IMPERIAL COLLEGE LONDON

ELECTRICAL AND ELECTRONIC ENGINEERING DEPARTMENT

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

May 16, 2019

Contents

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. However, little is known about how it works. By studying how the neurons structure to store and process information we can 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 where electrical spikes increase the likelihood of connected neurons to spike and therefore transmit a signal that travels as if it were a disease. By evaluating the time of "infection", the relationship between two neurons can be probabilistically estimated. After computing the relationship between all of 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.

2 Project Specification

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 in hand. If the simulation were to resemble more the behaviour of a network in a real world scenario it would provide more meaningful information as to how the algorithm would perform with measurements from the brain tissue. Finally, it is of great interest to employ the algorithm for a dataset of real neural network spikes and finding a way of measuring the performance when no ground truth is given. More information on how this will be achieved will be explained in section **??**.

3 Background

3.1 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 [11].

Functional connectivity is the temporal correlation between spatially remote neurophysiological events [12]. 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 [13], and the cross-correlation of the time series measurements from a pair of electrodes [14]. However, due to the volume conduction of brain tissue, the electrical activity from the scalp cannot infer the individual neuron behaviour below the electrode [11].

Effective connectivity was defined in [12] 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.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 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.

$$x_j = \Sigma \mathbf{W}_{ij} \times x_i \tag{1}$$

$$\frac{\mathbf{W}_{ij}}{\Sigma \mathbf{W}_{ij}} \tag{2}$$

3.2 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 [15] 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 [16] 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.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{3}$$
$$u' = a(bv - u) \tag{4}$$

with the auxiliary after-spike reseting

$$\text{if } v \geq 30\text{mV, then} \begin{cases} v & \leftarrow c \\ u & \leftarrow u + d \end{cases} \tag{5}$$

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. 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 [17, 18] 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 (thalamocortical) and the RZ (resonator). A visual representation of these neurons can be observed in figure 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 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.3 is input to the Brian Simulator. This library computes the membrane potential voltage of the interacting neurons in the

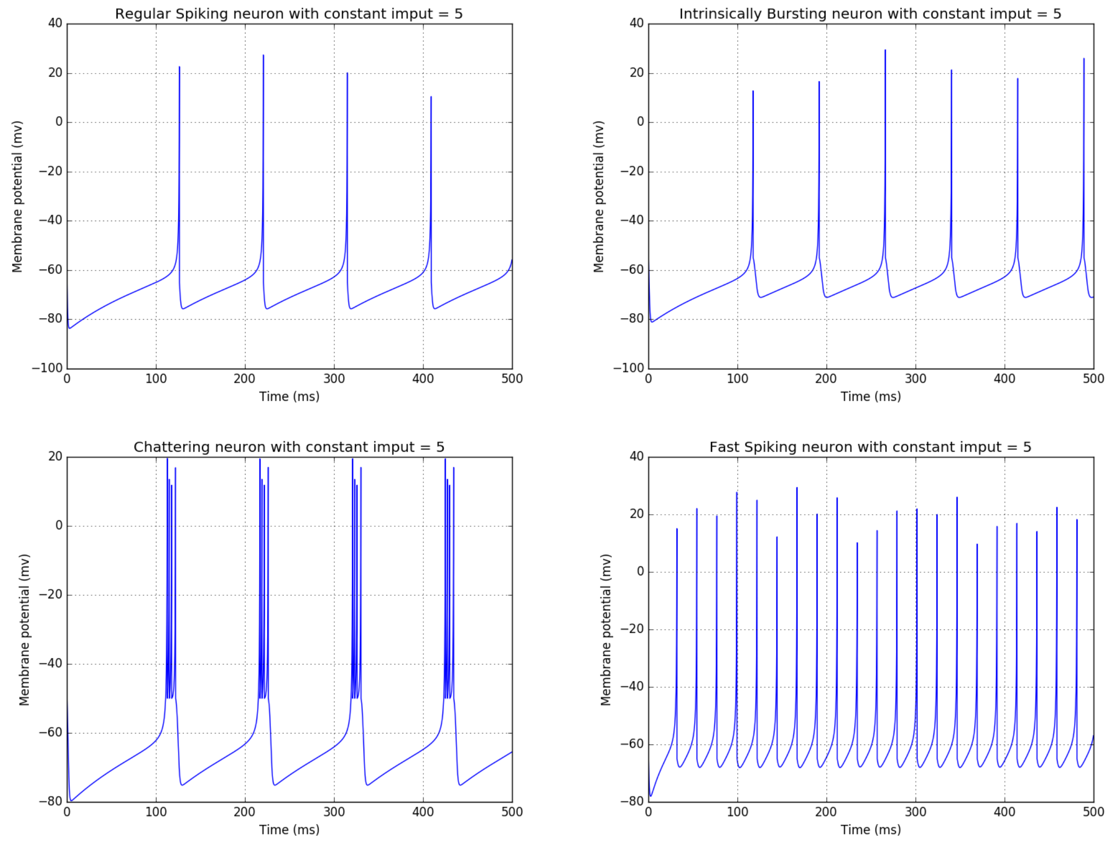


Figure 1: Types of neurons in the mammalian brain. Generated with the Brian Simulator [7] using the Izhikevich neuron model [6]

network and outputs all of their spiking times. This data will then be used to compute the NetRate algorithm.

3.3 Netrate

3.3.1 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.

3.3.2 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.

$$\mathbf{t}^c := (t_1^c, \cdots, t_N^c), \quad t_k^c \in [0, T^c] \cup \infty \tag{6}$$

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 $j \to i$:

$$S(t_i|t_j; \alpha_{j,i}) = 1 - F(t_i|t_j; \alpha_{j,i}) \tag{7}$$

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.8.

$$H(t_i|t_j; \alpha_{j,i}) = \frac{f(t_i|t_j; \alpha_{j,i})}{S(t_i|t_j; \alpha_{j,i})} \tag{8}$$

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.

3.3.3 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.9.

$$\prod_{t_k \leq T} S(T \mid t_k; \alpha_{k,i}) \tag{9}$$

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 10. The likelihood of the cascade then becomes the conditional likelihood of the infection time given the rest of the cascade.

$$f(\mathbf{t}^{\leq T};\mathbf{A}) = \prod_{t_i \leq T} f(t_i \mid t_1, \cdots, t_N \backslash t_i; \mathbf{A}) \tag{10}$$

As in [19], 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.9.

$$f(t_i \mid t_j; \alpha_{j,i}) \times \prod_{j \neq k, t_k < t_i} S(t_i \mid t_k; \alpha_{k,i}) \tag{11}$$

In this step, we calculate the conditional likelihood of Eq.10 by adding all the likelihoods of the mutually disjoint likelihoods that each potential parent is the first parent:

$$f(t_i \mid t_1, \cdots, t_N \backslash t_i; \mathbf{A}) = \sum_{j:t_j < t_i} f(t_i \mid t_j; \alpha_{j,i}) \times \prod_{j \neq k, t_k < t_i} S(t_i \mid t_k; \alpha_{k,i}) \tag{12}$$

Using Eq.10 and removing the condition $k \neq j$, the likelihood of infections then becomes:

$$f(\mathbf{t}^{\leq T};\mathbf{A}) = \prod_{t_i \leq T} \prod_{k:t_k < t_i} S(t_i \mid t_k; \alpha_{k,i}) \times \sum_{j:t_j < t_i} \frac{f(t_i \mid t_j; \alpha_{j,i})}{S(t_i \mid t_j; \alpha_{j,i})} \tag{13}$$

However, Eq.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.9 and replace the ratios from Eq.13 with hazard functions:

$$f(\mathbf{t};\mathbf{A}) = \prod_{t_i \leq T} \prod_{t_m > T} S(T \mid t_i; \alpha_{i,m}) \times \prod_{k:t_k < t_i} S(t_i \mid t_k; \alpha_{k,i}) \sum_{j:t_j < t_i} H(t_i \mid t_j; \alpha_{j,i}) \tag{14}$$

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.14:

$$\prod_{\mathbf{t}^c \in C} f(\mathbf{t}^c; \mathbf{A}) \tag{15}$$

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.

$$\text{minimize}_{\mathbf{A}} - \sum_{c \in C} \log f(\mathbf{t};\mathbf{A}) \tag{16a}$$

$$\text{subject to } \alpha_{j,i} \geq 0, i,j = 1, \cdots, N, i \neq j, \tag{16b}$$

.

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.16 found in [5] is given by Eq.17a. The survival and hazard functions are concave in the parameter(s) of the transmission likelihoods and, therefore, convexity of Eq.16 follows from linearity. The network inference problem from Eq.16 is thus convex for Power-Law, Rayleigh and Exponential models of the likelihood function.

$$L(\{\mathbf{t}^1 \cdots \mathbf{t}^{|C|}\}; \mathbf{A}) = \sum_c \Psi_1(\mathbf{t}^c; \mathbf{A}) + \Psi_2(\mathbf{t}^c; \mathbf{A}) + \Psi_3(\mathbf{t}^c; \mathbf{A}) \tag{17a}$$

$$\Psi_1(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \sum_{t_m > T} \log S(T \mid t_i; \alpha_{i,m}) \tag{17b}$$

$$\Psi_2(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \sum_{j:t_j < t_i} \log S(t_i \mid t_j; \alpha_{j,i}) \tag{17c}$$

$$\Psi_3(\mathbf{t}^c; \mathbf{A}) = \sum_{i:t_i \leq T} \log(\sum_{j:t_j < t_i} H(t_i \mid t_j; \alpha_{j,i})) \tag{17d}$$

The terms in Eq.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.

3.3.4 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.6 would look like this:

$$\mathbf{t}^c = \{\infty, 3, \infty, \infty, 0, 5\} \tag{18}$$

Remember from Eq.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.

3.3.5 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.

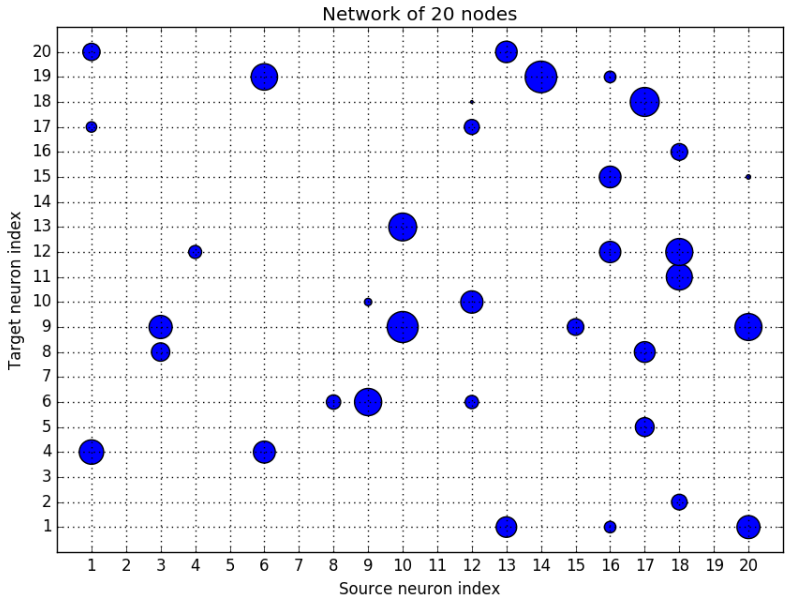


Figure 2: Adjacency matrix of a network of 20 nodes

3.4 Biological Neural Network

3.4.1 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 Fig.2 an adjacency matrix of a network of 20 nodes can be observed. When $\alpha_{j,i}^{BNN} \neq 0$, 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 $\alpha_{j,i}^{BNN}$ 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 $\alpha_{j,i}^{BNN}$, everytime j spikes, it causes the membrane potential v from Eq.3 to increase by $\alpha_{j,i}^{BNN}$. If neuron i crosses the threshold of 30mV, it spikes. This phenomenon is proved in [4] and explained in section 3.4.2. Using the example in figure 2, when node 10 spikes, it causes nodes 13 and 9 to increase by $\alpha_{10,13}^{BNN}$ and $\alpha_{10,9}^{BNN}$, respectively.

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.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.

3.4.2 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.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 (i.e 1, 3 and 5; 3 and 5; and 5). However, this breaks the requirement of independence of cascades seen in section 3.3.1, where no same spike can appear in two cascades. For this reason, a well studied approach was required for the selection of cascades.

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.

[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.

	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 1: Results for network inference obtained in [4]

3.4.3 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 3.3.4, 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.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.

3.4.4 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] can be seen in table 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].

4 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[2]. 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 [20, 21] (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 [22] 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.

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.

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

4.1 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 the processes 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 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 required for both of them to be 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.

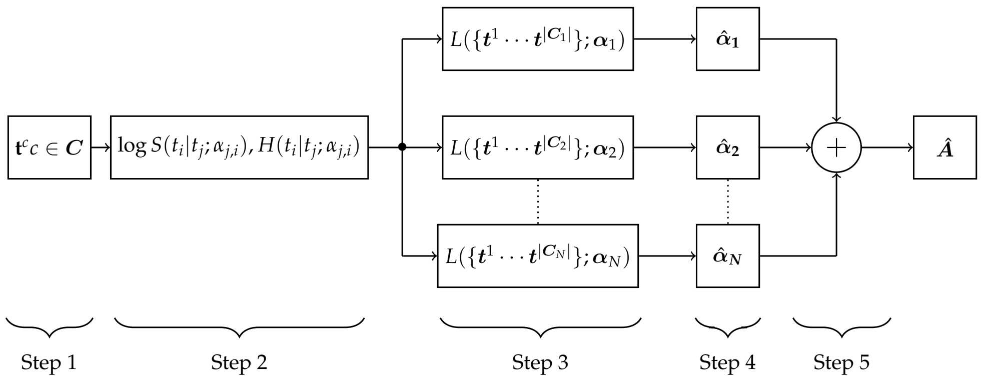

$\mathbf{t}^c c \in \boldsymbol{C}$
$\log S(t_i|t_j;\alpha_{j,i}), H(t_i|t_j;\alpha_{j,i})$
$L(\{\boldsymbol{t}^1 \cdots \boldsymbol{t}^{|C_1|}\};\boldsymbol{\alpha}_1)$
$L(\{\boldsymbol{t}^1 \cdots \boldsymbol{t}^{|C_2|}\};\boldsymbol{\alpha}_2)$
$L(\{\boldsymbol{t}^1 \cdots \boldsymbol{t}^{|C_N|}\};\boldsymbol{\alpha}_N)$
$\hat{\boldsymbol{\alpha}}_1$
$\hat{\boldsymbol{\alpha}}_2$
$\hat{\boldsymbol{\alpha}}_N$
$+$
$\hat{\boldsymbol{A}}$
Step 1
Step 2
Step 3
Step 4
Step 5

Figure 3: 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.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 figure 19As an illustration, a network of 10 neurons is simulated, and the number of cascades belonging to each node is shown in Eq.19.

$$NoC = \{12, 21, 62, 166, 21, 67, 17, 30, 18, 81\} \tag{19}$$

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 computed the cascades 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.20.

$$argmin_n \; p_i + NoC_n, \tag{20}$$

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.19 with $N = 10$, the distribution among $M = 4$ processors becomes:

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

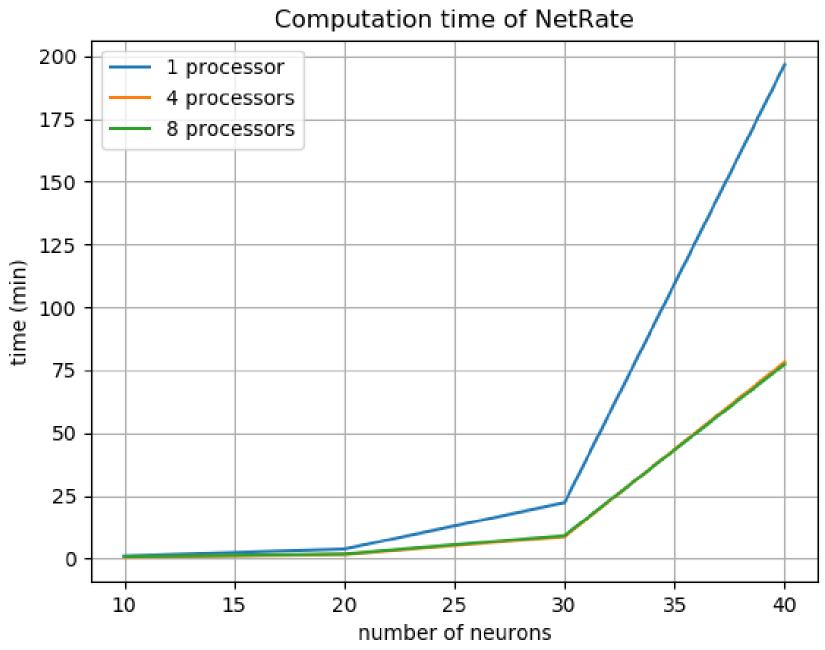
The resulting number of cascades for each of the processors becomes 166, 114, 106 and 109, respectively. This is a more balanced distribution than if the nodes had been assigned in any other way. Now, it remains to clarify in which order they must be computed. One constraint that limits the algorithm 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. Thus, it becomes prohibitive to use several processors at the same time for large networks. Each of the processors requires its own memory to perform NetRate in parallel. 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.

4.2 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 **??**. 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 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, where the optimization problem in 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 had been a significant improvement in the speed of NetRate, it was not as good as desired, and it also posed further restrictions on memory usage.

References

[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] 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`.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.