabilistic nature of the network, the ability to infer the connections is different each time the experiment is performed. To provide a better representation of the performance of the algorithm, an average of 10 simulations was made. The results published in [4] for a network of 10 neurons can be seen in table 2.1.

When analyzing the results, it can be observed that the algorithm is good at detecting the edges with large weights from the network since it has a high value for accuracy, precision and recall. However, the high MAE shows that the algorithm is unable to infer the weights of the edges $\alpha_{j,i}$ correctly. A more extensive explanation for the high MAE can be found in [3].

Chapter 3

Improving the speed of NetRate

NetRate is a powerful algorithm that can make a good estimate of the connections of the nodes in a network. It analyses the spiking time of numerous neurons and constructs cascades that are used in the optimization problem. However, this is a computationally expensive process due to the large number of interactions between each of the neurons in the network. Moreover, as the size of the network increases, the number of cascades that are built grows exponentially. For a network of 10 neurons it only takes 8 minutes to obtain a result using one processor[1]. However, for each addition of ten neurons to the system, the computation time increases threefold.

For this algorithm to eventually become useful in the area of neural signal processing it must be able to scale up and analyse systems of hundreds if not thousands of neurons. Fortunately, NetRate is an inherently parallel problem because it computes an independent optimization problem for each of the nodes of the network. A node j within a system of N nodes has $N-1$ directed connections to all the neurons but itself. This makes the diagonal entries of the adjacency matrix equal to zero. Remember that the transmission rate of a node with itself is null $\alpha_{j,i} = 0$ if $j = i$.

A set of cascades is obtained from the spiking times of the system and assigned to the neuron node that originated them. This is necessary in order to compute each of the rows of the adjacency matrix. Then, they are used to build the components of the optimization problem. Therefore, after the cascade information is ready, each of the rows of the adjacency matrix can be computed with a different processor.

The objective function of NetRate's optimization problem makes use of logarithmic functions. Some solvers, such as SDPT3 [22, 23] (the one used by CVX) do not have support for these kind of problems and make use of recursive quadratic programming. This is a relatively new field of research [24] where a quadratic approximation of the objective function is taken. The solution to the new problem will converge to the one of the original problem for a sufficient number of iterations at which the initialization values are shifted towards the solution of the previous iteration.

[1]The processors used throughout the whole work are the Intel(R) Core(TM) i7-4770 CPU @ 3.4GHz with 12GB of RAM

One of the problems encountered in [3] was the lack of parallelization capabilities of the CVX software package used for NetRate. The package was not built to be parallelized and, therefore, aiming to do so would require a low level redevelopment of the software. However, the attempts of speed up were always carried out from within MATLAB. For this reason, a new approach, were the parallelization is achieved by opening several MATLAB instances is presented.

3.0.1 Parallelization of NetRate

When an algorithm is parallelized and each of the processes are completely independent, all the information required for its computation must be available from the beginning. Otherwise, a special communication protocol between the processes must be carried out. Then, after all of them have finished, their outputs must be recombined in the same way as if only one processor had computed the whole algorithm. An analysis of the necessary steps for computing NetRate is critical to understand what benefit can be obtained from parallelization:

1. The two files obtained from the Brian Simulator containing the indexes and times of each of the spikes must be converted into cascades and assigned to each of the neurons in the network that originated them.
2. The components that constitute the objective function and the constraints are constructed for each of the nodes in the network. This requires the characterization of the hazard and log survival functions.
3. Each of these components is assembled together to form the optimization problem in 2.16 for each of the rows of the adjacency matrix.
4. The software package CVX is used to compute the optimization problem that returns the optimal weights.
5. Post-processing of the solution is carried out. This includes cutting off adjacency weights below a certain threshold in order to promote sparsity.

From the steps above, it can be observed that the one that requires the most amount of computation power is number 4, where CVX is executed. Moreover, the information required to compute each specific row of the adjacency matrix is obtained in step 2. Thus, the ramification of the jobs occurs from step 1 to 2. From this point onwards, the parallelization is possible. However, due to the insignificant computation time and an increased complexity of a parallelized step 2, it makes it unnecessary to parallelize. Steps 3 and 4 are very closely linked: in order to use CVX, the problem must be defined following the rules of CVX and, thus, it is easier if both of them are computed by the same processor.

It can be concluded that an optimal benefit from parallelization can be achieved by assigning the individual tasks corresponding to each of the rows of the adjacency matrix to the available number of processors. Let N be the number of nodes in a network, $\boldsymbol{\alpha}_n = \{\alpha_{n,1}, \alpha_{n,2}, \cdots, \alpha_{n,N}\}$

and let $\boldsymbol{C}_n \subset \boldsymbol{C}$, $n \in [1, N]$ be the set of cascades originated by node n. Then, the structure of the proposed parallelized NetRate is described in figure 3.1.

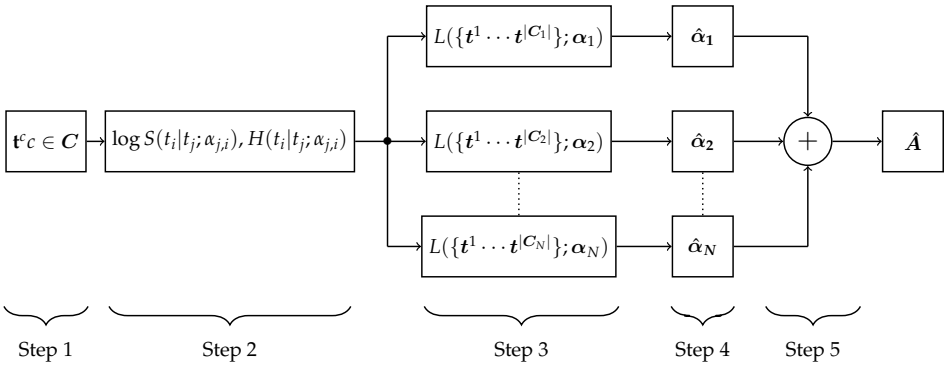


Figure 3.1: Diagram of the NetRate parallelization process

Originally, the algorithm computed steps 3 and 4 sequentially. Once the transmission rates of a row of the adjacency matrix were computed, it continued with the following one. However, in a parallelized NetRate, they are all computed at the same time with step 5 also involving the stacking of each of the $\hat{\boldsymbol{\alpha}}_n$ vectors to form the matrix $\hat{\boldsymbol{A}}$. The components in step 2 correspond to the ones seen in Eq.2.17a and whose solution is the assembled version of the one in step 3.

Once the structure of the algorithm is laid out, it still remains to plan how each of the jobs in steps 3 and 4 are will be assigned to the available processors. Not all the jobs take the same amount of time to be computed because they heavily depend on the number of cascades for their given node. As an illustration, a network of 10 neurons is simulated, and the number of cascades belonging to each node is shown in Eq.3.1.

$$NoC = \{12, 21, 62, 166, 21, 67, 17, 30, 18, 81\} \tag{3.1}$$

The mean of the distribution is 49 and the standard deviation 46. This means that the number of cascades differs significantly from node to node and that an appropriate way of distributing the jobs is required in order to keep all processors similarly busy. This is necessary because the whole algorithm will not finish until the last processor has estimated the weight parameters belonging to the last node.

Let M be the number of processors and N the number of nodes, where $N > M$. The first M with the largest number of cascades are assigned in order to each of the processors. Each of the remaining $N - M$ nodes is assigned to the processors following Eq.3.2.

$$argmin_n \; p_i + NoC_n, \tag{3.2}$$

Where $i \in [0, M]$ is the processor number, $n \in [0, N - M]$ the node index and p_i is the sum of all the number of cascades assigned to processor i. This method ensures that each of the processors has as close number of nodes as possible. Using Eq.3.1 with $N = 10$, the distribution among $M = 4$ processors becomes:

$$p_1 = \{166\}, p_2 = \{81, 21, 12\}, p_3 = \{67, 21, 18\}, p_4 = \{62, 30, 17\} \tag{3.3}$$

The resulting number of cascades for each of the processors becomes 166, 114, 106 and 109, respectively. This is a more balanced distribution than if the nodes had been assigned in any other way.

It remains to clarify in which order each of the nodes must be computed. One constraint that limits the algorithm is memory. The larger the number of cascades that need to be computed, the more memory is required for the algorithm to compute the weights. The relationship between the number of cascades and size of the network is exponential and, as it grows, NetRate makes use of a larger amount of memory. Each of the processors requires its own memory to perform NetRate in parallel. Thus, it becomes prohibitive to use several processors at the same time for large networks. However, some minor adjustments can be made to increase the capability of a parallelized NetRate by choosing which nodes are computed first. If all the largest nodes are computed at the same time the computer will not be able to finish the task for a sufficiently big network. For this reason, half of the processors will start with the nodes whose number of cascades is the lowest. The other half will do the opposite and compute the ones with the highest number of cascades. This way, the number of cascades computed at any given moment is levelled out and the likelihood of sudden spikes of memory usage are reduced.

As mentioned above, the CVX package cannot be parallelized naturally. Since the previous attempts to do so failed [3], a new method had to be implemented. For this project, instead of parallelizing NetRate from within MATLAB, several MATLAB instances are opened that work independently of each other. Each of these instances outputs a csv file and they are all combined by a Python script at the end of the computation.

3.0.2 Speed improvement results

In this section the speed performance of the parallelized NetRate is evaluated. The computation time of NetRate for networks of different sizes, a stimulation period of 4000 ms and using 1, 4 and 8 processors is displayed in figure 3.2. Due to memory constraints of the computer, only up to 40 nodes were evaluated. As explained above, the more processors used at the time the more memory is required and this made the algorithm stall when using 8 processors in a network of 50 neurons.

There is a significant improvement in the speed of the algorithm when comparing 1 processor to 4 and 8 processors. For 30 neurons, it takes 22, 8 and 9 minutes for 1, 4 and 8 processors

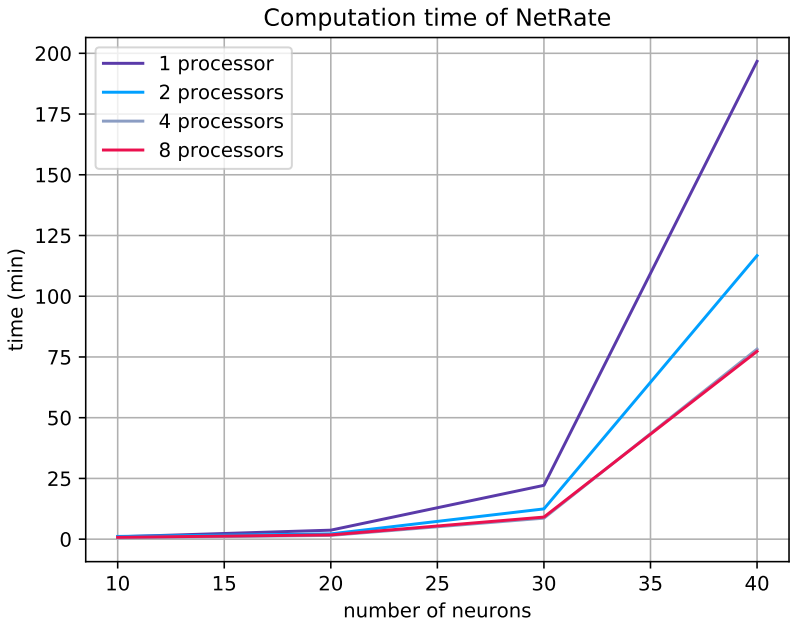


Figure 3.2: Computation time of NetRate

whereas for 40 neurons, it takes 196, 77 and 78 minutes. Although these are good results, they are far from ideal. Firstly, the time it takes to do the computation with 4 processors is not 4 times faster than with just one processor. It is, in fact, approximately 2.6 for the networks with 30 and 40 neurons. Moreover, using 8 processors instead of 4 results in no speed improvement for any of the networks. It is difficult to understand this behaviour because deeper analysis shows that all 8 processors are busy during the computation of the algorithm. One possible explanation is that the only steps that are actually being parallelized are number 3 from figure 3.1, where the optimization problem in 2.16 for each of the rows of the adjacency matrix is built, and half of step 4, where the CVX package is used but the solver has not been called. This means that the solver in the CVX package is a shared resource among all the running MATLAB instances and that each of the processors waits in a queue to compute its own row of the adjacency matrix. Further investigation into this hypothesis leads in this direction: when printing the progress of the optimization solver SDPT3, there is a linear behaviour i.e., all processes print in an orderly fashion and there is no overlapping between them. However, further research into this issue must be carried out.

In this section it has been described what the steps of NetRate are, how the parallelization is implemented and what its limitations are. It is necessary to have a fast algorithm for it to be used with large networks. Finally, it was shown that although there is a significant improvement in the speed of NetRate, it is not as good as desired, and it also poses further restrictions on memory usage.

Chapter 4

Simulating a biological neural network

Up until now the inference of networks using NetRate has been used only on simulations. A random structure was generated and the spikes simulated using the Brian simulator. This is very useful because it allows the possibility of comparing the inferred network to a ground truth and evaluating the performance of the algorithm. However, the goal is to be able to implement NetRate on real biological networks whose topology could provide insight to scientists. The analysis of real biological neural networks is met with many difficulties that need to be dealt with.

The main problem regarding the inference of the connectivity of a biological neural network is the lack of a ground truth with which to compare the capabilities of the algorithm. Although never to a full extent, there are certain ways that can help deal with this issue. The first is to simulate a network that replicates the characteristics of the real one. Under the assumption that the simulated network has a similar behaviour (not necessarily same connections) to the biological one, the algorithm can be implemented and tested. Then, the accuracy obtained can be taken to be an approximation to the accuracy of the inferred real network. However, this is a very big assumption and in reality the simulated network might be very different to the real one. However, the main reason for testing on a simulated network is to verify that any change in the stimulation model of the system or any new method of cascade generation is still valid. A deeper explanation of this topic will be given in section 4.3.

Another way of measuring performance would be to separate the dataset into training and testing and running NetRate on the training set. With the resulting estimated weights, given that a set of neurons have fired at time t, estimate the neuron with highest probability of spiking. The relevance of this analysis stems from the fact that if $\alpha_{j,i}$ is high, then the probability of neuron i spiking given that neuron j has fired is also high. If the accuracy of prediction is sufficiently high, the network can be taken to be correctly inferred.

It is important to find a suitable dataset to analyse. It must either be made out of voltage readings from an array of sensors in a cluster of neurons or spike times and indices[1]. The second option is preferable because it would not require spike sorting. It would also be advantageous if the dataset contained some biological information as to the type of neurons present in the system, how they are connected (if there is any biological way of measuring it) or where they are located. This information would help in creating a reliable simulation of the system and having a way of estimating a ground truth. In the next section, the dataset that is going to be used will be discussed.

4.1 Mouse somatosensory cortex neuron dataset

For this project, the dataset that will be studied is the CRNCNS mouse somatosensory cortex SSC-3 dataset [8, 9, 10]. This is a recording of the spiking activity from a mouse's somatosensory cortex brain cells. These cells were grown in cultures for 2-4 weeks and then measured with a 512 multi-electrode array that sensed the voltage in the culture for each of the neurons. These recordings were then spike sorted using PCA.

The somatosensory cortex is a set of modules located in the neocortex in the brain. The neocortex is vital in giving humans many of its cognitive abilities such as language processing, logic, sensory perception and many others. The neocortex shares many of its characteristics and architecture across different species of animals which makes it a very interesting subject of study. The somatosensory cortex is special because it is responsible for the touch sensations and because its anatomy and physiology has been intensively investigated [25].

The dataset consists of 25 different 1 hour recordings with a varying number of neurons ranging from 98 to 594. The average number of spikes per neuron for all datasets is 2.1 Hz and the sampling frequency is 20 kHz. The dataset also contains additional information on the x-y coordinates of each of the neurons in the cultures. With all this information, it is of interest to analyse what the distribution of spikes is. Neurons with larger connections will spike more often than the rest. Figure 4.1 displays how the spikes are distributed in the network. It is clear that the number of spikes per neuron follows an exponential distribution. Most of the neurons have sparse connections and will fire less than 6000 times during the length of the recordings while a few neurons will fire many times due to their high connectivity. In the next section, a simulated network will be implemented that tries to mimic the observable characteristics of the real 98 neuron network.

[1]Here, the index is the neuron number that generates a specific spike

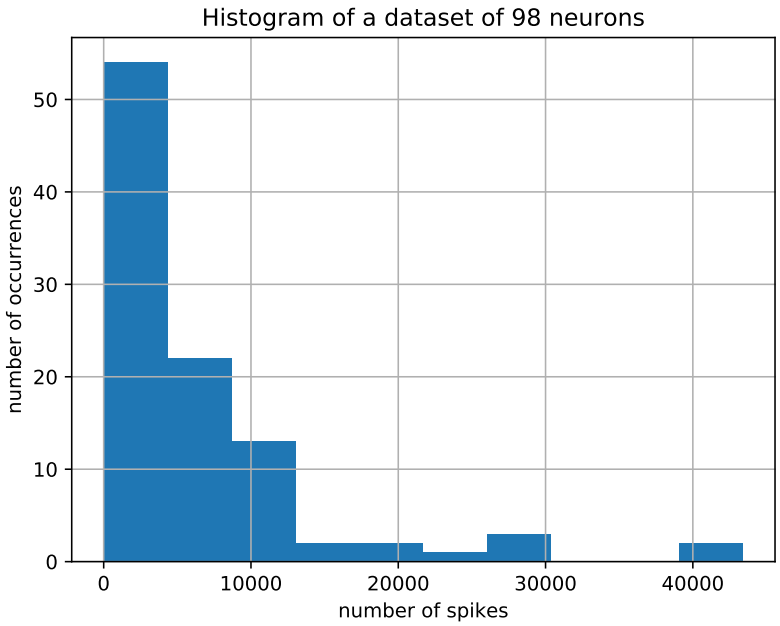


Figure 4.1: Histogram of the number of spikes in a network of 98 neurons

4.2 Input stimulus to the system

Previous work [4] simulated a network whose input was a DC voltage that stimulated each of the neurons one by one for a defined period of time (4 seconds). This proved to be very useful because it facilitated the analysis of each of the neurons regardless of their level of connectivity and because it provided a systematic way of generating cascades (more on this in section 4.4). However, the neural system cultures from the dataset are not stimulated. Instead, they are left alone to interact between each other. The neurons then communicate independently of the outside environment and spike at a lower frequency as a result.

The input stimulus cannot be left unchanged because the behaviour of the network would be radically different to that of the real one. It cannot be null either because then no interactions between the neurons would occur. Some type of random input noise must be present in order to have spikes. Moreover, the new model should have biological significance since a real network is being replicated. An option is investigated that could potentially meet these requirements.

4.2.1 System with random spikes

The proposed model intends to mimic biological behaviour by assuming that neurons spike when they want to transmit some information that they have conveyed by themselves. In order to do so, a random neuron is selected from the network and stimulated with random noise for some

Type of network	Neurons	Duration	Number of spikes	Mean	Freq spikes/neuron
Real	98	3600 s	636878	6498.2	1.2 Hz
Sim. DC stimulus	98	392 s	4038313	41207.3	105.12 Hz

Table 4.1: Spike characteristics of different types of neural networks

length of time. The benefit of this model arises from its ability to insure that every neuron has roughly the same probability of originating a spike train, from its random behaviour in the system and from its major biological resemblance than the model used in [4].

The noise that is input to the selected neuron is taken from the absolute value of a normal distribution of zero mean and standard deviation equal to α. Since, a model of only excitatory neurons is being implemented, negative values from the distribution are not valid.

The amount of time the selected neuron is stimulated for is also random. It is taken from a uniform distribution in the range 0 to 200 ms. There is no evidence to support the selection of this number since to this day, the behaviour of the neuron is not very well understood. However, it must be sufficiently large for the neuron to spike but to not too big so as to allow other neurons to be stimulated too.

4.3 Number of spikes

It can be seen in table 4.1 that one of the most obvious differences between the real dataset and the previous simulations is the total number of spikes. The number is significantly smaller than before even though the observation time is larger. This is because the stimulation model forced the neurons to spike very frequently. In [4] the total observation time was equal to the defined length of stimulation multiplied by the number of neurons. Since the stimulation period was found to be optimal for 4 seconds, then for a network of 98 neurons, this would result in a total of 392 seconds of recordings.

On the other hand, the real dataset consists of recordings of one hour. Therefore, the model of the simulation must be such that the number of spikes is roughly the same for a 1 hour simulation length. This can be achieved by varying the parameter α from the input stimulus of the model. This variable controls the standard deviation of the absolute normal distribution defined in section 4.2.1. The larger it is, the more voltage will be induced into the neuron and the higher its probability of spiking. Every simulation will have a considerably different number of spikes given the parameter α. However, its effect on the number of spikes is investigated for a 1 hour simulated network of 98 neurons. This will give a rough estimate of what the parameter value should be. Figure 4.2 shows that the most appropriate value of α is 4.

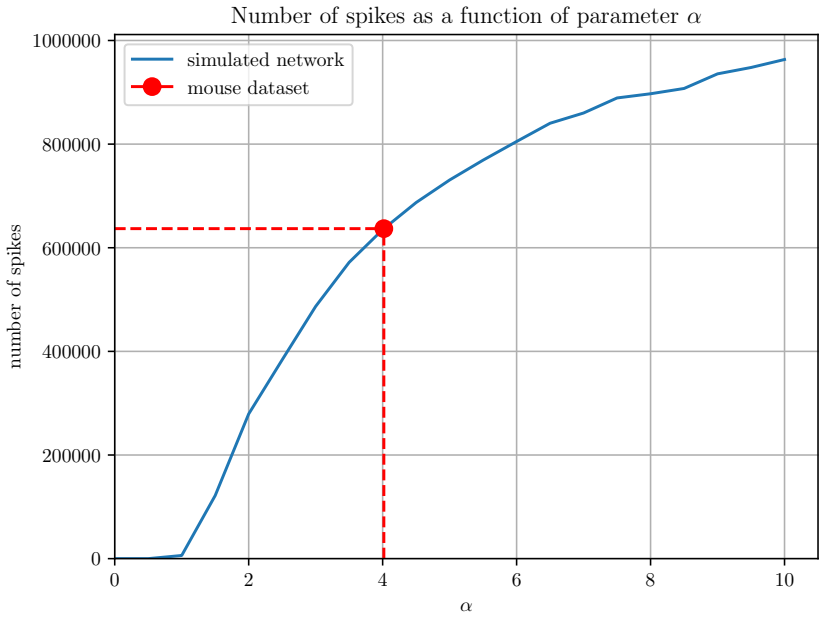


Figure 4.2: Number of spikes as a function of the parameter I for a 1 hour simulated network of 98 neurons and a random spike stimulus

4.4 Cascade generation

Once the network model is defined what remains to clarify is how the cascades will be built. The method used in [4] was simple and consistent. Since a DC stimulus was given to one neuron at a time, this neuron was selected to be the beginning of the cascade. This cascade would last for T length of time and only the first firing of each neuron would be taken into account. This insures that the third assumption from [5] is not broken i.e., observable artefacts in are binary. Moreover, with this cascade generation method, cascade independence was guaranteed because it provided a clear entry point into the diffusion process. Since the system no longer has a DC input, the cascade generation method must be changed. When selecting an appropriate method of cascade generation, certain factors need to be taken into account.

1. The higher the number of cascades that are generated, the more information will be conveyed and, in general, the better NetRate will be able to infer the connectivity of the network.

2. The more separated the cascades are from each other in the time domain, the more independent they will be. In [5], independence is defined as uniqueness of spikes for all the cascades in the diffusion process. However, for this project, an additional definition will be used i.e., the level of isolation between the different diffusion processes in the time series. Now, a set of independent cascades not only contains unique spikes but, each of the cascades in the set has no influence over each other.

3. A high sparsity insures that enough cascade information is obtained from all the neurons in the network. Otherwise, only neurons who spike often will be the ones generating cascades.

4.4.1 Method of maximum cascades

One way of building cascades is the proposed method of maximum cascades. This is a very simple approach that uses the first spiking neuron as the beginning of the cascade. This cascade lasts for the defined horizon and includes all the firing activity within this time period. This process is repeated after the last spike of the generated cascade and until the whole time series has been evaluated. It is worth remembering that, in the case of one neuron firing twice within the same time period, only the first one is taken into account and recorded in the cascade. As an example, let there be a network of 5 neurons that spike during a recording of 12 seconds and let the firing times and neuron indices be:

$$firings = \{1.5, 2.6, 4.7, 8.1, 8.4, 9.4, 11.2\} \tag{4.1}$$
$$indices = \{4, 3, 5, 3, 2, 2, 4\} \tag{4.2}$$

Then, using the method of maximum cascades, the resulting two cascades would be the ones seen below:

$$\mathbf{t}^1 = \{\infty, \infty, 1.1, 0, 3.2\} \tag{4.3}$$
$$\mathbf{t}^2 = \{\infty, 0.3, 0, 2.1, \infty\} \tag{4.4}$$

The first cascade starts at time $t = 1.5s$ and ends at $t = 6.5s$. During this time, neurons 3, 4 and 5 spike. Then, the second cascade, starts at time $t = 8.4s$ and finishes at $t = 12s$. Here, neurons 2, 3 and 4 spike. However, although neuron 2 spikes twice, only the first firing is taken into account. It is also worth remembering that, cascades are based on time differences rather than absolute times.

The advantage of this method stems from the large number of cascades that are generated. Every spike is used for generating cascades and, thus, the number of cascades is maximized. However, this is done irrespective of their quality. The quality of a cascade is defined as the fidelity of the probabilistic description of the network. In other words, how well it represents the true nature of the system. As an example, let the firings and indices of a simulation of 20 seconds with a horizon of 10 seconds be the ones in 4.6 and let the network connections be the ones in table 4.2.

$$firings = \{1, 9, 11, 12, 19\} \tag{4.5}$$
$$indices = \{1, 2, 3, 4, 2\} \tag{4.6}$$

Source	Target
1	2
1	4
2	3
2	4
3	2

Table 4.2: Network connections example

The resulting cascades would then be:

$$\mathbf{t}^1 = \{0, 8, \infty, \infty\} \tag{4.7}$$

$$\mathbf{t}^2 = \{\infty, 8, 0, 1\} \tag{4.8}$$

Although the above cascades are well constructed, the second cascade is of poor quality. This is because, under a deterministic model where there is no noise, the cascade would lead to believe that there is a connection between nodes 3 and 4 when, in fact, the spiking of node 4 was caused by either node 1 or 2. Since NetRate is made for inferring networks under stochastic conditions, this scenario could happen under an otherwise perfect system. However, the cascade method of maximum cascades is fundamentally flawed because cascades are not independent. The diffusion process from the first cascade has not reached a steady state and it has affected the second cascade. A new cascade method is proposed where the independence of cascades is increased.

4.4.2 Method of optimal independence

Under the new definition, independence of cascades cannot be guaranteed. Any spiking activity could cause another neuron to become unstable in the future or could, at least, contribute to this happening by increasing its membrane potential. However, it can be improved significantly by allowing the network to reach a steady state before starting the cascade.

The method of optimal independence poses a stability constraint on cascade generation. This is achieved by requiring the first spike of each cascade to be a specific amount of time far away from its immediate previous spike. For simplicity, in this project, this time variable will be equal to the time horizon used as the duration of the cascade.

This method increases the quality of the cascades in comparison to the one of maximum cascades. However, since it is more difficult to generate cascades that meet the requirements, there will also be fewer cascades. This is not always a problem, since, many times the number of cascades generated by the first method exceeds the computation capabilities of the computer at hand. By modifying the stability period parameter, the number of desired cascades can be chosen and only the ones with the highest independence will be used.

4.5 Mapping the weights of the adjacency matrix

The job of NetRate is to estimate the transmission rate of the nodes in a diffusion process. The transmission rate $\alpha_{j,i}$ is defined as the probability of a node i being infected by node j in any given cascade. However, this model of transmission is not suitable for the strength of connections in a neural network. This is because transmission rates lie in the range [0,1] while connections under the current network model are defined in the range [0,30].

A mapping was devised in [3] between these two parameters in order for NetRate to be used on neural networks. It consisted of a simple multiplication by 30 of the transmission rates. This mapping reduced the MAE from 0.99 to approximately 0.9. However, most of the inferred transmission rates lie in the range [0, 0.1] due to the large number of cascades where only one node spikes[2]. This corresponds to a mapping in the range [0,3], very far from the more simulated uniform distribution in the range [0,30]. For this reason, a new mapping is proposed. It consists of a linear mapping between the smallest and largest value in the set of inferred weights to the range [0,30] which, therefore, achieves the desired distribution.

4.6 Concluding remarks

At the beginning of this chapter the need for a simulated network that approximated the biological one was explained. Such a model can provide an approximation of the performance of the algorithm. It is based on the assumption that the simulation is a faithful representation of the real nature of the network. Moreover, the differences between previous simulated networks and this new real network make it vital to verify that the algorithm would still work under these new conditions.

A new model was defined that tries to mimic the behaviour of the real mouse recordings. For this reason, instead of simulating a network whose input consisted of a DC stimulus, a random input was established that consisted of the absolute value from a normal distribution of mean zero and standard deviation α. This stimulus would be given for a random amount time to a random neuron in the system.

Finally, two new methods of cascade generation were defined. This was necessary because the change of input stimulus made the previous approach obsolete. The proposed method of *maximum cascades* constructs cascades based on the firing times of the neurons and allows a large number of cascades to be generated. Finally, the method of *optimal independence* poses a constraint on the stability of the system thereby increasing the quality of the cascades.

[2] If only one neuron spikes during a cascade, NetRate will infer that whichever connection this node may have it will be of a low transmission rate.

Chapter 5

Testing performance without a ground truth

The analysis of NetRate's ability to infer the edges of a simulated network is based on the knowledge of a ground truth. The availability of such information allows the researcher to compare the structure of the inferred and original simulated networks. However, in a practical scenario, this is not possible because of the lack of such knowledge.

It is still of interest to have some sort of performance measure of how well the network has been inferred. Such a task would require splitting the recording into a training and testing set. The first one would be given to NetRate to obtain the adjacency matrix and the latter would be employed in the estimation of the performance of the algorithm on the training set. One disadvantage of this analysis comes from the fact that less data is given to NetRate and, therefore, its performance is hindered. In the following section, one such performance metric is proposed and discussed.

5.1 Spike prediction performance metric

A complete knowledge of connectivity of a network would allow a researcher to predict, up until a certain extent, the spiking activity of the system. This is because given the event of node j spiking and the knowledge of the directed connexions from this neuron, the membrane potential of these nodes can be computed. If any of them exceeds the membrane potential threshold, it will become unstable and, eventually, it will spike. The proposed metric of *spike prediction* builds on this idea.

Given the inferred adjacency matrix, the number K of firing events during a recording window of length T and the neuron index j of the first spike; the task is to predict which neurons are the ones that spike during the observation window. The predicted neurons will be the ones

Source node	Target node	Weight
1	2	25
1	3	5
1	5	1
2	4	10
4	2	2

Table 5.1: Example of connexions of a 5 neuron network

whose connectivity weights are the highest to the source neuron j by a path of any length. The path is defined as the number of nodes that are travelled to get to a certain destination node. The accuracy is defined as the ratio of correctly predicted neuron indexes over K. Neurons that spike twice are only taken into account once.

The introduction of the path into the prediction allows for the modelling of diffusion processes. This would not be possible if only the neurons that are directly connected to j were taken into account. Under this prediction model, a nested connexion whose path is of length equal or greater than 1 would be given preference over one of a shorter path length if the connectivity weight is larger. This is non ideal if the path length is too long, however, the probability of this occurring decreases significantly for every new step in the path.

As an example, let the connexions of a network of 5 neurons be defined by table 5.1 and let the firing indices in a period of observation be given by Eq.5.1. Then, $j = 1$ and the predicted neurons to spike would lie in the following order: 2, 4, 3, 5. Since $K = 4$, only the first three are kept i.e., 2 ,4, 3. Using this prediction the accuracy is equal to 0.67. The use of this metric is flawed for $K \approx N$, where N is the size of the network. This is because the chance of guessing correctly the neuron indices is high. However, it is used in this example for illustration purposes.

$$indices = \{1, 2, 4, 5\} \tag{5.1}$$

In order to see what the practical values of this metric are, a network of 15 neurons is simulated for 1500 seconds and the resulting spiking data is split into 90% training and 10% testing. The window length T used in the metric is set to the value of the horizon for simplicity. Using the knowledge of the ground truth network, the prediction accuracy is equal to 83.43%. However, if the inferred network, with an inference accuracy of 70.83%, is used, the prediction metric drops to 10.44%. The reason for this drastic fall lies on the high MAE (more on this in chapter 6).

Chapter 6

Inferring the connectivity of a mouse's neuronal system

In chapter 4 a neural network model was simulated to try and mimic the behaviour of the real network described in 4.1. A new input stimulus and two cascade generation methods were proposed. In this chapter, the suitability of these new methods will be proved by inferring the connectivity of the new simulation model outlined in section 4.2. Moreover, the effect of the simulation duration and the network size on the performance of the algorithm will be assessed.

Once the suitability has been demonstrated, the same algorithm will be used to infer the connectivity of one of the networks from the CRCNS dataset. Further analysis of this structure will be carried out and its main characteristics will be described.

In order to evaluate the performance of the algorithm, the four metrics described in 2.3.5 will be used. However, as outlined in [3], the structure of the network can have a great effect on NetRate's ability to estimate it. For this reason, for validating the suitability of the model and changes in performance, an average of 4 experiments will be used as the results displayed in this chapter.

6.1 Proof of the suitability of the algorithm

In this section, the suitability of the new network model and the consequent new cascade generation method are proved. In order to do so, a network of 20 neurons is simulated for 1500 seconds and its connectivity is inferred. If this estimation is sufficiently similar to the ground truth, the effectiveness of the algorithm will have been demonstrated. Figure 6.1 shows the averaged results of this analysis.

It is clear that, for the given network model and given the generated cascades, NetRate is relatively good at estimating the connections of the system. Just as in [4], MAE is very high, which

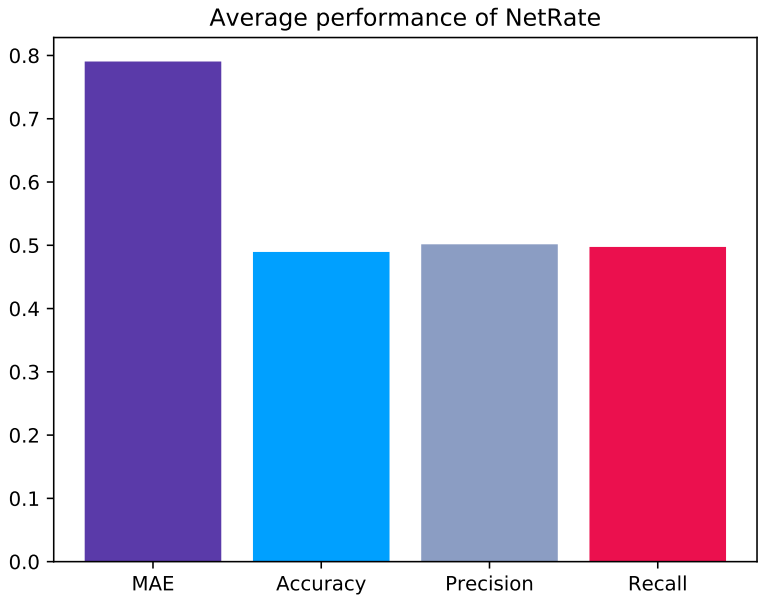


Figure 6.1: Average network inference performance for a neural network of 10 neurons and a stimulation time of 1500 seconds

means, it fails to infer the weights of the connections. To visualize the capabilities of the algorithm, the adjacency matrix is plotted in figure 6.2.

There is no obvious way of comparing the results achieved in this project with the ones obtained in [4]. This is because networks are simulated differently and the method for cascade generation needed to be changed. However, the DC input stimulation model gave way to cascades with a greater spiking activity. For this reason, it is difficult for the current model to achieve a higher performance than the one achieved in table 2.1. A big advantage of the new model, however, lies in its ability to infer networks without any external input to the system.

6.2 Effect of the size of the network and simulation time on the performance of NetRate

It is of interest to analyse how the performance of the algorithm varies with different simulation times. Figure 6.3 shows the results for this analysis. It can be observed that the larger the simulation time, the better NetRate's ability to infer the network. This finding is to be expected due to the fact that the longer the observation length, the more information is captured. However, this behaviour begins to plateau after 1250 seconds of simulation time.

Another variable that could have an effect on the performance of the algorithm is the size of

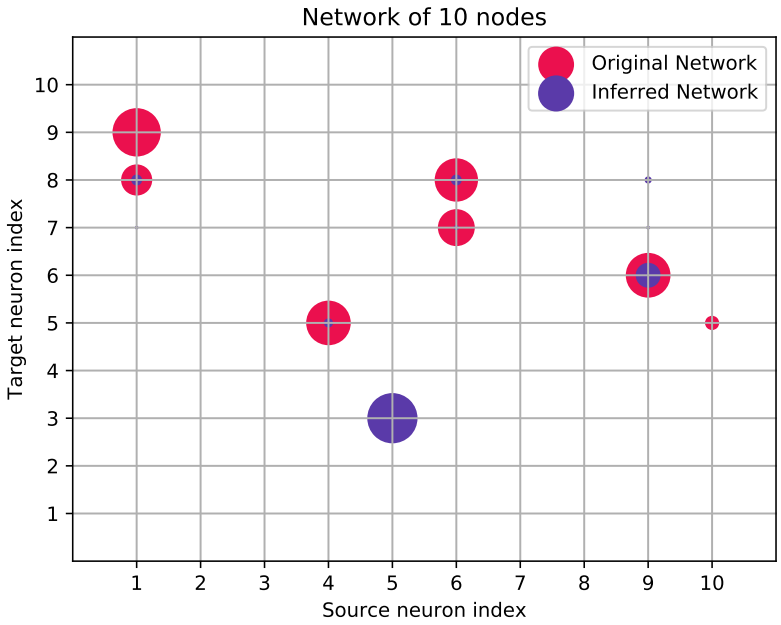


Figure 6.2: Adjacency matrix plot for the original and inferred networks of size 10.

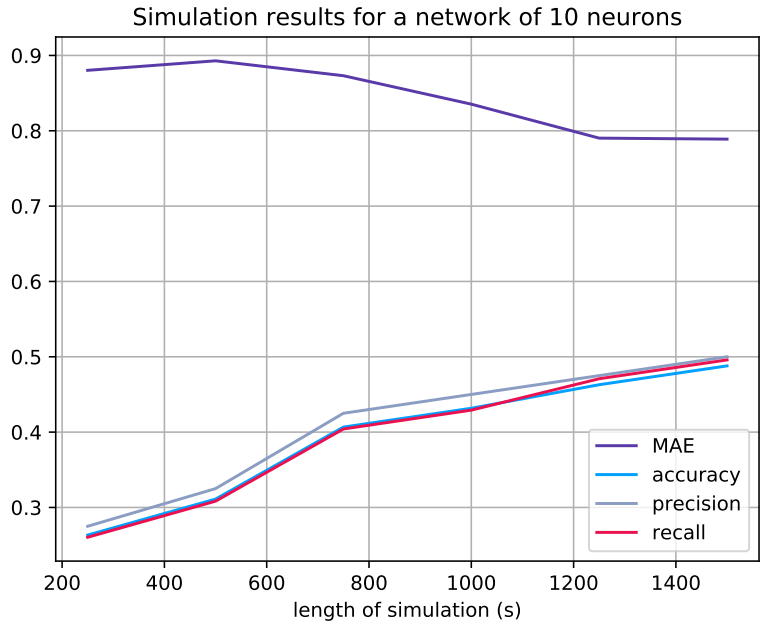


Figure 6.3: Network inference performance for a neural network of 10 neurons

Neuron number	1	2	3	4	5
Number of spikes	46	67	108	76	1630

Table 6.1: Spiking activity of a network of 5 neurons during a 1000 second simulation

Neuron number	1	2	3	4	5	6	7	8	9	10
Number of spikes	21	28	1642	42	846	1184	1065	1855	445	19

Table 6.2: Spiking activity of a network of 10 neurons during a 1000 second simulation

the network. The larger it is, the more connections are possible and the more information is required to infer the edges. Figure 6.4 shows the results of this study. Surprisingly, the inference ability of the algorithm goes up with the size of the network. In order find the reason for this phenomenon, a closer look needs to be taken to the number of firings for each network size. Spiking activity conveys information about the effect the neuron has on the network and vice-versa. For networks of 5 and 10 neurons and a simulation time of 1000 seconds, these can be seen in tables 6.1 and 6.2, respectively. When the network size is doubled, the ratio of spikes to neuron goes from 385.4 to 714.7. This is a significant increase which gives rise to this behaviour. With more spikes, more cascades are generated and, in general, the better NetRate will perform.

6.3 Inference results

Once the suitability of the algorithm has been proved, the connectivity of a biological neural network can be inferred. In this section, the 98 neuron network analysed in section 4.1 and whose evaluation suitability was proved in 6.1 is the one evaluated. Due to the dataset being large enough, the *optimal independence* method of cascade generation is employed. As explained in section 4.4.2, this will achieve a higher quality set of cascades.

When computing NetRate for a large dataset like the one at hand, the main constraint that the researcher has to deal with is memory. This is directly proportional to the number of cascades computed at any given moment. Therefore, if only one processor is being used, the constraint will be set by the node with the maximum number of cascades. For the current dataset and, using the *optimal independence* method of cascade generation, this corresponds to neuron 12 with 7956 cascades. The computation of this node requires an array of size 18Gb but, with the resources at hand, such an operation is not possible. For this reason, the dataset is partitioned into a smaller set containing the first half of the spike recordings. If the system is assumed to be static[1] for the whole length of the recording, then only the performance of NetRate could be affected by this split. The resulting maximum number of cascades is reduced to 5168, with the amount of memory required being lowered to 11Gb.

[1] A network is static if no connections are created or erased during the period of observation.

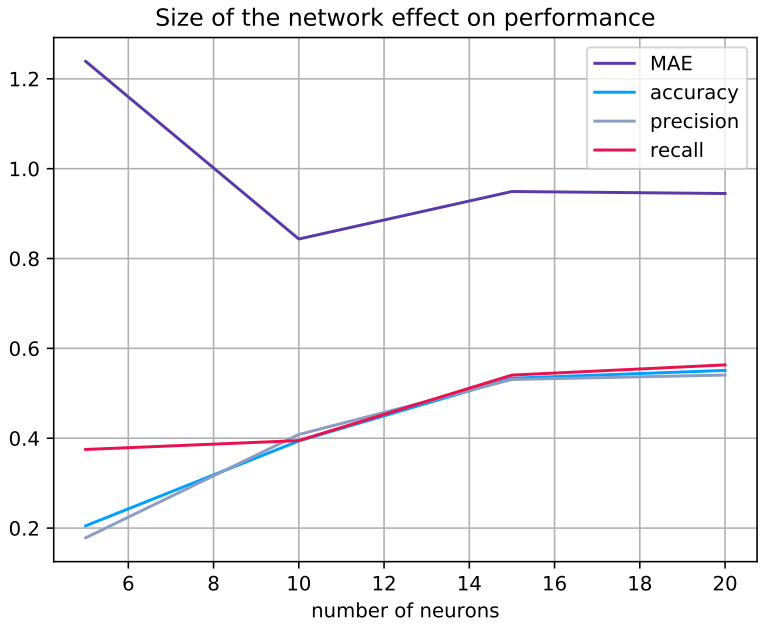


Figure 6.4: Average network inference performance for different network sizes

In figure 6.5, the resulting inferred network is displayed. Each of the nodes is represented by a circle whose connections are the lines adjacent to them. Since the network is directed, the edges are represented by arrows that indicate the source and target of the connections. Each of the nodes in the figure is coloured according to their degree. Nodes in directed networks can be described by their indegree and outdegree i.e., number of inward and outward connections, respectively. Red nodes have more inward connections and blue nodes more outward connections. If their inward and outward degree is the same, in the figure, these cancel out and their colour becomes purple, just like neurons that have no connections at all.

An illustration of the degree of nodes in the network is shown in figure 6.6. As it can be observed, the degree distribution follows an exponential behaviour. Most of the nodes have at least one connection and very few have more. In general, neurons with a greater indegree will spike more often since they are more influenced by their neighbours. This finding agrees with the behaviour of spiking activity shown in figure 4.1, where most neurons spike unfrequently and only few have a more intense spiking activity.

It is also of interest to analyse whether there exists any dependence between node distance and connectivity. In other words, whether two neurons close to each other have a higher chance of being connected. First, in order to test this hypothesis, the average distance between neurons is calculated. Since the dataset contains the x-y coordinates of all the neurons in the network, it is easy to see that this is equal to 831.57 μm. Next, the average edge distance is computed. The value

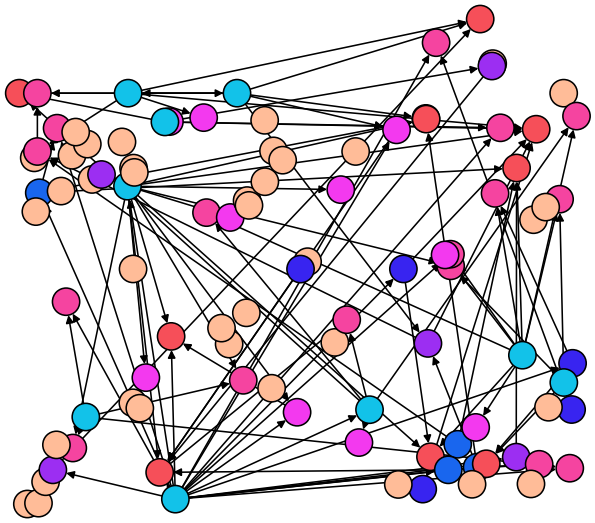


Figure 6.5: Inference of a biological neural network of 98 neurons. Red for indegree > outdegree, blue for outdegree > indegree and peach for non connected neurons.

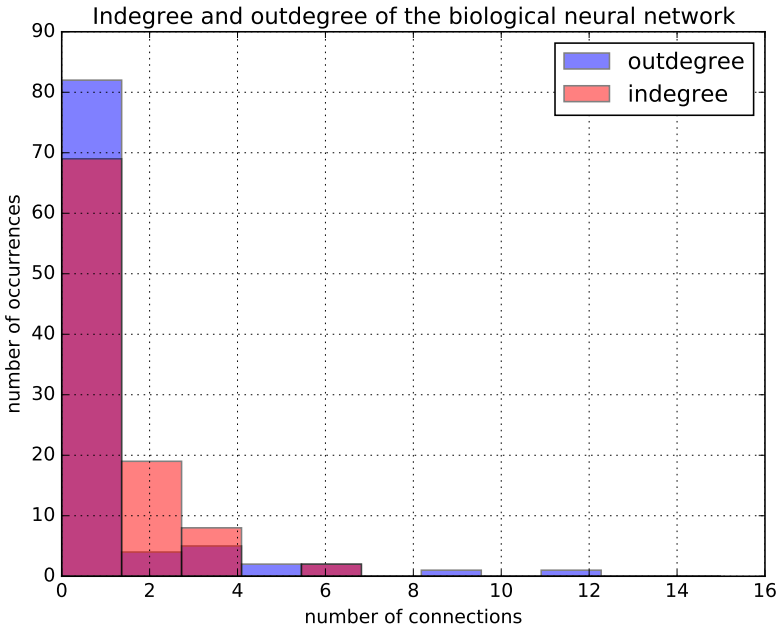


Figure 6.6: Indegree and outdegree histogram of a biological neural network

of this calculation is found to be 698.37 μm. Although this value is lower than that of the average node distance, the difference is not sufficiently large so as to prove the validity of the hypothesis.

Finally, since there is no ground truth, the performance metric outlined in chapter 5 is used to estimate the inferred knowledge of the biological neural network. Partitioning the dataset was not an issue for this network since, in any case, only 50% of it could be used. The result for this evaluation was of 7.69%. Although it is relatively low, it demonstrates that the algorithm has gained some level of understanding of the network. This is because for $K \approx 3$ and a network size of 98, random guessing would only achieve an accuracy of approximately 2%.

Chapter 7

Software Package

The final version of the Network Inference software package is fruit of the combined work from the authors in [4] and myself. Every effort has been made in this project to deliver an algorithm that provides clear and comprehensive data to the scientist that intends to use it. Further changes to how the network is simulated and cascade generation modalities can be easily made with the recent changes to the structure of the software. A Github repository has been made available here for use and further development.

7.1 Programming languages

The software that is used for this project is based on a set of scripts written in bash, Python and MATLAB. Each of these languages is used in different areas where they excel due to either availability of libraries or ease of use.

Bash scripts are used as the entry point to the software package, their main function is iterating through tests, calling the functions that simulate and infer networks, and move the output files in an organized manner. MATLAB is the language in which the optimization problem is defined. The need for a convex optimization tool that implements recursive quadratic programming made it imperative to use this programming language. CVX is a software package that meets this criteria. Although there is also a version of this package available for Python, it does not make use of recursive quadratic programming and the accuracy of NetRate is, consequently, drastically reduced.

Python is the language of preference for this project. Not only is it easy to program and full of libraries, but it is also open-source and, therefore, easily available. A great effort has been made to move into this language from the previous version used in [4], which was mostly written in MATLAB. The parallelization process was carried out using this programming language with the use of the *multiprocessing* library. Cascade generation and performance evalutation was translated from MATLAB to Python.

One of the major constraints that has to be taken into account when using NetRate is memory and computation power. This issue is accentuated when trying to infer networks with a large number of nodes or when the number of cascades is high. This issue can be dealt with by using large computers that have MATLAB and the CVX package installed. However, many times researches do not have this hardware capabilities at their hands. For this reason, the use of cloud computing services is critical for high performance data processing. However, many of these services do not offer MATLAB compatibility and the CVX package cannot, therefore, be used. A scalable network inference algorithm would need to be written in an open source language such as Python. One of the goals of this project has been to go in that direction.

7.2 Description of the algorithm

Once the required components for the Network Inference software have been installed, the package is ready to be used. The entry point to the algorithm is *runner.sh*. This is a bash script that defines the simulation variables, iterates through different testing options and calls *main.sh*. This later file calls each of the modules of the algorithm that are in charge of simulating the network, generating cascades, parallelizing NetRate and obtaining results.

The first module that is executed is the *izhikevichNetworkSimulation.py* script. It uses the Brian simulator to generate a network based on the equations in 2.3. It randomizes the connections between the neurons and applies a defined input stimulus to the system. Finally, the script outputs three csv files containing the network weights, the firing times and the indices of the neurons that fired.

After the network and spiking data is simulated, the next step is to generate the cascades. Given a cascade generation option, *generate_cascades.py* will produce these sets of spikes for NetRate to use. This script is also in charge of creating the *A_bad* and *A_potential* matrices. These contain the survival and hazard functions from step 2 in 3.1. Moreover, it also creates a vector containing the number of firings each of the neurons has in the set of cascades.

The script *initial_time.py* is a very simple program that starts measuring the time it takes for NetRate to infer the network. It stores the time in a pickle file that is latter opened in *compare_network.py*. This is used to test NetRate's parallelization performance.

The main module of the algorithm starts with *parallelize_cvx.py*. Given the number of processors, this python script is in charge of opening several MATLAB instances and telling each of them which nodes to compute NetRate on. This is done by calling several times (the number of processors) the function *parallel_cvx.m*. This function iterates through all the nodes assigned to its given processor and calls *solve_using_cvx.m*, the actual NetRate function that defines the optimization problem using the *A_bad* and *A_potential* matrices, and the number of firings vec-

tor from *generate_cascades.py*. Every time *parallel_cvx.m* has finished with one node, it outputs a *csv* file containing the estimated weights from that node. All of these files are then rejoined in *compare_network.py* and compared to the ground truth. For this purpose, the MAE, accuracy, precision and recall performance metrics are used. The results are finally stored in a *csv* file in its corresponding folder.

7.3 Folder architecture

Many files are used for this project. This is due to the large number of tasks that need to be carried out and that three different programming languages are used. Moreover, it is vital to keep track of all the output files from both the Brian Simulator and NetRate. Many different modalities of the algorithm can be run and they need to be stored in an organized manner. Moreover, once the simulation or the cascades have been generated, there is no need to recompute them again if they have been saved. In figure 7.1 is displayed the tree structure of the software package.

7.4 System requirements

The Network Inference software has been run using a Linux operating system. Python's *multiprocessing* library does not work on Windows. Moreover, the use of a linux terminal is required when using Windows because the algorithm makes use of bash scripts. Compatibility with MacOS can be achieved by translating these scripts. The necessary components of the algorithm can be found in table 7.1.

Component	Version
CVX	2.1
numpy	1.15.4
sympy	1.3
jinja2	2.10
brian2	2.2.1
matplotlib	1.5.3
pandas	0.23.4
networks	2.2

Table 7.1: System requirements for the network inference software package

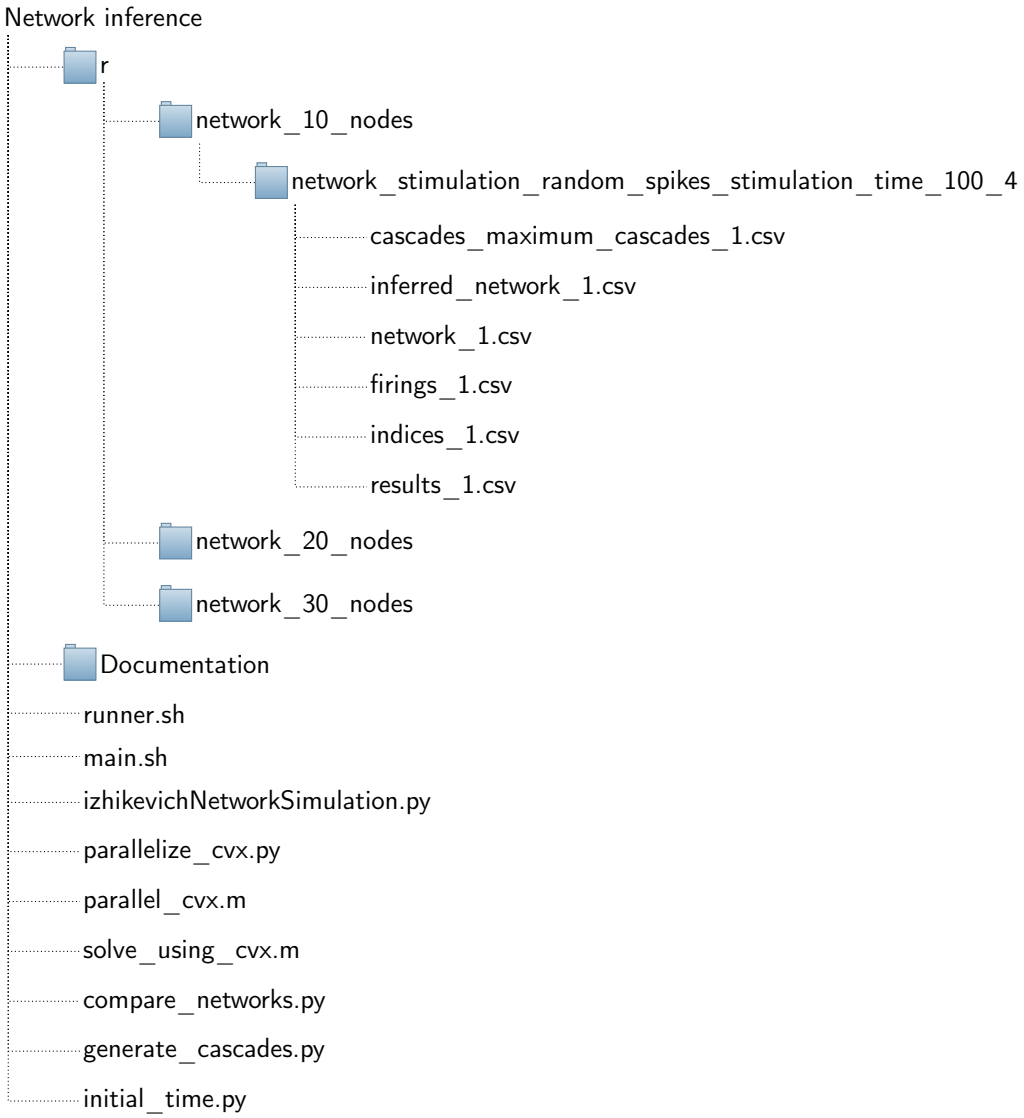


Figure 7.1: Folder architecture for the Network Inference software package

Chapter 8

Conclusion and future work

8.1 Summary of achievements

The objective of this project was to improve the state of the art of neural network inference by increasing NetRate's performance or scalability. It was also intended for this algorithm to be used for the network inference of a biological neural networks. This objective brought along many challenges that had to be dealt with beforehand such as changing the network model and updating the cascade generation method. Overall, this project has been successful at achieving the defined goals.

1. In chapter 1, a brief explanation of the project was given. It also included what the objectives were.
2. Chapter 2 explained the mathematical background behind the network inference algorithm and the previous work done in this area.
3. Chapter 3 explained how a novel method could achieve a certain level of parallelization in NetRate and increase the speed of the algorithm.
4. In chapter 4, the model for a biological neural network that mimicked a real biological one was devised. Moreover, two new cascade generation methods were proposed in order to adapt to the changes in the network model.
5. A new performance metric was proposed in chapter 5 for networks without a ground truth. Its drawbacks were analysed and illustrative results were shown.
6. In chapter 6 the suitability of the changes in the algorithm done in chapter 4 is proved. Then a biological neural network is inferred and its characteristics are analysed. Moreover, the performance metric outlined in chapter 5 is used to test the knowledge of the network. It was shown that NetRate is able to capture some network information from the real data.
7. Finally, in chapter 7, the software package used for this project is described. Moreover, the folder structure within which all the results are saved is explained.

8.2 Future work

8.2.1 Test on more recordings

In this project, only recording 4 from the 25 available was used. This was due to the fact that it was the dataset with the lowest number of neurons. An extension of this project could investigate the structure of the rest of recordings in the dataset.

8.2.2 Increase the complexity of the network model

The model of the network employed in this project consisted of only excitatory neurons. However, biological neural networks consist of many different kinds of neurons. Future work could study the effect of having inhibitory neurons in the system but inferring the connectivity of the network with the assumption of them all being excitatory.

8.2.3 Random spiking cluster model

The network model used in this project was the random spikes model. This was a very simple introductory method to mimicking biological neural networks. However, a more complex model could be studied where clusters of neurons spike at random times. Otherwise, due to the large differences yet to be solved between the real recordings and simulated network readings, more parameters can be changed so that the similarity is increased.

8.2.4 Update the spiking prediction metric

A method for evaluating the performance of NetRate on networks without a ground truth was proposed and tested in chapter 5. This is a very simple metric, however, it fails to make the best predictions possible. For this reason, an update to this metric could be proposed where not only the first neuron to spike is given, but also all the previous spikes in the recording. With such information it could be possible to estimate the membrane potential of all the neurons in the network and obtain a higher prediction score.

Bibliography

[1] Kim T. E. Olde Dubbelink et al. "Disrupted brain network topology in Parkinson's disease: a longitudinal magnetoencephalography study". In: *Brain* 137.1 (2014), pp. 197–207. ISSN: 0006-8950.

[2] S.C. Ponten, F. Bartolomei, and C.J. Stam. "Small-world networks and epilepsy: Graph theoretical analysis of intracerebrally recorded mesial temporal lobe seizures". In: *Clinical Neurophysiology* 118.4 (2007), pp. 918–927. ISSN: 1388-2457. DOI: https://doi.org/10.1016/j.clinph.2006.12.002.

[3] Pranav Malhotra. *Estimating the Topology of Networks from Distributed observations*. MEng FYP. Imperial College London, 2017.

[4] Roxana Alexandru et al. "Estimating the Topology of Neural Networks from Distributed Observations". In: *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE. 2018, pp. 420–424.

[5] Manuel Gomez Rodriguez, David Balduzzi, and Bernhard Schölkopf. "Uncovering the temporal dynamics of diffusion networks". In: *arXiv preprint arXiv:1105.0697* (2011).

[6] Eugene M Izhikevich. "Simple model of spiking neurons". In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[7] Dan Goodman and Romain Brette. "The Brian simulator". In: *Frontiers in Neuroscience* 3 (2009), p. 26. ISSN: 1662-453X. DOI: 10.3389/neuro.01.026.2009.

[8] Shinya Ito et al. "Spontaneous spiking activity of hundreds of neurons in mouse somatosensory cortex slice cultures recorded using a dense 512 electrode array. CRCNS. org". In: *CRCNS. org* (2016).

[9] Shinya Ito et al. "Large-scale, high-resolution multielectrode-array recording depicts functional network differences of cortical and hippocampal cultures". In: *PloS one* 9.8 (2014), e105324.

[10] AM Litke et al. "What does the eye tell the brain?: Development of a system for the large-scale recording of retinal output activity". In: *IEEE Transactions on Nuclear Science* 51.4 (2004), pp. 1434–1440.

[11] Michael Grant and Stephen Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. Mar. 2014.

[12] Michael Grant and Stephen Boyd. "Graph implementations for nonsmooth convex programs". In: *Recent Advances in Learning and Control*. Ed. by V. Blondel, S. Boyd, and H. Kimura. Lecture Notes in Control and Information Sciences. http://stanford.edu/~boyd/graph_dcp.html. Springer-Verlag Limited, 2008, pp. 95–110.

[13] Barry Horwitz. "The elusive concept of brain connectivity". In: *NeuroImage* 19.2 (2003), pp. 466–470. ISSN: 1053-8119. DOI: https://doi.org/10.1016/S1053-8119(03)00112-5.

[14] KJ Friston et al. "Functional connectivity: the principal-component analysis of large (PET) data sets". In: *Journal of Cerebral Blood Flow & Metabolism* 13.1 (1993), pp. 5–14.

[15] Gert Pfurtscheller and Colin Andrew. "Event-related changes of band power and coherence: methodology and interpretation". In: *Journal of clinical neurophysiology* 16.6 (1999), p. 512.

[16] Alan S Gevins et al. "Neurocognitive pattern analysis of a visuospatial task: Rapidly-shifting foci of evoked correlations between electrodes". In: *Psychophysiology* 22.1 (1985), pp. 32–43.

[17] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), pp. 500–544.

[18] Anthony N Burkitt. "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input". In: *Biological cybernetics* 95.1 (2006), pp. 1–19.

[19] Barry W Connors and Michael J Gutnick. "Intrinsic firing patterns of diverse neocortical neurons". In: *Trends in neurosciences* 13.3 (1990), pp. 99–104.

[20] Charles M Gray and David A McCormick. "Chattering cells: superficial pyramidal neurons contributing to the generation of synchronous oscillations in the visual cortex". In: *Science* 274.5284 (1996), pp. 109–113.

[21] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 137–146.

[22] Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. "SDPT3—a MATLAB software package for semidefinite programming, version 1.3". In: *Optimization methods and software* 11.1-4 (1999), pp. 545–581.

[23] Reha H Tütüncü, Kim-Chuan Toh, and Michael J Todd. "Solving semidefinite-quadratic-linear programs using SDPT3". In: *Mathematical programming* 95.2 (2003), pp. 189–217.

[24] Michael JD Powell and Y Yuan. "A recursive quadratic programming algorithm that uses differentiable exact penalty functions". In: *Mathematical Programming* 35.3 (1986), pp. 265–278.

[25] Henry Markram et al. "Reconstruction and simulation of neocortical microcircuitry". In: *Cell* 163.2 (2015), pp. 456–492.