Modeling neural connectivity of Caenorhabditis Elegans

by Guadalupe I. Fabre

S.B. Electrical Engineering and Computer Science

Massachusetts Institute of Technology, 2016

Submitted to the

Department of Electrical Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2017

Author: ______________________________

Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by: ______________________________

Steven W. Flavell
Assistant Professor of Brain and Cognitive Science
Thesis Supervisor

Accepted by: ______________________________

Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Modeling neural connectivity of Caenorhabditis Elegans

by

Guadalupe I. Fabre

Submitted to the Department of Electrical Engineering and Computer Science

June 9, 2017

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Little research has been done regarding the use of modeling techniques to estimate the connectivity of a nervous system by analyzing data recorded from its neurons. In this thesis, three main methods were implemented to solve this task: Bayesian inference, artificial neural networks, and ODE reverse engineering. The goal is to apply these methods to data recorded from C. elegans neurons and estimate the connections between these.

The top two performing methods on simulated data were feed-forward neural networks and ODE reverse engineering. The worst performing methods were recurrent neural networks and Bayesian inference.

Out of the methods tried, feed-forward neural network was the most robust to changes in parameters of the simulation network and noise. Furthermore, this technique is easily generalizable since it did not rely on any particular feature of the simulation network to achieve its good performance. Nevertheless, all methods are easily reproducible for any further research.

Thesis Supervisor: Steven W. Flavell

Title: Assistant Professor of Brain and Cognitive Science

Table of contents

ACKNOWLEDGMENT

I want to thank my family for filling my life with laughter, love, and support; the Flavell lab team for making me feel welcomed; Ni Ji and Steve Flavell for their consistent guidance, support, and reassurance; all MIT members who touched my life and allowed me to be a part of theirs, specially the few professors and students who believed in me, even if they had no reason to yet; lastly, my old and new friends who made MIT even more interesting.

Chapter I. Introduction

Problem and relevance

This thesis aims to evaluate three data analysis approaches to learn the correct connectivity of a nervous system given the neural activity of its cells. Specifically, these techniques will be used in the nervous system of Caenorhabditis Elegans, which consists of only 302 uniquely defined neurons [1]. Furthermore, the behavior of interest in this research is only long dwelling or roaming states of the worm. The neurons found to be associated with this behavior are in the order of 10 neurons [1], further decreasing the size of the network of interest. The connectome of the neurons involved in this process is depicted in Figure 1 below.

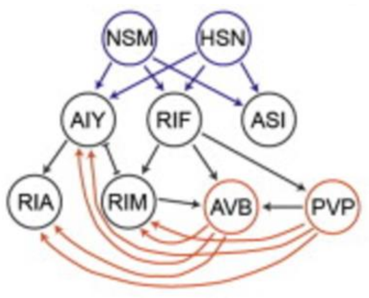


Figure 1. Connectome of C. elegans neurons that are involved in roaming and dwelling states [1]

There is no precedent of other research trying to solve this task. For instance, artificial neural networks have been shown to mimic the behavior of real nervous systems when trained to learn the same task [2]. However, current research is focused on modeling neural activity and/or outcomes of biological systems, using tools such as NEURON, based on ODEs [3], or various neural network architectures [2]. Still, most of the literature takes connectivity of the target nervous system to be known in full detail, which is not always correct to assume. For example, two neurons may communicate through neurotransmissions or diffusible neuromodulators; the latter case would not be visible in the connectome of the brain, causing a model based on it to underestimate the effect of a neuron on another [1].

History and overview of methods

The first step of the research was data acquisition, followed by data analysis. Data acquisition was done via simulation having biological applicability in mind. The three methods used in this research to estimate the connectivity of neurons in C. elegans are 1) Bayesian inference, 2) Artificial neural networks (ANN), and 3) Ordinary differential equations (ODE).

Bayesian inference

Bayesian inference is used to create belief networks, which are graphs that contain edges between dependent nodes [4]. Mutual information or other statistical metrics that find correlation between synchronous time series for various nodes are used to estimate dependency.

However, these techniques often fail in the presence of cycles [4]. Still, by manipulating the data to remove cycles, such as joining nodes or "unfolding" the cycles, these techniques have been applied to real data with some success [4].

Belief network creation does not provide information about the effect of an edge, such as inhibition or activation. Therefore, this technique is best paired with another algorithm that provides the extra information in order to complete the task of interest.

Artificial neural networks

A different approach that has gained interest recently are neural networks. Artificial neural networks were created in the 1940's as a computational approximation for real neurons [6]. Since then, neural networks have been used to solve various supervised learning problems [7].

The architectures of the neural networks that have been developed by researchers can be divided into two main kinds, feed forward and recurrent neural networks RNN [8]. In this research both architectures are used.

Even though RNNs have encountered theoretical and practical difficulties so far [8], some studies have found that when an RNN is trained to simulate the behavior of a biological system, the activity of the hidden nodes in the artificial network is similar to the neural activity of the biological system in question [2].

However, this has only been achieved with large networks that are formed by many layers and many nodes in each [2]. In general, neural networks are perceived as "black boxes," and often require hundreds of nodes to successfully simulate low-dimensional dynamics [9].

Ordinary differential equations

Differential equations are often used to create biologically realistic quantitative models of brain mechanisms and efficient simulations of the operation of these mechanisms [3]. Therefore, these are often used to simulate the behavior of nervous systems, assuming several parameters of the target mechanism are known [3].

In this research, the task is the opposite: use ODEs to reverse engineer the parameters of the model, assuming that ODEs are a correct model of the operation of neurons. Current research in the matter has shown that under some assumptions and multiple perturbations to the nodes of the network, the parameters of the model can be found by solving the differential equations in terms of the output and input time series [5].

However, this approach is sensitive to noise, and the parameters are not guaranteed to be successfully extracted if any of the necessary perturbations are not performed [5].

Chapter II. Data

Possible datasets

The overall goal is to use the methods developed in this thesis on recorded data from C. elegans neurons during roaming and dwelling states. However, obtaining this data and manipulating C. elegans is costly; therefore, the methods were first evaluated on simulations. Acquisition techniques for both sets of data, biological and simulated, are described below.

Biological data

The data available for C. elegans is calcium imaging of subsets of neurons, where calcium imaging is representative of the cell's activity [10]. Furthermore, the connectome of the relevant neurons is known [1], which is shown in Figure 1 below.

However, as mentioned before, the connectome may not contain all the types of interactions, and, therefore, lack important communication between neurons that does not occur through physical synapses.

Experiments where certain neurons are perturbed using optogenetics are possible, but costly; therefore, these must be maintained to a minimum. Perturbation through optogenetics effectively keeps one neuron constant for a desired amount of time, allowing the network to behave as if the node and its edges were absent.

Simulation

The simulation used in this research was based on coupled ODEs. Coupled ODE models with noise added to some or all the nodes are the most commonly used simulations for cell behavior [3]. The equation used to represent an ODE is as follows

$$\tau \frac{d}{dt}\, \dot{h}_i(t) = -\gamma_i h_i + F_i\, \sigma\left(-k_i\left(\Sigma\, h_{j_{j\neq i}}\, W_{ij} + p_i\right)\right) + q_i \tag{1}$$

Where τ is the time constant, which scales the change of the neuron; γ is the self-decay rate; *F* and *k* define the shape of the sigmoid function; W_{ij} is the weight of the connection from node *j* to node *i*, and *p* and *q* represent additional external inputs to the neuron. Sigmoid function is defined as

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

And the shape is shown in Figure 2 below

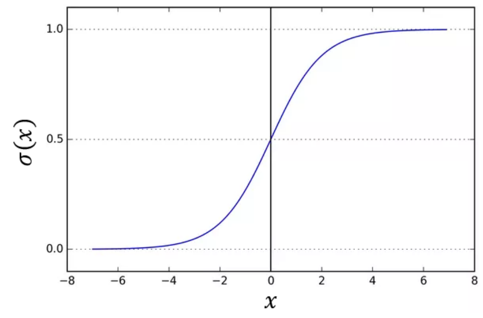


Figure 2. Sigmoid function

Therefore, *F* in equation (1) would scale the sigmoid function shown in Figure 2 vertically, while *k* scales it horizontally.

Additionally, from equation (1) it can be observed that if *q* >> *F*, then the effects of the connections defined by *W* are overpowered and masked by the large input. If the strong input is given for large periods of time, then, during that time, the partial derivate becomes a copy of the input.

Sparse input (i.e. input that has large magnitude only for short periods of time), however, allows the network to evolve according to its own intrinsic properties, while strong long-term input masks them. For this research, sparse input is preferred since the task is to solve for *W*, one of the intrinsic properties of the system.

The properties of ODEs therefore impose some restrictions on the types of simulations that can provide meaningful information about the underlying system:

1) Provide the network with sparse input.
2) Allow the system to progress normally for a considerable amount of time so that the intrinsic properties of the mechanism define the activity of each node.

Data acquisition

Abiding by the restrictions defined above, a concrete simulation was developed. The simulation used consisted of 4 nodes and 12 directed edges, including self-loops. The weight matrix is shown below

-1	0	-0.5503	0.4407
0.2523	-1	0	0.5709
-0.7127	-0.6575	-1	-0.2176
0	0.3859	0	-1

Figure 3. Weight matrix used for simulation

The value in row *i*, column *j* is equivalent to the weight of the edge that extends from node *j* to node *i*. For instance, there is no edge from node 2 to 1. All nodes have a self-regulation of -1. The simulation has positive, negative, and 0 weighted edges so that the algorithms can be evaluated on all possible types of connections.

As per the restrictions stated previously, sparse input was given to the nodes. A large input was given to one node at random at one time step, and then the network was allowed to progress without any other inputs for an arbitrarily long time, in this case 799 time steps. By the 800th time step, steady state has already been reached and maintained. After the 800th time step, a large input was given to another randomly chosen node for one time step and no input was given for the next 799 time steps. This procedure was repeated 199 times choosing the initial input and node that the input was given to randomly. The amount of times repeated was chosen arbitrarily.

For simplicity; the parameters of the ODE were fixed to be γ, F, k, and p, = 1; q = 0. Multiple simulations were done with different τ and dt values for further analysis, but for most algorithms the simulation with τ = .1, and dt = 2.5 was used. This combination was chosen arbitrarily.

A set of 800 points starting from the time step where a large input was given will be referred to as one simulation cycle. The activity for 3 simulation cycles, each of length 800, is shown below.

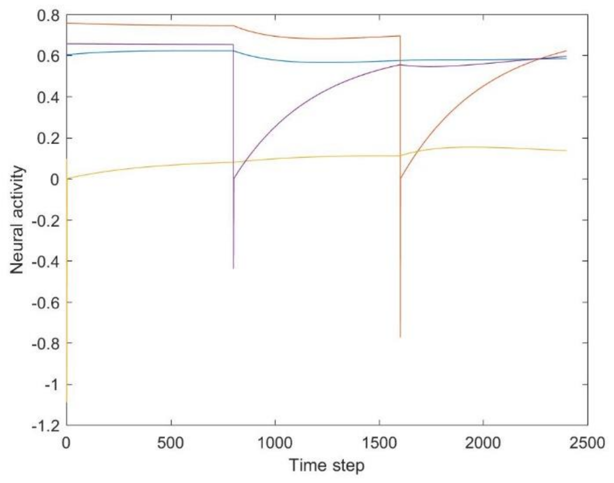


Figure 4. Neural activity of 3 simulation cycles

As can be observed, the nodes that received the large input undergo a drastic change at the beginning of the simulation, and eventually progress to their steady state.

Up to this point the training data set consists of two time series for each node of length 199 x 800 = 159,200. One of the time series is the input sequence, which consists of mostly zeros, except for nonzero values for one of the nodes at 1 + 800 c, where c is any integer from 0 to 198. The second time series is the resulting state h for each node.

Data and graph manipulation

Bayesian inference and feed-forward networks require the graph to be acyclic (more details about this in the subsequent chapters). However, biological nervous systems and their simulation networks are cyclic. Therefore, the graph must be manipulated to transform it into acyclic form, and an equivalent manipulation must be done in the data set.

Two procedures are commonly used to remove cycles from a graph: 1) join nodes that are known to be in a cycle into one super-node, 2) unfold cycles by creating two copies of the nodes, and allowing one of the copies to only have outgoing edges and the other only incoming edges. Procedure 1 is depicted in Figure 5.

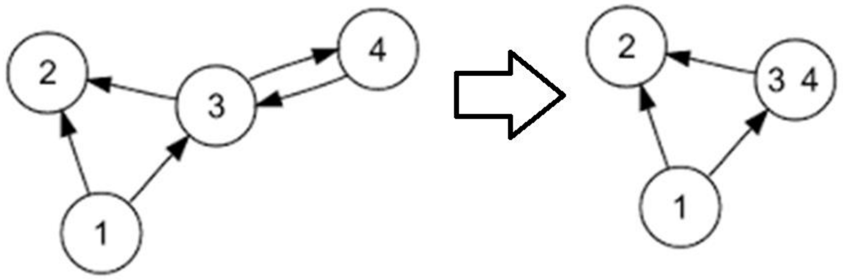


Figure 5. Transformed cyclic into acyclic graph by creating super-node

This procedure requires full knowledge of the underlying graph. However, as mentioned before, in biological systems not all connections are known. Therefore, this option is undesirable. Procedure 2 does not have this issue.

Procedure 2 creates two copies of each node. The first copy of node *i* will be named node *i'* and the second *i''* . The first set of copies is only allowed to have outgoing edges and the second only incoming ones. If the original graph has an edge from node 1 to 3, then the transformed graph will have an edge from node 1’ to 3”. This type of graph is called bipartite, and is acyclic by definition. Figure 6 depicts this transformation.

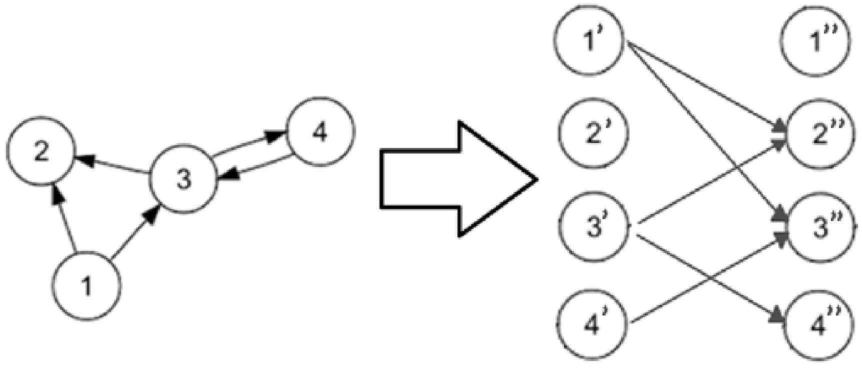


Figure 6. Creation of bipartite graph from cyclic graph

Procedure 2 does not require any prior knowledge of the edges of the graph. Suppose that our only knowledge about the network is the number of nodes *n* it contains. By fully connecting the *i'* layer to the *i”,* the bipartite graph is guaranteed to contain all the edges of the original graph, and still be acyclic.

When the graph is manipulated, the data fed to the algorithms must be updated to reflect this change. Two datasets must be specified: the input data that corresponds to the activity of the first set of copies, and the output data that corresponds to the activity of the second set of copies. Since the input and output nodes are copies of each other, the technique used to extract the input time series should be the same as that used to extract the output time series. Additionally, the output data must occur after the input data; let the difference in time between these be Δt.

There are multiple ways to define the new input and output time series. For instance, the entire dataset minus Δt could be used. However, this can lead to issues, because the entire time series contains perturbations from external inputs. Using this as the data would force the algorithm to learn not only the properties of the system, but also the perturbations that were artificially forced on the system.

An efficient way to extract the new input and output time series that avoids the issues encountered above is to sample the original dataset at a time point t and $t+\Delta t$, respectively, for every simulation cycle; call these h_t and $h_{t+\Delta t}$. Given that the simulation contains 199 of these, the resulting time series will be of length 199.

For instance, if t is set to 200, h_t would be the neural activity of all the nodes at time 200, 1000, 1800, and so forth for all 199 simulation cycles. If Δt is set to 1, and t is kept at 200, then $h_{t+\Delta t}$ would be samples of the neural activity at time 201, 1001, 1801, etc.

For easier calculations, Δh can be defined as the difference of the matrices $h_{t+\Delta t}$ and h_t.

In summary, 2 datasets from the same simulation were developed. Some of the main features of each are summarized on Table 1.

Table 1. Datasets available from simulated data

	Dataset 1: Raw data		Dataset 2: Manipulated data	
	Description	Size	Description	Size
Input	Sparse input	4 x 159,200	h_t defined above	4 x 199
Output	Neural activity	4 x 159,200	$h_{t+\Delta t}$ defined above	4 x 199
Comments	Every simulation cycle is 800 time steps		$\Delta h = h_{t+\Delta t} - h_t$	
Algorithms that use it	RNN		Bayesian, Feed-forward, and ODE reverse engineering	

Chapter III. Bayesian inference

Bayesian inference provides a way to determine dependencies between nodes in a graph. Two neurons that share any type of connection are in theory dependent on each other; therefore, the successful implementation of this algorithm would detect the presence or absence of connections between neurons.

Background

Bayesian techniques are used to construct belief networks. These are networks that contain an edge if the connected nodes are dependent, and lack one if they are not. However, these methods are used when the graph is assumed to be a directed acyclic graph (DAG) [11]. Without the acyclic property, the theory behind these algorithms falls apart. These techniques exploit the properties of conditional independence. Conditional independence allows algorithms to distinguish if node *a* is being directly affected by node b or if node c is a necessary intermediary between them. Mathematically, conditional independence is defined as

$$p(a|b,c) = p(a|b) \tag{3}$$

Where *a*, *b*, and *c* are the observations of three different nodes or sets of nodes in a graph. Equation (3) can also represent regular pairwise independence, if *c* is defined to be the empty set.

If the source graph is a DAG, and *c* is defined to be all the nodes in the graph excluding nodes *a* and *b*, equation (3) represents the lack of an edge between nodes *a* and *b* [11]. If equation (3) does not hold, then *a* and *b* are conditionally dependent. However, conditional dependence does not guarantee the existence of an edge between nodes *a* and *b* [11]. V-structures cause conditional dependence. a, b¸ and *c* are in a V-structure if *a* and *b* are both parents of node *c*. Observing *c* makes *a* and *b* dependent on each other, even if they do not share a direct edge. However, algorithms that rely on mutual information take these cases into account [4].

Methodology

[Graph used: Bipartite graph]

The belief network construction algorithm used in this method requires the calculation of mutual information of every pair of nodes. Mutual information of two nodes *i* and *j* is defined as

$$I(i,j) = \sum_{i,j} P(i,j) \log\left(\frac{P(i,j)}{P(i)P(j)}\right) \quad (4)$$

Where *P (i, j)* is defined as the joint probability of observing specific values for nodes *i* and *j*, and *P(i)* and *P(j)* are the probability of observing a specific value for node i, and node j, respectively.

However, for continuous datasets, the probability of observing a specific value is approximately 0, which requires discretization of the data: assign data points to bins that cover a range of possible values. This is non-trivial to choose. However, to solve this, run the algorithm multiple times, each time with different number of bins, and evaluate the performance; choose the optimum value of bins based on this.

The belief network construction algorithm found in literature is divided into three main phases 1) Drafting, 2) Thickening, and 3) Thinning [4]. A summary of the first phase is provided below.

Drafting:

a) Initialize a graph G with all the nodes, but no edges
b) Order all pairs of nodes from largest to smallest mutual information.
c) Remove pairs of nodes that have mutual information smaller than a predetermined threshold.
d) Consider the edge connecting the top-ranking edge; if these nodes are d-separated in graph G, add the edge to the graph.
e) Remove the top-ranking pair of nodes

f) Repeat d and e until there are no more pair of nodes to consider.

However, recall that in this method, since no cycles are allowed, the graph used is the bipartite graph. By definition, two nodes that are in opposite sides of a bipartite graph and do not share an edge are d-separated [11]. Therefore, the d) step does not reject any edges.

Thickening adds edges that had not been added due to the d-separation test. However, as mentioned before, given the properties of the graph in question, the d-separation test did not reject any edges, therefore the set of edges considered in this phase is empty.

Finally, the thinning step also relies on open paths and d-separation, which are none in this case.

Therefore, for the case of a bipartite acyclic graph, the algorithm used in literature [4] creates a graph G where all the edges that passed the threshold test on mutual information are added.

The result of this algorithm only provides information about the presence or absence of an edge. However, the successful reduction of edges would allow faster training of subsequent algorithms, since the search space of weight combination of the edges is decreased. Therefore, this technique is most useful when paired with neural networks.

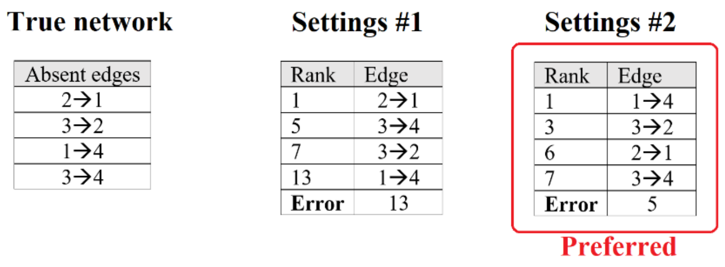
Cost function

Given that belief network construction only provides an insight into the presence or absence of an edge, the performance of this method can be evaluated by the amount of edges that were correctly removed and/or kept. However, as described in the methodology section, an edge is kept if the mutual information between the two nodes sharing the edge is greater than a predetermined threshold. The threshold is therefore an ambiguous parameter that is not specified in the available literature.

To isolate the effects of choosing a suboptimal threshold, the performance was measured as follows:

1) Rank the mutual information of all the pairs of nodes from smallest to largest.
2) Locate the four edges that should be absent, i.e. the edges that are not present in the network used to produce the data.
3) Ideally, these four edges should be the top four ranks. Therefore, define the error to be the sum of the difference of the rank of each of the four edges and 4 with a minimum boundary of 0.

An example of two rankings and their error is shown in Figure 7.



True network

Absent edges
2→1
3→2
1→4
3→4

Settings #1

Rank	Edge
1	2→1
5	3→4
7	3→2
13	1→4
Error	13

Settings #2

Rank	Edge
1	1→4
3	3→2
6	2→1
7	3→4
Error	5

Preferred

Figure 7. Evaluation of rankings

Experiments/Results

[Dataset used: 2, Manipulated data]

This technique uses as input h_t, and $h_{t+\Delta t}$. However, the three free parameters that still need to be specified the discretizing bins, t and Δt.

As defined in the methodology, the mutual information of all possibly connected pair of nodes was calculated (only pair of nodes where one of the elements is a node from the first half, and the other is from the second half were considered). This is equivalent to finding the mutual information between h_t and $h_{t+\Delta t}$, since these datasets correspond to the first and second half of the nodes, respectively. However, as mentioned in the methodology, the calculation of mutual information depends on the number of bins used to discretize the dataset.

To choose the best number of bins used to discretize the data, the cost function was calculated on the rankings produced by various numbers of bins (2 to 50). Recall that the length of the time series is 199. If the number of bins is greater than 50, then bins would contain, on average, less than 4 elements each, and mutual information for such small sets is small for all, and therefore not informative. Figure 8 shows the error based on ranking with respect to the number of bins.

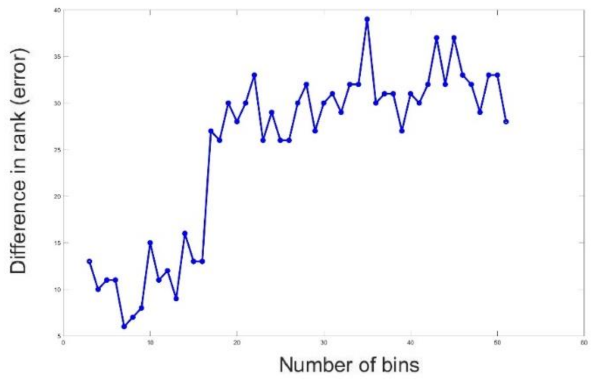


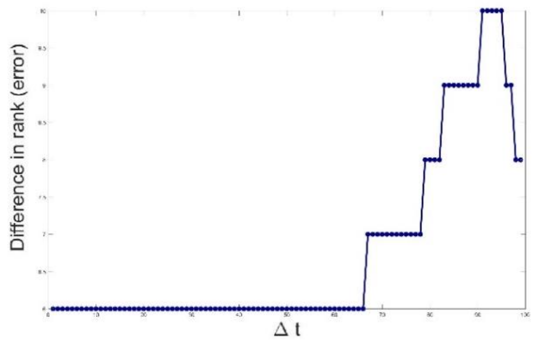
Figure 8. Error in rank as the number of bins changes

From this graph, the optimum number of bins was chosen to be 6.

Two more free parameters have to be chosen: the sample time point t for h_t, and the change in time Δt. To find the best combination of parameters the following procedure was used

1) Choose an arbitrary t
2) Evaluate all possible Δt
3) Fix Δt to be the optimum value, and evaluate all possible t
4) To verify, fix t to be the optimum value, and evaluate the performance of all Δt

Figure 9 shows the error in rank as *Δt* increases, while Figure 10 shows the error as *t* increases. Note, since the error monotonically increased after Δ*t* surpassed 100, the points afterwards are not plotted.



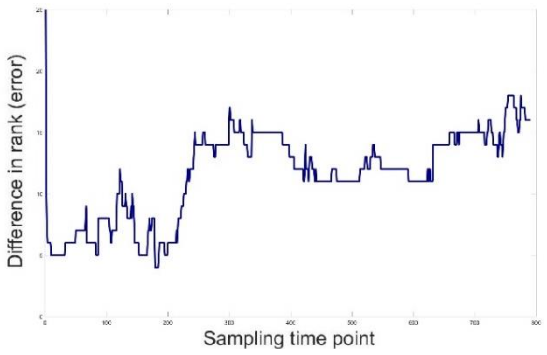


Figure 9. Error in rank as Δ*t* changes

Figure 10. Error in rank as *t* changes

As can be observed from the plots above, this technique is robust to changes in Δ*t*, which is beneficial since an increase in Δ*t* is equivalent to slower frames per second when recording neural activity of biological systems, decreasing cost and storage. The optimum *t* was obtained at time step 182, however, time points before 200 have on average smaller error than points after 200, in general any point before 200 performs better than any point after.

After setting the free parameters to be the optimum values, the algorithm was run on the data. For comparison, edges were ordered by their true weight from smallest to largest; therefore, the first four edges are the absent edges. The mutual information of each edge was then plotted on the same graph. The ideal result would be to have a clear bimodal distribution where the first four edges are small compared to the rest. Figure 11 shows the distribution obtained.

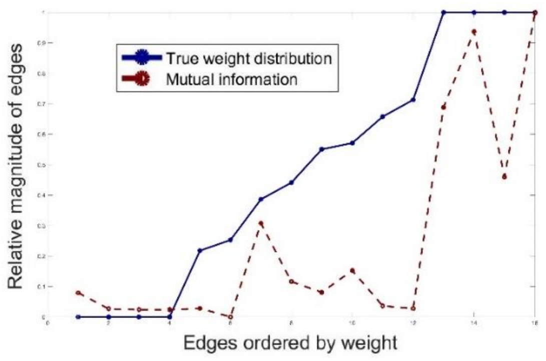


Figure 11. Mutual information of edges ordered by weight

Even though the edges that are absent have low mutual information, there are other edges that are present that have comparable mutual information. For instance, if the threshold set to reject edges is .1, then the error rate is 31.25%. Table 2 shows a breakdown of the mistakes made by this threshold.

Table 2. Error breakdown of mutual information threshold = .1

	Edge is absent	Edge is present
Edge was kept	0%	58.3%
Edge was removed	100%	41.7%

5 edges that should have been kept were removed, but all edges that were absent were successfully removed. As can be observed in Figure 11, any threshold bigger than 0 would incorrectly remove at least one edge, since the edge with the least mutual information is an edge that is present, which is an undesired result, since incorrect removal of edges is one of the issues that this research is trying to solve.

Discussion

Some of the positive results of this algorithm are that it is robust to changes in Δt, which allows recordings to have less frames per second, and there seems to be some correlation between the magnitude of the weight of the edge and the mutual information of the nodes connected by this, as shown in Figure11, which is promising.

However, edges that should be present are removed. This deems the algorithm useless because finding missing edges from the true network is one of the priorities of this research, and the algorithm clearly fails in this task.

Nevertheless, there are some possible ways to improve the algorithm, such as feeding it more data. The more data, the more sampling points are available, which increases the amount of data in each bin, improving the accuracy of the mutual information calculation. By having accurate mutual information, then dependency and independency calculations will also be more accurate, improving performance.

Chapter IV. Artificial Neural Networks

Artificial neural networks (ANN) provide a way to model behavior by defining very simple parameters. These parameters are mainly the ANN architecture, cost function, and the input and target time series. Once those parameters are defined, the training algorithm takes care of the rest.

Background

Artificial neural networks are a combination of nodes and edges. Each edge feeds the output of one node as the input to another. Each node imposes a nonlinearity on the sum of its inputs. The most commonly used nonlinearities in neural networks are sigmoid, rectified linear units, and hyper tangent. The nonlinearity function of each neuron is called the *activation function* of the neuron. There are currently several types of neural networks. A neural network is a combination of several interconnected nodes called neurons. The way the nodes are connected defines the type of neural network it is. If the connections create a cycle, then it is a type of recurrent neural network

(RNN), if it contains no cycles, then it is a feed-forward network. Within these two kinds of neural networks there are several more subtypes.

ANNs are trained using stochastic gradient descent to try to find the optimum set of parameters that best minimizes a cost function. The error is back propagated from the output layer to the input layer.

The neural networks considered in this research have a very limited architecture. One of the neural networks used is a regular RNN, and the other is a two-layer feedforward network.

RNNs however have issues when training such as vanishing gradient descent, meaning that the error encountered at time point *t* quickly dies off and does not affect time steps at a reasonable distance. By performing bipartite transformation on the cyclic graph, the network can now be modeled by a very small, well behaved 2-layer feed-forward network.

General specifications

If the goal is to keep the structure of the artificial neural network the same as that of the real nervous system, then the artificial neural network used must be an RNN, since the nervous system has several cycles [8]. However, by applying the bipartite transformation specified in "data and graph manipulation", the cyclic graph can be transformed to an acyclic form, and modeled by a regular feed forward neural network.

Structure

Both ANN architectures will share most of the specifications. For instance, no hidden units are desired in either the feedforward or recurrent networks. Given an input to the network, there exists a target neural activity for each node in the graph. Any difference between the target activity and the activity of the artificial node must be penalized, and this can only be accomplished if the nodes in question are part of the output layer. The trained parameters are therefore only the weights of the connections between the nodes.

Given that the ODEs used in the simulation are sigmoid based, the activation function for all the neurons in both architectures was set to be the sigmoid function.

Further constraints in the architecture can be chosen to be imposed on either type of network. For instance, if for any reason, it is determined that an edge between two nodes cannot exist, this edge can be removed from the network, and it is guaranteed to be removed from the network entirely. Regardless of the architecture and training algorithm, any edge that does not exist in the initialization will not exist after training. This allows the network to search through less possible weight conformations.

Cost function

The target is to obtain a weight matrix, which contains the weights of all the edges in the graph, closest to the weight matrix that produced the neural activity observed. However, the real weight matrix is not known for biological systems. Therefore, a cost function based on observable data must be defined. Given that the output of the network is defined by the input sequence and weight matrix, the error of the output sequence should be related to the cost of the weight matrix. Consequently, the next best option is to define cost as the square difference between the neural activity of the biological system and the artificial network.

Evaluation

To evaluate the performance of the neural networks on the true task at hand: estimate the weight matrix of the graph, a metric must be defined. Regular Euclidean distance between the real and estimated weight matrices is not desired because scalar multiples of the weights must not be penalized as incorrect as long as the scalar is positive. A distance metric that does not penalize scalar multiplication is cosine distance, which is defined as

$$distance = 1 - \frac{A \cdot B}{||A||_2 ||B||_2} \tag{5}$$

Where A and B are the two vectors compared. Therefore, the performance of either neural network will be defined by the cosine distance of the trained weight matrix and the real underlying weight matrix, which is shown in Figure 3.

Regularization

In addition to the loss function used to train the network, which is the mean squared error (MSE) of the target sequence and the output of the network, L1 or L2 regularization is often added to impose limitations on the edges of the network [12]. Table 3 provides a summary of L1 and L2 regularization based on literature.

Table 3. Summary of L1 and L2 regularizations [12]

L1	Cost = Loss + λ || θ ||	Minimizes number of edges
L2	Cost = Loss + λ || θ ||^2	Minimizes weight of edges

The regularization is imposed by adding a penalization to the cost function, which is minimized by stochastic gradient descent when training an artificial neural network.

These tools can be applied to any type of neural networks. The training techniques for the two neural network architectures used in this thesis are described below.

Methodology – Recurrent neural networks

[Graph used: Original cyclic graph]

RNNs are usually trained using back-propagation through time (BPTT), where the neural network is "unfolded" for some predetermined number of time steps. Figure 12 shows an example of an RNN unfolded for three time steps.

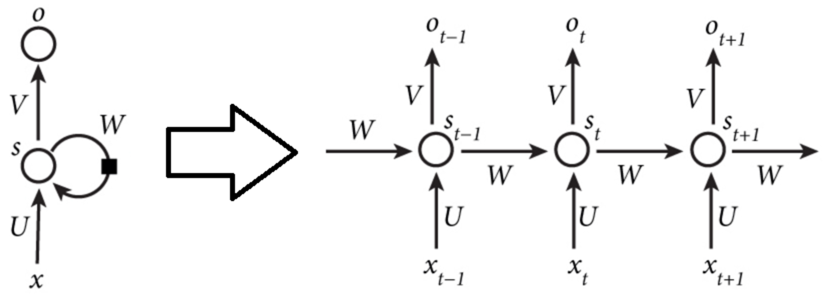


Figure 12. "Unfolding" of an RNN [15]

Error is propagated backwards from every output to the weight matrix. Given that in the case of interest there are no hidden units, the state variable *s* in Figure 12 at time *t* will also be the output *o* at time *t*. Effectively, *V* is fixed to be all ones.

Experiments/Results – Recurrent neural networks

[Dataset used: 1, Raw data]

RNN was initialized to have four nodes, like the simulation, but in this case the graph was fully connected. The input sequence was one simulation cycle of the sparse input given as input to the original graph. The target series was one simulation cycle, and the target function was the mean squared error (MSE) between the target series and the output of the neurons, with no L1 or L2 regularization. The network was trained with 400 epochs. Figure 13 shows the error of the network after every epoch.

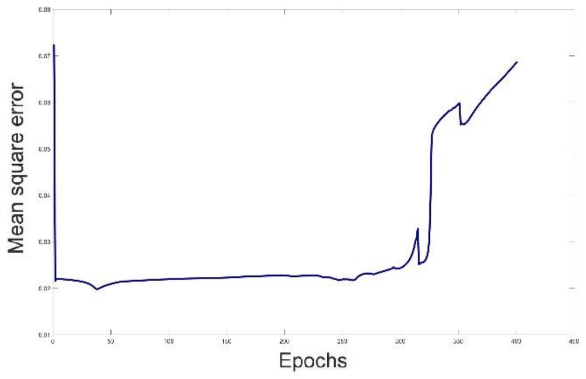


Figure 13. MSE of RNN output and target time series

As can be observed, mean squared error is kept under .08. The last 100 epochs seem to hurt performance, which can represent a local minimum that misled the stochastic gradient descent. Nevertheless, the error is consistently small.

Figure 14 shows the resulting cosine distance of the learned weight matrix and the real weight matrix.

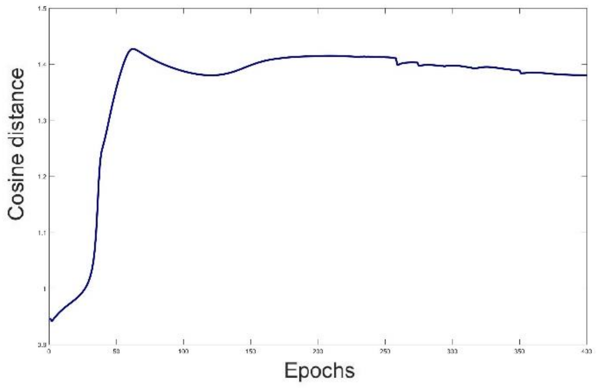


Figure 14. Cosine distance of real vs trained weight matrix

As can be observed, cosine distance rapidly increases and then stays constant. Furthermore, the distance is very high, meaning that the learned weight matrix is far from correct.

L1 regularization slightly improved performance of the learned weight matrix. Also, limiting the edges to only those present in the ground truth network improved performance. The respective error curves are shown in Figure 15 and 16.

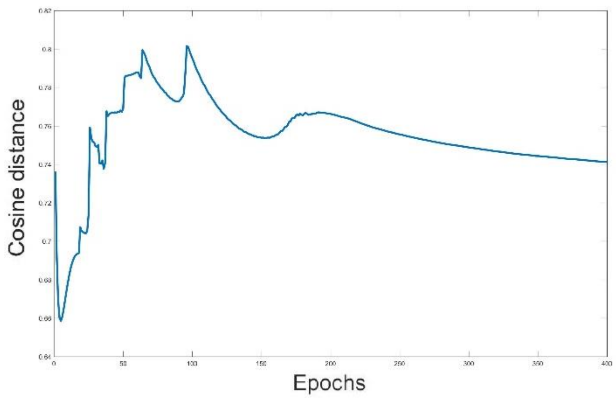


Figure 15. Error of weight matrix with L1 reg.

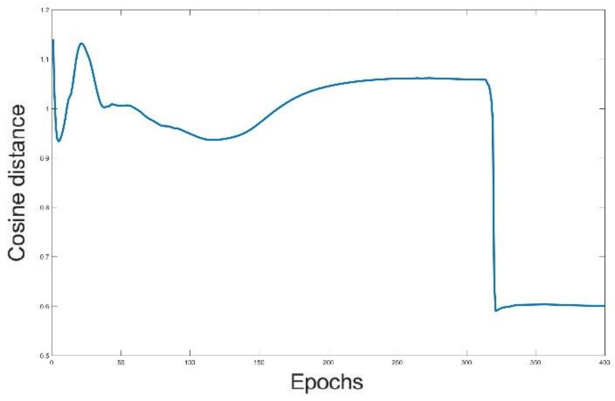


Figure 16. Error of weight matrix with L1 and reduction of edges

Nevertheless, when cosine distance was smallest, using L1 regularization and reduction of edges, the learned weight matrix still assigned the incorrect sign to 4 out of the 12 remaining edges. The learned weight matrix is shown below.

0.2482	0	-179.8	-1.485
9.777	10.88	0	-94.71
-7.797	-0.7620	-224.8	-8.733
0	0.6035	0	-4.170

Figure 17. Learned weight matrix. Edges that were assigned the incorrect signed are highlighted

Interestingly, the errors are limited to only two of the rows. Further analysis can be made to see if imposing self-regulation, which is common in biological systems, by penalizing positive self-loops helps performance. However, these are assumptions that may not always be correct, and the less external influence in the learning algorithm, the better.

Discussion – Recurrent neural networks

RNNs performed poorly. However, performance was greatly improved when the number of edges was reduced to those known to be present in the network. This suggests that if the Bayesian inference algorithm was fixed, then a combination of Bayesian inference followed by RNN learning would be promising.

RNN is the only technique addressed in this thesis that used the raw data, which is an advantage over other techniques that can fail in the absence of external input. Nevertheless, neural networks, specially RNNs are known to be "data hungry," therefore the more meaningful data given to the network, the better it will perform theoretically.

Methodology – Feed forward neural network

[Graph used: Bipartite graph]

Regular feed forward neural networks are trained through back-propagation from the output layer to the input layer [7]. In this case, the network is comprised of only these two layers, input and output. Unless constraints are imposed on the possible edges of the network, all input nodes are fully connected with the output layer, this is the same structure as the resulting bipartite graph obtained when cycles are removed from the original network. Back-propagation will propagate the error based on the predetermined cost function from the output to the input layer. However, due to the limited architecture of the network, the only free parameters are the weights, and thus are updated according to the error of the output of the network.

Experiments/Results – Feed-forward neural networks

[Dataset used: 2. Manipulated data]

The input data is h_t, and the target series is $h_{t+\Delta t}$. The free parameters of this technique that must be determined are t, Δt, and the use of tools such as extra regularization and/or reduction of edges.

The performance of the neural network varying Δt from 1 to 550 was evaluated, and shown in Figure 18. All training was done for 400 epochs.

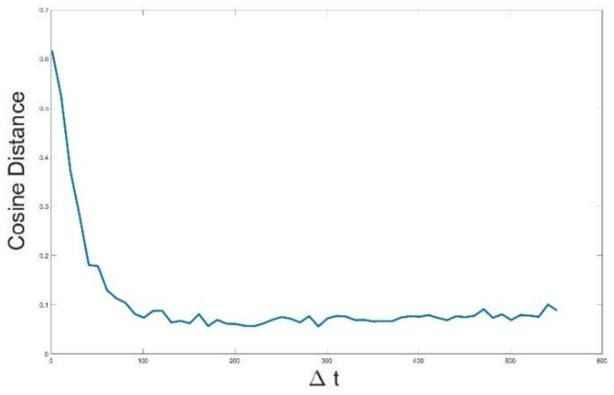


Figure 18. Cosine distance of weight matrices as Δt increased in feed-forward network

t was also varied from 1 to 550 setting Δt = 200. The cosine distance of the weight matrix as t changed is shown in Figure 19.

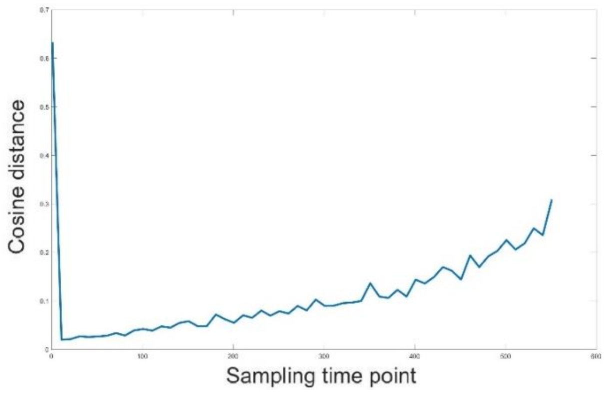


Figure 19. Cosine distance as sampling point t increased in feed-forward new

Once *t* and Δ*t* were set to be the values that produced the least error rate, training was also done with regularization, with reduction of the edges known to be absent, and with both. The resulting error after the training of each is shown in Table 4 below.

Table 4. Error rate after 400 epochs using different training specs

Training specs	Error
Normal	**.0175**
Regularization	.1110
Reduction	**.0197**
Regularization + reduction	.1067

In this case, regularization and edge reduction did not improve performance. However, the learned weight matrix of the two best performing training specs (normal, and reduction) are shown below.

-5.428	-0.1390	-1.725	1.757
0.3615	-4.779	0.2800	1.467
-1.827	-1.972	-3.667	-0.9243
-0.1378	1.229	0.1797	-5.257

-5.415	0	-1.519	1.647
0.5167	-4.728	0	1.517
-1.746	-2.150	-3.554	-0.8297
0	1.069	0	-5.262

Figure 20. Weight matrix learned from normal specs and weight matrix learned imposing reduction

All signs were correctly learned for the reduced case. In the normal case, all signs of the edges present were correctly learned. Furthermore, the absent edges had the smallest magnitudes. This is extraordinary performance.

Discussion – Feed-forward neural networks

The accuracy of this method was extremely high. Furthermore Figure 18 shows extreme robustness to changes in Δt, which allows recordings to decrease the number of frames per second, and therefore also decrease storage.

As expected, close to the perturbation time, error is highest because the properties of the system are overpowered by the large input. Also, unlike ODEs, this method is robust to noise, and faulty data points.

However, partially responsible for the success is the fact that the activation function of the nodes is the same nonlinearity used to produce the simulation, which might not necessarily be true in real biological systems.

Chapter V. ODE reverse engineering

Reverse engineering of ODEs is the use of matrix operations to solve for the parameter *W* in the definition of an ODE, such as that shown in equation (1), which represents the weight matrix of the neurons in the model. However, some of the matrix calculations required to solve for *W* is the inverse of the time series h_t, which is sometimes challenging since matrices that are not full rank are not invertible, and real data is oftentimes not [13]. Nevertheless, research has shown that perturbations of all the modules in the guarantees this property.

Background

ODEs are often used to simulate biological neural activity, such as how they are used in this thesis to produce the simulation data. Since ODEs approximate the behavior of neurons; the parameter *W*, which determines the interaction between nodes in the network, should also approximate interactions in real biological systems. ODEs are usually used to find the change in neural activity of a node. However, ODEs can be used to do the reverse. From an observed state *h*, and Δh, the weight matrix can be solved.

Methodology

[Graph used: Bipartite graph]

Variable manipulation and discretization of the derivative simplifies equation (1) to matrix operations as shown below

$$[F_i \sigma\left(k_i \Sigma \mathrm{h}_{\mathrm{j}_{\mathrm{j}\neq i}}\, w_{\mathrm{ij}} + \mathrm{p_i}\right) - \gamma_i h_i]\; dt/\tau_i = \Delta h_i \tag{6}$$

$$\left[F \cdot \sigma\left(\overrightarrow{k}\ \cdot\ \vec{h} \times W\right) - h \times\ \vec{\gamma}\right]\left(\frac{dt}{\tau}\right) = \Delta h \tag{7}$$

Where σ is the sigmoid function, and its inverse is defined as

$$\sigma^{-1}(x) =\ -\log\left(\frac{1}{x} - 1\right) \tag{8}$$

Combining equation (7) and (8) provides a solution for *W*, which is

$$W\ = \sigma^{-1}\left(\frac{\left(\frac{\Delta h\,\tau}{dt} + h\times\vec{\gamma}\right)}{F}\right)\left(\overrightarrow{k}\ \cdot\ \vec{h}\right)^{-1} \tag{9}$$

However, there are many unknown parameters, such as *F, γ,* and *k;* for simplicity, we will assume these to be all ones. Therefore equation (9) can be rewritten as

$$W\ = \sigma^{-1}\left(\frac{\Delta h\,\tau}{dt} + \vec{h}\right)\vec{h}^{-1} \tag{10}$$

Previous research has successfully tackled the issue of possible invertibility of h_t by performing successive perturbations on all the modules of a network [14], which is already done in the simulation data.

Once the weight matrix is solved, this can be applied to a test input sequence to find the resulting estimated Δh. This Δh will be defined as the output of the network, and it will be compared against the real Δh of the simulation.

Cost function and evaluation

Cost function

Like neural networks, the cost function of the time series was mean square error.

Evaluation

Since the additional parameters are set to be the same as those of the simulation, the weight matrices should be identical; therefore, the metrics used to evaluate the estimated weight matrix is Euclidean distance.

Experiments/Results

[Data used: 2. Manipulated data]

W was solved by using the time series h_t and Δh and the τ and dt were extracted from the parameters of the simulation. The free parameters that still need to be optimally chosen are t, and Δt.

Therefore, various t, and Δt were tested, for different τ and dt parameters.

The effect of increasing Δt on the cost of the test series and that of the weight matrix for various τ/dt is shown in Figure 21.

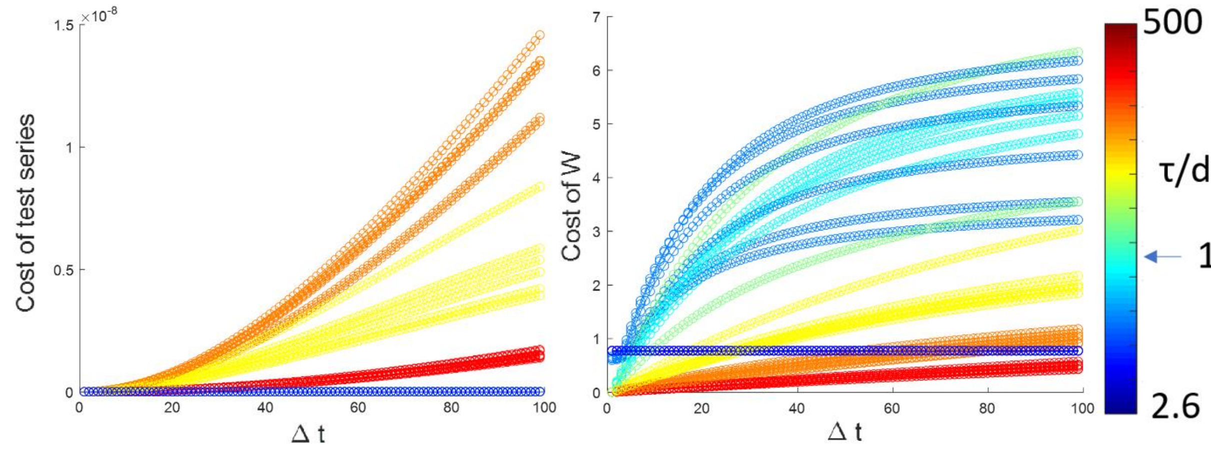


Figure 21. Cost of test series and cost of W as Δ*t* increases for various τ/*dt*

As shown in Figure 21, time series with similar τ/*dt* behave similarly. This is because τ/*dt* defines the decay of the time series. Therefore, time series with the same τ/*dt* will reach steady state at the same rate.

The effect of increasing the sampling point *t* on the two costs is shown in Figure 22.

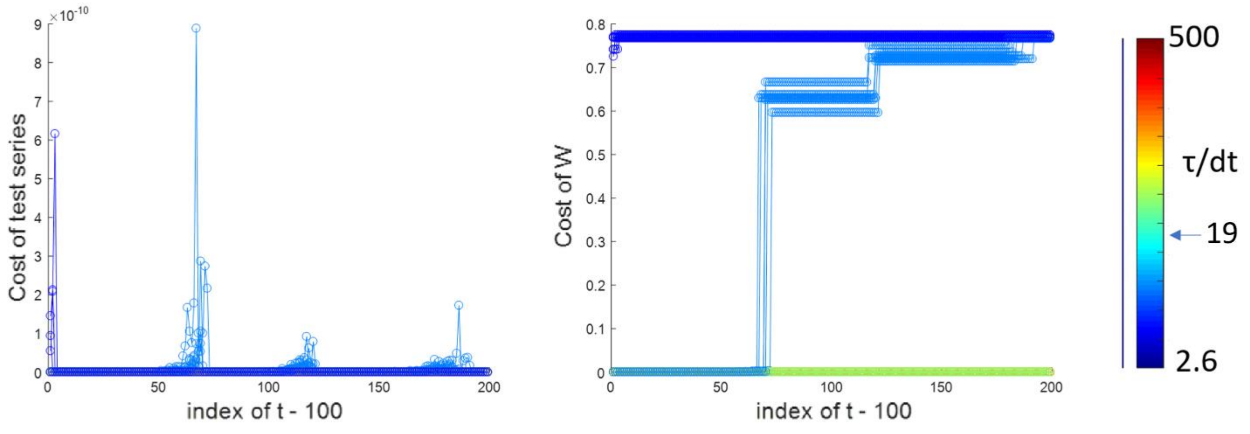


Figure 22. Cost of test series and cost of W as *t* increases for various τ/*dt*

As τ/*dt* increases, the cost of W decreases for larger *t*. This is again due to the slower decay of these time series; therefore, sampling at a later time point still provides information about the system. If the system has reached its steady state, no node affects one another, and therefore any matrix W can produce the resulting time series. It is evident again that time series with the same τ/*dt* behave the same regarding the cost of the time series and of W. The jumps of the cost of W occur at the same *t* as the spikes on the cost of the test series.

Taking any of the optimum configurations, which minimize the cost of W, and solving for W results in the same weight matrix W as the one used to produce the simulation. Therefore, the error in this case is 0%. However, it is worth noting that there are conformations that minimize the cost of the test series, but produce a high cost of W, as can be observed in Figure 22.

Discussion

Reverse engineering of ODEs performed extremely well. However, many of the parameters had to be hard-coded based on the knowledge of the specifications used to produce the simulation data. This is not projectable to the analysis of biological data. However, most of the parameters can be combined and/or derived with further analysis of the data.

A suggested approach to solve for the missing parameters, which should be evaluated in the future is described below.

Starting from equation (1), some manipulations can be done on the ODE formula minimize the amount of parameter needed to be found. For instance, the entire equation can be divided by γ and the new parameter renamed to reflect this

$$\tau' \frac{d}{dt}\, \dot{h}_i(t) = -h_i + F'_i\, \sigma\left(-k_i \left(\Sigma\, h_{j_{j\neq i}}\, W_{ij} + p_i\right)\right) + q'_i \qquad (11)$$

From here, as observed in the results, the time that it takes a node to reach steady state from undergoing a perturbation is fully determined by τ'/*dt*.

If τ'/*dt* is known, then

$$\tau' \frac{d}{dt} \dot{h}_i(t) + h_i = F'_i \, \sigma\left(-k_i \left(\Sigma\, h_{j_{j\neq i}} W_{ij} + p_i\right)\right) + q'_i \tag{12}$$

The left side of the equation is completely determined. According to the right side, the this will have the shape of a sigmoid, such as that in Figure 2. However, the difference between the minimum and maximum will be equivalent of the distribution will be equivalent to *F'*. The offset between the minimum of the distribution from 0 is equivalent to *q'*. Lastly, *k* can be included into *W* as a different scalar factor applied to each column.

By doing this, ODE reverse engineering can potentially be applicable to networks were none of its parameters are known.

Chapter VI. Conclusion

The goal of estimating the connectivity of a network using time series of the nodes in the network was successfully accomplished. The two best performing methods were feed-forward neural network and ODE reverse engineering.

Feed-forward neural network was shown to be robust to changes in sampling rate and sampling points. Additionally, this technique imposes no constraints in data, and is also more tolerant to noise, a faulty data point does not have strong repercussions on the resulting solution. Because of these reasons, this method is also expected to generalize well to biological systems.

ODE reverse engineering achieved the performance; however, most of the parameters that define the ODE were taken to be known, which is not true for biological systems. Furthermore, noise or insufficient perturbations of the network modules could make it impossible to solve for the weight matrix.

The two methods that were the least successful were Bayesian inference and RNNs. Both methods would benefit from more data. Bayesian inference requires discretization of data, and then the resulting discretized data is used to form an estimate of the underlying distribution that defines the data. More data would allow better approximation of the correct discretization, and of the underlying distribution. RNN achieved small time series error, but high cost for the weight matrix. Therefore, more data, if it differs from the data already available, would provide the network with more information about the possible interactions between the nodes.

A procedure that successfully extracts connectivity information from recorded neural activity of a small nervous system can potentially be generalized to extract meaningful information from neural activity of more complex systems, and thus provide insight about the structure of these. Given the robustness of feed-forward neural network this appears to be the most appropriate method for future work.

Bibliography

[1] S. W. Flavell, N. Pokala, E. Z. Macosko, D. R. Albrecht, J. Larsch, and C. I. Bargmann, “Serotonin and the neuropeptide PDF initiate and extend opposing behavioral states in C. elegans,” *Cell*, vol. 154, pp. 1023-1035, 2013.

[2] L. Abbott, “Unmarrying the perceptron: Lessons in cerebellar computing from Fish and Flies,” *Phillip A. Sharp Lecture in Neural Circuits*, 2017.

[3] M. L. Hines and N. T. Carnevale, “The NEURON simulation environment,” *Neural Computation*, vol. 9, no. 6, pp. 1179-1209, 1997.

[4] J. Cheng, D. A. Bell, W. Liu, “An algorithm for Bayesian belief network construction from data,” *Proceedings of Artificial Intelligence and Statistics*, pp. 83-90, 1997.

[5] E. Sontag, A. Kiyatkin and B. Kholodenko, “Inferring dynamic architecture of cellular networks using time series of gene expression, protein and metabolite data,” *Bioinformatics*, vol. 20, no 12, pp. 1877-1886, 2004.

[6] W. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115-133, 1943.

[7] M. Hermans and B. Schrauwen, “Training and analyzing deep recurrent neural networks,” *Advances in Neural Information Processing Systems (NIPS Proceedings),* no. 26, 2013.

[8] H. Jaeger, “A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the ‘echo state network’ approach,” *German National Research Center for Information Technology, GMD Report 159*, pp. 48-94, 2002.

[9] D. Sussillo and O. Barak, “Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks,” *Neural Computation*, vol. 25 pp. 626-649, 2013.

[10] B. F. Grewe, D. Langer, H. Kasper, B. M. Kampa, and F. Helmchen, “High-speed in vivo calcium imaging reveals neural network activity with near-millisecond precision,” *Nature methods*, vol. 7, pp. 399-405, 2010.

[11] D. Shah, “Learning structure in directed graphical models,” *Algorithms for Inference, MIT, Lecture 21*, 2016.

[12] R. C. Moore and J. DeNero, “L1 and L2 regularization for multiclass hinge loss models,” *Symposium on Machine Learning in Speech and Language Processing,* 2011.

[13] A. Barnett, “The invertible matrix theorem,” *M22F06 lecture Dartmouth*, math.dartmouth.edu, 2006.

[14] B. N. Kholodenko, A. Kiyatkin, F. J. Bruggeman, E. Sontag, H. V. Westerhoff, and J. B. Hoek, “Untangling the wires: A strategy to trace functional interactions in signaling and gene networks,” *Cell Biology PNAS,* vol. 99, no. 20, 2002.

List of Figures