Modeling *C. elegans* Nervous System's Behavior using Machine Learning Techniques

Gonçalo Mestre
goncalo.mestre@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2021

Abstract

Given its inner complexity and the potential for human advancement resulting from a deeper understanding of its structure and functioning, the study of the human brain and nervous system is one of the greatest challenges in computational neuroscience. In order to understand its dynamics, insight may be gained from deeper knowledge of simpler and smaller organisms like the *Caenorhabditis Elegans* (*C. elegans*).

This nematode has a rather small nervous system that allowed for its almost complete description from different perspectives and scales, leading to the creation of detailed models based on its complete connectome. Scaling these models for other organisms is not an easy task as the amount of information used to completely describe its nervous system may lead to computationally demanding, potentially intractable models, whereas sometimes one is only interested in accurately predicting the response of the nervous system to a variety of stimuli.

With the goal of modeling the input-output behavior of the *C. elegans* nervous system, this work addresses the problem of determining whether black-box techniques can accurately predict such behavior using only observable peripheral information. To accomplish this, a high-fidelity, detailed model of the worm is used within the NEURON simulator for generating datasets corresponding to two specific, well characterized, behaviors. This data is then used to test the ability of five different artificial neural network architectures to model these behaviors. Two of these architectures in particular, the LSTM and GRU, prove successful in modeling the system behavior with high accuracy as well as in creating models that are able to replicate two different behaviors on a diverse set of output neurons.

Keywords: Nervous Systems, *C. Elegans*, Time Series Prediction, Machine Learning, Artificial Neural Networks

1. Introduction

The study of the human brain and nervous system is a very challenging and stimulating problem and one of the greatest challenges in computational neuroscience. This is such a compelling and difficult problem due to the inordinate multitude of tasks it can accomplish and the enormous complexity of the human brain that is composed of approximately 8.61×10^{10} neurons [8], making it one of the largest complex systems available to study.

In order to study such a complex nervous systems, insight may be gained from simpler and smaller organisms, for which the underlying principles of network organization and information processing are easier to postulate due to their simplicity and availability of recordings. One of these is the *Caenorhabditis Elegans* (*C. elegans*), a small nematode of about 1 mm in length, for which synthetic data of the nervous system is used in this work to test different machine learning techniques.

The *C.elegans* has a compact nervous system consisting of less than 1000 cells across all sexes and around 15000 connections [14]. The complete connectome of geometrically distributed neurons and synapses for the *C. elegans* is composed of 302 neurons for the adult hermaphrodite [41], and of 385 neurons for the adult male [14]. For the male *C. elegans* the complete 3D reconstructions of the neurons are not published yet [18], with only the the posterior nervous system 3D recordings of 144 neurons having been published [24].

The connectome can be seen as a graph where the neurons and synapses are represented by the nodes and the edges in it, respectively. It contains complex descriptions which may lead to complicated models, implying more computationally demanding, potentially intractable simulations of its dynamic behavior. This increased complexity is a side effect of the detailed modeling of the internal structure, whereas often one is only interested in peripheral input-output behavior.

Unlike white-box models, black-box models do not seek to understand the internal structure of a system, but merely to mimic and predict its input-output behavior. While this limits the amount of insight one can obtain into the governing phenomena, it provides a simplified path to modeling as one does not need to fully understand the internal dynamics but merely predict its external behavior, thus leading to computationally simpler, i.e. reduced, models.

In order to create an approach to model peripheral input-output behavior of complex nervous systems, this work tests different machine learning techniques on their ability to create data-driven black box models of peripheral input-output behavior of the *C. elegans* nervous system.

These techniques were not only tested in their ability to accurately reproduce the system behavior, but also on how reduced the models created can be. The models were experimented with data using different sampling rates, testing the effect of this sampling rate on their performance. A comparison on the performance of the different models was also conducted, taking into account the size of the models. The models were also tested on their ability to reproduce the behavior of the system when deprived of all the inputs, and on their ability to replicate different behaviors simultaneously.

2. *Caenorhabditis Elegans*

C. elegans is a small transparent non-parasitic soil nematode, with a cylindrical body of approximately 1.0mm in length and a weight of around 5µg.

Despite the rather small nervous system of this organism, it is able to process various stimuli from the environment. These stimuli allow the worm to solve basic but essential problems for its survivability like feeding, toxin detection, mate-finding, predator avoidance or even forward and backward locomotion [45]. This ability to solve basic problems makes this organism a good benchmark for techniques employed in computational neuroscience, along with the fact that its cell-lineage and anatomy are invariant.

Due to the usefulness of this organism in computational neuroscience studies, extensive research has already been done in order to gain a good understanding of the behavioral and structural biology of the *C. elegans*. Since it has a rather small nervous system, this research has already resulted in its almost complete description from different perspectives and scales. This descriptions are available in databases of genetics and genomics [5], [1], [22], online books [6], [23], atlases of neurobiology, structural and behavioral anatomy [15] and electron micrographs and associated data [16].

Even though there is a lot of information available on the *C. elegans* nervous system, creating a model that encapsulates all of this information is not a trivial task due to its diversity and complexity. Nevertheless, there are available open-source databases of digitally 3D neuron reconstructions [2], [3], [20] and computational models [4], [39]. These models are based on the connectome, which is the map of the neuronal connections in the nervous system, with the detailed information described previously.

Research in this area assumes very diverse forms with works modeling different parts of the system, in tasks as different as touch induced movement [35], [9], chemotaxis [37] [46], nictation [9] or temperature and smell based movement [38]. Other works also try to model the entire system, showing how diverse is the research on modeling this nervous system and its behavior [33].

Usually these techniques are white-box techniques, with the exception of [9], where a black-box technique is used to find a model. Contrary to this work, most of the research found on modeling the *C. elegans* nervous system also usually involves comparing the behavior of the obtained models with the behavior of real nematodes in a laboratory or using data obtained in a laboratory.

This work does not use real *C. elegans* data, instead, different datasets concerning the behavior of this nematode are generated using the NEURON simulator [11] and the model also used in [9] with the collaboration of the authors.

3. Generated Data

The generated data concerns two different behaviors, which correspond to a Posterior Touch Response Stimulation (PTRS) scenario, and a Nictation scenario.

For each of these scenarios a dataset with a smaller number of outputs was generated, in order to minimize computational costs in the experiments made. Then, for each of these two datasets, different versions were obtained using a time step between samples of 0.05ms, 0.1ms, 0.5ms, 1ms, 5ms and 10ms. Later, a dataset with the full set of outputs using a time step between samples of 0.5ms was also obtained for each of the scenarios, and a last dataset combining the data from both scenarios was generated.

Each dataset randomly divided into three sets, with 50% of the data for the training set and 25% for the validation and test sets each, with each dataset having in total 40 different sequences. For all the datasets, the different stimuli provided to the system in each scenario were chosen in a random way, combining different pulses and ramps as current inputs.

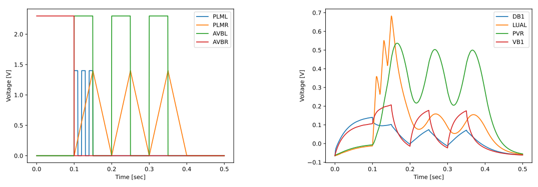

Figure 1: Example inputs (left) and outputs (right) for one of the sequences generated for the first dataset on the PTRS scenario.

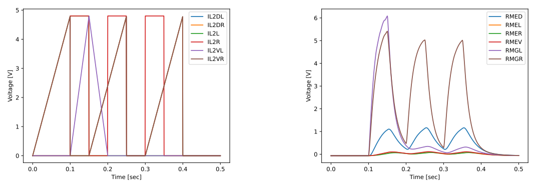

Figure 2: Example inputs (left) and outputs (right) for one of the sequences generated for the first dataset on the Nictation scenario.

3.1. Posterior Touch Response Stimulation

In this scenario, described in [25], the response of the *C. elegans* when stimulated in the posterior part of its body by touch is represented, and is known to produce a forward crawling response by the worm. This behavior of the nematode's nervous system usually involves stimulating PLM sensory neurons, which excite the motor neurons associated with forward crawling, and AVB interneurons, known to be driver cells for the forward movement of the worm. This behavior is known to affect over one hundred neurons of the *C. elegans* nervous system with the majority of these being motor neurons [9], but for this case, a sample known to be large enough, composed of 16 neurons was chosen. These specific neurons are known to show strong responses during the behavior being studied: AS1, AS7, DA1, DA9, DB1, DB5, DD1, DVA, LUAL, PHCL, PLML, PVCL, PVR, SIBVL, VB1, VD1.

The stimuli provided to the PLM and AVB neurons were given as described previously with a maximum current of 1.4nA for PLM neurons and 2.3nA for AVB neurons. The subset of outputs used for the first datasets on this scenario only includes as output neurons DB1, LUAL, PVR and VB1. An example of one of the sequences generated can be seen in Figure 1, with the corresponding inputs and outputs.

3.2. Nictation

The Nictation scenario studied refers to a behavior in which the worm stands on its tail and waves its head in three dimensions. This behavior is regulated by a set of sensory neurons located in the head of the worm, IL2 neurons: IL2DL, IL2DR, IL2L, IL2R, IL2VL IL2VR [27]. It is known that when these six neurons are stimulated, the worm tends to produce this movement, in which strong activity can usually be observed mostly in the SMD motor neurons (SMDDL, SMDDR, SMDVL, SMDVR) and in the neurons associated with the head muscles: RMED, RMEL, RMER, RMEV, RMGL, RMGR, RMHL and RMHR.

The stimuli provided to the IL2 neurons in a random way, uses a maximum current of 4.8nA. The subset of outputs used for the first dataset in this scenario concerns only the data of 6 neurons: RMED, RMEL, RMER, RMEV, RMGL and RMGR. An example of one of the sequences generated can be found in Figure 2, with the corresponding inputs and outputs.

4. Machine Learning Models

The representation obtained for the peripheral input-output behavior of the *C. elegans* from simulations of the realistic connectome-based model is in the form of multivariate time series. Since the task of obtaining a set of multivariate time series, based on another set of multivariate time series, is a sequence to sequence modeling task, different works on this topic were reviewed. A task with some similarities to the task performed is also the task of forecasting time series, for which different works were also reviewed in order to find models that are suitable to be adapted for this work.

Based on the review performed, for which the reader is referred to [32], five different neural network architectures were used, testing their ability to model the *C. elegans* behavior. The different architectures used were: Time Delay Neural Network (TDNN), Neural Autoregressive Exogenous Model (NARX), Recurrent Neural Network (RNN), Long Short-Term Memory Unit (LSTM) and Gated Recurrent Unit (GRU).

4.1. Time Delay Neural Network

The TDNN, which was first proposed for speech processing data in [42], is an adaptation of the classical Multilayer Perceptron (MLP), where time is converted into spatial representation. The MLP receives as input an array of values per observation, where each value corresponds to a different variable, but when dealing with time series this is not wise, since there is also a time relation between observations. This is where the TDNN comes in, by using a tapped delay line with a delay of size p at the input layer. With this addition, instead of using as input one value per input time series, one uses $p+1$ values per time series, multiplying the total of input variables by $p+1$, as can be seen in Figure 3. So, when predicting one $\mathbf{y}_t$ value of a given time series, the values $\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-p}$ are used as inputs. For this work, since the goal is to

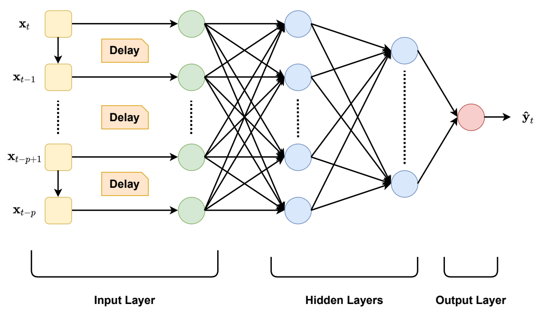

Figure 3: Time Delay Neural Network Graph with 2 hidden layers

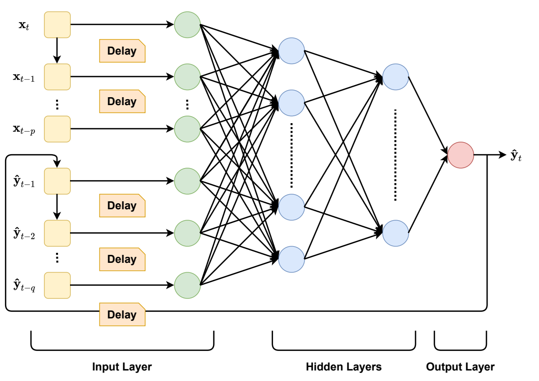

Figure 4: Neural Autoregressive Exogenous Model Graph for a Network with 2 hidden layers

model non-linear dynamical systems, the network would have to predict all $\mathbf{y}_t$ values for any given time series, instead of only predicting one value. Note that this model, as the MLP, is also trained using the backpropagation algorithm [36].

This model maintains the capacity of the MLP to model non-linear processes, while adding the possibility of looking at past values of the input, which is a clear advantage, making it a clear candidate for the work at stake. Nevertheless, this model still has a considerable downside, since with larger delays, the model will become more complex, increasing the computational costs, both in space and time. Another issue that might appear with this model when compared to some of the other models used in this work is that it has no access to previous output values of the time series, therefore having no access to the state of the system being modeled.

4.2. Neural Autoregressive Exogenous Model

The NARX is a type of recurrent neural network [29], very similar to the TDNN, with the main difference residing on the use of previous predicted values of the output time series to predict the current value of the output, as can be seen in Figure 4. The introduction of the recurrent connection with delay from the output of the network to the input allows the NARX to take into account the previous predicted values of the output, giving the NARX information on the system state, which was lacking

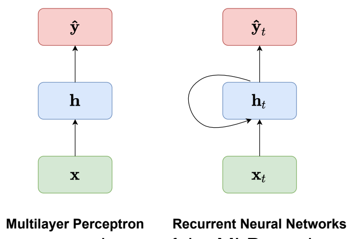

Figure 5: A compact scheme of the MLP graph vs a rolled RNN graph.

in the TDNN.

Due to this recurrent connection, the NARX has two different delays, one for the input variables, p, and one for the output variables, q. This means that the inputs of the network are the input values $\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-p}$ and the output predicted values $\hat{\mathbf{y}}_{t-1}, \hat{\mathbf{y}}_{t-2}, ..., \hat{\mathbf{y}}_{t-q}$, all used to predict the output at time t, $\hat{\mathbf{y}}_t$.

The training process of the NARX is usually done using the backpropagation algorithm [36], with the recurrent connection used open, being only closed to predict values after the training process is concluded. This is done with the goal of assuring a better convergence of the model parameters, since if the network is trained with a closed loop, the first predicted values during the training procedure might have such a high error that it leads to wrong inputs and eventually makes the training process diverge.

The NARX is able to fix a potential issue of the TDNN, since it has knowledge of the system state by using the predicted output, but maintains the scaling problem from the TDNN, since the larger the delays, the more complex the network gets, increasing the requirements both in space and time for using the model. Another issue that this model might present is the modeling of long term dependencies, since if the output depends on distant previous values the needed delay is also large, increasing the model requirements, sometimes to unusable sizes.

4.3. Recurrent Neural Network

RNNs are a family of neural networks that appeared to process sequential data [44], [17]. They belong to a different family of neural networks than the MLP, since they appear from the relaxation of the condition of the MLP that neurons in a given layer do not form connections among themselves as can be seen in Figure 5, where compact schemes of a MLP and a RNN are shown side by side, both with only one hidden layer. RNNs are also able to process sequences of values $\mathbf{x}_1, ..., \mathbf{x}_t$, in most cases also being able to process sequences of variable length.

Equations (1) and (2) describe the MLP, while equations (3) and (4) describe the RNN:

$$\mathbf{h} = \phi(\mathbf{Vx} + \mathbf{c}) \tag{1}$$
$$\hat{\mathbf{y}} = \mathbf{Wh} + \mathbf{b} \tag{2}$$
$$\mathbf{h}_t = \phi(\mathbf{Vx_t} + \mathbf{Uh_{t-1}} + \mathbf{c}) \tag{3}$$
$$\hat{\mathbf{y}}_t = \mathbf{Wh}_t + \mathbf{b} \tag{4}$$

In these equations $\mathbf{x}$ denotes the input vector, while $\mathbf{x_t}$ refers to the input vector values at time t, and the parameters learned by the model are the weights $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{U}$ and the biases $\mathbf{b}$ and $\mathbf{c}$. The activation function, which is usually a hyperbolic tangent, is indicated by $\phi(\cdots)$ and the outputs of the hidden layer are represented by $\mathbf{h}$ and $\mathbf{h_t}$, for the MLP and the RNN, respectively. The representation of the outputs is different between the models, while for the MLP the outputs are represented by $\hat{\mathbf{y}}$, for the RNN the output is represented by the value of the produced sequence at time t, $\hat{\mathbf{y}}_\mathbf{t}$.

Figure 5 shows how similar the two structures are, which can also be seen by the similarities between equations (2) and (4) that both compute the output in analogous fashion. The main difference between equations (1) and (3) is the inclusion of values from previous units in the same layer of the RNN. This difference, also apparent in Figure 5, is what makes the RNN suitable for time series data, since the RNN encodes the idea of sequence by taking into account values from previous units in the same layer.

RNNs, which are trained using Backpropagation Through Time [43], suffer from two main problems during the training procedure: vanishing and exploding gradients [10]. Exploding gradients refer to a large increase in the norm of the gradient during training, which is the less common problem, for which known solutions already exist such as the gradient clipping technique [34].

The vanishing gradient problem [10] happens when long term components go exponentially fast to norm 0, implying that distant events stop impacting the predictions of the model. This problem is what leads to the main difficulties when using RNN, which is related to the inability of this model to learn Long Term Dependencies.

Since there is no simple solution for this problem, the RNN may experience difficulties in learning the *C. elegans* data, since the used data has fine time steps, implying that there might be a large temporal distance between related time points, which may be hard for the RNN to learn.

4.4. Long Short-Term Memory Unit

The LSTM is a gated unit used in RNNs as a replacement for the simple unit which is only composed of a hyperbolic tangent function. This unit was proposed in 1997 [21], containing only two gates, the input and output gate, and was later improved with the inclusion of the forget gate [19]. This gated unit is already able to model long term dependencies much better than the RNN, due to the inclusion of three gates described as follows:

$$\mathbf{i_t} = \sigma(\mathbf{W_i x_t} + \mathbf{U_i h_{t-1}} + \mathbf{b_i}) \tag{5}$$
$$\mathbf{f_t} = \sigma(\mathbf{W_f x_t} + \mathbf{U_f h_{t-1}} + \mathbf{b_f}) \tag{6}$$
$$\mathbf{o_t} = \sigma(\mathbf{W_o x_t} + \mathbf{U_o h_{t-1}} + \mathbf{b_o}) \tag{7}$$

Each of these three gates holds a different function. The input gate controls whether the cell state is updated or not, equation (5), the forget gate defines how the previous memory cells affect the current one, equation (6) and the output gate controls how the hidden state is updated, equation (7).

Unlike the simple unit used in the RNN, the LSTM unit has two outputs, a hidden state, which is the output of the hidden layer that is also passed to the next LSTM unit in the network, and a cell state which is passed to the next LSTM unit in the network.

To compute these states, the following equations are used:

$$\tilde{\mathbf{c}}_\mathbf{t} = \phi(\mathbf{W_c x_t} + \mathbf{U_c h_{t-1}} + \mathbf{b_c}) \tag{8}$$
$$\mathbf{c_t} = \mathbf{f_t} \circ \mathbf{c_{t-1}} + \mathbf{i_t} \circ \tilde{\mathbf{c}}_\mathbf{t} \tag{9}$$
$$\mathbf{h_t} = \mathbf{o_t} \circ \phi(\mathbf{c}_t), \tag{10}$$

Equation (8) computes the candidate cell state which is then used along with the previous cell state to obtain the cell state on equation (9), by using the input and forget gates. The hidden state is computed from the cell state and the output gate using equation (10).

In equations (5) to (10) that define the LSTM unit, the 12 parameters are the weights, $\mathbf{W}_i$, $\mathbf{U}_i$, $\mathbf{W}_f$, $\mathbf{U}_f$, $\mathbf{W}_o$, $\mathbf{U}_o$, $\mathbf{W}_c$ and $\mathbf{U}_c$ and the biases, $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$ and $\mathbf{b}_c$, while the used activation functions $\sigma(\cdots)$ and $\phi(\cdots)$ are the logistic sigmoid and the hyperbolic tangent activation functions.

The LSTM is also trained using BPTT [43]. A scheme of the LSTM unit can be found in Figure 6 alongside the simple RNN unit and the GRU discussed next.

When using this unit it is important to take away that it handles modeling long term dependencies much better than the RNN, but that this does not come without a cost, since the number of total parameters of the model is much higher for the LSTM than for the RNN.

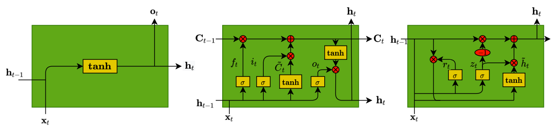

Figure 6: Comparison between the three different units, RNN, LSTM and GRU, respectively.

4.5. Gated Recurrent Unit

In 2014 a new type of unit for RNNs was proposed, the GRU [12]. This new unit was proposed to reduce the size of used networks when compared to the LSTM unit, while still maintaining the capability of modeling long term dependencies. This need for a smaller unit appeared due to the complexity of the most used LSTM unit, which has many parameters to train, increasing both the size of the model and the time used to train it.

The GRU is only composed of two gates defined by the following equations:

$$\mathbf{z}_t = \sigma(\mathbf{W_z x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \tag{11}$$

$$\mathbf{r}_t = \sigma(\mathbf{W_r x_t} + \mathbf{U_r h_{t-1}} + \mathbf{b_r}) \tag{12}$$

Equation (11) defines the update gate, which is used to fulfil the same function of the input and forget gates used in the LSTM together, deciding which information to maintain and which information to throw away, while equation (12) defines the reset gate that is used to decide how much past information should be forgotten.

Unlike the LSTM and similar to the simple RNN unit, the GRU has only one output, the hidden state, which is used as an input for the next unit and as an output of the hidden layer. This output is computed using the following equations:

$$\hat{\mathbf{h}}_\mathbf{t} = \phi(\mathbf{W_h x_t} + \mathbf{U_h}(\mathbf{r_t} \circ \mathbf{h_{t-1}} + \mathbf{b_h})) \tag{13}$$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{z_t}) \circ \mathbf{h_{t-1}} + \mathbf{z_t} \circ \hat{\mathbf{h}}_\mathbf{t}. \tag{14}$$

Equation (13) defines the candidate hidden state, which is then used along with the update gate and the hidden state of the previous unit to then compute the hidden state in equation (14).

In equations (11) to (14) that define the GRU, the parameters that the model should learn are the weights, $\mathbf{W}_x$, $\mathbf{U}_x$, $\mathbf{W}_r$, $\mathbf{U}_r$, $\mathbf{W}_h$ and $\mathbf{U}_h$, and the biases $\mathbf{b}_x$, $\mathbf{b}_r$ and $\mathbf{b}_h$. $\sigma(\cdots)$ and $\phi(\cdots)$ correspond again to the logistic sigmoid and the hyperbolic tangent activation functions, respectively.

RNNs using the GRU are also trained with BPTT [43] and its scheme can be seen in Figure 6. This model is still capable of modeling long term dependencies like the LSTM, but using less parameters, which is a major improvement that reduces the complexity of the model. Nevertheless, this model still has much more parameters than the simple RNN unit.

5. Methodology

5.1. Implementation Setting

The five models were not all implemented in the same environment, with the TDNN and NARX models being implemented using the MATLAB language [31] and its Deep Learning Toolbox, which provides a working implementation of both types of network architectures. The implementation of the RNN, LSTM and GRU models was handled in a different way, using python [40] and its keras [13] and tensorflow [7] libraries.

The training procedure is also performed differently for the five models, even though the differences are small. All the models were trained for 1000 epochs. For the TDNN and NARX models, the optimization algorithm used is the MATLAB implementation of the Levenberg-Marquardt Algorithm [28], [30]. The RNN, LSTM and GRU models were optimized using the Adam algorithm [26] and at the end, the model that performed best on the validation set was selected from the models obtained at each iteration.

All the code used to implement these models, along with the results of every experiment performed are made available at a github repository: `https://github.com/gmestre98/Thesis`.

5.2. Model Evaluation

Since the different machine learning techniques are compared in this work, it is of major importance to be able to evaluate these and compare their results in an accurate and fair way. In order to evaluate and compare the results produced by different techniques, a good metric for assessing the similarity of the produced results by any model against the synthetic *C. elegans* data is needed, along with a good metric for comparing the complexity of these models.

The models produced are regression models, hence, common classification accuracy metrics cannot be used since regression models try to predict a numerical value, contrary to classification models that try to predict a class value. When predicting a numerical value there is an almost infinite number of possible results, some closer to the correct result than others. Therefore, metrics specifically designed for regression tasks were used, with the Mean Squared Error (MSE), equation (15), used as the loss function for the training process and the Root Mean Squared Error (RMSE), equation (16) used to evaluate the predictions obtained by the models.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{0}y_i)^2 \tag{15}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{16}$$

In equations (15) and (16), N refers to the number of samples in the sequence predicted by the model, y_i refers to the expected results at time step i and $\hat{y}_i$ refers to the results predicted by the model at time step i.

Regarding the evaluation of the models' complexity, two main characteristics can be evaluated, the time and the space required by the models to run. In this work the comparison between the time taken by each model to run is neglected, since the models were implemented in different programming languages with different frameworks, which impacts the time the models take to run, biasing the comparisons that could be made.

A comparison on the size of the models cannot be done using only the number of neurons in the neural networks implemented, since the networks used possess different types of units, some having more parameters than others. This comparison using the number of neurons in the neural network can only be done for models that use the same architecture. Therefore, the comparison of models using different architectures is made on the number of parameters that each corresponding model uses. For this, a formula to compute the total number of parameters of each model is provided for the TDNN, the NARX, the RNN, the LSTM and the GRU, in equations (17), (18), (19), (20) and (21), respectively.

$$\begin{aligned} \#TDNN &= s \times [i \times (d_x + 1) + 1] \\ &\quad + o \times (s+1) \end{aligned} \tag{17}$$

$$\begin{aligned} \#NARX &= s \times [i \times (d_x + 1) + o \times d_y + 1] \\ &\quad + o \times (s+1) \end{aligned} \tag{18}$$

$$\#RNN = s \times (i + s + 1) + o \times (s+1) \tag{19}$$

$$\#LSTM = 4 \times s \times (i + s + 1) + o \times (s+1) \tag{20}$$

$$\#GRU = 3 \times s \times (i + s + 1) + o \times (s+1) \tag{21}$$

This set of five equations is used throughout the remainder of this work to compute the total number of parameters of the models so that models with similar size are compared.

5.3. Hyperparameters Optimization

An important step to develop these neural network models is to understand which hyperparameters should be used for each of the architectures in order to better model the *C. elegans* nervous system behavior. This optimization was made for some of the hyperparameters using the dataset on the PTRS scenario, with only the four measured outputs and a time step between samples of 0.5ms.

Since these neural networks are initialised with random values, the results of the training procedure may differ from one simulation to another, which means that it is not wise to train them only a single time and compare the results obtained. Instead, each of the models is trained and tested on the validation data multiple times. In this work it was chosen to do it ten times. The average RMSE values for each model with the specific hyperparameter values were then computed.

Based on the obtained results, which may be consulted in [32], it was concluded that the wisest choice of delay on the input for the TDNN would be of 1 sample and that the model does not perform well enough, not even when the size of the network is increased. Regarding the NARX model, an accurate performance was shown, with the delay on the input and on the output being fixed at 4 and 2 samples, respectively, also concluding that the model performs accurately enough with a size of 16 or 32 units for the hidden layer, showing a tendency to overfit the training data for larger values.

For the RNN, a learning rate of 0.001 and a batch size of 32 were chosen, with the model not performing well enough, but showing a tendency to decrease the RMSE as the size of the network increases. The LSTM and GRU models had a learning rate of 0.05 and a batch size of 32 chosen, with these models performing accurately, being able to replicate the behavior of the *C. elegans* nervous system with high fidelity in this scenario.

5.4. Sample Rate

One important part of this work described previously is the generation of the data on the *C. elegans* behavior. So, finding how many samples are needed for the models to accurately reproduce this dynamical system behavior is relevant in order to understand the limitations and needs of these techniques. With this goal, a study on how many samples are needed for each example in order to accurately model the available data was performed.

Even though there are two models that did not perform well enough, the TDNN and the RNN, these are also tested in order to understand if using a finer or a coarser time step helps the models learn the dynamical behavior. Therefore, the five different techniques are tested, using a hidden layer of 16 units for the five models, since this value was in the set of optimal values experimented for all the models hidden layer sizes.

These techniques are tested on both the PTRS and the Nictation scenarios, with each of the two datasets generated in six different versions, using a time step between samples of 10ms, 5ms, 1ms, 0.5ms, 0.1ms and 0.05ms.

The overall results indicate that the TDNN and RNN still have a poor performance, although this performance improves with the increase of the time step used, with the RNN producing significantly more accurate predictions than the TDNN. The NARX reveals a convergence problem, not converging in the majority of simulations on the PTRS scenario except for a time step of 0.5ms, the one used in the hyperparameter optimization. This model also proves unable to provide accurate predictions for the Nictation scenario datasets. LSTM and GRU models tend to produce a good performance regardless of the time step used, but still performing better on data with specific time steps. Based on these results, although a time step of 1ms was considered, it was decided to use a time step between samples of 0.5ms for the remainder of this work in order for the NARX to still be comparable with the other models.

6. Experiments

6.1. Models Comparison

Having now chosen an appropriate time step for the generated data, the next step is to compare the performance of the different models. The focus of this experiment is to compare the different models on two datasets, also testing if the size of the models has an impact on the performance.

Since this is just a performance comparison, it is only used the test set RMSE, so that the models are only compared on unseen data, testing the ability of these to generalize. As discussed previously, the different models compared should have a similar number of parameters. So, a reference model is chosen with three different hidden sizes, and for the remaining models the size is chosen accordingly so that the total of parameters of the models being compared is similar. It is important to note that the number of parameters was computed for the PTRS scenario, since it would diverge between both scenarios. The model that showed a better performance during the previous tests was the GRU, hence was chosen as the reference model for this experiment, with three different sizes for the hidden layer tested, 8, 16 and 32 units.

For the GRU with 8 units in the hidden layer, the sizes of the hidden layer that lead the corresponding models to have a similar number of parameters are 26, 10, 15 and 7, for the TDNN, NARX, RNN and LSTM models, respectively. Regarding the other sizes, the GRU with 16 units leads to 82, 32, 29 and 14 for the TDNN, NARX, RNN and LSTM models, respectively, and the GRU with 32 units leads to 283, 112, 56 and 27 for the TDNN, NARX, RNN and LSTM models, respectively.

Throughout the comparisons in Tables 1, 2 and 3, it is possible to see that the GRU models are the ones that perform better, but finding a strong contender in accurately reproducing the behavior of the dynamical system.

When comparing the rest of the models to these two, there is a clear difference, with the NARX only being able to perform accurately on the PTRS scenario with 10 and 32 units in the hidden layer and the remaining two models, TDNN and RNN, not performing accurately on any of the datasets. It can also be concluded that further study on how to improve the training process of the NARX models might be useful, since these models produce very accurate reproductions of the system's behavior in the PTRS scenario.

6.2. Needed Stimuli Input

It is often the case that even though all inputs to a system are by definition external to the system, one might not be able to measure or observe all of these outside stimuli given to the system. Therefore, this experiment tests the ability of the models to accurately learn the behavior of the system using less inputs than the ones actually used for data generation.

Since the previous experiment indicates that the best performing models are the LSTM and GRU, the current one tests only these models with recurrent layers of size 8, 16 and 32. In order to test how the lack of input information affects the models, five different tests are performed, two concerning PTRS and the other three concerning Nictation.

The first test concerning the PTRS scenario evaluates the performance of the models when deprived from the information of the input on AVB neurons, while for the second one the information of the input on PLM neurons is missing, in Table 4.

The average RMSE of all the simulations performed in the absence of the information of the input given to the AVB neurons are approximately the same for both LSTM and GRU models for the different experimented sizes. The models are able to accurately replicate the output voltage of two neurons in the system, LUAL and PVR, for all the test examples. For the remaining two neurons, DB1 and VB1, the system's behavior is not predicted well in seven out of ten test examples, is predicted with some error in two out of ten and is predicted accurately in one out of ten.

The simulations using no information on the inputs provided to the PLM neurons also obtain approximately the same RMSE for both models with the different sizes, but with an order of magnitude higher. The results obtained by the models in these simulations are clearly unacceptable, since the models fail to predict the output voltage of the LUAL and PVR neurons, only being able to replicate, with some error, the behavior of the output voltage of the DB1 and VB1 neurons in some of the simulations performed in five of the ten test examples.

The tests on the Nictation dataset, **Nictation-20-0.5** are performed without the use of one of three groups of neurons for the three different cases: IL2D, IL2 and IL2V. The results of these tests concerning the Nictation behavior are available in Table 5.

The average RMSE of the obtained simulations for the cases with no information on the input provided to the IL2D and IL2V neurons is of similar

Table 1: The average RMSE obtained for ten simulations, comparing different models using as reference the GRU with a hidden layer composed of 8 units on the PTRS and on the Nictation scenarios.

	TDNN-26	NARX-10	RNN-15	LSTM-7	GRU-8
PTRS	1.9828e-01	1.0186e-02	1.3171e-01	1.5435e-02	7.0780e-03
Nictation	1.1641e+00	2.4073e+00	6.31693-01	9.3827e-02	5.5078e-02

Table 2: The average RMSE obtained for ten simulations, comparing different models using as reference the GRU with a hidden layer composed of 16 units on the PTRS and on the Nictation scenarios.

	TDNN-82	NARX-32	RNN-29	LSTM-14	GRU-16
PTRS	8.5547e-01	5.9673e-03	1.1952e-01	1.3338e-02	5.9024e-03
Nictation	3.7449e+00	9.4993e-01	5.8187e-01	1.1868e-01	6.3612e-02

Table 3: The average RMSE obtained for ten simulations, comparing different models using as reference the GRU with a hidden layer composed of 32 units on the PTRS and on the Nictation scenarios.

	TDNN-283	NARX-112	RNN-56	LSTM-27	GRU-32
PTRS	1.9064e+00	3.3905e-01	9.0798e-02	1.6765e-02	7.4332e-03
Nictation	5.8773e+00	2.9169e+00	5.1648e-01	1.0663e-01	6.6432e-02

Table 4: The average RMSE obtained for ten simulations, comparing the performance of the models when the PLM neurons input information is not available and when the same information is not available for the AVB neurons.

	LSTM-8	LSTM-16	LSTM-32	GRU-8	GRU-16	GRU-32
No AVB	6.7437e-02	6.7918e-02	6.8155e-02	6.8303e-02	6.7904e-02	6.5765e-02
No PLM	2.0008e-01	2.0352e-01	2.0190e-01	1.9634e-01	1.9665e-01	1.9769e-01

magnitude, just slightly higher for the GRU models, than it is for the LSTM. These errors are also slightly higher in these two cases than the results obtained for the Nictation scenario with LSTM and GRU models in the previous experiment, providing accurate predictions when being deprived of the IL2D or IL2V neurons input information.

When the models are deprived of the IL2 inputs, the performance is really poor on all simulations, indicating that these inputs are fundamental for the LSTM and GRU neural networks to model this dynamic behavior.

This experiment leads to the conclusion that depending on the behavioral scenario simulated for the worm, some inputs are more relevant than others, for the models to predict the output of some neurons in the *C. elegans* nervous system. This indicates that information on some inputs may not be measured when using real data for these nematodes in a laboratory, potentially facilitating future work on modeling the behavior of the *C. elegans*.

6.3. Full Output Prediction

Previous experiments used just a small number of outputs for prediction, not predicting the output behavior of all interneurons and motor neurons that are usually studied in these behaviors. This experiment tests the models prediction capabilities for the full set of output variables measured and listed previously. This is made so that the models are tested on replicating the *C. elegans* behavior as extensively as possible, trying to predict all expected outputs on the given task.

The models tested are the same as in the previous section, LSTM and GRU, with a recurrent layer with three different sizes for each model, 8, 16 and 32 units and the results obtained can be found in Table 6.

For the PTRS scenario, the models with 8 units show a slightly worse performance, with both the LSTM and the GRU obtaining similar RMSE values to what these models already had on the previous simulations. This indicates that the models perform well in this scenario, meaning that they are able to scale the number of outputs predicted still providing correct predictions and without a significant increase on the size of the models.

For the Nictation scenario, the RMSE obtained also has a similar magnitude to what was previously obtained, with the models with 16 hidden units performing slightly better. The RMSE obtained is in agreement with the predicted voltages for each of the output neurons. The predictions obtained show a response not as accurate as desirable for the SMD motor neurons, which can be explained by the fact that these have a small absolute magnitude, when compared with the other neurons. This indicates the need for the use of a relative error metric for the loss function used in the training process in future works, when predicting multiple variables with values of different magnitudes.

Table 5: The average RMSE obtained for ten simulations on the Nictation scenario, comparing the performance of the models not using one of three sets of neurons: IL2D, IL2 and IL2V.

	LSTM-8	LSTM-16	LSTM-32	GRU-8	GRU-16	GRU-32
No IL2D	6.1386e-02	6.3848e-02	7.5710e-02	4.4289e-02	3.6634e-02	4.1057e-02
No IL2	9.2529e-01	9.4015e-01	9.4099e-01	9.0759e-01	9.1935e-01	9.6122e-01
No IL2V	7.5938e-02	7.8919e-02	8.5085e-02	4.8651e-02	4.4650e-02	4.8579e-02

Table 6: The average RMSE obtained for ten simulations, comparing different models on the PTRS and Nictation behaviors, using the full set of outputs for each dataset.

	LSTM-8	LSTM-16	LSTM-32	GRU-8	GRU-16	GRU-32
PTRS	1.8072e-02	1.5438e-02	1.5992e-02	8.3349e-03	6.0851e-03	5.8352e-03
Nictation	1.1620e-01	8.4321e-02	1.0814e-01	6.1797e-02	4.8148e-02	5.1163e-02

Table 7: The average RMSE obtained for ten simulations, comparing different techniques on their ability to create a model that can replicate the PTRS and Nictation behaviors, using the full set of outputs for each dataset.

	LSTM-8	LSTM-16	LSTM-32	GRU-8	GRU-16	GRU-32
Two-Behaviors	4.3204e-02	3.9654e-02	4.0316e-02	3.7441e-02	2.3720e-02	2.6317e-02

6.4. Predicting Two Behaviors Simultaneously

This last experiment tests the performance of the models on only one dataset, as opposed to the previous experiment, where the two scenarios are combined, using the same architectures of the two previous experiments. This is made in order to test the ability of the machine learning techniques used to create a single model that can replicate the behavior of the *C. Elegans* in different tasks. The results obtained can be found in Table 7.

The results obtained indicate a slightly superior performance of the GRU, obtaining a low RMSE. This indicates the ability of these models to provide accurate predictions, creating a model that can replicate the behavior of the *C. elegans* in two different scenarios, using a number of neurons for the recurrent layer inferior to the total of output neurons for which the voltage is predicted.

The models are able to predict accurately the behavior of the system in both scenarios, on the neurons which are object of study for the given scenario.The models only fail to predict the behavior of the neurons where there is little to no activity. This issue is natural, since a relative error metric was not used for the loss function as explained previously. Nevertheless, this fact constitutes no problem for most applications, because the object of study usually are the neurons affected by the given behavior and not the neurons where there is a lack of activity. The only exception to this conclusion are the SMD neurons predictions for the PTRS behavior, which is natural since these show some activity during the forward locomotion of the worm.

7. Conclusion

In this work different datasets on two behavioral scenarios of the *C. elegans* nervous system were generated using the NEURON simulator, which were used to test the performance of different machine learning techniques replicating the behavior of the *C. elegans* nervous system on two different tasks, PTRS and Nictation.

These models were then compared, and the two best performing models, LSTM and GRU, proved able to accurately predict the behavior of the system when being deprived of some of the input information, but only on specific cases. The two models were also able to provide accurate predictions when computing the full set of outputs or when modeling data from the PTRS and Nictation scenarios simultaneously. This was possible without a need to increase the size of the network used, proving their usefulness and scalability capacity to predict larger sets of outputs.

Regarding future work, many directions may be taken such as testing on more targeted datasets, specifically designed by neuroscience researchers or even testing if these techniques are able to perform at the same level when dealing with data from more complex organisms, like the fruit fly or even the human nervous system.

Other directions may reside on testing other machine learning techniques or performing studies on why the RNN and the NARX fail to reproduce the *C. elegans* behavior, and how these can be improved. The training process may also be improved by experimenting with more hyperparameters than the ones experimented with in this work.

The above suggestions are specific items that the current work unearthed but there are likely a myriad others that could equally well be raised and deserved further research.

References

[1] Gene expression database. https://www.gfpworm.org/. Accessed: 2021-09-03.

[2] Neuromorph.org. https://neuromorph.org/. Accessed: 2021-09-04.

[3] Open Source Brain. https://www.opensourcebrain.org/. Accessed: 2021-09-04.

[4] OpenWorm. http://openworm.org/index.html. Accessed: 2021-09-04.

[5] Wormbase, ws280. https://www.wormbase.org/. Accessed: 2021-09-02.

[6] Wormbook, the online review of c. elegans biology. http://www.wormbook.org/. Accessed: 2021-09-03.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[9] R. Barbulescu and L. M. Silveira. Black-box model reduction of the C. elegans nervous system. *43rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2021.

[10] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[11] N. T. Carnevale and M. L. Hines. *The NEURON book*. Cambridge University Press, 2006.

[12] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[13] F. Chollet et al. Keras. https://keras.io, 2015.

[14] S. J. Cook, T. A. Jarrell, C. A. Brittin, Y. Wang, A. E. Bloniarz, M. A. Yakovlev, K. C. Nguyen, L. T.-H. Tang, E. A. Bayer, J. S. Duerr, et al. Whole-animal connectomes of both Caenorhabditis elegans sexes. *Nature*, 571(7763):63–71, 2019.

[15] A. E. C. o. M. Department of Neuroscience. Wormatlas, a database featuring behavioral and structural anatomy of caenorhabditis elegans. https://www.wormatlas.org/. Accessed: 2021-09-02.

[16] A. E. C. o. M. Department of Neuroscience. The WormImage Database. https://www.wormimage.org/. Accessed: 2021-09-03.

[17] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[18] A. E. C. o. M. Emmons Lab. WormWiring, Nematode Connectomics. https://www.wormwiring.org/. Accessed: 2021-09-04.

[19] F. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999.

[20] P. Gleeson, M. Cantarelli, B. Marin, A. Quintana, M. Earnshaw, S. Sadeh, E. Piasini, J. Birgiolas, R. C. Cannon, N. A. Cayco-Gajic, et al. Open source brain: a collaborative resource for visualizing, analyzing, simulating, and developing standardized models of neurons and circuits. *Neuron*, 103(3):395–411, 2019.

[21] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[22] R. Hunt-Newbury, R. Viveiros, R. Johnsen, A. Mah, D. Anastas, L. Fang, E. Halfnight, D. Lee, J. Lin, A. Lorch, et al. High-throughput in vivo analysis of gene expression in caenorhabditis elegans. *PLoS Biol*, 5(9):e237, 2007.

[23] B. M. Jackson, P. Abete-Luzi, M. W. Krause, and D. M. Eisenmann. Use of an activated beta-catenin to identify wnt pathway target

genes in caenorhabditis elegans, including a subset of collagen genes expressed in late larval development. *G3: Genes, Genomes, Genetics*, 4(4):733–747, 2014.

[24] T. A. Jarrell, Y. Wang, A. E. Bloniarz, C. A. Brittin, M. Xu, J. N. Thomson, D. G. Albertson, D. H. Hall, and S. W. Emmons. The connectome of a decision-making neural network. *Science*, 337(6093):437–444, 2012.

[25] J. Kim, W. Leahy, and E. Shlizerman. Neural interactome: Interactive simulation of a neuronal system. *Frontiers in Computational Neuroscience*, 13:8, 2019.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[27] H. Lee, M.-k. Choi, D. Lee, H.-s. Kim, H. Hwang, H. Kim, S. Park, Y.-k. Paik, and J. Lee. Nictation, a dispersal behavior of the nematode caenorhabditis elegans, is regulated by il2 neurons. *Nature neuroscience*, 15(1):107–112, 2012.

[28] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[29] T. Lin, B. Horne, P. Tino, and C. Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[30] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[31] MATLAB. *Matlab 2020a*. The MathWorks Inc., Natick, Massachusetts, 2020.

[32] G. Mestre. Modeling C. elegans nervous system's behavior using machine learning techniques. Master's thesis, Instituto Superior Técnico, 2021.

[33] A. Palyanov, S. Khayrulin, S. D. Larson, and A. Dibert. Towards a virtual c. elegans: A framework for simulation and visualization of the neuromuscular system in a 3d physical environment. *In silico biology*, 11(3, 4):137–147, 2012.

[34] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[35] S. Patil, K. Zhou, and A. C. Parker. Neural circuits for touch-induced locomotion in caenorhabditis elegans. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[37] K. Sakamoto, Z. Soh, M. Suzuki, Y. Kurita, and T. Tsuji. A neural network model of caenorhabditis elegans and simulation of chemotaxis-related information processing in the neural network. In *2015 SAI Intelligent Systems Conference (IntelliSys)*, pages 668–673, 2015.

[38] M. Suzuki, T. Tsuji, and H. Ohtake. A neuromuscular model of c. elegans with directional control. In *Proc. of the first international conference on complex medical engineering*, pages 167–172, 2005.

[39] B. Szigeti, P. Gleeson, M. Vella, S. Khayrulin, A. Palyanov, J. Hokanson, M. Currie, M. Cantarelli, G. Idili, and S. Larson. Open-Worm: an open-science approach to modeling Caenorhabditis elegans. *Frontiers in computational neuroscience*, 8:137, 2014.

[40] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[41] L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii. Structural properties of the Caenorhabditis elegans neuronal network. *PLoS Comput Biol*, 7(2):e1001066, 2011.

[42] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

[43] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78:1550 – 1560, 11 1990.

[44] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[45] J. G. White, E. Southgate, J. N. Thomson, S. Brenner, et al. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340, 1986.

[46] J.-X. Xu, X. Deng, and D. Ji. Study on c. elegans behaviors using recurrent neural network model. In *2010 IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, 2010.