University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

Inference of Neuronal Network Structure and Dynamics using Machine Learning

By Henry Porter

27/08/2021

Supervisor: Dr Markus Brede

Second Examiner: Dr Srinandan Dasmahapatra

A dissertation submitted in partial fulfilment of the degree of MSc Data Science

Abstract

The ability to estimate the connectivity of brain regions has proven valuable in helping to categorise brain function and classify certain brain types or pathologies. Recent advances in machine learning have led to the development of techniques which can make use of the large amount of data generated from electroencephalogram (EEG) scans and in turn use it to understand brain topology and function.

This report gives a comprehensive review of current literature concerning how machine learning and statistical techniques have been used in neuroscience, techniques for network inference and which mathematical models are viable for representing brain functionality. Next, a method for generating a synthetic neuronal network dataset is described alongside a variety of methods for performing network inference. The methods are tested on the synthetic dataset in a range of scenarios, testing their robustness to data sparsity, noise and the amount of synchronisation present in the network.

This work demonstrates how a neural network and closed form solution both perform excellently for network reconstruction. The random forest method also exhibited relatively strong robustness when faced with noisy or highly synchronised network data.

Contents

List of Figures

1 Introduction

1.1 Background

Many real world entities can be described through the mathematical framework of graph theory, comprising vertices which represent a set of objects and edges which define how each vertex is connected to the other vertices in the network [1]. Being able to understand and characterise these networks provides both a means of predicting the propagation of information within a system as well as insight into how one might control the spread of that information. For example, the knowledge of how a population is interconnected may provide a better understanding of how disease spreads within that population [2].

'Complex networks' exhibit non-trivial properties and are generally harder to characterise and therefore study. These types of networks are more prevalent in real world cases and occur in a multitude of biological, technological and social settings [3].

Due to the size and intricacies of many complex networks, the real world data describing them may be incomplete, and they are often difficult to characterise with simple models in graph theory. This makes them 'black boxes', where certain information about the state of each node in the network can be observed (such as whether an individual in a population has a disease or not), but the underlying structure of the network and dynamics determining how information propagates (such as how a disease may spread) are unknown quantities.

Network inference is a data driven approach for understanding complex networks; to use the phenotype of intricate systems in deducing which underlying mathematical structure is responsible for governing how such systems behave.

In the case of neuroscience, being able to mathematically describe the interactions of regions of the brain can provide significant steps towards understanding the brain and predicting patterns of disease in neuronal networks.

1.2 Project Aims

The objective of this paper is to assess the performance of a variety of machine learning techniques on their ability to perform network inference from time series data resembling electroencephalogram (EEG) scans for various regions of the brain. Existing techniques for retrieving the topology of networks will be researched and adapted for the case of EEG brain scans. These techniques will then be tested against one another to deduce how accurately

the strengths of the connections in a network can be inferred from EEG data. In addition, the models will be evaluated on their robustness to varying levels of noise and on how well they perform as a number of parameters dictating the behaviour of the network are changed.

In order to deploy a range of scenarios in which to evaluate the machine learning models, a necessary objective prior to the development of any network inference techniques will be to develop a way of generating plausible synthetic time series data mimicking brain activity for an EEG scan of the brain over a period of time.

The development of machine learning techniques has played a vital part in the development of a plethora of disciplines and problems. In an era when we have more data than ever and more sophisticated technology year on year, machine learning utilises both of these advances in being able to analyse a lot of data and find meaningful trends within it at a speed unobtainable with previous technologies.

The brain is one of the most complex dynamic systems ever to be observed [4]. Whilst we know much about the function of individual neurons and the role which certain regions of the brain play, many of the observed functions of the brain, such as how information is categorised or how the brain can focus attention are unknown. Counter-intuitively these enigmas are described as 'easy problems' in neuroscience, with 'The Hard Problem' being to fully understand consciousness itself [5].

The complexity of the brain (consisting of approximately 86 billion neurons [6]) means that many of the 'easy' problems lend themselves to machine learning as a way of solving them. Machine learning has already shown its value in accelerating the rate at which we are understanding some of the most complex and data rich problems in biology. In July of 2021, it was reported that DeepMind AI had discovered the structure of 98.5% of all the proteins in the human body; prior to this, decades of research had led to the discovery of a comparatively small 17% of proteins in the human body [7].

1.3 Report Structure

This paper first reviews background research into existing techniques for network topology and dynamics inference in addition to research into how a synthetic data set resembling EEG brain scans may be implemented (Chapter 2). The methodology of the techniques selected and adapted to achieve the objectives of this project are discussed in Chapter 3, alongside an explanation of how the data set was generated. Chapter 4 compares the developed techniques, presenting their performances on a variety of different data sets. The results are then valuated and the limitations of the research approaches

discussed. The paper concludes with the outcomes of the research and what future research might entail.

2 Literature Review

The rapid development of machine learning methods as well as the technology required to implement them has proved to be a catalyst for the progression of many areas in neuroscience. In addition to providing a greater understanding of how our brains function, machine learning has also led to a multitude of real world technologies being developed to combat neurological pathology. The following sections will give a brief overview into how machine learning has aided the development of neuroscience followed by an evaluation of existing techniques for performing network inference together with existing research which would aid the development of a credible synthetic dataset resembling the readings taken from EEG brain scans.

2.1 Machine learning in neuroscience

In recent times several projects have been set up with the aim of collating a large number of neuroimaging datasets, notably the Human Connectome Project [8] and Imagen [9]. Studies examining neuroimage data are often focused on identifying certain diseases from images alone. The predominance of large samples of high dimensional data in these studies has made machine learning techniques invaluable tools for drawing meaningful conclusions from the data.

The work of Adar et al. [10] used magnetic resonance imaging (MRI) scans of dementia patients in order to distinguish between Alzheimer's disease, frontotemporal dementia, and vascular dementia. The method used involves first preprocessing the scans to retrieve values for a number of key features (such as the grey matter volume). Then a combination of a naive Bayes classifier and support vector machine is used to determine which of these image features are the best predictors for a specific type of dementia.

Deep learning models have furthered the use of machine learning in neuroimaging. In this field, the input data is inherently high dimensional and contains non-linearities for which previous machine learning techniques would have required significant data preprocessing. The literary review [11] carried out at the University of Newcastle demonstrates the success of deep models, in taking only image data as an input and being able to diagnose early stage Alzheimer's in patients, alluding to a future where the diagnosis of certain brain diseases could be made more efficient with the use of machine learning.

Deep learning has also proved invaluable in helping to develop prosthetics. The somatosensory system is a complex network of neurons and neural pathways responsible for how our brains can interpret external stimuli such

as touch [12]. In cases where there has been damage to this network, through events such as a stroke or injury, being able to 'relearn' these connections, or provide prosthetics capable of mimicking previous somatosensory connections has proven a complex problem. The work of A. Loutit and J. Potas [13] demonstrates the effectiveness of supervised back-propagation neural networks in being able to decode somatosensory stimuli which could lead to a huge enhancement of tactile feedback in prosthetics.

Whilst these examples do not provide a background into the state of network inference for neuronal networks, they give a good context for the sort of problems that neuronal network inference could combat. In being able to determine the connectivity of particular brain regions, further strides may be possible in distinguishing types of dementia as well as detecting which regions of the somatosensory network become more connected for different types of required movement: this could aid the design of prosthetics. The examples also demonstrate that deep learning methods have been particularly useful in neuroscience, perhaps due to the complexity of the trends in the data taken from brains which more traditional machine learning methods find harder to exploit.

2.2 Network Inference

The field of network inference is extensive due to the many different areas in which graph theory offers a good mathematical framework for a problem. Inference problems tend to fall into one of the following categories:

- determining the topology of a network given information from all the nodes in a network

-determining the topology of a network given information from a partial set of nodes in a network and

-determining both the topology and underlying dynamics of a how a network operates given information from either a whole or partial set of nodes from the network [12].

Both statistical and machine learning approaches have been utilised to solve these problems.

2.2.1 Statistical approaches

A large proportion of statistical approaches to network inference seek to solve the problem through static data (where no assumptions about the underlying network dynamics are made).

Gene regulatory networks (GRNs) describe the interactions between each gene in determining the expression profile of a cell. The ability to successfully

infer the structure of GRNs would allow a much greater understanding of the role of each gene in a cell's genome as well as playing a central role in the advancement of morphogenesis, the synthetic development of body parts [13]. Much of the literature surrounding network inference has been with the aim of understanding gene regulatory networks.

Mutual information networks have been developed as a solution to infer the structure of GRNs. These techniques use pairwise mutual information to determine if any two nodes in a network are connected. ARACNe was an algorithm developed for GRNs using a mutual information approach [14]. Whilst this method can be effective with only a few number of samples and little computational power, making link predictions based on mutual information means that nodes could be highly co-regulated and therefore express a high level of mutual information due to intermediary nodes dictating their expression. This could prove particularly problematic in translating the technique for use on brain networks as nodes may have a high level of synchronization without being directly linked. Mutual information techniques also give no insight into the dynamics of a network.

N. Kiani et al. [15] looks into a modification on mutual information techniques, Gaussian graphical methods (GGMs). Instead of using a correlation matrix to determine network topology, GGMs use a precision matrix for link prediction, which takes into account the influence of potential intermediary nodes and therefore helps to deal with the issue of high co-regulation. However, GGMs still fail in networks which are inherently highly synchronised as they rely on a (relatively small) number of cases which show when two co-regulated nodes display little correlation.

A statistical inference technique which may be more suited to the domain of neuronal network prediction is that of probabilistic parameter estimation for a model describing the dynamics of a network. The work of Nachman et al. [16] makes an asuumption of the dynamics of a gene regulatory network and then uses a Bayesian network approach to approximate the parameters of the equations describing those dynamics. This method allows for both the topology and dynamics of a network to be inferred (as the entries in the adjacency matrix would be parameters in the dynamics equations for the system).

Similarly Kim and Leskovec [17] use a method of model fitting, through the framework of expectation maximisation, to solve the network completion problem (being able to infer the existence of missing nodes in a network). At each iteration of the algorithm, a data set is sampled using Metropolized Gibbs sampling [18] and fed through a model whose parameters follow a probabilistic distribution. These distributions are then adjusted based on how well the model predicts the target variable. When compared to other

statistical methods, the main issues with model fitting methods arise from the need for prior knowledge of the system being observed.

2.2.2 Machine learning approaches

As a continuation of the last statistical network inference technique discussed, most machine learning methods make an assumption about the underlying dynamics of a network and then use machine learning techniques to find the optimal parameters for the dynamic's equations.

Shandilya [19] sets up a problem requiring the estimation of parameters for a system, where the state of each of the nodes in the system can be observed, and their rates of change linearly approximated. The problem's solution is to minimize the difference between the rates of change of the nodes in a network and some linear combination of known but arbitrarily complex functions of node values; the weights involved in the linear combination represent the parameters in the model being estimated. Formulating the network inference problem as a linear regression problem has a number of solutions for which many machine learning techniques have been developed. Shandilya provides a solution through a closed form method, producing a solution via only one equation and requiring the calculation of the pseudo-inverse of a data matrix.

M. Panaggio et al. [20] expand on this closed form solution in estimating the connectivity of networks of coupled oscillators. The work is particularly relevant to the case of network inference for neuronal networks. The paper describes how a synthetic data set is constructed using Kuramoto dynamics equations, which is a good approximator of neuronal dynamics [21]. Whilst the work demonstrates very high reconstruction accuracy (for a range of scenarios including varying levels of noise) it fails to demonstrate how the model deals with smaller sample sizes or different levels of synchronisation in the network.

Rather than making an assumption about the dynamics equations for a network, the ICON technique [22] formulates a method for predicting the system dynamics using a number of basis functions such as a Fourier series, and estimates the weights for these functions using the pseudo-inverse solution described above.

The ICON technique has been successfully utilised to predict the structure of and dynamics of neuronal networks [4]. It then uses the connectivity values for a number of different brains in attempting to predict epileptic seizures, reporting an accuracy of 93.8% in identifying seizures. This paper shows an extremely impactful use of network inference in neuroscience. However, since it only evaluates the ICON method on real world data, the 'true'

connections of the network being inferred remain unknown. Therefore, there is no evaluation of how accurately the ICON method retrieves the true values for the parameters of dynamics equations describing neuronal networks, nor is there testing of the method with changes to a number of parameters such as the data noise, synchronisation or the gaps in data samples (as this is negatively impacts the linear approximation of gradients which the method makes use of).

Closed form solutions to the network inference problem have been proven to be effective, however they all require both a rough idea of the dynamics dictating the spread of information in a network and an accurate approximation of the rate of change of node states.

The Gumbel Graph Network (GGN) [23] takes only the time series data of node values to train two neural networks: one to learn the adjacency matrix weights, and another to learn the dynamics of the system. Each piece of data in the time series is fed through the GGN to predict what the next set of network data will be. The difference between the predicted data and true data values is calculated and then used to update the weights in the network via back propagation. This technique has a number of benefits. Firstly, by giving the algorithm only the node values at each timestep, no extra data preprocessing is required (such as calculating a gradient approximation). By making use of a neural network, the GGN can exploit non-linearities to understand the complexities in both the relationships between nodes, and in the dynamic equations determining the state change of a node. Finally, the algorithm requires no assumptions about the dynamics of the system before training; whilst in many cases, there will be some knowledge of this, it is still a powerful feature of the GGN. The key issue in the algorithm itself is the lack of interpretability; although the weights of the neural network dedicated to finding the adjacency matrix may be easy to understand, interpreting the combination of a number of common neural network activation functions may be less intuitive. Z. Zhang et al. evaluate the GGN trained on a data set comprising coupled oscillators, whose dynamics are determined by Kuramoto dynamics; it achieved 100% reconstruction accuracy, but there is very little information on how the GGN performs with noisy, sparse or highly synchronised data.

M. Chen et al. [12] make further developments to the GGN for use of the network reconstruction problem. For a range of data sets, the network completion GGN (NC-GGN) was able to completely learn the topology of networks with only 90% of nodes available to observe. The NC-GGN may be useful in gaining an extremely insightful view into brain functionality by observing only a limited portion of it. However, for existing uses of brain inference techniques, the implications of the NC-GGN are less easy to re-

alise; in most cases, the purpose of inferring neuronal network structure is to understand how connected particular regions are in being able to diagnose particular pathologies, not to infer the existence of missing regions.

Feature selection methods have also been studied for inferring the topology of networks. V. Huynh-Thu et al. [24] study tree based ensemble methods for determining the structure of GRNs. The nodes in a network are used here as features in the regression trees algorithm. An individual binary tree splits the data based on some arbitrary function of a particular node (gene) in the network at each level of the tree. An optimal tree would minimise the variance within the subsamples at each step as the eventual result of such a tree would group identical sample values. Since each tree is randomised, many trees are created and a weighted average is taken based on their performance. Feature selection works by determining which genes consistently lead to the minimising the within sample variance. The most prominent features dictating the values for any one node are selected as the neighbours for that node. Tree-based ensemble methods excel in being generally robust and easy to interpret. However, for the case of neuronal networks, continuous adjacency matrix entries may be harder to interpret as the algorithm is not set up to find specific adjacency matrix values, but rather to determine how well particular nodes can be used as predictors for the values of other nodes.

Genetic algorithms are a group of machine learning algorithms inspired by natural selection [25]. They work by taking a population (set of possible solutions), testing their 'fitness' by seeing how low they score on a loss function and then culling the 'weaker' members of the population and 'breeding' the members of the population which scored lower on the loss function. The genetic algorithm-based Boolean network inference (GABNI) [26] looks into using a combination of both boolean network inference and a genetic algorithm to find the topology of a GRN. Similarly to the statistical methods, GABNI first starts by inferring network topology from how correlated any two nodes are with boolean network inference. A genetic algorithm is then applied to search through a range of solutions to find the optimal one. The results presented show how GABNI outperformed a range of other statistical techniques including ARACNe [14]. GABNI shows how machine learning techniques and statistical techniques could be used to complement one another. Generating a 'population' of solutions based on correlative measurements and then searching through that generated solution set may be useful in combating highly synchronised networks, which would cause issues for more simple statistical methods. Despite this, without any testing on a more complex data set, it remains unclear as to whether the introduction of a genetic algorithm would remedy the issues apparent in other statistical methods.

2.3 Synthesising EEG brain scan data

Being able to understand the structure and dynamics of the brain has been the subject of scientific debate for centuries. Even as far back as the 4th century B.C, Aristotle believed that the brain was a supplementary organ to the heart in regulating its heat [26]. More recent scientific discoveries have led to a model in which we have some understanding of the role of particular sub areas of the brain, such as the cerebellum or frontal lobe [27]. There is also extensive work dedicated to understanding the cells which make up the brain such as neurons.

It was only as recently as 2005 when a concerted, global effort to understand the connectivity of these neurons and brain regions for humans in the domain of graph theory when Spoorns et al. conceptualised the human connectome project [8]. The initial paper introducing the project [28] proposed a method to collate a large range of human brain data with the aim of understanding its connectivity.

The Kuramoto model, developed by Yoshiki Kuramoto, is used to model of N coupled oscillators [29]. It describes the rate of change of the i^{th} oscillator in a system as being dependent on both its own natural phase ω_i and the sum of the sinusoidal differences between its own phase and the phases of its neighbouring nodes in the network as shown in Equation 1.

$$\dot{\theta}_i = \omega_i + \frac{K}{N} \sum_{j=1}^{N} \mathbf{A}_{ij} sin(\theta_j - \theta_i) \tag{1}$$

K is known as the coupling constant and determines how synchronised the system becomes and $\mathbf{A}_{ij}$ is the coupling strength between nodes i and j. Several works have shown that the Kuramoto model is a good choice of model for brain regions. The work of J. Cabral et al. [30] analyses a number of models in their ability to replicate brain functionality. The work is not conclusive as to which model is best, but suggests that data generated with the Kuramoto model shows a good correlation with real world data whilst also being relatively simple.

The Kuramoto model also gives a measure of synchronization between nodes in a network. Studying the synchronisation of brain regions may help in understanding previously unknown links determining the functionality of the brain. The work of Gomez-Gardenes et al. [31] studies the synchronisation of a number of regions in a cat brain. The work is specifically interested in the evolution of synchronisation across time series data for the brain. The author comments on the Kuramoto model providing a method for understanding brain synchronisation where previously prevalent models for describing neuronal dynamics either could not, or did so poorly. This work

demonstrates why the Kuramoto model could prove useful in any study requiring synthetically generated data for brain activity. It also shows how the synchronisation measurement provided by the Kuramoto model provides an extra way of describing the brain which may help in the classification of certain brain types and the diagnosis of any neurological disease.

3 Methodology

The current state of network inference, as described in the literature review, demonstrates a broad range of both machine learning and statistical approaches to the problem domain. Several successful attempts have been made to estimate brain network structure and function [22] and the ability to infer network connectivity has even been successfully applied to a real world case in identifying brain pathology [4].

In conducting the literature review, it was apparent that comparative research was lacking on how well different techniques performed. This is perhaps due to how bespoke most techniques are to their specific sub-domain of network inference. Nor is there an extensive amount of testing of any one technique into its robustness to noise, data sparsity or how synchronised a network is.

The goal of the research presented in this paper is therefore to provide a quantitative comparison of a variety of network inference techniques on their ability to estimate the connectivity of neuronal networks as well as their robustness to a number of factors including noise.

The structure of the rest of this section will be as follows: First, a description of how a synthetic data set will be generated is presented. Following this, a problem statement will be formulated, giving a standardised way of testing a range of inference methods. Finally, the list of inference models being implemented and tested will be described in full, alongside the motivations for why they were chosen.

3.1 Data Generation

In order to test the performance of multiple network inference techniques, the formulation of a supervised learning problem is required: the true weightings of the adjacency matrix will have to be known in order to test the techniques [32]. For real world neuronal network data, this adjacency matrix is not fully known, so an accurate measure of the performance of inference techniques is difficult. In addition to this, systematically tuning the variables responsible for generating real world data is hard, and harder still for data which has already been gathered (apart from adding synthetic measurement noise). For this reason, the decision was made to synthesise a data set resembling a neuronal network, for which there would be greater control on a number of factors including the amount of data generated and parameters dictating the generation of the data.

The literature review describes how Kuramoto dynamics have provided

a good approximation for the interaction of brain regions [30][31] as well as showing it being a successful model to complement the network inference of neuronal networks in a real world setting [4].

For N oscillators (nodes) in a network, the differential equation for the phase of the i^{th} oscillator is shown in Equation 1. w_i is the natural frequency of the oscillator, K is the coupling constant (if K is higher then the network tends to be more synchronised) for the network and A_{ij} is the coupling strength between nodes i and j. The matrix $\mathbf{A}$ is therefore the network's adjacency matrix being approximated by the inference algorithm.

A noise parameter can also be added to the model to give:

$$\dot{\theta}_i = \omega_i + \frac{K}{N}\sum_{j=1}^{N} \mathbf{A}_{ij} sin(\theta_j - \theta_i) + \xi_i(t) \tag{2}$$

Where

$$\xi_i(t) = \lambda X \delta_{i,j} \Delta t \tag{3}$$

Here, λ is the strength of the noise and δ_{ij} is 0 for i=j (so that no noise is added when considering the link between a node and itself) and 1 for i≠j. X is a random variable from a Gaussian distribution of zero mean and unit variance (the system noise is therefore uncorrelated white Gaussian noise). Δt is the time step size between two measurements of the system.

Before generating data, the 'true' adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is randomly generated. For the project, a binary matrix was used to test all of the methods to give a clear indication of any large failures of the methods. The process of data generation requires generating continuous data describing the phase of each oscillator over a period of time T using Equation 2. In order to use the data to train a network inference algorithm, the generated data is then sampled at M evenly spaced time intervals to get a data matrix $\mathbf{D} \in \mathbb{R}^{N \times M}$. Further Gaussian noise could be added to each entry in the data matrix representing some measurement error, however this project is concerned with the performance of network inference models. Many preprocessing techniques exist to deal with measurement error so this form of error was not included when generating data as it would take away from focussing on the performance of the inference techniques.

Although increasing the value of K in equation 2 increases the level of synchronisation in the network, an actual measure of a network's synchronisation at any one time can be calculated via the Kuramoto order parameter (see Equation 4).

$$R(t) := \frac{1}{N} \mid \sum_{j=1}^{N} e^{i\theta_j(t)} \mid \tag{4}$$

Where R takes a value between 0 and 1. R is 1 if the network is completely synchronised (all nodes are at the same phase). This model provides a means of generating an arbitrarily large data set, whose noise and synchronisation parameters are adjustable. With a method of generating the data in place, a mathematical formulation of the problem can be made.

3.2 Problem Statement

Given time series data, generated using Kuramoto dynamics (see Equation 2), the high level problem is to be able to retrieve the adjacency matrix $\mathbf{A}$ from this data. For data with N nodes and M time steps, the data matrix takes the shape $\mathbf{D} \in \mathbb{R}^{N \times M}$. If an assumption is made that the data has been generated with the Kuramoto model, then the problem can be split into N sub-problems:

For each node in the network, the goal is to find the set of weights in the adjacency matrix (the i^{th} row in $\mathbf{A}$), denoted $\hat{\mathbf{A}}_i$ which minimises:

$$(\dot{\boldsymbol{\theta}}_{i,m} - (\omega_i + \frac{K}{N} \sum_{j=1}^{N} \hat{\mathbf{A}}_{ij} sin(\theta_{j,m} - \theta_{i,m}) + \xi_i(t)))^2 \tag{5}$$

Here $\boldsymbol{\theta}_{i,m}$ is the rate of change of the phase for the i^{th} oscillator for the m^{th} timestep. Therefore, $\hat{\mathbf{A}}_i$ is the only part of the adjacency matrix needed to be predicted to determine the values for the i^{th} node in the network.

Over every piece of available data, the loss function becomes:

$$\mathbf{L}_i := \sum_{m=1}^{M} (\dot{\boldsymbol{\theta}}_{i,m} - (\omega_i + \frac{K}{N} \sum_{j=1}^{N} \hat{\mathbf{A}}_{ij} sin(\theta_{j,m} - \theta_{i,m}) + \xi_i(t)))^2 \tag{6}$$

This can be reformulated in a way more suitable for machine learning and statistical techniques to solve.

In reality, the true values of the gradients are not observable, so an approximation must be made from the observed phases of the nodes.

Let $\dot{\mathbf{X}} \in \mathbb{R}^{N \times M}$ be the matrix of the approximated gradients of the phases of the oscillators at each time step. The gradients are estimated using a linear approximation:

$$\dot{\mathbf{X}}_{i,m} := \frac{\mathbf{D}_{i,m} - \mathbf{D}_{i,m-1}}{t_m - t_{m-1}} \tag{7}$$

Let $\mathbf{Y} \in \mathbb{R}^{N \times M}$ be the tensor defined such that

$$\mathbf{Y}_{i,m} := \dot{\mathbf{X}}_{i,m} - \omega_i \tag{8}$$

And $\mathbf{G} \in \mathbb{R}^{N \times M \times N}$ be the matrix with entries defined with

$$\mathbf{G}_{i,m,j} := \frac{K}{N} sin(\theta_{i,m} - \theta_{j,m}) \tag{9}$$

With these new definitions, the loss function from Equation 6 for all nodes can be rewritten:

$$\mathbf{L} := \sum_{m=1}^{M} (\mathbf{Y} - \hat{\mathbf{A}}\mathbf{G})^2 \tag{10}$$

In this instance, where an assumption about the underlying dynamics of the system is made, inferring the network structure involves solving N multilinear regression problems, by finding the best $\hat{\mathbf{A}}_i$ for each node and combining these N solutions to form an estimation of the adjacency matrix [33].

This is a valuable formulation of the original problem; by simplifying it to a linear regression problem, it lends itself to a multitude of machine learning and statistical algorithms which have been developed to solve such problems. All of the techniques discussed in the following section are purpose built to solve this expression of the network inference task.

3.3 Methods

3.3.1 Closed Form Solution

The first method to be developed for finding a solution to the loss function (see Equation 10) is a relatively simple solution. It finds the pseudo-inverse of the matrix $\mathbf{G}$ to rearrange the loss function for finding $\hat{\mathbf{A}}_i$ directly:

$$\hat{\mathbf{A}}_i = \mathbf{Y}_i \mathbf{G}_i^T (\mathbf{G}_i \mathbf{G}_i^T)^{-1} \tag{11}$$

This method is appropriate in that it is simple, requiring only one calculation to be made. It is also proven to be effective in a number of papers [19]-[22] and [4]. However it is not made entirely clear in its previous implementations, under which circumstances it is most effective or when it fails. Theoretically, this analytical method should achieve complete network construction for perfect measurements of the rate of change of the phase of each network oscillator. However, as has been stated, this is not possible in reality. The premise of this research is to see which methods fare best when only

an estimate of the gradient is available and this provides the motivation for testing methods other than this closed form solution.

In implementing this closed form solution, it is a necessary step to find the inverse of the matrix G, which through its construction, has a determinant of 0 (having a leading diagonal of zero entries). This in turn makes it a singular matrix, one whose determinant is zero meaning the inverse is impossible to find. There are several methods for dealing with this, however a fairly easy method is to add a very small value to the leading diagonal of 0's in the matrix.

Another bonus of this method is that, if needed, it could be developed for an online learning domain; where data is only available to train a model one piece at a time. This may be more useful in a realistic setting, where data comes through piece by piece. The adaptation made to the original algorithm for online learning would be to change the algorithm to a recursive least squares solution [34].

3.3.2 Random Forest

Decision trees take in a data set and progressively split it into partitions based off of a particular rule. For a data set with n features, each node of the tree splits the data based on a rule for one of those n features. For supervised regression problems, the strength of a decision tree is measured by its ability to group together similar target variables using the feature variables. One of the biggest issues with a single decision tree is that it can be quite unstable, discreetly grouping data on a finite number of rules means that only a small change in a feature variable can drastically change the classification of the piece of data it represents. The ultimate goal of this method is to reduce the within class variance of the data partitions at the end of the decision tree.

Ensemble methods help to deal with this issue by averaging the results of several learners, thus reducing the variance and improving generalisation. The random forest method is an ensemble method for decision trees.

In the context of finding an optimal solution for Equation 9, the decision trees split the data in the matrix $\mathbf{G}_i$ based on one of the N features for each entry in the matrix. The random forest takes many of these trees and averages the results. The performance of each tree is evaluated based on how closely data with similar target variables (represented by the values in the matrix Y) are grouped together (see Figure 1).

Normally, decision trees are not ideal as predictors for continuous data as they group data into only a finite number of partitions. However, for this problem, trees are more appropriate as they are used as a feature selector. As previously stated, each node in a tree splits the data. The predictive

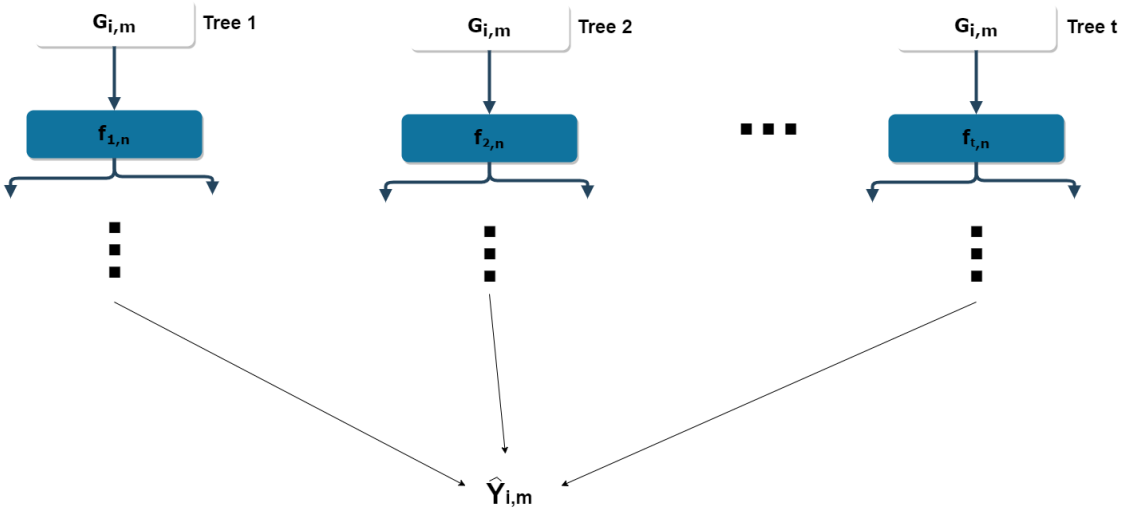

Figure 1: A representation of the random trees method. In making a prediction for the gradient of the i^{th} node at time m, a weighted average of the prediction of t many decision trees is taken. Each Decision tree splits the data $G_{i,m}$ based off of a number of functions based on the n^{th} node in the data at each level of the tree

value of each node is determined by how much it minimises the within class variance of the target variables for the resulting partitions of the data. In the random forest, the average predictive value for each feature gives a value for the connectivity of the i^{th} node in a network to every other node in the adjacency matrix. The adjacency matrix is then formed by repeating this for every node in the network.

This method was selected due to its previous success in predicting the connectivity of GRNs [24]. Hopefully the ensemble of many decision trees also means that this method may be more robust to noise in the data. A drawback of this method is that it does not make a direct calculation of the entries in the adjacency matrix, it only gives a value of how good they are as predictors of the values of other nodes in the network. Theoretically, this would suggest that the method is best suited to binary adjacency matrices, where the predicted connectivity values could be thresholded to set values to 0s and 1s.

3.3.3 Kalman Filter

Taking inspiration from some of the papers using a statistical approach to network inference [16]-[18], the Kalman filter was chosen as a method to

sequentially estimate the parameters $\hat{\mathbf{A}}_i$ in Equation 10 for each node in the network.

The Kalman filter is an online learning approach to finding the optimal set of parameters $\hat{\mathbf{A}}_i$ [35]. After an initial random estimate of these parameters ($\boldsymbol{\chi}_0$), the error between the estimated value of the gradient using $\boldsymbol{\chi}_0$ and the true value of the gradient of the node at that time is calculated:

$$E := (\mathbf{Y}_{i,m} - \boldsymbol{\chi}_0 \mathbf{G}_{i,m})^2 \tag{12}$$

Posterior updates are then made to the estimates of both $\boldsymbol{\chi}_m$ (for timestep m; the m^{th} estimate of the weights in $\hat{\mathbf{A}}_i$), as well as a measure of the error covariance matrix:

$$\boldsymbol{\chi}_{m+1} := \boldsymbol{\chi}_m + E\mathbf{k}(m) \tag{13}$$

$$P_{m+1} := (\mathbf{I} - \mathbf{k}(m)\mathbf{G}_{i,m}^T)P_m \tag{14}$$

Where $\mathbf{k}$(m) is the Kalman gain given by:

$$\mathbf{k}(m) := \frac{P_{m-1}\mathbf{G}_{i,m}}{\mathbf{R} + \mathbf{G}_{i,m}^T P_m \mathbf{G}_{i,m}} \tag{15}$$

Where R is a random noise matrix (with values taken from a Gaussian distribution); it is re-generated for each iteration of the algorithm. This method allows the adjacency matrix to be estimated sequentially, the real-world advantages of being able to do this have already been stated in Section 3.3.1. As with all of the methods presented, it takes advantage of having a knowledge of the equations governing the phase of the oscillators in the network. The Kalman update equations capture all previous data used to train the model, meaning the next update only requires the most recent estimate of the state variables in a Markov chain. This makes the model relatively light on how much memory it has to be using at any one time. The technique also provides a running measure of the quality of the estimate of the model parameters at each time step (via the measure of P in Equation 14) making analysing its performance easier.

3.3.4 Neural Network

The way that the original problem is formulated, in that it is a linear regression problem, means that the solution need not be very complicated. The solution requires a set of weights to be found, which would then be linearly combined with a set of data to get as close as possible to a target variable.

Taking this into account, the application of a neural network to solving the problem may seem unnecessarily complicated. However, the network built for solving Equation 10 is suitably simple whilst still drawing on the advantages of neural networks.

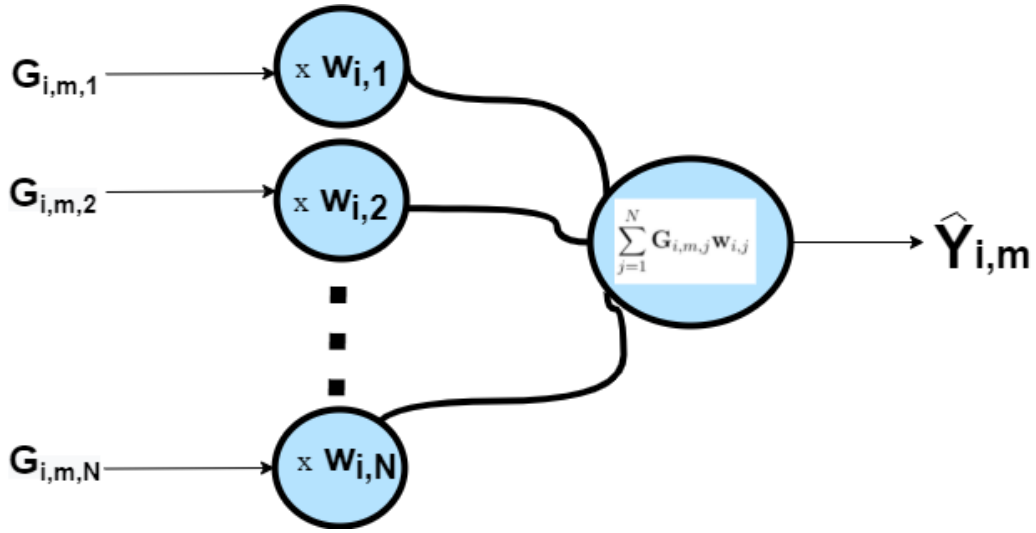

Figure 2: A representation of the neural network for predicting the gradient of the i^{th} oscillator in a system at timestep m.

$$\hat{\mathbf{Y}}_{i,m} := \sum_{j=1}^{N} \mathbf{G}_{i,m,j}\mathbf{w}_{i,j} \tag{16}$$

The equation for the i^{th} neural network is shown in Equation 16 where $\mathbf{w}_i$ are the weights of the network and $\mathbf{G}_i$ is contains the data used to train the network.

The feedforward neural network, as shown in Figure 2, shows how every part of the problem is mapped to a feature of the neural network, with the weights in the adjacency matrix being represented by the weights in the network. Equation 16 shows this in mathematical form. Although one neural network could be used for the entire problem, N individual neural networks were chosen to solve the N different sub problems described in the problem formulation section. This is to keep the process of solving Equation 10 consistent; every other method splits the task up into N sub problems, each of which finds the connectivity between the i^{th} node in the network and every other node.

The weights in this network are updated via a back propagation algorithm. For each piece or group of data from the matrix $\mathbf{G}$, the network pro-

cesses the data via Equation 16. The output of the network is the estimate of the corresponding entries in the matrix $\dot{\mathbf{X}}_{i,m}$.This estimate is compared with the true values in Y to give an error value calculated via:

$$\mathbf{E}_{i,m} := (\mathbf{Y}_{i,m} - \hat{\mathbf{Y}}_{i,m})^2 = (\mathbf{Y}_{i,m} - \sum_{j=1}^{N} \mathbf{G}_{i,m,j}\mathbf{w}_{i,j})^2 \tag{17}$$

Backpropagation works by differentiating this error function with respect to the j^{th} weight being estimated and then uses that to update the j^{th} weight via stochastic gradient descent:

$$\mathbf{w}_{i,j} = \mathbf{w}_{i,j} - r\nabla_{\mathbf{w}_{i,j}}\mathbf{E}_{i,m} \tag{18}$$

Where r is known as the learning rate. One of the benefits of this method is that many techniques exist for optimising this gradient descent method to either speed up the rate of convergence to a true solution, or to avoid getting stuck in local minima. Firstly the learning rate in Equation 18 can be adjusted to find the best value in allowing the algorithm to reach an optimal solution quickly and smoothly. For high dimensional loss landscapes the introduction of a momentum term can help the algorithm converge to an optimal solution where otherwise it may be difficult. Other methods used to help improve the gradient descent algorithm include AdaDelta and adaptive moment estimation (ADAM) [36].

Although most neural networks are heralded for their ability to learn complex relationships (rather than simply the weights in linear regression) the application of a simple feedforward neural network is still justified: The use of a neural network to solve the problem has already proved to be effective in predicting the topology of a network, as well as learning the equations determining the propagation of information in the network [12] and [23]. Additionally, such a method could also be trained with one piece of data at a time lending itself to an online (and potentially more realistic) setting.

3.3.5 Logistic Neural Network

One issue that can occur commonly with neural networks is that they can over fit data, often due to an ability to learn 'too well' from input data and so do not generalise well to fit data not used for training. In this context, it could lead to weights being far too high, or leading to negative weight values which may fit the training data better, but not be close to the true adjacency matrix. One approach could be to punish weights with a penalty in the loss function. An approach to ensuring values fall in a certain range would be to

add a logistic function for each weight in the network before summing them as displayed in Equation 19.

$$\hat{\mathbf{Y}}_{i,m} := \sum_{j=1}^{N} \mathbf{G}_{i,m,j} expit(\mathbf{w}_{i,j}) \tag{19}$$

Where expit is the logistic (or expit) function applied to the weights in the neural network (see Equation 20).

$$expit(\mathbf{w}_{i,j}) := \frac{L}{1 + e^{-a(\mathbf{w}_{i,j} - x_0)}} \tag{20}$$

Here, L is the maximum height of the function, a is the steepness of the curve and x_0 is the midpoint of the curve. The bonus of adding a logistic function is that it forces the weights within a certain range. It is also a differentiable function, so could still be used in conjunction with the gradient descent methods discussed in Section 3.3.4 for updating the weights.

3.3.6 Genetic Algorithm

Although many of the methods discussed so far have already been implemented for solving a network inference problem of some type, one of the main goals of this project was to see which methods would be more robust to the amount of noise present in the data as well as in how synchronised the data set may be. One of the reasons that a genetic algorithm was chosen to be tested is that they are renowned for being good at dealing with noise in data [37]. They are also known for not getting stuck in local minima when searching the loss landscape for finding an optimal solution to a problem. This may be in part because the algorithm starts with a population of solutions rather than just one. The part of the algorithm known as 'mutation' also means that there is a stochastic element to the algorithm. A final motivation for using this method is that all values in the solutions found are already 1s and 0s so no thresholding is required.

For each node, the genetic algorithm would generate a population of p solutions (each one being an estimate of the true adjacency matrix) of length N (for N nodes in a network). Then, for each epoch of algorithm, each member of the population will be given a fitness score based on the value they give when substituted into Equation 10, using either a piece or group of data.

The half of the population with the highest scores are then removed from the solution set, and the remaining members of the population are selected to 'breed', producing as many offspring as were removed from the population.

Each offspring has two different parents chosen from the population randomly with a uniform distribution. A random point in the parent string of values is chosen, any values before this point are taken from one parent and any values after this point are taken from another, in turn the values are then combined to produce a child.

The entire population is then mutated, by randomly flipping the sign of a part of each member. The genetic algorithm is described in full in Algorithm 1.

Algorithm 1 Genetic Algorithm (PopSize, n, Generations, Data)

$Population \leftarrow initialPop(PopSize, n)$ ▷ Generate 'PopSize' many n × n binary matrices
for i in range(Generations) **do**
  for j in Population **do**
    $Loss_j = Equation10(j, Data[i])$ ▷ Calculate loss function value
  end for
  $Population = Order(Population, Loss)$ ▷ Sort population by losses
  $Population = Cull(Population)$ ▷ Remove lower half of population
  for j in range(len(Population/2) **do**
    $p_1, p_2 = parents(Population)$ ▷ Select two parents randomly
    $p_{new} = breed(p_1, p_2)$ ▷ Mix p_1 and p_2 to make a child
    $Population = Population.add(p_{new})$ ▷ Add child to population
    $Population = mutate(Population)$ ▷ Randomly change solutions
  end for
end for
for j in Population **do**
  $Loss_j = Equation10(j, Data[i])$ ▷ Calculate loss function value
end for
$Population = Order(Population, Loss)$ ▷ Sort population by losses
return $Population[0]$ ▷ Return best solution

This process can be repeated for each bit or group of data from the data set, followed by selecting the member of the population with the lowest 'fitness' score as the solution to the problem. Solutions for each node in the network are then combined to form the predicted adjacency matrix.

One clear drawback from this solution is that it requires binary values so would not be appropriate for adjacency matrices with non binary values. It also takes no account of which parts of a solution may have contributed more heavily to it being a successful or failed solution as previous methods (such as the neural network) may have done.

4 Implementation & Results

This section presents the details for implementing all of the methods discussed in Section 3 as well as the performances of the methods on a variety of data sets. Firstly, baseline measures of the performances of the methods will be given; demonstrating their performance on a relatively small network with no noise present, low data sparsity and low synchronisation. Following this, the methods will be compared on their robustness to noise, data sparsity and high synchronisation. Finally, the two best methods will be selected to be tested on a much larger network.

The creation of the synthetic dataset and the implementation of the models were both carried out in Python. It is a programming language with a range of libraries such as Scipy and Numpy [38] and [39] to help implement dynamic modelling and also machine learning models.

4.1 Baseline Tests

4.1.1 Synthetic Dataset

For the baseline tests, an adjacency matrix of size 15 $\times$ 15 was randomly generated with values of either 1 or 0 and a leading diagonal of 0s. A size of 15 was used so that the networks being represented were only very small which was appropriate for baseline testing. Data generation then took place by running Equation 2 for 30 seconds, taking measurements at every half a second, this was made possible using Scipy's 'odeint' package. The natural frequencies of each node ω_i were randomly sampled from a Gaussian distribution with mean 1 and a standard deviation of 0.5. The coupling constant K was set to 1 (meaning there was little synchronisation in the network), there were 15 nodes in the network so N=15. No noise was added to the network for the baseline tests, meaning that λ=0 in Equation 3. The data matrix $\mathbf{D} \in \mathbb{R}^{15\times60}$ was then generated using 60 measurements for 15 nodes. The matrices $\mathbf{G}$ (used as data to train the models) and $\mathbf{Y}$ (used as target variables to evaluate the model performance during training) were then created as per the process described in Section 3.2. To create multiple samples, the simulation can simply be run multiple times. For the baseline tests, each method was tested for 1 to 30 samples. For each sample size, the average performance of the model over 10 tests was used as its final performance score.

The justification for these parameter values come from [20] which tests a network inference algorithm on data generated with Kuramoto dynamics. The natural frequencies were assigned as described above. The simulation

was sampled every 0.1 seconds for a number of oscillators ranging from 5 to 40. Using similar or identical values for the parameters used to generate data in this paper means that the results will be able to be evaluated against existing data.

The performance of each algorithm was measured using how many correct entries of the adjacency matrix were made.

$$Score = \frac{\#CorrectAdjacencyMatrixEntries}{\#SizeOfTheMatrix} \times 100\% \tag{21}$$

4.1.2 Closed Form Solution-Baseline

The closed form solution was implemented as per the description in Section 3.3.1, the results of the algorithm were then thresholded; any values beneath the threshold value were set to 0 and any above it were set to 1. For this algorithm a threshold value of 0.85 was used. In all cases, the specific threshold values used were chosen based on which gave the best results in preliminary testing. It is also worth stressing that the prelimary testing data was separate to the data used for the results presented in the paper; preliminary tests were used to determine a range of parameter values for al the methods implemented in this paper. Figure 3 shows the results of the baseline test for this method.

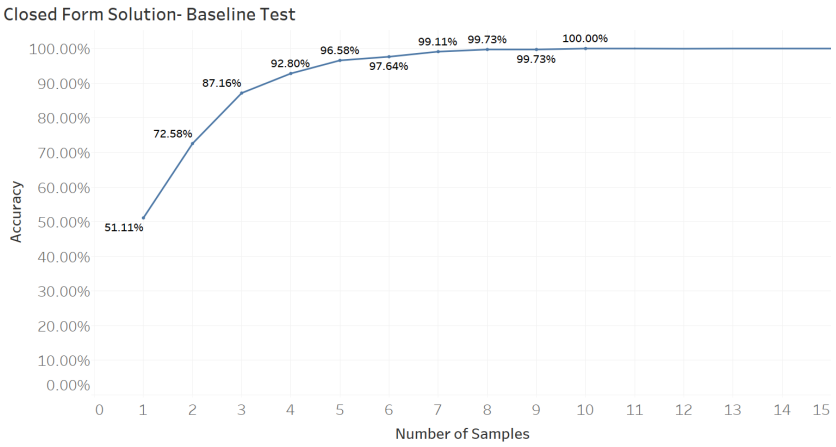

Figure 3: Baseline reconstruction accuracy for the closed form solution

The method required 10 samples (600 data points) to reach 100% accuracy, the average time for this method to run on 10 samples was 18.03 seconds.

4.1.3 Random Forest-Baseline

The random forest method was implemented with 1000 decision trees. The number of trees was chosen so that the algorithm had enough learners to give accurate results, but not so many that the algorithm would take a very long time to run. The results were thresholded with a value of 0.03.

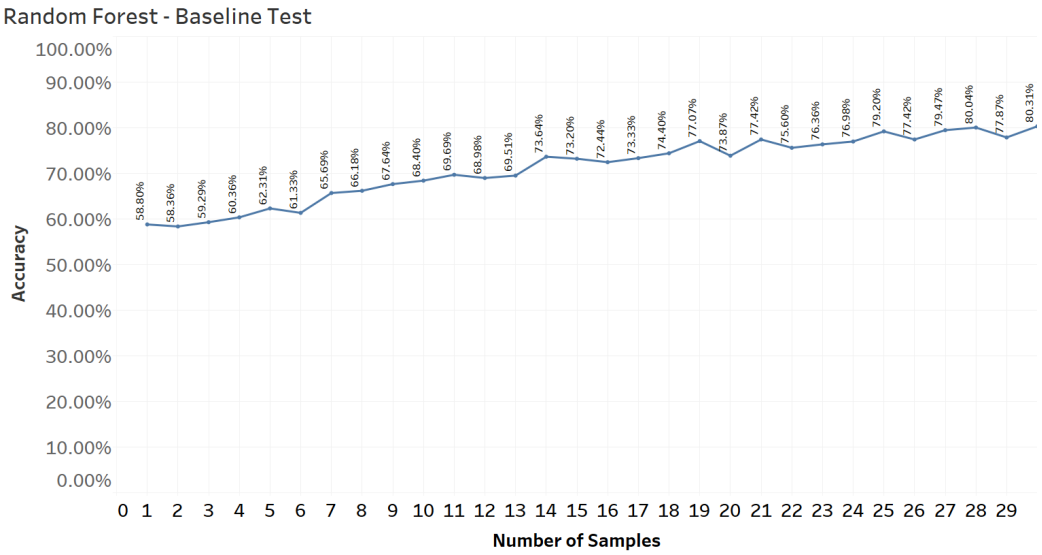

Figure 4: Baseline reconstruction accuracy for the random forest method

The results of the test are shown in Figure 4. The best accuracy for the algorithm was 80.31% which was achieved with 30 samples and took an average of 310.08 seconds. For 10 samples the algorithm took 101.50 seconds to run.

4.1.4 Kalman Filter-Baseline

The Kalman filter used a threshold of 0.5. Its baseline test performance is shown in Figure 5.

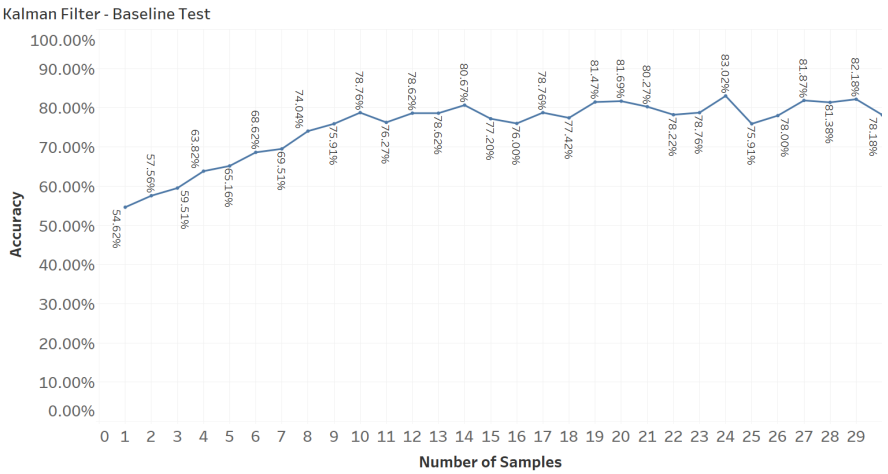

Figure 5: Baseline reconstruction accuracy for the Kalman filter method

The highest accuracy for the algorithm was 83.02% which was achieved with 24 samples and took 69.09 seconds. For 10 samples, the algorithm took 30.80 seconds.

4.1.5 Neural Network-Baseline

The neural network method used a threshold value of 0.9. The gradient descent method chosen to update the weights was stochastic gradient descent (see Equation 18). The learning rate was set at 0.5. Figure 6 shows the results of the algorithm.

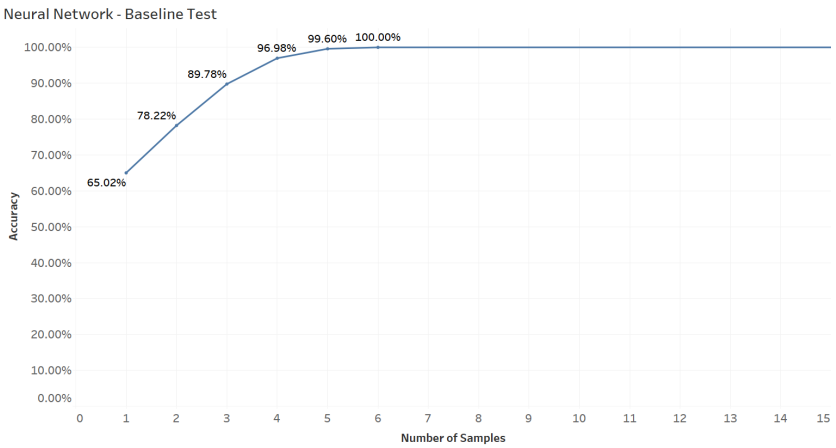

Figure 6: Baseline reconstruction accuracy for the neural network method with stochastic gradient descent

The method required 6 samples to reach an accuracy of 100% and took on average 173.59 seconds to do so. To perform on 10 samples, the algorithm took 284.51 seconds.

4.1.6 Logistic Neural Network-Baseline

The logistic neural network used the same paramters for training as the neural network. The logistic functions processing the weights of the network had maximum values of 1 (to bound the weights between 0 and 1), mid points of 0 and a logistic growth rate of 0.1 (a relatively low value to exaggerate the difference required of a weight to make either 0 or 1).

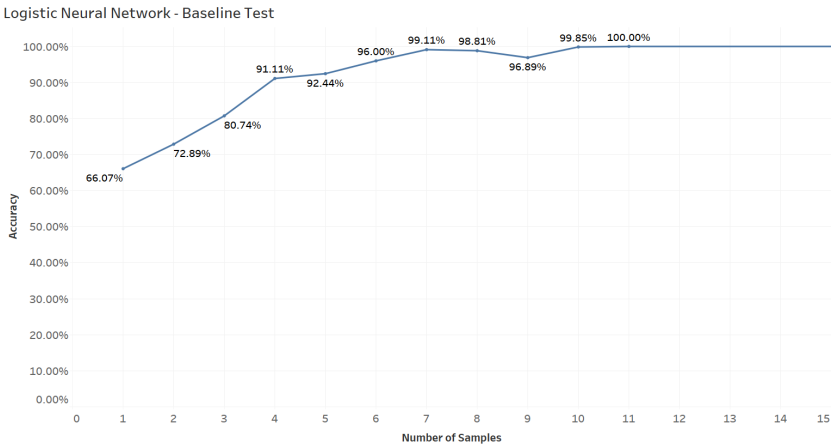

Figure 7: Baseline reconstruction accuracy for the logistic neural network method with stochastic gradient descent

The method reached an accuracy of 100% with 11 samples (see Figure 7), taking 375.80 seconds on average. For 10 samples, the algorithm took an average of 341.91 seconds.

4.1.7 Genetic Algorithm-Baseline

The genetic algorithm was implemented as described in Algorithm 1. For breeding two solutions, each row in each parent was split at a random point; the child took the first part of the division from one parent and the second part from the other. For mutation, each member of the population had a 100% chance of being altered. The chance for each of its rows having one bit changed had a probability of 90%. The motivation for a very high amount of mutation was so that the algorithm would not get stuck with a 'good' solution to the loss function without ever finding the optimal one. The population size used was 100. The results of the algorithm are shown in Figure 8.

The method never did better than randomly guessing the adjacency matrix. The accuracy was always above 50% as the leading diagonal of each solution was always 0. For 10 samples, the algorithm took an average of

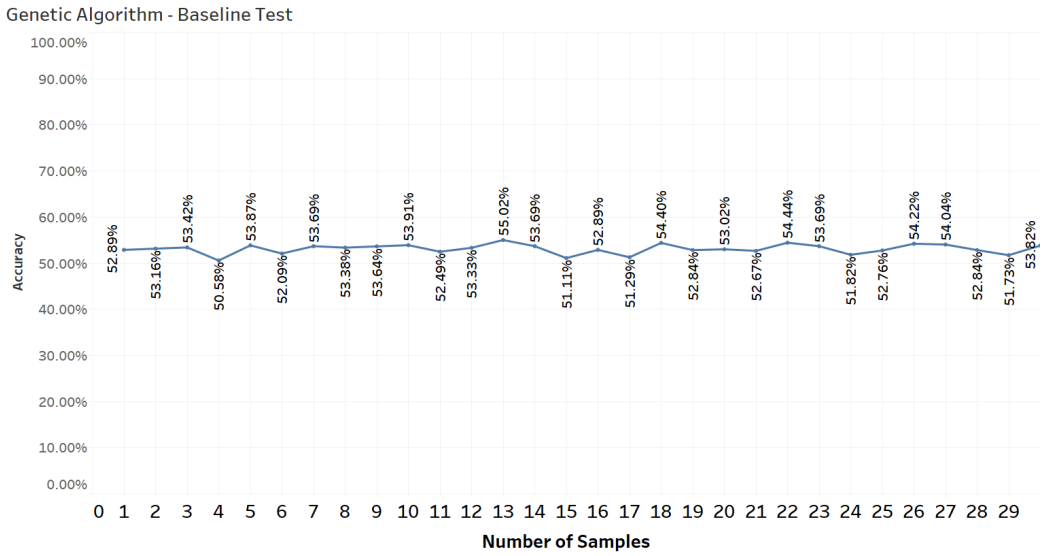

Figure 8: Baseline reconstruction accuracy for the genetic algorithm

20.00 seconds to run. The algorithm's performance meant that it was used for no further testing.

	0.5s	1s	2s	5s
Closed Form	**100%**	**100%**	**92.89%**	**78.89%**
Random Forest	80.30%	79.38%	75.93%	70.52%
Kalman Filter	78.18%	51.24%	52.67%	50.93%
Neural Network	**100%**	96.44%	81.48%	78.78%
Logistic Neural Network	**100%**	82.67%	61.19%	53.93%

Table 1: Reconstruction accuracy of the inference networks for varying levels of sparsity in the training data

4.2 Testing Sparsity

Each method was tested on 4 different datasets generated with different levels of sparsity. The data sets were generated in the same way as in section 4.1 but instead of taking samples of the data at half second intervals, samples were taken at 1, 2 and 5 second intervals. To make sure the same amount of data was still being used to test each method, the simulation was run for longer for greater time intervals between samples. Table 1 shows the accuracy of each algorithm for datasets generated with varying levels of sparsity. Each result is the average of testing the algorithm 10 times on 30 simulation samples.

4.3 Testing Noise

Each method was tested on 4 different datasets generated with different levels of noise. The data sets were generated in the same way as in section 4.1 however, the lambda values in Equation 3 were set at values of 0.0001, 0.0005 and 0.001 for three different tests. The motivation for using these λ values for the dynamic noise is that M. Panaggio et al. [20] use a similar range of dynamic noise values (0.00001, 0.0001 and 0.001) in their tests. They sample the natural frequencies of the network oscilators in the same way for a similar number of osciallators (5, 10, 20 and 40). Since the work uses similar conditions for generating data to this study, using similar noise values meant results could more easily bew compared. Table 2 shows the accuracy of each algorithm for datasets generated with varying levels of noise. Each result is the average of testing the algorithm 10 times on 30 simulation samples.

	$\lambda = 0$	0.0001	0.0005	0.001
Closed Form	**100%**	**83.11%**	47.32%	48.23%
Random Forest	80.30%	63.56%	**60.41%**	**53.19%**
Kalman Filter	78.18%	55.54%	49.33%	49.02%
Neural Network	**100%**	79.56%	51.38%	49.73%
Logistic Neural Network	**100%**	79.56%	50.22%	51.40%

Table 2: Reconstruction accuracy of the inference networks for varying levels of noise in the training data

	K=1	5	10	15
R_{Avg}	0.316	0.835	0.943	0.969
R_{Final}	0.561	0.954	0.988	0.991
Closed Form	**100%**	93.04%	73.78%	69.33%
Random Forest	80.30%	75.26%	68.59%	67.36%
Kalman Filter	78.18%	54.81%	49.78%	54.37%
Neural Network	**100%**	**100%**	**95.85%**	**82.52%**
Logistic Neural Network	**100%**	77.78%	52.89%	53.93%

Table 3: Reconstruction accuracy of the inference networks for varying levels of synchronisation in the training data. R_{Avg} and R_{Final} give average measures of the average synchronisation level for the simulation and average final value of the synchronisation as measured by the Kuramoto order parameter (See equation 4).

4.4 Testing Synchronisation

Each method was tested on 4 different datasets generated with different levels of synchronisation. The data sets were generated in the same way as in section 4.1 but used varying levels for the coupling constant K. Table 3 shows the accuracy of each algorithm for datasets generated with varying levels of synchronisation. Each result is the average of testing the algorithm 10 times on 30 simulation samples.

4.5 50 Node Dataset

Having generally performed as the best methods on the tests described so far in this section, the closed form solution and neural network methods were

selected for testing on a dataset generated from a 50 node network. Two tests were conducted for each method. One ran the same simulation as in Section 4.1 (with no noise, low synchronisation and samples taken every 0.5 seconds) but with 50 nodes. The other used a 50 node network but used a coupling constant of K=8 and had a small amount of noise added ($\lambda = 0.0001$) in the attempt to make a more 'realistic' setting for the methods to be tested on. The data was generated by running the simulation for 30 seconds, and then repeating this 100 times taking samples at every half a second. The amount of data used to train the algorithms was increased for this test as there were more unknowns (links) in the network to estimate.

On the first test, the neural network achieved an average accuracy (over 10 runs) of 99.92% whilst the closed form solution achieved an accuracy of 99.80. On the second test, the neural network achieved an average accuracy (over 10 runs) of 52.92% whilst the closed form solution achieved an accuracy of 49.22%.

5 Analysis & Discussion

Having tested the models on a range of data sets, the results of these tests will now be evaluated. The results will also be discussed with regards to the domain of neuronal network inference and for determining what future work could focus on. The initial analysis will systematically evaluate the results of each test from Section 4 in the same order in which they appear in that section. Following this, a broader analysis of how the methods performed over all the tests will be given alongside how this compares with results given in existing literature on the topic of neuronal network inference and any limitations of this project. Finally recommendations for how this research could be taken further will be provided.

5.1 Baseline Analysis

On the baseline tests, the closed form solution, neural network and logistic neural network all achieved an accuracy of 100%. The neural network performed best as it managed to achieve this accuracy with only 6 samples; the closed form solution and logistic neural network achieved a 100% accuracy with 10 and 11 samples respectively.

Whilst the accuracy of the random forest method was considerably lower, reaching a maximum of 80.31% accuracy for a dataset of 30 samples, Figure 4 does show how the accuracy gradually increases with more data. This suggests that, with enough data, the method may well reach 100% accuracy. In comparison, although the Kalman Filter method reached a better accuracy for its best result at 83.02%, after about 10 samples the results fluctuate between around 75% and 85%, with no obvious improvement with additional samples. This suggests that more samples may not be beneficial to the performance of the model. The closed form solution also performed much better than the other algorithms for the amount of time taken to make an estimate of the adjacency matrix, taking 18.03 seconds to achieve a 100% accuracy. This was much better than the other models which achieved a high accuracy, with the neural network and logistic neural network taking 284.51 and 341.92 seconds respectively for a solution generated using 10 samples.

The baseline tests show clearly how, with low synchronisation and no noise, the neural network methods and closed form solution work very well. The results also suggest that the same could be said for the Random Forest method were there enough data available.

The genetic algorithm performed poorly, never achieving an average accuracy of over 56%. This may be due to the fact that the algorithm does

not make use of which parts of its solution have performed better than others when creating new members for the population; something which all the other algorithms do. An improvement could be made to this algorithm in future research to make the 'breeding' part of the algorithm analyse which parts of the parents performed best on the task. When compared to existing literature which has implemented a genetic algorithm, it is not massively surprising that the algorithm has performed poorly. The GABNI algorithm [26] uses a genetic algorithm to supplement a statistical algorithm. Future work could look into pairing a genetic algorithm with one of the other techniques discussed in this paper.

5.2 Sparsity Analysis

The results in Section 4.2 show how the closed form solution outperformed every other algorithm for accuracy when left to deal with a sparse data set, with the neural network having a slightly worse accuracy on these tests. With sparser data, the approximation of the gradient (using Equation 7) becomes much weaker. For time samples with 5 second intervals, the gradient approximation may be very far off as the node's phase may not increase linearly at all. For this reason, algorithms which update the model parameters in an online setting may be more sensitive to a small amount of data which is 'incorrect' (in that the gradient approximation is far off the true gradient value). This may explain why the closed form solution performed best (achieving an accuracy of 78.89% for 5 second intervals) and why the Kalman filter performed so poorly (reaching an average accuracy of 51.24% even with only 1 second time intervals). It is of note that the random forest decreased the least in accuracy from its baseline test for the sparsity tests; it dropped only $\sim$10% in accuracy when trained on a dataset with 5 second sampling intervals.

A criticism of the study may be that making a linear gradient approximation for such large time intervals between observations may not yield any clarity as to which methods are best. This is because the approximated gradient (Equation 7) is too far from the true gradient value. However, there are several rebuttals to this argument when considering the bigger picture: Firstly, the best methods still achieve a good reconstruction accuracy even when using the linearly approximated gradients which ultimately shows that the linear approximation is not a massive issue. Secondly, in a real world context, EEG data is recorded at far more regular time intervals than 5 seconds so linearly approximating the gradient in this context would be entirely appropriate. Finally, all existing literature [19]-[22] studied for this project uses the linear approximation of the gradient in testing inference models on

data generated with the Kuramoto model. Whilst using another method of approximating a gradient may give a more accurate prediction for greater timesteps, it would make the methods discussed here harder to compare to existing work.

During training, the two neural networks were updated, one piece of data at a time. Future research could look into whether updating the parameters of the model with batch data may help to make these models less sensitive to data sparsity.

In a 'real world' context, EEG scans should be able to provide data at more regular intervals than even 0.5 seconds. This test showed that, if this is not possible, that a closed form solution would be of most value, provided the updates to the adjacency matrix estimation were not needed in an online learning setting.

5.3 Noise Analysis

The results of the noise tests show that, over all, every method was particularly sensitive to any addition of noise. A limitation of this test was that little literature exists which describes any additional noise when testing the reconstruction of neuronal networks. It was therefore difficult to judge both how much noise to add and what constituted as a 'good' performance on noisy data. Even for a low amount of noise ($\lambda = 0.0001$), the methods which had performed best on the the baseline tests dropped significantly in their reconstruction accuracies; the closed form solution dropped to an accuracy of 83.11%.

As in the sparsity test, the random forest method dropped the least in accuracy for noisier data. It performed best (with an accuracy of 60.41%) on a data set for which $\lambda = 0.0005$. Aside from this, every method achieved a low accuracy (each one obtaining $\sim$50% accuracy) for both the $\lambda = 0.0005$ and $\lambda = 0.001$ tests. Future work should certainly focus on how to help improve network reconstruction methods when this type of noise is present in data generated with Kuramoto dynamics.

For comparison, M. Panaggio et al. [20] demonstrate a 100% reconstruction accuracy from a neural network method when $\lambda = 0.001$ with data generated in the same way as described here in Section 4.3. It is of note that they used time intervals of 0.01s, further research could see whether these methods are more robust to noise with significantly shorter time intervals.

The robustness of the random forest method is also of note. Future work could look into how a random forest method with better baseline results may perform on noisy data.

5.4 Synchronisation Analysis

The literature surrounding network inference provides little information as to how synchronisation affects the ability to reconstruct a network. There is no comparative research between methods on their ability to deal with greater levels of synchronisation. This may be due to synchronisation being a quality which is unique to modelling network dynamics with the Kuramoto model.

For all of the different levels of synchronisation, the neural network outperformed the other methods. It achieved a reconstruction accuracy of 82.52% for K=15 when the next highest accuracy was 69.33% (achieved by the closed form solution). Again, the Kalman filter performed poorly for any K value above 1.

With higher synchronisation, the network performs in a potentially more complex way, being more dictated by the phase differences between nodes. This could be the reason for the success of the neural network for highly synchronised set as such methods are known for better dealing with complexity in data compared to other machine learning and statistical approaches. However, this conclusion is fairly speculative, the problem is not formulated in a different way for the neural network method (in that it is asked simply to find the weights in a linear regression problem). In addition to this, the logistic neural net performed fairly poorly, achieving accuracies of $\sim 50\%$ on datasets generated for K=10 and K=15.

As was the case with the other two parameter tests, the random forest method was least affected with extra synchronisation, dropping only $\sim$13% in accuracy from a K value of 1 to 15. This once again demonstrates its robustness to more challenging parameter values.

5.5 50 Node Analysis

Testing the methods on a 50 node network was carried out in the hope of seeing how the methods performed in a more 'realistic' setting. Having generally performed better on every test up to this point, the closed form solution and neural network were selected for testing on a larger network.

The first test was carried out with no noise and low synchronisation, with samples taken every 0.5 seconds to see how the methods performed on a scaled up network. Both methods performed well on this dataset, with both methods achieving accuracies around 99.90% over an average of 10 runs. This demonstrates that the methods achieve near-perfect neuronal network reconstruction even when the network size is scaled up considerably, providing there is little noise and a sufficient number of data points.

The second test was carried out with a small amount of both noise and synchronisation, further demonstrated how poor the methods performed in the presence of noise. The neural network achieved an accuracy of 52.92% and the closed form solution achieved an accuracy of 49.22%. This further stresses that further research should be carried out to improve the robustness of models to noise.

5.6 Final Discussion Points & Future Work

Across all of the tests, it was made clear that, in general, the neural network method and closed form solution performed best, achieving the highest reconstruction accuracies compared to the other methods in most cases.

In relation to the original goals of the project, the test results demonstrate a clear gap in performance between the two best methods (the closed form solution and neural network) and the others. The results also give a clear evaluation of which aspects in the generation of data lead to poorer results. Whilst the top methods were fairly robust to significant data sparsity and synchronisation, they were very poor at dealing with noise. As previously specified, this suggests that future research should focus on developing methods which are more robust to noise.

The Kalman filter has ways in which it could be modified to be more suited to noisy or highly synchronised datasets. However, its generally low baseline performance compared to other methods and very high sensitivity to noise, data sparsity and synchronisation mean that future research should focus on the development of other techniques instead.

The logistic neural network performed worse than the neural network in almost all cases, suggesting that the addition of a logistic function to bound values had little (even negative) effect. In a real world context, values would not be binary anyway so the outcomes of these tests suggests that the method is fairly redundant.

As a more general point, so that the research has more real world application potential, research should be done into which parameter values (for noise and synchronisation) lead to synthetic data which best resembles EEG scans of real brain regions.

One limitation or criticism of this research, is that it makes a large assumption about the equations governing the dynamics of brain regions (presuming that it is dictated by Kuramoto dynamics). Whilst this provides a simple framework to isolate only the ability of inference methods to estimate the network topology, further work could look more into understanding how such methods fare in predicting the dynamics of a system as well. W.Bomella et al. [4] show how, instead of making an assumption of the dynamics equa-

tions of a network, that many basis functions could be used to approximate such a function. For real world data, this may prove more accurate in making predictions about how data propagates in a neuronal network as it is less constraining on the form of the equation used to describe the network dynamics. An inference method which could accurately estimate both the adjacency matrix and parameters used for many basis functions would be far more valuable in a real world setting.

An interesting outcome of the study was how robust the random forest method was to noise and synchronisation. Were the base level performance of this method to be improved (perhaps with more data availability or more decision trees) then there is potential for it to outperform the closed form and neural network methods. This leads to a more general point about one of the biggest limitations of the project. All of the research was carried on a CPU with 32GB of RAM. If the tests were run on a more powerful computer or utilised a graphical processing unit or even a tensor processing unit, then the lack of data in training models would be less of a limiting factor. In turn, this may lead to the potential realisation of the random forest method being superior to other methods.

The most important focus of future research should be to test methods on their ability to reconstruct adjacency matrices with continuous values (rather than binary ones), this would give a better idea for how methods would perform in a real world context. This is because connectivity values in a network of brain regions would not be binary. The higher threshold value of 0.9 used when creating an adjacency matrix estimation using the neural network method (whilst still maintaining a high reconstruction accuracy) suggests that it may perform better in reconstructing a matrix with continuous values.

Finally, in the context of using machine learning to further neuroscience, future work should look more into how network inference techniques can be used to categorize brain types and hopefully into how the estimated network topology could be a good predictor of certain brain pathologies.

6 Conclusion

The report has given a comprehensive comparison of a variety of techniques for estimating the topology of synthetic neuronal networks. This is an improvement on previous work detailing the reconstruction of neuronal networks, where little comparative research is available and rarely has any one method been tested on a range of parameter values. Over all tests, the neural network and closed form solution methods performed consistently better than all other methods. They also performed excellently on a much larger network when there was little noise present and a low level of synchronisation. One of the biggest weaknesses of all the of the methods was their performance when noise was present and any future work should focus on trying to optimise these methods to deal with noise.

The method of most note was the random forest technique, which demonstrated the best robustness of all of the techniques; its reconstruction accuracy was the least affected by additional noise, synchronisation or data sparsity.

In the context of network inference, the results give a clear view of the strengths and weaknesses of the chosen network inference methods. Although binary matrices (which the methods in this paper were implemented to estimate) are a worse representation of a real life network representing brain region interaction, they do provide a means of clearly showing when a method has performed well or poorly; by not forcing the methods to find the precise values in the adjacency matrices (and instead thresholding the values they give) any disadvantages of a method are exaggerated.

With this in mind, the research presented provides grounds upon which further research into the inference of neuronal networks can be carried out. Knowledge of the strengths of the neural network and closed form method as well as some idea into the potential of the Random forest method, means that future work need not look into the performance of weaker methods. Instead the top performing methods here can be improved upon and tested on more 'realistic' data sets with continuous valued adjacency matrices and perhaps more data.

The knowledge of how a particular method copes with a variety of conditions, such as higher network synchronisation, may be of great use when the method is applied to real world data. Brain regions are unlikely to exhibit the same amount of synchronisation at all times or for different arrangements of where EEG data is taken from. Having a method which is robust to a variety of factors would make it more suitable for tests looking into a variety of EEG data sets and would therefore aid a more thorough detailing of the

brain. In turn, this should make the categorisation of certain brain diseases or understanding of brain functionality easier.

References

[1] C. Oates and S. Mukherjee, "Network inference and biological dynamics," *The annals of applied statistics*, vol. 6, no. 3, 2012.

[2] F. D. Sahneh and C. Scoglio, "Epidemic spread in human networks," *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, 2011.

[3] *Complex network*, Available at `https://en.wikipedia.org/wiki/Complex_network`. (visited on 02/08/2021).

[4] W. Bomela, S. Wang, C.-A. Chou, and J.-S. Li, "Real-time inference and detection of disruptive eeg networks for epileptic seizures," *Scientific Reports*, vol. 10, 2020.

[5] R. J. Howell and T. Alter, "Hard problem of consciousness," *Scholarpedia*, vol. 4, no. 6, 2009.

[6] J. H. Morrison* and P. R. Hof, "Hard problem of consciousness," *Science*, vol. 278, no. 5337, 1997.

[7] Ruetir, *Deepmind ia discovers the structure of 98.5% of human proteins*, Available at `https://www.ruetir.com/2021/07/22/deepmind-ia-discovers-the-structure-of-98-5-of-human-proteins/`. (visited on 02/08/2021).

[8] D. Van Essen, K. Ugurbil, E. Auerbach, D. Barch, T. Behrens, R. Bucholz, A. Chang, L. Chen, M. Corbetta, S. Curtiss, S. Della Penna, D. Feinberg, M. Glasser, N. Harel, A. Heath, L. Larson-Prior, D. Marcus, G. Michalareas, S. Moeller, R. Oostenveld, S. Petersen, F. Prior, B. Schlaggar, S. Smith, A. Snyder, J. Xu, and E. Yacoub, "The human connectome project: A data acquisition perspective," *NeuroImage*, vol. 62, no. 4, 2012.

[9] G. Schumann, E. Loth, and T. Banaschewski, "The imagen study: Reinforcement-related behaviour in normal brain function and psychopathology," *Molecular Psychiatry*, vol. 15, 2010.

[10] N. Adar, S. Okyay, K. Ozkan, S. Şaylısoy, B. D. Özbabalık, and B. Adapınar, "Feature selection on mr images using genetic algorithm with svm and naive bayes classifierss," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 4, 2016.

[11] *Somatosensory system*, Available at `https://en.wikipedia.org/wiki/Somatosensory_system`. (visited on 02/08/2021).

[12] M. Chen, J. Zhang, Z. Zhang, L. Du, Q. Hu, S. Wang, and J. Zhu, "Inference for network structure and dynamics from time series data via graph neural network," 2020.

[13] *Gene regulatory network*, Available at `https://en.wikipedia.org/wiki/Gene_regulatory_network`. (visited on 02/08/2021).

[14] A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. D. Favera, and A. Califano, "Aracne: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context," *BMC Bioinformatics*, vol. 7, no. S7, 2006.

[15] N. A. Kiani, H. Zenil, J. Olczak, and J. Tegnér, "Evaluating network inference methods in terms of their ability to preserve the topology and complexity of genetic networks," *Seminars in Cell Developmental Biology*, vol. 51, 2016.

[16] I. Nachman, A. Regev, and N. Friedman, "Inferring quantitative models of regulatory networks from expression data," *Bioinformatics*, vol. 20 Suppl 1, 2004.

[17] M. Kim and J. Leskovec, "The network completion problem: Inferring missing nodes and edges in networks," *Proceedings of the 11th SIAM International Conference on Data Mining, SDM 2011*, 2011.

[18] J. Geweke and H. Tanizaki, "Bayesian estimation of state-space models using the metropolis–hastings algorithm within gibbs sampling," *Computational Statistics Data Analysis*, vol. 37, no. 2, 2001.

[19] S. Gorur-Shandilya and M. Timme, "Inferring network topology from complex dynamics," *New Journal of Physics*, vol. 13, 2010.

[20] M. J. Panaggio, M.-V. Ciocanel, L. Lazarus, C. M. Topaz, and B. Xu, "Model reconstruction from temporal data for coupled oscillator networks," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, 2019.

[21] G. Ódor and J. Kelling, "Critical synchronization dynamics of the kuramoto model on connectome and small world graphs," *Scientific Reports volume*, vol. 9, 2019.

[22] S. Wang, E. D. Herzog, I. Z. Kiss, W. J. Schwartz, G. Bloch, M. Sebek, D. Granados-Fuentes, L. Wang, and J.-S. Li, "Inferring dynamic topology for decoding spatiotemporal structures in complex heterogeneous networks," *Proceedings of the National Academy of Sciences*, vol. 115, no. 37, 2018.

[23] Z. Zhang, Y. Zhao, J. Liu, S. Wang, R. Tao, R. Xin, and J. Zhang, "A general deep learning framework for network reconstruction and dynamics learning," *Applied Network Science*, vol. 4, no. 110, 2019.

[24] V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts, "Inferring regulatory networks from expression data using tree-based methods," *PLoS ONE*, vol. 5, no. 9, 2010.

[25] *Genetic algorithm*, Available at `https://en.wikipedia.org/wiki/Genetic_algorithm`. (visited on 02/08/2021).

[26] C. G. Gross, "Aristotle on the brain," *The Neuroscientist*, vol. 1, no. 4, 1995.

[27] *Human brain*, Available at `https://en.wikipedia.org/wiki/Human_brain#Gross_anatomy`. (visited on 02/08/2021).

[28] O. Sporns, G. Tononi, and R. Kotter, "The human connectome: A structural description of the human brain," *PLoS Computational Biology*, vol. 1, no. 4, 2005.

[29] *Kuramoto model*, https://en.wikipedia.org/wiki/Kuramoto$_m$*odel*. (visited on 02/08/2021).

[30] J. Cabral, M. L. Kringelbach, and G. Deco, "Exploring the network dynamics underlying brain activity during rest," *Progress in Neurobiology*, vol. 114, 2014.

[31] J. Gómez-Gardeñes, G. Zamora-López, Y. Moreno, and A. Arenas, "From modular to centralized organization of synchronization in functional areas of the cat cerebral cortex," *PLoS One*, vol. 5, no. 8, 2010.

[32] P. Cunningham, M. Cord, and S. J. Delany, *Machine Learning Techniques for Multimedia*. Berlin, Heidelberg: Springer, 2008.

[33] Y. Su, X. Gao, X. Li, and D. Tao, "Multivariate multilinear regression," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 6, 2012.

[34] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, 2004.

[35] G. Welch and G. Bishop, "An introduction to the kalman filter," *Department of Computer Science*, 1997.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, 2016.

[37] S. Mirjalili, *Genetic Algorithm. In: Machine Learning Techniques for Multimedia*. Springer, Cham, 2019, vol. 780.

[38] *Scipy*, https://www.scipy.org/. (visited on 02/08/2021).

[39] *Numpy*, https://numpy.org/. (visited on 02/08/2021).